Wireguard Docker

- Les commandes Wireguard
- Mise en place de Wireguard Serveur
- Mise en place de Wireguard Client
  - 1 - Via l'utilisation de l'image docker
  - 2 - Via l'utilisation du package Linux
  - 3 - via le logiciel Windows

- Diagnostic WireGuard — Linux avec NetworkManager (GNOME)
- Configuration WireGuard Mesh — Documentation complète
  - Architecture WireGuard Mesh
  - Processus de déploiement
  - Configuration Wireguard Docker
  - Configuration Serveurs
  - Configuration Clients
  - Configuration Pare-feu

Les commandes Wireguard

Générer une paire de clé privé/public

Vous utiliserez les commandes intégrées `wg genkey` et `wg pubkey` pour créer les clés, puis ajouterez la clé privée au fichier de configuration de WireGuard.

Génération de la clé privée

```
wg genkey >> private.key
```

Une fois la clé privé généré, il faut lui enlever les droits des utilisateurs et des groupes :

La `sudo chmod go=...` commande supprime toutes les autorisations sur le fichier pour les utilisateurs et les groupes autres que l'utilisateur root afin de s'assurer que lui seul peut accéder à la clé privée.

```
chmod go= private.key
```

Génération de la clé publique

```
cat private.key | wg pubkey >> public.key
```

Ajouter un nouveau peer dans la configuration

```
wg set <interface_name> peer <public_key_client> allowed-ips <adress_ip_vpn_client>
```

/!\ Ne pas oublier d’enregistrer les modifications effectuées sur la configuration serveur : `wg-quick save <interface_name>`

Prendre en compte des modifications de la configuration

```
wg-quick save <interface_name>
```

Afficher la configuration prise en compte

```
wg show <interface_name>
```

Supprimer un peer sur le serveur

Pour supprimer un peer de la configuration serveur on a besoin :

- du nom de l'interface
- de la clé public du peer à supprimer

```
wg set <interface_name> peer <public_key_peer> remove
```

Si le serveur se trouve dans un conteneur

```
docker exec <container_name> wg set <interface_name> peer <public_key_peer> remove
```

/!\ Ne pas oublier d’gistrer les modifications effectuées sur la configuration serveur : `wg-quick save <interface_name>`

Comment surveiller qui se connecter à votre VPN Wireguard ?

La chose la plus simple que vous puissiez faire est de vous connecter en SSH à chacun des hôtes WireGuard sur votre réseau, et d'utiliser l'affichage d'état intégré de WireGuard pour vérifier l'état actuel de chaque interface et de chaque pair. Cela peut être faisable si vous n'avez que quelques serveurs VPN primaires à travers lesquels vos points de terminaison se connectent (plutôt qu'un réseau de points de terminaison connectés point à point).

Si vous vous connectez en SSH à un hôte exécutant WireGuard, vous pouvez obtenir un affichage en ligne de commande de chaque interface WireGuard active sur l'hôte, ainsi qu'une liste de chaque pair configuré pour l'interface, via la commande `wg` :

```
sudo wg show
```

```
interface: wg1
  public key: /TOE4TKtAqVsePRVR+5AA43HkAK5DSntkOCO7nYq5xU=
  private key: (hidden)
  listening port: 51821

peer: fE/wdxzl0klVp/IR8UcaoGUMjqaWi3jAd7KzHKFS6Ds=
  endpoint: 172.19.0.8:51822
  allowed ips: 10.0.0.2/32
  latest handshake: 1 minute, 22 seconds ago
  transfer: 3.48 MiB received, 33.46 MiB sent

peer: jUd41n3XYa3yXBzyBvWqlLhYgRef5RiBD7jwo70U+Rw=
  endpoint: 172.19.0.7:51823
  allowed ips: 10.0.0.3/32
  latest handshake: 2 hours, 3 minutes, 34 seconds ago
```

```
  transfer: 1.40 MiB received, 19.46 MiB sent
```

Chaque pair répertorié comprendra la clé publique utilisée par le pair, ainsi que quatre autres champs :

point final
Adresse IP publique actuelle (et port UDP) utilisée par l'homologue. Cette valeur est mise à jour à partir de la valeur initialement configurée pour l'homologue chaque fois que l'interface locale reçoit un nouveau paquet de l'homologue avec une adresse source (ou un port) différente.

ips autorisés
Adresses IP que l'interface WireGuard locale acheminera vers le pair.

dernier handshake
Si l'interface locale s'est connectée avec succès au pair distant depuis que l'interface a été démarrée, ceci indique la dernière fois que la connexion a été recodée. La "poignée de main" de reconnexion se produit toutes les 2 ou 3 minutes lorsque la connexion est activement utilisée (et uniquement lorsqu'elle est utilisée), ce qui vous donne une bonne approximation de la dernière fois que la connexion a été active. Ce champ sera omis si l'interface n'a pas réussi à se connecter à l'homologue depuis le démarrage de l'interface.

transfert
Si l'interface locale a tenté de se connecter à l'homologue distant depuis le démarrage de l'interface, ce champ indique la quantité de données reçues et envoyées à l'homologue. Ce champ sera omis si l'interface n'a pas tenté de se connecter à l'homologue depuis le démarrage de l'interface.

Les inconvénients évidents de l'utilisation de cette interface comme seul outil de surveillance sont que vous devez garder une session SSH ouverte sur chaque hôte que vous souhaitez surveiller, et que vous ne pouvez voir que ce qui se passe actuellement (ou la dernière fois que chaque pair était actif) - et non ce qui s'est passé aux moments spécifiques que vous pourriez vouloir examiner ou analyser.

Mise en place de Wireguard Serveur

> WireGuard® est un VPN extrêmement simple, rapide et moderne qui utilise une cryptographie de pointe. Il se veut considérablement plus performant qu'OpenVPN.
>
> WireGuard est conçu comme un VPN à usage général fonctionnant aussi bien sur des interfaces embarquées que sur des superordinateurs, adapté à de nombreuses circonstances différentes.
>
> Initialement publié pour le noyau Linux, il est désormais multiplateforme (Windows, macOS, BSD, iOS, Android) et largement déployable. Il fait actuellement l'objet d'un développement intensif, mais il peut déjà être considéré comme la solution VPN la plus sûre, la plus facile à utiliser et la plus simple du secteur.

image docker : https://github.com/linuxserver/docker-wireguard

Utilisation de l'image docker

Mode serveur

Si la variable d'environnement `PEERS` est définie à un nombre ou à une liste de chaînes séparées par une virgule, le conteneur fonctionnera en mode serveur et les confs serveur et pair/client nécessaires seront générées.

Les qr codes de configuration des pairs/clients seront affichés dans le journal de docker si `LOG_CONFS` est défini à `true`. Ils seront également sauvegardés au format texte et png sous `/config/peerX` dans le cas où `PEERS` est une variable et un entier ou `/config/peer_X` dans le cas où une liste de noms a été fournie à la place d'un entier.

Les variables `SERVERURL`, `SERVERPORT`, `INTERNAL_SUBNET`, `PEERDNS`, `INTERFACE`, `ALLOWEDIPS` et `PERSISTENTKEEPALIVE_PEERS` sont des variables optionnelles utilisées pour le mode serveur. Toute modification de ces variables d'environnement déclenchera la régénération des confs du serveur et des pairs. Les confs pairs/clients seront recréées avec les clés privées/publiques existantes.

--> Supprimez les dossiers des pairs pour que les clés soient recréées en même temps que les confs.

Pour ajouter ultérieurement d'autres pairs/clients, vous devez incrémenter la variable d'environnement PEERS ou ajouter d'autres éléments à la liste et recréer le conteneur.

Pour afficher à nouveau les codes QR des pairs actifs, vous pouvez utiliser la commande suivante et lister les numéros de pairs comme arguments : `docker exec -it wireguard /app/show-peer 1 4 5` ou `docker exec -it wireguard /app/show-peer myPC myPhone myTablet` (Gardez à l'esprit que les codes QR sont également stockés sous forme de PNG dans le dossier config).

Les modèles utilisés pour la configuration du serveur et des pairs sont enregistrés dans le dossier `/config/templates`. Les utilisateurs avancés peuvent modifier ces modèles et forcer la génération de confs en supprimant `/config/wg0.conf` et en redémarrant le conteneur.

docker-compose :

```
version: "2.1"

services:

  wireguard:
    image: lscr.io/linuxserver/wireguard:latest
    container_name: wireguard
    cap_add:
      - NET_ADMIN
      # - SYS_MODULE  # optional already active
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Paris
      - SERVERURL=vpn.domaine-name.com
      - SERVERPORT=51820
      - PEERS=pcApple,pcGamer,telUser1,castHome
      - PEERDNS=auto
      #- INTERNAL_SUBNET=10.13.13.0 #optional
      #- ALLOWEDIPS=0.0.0.0/0 #optional
      #- PERSISTENTKEEPALIVE_PEERS= #optional
      - LOG_CONFS=true
    volumes:
      - /etc/container-conf/wireguard/appdata/config:/config
```

```
      - /etc/container-conf/wireguard/lib/modules:/lib/modules
    ports:
      - 51820:51820/udp
    restart: unless-stopped

volumes:
 wireguard:
```

Paramètres

Les images des conteneurs sont configurées à l'aide de paramètres transmis au moment de l'exécution (tels que ceux indiqués ci-dessus). Ces paramètres sont séparés par deux points et indiquent `<external>:<internal>` respectivement.

Par exemple, `-p 8080:80` expose le port `80` à l'intérieur du conteneur pour qu'il soit accessible à partir de l'IP de l'hôte sur le port `8080` à l'extérieur du conteneur.

Parameter	Function
`51820/udp`	wireguard port
`PUID=1000`	for UserID - see below for explanation
`PGID=1000`	for GroupID - see below for explanation
`TZ=Etc/UTC`	specify a timezone to use, see this list.
`SERVERURL=vpn.domain.com`	IP externe ou nom de domaine de l'hôte docker. Utilisé en mode serveur. Si la valeur est auto, le conteneur essaiera de déterminer et de définir l'IP externe automatiquement.
`SERVERPORT=51820`	Port externe pour l'hôte Docker. Utilisé en mode serveur.
`PEERS=1`	Nombre de pairs pour lesquels créer des confs. Requis pour le mode serveur. Peut également être une liste de noms : myPC,myPhone,myTablet (alphanumérique uniquement).
`PEERDNS=auto`	Serveur DNS défini dans les configurations homologue/client (peut être défini comme 8.8.8.8). Utilisé en mode serveur. La valeur par défaut est auto, ce qui utilise le DNS de l'hôte wireguard docker via la redirection CoreDNS incluse.
`INTERNAL_SUBNET=10.13.13.0`	Sous-réseau interne pour le wireguard, le serveur et les pairs (à ne modifier qu'en cas de conflit). Utilisé en mode serveur.

Parameter	Function
`ALLOWEDIPS=0.0.0.0/0`	Les IP/plages que les homologues pourront atteindre en utilisant la connexion VPN. Si elle n'est pas spécifiée, la valeur par défaut est : '0.0.0.0/0, ::0/0' Tout le trafic transitera par le VPN. Si vous souhaitez un tunnel divisé, réglez cette valeur sur les IP que vous souhaitez utiliser dans le tunnel ET sur l'IP du WG du serveur, par exemple 10.13.13.1.
`PERSISTENTKEEPALIVE_PEERS=`	Réglé sur tous ou sur une liste de pairs séparés par des virgules (par exemple 1,4,laptop) pour que le serveur wireguard envoie des paquets keepalive aux pairs listés toutes les 25 secondes. Utile si le serveur est accessible via un nom de domaine et possède une IP dynamique. Utilisé uniquement en mode serveur.
`LOG_CONFS=true`	Les codes QR générés seront affichés dans le journal du docker. Mettez false pour ignorer la sortie du journal.
`/config`	Contains all relevant configuration files.
`/lib/modules`	Host kernel modules for situations where they're not already loaded.

Ouverture des ports

Maintenant que le serveur est configuré il ne faut pas oublier d'ouvrir le port concerné.
Dans notre cas nous devons ouvrir le port 51820 sur le serveur et sur la box internet car le serveur se trouve derrière une box internet.

Sur le serveur (avec `ufw`) :

```
sudo ufw allow 51820
```

Sur la box internet : *(Voir comment configurer la redirection de port sur le panneau de configuration de la box internet de votre fournisseur)*

Accès Container Docker via Wireguard

Pour accéder aux containers Docker du serveur via Wireguard il est nécessaire d'ouvrir le par-feu comme ci dessous :

Adresse IP du container Wireguard : 192.168.128.2 (`docker inspect <containername>`)

Via UFW : `sudo ufw allow from 192.168.128.2`

Ce qui donne la configuration suivante :

```
> ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[1] 22/tcp                     ALLOW IN    Anywhere
[2] 80/tcp                     ALLOW IN    Anywhere
[3] 443                        ALLOW IN    Anywhere
[4] Anywhere                   ALLOW IN    192.168.128.2
[5] 51820                      ALLOW IN    Anywhere
[6] 22/tcp (v6)                ALLOW IN    Anywhere (v6)
[7] 80/tcp (v6)                ALLOW IN    Anywhere (v6)
[8] 443 (v6)                   ALLOW IN    Anywhere (v6)
[9] 51820 (v6)                 ALLOW IN    Anywhere (v6)
```

Mise en place de Wireguard Client

Pour utiliser Wireguard en mode client, deux choix s'offre à vous :
1- Le premier, vous créez un conteneur Docker en mode client
2- Le deuxième vous installer le package et vous configurer Wireguard client directement sur votre machine.

1 - Via l'utilisation de l'image docker

Via l'utilisation de l'image docker

Ne définissez pas la variable d'environnement `PEERS`. Déposez votre conf client dans le dossier config en tant que `/config/wg0.conf` et démarrez le conteneur.

Si vous obtenez des erreurs liées à IPv6 dans le journal et que la connexion ne peut pas être établie, modifiez la ligne `AllowedIPs` dans votre pair/client wg0.conf pour inclure uniquement `0.0.0.0/0` et non `::/0`; et redémarrez le conteneur.

docker-compose :

```
version: "2.1"

services:

  wireguard:
    image: lscr.io/linuxserver/wireguard:latest
    container_name: wireguard
    cap_add:
      - NET_ADMIN
      # - SYS_MODULE  # optional already active
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Paris
      - LOG_CONFS=true
    volumes:
      - /etc/container-conf/wireguard/appdata/config:/config
      - /etc/container-conf/wireguard/lib/modules:/lib/modules
    ports:
      - 51820:51820/udp
```

```
    restart: unless-stopped
    sysctls:
      - net.ipv4.conf.all.src_valid_mark=

volumes:
 wireguard:
```

2 - Via l'utilisation du package Linux

Via l'utilisation du package Wireguard

Sur le client

1/ Tout d'abord vous devez installer wireguard sur la machine :

```
sudo apt-get install wireguard
```

Puis se déplacer dans le répertoire de wireguard, par défaut le dossier est vide :

```
root@linux: cd /etc/wireguard/
root@linux: ls -al
total 28
drwx------   2 root root  4096 juin  13 12:03 ./
drwxr-xr-x 166 root root 12288 juin  13 11:24 ../
```

2/ Maintenant nous allons générer la clé privé et la clé publique :

```
wg genkey | tee privatekey | wg pubkey | tee publickey
```

Nous avons maintenant 2 fichiers dans le dossier wireguard :

```
root@linux: wg genkey | tee privatekey | wg pubkey | tee publickey
WKtF71pM6wObswaXkbxJgwPhk8a6lqrPb9oKFjaGOmM=
root@linux: ls -al
total 28
drwx------   2 root root  4096 juin  14 15:55 .
drwxr-xr-x 166 root root 12288 juin  13 11:24 ..
-rw-r--r--   1 root root    45 juin  14 15:55 privatekey
-rw-r--r--   1 root root    45 juin  14 15:55 publickey
```

3/ Pour continuer la configuration il faut noter le contenu des 2 clés générés :

```
root@linux: cat privatekey
iApdu05JqNGSp/r7PSpJ5Zqxs4kWSR6qYv9onitvsmo=

root@linux: cat publickey
WKtF71pM6wObswaXkbxJgwPhk8a6lqrPb9oKFjaGOmM=
```

> Attention ne partagez jamais votre clé privé !!

Clé privé : iApdu05JqNGSp/r7PSpJ5Zqxs4kWSR6qYv9onitvsmo=
Clé publique : WKtF71pM6wObswaXkbxJgwPhk8a6lqrPb9oKFjaGOmM=

4/a) Création du fichier de configuration Wireguard client :

```
touch wg0.conf
```

4/b) Préparer dans un bloc note la configuration suivante :

Il faut bien veiller à remplacer les variables :

- `<private_key_client>` -> Clé privé généré précédemment
- `<range_ip_vpn>` -> Adresse IP VPN du client
- `<public_key_server>` -> Clé publique du serveur (A récupérer sur le serveur)
- `<ip_server>` -> Adresse IP du serveur
- `<port_server>` -> Port écoutant sur le serveur

```
[Interface]
PrivateKey = <private_key_client>
Address = <range_ip_vpn>

[Peer]
###Public of the WireGuard VPN Server
PublicKey = <public_key_server>

### IP and Port of the WireGuard VPN Server
Endpoint = <ip_server>:<port_server>

### Allow all traffic
AllowedIPs = 0.0.0.0/0
```

4/c) Editer le fichier de configuration en collant la configuration complété :

```
nano wg0.conf
# ou
vim wg0.conf
```

Ce qui donne :

```
[Interface]
PrivateKey = iApdu05JqNGSp/r7PSpJ5Zqxs4kWSR6qYv9onitvsmo=
Address = 192.168.10.15/24

[Peer]
###Public of the WireGuard VPN Server
PublicKey = <public_key_server>

### IP and Port of the WireGuard VPN Server
Endpoint = vpn.domaine-name.com:51820

### Allow all traffic
AllowedIPs = 0.0.0.0/0
```

> Le reste de la configuration continue sur la partie serveur

Sur le serveur

> `wg0` est l'interface référençant tous les appareils pouvant se connecter au VPN.

Préparer la commande suivante :

```
wg set wg0 peer <public_key_client> allowed-ips <adress_ip_vpn_client>
```

Ce qui donne :

```
wg set wg0 peer WKtF71pM6wObswaXkbxJgwPhk8a6lqrPb9oKFjaGOmM= allowed-ips 192.168.10.15
```

Puis exécuter la commande.

Pour être sur du bon fonctionnement de la commande précédente, nous pouvons lister les clients enregistré dans le fichier de conf du serveur :

```
wg show wg0
```

Maintenant il faut confirmer et enregistrer la modification du fichier de configuration :

```
wg-quick save wg0
```

Sur le client

Nous avons créer le client et nous l'avons enregistré dans la configuration serveur. Maintenant il nous reste plus qu'a démarrer le VPN sur le client :

```
root@linux: wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.13.13.6/32 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] wg set wg0 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev wg0 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] nft -f /dev/fd/63
```

Vérification du fonctionnement du VPN côté client :

```
ifconfig
```

Vérification du fonctionnement du nouveau client côté serveur :

```
wg show wg0
```

3 - via le logiciel Windows

source : https://www.youtube.com/watch?v=u8w6jIdU39s&t=0s

Diagnostic WireGuard — Linux avec NetworkManager (GNOME)

Guide de diagnostic pour les problèmes de connectivité WireGuard sur Linux Ubuntu avec NetworkManager, notamment l'absence d'accès aux services du réseau local du serveur VPN.

Symptômes typiques

- Internet fonctionne via le VPN (`ping 8.8.8.8` OK)
- Le serveur VPN est joignable (`ping 10.13.13.1` OK)
- Les services accessibles uniquement via VPN (ex: `192.168.1.x`, `10.13.13.x`) sont **inaccessibles**
- Le même profil fonctionne sur un autre appareil (téléphone, autre OS)

Étape 1 — Vérifier l'état du tunnel WireGuard

```
sudo wg show
```

Vérifier que :

- L'interface est bien active
- Un `latest handshake` récent est visible (tunnel actif)
- `allowed ips` contient bien `0.0.0.0/0, ::/0` ou les plages nécessaires

Étape 2 — Vérifier les routes appliquées

```
# Routes globales (le VPN doit apparaître ici)
ip route show

# Routes spécifiques à l'interface WireGuard (remplacer "Home" par le nom de ton interface)
ip route show dev Home
```

Problème détecté si : `ip route show dev Home` ne retourne rien malgré un tunnel actif.

Cela indique que NetworkManager n'a pas injecté les routes dans la table de routage principale — bug connu de NetworkManager avec WireGuard.

Étape 3 — Vérifier les règles de routage

```
ip rule show
```

Exemple de sortie révélant le problème :

```
0:      from all lookup local
31076:  from all lookup main suppress_prefixlength 0
31077:  not from all fwmark 0xcb8e lookup 52110
32766:  from all lookup main
32767:  from all lookup default
```

La règle `not from all fwmark 0xcb8e lookup 52110` indique que WireGuard utilise une table de routage séparée. Vérifier son contenu :

```
ip route show table 52110
```

Étape 4 — Vérifier le DNS

```
resolvectl status
```

Vérifier que l'interface WireGuard apparaît avec le bon serveur DNS (ex: `10.13.13.1`).

```
# Test de résolution DNS via le tunnel
dig @10.13.13.1 monservice.domaine.local
```

Étape 5 — Tester la connectivité par IP directe

Avant de corriger, tester si le problème est DNS ou routage :

```
# Ping du serveur VPN
ping 10.13.13.1

# Accès direct par IP à un service (sans nom de domaine)
curl -v http://IP_DU_SERVICE:PORT
```

- Si ça fonctionne par IP → problème DNS uniquement
- Si ça échoue par IP → problème de routage (continuer ci-dessous)

Correction — Ajouter les routes manquantes

Cas 1 : Le service est sur le réseau WireGuard (`10.13.13.x`)

```
sudo nmcli connection modify <NOM_CONNEXION> +ipv4.routes "10.13.13.0/24 10.13.13.1"
sudo nmcli connection reload
```

Cas 2 : Le service est sur le LAN du serveur (`192.168.1.x` par exemple)

```
sudo nmcli connection modify <NOM_CONNEXION> +ipv4.routes "192.168.1.0/24 10.13.13.1"
sudo nmcli connection reload
```

> ⚠ **Attention** : si ton PC est lui-même sur `192.168.1.x`, ajouter cette route crée un conflit. Dans ce cas, les services doivent être exposés directement sur l'IP WireGuard du serveur (`10.13.13.1`).

Cas 3 : Tout le trafic doit passer par le VPN (full tunnel)

```
sudo nmcli connection modify <NOM_CONNEXION> +ipv4.routes "0.0.0.0/0 10.13.13.1"
sudo nmcli connection reload
```

> ⚠ Cette route remplace la route par défaut. Si elle provoque une perte d'internet, la supprimer immédiatement (voir ci-dessous).

Annuler une modification (rollback)

```
# Supprimer une route ajoutée par erreur
sudo nmcli connection modify <NOM_CONNEXION> -ipv4.routes "CIDR GATEWAY"

# Exemple
sudo nmcli connection modify Home -ipv4.routes "0.0.0.0/0 10.13.13.1"

sudo nmcli connection reload
```

Puis déconnecter et reconnecter le VPN depuis GNOME.

Vérifier la configuration stockée

Les configs NetworkManager sont dans `/etc/NetworkManager/system-connections/` :

```
sudo ls /etc/NetworkManager/system-connections/
sudo cat /etc/NetworkManager/system-connections/<NOM>.nmconnection
```

Exemple de bloc `[ipv4]` correct avec routes :

```
[ipv4]
address1=10.13.13.3/32
dns=10.13.13.1;
method=manual
route1=192.168.1.0/24,10.13.13.1
```

Checklist récapitulatif

Vérification	Commande	Résultat attendu
Tunnel actif	`sudo wg show`	`latest handshake` récent
Routes VPN présentes	`ip route show dev <interface>`	Au moins une route affichée
Règles de routage	`ip rule show`	Table WireGuard présente
DNS correct	`resolvectl status`	DNS = IP du serveur VPN
Ping serveur VPN	`ping 10.13.13.1`	Réponse OK
Accès service par IP	`curl -v http://IP:PORT`	Connexion établie

Causes racines fréquentes

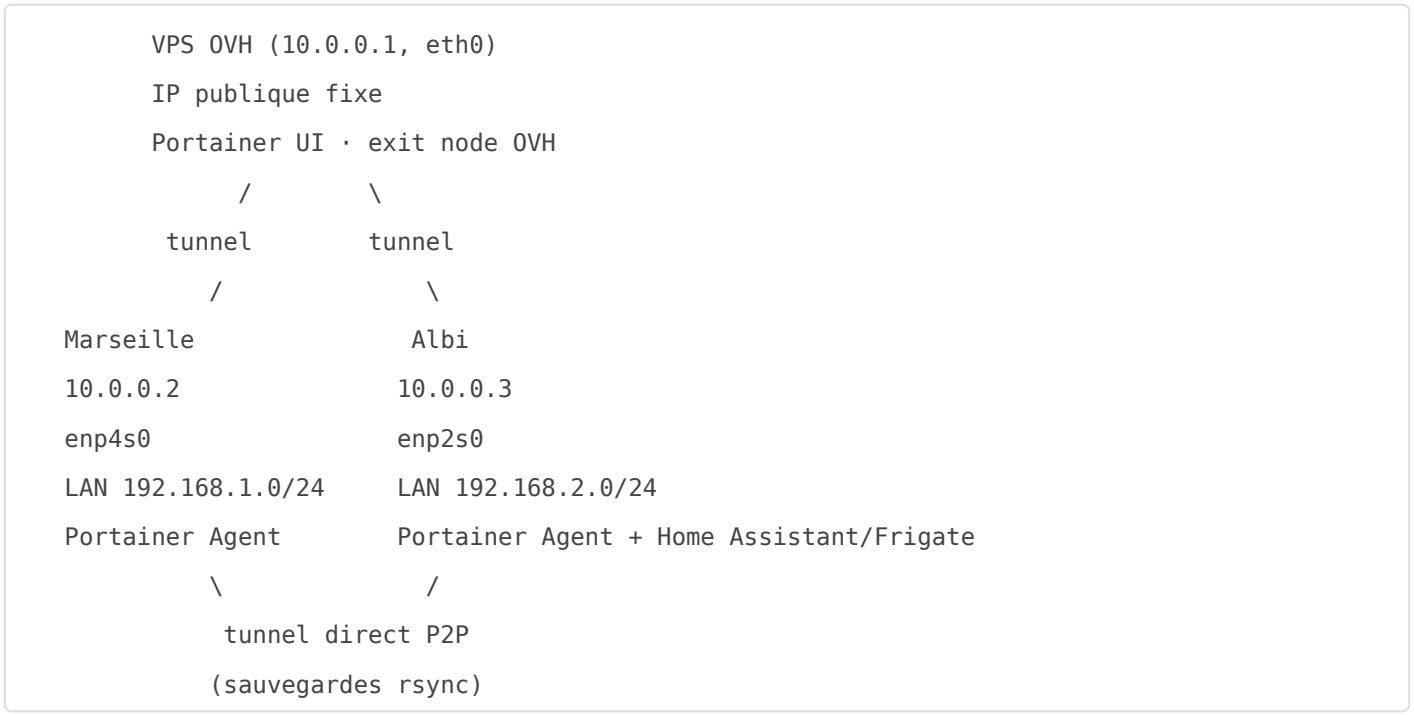
Symptôme	Cause probable
`ip route show dev <iface>` vide	NetworkManager n'injecte pas les routes (bug NM+WireGuard)
DNS OK mais IP inaccessible	Réseau cible absent des `AllowedIPs` ou routes manquantes
IP accessible mais domaine KO	DNS mal configuré ou non utilisé par le système
Internet coupé après modification	Route `0.0.0.0/0` en conflit avec route par défaut WiFi

Configuration WireGuard Mesh — Documentation complète

Architecture WireGuard Mesh

Architecture

Les 3 serveurs forment un **mesh** : chacun parle directement aux deux autres. Marseille et Albi peuvent communiquent en direct, sans passer par OVH.



Les besoins fonctionnelles & techniques

Cette architecture couvre des besoins bien définit qui pourrait résoudre des problématiques de sécurité entre plusieurs serveurs et services.

Voici la liste des besoin couvert par cette architecture :

- 1 - Se connecter au VPN des différents serveur
  - Hériter de l'adresse IP publique associé
  - Accéder au réseau local associé au serveur
  - Ne pas accéder au réseau local des AUTRES serveurs
  - Par exemple, je suis connecté au VPN du serveur de Marseille. J'ai alors l'adresse IP public de la box internet de MRS, j'ai accès à tous les appareils du réseau local de la box internet. Par contre je ne peux pas voir les autres appareils du réseau local de la box d'Albi.
- 2 - Faire communiquer les différents serveurs dans un tunnel privé et sécurisé

    - Chaque serveur est joignable via ce réseau privé
    - Les services Docker peuvent utiliser ce réseau privé pour communiquer avec les services des autres serveurs
    - Ce tunnel privé ne doit pas faire partie de la même configuration que le besoin 1
- 3 - Cette architecture doit fonctionner avec un seul et même Wireguard Docker par serveur

Les définitions

Mesh (ou réseau maillé) est une topologie réseau où **chaque nœud est connecté directement à plusieurs autres**, sans point central obligatoire. Dans notre cas, les 3 serveurs forment un mesh : OVH parle à Marseille, OVH parle à Albi, et Marseille parle directement à Albi. C'est l'opposé du hub-and-spoke où tout passe obligatoirement par le centre. L'avantage : si OVH tombe, Marseille et Albi continuent de se parler directement.

Exit Node est le nœud par lequel **votre trafic internet "sort"** vers le monde. Quand vous activez un profil exit node, vous dites à votre appareil "fais passer tout ton trafic internet par ce serveur". Le site que vous visitez voit alors l'IP publique du serveur exit node, pas votre vraie IP. C'est exactement le principe d'un VPN classique grand public (NordVPN, ExpressVPN…), sauf qu'ici vous gérez vous-même le serveur de sortie. Le terme vient de Tailscale qui l'a popularisé, mais le mécanisme (`AllowedIPs = 0.0.0.0/0` + masquerading) existe depuis les débuts de WireGuard.

Structure des fichiers

```
├── Configuration servers/
│   ├── ovh-wg0.conf           # Config WireGuard du VPS OVH
│   ├── marseille-wg0.conf     # Config WireGuard de Marseille
│   └── albi-wg0.conf          # Config WireGuard d'Albi
├── Confiiguration clients/
│   ├── ovh-exitnode.conf      # Client : IP publique = OVH
│   ├── mrs-exitnode.conf      # Client : IP publique = Marseille
│   ├── albi-exitnode.conf     # Client : IP publique = Albi
│   └── private-only.conf      # Client : réseau privé uniquement
├── Configuration Docker
│   ├── docker-compose-all.yml # docker-compose OVH + Marseille/Albi
├── Configuration Par-feu
│   ├── ovh-ufw.sh           # Config UFW du VPS OVH
│   ├── marseille-ufw.sh     # Config UFW de Marseille
│   └── albi-ufw.sh          # Config UFW d'Albi
```

Plan d'adressage complet

Matrice de sécurité

Profil client	IP mesh	Internet	Mesh 10.0.0.x	LAN MRS 192.168.1.x	LAN Albi 192.168.2.x
ovh-exitnode	10.0.0.10	IP OVH	✓	✗	✗
mrs-exitnode	10.0.0.20	IP MRS	✓	✓	✗
albi-exitnode	10.0.0.30	IP Albi	✓	✗	✓
private-only	10.0.0.40	propre	✓	✗	✗
Serveur OVH	10.0.0.1	—	✓	✗	✗
Serveur MRS	10.0.0.2	—	✓	—	✗
Serveur Albi	10.0.0.3	—	✓	✗	—

Double protection :

- (1) les LAN ne sont dans les AllowedIPs d'aucun peer serveur → WireGuard refuse de les router ;
- (2) UFW pose FORWARD = DROP et n'autorise le LAN qu'à l'IP exit-node du site concerné.

Serveurs (peers fixes)

Serveur	IP WireGuard	IP publique	LAN local	Commentaire
VPS OVH	10.0.0.1	<IP_PUBLIQUE_OVH>	—	—
Marseille	10.0.0.2	<IP_PUBLIQUE_MRS>	192.168.1.0/24	—
Albi	10.0.0.3	<IP_PUBLIQUE_ALBI>	192.168.2.0/24	—

Clients (IP réservées)

IP	Usage	Profil	Commentaire
10.0.0.10	Exit node OVH	ovh-exitnode.conf	IP publique = OVH
10.0.0.20	Exit node Marseille	mrs-exitnode	IP publique = Marseille + LAN
10.0.0.30	Exit node Albi	albi-exitnode	IP publique = Albi + LAN
10.0.0.40	Réseau privé OVH	private-only.conf	Réseau privé seul (Mesh), entrée par OVH
10.0.0.41	Réseau privé MRS	private-only.conf	Réseau privé seul (Mesh), entrée par MRS

IP	Usage	Profil	Commentaire
10.0.0.42	Réseau privé Albi	private-only.conf	Réseau privé seul (Mesh), entrée par Albi

---

Commandes utiles

Vérifier l'état des tunnels

```
# Sur n'importe quel serveur
docker exec wireguard-ovh wg show       # OVH
docker exec wireguard-mrs wg show       # Marseille
docker exec wireguard-albi wg show      # Albi

# Résultat attendu : chaque peer doit avoir un "latest handshake" récent
# et un compteur de bytes "received/sent" qui monte
```

Tester la connectivité mesh

```
# Depuis OVH, pinger Marseille et Albi
docker exec wireguard-ovh ping 10.0.0.2 -c 3
docker exec wireguard-ovh ping 10.0.0.3 -c 3

# Depuis Marseille, pinger Albi directement (tunnel P2P)
docker exec wireguard-mrs ping 10.0.0.3 -c 3

# Vérifier que le LAN de Marseille est accessible depuis Albi
docker exec wireguard-albi ping 192.168.1.1 -c 3
```

Vérifier que Portainer UI n'est pas exposé

```
# Cette commande doit échouer (connexion refusée) depuis internet
curl -v http://<IP_PUBLIQUE_OVH>:9000

# Ceci doit fonctionner uniquement avec le VPN actif
curl -v http://10.0.0.1:9000
```

Tester une sauvegarde rsync inter-serveurs

```
# Depuis Marseille vers Albi (tunnel P2P direct)
docker run --rm --network host \
  -v /data/backups:/source:ro \
  instrumentisto/rsync-ssh \
  rsync -avz --progress /source/ user@10.0.0.3:/backup/marseille/

# Vérifier que le trafic ne passe PAS par OVH :
# Sur OVH pendant le rsync : docker exec wireguard-ovh wg show
# → les bytes du peer MRS et Albi ne doivent PAS augmenter
```

Générer un QR code pour mobile (iOS/Android)

```
# Installer qrencode
apt install qrencode

# Générer le QR pour le profil OVH exit node
qrencode -t ansiutf8 < clients/ovh-exitnode.conf

# Scanner avec l'app WireGuard sur le téléphone
```

Redémarrer un tunnel après modification de config

```
docker compose restart wireguard
# OU
docker exec wireguard-ovh wg syncconf wg0 <(wg-quick strip wg0)
```

Sécurité — points d'attention

1. **Ne jamais committer les clés privées** dans Git. Ajouter `*privatekey*` au `.gitignore`.
2. **Portainer UI bind sur 10.0.0.1** uniquement — vérifier avec `ss -tlnp | grep 9000`.
3. **Portainer Agent bind sur 10.0.0.x** uniquement — même vérification sur chaque serveur.
4. **Les fichiers .conf clients** contiennent des clés privées — les traiter comme des mots de passe.
5. **Renouveler les clés** si un appareil client est perdu ou compromis : supprimer son [Peer] de tous les serveurs et faire `docker compose restart wireguard`.
6. Caveat UFW + Docker : Docker contourne UFW pour les ports qu'il *publie*. Ici Portainer est bindé sur 10.0.0.x → déjà limité au mesh. Pour tout futur conteneur, ne pas supposer qu'UFW protège un `ports: "8080:8080"`.

Processus de déploiement

Étape 1 — Générer les clés

Sur chaque serveur, générer une paire de clés :

```
# Sur OVH
wg genkey | tee /etc/wireguard/privatekey-ovh | wg pubkey > /etc/wireguard/publickey-ovh

# Sur Marseille
wg genkey | tee /etc/wireguard/privatekey-mrs | wg pubkey > /etc/wireguard/publickey-mrs

# Sur Albi
wg genkey | tee /etc/wireguard/privatekey-albi | wg pubkey > /etc/wireguard/publickey-albi

# Pour chaque client
wg genkey | tee privatekey-client | wg pubkey > publickey-client
```

Étape 2 — Remplacer les placeholders

Dans tous les fichiers .conf, remplacer :

- `<CLE_PRIVEE_OVH>` → contenu de privatekey-ovh
- `<CLE_PUBLIQUE_OVH>` → contenu de publickey-ovh
- `<CLE_PRIVEE_MRS>` → contenu de privatekey-mrs
- `<CLE_PUBLIQUE_MRS>` → contenu de publickey-mrs
- `<CLE_PRIVEE_ALBI>` → contenu de privatekey-albi
- `<CLE_PUBLIQUE_ALBI>` → contenu de publickey-albi
- `<IP_PUBLIQUE_OVH>` → IP publique du VPS OVH
- `<IP_PUBLIQUE_MRS>` → IP publique de la box Marseille
- `<IP_PUBLIQUE_ALBI>` → IP publique de la box Albi
- (idem pour les clés clients)

Étape 3 — Configuration des box internet

Box Marseille & Box Albi

Effectuer dans l'interface admin de chaque box :

Port forwarding :

```
Protocole : UDP
Port externe : 51820
IP destination : 192.168.X.100 (IP locale du serveur)
Port destination : 51820
```

Étape 4 — Déployer dans l'ordre

```
# 1. Démarrer le hub OVH en premier
ssh ovh "cd /opt/wireguard && docker compose up -d"

# 2. Démarrer Marseille
ssh marseille "cd /opt/wireguard && docker compose up -d"

# 3. Démarrer Albi
ssh albi "cd /opt/wireguard && docker compose up -d"

# Une fois wg0 créé, appliquer le pare-feu :
sudo bash ovh-ufw-setup.sh         # sur OVH
sudo bash marseille-ufw-setup.sh   # sur Marseille
sudo bash albi-ufw-setup.sh        # sur Albi
```

Pourquoi le conteneur WireGuard a besoin de host ?

Le point dur, c'est le **routage et le NAT au niveau de l'hôte**. Tes exit-nodes et l'accès LAN reposent sur le fait que le trafic traverse le serveur entre `wg0` et l'interface physique (`enp4s0`, etc.), avec masquerading. Pour que ça fonctionne, `wg0` doit vivre dans le namespace réseau de l'hôte et UFW doit pouvoir filtrer ce trafic au niveau hôte.

Si tu mets WireGuard dans un réseau Docker isolé, `wg0` est créé dans le namespace du conteneur. Tu peux router le mesh entre conteneurs, mais dès que tu veux :

- faire sortir un client sur internet via l'IP publique du serveur (exit node),
- ou atteindre le LAN physique `192.168.x.0/24`,

tu dois faire ressortir le trafic du namespace conteneur vers l'hôte puis vers l'interface physique. C'est faisable mais ça demande des `iptables` supplémentaires et une gymnastique de routage qui annule l'intérêt de l'isolation. **Pour le rôle de passerelle VPN, host est le choix pragmatique.**

Étape 5 — Vérification de sécurité

Note Docker (importante pour Home Assistant)

Après `ufw enable`, si les conteneurs (Home Assistant, Frigate…) perdent l'accès réseau, redémarrer Docker **une seule fois** pour qu'il replace ses règles FORWARD au-dessus de celles d'UFW :

```
sudo systemctl restart docker
```

À faire à un moment calme (cela redémarre HA et Frigate). En pratique HA en host n'a pas besoin de la chaîne FORWARD, donc ce redémarrage est rarement nécessaire — mais c'est le réflexe si un conteneur perd le réseau.

Vérifier les règles Par-feu

```
ufw status verbose
ufw route status                        # liste les règles FORWARD VPN
iptables -t nat -L POSTROUTING -n | grep MASQUERADE
```

Vérifier que Portainer n'est PAS exposé sur l'IP publique

```
# Depuis internet (doit échouer — connexion refusée ou timeout)
curl -v --max-time 5 http://<IP_PUBLIQUE_OVH>:9000

# Depuis le VPN avec private-only actif (doit fonctionner)
curl -v http://10.0.0.1:9000
```

Tester le cloisonnement LAN

```
# Avec private-only.conf actif, tenter d'atteindre le LAN de Marseille
ping 192.168.1.1    # doit échouer (pas de route + iptables DROP)
ping 192.168.2.1    # doit échouer

# Avec mrs-exitnode.conf actif
ping 192.168.1.1    # doit répondre (autorisé)
ping 192.168.2.1    # doit échouer (bloqué)

# Avec albi-exitnode.conf actif
ping 192.168.1.1    # doit échouer
ping 192.168.2.1    # doit répondre (autorisé)
```

Vérifier l'état du mesh

```
# Sur OVH — doit lister MRS et Albi avec handshake récent
docker exec wireguard-ovh wg show

# Exemple de sortie attendue :
# peer: <CLE_PUBLIQUE_MRS>
#   endpoint: <IP_MRS>:51820
#   allowed ips: 10.0.0.2/32
#   latest handshake: X seconds ago
#   transfer: X MiB received, X MiB sent
```

Vérifier les sauvegardes rsync (tunnel direct P2P)

```
# Depuis Marseille vers Albi — ne doit PAS faire monter les compteurs OVH
docker exec wireguard-mrs rsync -avz /data/ user@10.0.0.3:/backup/

# Pendant le rsync, vérifier sur OVH que les bytes N'augmentent PAS :
watch -n1 "docker exec wireguard-ovh wg show | grep -A4 'marseille\|albi'"
```

Étape 6 — Configurer Portainer UI

1. Ouvrir http://10.0.0.1:9000 avec le profil private-only.conf actif
2. Aller dans Environments > Add environment > Agent
3. Ajouter Marseille : URL = `10.0.0.2:9001`, nom = "Marseille"
4. Ajouter Albi : URL = `10.0.0.3:9001`, nom = "Albi"

Configuration Wireguard Docker

Docker compose

```
# ==============================================================================
# docker-compose.yml — VPS OVH  (WireGuard Hub + Portainer UI)
# Version 3 — pare-feu géré par UFW (pas de règles dans wg0.conf)
# ==============================================================================
# Arborescence :
#   /opt/wireguard-ovh/
#   ├── docker-compose.yml
#   └── wg-config/wg0.conf      (= servers/ovh-wg0.conf renommé)
#
# Démarrage :
#   docker compose up -d
#   sudo bash ovh-ufw-setup.sh   # APRÈS, pour que wg0 existe
# ==============================================================================

services:

  wireguard:
    image: lscr.io/linuxserver/wireguard:latest
    container_name: wireguard-ovh
    # network_mode: host — cohabite sans souci avec d'autres conteneurs host
    # (ex : Home Assistant sur Albi). Indispensable pour que wg0 et le
    # routage soient au niveau de l'hôte, où UFW applique ses règles.
    network_mode: host
    cap_add:
      - NET_ADMIN
      - SYS_MODULE
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Paris
    volumes:
      - ./wg-config:/config
      # Accès au module kernel wireguard de l'hôte
```

```
      - /lib/modules:/lib/modules:ro
    restart: unless-stopped

  portainer:
    image: portainer/portainer-ce:latest
    container_name: portainer-ui
    # Bind sur l'IP WireGuard uniquement : invisible depuis l'IP publique OVH.
    ports:
      - "10.0.0.1:9000:9000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer-data:/data
    restart: unless-stopped
    depends_on:
      - wireguard

volumes:
  portainer-data:

---
# ================================================================================
# docker-compose.yml — Serveur Marseille  (WireGuard + Portainer Agent)
# wg-config/wg0.conf = servers/marseille-wg0.conf
# Lancer ensuite : sudo bash marseille-ufw-setup.sh
# ================================================================================

services:

  wireguard:
    image: lscr.io/linuxserver/wireguard:latest
    container_name: wireguard-mrs
    network_mode: host
    cap_add:
      - NET_ADMIN
      - SYS_MODULE
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Paris
    volumes:
```

```
      - ./wg-config:/config
      - /lib/modules:/lib/modules:ro
    restart: unless-stopped

  portainer-agent:
    image: portainer/agent:latest
    container_name: portainer-agent
    # Bind sur l'IP WireGuard de Marseille — joignable seulement via le mesh.
    ports:
      - "10.0.0.2:9001:9001"
    environment:
      - AGENT_PORT=9001
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/volumes:/var/lib/docker/volumes
    restart: unless-stopped
    depends_on:
      - wireguard

---
# ================================================================================
# docker-compose.yml — Serveur Albi  (WireGuard + Portainer Agent)
# wg-config/wg0.conf = servers/albi-wg0.conf
# Lancer ensuite : sudo bash albi-ufw-setup.sh
#
# Home Assistant / Frigate tournent déjà sur ce serveur : aucun changement
# requis pour eux. WireGuard en host cohabite avec HA en host.
# ================================================================================

services:

  wireguard:
    image: lscr.io/linuxserver/wireguard:latest
    container_name: wireguard-albi
    network_mode: host
    cap_add:
      - NET_ADMIN
      - SYS_MODULE
    environment:
      - PUID=1000
```

```
      - PGID=1000
      - TZ=Europe/Paris
    volumes:
      - ./wg-config:/config
      - /lib/modules:/lib/modules:ro
    restart: unless-stopped

  portainer-agent:
    image: portainer/agent:latest
    container_name: portainer-agent
    # Bind sur l'IP WireGuard d'Albi.
    ports:
      - "10.0.0.3:9001:9001"
    environment:
      - AGENT_PORT=9001
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/volumes:/var/lib/docker/volumes
    restart: unless-stopped
    depends_on:
      - wireguard
```

Configuration Serveurs

Serveur OVH

```
# ================================================================================
# WireGuard — VPS OVH (Hub du mesh)
# IP WireGuard : 10.0.0.1/24
# Rôle : hub + exit node OVH + Portainer UI + entrée private-only (10.0.0.40)
# ================================================================================
#
#   - Les LAN (192.168.1.0/24 et 192.168.2.0/24) ne font pas parti des AllowedIPs
#     des peers serveurs. OVH ne "connaît" plus les LAN des sites physiques.
#   - La politique FORWARD passe à DROP par défaut : seul le trafic
#     explicitement autorisé est routé. Tout le reste est silencieusement jeté.
#   - Conséquence : un client connecté à OVH ne peut PAS atteindre
#     192.168.1.x ou 192.168.2.x, même s'il connaît l'adresse.
#
# PRÉREQUIS :
#   - Port UDP 51820 ouvert dans le firewall OVH ET dans UFW
#   - Vérifier le nom de l'interface réseau : ip link show
#     (peut être eth0, ens3, ens4, enp1s0 selon le VPS)
#   - ip_forward activé au niveau hôte (fait par le script UFW)
#
# ================================================================================

[Interface]
Address    = 10.0.0.1/24
ListenPort = 51820
PrivateKey = <CLE_PRIVEE_OVH>

# ================================================================================
# PEER : Serveur Marseille
# ================================================================================
[Peer]
PublicKey = <CLE_PUBLIQUE_MRS>
Endpoint  = <IP_PUBLIQUE_MRS>:51820
```

```
#   10.0.0.2/32   → IP WireGuard de MRS (mesh)
#   10.0.0.20/32  → client mrs-exitnode : son trafic retour repasse par MRS
#   10.0.0.41/32  → client private-only entré via MRS : retour via MRS
# Pas de 192.168.1.0/24 : OVH ne route jamais vers le LAN de Marseille.
AllowedIPs          = 10.0.0.2/32, 10.0.0.20/32, 10.0.0.41/32
PersistentKeepalive = 25


# ================================================================================
# PEER : Serveur Albi
# ================================================================================
[Peer]
PublicKey = <CLE_PUBLIQUE_ALBI>
Endpoint  = <IP_PUBLIQUE_ALBI>:51820

#   10.0.0.3/32   → IP WireGuard d'Albi (mesh)
#   10.0.0.30/32  → client albi-exitnode : retour via Albi
#   10.0.0.42/32  → client private-only entré via Albi : retour via Albi
AllowedIPs          = 10.0.0.3/32, 10.0.0.30/32, 10.0.0.42/32
PersistentKeepalive = 25


# ================================================================================
# PEER : Client ovh-exitnode
# Tout le trafic internet sort par OVH. IP publique = VPS OVH.
# ================================================================================
[Peer]
PublicKey  = <CLE_PUBLIQUE_CLIENT_OVH_1>
AllowedIPs = 10.0.0.10/32


# ================================================================================
# PEER : Client private-only entré via OVH
# ================================================================================
[Peer]
PublicKey  = <CLE_PUBLIQUE_CLIENT_PRIVATE_OVH>
AllowedIPs = 10.0.0.40/32
```

Serveur Marseille

```
# ================================================================================
# WireGuard — Serveur physique Marseille (Spoke)
# IP WireGuard : 10.0.0.2/24
# LAN local    : 192.168.1.0/24
# Rôle         : exit node MRS (+ accès LAN MRS) + Portainer Agent
#                + entrée private-only (10.0.0.41) + sauvegardes P2P avec Albi
# ================================================================================
#
# ACCÈS AU LAN 192.168.1.0/24 :
#   Autorisé UNIQUEMENT au client mrs-exitnode (10.0.0.20) via UFW.
#   Voir ufw/marseille-ufw-setup.sh pour les règles de filtrage.
#
#   Tout autre peer (OVH, Marseille, clients exit-MRS, private-only…)
#   est bloqué par le Pare-feu.
#
# PRÉREQUIS SUR LA BOX :
#   1. Port forwarding UDP 51820 → 192.168.1.100 (IP locale du serveur)
#
# ================================================================================

[Interface]
Address    = 10.0.0.2/24
ListenPort = 51820
PrivateKey = <CLE_PRIVEE_ALBI>



# ================================================================================
# PEER : VPS OVH
# ================================================================================
[Peer]
PublicKey = <CLE_PUBLIQUE_OVH>
Endpoint  = <IP_PUBLIQUE_OVH>:51820

# Destinations routées VERS OVH depuis Marseille :
#   10.0.0.1/32   → IP WireGuard d'OVH (mesh, Portainer UI)
#   10.0.0.10/32  → client ovh-exitnode : retour via OVH
#   10.0.0.40/32  → client private-only entré via OVH : retour via OVH
AllowedIPs          = 10.0.0.1/32, 10.0.0.10/32, 10.0.0.40/32
PersistentKeepalive = 25
```

```
# ================================================================================
# PEER : Serveur Albi (tunnel P2P direct — sauvegardes sans passer par OVH)
# ================================================================================
[Peer]
PublicKey = <CLE_PUBLIQUE_ALBI>
Endpoint  = <IP_PUBLIQUE_ALBI>:51820

# Destinations routées VERS Albi :
#   10.0.0.3/32   → IP WireGuard d'Albi (sauvegardes rsync/SSH)
#   10.0.0.30/32  → client albi-exitnode (symétrie mesh)
#   10.0.0.42/32  → client private-only entré via Albi
# Pas de 192.168.2.0/24 : MRS ne route jamais vers le LAN d'Albi.
AllowedIPs          = 10.0.0.3/32, 10.0.0.30/32, 10.0.0.42/32
PersistentKeepalive = 25



# ================================================================================
# PEER : Client mrs-exitnode (IP publique = MRS, accès LAN MRS)
# ================================================================================
[Peer]
PublicKey  = <CLE_PUBLIQUE_CLIENT_MRS>
AllowedIPs = 10.0.0.20/32



# ================================================================================
# PEER : Client private-only entré via MRS
# ================================================================================
[Peer]
PublicKey  = <CLE_PUBLIQUE_CLIENT_PRIVATE_MRS>
AllowedIPs = 10.0.0.41/32
```

Serveur Albi

```
# ================================================================================
# WireGuard — Serveur physique Albi (Spoke)
# IP WireGuard : 10.0.0.3/24
# Interface physique : enp2s0
# LAN local : 192.168.2.0/24
# Rôle : exit node Albi (+ accès LAN Albi) + Portainer Agent
```

```
#        + entrée private-only (10.0.0.42) + sauvegardes P2P avec MRS
# ================================================================================
#
# COHABITATION AVEC HOME ASSISTANT / FRIGATE :
#   WireGuard tourne en network_mode: host, comme Home Assistant.
#   Aucun conflit : plusieurs conteneurs peuvent partager la pile réseau
#   de l'hôte. Le filtrage FORWARD posé par UFW n'affecte pas la
#   communication HA <-> Frigate (trafic local ou géré par Docker),
#   il ne filtre que le trafic VPN routé.
#
# ACCÈS AU LAN 192.168.2.0/24 :
#   Autorisé UNIQUEMENT au client albi-exitnode (10.0.0.30) via UFW.
#
# PRÉREQUIS BOX ALBI :
#   - Port forwarding UDP 51820 → 192.168.2.100 (IP locale du serveur)
#
# ================================================================================

[Interface]
Address    = 10.0.0.3/24
ListenPort = 51820
PrivateKey = <CLE_PRIVEE_ALBI>



# ================================================================================
# PEER : VPS OVH
# ================================================================================
[Peer]
PublicKey = <CLE_PUBLIQUE_OVH>
Endpoint  = <IP_PUBLIQUE_OVH>:51820

#   10.0.0.1/32   → IP WireGuard d'OVH
#   10.0.0.10/32  → client ovh-exitnode : retour via OVH
#   10.0.0.40/32  → client private-only entré via OVH
AllowedIPs          = 10.0.0.1/32, 10.0.0.10/32, 10.0.0.40/32
PersistentKeepalive = 25



# ================================================================================
# PEER : Serveur Marseille (tunnel P2P direct)
```

```
# ================================================================================
[Peer]
PublicKey = <CLE_PUBLIQUE_MRS>
Endpoint  = <IP_PUBLIQUE_MRS>:51820

#   10.0.0.2/32   → IP WireGuard de MRS (sauvegardes)
#   10.0.0.20/32  → client mrs-exitnode (symétrie mesh)
#   10.0.0.41/32  → client private-only entré via MRS
# Pas de 192.168.1.0/24 : Albi ne route jamais vers le LAN de Marseille.
AllowedIPs          = 10.0.0.2/32, 10.0.0.20/32, 10.0.0.41/32
PersistentKeepalive = 25



# ================================================================================
# PEER : Client albi-exitnode (IP publique = Albi, accès LAN Albi)
# ================================================================================
[Peer]
PublicKey  = <CLE_PUBLIQUE_CLIENT_ALBI>
AllowedIPs = 10.0.0.30/32



# ================================================================================
# PEER : Client private-only entré via Albi
# ================================================================================
[Peer]
PublicKey  = <CLE_PUBLIQUE_CLIENT_PRIVATE_ALBI>
AllowedIPs = 10.0.0.42/32
```

Configuration Clients

Client Mesh

```
# ================================================================================
# Profil client — Réseau privé mesh uniquement (private-only)
# IP publique vue : INCHANGÉE (la propre connexion du client)
# Accès LAN : AUCUN (ni Marseille ni Albi)
# ================================================================================
#
# USAGE :
#   - Accès aux serveurs du mesh uniquement :
#       http://10.0.0.1:9000  → Portainer UI
#       ssh user@10.0.0.1 / .2 / .3  → SSH sur chaque serveur
#   - Internet du client : passe par sa connexion normale (pas d'exit node)
#   - 192.168.1.x / 192.168.2.x : INACCESSIBLES (UFW DROP côté serveurs)
#
# --------------------------------------------------------------------------------
# CHOIX DU POINT D'ENTRÉE
# --------------------------------------------------------------------------------
# Ce fichier ci-dessous est configuré pour entrer par OVH (cas par défaut).
# Pour entrer par un AUTRE serveur, 3 valeurs changent : Address, PublicKey
# du peer, et Endpoint. AllowedIPs reste TOUJOURS 10.0.0.0/24.
#
# IMPORTANT : chaque point d'entrée utilise une IP mesh DIFFÉRENTE.
# C'est obligatoire pour le routage WireGuard (une IP ne peut être routée
# que par un seul peer par serveur). Générez une paire de clés par variante,
# ou réutilisez la même clé en changeant seulement l'Address selon l'entrée.
#
#   ┌──────────────┬──────────────┬──────────────────────────────┬──────────────────────────┐
#   │ Entrée par   │ Address      │ [Peer] PublicKey             │ [Peer] Endpoint          │
#   ├──────────────┼──────────────┼──────────────────────────────┼──────────────────────────┤
#   │ OVH          │ 10.0.0.40/32 │ <CLE_PUBLIQUE_OVH>           │ <IP_PUBLIQUE_OVH>:51820  │
#   │ Marseille    │ 10.0.0.41/32 │ <CLE_PUBLIQUE_MRS>           │ <IP_PUBLIQUE_MRS>:51820  │
#   │ Albi         │ 10.0.0.42/32 │ <CLE_PUBLIQUE_ALBI>          │ <IP_PUBLIQUE_ALBI>:51820 │
#   └──────────────┴──────────────┴──────────────────────────────┴──────────────────────────┘
```

```
#
# Rappel côté serveur : la clé publique du client doit être déclarée dans
# le [Peer] correspondant du serveur d'entrée (10.0.0.40 sur OVH,
# 10.0.0.41 sur MRS, 10.0.0.42 sur Albi). C'est déjà prévu dans les
# fichiers serveurs v3.
# ================================================================================

[Interface]
# ---- Entrée OVH (défaut) : 10.0.0.40/32 ----
# ---- Entrée MRS          : 10.0.0.41/32 ----
# ---- Entrée Albi         : 10.0.0.42/32 ----
Address    = 10.0.0.40/32
PrivateKey = <CLE_PRIVEE_CLIENT_PRIVATE>
# Pas de redirection DNS — on garde le DNS habituel du client.
# DNS = 10.0.0.1   ← décommenter si un DNS est déployé dans le mesh

[Peer]
# ---- Entrée OVH (défaut) ----
PublicKey = <CLE_PUBLIQUE_OVH>
Endpoint  = <IP_PUBLIQUE_OVH>:51820
#
# ---- Entrée MRS : remplacer les 2 lignes ci-dessus par ----
# PublicKey = <CLE_PUBLIQUE_MRS>
# Endpoint  = <IP_PUBLIQUE_MRS>:51820
#
# ---- Entrée Albi : remplacer les 2 lignes ci-dessus par ----
# PublicKey = <CLE_PUBLIQUE_ALBI>
# Endpoint  = <IP_PUBLIQUE_ALBI>:51820

# AllowedIPs : identique quel que soit le point d'entrée.
# Seul le mesh, aucun LAN, aucun exit node.
AllowedIPs          = 10.0.0.0/24
PersistentKeepalive = 25
```

Client Exit Node OVH

```
# ================================================================================
# Profil client — Exit node OVH
# Fichier : ovh-exitnode.conf   |   IP mesh : 10.0.0.10/32
# IP publique vue : VPS OVH     |   Accès LAN : aucun
```

```
# ================================================================================
#
# USAGE :
#   - Tout le trafic internet sort par OVH (IP publique = OVH)
#   - Accès au mesh (Portainer UI sur http://10.0.0.1:9000)
#   - 192.168.1.x et 192.168.2.x INACCESSIBLES (non routés + UFW DROP)
#
# ================================================================================

[Interface]
Address    = 10.0.0.10/32
PrivateKey = <CLE_PRIVEE_CLIENT_OVH>
DNS        = 1.1.1.1, 8.8.8.8

[Peer]
PublicKey           = <CLE_PUBLIQUE_OVH>
Endpoint            = <IP_PUBLIQUE_OVH>:51820
AllowedIPs          = 0.0.0.0/0, ::/0
PersistentKeepalive = 25
```

Client Exit Node Mrs + LAN

```
# ================================================================================
# Profil client — Exit node Marseille
# Fichier : mrs-exitnode.conf   |   IP mesh : 10.0.0.20/32
# IP publique vue : box Marseille  |  Accès LAN : 192.168.1.0/24 (LAN MRS)
# ================================================================================
#
# USAGE :
#   - Tout le trafic internet sort par Marseille (IP = box MRS)
#   - Accès complet au LAN de Marseille (NAS, caméras, IoT…)
#   - Accès au mesh (Portainer UI sur http://10.0.0.1:9000)
#   - 192.168.2.x (LAN Albi) INACCESSIBLE
#
# CONNEXION : directe vers Marseille (pas de transit par OVH).
#
# ================================================================================

[Interface]
Address    = 10.0.0.20/32
```

```
PrivateKey = <CLE_PRIVEE_CLIENT_MRS>
# DNS local de Marseille en priorité (résolution des noms du LAN), public en secours
DNS        = 192.168.1.53, 1.1.1.1

[Peer]
PublicKey           = <CLE_PUBLIQUE_MRS>
Endpoint            = <IP_PUBLIQUE_MRS>:51820
# 0.0.0.0/0 : tout passe par Marseille. Le LAN 192.168.1.0/24 est inclus
# dedans et son accès est autorisé côté serveur par UFW (uniquement pour
# cette IP 10.0.0.20). L'accès au LAN d'Albi reste bloqué (pas de route MRS
# vers 192.168.2.0/24 + UFW DROP côté Albi).
AllowedIPs          = 0.0.0.0/0, ::/0
PersistentKeepalive = 25
```

Client Exit Node Albi + LAN

```
# ================================================================================
# Profil client — Exit node Albi
# Fichier : albi-exitnode.conf   |   IP mesh : 10.0.0.30/32
# IP publique vue : box Albi      |  Accès LAN : 192.168.2.0/24 (LAN Albi)
# ================================================================================
#
# USAGE :
#   - Tout le trafic internet sort par Albi (IP = box Albi)
#   - Accès complet au LAN d'Albi (NAS, caméras Frigate, IoT…)
#   - Accès au mesh (Portainer UI sur http://10.0.0.1:9000)
#   - 192.168.1.x (LAN Marseille) INACCESSIBLE
#
# CONNEXION : directe vers Albi (pas de transit par OVH).
#
# ================================================================================

[Interface]
Address    = 10.0.0.30/32
PrivateKey = <CLE_PRIVEE_CLIENT_ALBI>
DNS        = 192.168.2.53, 1.1.1.1

[Peer]
PublicKey           = <CLE_PUBLIQUE_ALBI>
Endpoint            = <IP_PUBLIQUE_ALBI>:51820
```

```
AllowedIPs          = 0.0.0.0/0, ::/0
PersistentKeepalive = 25
```

Configuration Pare-feu

Par-feu OVH

```
#!/usr/bin/env bash
# ================================================================================
# Configuration UFW — VPS OVH
# Interface physique : eth0   |   IP WireGuard : 10.0.0.1
# ================================================================================
#
# À exécuter en root APRÈS avoir démarré le conteneur WireGuard (wg0 doit exister).
#   sudo bash ovh-ufw-setup.sh
#
# Ce script :
#   1. Active le forwarding IP au niveau hôte
#   2. Ajoute le masquerading (NAT) pour les clients exit-node OVH
#   3. Pose une politique FORWARD = DROP et n'autorise que le trafic VPN voulu
#   4. Ouvre les ports nécessaires (tunnel, SSH, Portainer UI sur le mesh)
#
# IMPORTANT : ne casse PAS Docker. UFW pose FORWARD à DROP, ce que Docker
# fait déjà de son côté ; Docker gère ses propres règles ACCEPT via
# DOCKER-USER. La communication entre conteneurs reste fonctionnelle.
# ================================================================================
set -euo pipefail

IFACE_PHYS="eth0"
WG_IFACE="wg0"
MESH="10.0.0.0/24"

echo "[1/5] Activation du forwarding IP (hôte)"
# net/ipv4 syntaxe attendue par /etc/ufw/sysctl.conf
grep -q "net/ipv4/ip_forward=1" /etc/ufw/sysctl.conf || \
  echo "net/ipv4/ip_forward=1" >> /etc/ufw/sysctl.conf
grep -q "net/ipv6/conf/all/forwarding=1" /etc/ufw/sysctl.conf || \
  echo "net/ipv6/conf/all/forwarding=1" >> /etc/ufw/sysctl.conf
# src_valid_mark requis par wg-quick (routage marqué)
```

```
echo "net.ipv4.conf.all.src_valid_mark=1" > /etc/sysctl.d/99-wireguard.conf
sysctl -p /etc/sysctl.d/99-wireguard.conf >/dev/null

echo "[2/5] Politique FORWARD par défaut = DROP"
sed -i 's/^DEFAULT_FORWARD_POLICY=.*/DEFAULT_FORWARD_POLICY="DROP"/' /etc/default/ufw

echo "[3/5] Masquerading NAT dans before.rules"
# On insère un bloc *nat en tête de /etc/ufw/before.rules s'il n'y est pas déjà.
if ! grep -q "WG-MASQUERADE-OVH" /etc/ufw/before.rules; then
  cp /etc/ufw/before.rules /etc/ufw/before.rules.bak.$(date +%s)
  cat > /tmp/wg-nat.rules <<EOF
# WG-MASQUERADE-OVH (ne pas supprimer cette ligne, sert de marqueur)
*nat
:POSTROUTING ACCEPT [0:0]
# Le trafic des clients VPN qui sort par eth0 prend l'IP publique d'OVH
-A POSTROUTING -s ${MESH} -o ${IFACE_PHYS} -j MASQUERADE
COMMIT

EOF
  # Préfixe le fichier existant avec le bloc nat
  cat /tmp/wg-nat.rules /etc/ufw/before.rules > /tmp/before.rules.new
  mv /tmp/before.rules.new /etc/ufw/before.rules
  rm -f /tmp/wg-nat.rules
fi

echo "[4/5] Ports d'entrée (INPUT)"
ufw allow OpenSSH                                    # ne pas se verrouiller dehors
ufw allow 51820/udp comment 'WireGuard tunnel'
# Portainer UI : accessible UNIQUEMENT via le mesh (interface wg0)
ufw allow in on ${WG_IFACE} to any port 9000 proto tcp comment 'Portainer UI (mesh)'
# SSH via le mesh (administration interne)
ufw allow in on ${WG_IFACE} to any port 22 proto tcp comment 'SSH mesh'

echo "[5/5] Règles de routage VPN (FORWARD)"
# --- Client ovh-exitnode (10.0.0.10) ---
# Sortie internet par OVH
ufw route allow in on ${WG_IFACE} out on ${IFACE_PHYS} from 10.0.0.10 comment 'ovh-exitnode
internet'
# Accès au reste du mesh (relais vers MRS/Albi)
ufw route allow in on ${WG_IFACE} out on ${WG_IFACE} from 10.0.0.10 to ${MESH} comment 'ovh-
```

```
exitnode mesh'
# --- Client private-only entré via OVH (10.0.0.40) ---
# Mesh uniquement, AUCUNE sortie eth0 (pas d'internet, pas de LAN)
ufw route allow in on ${WG_IFACE} out on ${WG_IFACE} from 10.0.0.40 to ${MESH} comment
'private-only-ovh mesh'

echo "Activation/rechargement d'UFW"
ufw --force enable
ufw reload

echo
echo "=== Terminé. Vérifs utiles : ==="
echo "  ufw status verbose"
echo "  ufw route status"
echo "  iptables -t nat -L POSTROUTING -n | grep MASQUERADE"
echo
echo "Si les conteneurs Docker perdent le réseau, redémarrez Docker UNE fois"
echo "pour qu'il réinsère ses règles au-dessus de celles d'UFW :"
echo "  systemctl restart docker"
```

Par-feu Marseille

```
#!/usr/bin/env bash
# ============================================================================
# Configuration UFW — Serveur Marseille
# Interface physique : enp4s0   |   IP WireGuard : 10.0.0.2
# LAN local : 192.168.1.0/24
# ============================================================================
#
#   sudo bash marseille-ufw-setup.sh
#
# Accès LAN Marseille (192.168.1.0/24) autorisé UNIQUEMENT au client
# mrs-exitnode (10.0.0.20). Tout autre peer est bloqué par le FORWARD DROP.
# Aucune route statique sur la box n'est nécessaire (masquerading sur enp4s0).
# ============================================================================
set -euo pipefail

IFACE_PHYS="enp4s0"
WG_IFACE="wg0"
MESH="10.0.0.0/24"
```

```
LAN="192.168.1.0/24"

echo "[1/5] Forwarding IP (hôte)"
grep -q "net/ipv4/ip_forward=1" /etc/ufw/sysctl.conf || \
  echo "net/ipv4/ip_forward=1" >> /etc/ufw/sysctl.conf
grep -q "net/ipv6/conf/all/forwarding=1" /etc/ufw/sysctl.conf || \
  echo "net/ipv6/conf/all/forwarding=1" >> /etc/ufw/sysctl.conf
echo "net.ipv4.conf.all.src_valid_mark=1" > /etc/sysctl.d/99-wireguard.conf
sysctl -p /etc/sysctl.d/99-wireguard.conf >/dev/null

echo "[2/5] Politique FORWARD = DROP"
sed -i 's/^DEFAULT_FORWARD_POLICY=.*/DEFAULT_FORWARD_POLICY="DROP"/' /etc/default/ufw

echo "[3/5] Masquerading NAT (enp4s0)"
if ! grep -q "WG-MASQUERADE-MRS" /etc/ufw/before.rules; then
  cp /etc/ufw/before.rules /etc/ufw/before.rules.bak.$(date +%s)
  cat > /tmp/wg-nat.rules <<EOF
# WG-MASQUERADE-MRS (marqueur — ne pas supprimer)
*nat
:POSTROUTING ACCEPT [0:0]
# Couvre l'exit node (internet) ET l'accès au LAN : dans les deux cas le
# trafic sort par enp4s0 et prend l'IP locale du serveur (ex 192.168.1.100),
# ce qui évite toute route statique sur la box.
-A POSTROUTING -s ${MESH} -o ${IFACE_PHYS} -j MASQUERADE
COMMIT

EOF
  cat /tmp/wg-nat.rules /etc/ufw/before.rules > /tmp/before.rules.new
  mv /tmp/before.rules.new /etc/ufw/before.rules
  rm -f /tmp/wg-nat.rules
fi

echo "[4/5] Ports d'entrée (INPUT)"
ufw allow OpenSSH
ufw allow 51820/udp comment 'WireGuard tunnel'
# Portainer Agent : joignable uniquement par Portainer UI via le mesh
ufw allow in on ${WG_IFACE} to any port 9001 proto tcp comment 'Portainer Agent (mesh)'
# SSH via mesh — sert aussi aux sauvegardes rsync depuis Albi
ufw allow in on ${WG_IFACE} to any port 22 proto tcp comment 'SSH mesh / backups'
```

```
echo "[5/5] Règles de routage VPN (FORWARD)"
# --- Client mrs-exitnode (10.0.0.20) : internet + LAN Marseille ---
ufw route allow in on ${WG_IFACE} out on ${IFACE_PHYS} from 10.0.0.20 comment 'mrs-exitnode internet+LAN'
ufw route allow in on ${WG_IFACE} out on ${WG_IFACE} from 10.0.0.20 to ${MESH} comment 'mrs-exitnode mesh'
# --- Client private-only entré via MRS (10.0.0.41) : mesh uniquement ---
# Pas de règle "out on enp4s0" => ni internet ni LAN pour ce client.
ufw route allow in on ${WG_IFACE} out on ${WG_IFACE} from 10.0.0.41 to ${MESH} comment 'private-only-mrs mesh'

echo "Activation/rechargement d'UFW"
ufw --force enable
ufw reload

echo
echo "=== Terminé. Vérifs : ufw status verbose ; ufw route status ==="
echo "Test attendu :"
echo "  - mrs-exitnode atteint 192.168.1.x : OUI"
echo "  - private-only / autres profils atteignent 192.168.1.x : NON"
echo
echo "Si Docker perd le réseau : systemctl restart docker (une fois)"
```

Par-feu Albi

```
#!/usr/bin/env bash
# ==============================================================================
# Configuration UFW — Serveur Albi
# Interface physique : enp2s0   |   IP WireGuard : 10.0.0.3
# LAN local : 192.168.2.0/24
# ==============================================================================
#
#   sudo bash albi-ufw-setup.sh
#
# Accès LAN Albi (192.168.2.0/24) autorisé UNIQUEMENT au client
# albi-exitnode (10.0.0.30).
#
# COHABITATION DOCKER : Home Assistant et Frigate ne sont pas affectés.
# Leur trafic est local (OUTPUT/INPUT) ou géré par Docker, pas par la
# chaîne FORWARD qu'on filtre ici. Voir la note Docker en fin de script.
```

```
# ============================================================================
set -euo pipefail

IFACE_PHYS="enp2s0"
WG_IFACE="wg0"
MESH="10.0.0.0/24"
LAN="192.168.2.0/24"

echo "[1/5] Forwarding IP (hôte)"
grep -q "net/ipv4/ip_forward=1" /etc/ufw/sysctl.conf || \
  echo "net/ipv4/ip_forward=1" >> /etc/ufw/sysctl.conf
grep -q "net/ipv6/conf/all/forwarding=1" /etc/ufw/sysctl.conf || \
  echo "net/ipv6/conf/all/forwarding=1" >> /etc/ufw/sysctl.conf
echo "net.ipv4.conf.all.src_valid_mark=1" > /etc/sysctl.d/99-wireguard.conf
sysctl -p /etc/sysctl.d/99-wireguard.conf >/dev/null

echo "[2/5] Politique FORWARD = DROP"
sed -i 's/^DEFAULT_FORWARD_POLICY=.*/DEFAULT_FORWARD_POLICY="DROP"/' /etc/default/ufw

echo "[3/5] Masquerading NAT (enp2s0)"
if ! grep -q "WG-MASQUERADE-ALBI" /etc/ufw/before.rules; then
  cp /etc/ufw/before.rules /etc/ufw/before.rules.bak.$(date +%s)
  cat > /tmp/wg-nat.rules <<EOF
# WG-MASQUERADE-ALBI (marqueur — ne pas supprimer)
*nat
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -s ${MESH} -o ${IFACE_PHYS} -j MASQUERADE
COMMIT

EOF
  cat /tmp/wg-nat.rules /etc/ufw/before.rules > /tmp/before.rules.new
  mv /tmp/before.rules.new /etc/ufw/before.rules
  rm -f /tmp/wg-nat.rules
fi

echo "[4/5] Ports d'entrée (INPUT)"
ufw allow OpenSSH
ufw allow 51820/udp comment 'WireGuard tunnel'
ufw allow in on ${WG_IFACE} to any port 9001 proto tcp comment 'Portainer Agent (mesh)'
ufw allow in on ${WG_IFACE} to any port 22 proto tcp comment 'SSH mesh / backups'
```

```
echo "[5/5] Règles de routage VPN (FORWARD)"
# --- Client albi-exitnode (10.0.0.30) : internet + LAN Albi ---
ufw route allow in on ${WG_IFACE} out on ${IFACE_PHYS} from 10.0.0.30 comment 'albi-exitnode internet+LAN'
ufw route allow in on ${WG_IFACE} out on ${WG_IFACE} from 10.0.0.30 to ${MESH} comment 'albi-exitnode mesh'
# --- Client private-only entré via Albi (10.0.0.42) : mesh uniquement ---
ufw route allow in on ${WG_IFACE} out on ${WG_IFACE} from 10.0.0.42 to ${MESH} comment 'private-only-albi mesh'

echo "Activation/rechargement d'UFW"
ufw --force enable
ufw reload

echo
echo "=== Terminé. ==="
echo "NOTE DOCKER (Home Assistant / Frigate) :"
echo "  Si après activation d'UFW les conteneurs perdent l'accès réseau,"
echo "  redémarrez Docker UNE fois pour qu'il réinsère ses règles FORWARD"
echo "  au-dessus de celles d'UFW :"
echo "    systemctl restart docker"
echo "  (cela redémarre Home Assistant et Frigate — à faire à un moment calme)"
```