

Documentation GitLab + WireGuard VPN



Table des matières

Vue d'ensemble	2
Objectifs du projet.....	2
GitLab.....	2
l'architecture.....	4
Matrice de sécurité.....	6
Prérequis système	6
Déploiement GitLab CE	8
Lancement du conteneur GitLab	9
Installation et configuration WireGuard.....	12
Installation du paquet WireGuard	12
Génération des clés serveur	12
Configuration de l'interface WireGuard.....	12
Activation du service WireGuard	13
Gestion des clients VPN	14

Vue d'ensemble

Objectifs du projet

Ce document présente une architecture DevOps sécurisée et professionnelle. Elle repose sur un bastion de développement, un serveur GitLab et un VPN WireGuard.

L'objectif est de séparer clairement les rôles. L'exposition des services est volontairement limitée. Les flux de code sont strictement maîtrisés.

GitLab

GitLab est utilisé exclusivement pour :

- la gestion des dépôts Git
- L'authentification des utilisateurs
- l'exécution des pipelines CI/CD



Caractéristiques :

- Point d'accès unique et centralisé
- Authentification multi-niveaux (VPN + GitLab)
- Isolation réseau stricte
- Journalisation exhaustive
- Principe du moindre privilège

1.1 Stack technique

Composant	Version	Rôle
Debian	11/12	Système d'exploitation hôte
Docker Engine	24.x+	Containerisation
GitLab CE	latest	Bastion applicatif (SCM + CI/CD)
WireGuard	1.x	Tunnel VPN sécurisé
UFW	0.36+	Firewall applicatif
GitLab Runner	latest	Exécution des pipelines

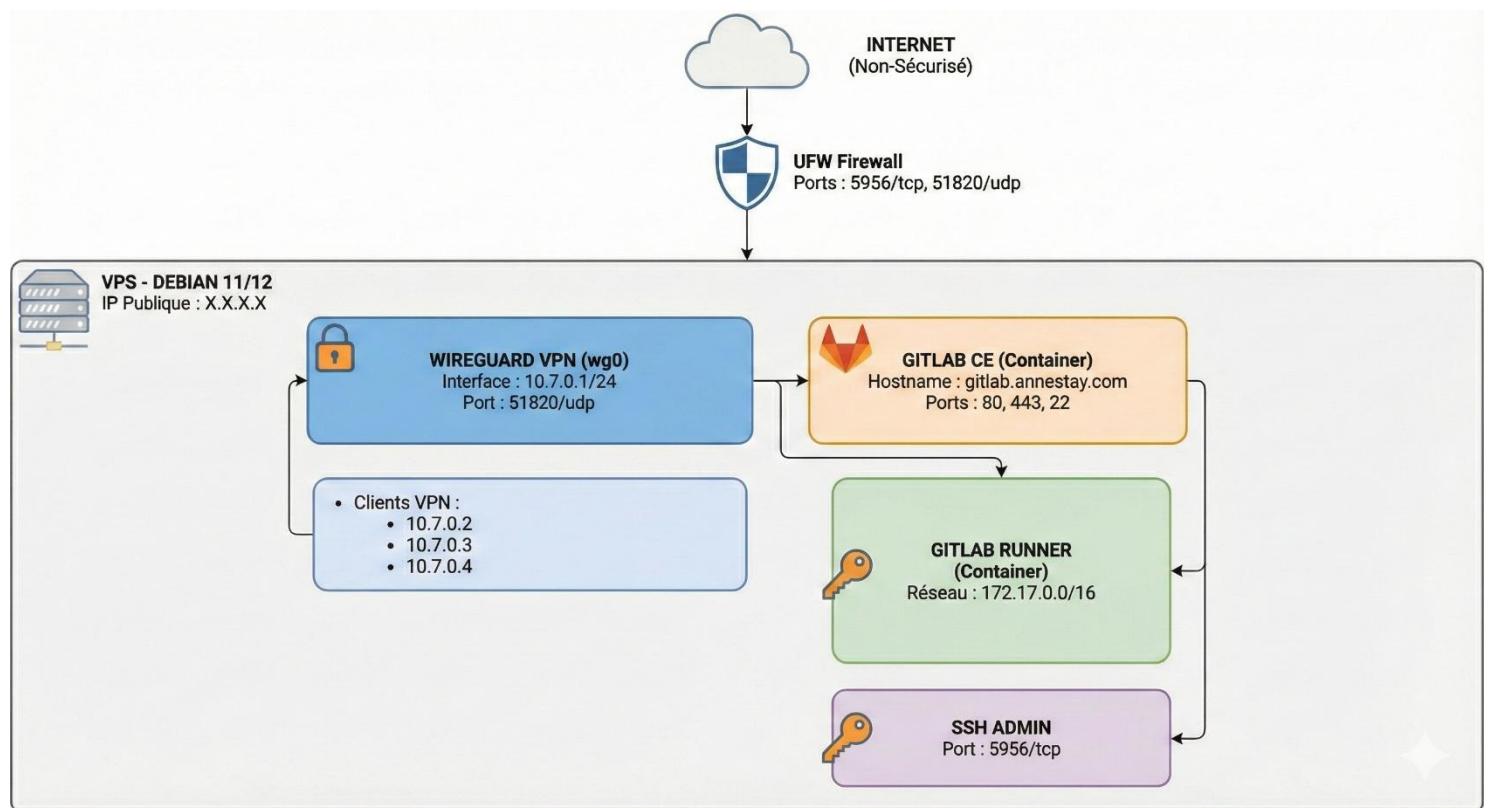
l'architecture

L'utilisateur ne se connecte jamais directement aux services internes.

L'accès se fait uniquement après établissement du tunnel VPN WireGuard.

Le bastion constitue la zone de travail unique pour le développement.

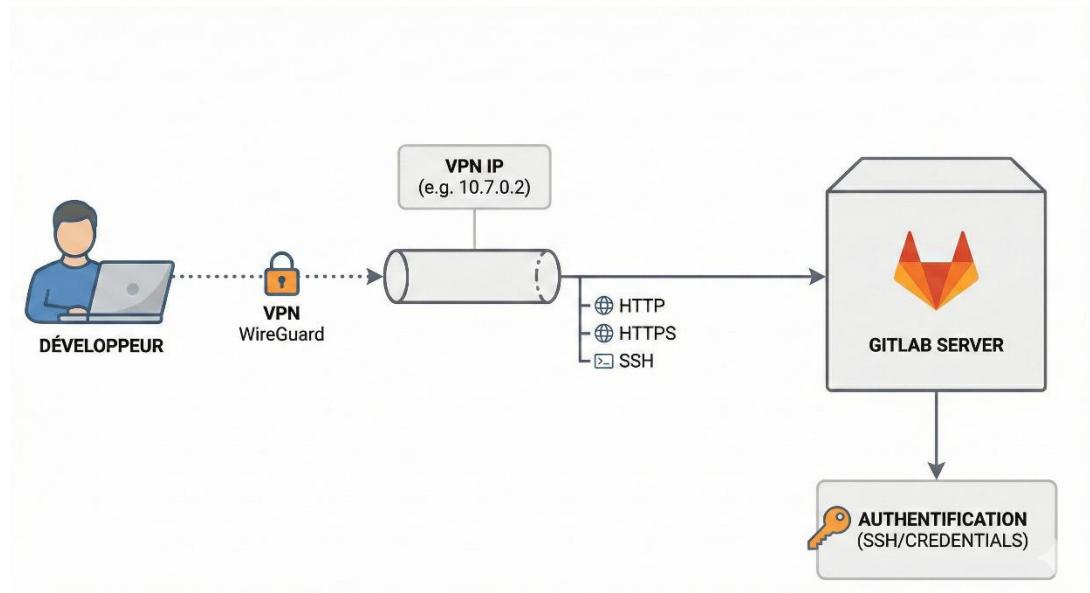
GitLab agit comme dépôt central et plateforme CI/CD.



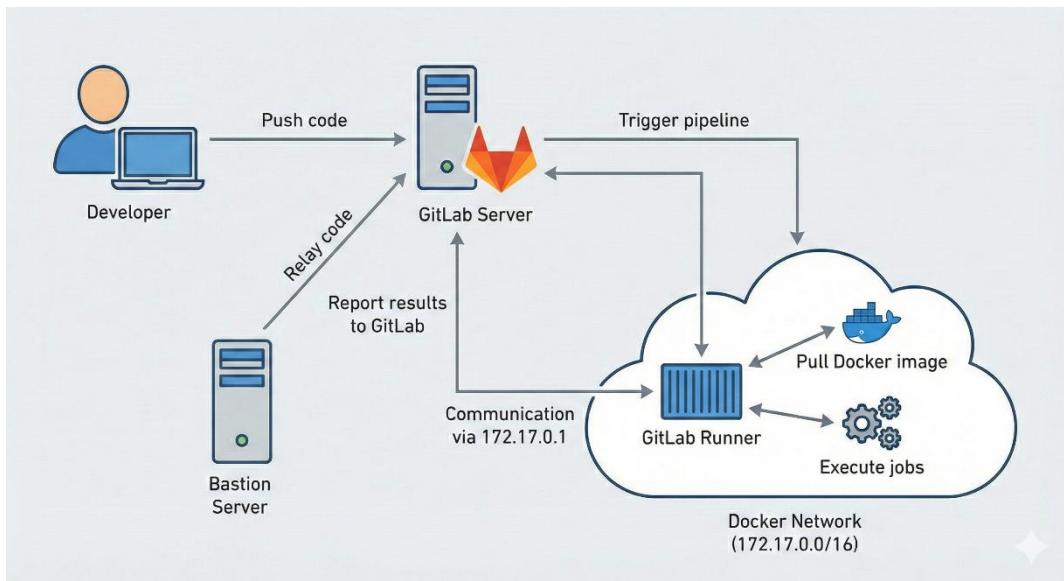
Le flux autorisé est le suivant : poste développeur, VPN, bastion, GitLab.

Schéma d'architecture

Accès développeur au Gilab



Exécution pipeline CI/CD



Matrice de sécurité

Source	Destination	Port/Proto	Action	Justification
Internet	VPS	51820/udp	ALLOW	Tunnel VPN WireGuard
Internet	VPS	80/tcp	DENY	GitLab HTTP (VPN uniquement)
Internet	VPS	443/tcp	DENY	GitLab HTTPS (VPN uniquement)
10.7.0.0/24	VPS	80/tcp	ALLOW	Accès GitLab via VPN
10.7.0.0/24	VPS	443/tcp	ALLOW	Accès GitLab via VPN
172.17.0.0/16	VPS	80/tcp	ALLOW	Runner → GitLab (interne)
172.17.0.0/16	VPS	443/tcp	ALLOW	Runner → GitLab (interne)

Prérequis système

Serveur VPS requis :

CPU	8 GB minimum
Stockage	50 GB recommandé
OS	Debian 13
Accès root SSH	IP publique statique
Stockage	50 GB recommandé

- Pour les développeurs : Client WireGuard installé (Windows/Linux/MacOS) Client Git 2.x
- Client SSH (OpenSSH) Navigateur web moderne

```

bash

# Vérifier la version Debian
cat /etc/debian_version

# Vérifier les ressources système
free -h
df -h
nproc

# Mettre à jour le système
apt update && apt upgrade -y

# Vérifier la connectivité Internet
ping -c 4 8.8.8.8

# Vérifier les ports disponibles
ss -tuln | grep -E '80|443|22|51820'

```

Checklist pré-installation

- Installation et configuration
- Installation Docker Engine
- Ajout du dépôt officiel Docker

```

bash

# Installation des dépendances
apt-get update
apt-get install -y ca-certificates curl gnupg

# Ajout de la clé GPG Docker
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | \
gpg --dearmor -o /etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg

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

```

Installation des paquets Docker

```
bash

apt-get update
apt-get install -y \
docker-ce \
docker-ce-cli \
containerd.io \
docker-buildx-plugin \
docker-compose-plugin
```

Validation de l'installation

```
bash

# Vérifier le service Docker
systemctl status docker
systemctl enable docker

# Tester le fonctionnement
docker run --rm hello-world

# Vérifier la version
docker --version
docker compose version
```

Output attendu :

```
Docker version 24.0.x, build xxxxx
Docker Compose version v2.x.x
```

Déploiement GitLab CE

Création de l'arborescence persistante

```
# Structure de données GitLab
mkdir -p /srv/gitlab/{config,logs,data}

# Permissions appropriées
chmod 755 /srv/gitlab
chmod 755 /srv/gitlab/{config,logs,data}

# Vérification
tree -L 2 /srv/gitlab
```

Structure obtenue :

```
/srv/gitlab/
├── config/ → Configuration GitLab (gitlab.rb)
├── logs/ → Logs applicatifs
└── data/ → Données Git + PostgreSQL + Redis
```

Lancement du conteneur GitLab

```
bash

docker run --detach \
--hostname gitlab.annestay.com \
--publish 443:443 \
--publish 80:80 \
--publish 22:22 \
--name gitlab \
--restart always \
--volume /srv/gitlab/config:/etc/gitlab \
--volume /srv/gitlab/logs:/var/log/gitlab \
--volume /srv/gitlab/data:/var/opt/gitlab \
--shm-size 256m \
gitlab/gitlab-ce:latest
```

Paramètres expliqués :

- FQDN du bastion GitLab --hostname
 - Mapping des ports (hôte:conteneur) --publish
 - Redémarrage automatique après reboot --restart always
 - Persistance des données hors conteneur --volume
 - Mémoire partagée pour Prometheus --shm-size

```
bash

# Suivre les logs en temps réel
docker logs -f gitlab

# Attendre le message de fin de démarrage
# "gitlab Reconfigured!"
```

Récupération du mot de passe root

```
bash

# Extraction du mot de passe initial
docker exec -it gitlab grep 'Password:' /etc/gitlab/initial_root_password

# Output:
# Password: 5iveL!fe1sH3110nnnnnnnnnnn

# ⚠️ IMPORTANT : Changer ce mot de passe immédiatement après la première connexion
```

Première connexion :

Ouvrir <http://10.7.0.1> (via VPN)

Login :

Password : (récupéré ci-dessus)

Aller dans Admin Area > Settings > General > Sign-in restrictions

Changer le mot de passe root

bash

Éditer la configuration

nano /srv/gitlab/config/gitlab.rb

Exemples de paramètres à ajuster :

external_url 'http://gitlab.annestay.com'

gitlab_rails['gitlab_shell_ssh_port']=22

gitlab_rails['time_zone'] = 'Europe/Paris'

gitlab_rails['backup_keep_time'] = 604800 # 7 jours

Appliquer les modifications

docker exec -it gitlab gitlab-ctl reconfigure

Installation et configuration WireGuard

Installation du paquet WireGuard

```
bash  
  
apt-get update  
apt-get install -y wireguard wireguard-tools
```

Génération des clés serveur

```
bash  
  
cd /etc/wireguard  
umask 077  
  
# Génération de la paire de clés  
wg genkey | tee privatekey | wg pubkey > publickey  
  
# Vérification  
ls -la /etc/wireguard/  
# -rw----- 1 root root 45 privatekey  
# -rw----- 1 root root 45 publickey
```

/etc/wireguard/wg0.conf

Configuration de l'interface WireGuard

```
bash  
  
# Création du fichier de configuration  
nano /etc/wireguard/wg0.conf
```

```

[Interface]
# Clé privée du serveur (NE JAMAIS PARTAGER)
PrivateKey = <CONTENUE_TWIGUARDKEY>

# Adresse IP du serveur VPN
Address = 107.0.1.24

# Port d'écoute UDP
ListenPort = 51820

# Règles iptables pour le routage (si NAT nécessaire)
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE

# Note: Remplacer eth0 par le nom de votre interface publique si différent
# Vérifier avec: ip link show

```

Activation du service WireGuard

```

bash

# Activer au démarrage
systemctl enable wg-quick@wg0

# Démarrer le service
systemctl start wg-quick@wg0

# Vérifier le statut
systemctl status wg-quick@wg0

# Vérifier l'interface
wg show
ip addr show wg0

```

Output attendu :

```

interface: wg0
public key:<CLÉ_PUBLIQUE_SERVEUR>
private key: (hidden)
listening port: 51820

```

```
# Activer le forwarding
echo "net.ipv4.ip_forward=1">>>/etc/sysctl.conf
sysctl -p

# Vérification
cat /proc/sys/net/ipv4/ip_forward
# Output attendu : 1
```

Gestion des clients VPN

Processus de provisioning d'un client

Étape 1 : Génération des clés client (sur le serveur)

```
bash

cd /etc/wireguard/clients

# Créer un répertoire pour chaque client
mkdir -p client_dev1

# Générer les clés
wg genkey | tee client_dev1/privatekey | wg pubkey > client_dev1/publickey

# Sauvegarder les clés de manière sécurisée
chmod 600 client_dev1/*
```

Étape 2 : Déclaration du peer sur le serveur

```
bash

# Ajouter le peer au fichier de configuration
nano /etc/wireguard/wg0.conf
```

Ajouter à la fin du fichier :

```
ini
```

[Peer]

```
# Client : Dev 1 - John Doe
```

```
P u b l i c K e y =<CONTENU_DE_client-devel/publickey>
```

```
A ll o w e d I P s = 10.7.0.2/32
```

```
bash
```

```
# Méthode 1 : Rechargement à chaud
```

```
wg addconfwg0 <(wg-quick strip wg0)
```

```
# Méthode 2 : Redémarrage du service (coupure brève)
```

```
systemctl restart wg-quick@wg0
```

```
# Vérification
```

```
wg show
```

Étape 4 : Génération du fichier de configuration client

```
bash
```

```
nano /etc/wireguard/clients/client_devl/client_devl.conf
```

Contenu du fichier client :

Étape 5 : Transmission sécurisée au client

```
bash
```

```
# Créeer une archive chiffrée (recommandé)
cd /etc/wireguard/clients
tar -czf client_devl.tar.gz client_devl/
gpg --symmetric --cipher-algo AES256 client_devl.tar.gz
```

Transmettre via canal sécurisé :

- Fichier .conf via email chiffré
- QR Code (pour mobile) : qrencode -t ansiutf8 < client_devl.conf
- Gestionnaire de secrets (Vault, 1Password, etc.)

Tableau de suivi des clients

ID	Nom	IP VPN	Date création	Statut	Notes
1	Dev 1 (John Doe)	10.7.0.2	2025-12-26	Actif	Développeur frontend
2	Dev 2 (Jane Smith)	10.7.0.3	2025-12-26	Actif	Développeur backend
3	CI/CD External	10.7.0.10	2025-12-26	Actif	Pipeline externe

Déploiement GitLab Runner Lancement du conteneur Runner

bash

```
# Créez le répertoire de configuration
mkdir -p /srv/gitlab-runner/config

# Lancez le conteneur
docker run -d \
--name gitlab-runner \
--restart always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /srv/gitlab-runner/config:/etc/gitlab-runner \
gitlab/gitlab-runner:latest
```

Récupération du token d'enregistrement Dans l'interface GitLab :

Se connecter en tant que **root**

Aller dans Admin Area > CI/CD > Runners

Section "Set up a shared runner manually"

Copier le registration token

Enregistrement du Runner

bash

```
docker exec -it gitlab-runner gitlab-runner register \
--non-interactive \
--url "http://10.7.0.1/" \
--registration-token "VOTRE_TOKEN_ICI" \
--executor "docker" \
--docker-image alpine:latest \
--description "Runner Docker Production" \
--maintenance-note "Runner principal bastion GitLab" \
--tag-list "docker,production,bastion" \
--run-untagged="true" \
--locked="false"
```

Vérification du Runner

bash

Vérifier l'enregistrement

```
docker exec -it gitlab-runner gitlab-runner verify
```

Lister les runners

```
docker exec -it gitlab-runner gitlab-runner list
```

Vérifier les logs

```
docker logs gitlab-runner
```

Dans GitLab :



Aller dans Admin Area > CI/CD > Runners

Le runner doit apparaître avec une pastille verte (actif)

Sauvegarde automatisée

```
bash
```

```
# Éditer la configuration GitLab  
nano /srv/gitlab/config/gitlab.rb
```

```
ruby
```

```
# Sauvegardes quotidiennes à 2h du matin  
gitlab_rails['backup_schedule'] = '0 2 * * *'  
  
# Rétention : 7 jours  
gitlab_rails['backup_keep_time'] = 604800  
  
# Emplacement des backups  
gitlab_rails['backup_path'] = '/var/opt/gitlab/backups'
```

Ajouter :

```
bash

# Appliquer la configuration
docker exec -it gitlab gitlab-ctl reconfigure

# Tester une sauvegarde manuelle
docker exec -it gitlab gitlab-backup create
```