Cloudflare Tunnel : Exposer un Serveur sans Ouvrir de Port

Cloudflare Tunnel expose vos serveurs locaux sur Internet via un daemon **cloudflared** qui ouvre une connexion **sortante** chiffrée vers les edge servers Cloudflare — sans jamais ouvrir un port entrant sur votre pare-feu. Fin des règles NAT, des IP publiques exposées et des DMZ complexes à maintenir.

Exposer un **serveur local** sur Internet sans ouvrir un seul port entrant, c'est exactement ce que permet le **Cloudflare Tunnel** avec le daemon `cloudflared`. La logique est inversée par rapport au port forwarding classique : plutôt que d'ouvrir votre réseau vers l'extérieur, c'est votre réseau qui "sort" vers les edge servers Cloudflare via une connexion **mTLS** persistante et chiffrée. Votre IP publique reste invisible aux visiteurs, la protection DDoS L3/L7 est appliquée automatiquement en amont, et votre pare-feu n'accepte littéralement aucune connexion entrante. En 2026, Cloudflare recense plus de 2 millions de tunnels actifs dans son infrastructure. Pour les homelabs, les équipes DevOps, les développeurs en télétravail et les PME qui cherchent à remplacer leur DMZ classique, cloudflare tunnel serveur représente une alternative sérieuse aux solutions traditionnelles : plus simple à déployer qu'un bastion, plus sécurisé qu'une règle NAT ouverte, et intégrable nativement avec Cloudflare Access pour du Zero Trust en quelques minutes. Ce guide couvre l'installation complète de cloudflared, la configuration YAML multi-services, l'accès SSH et RDP sécurisé, les cas d'usage réels (Gitea, Proxmox, applications internes) et les pièges classiques à éviter en production.

À RETENIR

À retenir

Zéro port ouvert : cloudflared crée une connexion *sortante* vers Cloudflare — votre pare-feu reste fermé en entrée.

IP serveur masquée : les visiteurs voient l'IP Cloudflare, jamais la vôtre ; protection DDoS incluse automatiquement.

Config YAML multi-services : un seul tunnel expose Gitea, Proxmox, Vaultwarden et une app Node simultanément.

Zero Trust natif : Cloudflare Access s'intègre directement pour restreindre l'accès par identité (Google, GitHub, email OTP).

SSH et RDP sécurisés : cloudflared permet l'accès SSH et RDP sans exposer ces ports sur Internet, via le client WARP ou le navigateur.

Comment fonctionne cloudflared techniquement ?

Cloudflare Tunnel repose sur le daemon **cloudflared**, un binaire Go open-source maintenu par Cloudflare. Quand vous lancez `cloudflared tunnel run`, le daemon établit **quatre connexions simultanées** vers les edge servers Cloudflare les plus proches — deux pour la redondance géographique, deux pour la redondance de chemin réseau. Ces connexions utilisent **QUIC** (HTTP/3) par défaut, comme documenté dans la documentation officielle Cloudflare Tunnel, avec fallback automatique sur HTTP/2 si QUIC est filtré par votre FAI ou votre réseau d'entreprise.

Chaque connexion est authentifiée par un certificat TLS client généré lors de la création du tunnel (`cert.pem` stocké dans `~/.cloudflared/`). Ce certificat prouve à Cloudflare que le daemon est bien autorisé à faire transiter le trafic pour votre domaine. Du côté visiteur, Cloudflare agit comme proxy inverse classique — terminaison TLS, injection de headers de sécurité, filtrage DDoS — puis route les requêtes vers votre tunnel sans que l'IP de votre serveur ne soit jamais révélée.

Un point souvent mal compris : contrairement à un VPN, **cloudflared ne crée pas de route réseau**. Il expose uniquement des services spécifiques via des règles d'ingress que vous définissez dans le fichier de config YAML. Vous pouvez exposer un port HTTP sans exposer le SSH du même serveur. C'est une distinction fondamentale en termes de surface d'attaque — vous contrôlez précisément ce qui est accessible, service par service.

Environnement testé : Debian 12 (serveur local), cloudflared 2024.9.1, domaine géré par Cloudflare, connexion FTTH 1 Gbps. Tunnel exposant Gitea 1.22, Vaultwarden et Proxmox VE 8.2. Latence ajoutée observée : 8–12 ms (PoP Cloudflare Paris).

Prérequis avant de créer votre premier tunnel

Avant de lancer la première commande, trois éléments sont indispensables. D'abord, un **domaine géré par Cloudflare** — pas seulement les nameservers, mais le domaine doit être dans votre dashboard Cloudflare avec le proxy activé (nuage orange). Ensuite, un compte Cloudflare avec accès à la section "Zero Trust". Enfin, un serveur Linux ou Windows avec une connexion Internet sortante sur les ports 443 et 7844 (QUIC).

Installation de cloudflared sur Debian/Ubuntu :

```
# Ajouter le dépôt officiel Cloudflare
curl -fsSL https://pkg.cloudflare.com/cloudflare-main.gpg | sudo gpg --dearmor   -o /usr/share/ke

echo 'deb [signed-by=/usr/share/keyrings/cloudflare-main.gpg]   https://pkg.cloudflare.com/cloudf

sudo apt-get update && sudo apt-get install cloudflared -y

# Vérifier la version installée
cloudflared --version
# cloudflared version 2024.9.1 (built 2024-09-10...)
```

Sur les distributions RPM (Rocky Linux 9, RHEL, AlmaLinux) :

```
curl -fsSL https://pkg.cloudflare.com/cloudflared-ascii.repo |   sudo tee /etc/yum.repos.d/cloudf
sudo yum install cloudflared -y
```

Créer, authentifier et router votre tunnel

L'authentification génère le certificat client qui autorise votre daemon à opérer. Cette commande ouvre un navigateur (ou affiche une URL si vous êtes en SSH) pour vous connecter à votre compte Cloudflare et sélectionner le domaine à utiliser :

```
# Étape 1 : Authentification — génère ~/.cloudflared/cert.pem
cloudflared tunnel login

# Étape 2 : Créer un tunnel avec un nom explicite
cloudflared tunnel create mon-serveur-maison
# Returned: Created tunnel mon-serveur-maison with id 2d4e5f6a-xxxx

# Étape 3 : Créer le CNAME DNS qui pointe vers le tunnel
cloudflared tunnel route dns mon-serveur-maison app.exemple.fr

# Vérification DNS — doit pointer vers UUID.cfargotunnel.com
dig app.exemple.fr CNAME +short
# 2d4e5f6a-xxxx.cfargotunnel.com.
```

Le record DNS est créé avec le proxy Cloudflare activé et un TTL court (5 minutes). Votre IP réelle n'apparaît nulle part dans ce record — impossible pour un attaquant de la déduire depuis le DNS.

Fichier de configuration YAML : exposer plusieurs services

La vraie puissance de Cloudflare Tunnel réside dans la configuration YAML multi-services. Un seul tunnel peut exposer une dizaine d'applications sur des sous-domaines différents, avec des paramètres par service :

```
 # ~/.cloudflared/config.yml
tunnel: 2d4e5f6a-xxxx-xxxx-xxxx-xxxxxxxxxxxx
credentials-file: /root/.cloudflared/2d4e5f6a-xxxx.json

# Métriques Prometheus (optionnel mais recommandé)
metrics: localhost:60123

ingress:
  # Gitea sur git.exemple.fr
  - hostname: git.exemple.fr
    service: http://localhost:3000

  # Proxmox VE — certificat auto-signé donc noTLSVerify
  - hostname: proxmox.exemple.fr
    service: https://localhost:8006
    originRequest:
      noTLSVerify: true

  # Vaultwarden — gestionnaire de mots de passe
  - hostname: vault.exemple.fr
    service: http://localhost:8080

  # App Node.js avec timeout personnalisé
  - hostname: app.exemple.fr
    service: http://localhost:3000
    originRequest:
      connectTimeout: 10s
      keepAliveTimeout: 90s

  # Règle catch-all OBLIGATOIRE — doit être en dernier
  - service: http_status:404
```

La règle `http_status:404` en fin de configuration est **obligatoire** — sans elle, cloudflared refuse de démarrer avec une erreur explicite. Pour lancer le tunnel en service systemd persistant :

```
 # Installer le service systemd automatiquement
sudo cloudflared service install

# Activer et démarrer
sudo systemctl enable cloudflared
sudo systemctl start cloudflared

# Vérifier le statut et les logs
sudo systemctl status cloudflared
journalctl -u cloudflared -f --no-pager
```

Accès SSH sécurisé via Cloudflare Tunnel

SSH est l'une des cibles favorites des scanners automatisés sur Internet. Exposer le port 22 directement, même avec fail2ban, c'est prendre un risque inutile. Cloudflare Tunnel permet d'accéder à SSH sans ouvrir le port, en routant la connexion à travers le tunnel. C'est l'une des fonctionnalités les plus sous-exploitées.

Côté serveur, ajoutez dans config.yml :

```
ingress:
  - hostname: ssh.exemple.fr
    service: ssh://localhost:22
  - service: http_status:404
```

Côté client (sur votre machine locale) :

```
 # Installer cloudflared sur le client également
# Ajouter dans ~/.ssh/config :
Host ssh.exemple.fr
  ProxyCommand cloudflared access ssh --hostname %h
  User votre_utilisateur
  IdentityFile ~/.ssh/id_ed25519

# Connexion SSH standard — cloudflared s'authentifie automatiquement via Cloudflare Access
ssh ssh.exemple.fr
```

Cloudflare Access s'intercale et vérifie l'identité avant d'autoriser la connexion TCP. **Aucun port 22 n'est exposé sur Internet.** Les logs d'accès SSH sont enregistrés dans le dashboard Cloudflare Zero Trust avec timestamp, email utilisateur et durée de session.

Cloudflare Tunnel vs reverse proxy classique : que choisir ?

La question revient systématiquement lors des choix d'architecture. La réponse dépend de votre contexte — il n'y a pas de solution universellement supérieure.

Critère	Cloudflare Tunnel	Nginx + port forwarding	WireGuard VPN
Ports ouverts entrants	Aucun	80, 443 minimum	UDP 51820
IP serveur exposée	Non (masquée)	Oui	Oui
Protection DDoS	Incluse L3/L7	Non sans CDN séparé	Non
SSH/RDP sans port ouvert	Oui (Access)	Non	Oui (accès réseau)
Dépendance externe	Cloudflare (forte)	Faible	Aucune
Latence ajoutée	5–30 ms	Négligeable	Variable
Coût	Gratuit / 7$/user (Zero Trust)	Gratuit (hors infra)	Gratuit

Mon biais personnel : pour un homelab ou une startup sans équipe réseau dédiée, **Cloudflare Tunnel gagne**. La sécurité par défaut est supérieure, la maintenance est quasi nulle, et la protection DDoS incluse vaut largement quelques millisecondes de latence. Pour une

infrastructure critique où vous refusez toute dépendance à un tiers, un bastion Nginx hardené reste pertinent.

Intégration Zero Trust avec Cloudflare Access

Exposer un service via cloudflare tunnel ne signifie pas que tout le monde y accède. **Cloudflare Access** s'intercale entre le visiteur et votre tunnel pour exiger une authentification préalable. La configuration se fait entièrement dans le dashboard Zero Trust, sans modifier le daemon cloudflared :

Créer une Application dans Zero Trust → Access → Applications → "Self-Hosted". Renseigner le domaine (`proxmox.exemple.fr`).

Définir une Policy : autoriser uniquement les emails `@votreentreprise.fr` , une liste d'emails nominatifs, un groupe GitHub/Google, ou exiger une authentification TOTP.

Choisir un Identity Provider : Google, GitHub, Microsoft Entra ID, Okta, ou OTP par email sans aucun compte tiers.

Le plan Zero Trust Free couvre jusqu'à 50 utilisateurs. Pour les homelabs et les petites équipes, c'est amplement suffisant. La philosophie Zero Trust appliquée ici s'appuie sur les recommandations du NIST SP 800-207 Zero Trust Architecture. Pour une architecture couvrant aussi RDP et SSH, notre guide Cloudflare Zero Trust complet détaille chaque module. Une fois Access configuré, votre Proxmox ou votre Gitea ne répond qu'après authentification — même si quelqu'un devine l'URL exacte. Pour aller plus loin dans l'implémentation Zero Trust, consultez également notre guide sur l'architecture Zero Trust et sa mise en œuvre réseau dans l'article Zero Trust Network 2026.

Astuce expert : Activez "Service Auth" pour les apps qui communiquent entre elles via tunnel sans navigateur. Cloudflare génère des tokens JWT de service avec TTL configurable — pas de secret partagé statique, rotation automatique.

Monitoring : métriques et logs structurés

Un tunnel silencieux qui plante à 3h du matin, ça arrive. Cloudflare propose plusieurs niveaux d'observabilité qui s'intègrent dans votre stack de monitoring existant :

```
 # Vérifier les métriques exposées par cloudflared
curl -s localhost:60123/metrics | grep -E "cloudflared_tunnel|tunnel_"

# Métriques clés à surveiller :
# cloudflared_tunnel_total_requests — trafic total
# cloudflared_tunnel_request_errors — erreurs 4xx/5xx origin
# cloudflared_tunnel_inactive_connections — connexions idle
```

Pour intégrer dans Prometheus/Grafana, ajoutez un scrape job sur `localhost:60123`. Les logs structurés JSON sont disponibles avec `loglevel: info` dans config.yml, lisibles directement par Loki ou Elasticsearch. Les reconnexions automatiques après coupure réseau sont loguées avec timestamp et raison — pratique pour auditer la stabilité du tunnel.

Pour une perspective défensive sur comment distinguer tunnels légitimes et malveillants dans vos logs réseau, notre article sur la détection du DNS tunneling pour les analystes SOC donne des patterns de détection concrets.

Sécuriser votre déploiement cloudflared en production

L'exposition via Cloudflare Tunnel réduit la surface d'attaque externe, mais plusieurs vecteurs côté serveur restent à adresser :

Fichier credentials JSON : le fichier `~/.cloudflared/UUID.json` donne accès total au tunnel. Permissions 600, propriétaire du compte système cloudflared uniquement, jamais versionné dans Git.

Compte système dédié : cloudflared doit tourner sous un compte sans shell, sans sudo, avec accès minimal au système de fichiers. `sudo cloudflared service install` crée ce compte automatiquement.

Bind localhost uniquement : vos services backend (Gitea, Vaultwarden, Node) ne doivent écouter que sur `127.0.0.1`, jamais sur `0.0.0.0` — cloudflared est le seul point d'entrée réseau.

Règles WAF Cloudflare : même en tunnel, vous pouvez appliquer des règles WAF et des listes d'IP autorisées dans le dashboard pour bloquer des pays entiers ou des plages IP connues malveillantes.

Rotation des credentials : `cloudflared tunnel token --cred-file nouveau-fichier.json` génère de nouveaux credentials sans recréer le tunnel. Rotez après départ d'un collaborateur ayant eu accès au serveur.

Pour un accompagnement structuré sur la sécurisation de vos services exposés, notre service RSSI externalisé couvre l'architecture tunnel, la configuration Zero Trust et le monitoring continu de vos expositions.

Limites et pièges à ne pas ignorer

Cloudflare Tunnel n'est pas une solution universelle. Voici ce qui ne fonctionne pas ou pose des problèmes en production.

Protocoles non-HTTP/TCP : les services nécessitant UDP (jeux en réseau, certains VoIP, DNS custom) ne passent pas par cloudflared en mode standard. Cloudflare WARP + Gateway couvre ce besoin mais avec une architecture différente et un coût supérieur.

Dépendance unique à Cloudflare : si Cloudflare est down (incidents rares mais réels, notamment en juin 2022 et novembre 2023), vos services deviennent inaccessibles. Pour les applications critiques, maintenez un accès de secours parallèle — VPN IPsec ou WireGuard — testé régulièrement.

Inspection du contenu en edge : le trafic HTTPS est déchiffré au niveau des edge servers Cloudflare pour appliquer WAF et DDoS filtering. Vos données transitent en clair dans l'infrastructure Cloudflare. Acceptable pour la majorité des cas d'usage, à évaluer soigneusement pour des données soumises à des contraintes de résidence strictes (HDS, secret médical, données gouvernementales).

Latence sur applications temps réel : si votre serveur est en France et vos utilisateurs aussi, le tunnel ajoute typiquement 8–20 ms selon le PoP Cloudflare. Acceptable pour des web apps, potentiellement problématique pour du streaming basse latence ou des applications industrielles temps réel.

Déployer cloudflared dans Docker : alternative au service systemd

Si votre infrastructure repose entièrement sur Docker (Portainer, Docker Swarm, Kubernetes), cloudflared s'intègre nativement comme container plutôt que comme service systemd. Cette approche simplifie la gestion des mises à jour et l'isolation des processus.

```
# docker-compose.yml — cloudflared en container
version: '3.8'
services:
  cloudflared:
    image: cloudflare/cloudflared:latest
    restart: always
    command: tunnel --config /etc/cloudflared/config.yml run
    volumes:
      - ~/.cloudflared:/etc/cloudflared:ro
    network_mode: host  # Accès direct au réseau hôte pour les services locaux
    environment:
      - TUNNEL_TOKEN=${CLOUDFLARE_TUNNEL_TOKEN}  # Alternative aux credentials JSON
```

L'alternative au fichier credentials JSON est l'utilisation d'un **tunnel token** — une chaîne base64 qui encode les credentials et que vous définissez comme variable d'environnement. Plus pratique en environnement Docker ou Kubernetes (stockage en Secret K8s) :

```
# Obtenir le token du tunnel
cloudflared tunnel token mon-serveur-maison
# Résultat : eyJhIjoiYWJjZGVm...

# Lancer cloudflared directement avec le token
docker run -d   --name cloudflared   --restart always   cloudflare/cloudflared:latest   tunnel ru
```

Sur **Kubernetes**, stockez le token dans un Secret et montez-le comme variable d'environnement dans le Deployment. Les 4 connexions simultanées que cloudflared maintient vers l'edge Cloudflare sont compatibles avec les contraintes réseau des clusters K8s — pas besoin de règles réseau spéciales, uniquement des sorties HTTPS sur 443.

Cas d'usage avancé : exposer Proxmox VE en remote sans IP publique

Proxmox VE (Virtual Environment) est un hyperviseur qui tourne sur des serveurs sans nécessairement d'IP publique routée — fréquent en homelab ou dans des datacenters qui ne proposent qu'une seule IP publique. Cloudflare Tunnel permet d'accéder à l'interface web Proxmox (port 8006) et au shell des VMs depuis n'importe où sans exposer le serveur.

Configuration spécifique pour Proxmox (certificat auto-signé) dans config.yml :

```
ingress:
  - hostname: proxmox.exemple.fr
    service: https://localhost:8006
    originRequest:
      noTLSVerify: true          # Ignorer le cert auto-signé Proxmox
      httpHostHeader: proxmox.exemple.fr   # Header Host correct pour Proxmox
      originServerName: proxmox  # SNI correct
```

Une fois le tunnel configuré avec Cloudflare Access (identifiants d'administrateur ou OTP email), vous accédez à votre Proxmox depuis n'importe quel navigateur dans le monde sans jamais ouvrir le port 8006 sur votre routeur. C'est une réduction de surface d'attaque significative pour un hyperviseur qui gère potentiellement des données sensibles. Pour des architectures virtualisées sécurisées, consultez nos ressources sur l'implémentation Zero Trust réseau qui couvre les patterns d'accès aux hyperviseurs dans un cadre Zero Trust.

Cloudflare Tunnel n'est qu'un des composants d'une stratégie de sécurité complète. Pour un accompagnement global sur votre posture de sécurité, notre service RSSI externalisé propose un audit et une roadmap sécurité adaptée à votre infrastructure.

Questions fréquentes

Cloudflare Tunnel est-il vraiment gratuit ?

Oui, les tunnels HTTP/HTTPS et TCP sont inclus dans le plan Free de Cloudflare, sans limite de bande passante annoncée. Cloudflare Access (authentification des utilisateurs) est gratuit jusqu'à 50 utilisateurs dans le plan Zero Trust Free. Les fonctions avancées — Browser Isolation pour RDP via navigateur, Device Posture, accès WARP-to-Tunnel — nécessitent un plan payant à partir de 7 $/utilisateur/mois. Pour un homelab personnel ou une petite équipe, le Free couvre 95 % des besoins.

Peut-on exposer plusieurs services avec un seul tunnel ?

Absolument, c'est même la configuration recommandée. Un seul daemon cloudflared avec un fichier `config.yml` expose autant de services que nécessaire via des règles `ingress` différenciées par hostname. Un seul tunnel, une dizaine de sous-domaines, zéro overhead supplémentaire côté Cloudflare. La limite pratique vient des ressources du serveur hébergeant cloudflared, pas d'une contrainte technique de la plateforme.

Cloudflare Tunnel peut-il remplacer un VPN d'entreprise ?

Pour l'accès à des services web spécifiques, oui dans la majorité des cas. Pour un accès réseau complet (toute la plage IP interne, protocoles non-HTTP/TCP), non — utilisez **Cloudflare WARP** combiné à des Private Networks exposés via tunnel. La distinction est importante : cloudflared expose des services précis, pas un réseau entier. Pour SSH et RDP sans port ouvert, Cloudflare Access avec le client WARP ou le mode browser couvre le besoin.

Comment sécuriser cloudflared en production ?

Cinq mesures essentielles : (1) cloudflared tourne sous un compte système dédié sans shell ni sudo, (2) le fichier credentials JSON est en mode 600 propriétaire du compte cloudflared, (3) vos applications backend écoutent uniquement sur 127.0.0.1, jamais sur 0.0.0.0, (4) Cloudflare

Access est activé sur tous les services sensibles — le tunnel expose, Access contrôle l'accès, (5) les credentials sont rotés après tout départ d'équipe. Ajoutez des règles WAF et des listes d'IP autorisées dans le dashboard Cloudflare pour une défense en profondeur.

Que se passe-t-il si cloudflared plante ou si le serveur redémarre ?

Avec le service systemd installé via `cloudflared service install`, le daemon redémarre automatiquement en cas de crash ou de reboot serveur. Cloudflare maintient votre tunnel en "standby" côté edge pendant les reconnexions — la plupart des clients ne voient qu'un bref timeout. Pour la haute disponibilité, cloudflared supporte le mode répliqué : plusieurs instances sur des machines différentes pour le même tunnel UUID, avec basculement automatique en cas de panne d'une instance. Cloudflare détecte la perte d'une instance en moins de 30 secondes et route vers les instances saines.

