

# Limōps

saé 34 : Infrastructure virtualisée

Nathan DECOOL

# Qui suis-je ?



Nathan DECOOL - 28 ans

Ingénieur Réseau et Télécom - Ingénieur NetOps



# Qui êtes vous ?

Type de bac

Expérience (Stage/alternance)

Projet Pro

Projet Post BUT

activité/projet extra scolaire

# Modalités

1

Une restitution de 20 minutes par groupe de 3

2

Attendus

40% - Réalisation Technique : "Ça marche." (Le `docker-compose up` lance tout, les tests locaux répondent).

40% - Synthèse & Mise en abyme : "Je comprends." (Expliquer comment cette architecture locale s'appliquerait à une vraie prod).

20% - Esprit Critique : "Je prends du recul." (Sécurité, persistance des données, limites de la simulation localhost).

# Sommaire

**1** Le projet

**2** Les services

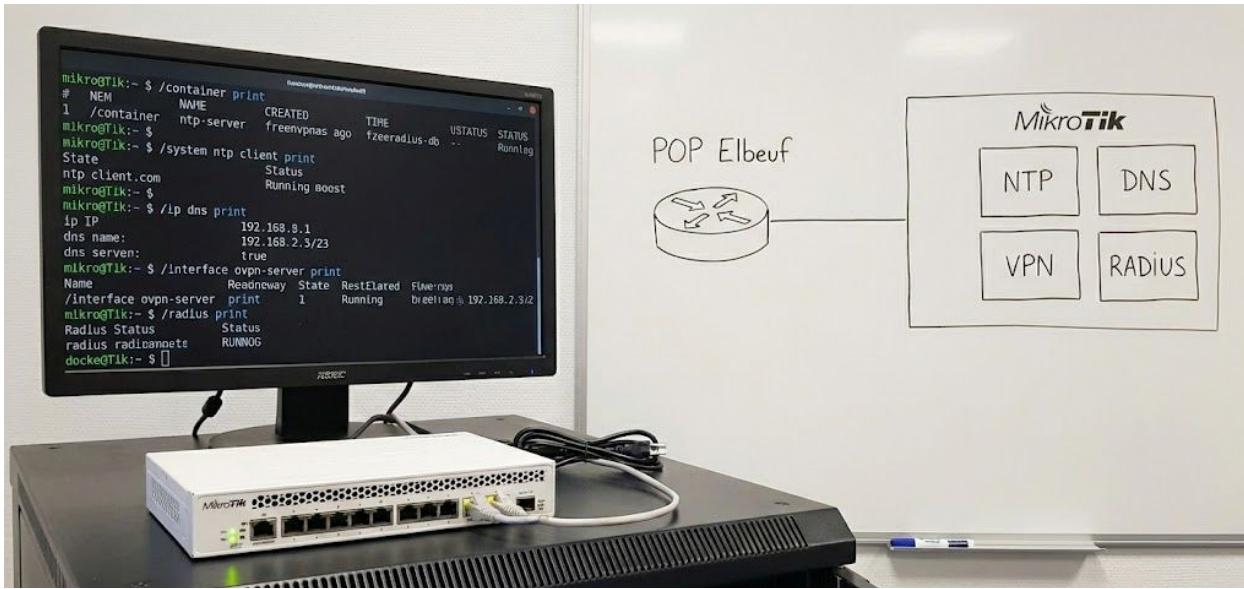
**3** Docker

**4** GIT

# 1.1 Le projet - Contexte

**Contexte** : Vous êtes technicien réseau chez un opérateur.

**Mission** : Prototyper une stack de services réseau conteneurisés (Docker) destinée à être déployée en production sur des routeurs (MikroTik) ou des serveurs de POP.



## 1.2 Le projet - Défi

**En Production (La vraie vie)** : Ces services tourneraient sur des serveurs rackables ou directement sur les routeurs (Edge Routers).

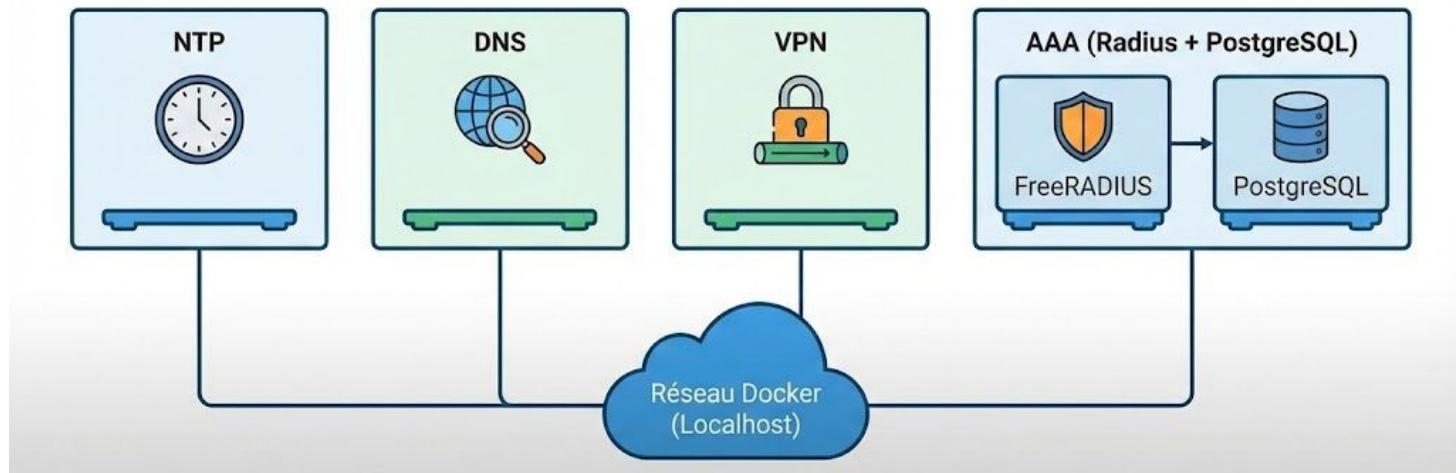
**En SAÉ (Aujourd'hui)** : Tout se passe sur votre machine.

- Côté Serveur : Docker héberge l'infrastructure (NTP, DNS, Radius, DB).
- Côté Client : Votre propre OS (Windows/Linux) ou des outils de test ([nslookup](#), [radtest](#)) simuleront les requêtes du réseau.

**Avantage** : Développement rapide et tests instantanés sans dépendance matérielle.

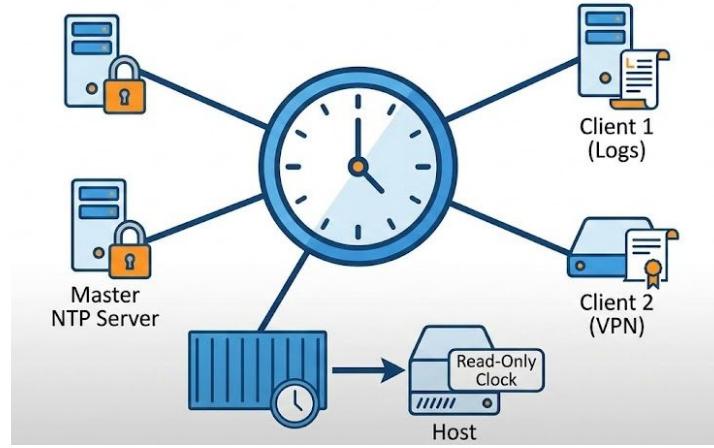
## 2.1 Les services - Stack

### LA STACK “OPÉRATEUR”



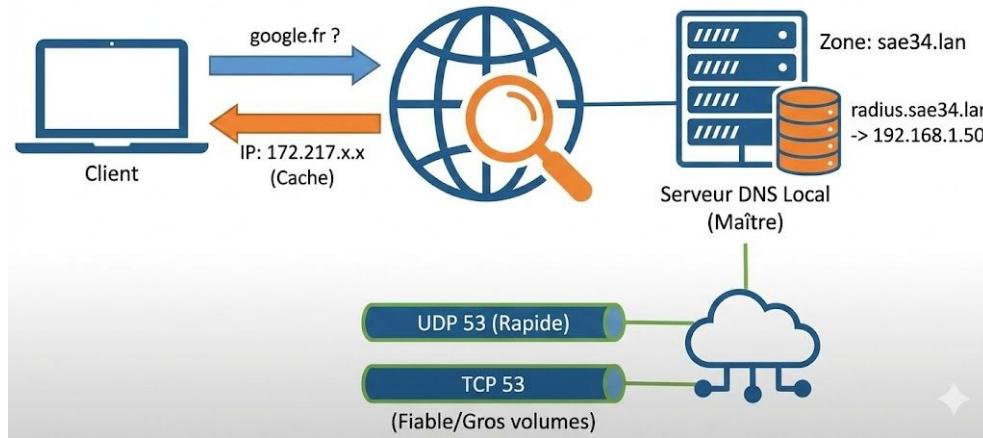
## 2.2 Les services - NTP

- Rôle : Garantir une horloge unique pour la corrélation des logs et la validité des certificats, les routeurs clients et tous les équipements nécessaire.
- Challenge Technique : Un conteneur ne peut pas changer l'heure du noyau hôte par défaut. Il doit agir en relais.
- Logiciel suggéré : Chrony.
- Test de validation : `ntpdate -q 127.0.0.1`.



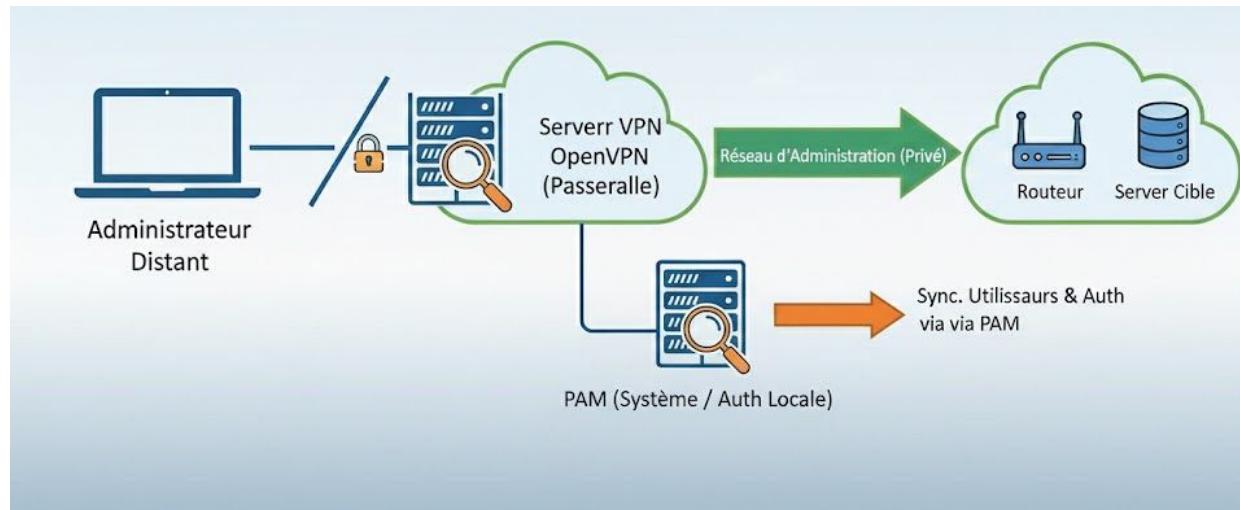
## 2.3 Les services - Résolveur DNS

- Rôle : Resolver (Cache pour accélérer le web) + Zone Locale optionnelle (Authoritative pour `sae34.lan`)
- Logiciel suggéré : BIND9.
- Ports : Attention, le DNS utilise UDP/53 (standard) ET TCP/53 (réponses > 512 octets ou transferts de zone).
- Test de validation : `dig @127.0.0.1 -p 53 google.fr`



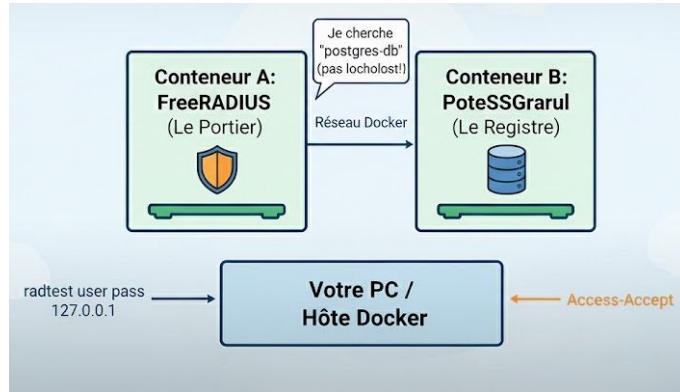
## 2.4 Les services - Serveur VPN

- Rôle : Permettre à un administrateur de se connecter au réseau de gestion de manière chiffrée.
- Spécificité Docker : Le conteneur a besoin de privilèges élevés (**NET\_ADMIN**) pour créer l'interface réseau virtuelle (**tun0**).
- Logiciel : OpenVPN
- Test de validation : nc -u -v -z 127.0.0.1 1194

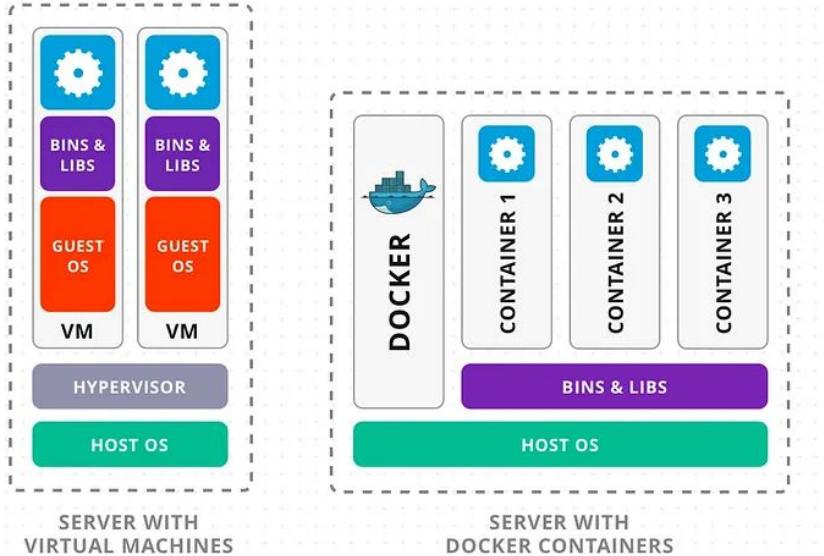


## 2.5 Les services - AAA

- Rôle : Authentifier les utilisateurs (Radius) et stocker leurs données (PostgreSQL).
- Architecture Micro-Services :
  - a. Conteneur A (Cerveau) : FreeRADIUS.
  - b. Conteneur B (Mémoire) : PostgreSQL.
- Le Piège : FreeRADIUS ne doit pas chercher la base de données sur `localhost` (sa propre machine virtuelle), mais sur le nom DNS du conteneur DB via le réseau Docker.
- Test de validation : `radtest user pass 127.0.0.1 1812 secret`



# 3.1 Docker - KESAKO ?



Définition simple : Docker est une technologie de conteneurisation. Il permet d'isoler des applications avec leurs dépendances (bibliothèques, binaires) dans des "boîtes" appelées conteneurs.

- Avantage clé : La Légèreté.
  - VM (Machine Virtuelle) : Chaque VM a son propre OS complet (noyau + applications). C'est lourd et lent.
  - Conteneur Docker : Partage le noyau Linux de la machine hôte. Il n'embarque que l'application et ses bibliothèques spécifiques. C'est beaucoup plus léger et démarre en quelques secondes.
- Le Mot-clé : Portabilité. Un conteneur Docker fonctionne exactement de la même manière sur n'importe quelle machine (Linux, Mac, Windows avec WSL) qui a Docker installé.

## 3.2 Docker - La Contrainte "From Scratch"

- Règle du jeu : Toutes vos images doivent partir de `FROM debian:trixie-slim` ou de toute autre distribution linux comme `ubuntu` ou `alpine` par exemple.
- Pourquoi Trixie (Debian 13) ? C'est la version actuelle stable de debian. En tant d'administrateurs réseau, vous devez savoir préparer l'infrastructure de demain avec les paquets les plus récents.
- L'Exercice : Pas de script magique. Vous installez (`apt`), configurez et lancez les services vous-mêmes.



## 3.3 Docker - Le Piège du PID 1 (Foreground vs Background)

- Problème : Dans un conteneur Debian de base, `systemd` n'existe pas. Les commandes classiques (`service start`) lancent le processus en arrière-plan.
- Conséquence : Le conteneur s'arrête immédiatement car Docker pense que le travail est fini.
- Solution : Chaque service doit être lancé avec une commande qui le maintient au premier plan (ex: `named -g`, `chronyd -d`, `freeradius -X`, `postgres -D`).

## 3.4 Docker - L'Orchestration (Docker Compose)

- **L'outil :** Le fichier `docker-compose.yml`.
- **Fonction :** Décrit et coordonne toute l'infrastructure multi-conteneurs (réseaux virtuels, volumes, ports).
- **Gestion des Dépendances :** Il permet de définir l'ordre de démarrage (ex: la base de données PostgreSQL doit être lancée avant FreeRADIUS).

```
version: '3.8'  
services:  
  ntp:  
    build: ./ntp  
    cap_add: [ ??? ]      # TODO: Droit pour l'heure système ?  
    ports: [ "????:????/udp" ] # TODO: Port NTP ?
```

```
dns:  
  build: ./dns  
  ports:  
    - "????:????/udp"      # TODO: Ports DNS ?  
    - "????:????/tcp"
```

```
vpn:  
  build: ./vpn  
  cap_add: [ ??? ]      # TODO: Droit pour l'interface TUN ?  
  devices: [ "/dev/net/tun" ]  
  ports: [ "1194:1194/udp" ]
```

```
postgres:  
  build: ./postgres  
  environment:  
    POSTGRES_PASSWORD: ???  
  volumes: [ "pg-data:/var/lib/postgresql/data" ]  
  
radius:  
  build: ./radius  
  ports: [ "1812:1812/udp" ]  
  depends_on: [ ??? ]      # TODO: Ordre de démarrage ?  
  environment:  
    DB_HOST: ???          # TODO: Nom du service DB ?  
  
networks: { pop-net: {} }  
volumes: { pg-data: {} }
```

# 3.5 Docker - Volume, Réseau, Port Mapping

- 1. Le Réseau Interne (Bridge)
  - C'est quoi ? Un switch virtuel isolé créé par Docker.
  - Usage : Permet aux conteneurs de discuter entre eux (ex: Radius ↔ PostgreSQL).
  - Magie : Résolution DNS automatique par nom de service (`ping postgres` fonctionne !).
- 2. Le Port Mapping (Exposition)
  - C'est quoi ? Une règle de NAT (Translation d'Adresse) entre votre PC et le conteneur.
  - Usage : Permet d'accéder aux services depuis votre Windows/Linux (ex: `127.0.0.1:53 → Conteneur:53`).
  - Syntaxe : `-p PortHôte:PortConteneur` (ex: `8080:80`).
- 3. Les Volumes (Persistance)
  - C'est quoi ? Un dossier de votre disque dur physique "monté" dans le conteneur.
  - Usage : Sauvegarder les données critiques (Base de données, Clés VPN).
  - Sans volume : Si le conteneur redémarre, tout est effacé.

# 4.1 GIT - KESAKO

C'est quoi ? L'historique de vos changements. Votre "Machine à remonter le temps".

Pourquoi ? Permet à plusieurs personnes de travailler sur le même projet sans s'écraser les fichiers et d'avoir du versionnage

Le principe :

- `git pull` : Récupérer le travail des autres.
- `git add + git commit + git push` : Enregistrer et partager votre travail.

⚠ Règle d'or : Toujours faire un `git pull` avant de commencer à coder.

Ici je vous conseille d'utiliser GITHUB qui est gratuit, attention malgré tout à passer vos repos en privés.



## 4.2 GIT - Organisation

Règle d'or : Un seul dépôt Git par groupe.

Structure imposée : Pour faciliter l'intégration, respectez cette arborescence :

```
/sae34-groupe-X/
  └── dns/
      ├── Dockerfile
      └── config/ (named.conf, zones...)
  └── ntp/
      ├── Dockerfile
      └── chrony.conf
  └── radius/
      ├── Dockerfile
      └── raddb/ (users, clients.conf)
          └── init_db.sql (pour PostgreSQL)
  └── vpn/
      ├── Dockerfile
      └── server.conf
└── docker-compose.yml
```

Pour une bonne répartition des tâches je vous conseille de vous séparer les tâches comme suit :

- L'Intégrateur (A) : Le garant du `docker-compose.yml`, du VPN et du réseau.
- L'Admin Services (B) : Le responsable du DNS et du NTP.
- L'Admin Backend (C) : Le responsable du couple FreeRADIUS/ PostgreSQL

# saé 34 : Infrastructure virtualisée - la restitution

**Livrable Final :** Le dépôt Git doit être parfaitement fonctionnel.

**Démonstration :** Vous lancez docker-compose up et vous exécutez les "Tests Faciles" devant l'intervenant.

**Esprit Critique (20%) :** Vous identifiez ce qui ne va pas pour une *vraie* production (manque de redondance, secrets en clair, pas de HTTPS).

Des questions ? Vous pouvez me contacter par mail à l'adresse :

[nathan@limops.net](mailto:nathan@limops.net)