Noyau 5G (5GC)

Le noyau 5G autonome (SA) est déployé par le playbook `services/5gc.yml`. Un seul rôle partagé, `5gc_nf`, installe chaque fonction réseau (NF). Le rôle est paramétré par NF. Cette page explique ce rôle, les NF qu'il construit et la configuration que chacune lit.

Aperçu

Plutôt qu'un rôle par NF, OmniCore utilise **un rôle (`5gc_nf`) pour toutes les onze NF 5GC**. Le playbook invoque le rôle une fois par NF et passe une variable `nf_package` qui sélectionne le paquet Debian, le service systemd et le modèle de configuration Jinja2. Cela maintient l'installation, la version de configuration, la licence et la logique de redémarrage identiques à travers chaque NF.

```
- name: Setup AMF
  hosts: amf
  roles:
    - {role: 5gc_nf, nf_package: omniamf, become: yes}
```

Fonctions Réseau

Chaque NF a son propre groupe d'inventaire, paquet, service systemd et modèle de configuration (`roles/5gc_nf/templates/<nf_package>.j2`).

NF	`nf_package`	Groupe d'inventaire	Modèle de configuration	Ré
Fonction de Répertoire Réseau	`omninrf`	`nrf`	`omninrf.j2`	Découv service Enregist NF
Gestion de l'Accès et de la Mobilité	`omniamf`	`amf`	`omniamf.j2`	Signalis N1/N2, enregist gestion connexi
Serveur d'Authentification	`omniausf`	`ausf`	`omniausf.j2`	Authent 5G-AKA AKA'
Fonction de Support de Liaison	`omnibsf`	`bsf`	`omnibsf.j2`	Liaison pour un session
Fonction de Facturation	`omnichf`	`chf`	`omnichf.j2`	Factura ligne/ho converg
Sélection de Tranche Réseau	`omnissf`	`nssf`	`omnissf.j2`	NSSAI / sélectio tranche
Fonction de Contrôle de Politique	`omnipcf`	`pcf`	`omnipcf.j2`	Règles politiqu
Proxy de Communication de Service	`omniscp`	`scp`	`omniscp.j2`	Routage messag

NF	`nf_package`	Groupe d'inventaire	Modèle de configuration	Rô
				commu indirect
Fonction de Gestion de Session	`omnismf`	`smf`	`omnismf.j2`	Contrôle session UPF (N4
Gestion Unifiée des Données	`omniudm`	`udm`	`omniudm.j2`	Donnée d'abonn dé- conceal SUCI
Répertoire Unifié des Données	`omniudr`	`udr`	`omniudr.j2`	Magasir donnée soutien UDM/PC

Comment fonctionne le rôle `5gc_nf`

Pour chaque NF, le rôle effectue la même séquence :

1. **Résoudre `group_vars_omnicore`** : défini sur `<inventory>/group_vars/`, afin que les recherches de configuration se résolvent par rapport à l'inventaire actif.
2. **Découvrir les serveurs de licence** : si `license_server_api_urls` n'est pas défini, il est dérivé du groupe d'inventaire `license_server` (`https://<host>:8443/api`).
3. **Installer les prérequis** : `lsb-release`, `curl`, `gpg`, `unzip`, `libglu1-mesa`, `libsctp1`.
4. **Installer le paquet NF** depuis le dépôt APT d'Omnitouch. Dernière version par défaut ; épinglez avec `nf_version` (voir Version pinning).

5. **Modèle de la configuration d'exécution (versionnée)** : rend `templates/<nf_package>.j2` vers `/etc/<nf_package>/runtime-<deb_version>.exs`.
6. **Créer un lien symbolique `runtime.exs`** → la configuration de la version active, et refléter ce lien symbolique dans chaque répertoire `/opt/<nf_package>/releases/*`.
7. **Redémarrer la NF** (gestionnaire) chaque fois que la configuration active change.

Version pinning et rollback

La configuration est écrite par version Debian (`runtime-<deb_version>.exs`) et `runtime.exs` est un lien symbolique vers la version active. Les configurations précédentes sont préservées sur le disque, donc un rollback consiste simplement à réinstaller la version de paquet plus ancienne :

```
- {role: 5gc_nf, nf_package: omniamf, nf_version: "1.2.2-1",
become: yes}
```

Omettez `nf_version` pour suivre le dernier paquet.

Clés HNET UDM (dé-concealement SUCI)

L'UDM (`nf_package: omniudm`) a des étapes supplémentaires pour soutenir le dé-concealement SUCI (TS 33.501 §6.12). Le dé-concealement récupère le SUPI à partir du SUCI dissimulé qu'un UE présente lors de l'enregistrement. Il a besoin des clés **privées** du Réseau Domiciliaire qui s'associent aux clés **publiques** du Réseau Domiciliaire sur les USIM.

- Les clés se trouvent dans `group_vars/hnet_udm/` sur le contrôleur Ansible sous la forme `<curve>-<id>.key` (PEM), où `<curve>` est `curve25519` (Profil A, X25519) ou `secp256r1` (Profil B, prime256v1), et `<id>` est l'identifiant de clé publique du Réseau Domiciliaire.

- Si le répertoire est **vide**, une clé par défaut `curve25519-1.key` est auto-générée afin que le dé-concealement fonctionne dès la première utilisation. Cette étape est idempotente. Une clé existante n'est jamais écrasée.
- Pour chaque `curve25519-*.key`, la valeur publique brute correspondante de 32 octets est exportée (hex) vers un fichier `.pub` pour la provision de l'USIM. Les fichiers `.pub` restent dans l'inventaire ; seuls les fichiers `*.key` sont déployés vers `/etc/omniudm/hnet/` sur l'UDM (mode `0600`, root).

> Une clé auto-générée est aléatoire et ne dé-conceale que les SUCI des USIM programmés avec sa clé publique **correspondante**. Prenez le `.pub` exporté et provisionnez-le comme la Clé Publique du Réseau Domiciliaire sur l'USIM. Les SIM utilisant le schéma nul n'ont pas besoin de clé HNET et peuvent laisser le répertoire vide.

Détails complets et exemples de disposition de fichiers : Configuration des Variables de Groupe : Exemple 8.

Documentation Connexe

- Playbooks de Service - Tous les playbooks de service et la hiérarchie de déploiement
- Noyau Commuté - Le domaine CS hérité (BSC, MSC, SS7)
- Configuration des Variables de Groupe - Modèles de configuration par NF et clés HNET
- Architecture de Déploiement - Comment les domaines s'assemblent
- **Configuration OmniCore** : https://docs.omnitouch.com.au/docs/repos/OmniCore - Ce qui va dans les fichiers de configuration NF

Norme de Planification IP OmniCore

Vue d'ensemble

Ce document décrit l'approche standard de planification IP pour les déploiements OmniCore. L'architecture nécessite **quatre sous-réseaux internes** pour segmenter correctement les fonctions réseau pour la sécurité, la performance et la clarté opérationnelle.

Exigences d'Allocation IP

Allocation Standard : Quatre Sous-Réseaux /24

Chaque déploiement OmniCore nécessite quatre sous-réseaux distincts pour le réseau interne :

1. **Réseau de Noyau de Paquet** - Premier /24
2. **Réseau de Signalisation** - Deuxième /24
3. **Réseau Interne IMS** - Troisième /24
4. **Réseau Public IMS** - Quatrième /24

Important : Ce sont des Recommandations, Pas des Exigences

L'allocation de sous-réseaux décrite dans ce document est une **meilleure pratique recommandée** pour organiser les déploiements OmniCore. Cependant, l'architecture est **complètement flexible** :

- **Tous les hôtes dans un sous-réseau** : Vous pouvez placer tous les composants dans un seul sous-réseau si cela convient à vos besoins de déploiement

- **Chaque type d'hôte dans son propre sous-réseau** : Vous pouvez créer des sous-réseaux séparés pour chaque type de composant (un pour les MMEs, un pour les HSS, etc.)
- **Groupements personnalisés** : Vous pouvez organiser les hôtes dans toute structure de sous-réseau qui a du sens pour vos exigences spécifiques
- **Mélanger les IP internes et publiques** : Certains hôtes peuvent utiliser des adresses internes (RFC 1918) tandis que d'autres utilisent des IP publiques, le tout dans le même déploiement

L'approche recommandée à quatre sous-réseaux offre une **isolation de sécurité, une gestion du trafic et une clarté opérationnelle** optimales, c'est pourquoi nous la suggérons pour les déploiements en production. Cependant, vous devez adapter le plan IP pour l'adapter à votre topologie réseau spécifique, à l'espace d'adresses disponible et à vos exigences opérationnelles.

Répartition des Segments Réseau

1. Réseau de Noyau de Paquet (Premier /24)

Objet : Éléments du plan de données utilisateur et du plan de contrôle central

Composants :

- OmniMME (Entité de Gestion de Mobilité)
- OmniSGW (Passerelle de Service)
- OmniPGW-C (Plan de Contrôle de la Passerelle PDN)
- OmniUPF/PGW-U (Fonction de Plan Utilisateur / Passerelle PDN Plan Utilisateur)

Exemple : `10.179.1.0/24`

```
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.179.1.15
      gateway: 10.179.1.1
      host_vm_network: "v400-omni-packet-core"
```

---

2. Réseau de Signalisation (Deuxième /24)

Objet : Signalisation Diameter, politique, facturation et fonctions de gestion

Composants :

- OmniHSS (Serveur d'Abonnés à Domicile)
- OmniCharge OCS (Système de Facturation en Ligne)
- OmniHSS PCRF (Fonction de Règles de Politique et de Facturation)
- OmniDRA DRA (Agent de Routage Diameter)
- Serveurs DNS (DNS interne/infrastructure - distinct du DNS orienté client dans le segment Public IMS ; les deux sont intentionnels)
- Serveurs TAP3/CDR
- Surveillance/OAM
- Capture SIP
- Serveur de Licences
- Moniteur RAN
- Omnitouch Warning Link CBC (Centre de Diffusion Cellulaire) - s'il est déployé

Exemple : `10.179.2.0/24`

```yaml
hss:
  hosts:
    omni-site-hss01:
      ansible_host: 10.179.2.140
      gateway: 10.179.2.1
      host_vm_network: "v401-omni-signalling"
```

---

3. Réseau Interne IMS (Troisième /24)

Objet : Signalisation et services IMS de base (signalisation SIP interne)

Composants :

- OmniCSCF S-CSCF (Fonction de Contrôle de Session d'Appel de Service)
- OmniCSCF I-CSCF (Fonction de Contrôle de Session d'Appel Interrogative)
- OmniTAS (Serveur d'Applications de Téléphonie / Serveur d'Applications)
- OmniMessage (Contrôleur SMS, SMPP, IMS)

Exemple : `10.179.3.0/24`

```yaml
scscf:
  hosts:
    omni-site-scscf01:
      ansible_host: 10.179.3.45
      gateway: 10.179.3.1
      host_vm_network: "v402-omni-ims-internal"
```

---

4. Réseau Public IMS (Quatrième /24)

Objet : Services IMS orientés utilisateur, DNS orienté client, et interconnexion SS7/legacy

Composants :

- OmniCSCF P-CSCF (Fonction de Contrôle de Session d'Appel Proxy)
- Serveurs XCAP
- Serveurs de Messagerie Vocale Visuelle
- DNS Client
- OmniSS7 STP (Point de Transfert de Signalisation SS7)
- OmniSS7 HLR (Registre de Localisation à Domicile) - pour 2G/3G
- OmniSS7 IP-SM-GW (MAP SMSc)
- OmniSS7 Passerelle CAMEL
- MSC (Centre de Commutation Mobile)

> **Remarque :** Les fonctions SS7/HLR et d'interconnexion legacy associées se trouvent généralement dans le segment public, car elles se connectent avec des réseaux externes 2G/3G via SIGTRAN/M3UA plutôt que via le SIP IMS interne.

Exemple : `10.179.4.0/24`

```
pcscf:
  hosts:
    omni-site-pcscf01:
      ansible_host: 10.179.4.165
      gateway: 10.179.4.1
      host_vm_network: "v403-omni-ims-public"
```

Méthodes de Mise en Œuvre

OmniCore prend en charge deux méthodes principales pour mettre en œuvre cette segmentation réseau :

Méthode 1 : Interfaces Réseau Physiques/Virtualisées (Recommandé pour la Production)

Utilisez des NIC séparés ou des ponts virtuels pour chaque segment de réseau. Cela offre la meilleure isolation et est l'approche recommandée pour les déploiements en production. La convention est de nommer chaque pont d'après son segment (`v400-omni-packet-core`, `v401-omni-signalling`, `v402-omni-ims-internal`, `v403-omni-ims-public`), bien qu'un seul pont partagé (par exemple `vmbr0`) soit également valide lorsque la segmentation est gérée ailleurs.

Exemple :

```yaml
# Noyau de Paquet - v400-omni-packet-core
mme:
  hosts:
    omni-site-mme01:
      ansible_host: 10.179.1.15
      gateway: 10.179.1.1
      host_vm_network: "v400-omni-packet-core"

# Signalisation - v401-omni-signalling
hss:
  hosts:
    omni-site-hss01:
      ansible_host: 10.179.2.140
      gateway: 10.179.2.1
      host_vm_network: "v401-omni-signalling"

# IMS Interne - v402-omni-ims-internal
icscf:
  hosts:
    omni-site-icscf01:
      ansible_host: 10.179.3.55
      gateway: 10.179.3.1
      host_vm_network: "v402-omni-ims-internal"

# IMS Public - v403-omni-ims-public
pcscf:
  hosts:
    omni-site-pcscf01:
      ansible_host: 10.179.4.165
      gateway: 10.179.4.1
      host_vm_network: "v403-omni-ims-public"
```

Méthode 2 : Segmentation Basée sur VLAN

Utilisez une seule interface physique avec un balisage VLAN pour séparer les réseaux. Cela convient aux déploiements plus petits ou lorsque les NIC physiques sont limitées.

Exemple :

```yaml
# Tous les composants partagent un pont (ovsbr1) ; chaque segment utilise un tag VLAN différent.
# applicationserver (TAS) est un composant IMS Interne -> VLAN 402 / 10.178.3.x
applicationserver:
  hosts:
    omni-site-sbc01:
      ansible_host: 10.178.3.213
      gateway: 10.178.3.1
      host_vm_network: "ovsbr1"
      vlanid: "402"

# dra et dns sont des composants de Signalisation -> VLAN 401 / 10.178.2.x
dra:
  hosts:
    omni-site-dra01:
      ansible_host: 10.178.2.211
      gateway: 10.178.2.1
      host_vm_network: "ovsbr1"
      vlanid: "401"

dns:
  hosts:
    omni-site-dns01:
      ansible_host: 10.178.2.178
      gateway: 10.178.2.1
      host_vm_network: "ovsbr1"
      vlanid: "401"
```

Configuration Réseau :

- Configurez les VLAN sur le commutateur physique
- Taguez le trafic de manière appropriée au niveau de l'hyperviseur
- Routez entre les VLAN au niveau de la passerelle/pare-feu

Exemple de Mappage VLAN :

```
VLAN 400 : 10.x.1.0/24 (Noyau de Paquet)
VLAN 401 : 10.x.2.0/24 (Signalisation)
VLAN 402 : 10.x.3.0/24 (IMS Interne)
VLAN 403 : 10.x.4.0/24 (IMS Public)
```

Travailler avec des Adresses IP Publiques

Vue d'ensemble

De nombreux déploiements OmniCore nécessitent que certains composants aient des adresses IP publiques pour la connectivité externe, telles que :

- **DRA** - Pour la signalisation diameter en itinérance avec des opérateurs externes
- **SGW/PGW en itinérance** - Pour le trafic GTP des partenaires d'itinérance
- **ePDG** - Pour les appels WiFi (tunnels IPsec depuis les UE)
- **Passerelle SMSC** - Pour les connexions SMPP aux agrégateurs SMS externes
- **P-CSCF** (dans certains déploiements) - Pour l'enregistrement SIP direct des UE

Comment Assigner des IP Publiques

Les IP publiques sont gérées **exactement de la même manière que les IP internes** dans vos fichiers d'inventaire d'hôtes. Il suffit de spécifier l'adresse IP publique dans le champ `ansible_host` avec la passerelle et le masque de sous-réseau appropriés.

Exemple : SGW/PGW en itinérance avec IP Publiques

```yaml
sgw:
  hosts:
    # SGWs internes sur le réseau privé
    customer-site-sgw01:
      ansible_host: 10.4.1.25
      gateway: 10.4.1.1
      host_vm_network: "v400-omni-packet-core"

    # SGWs en itinérance avec des IP publiques
    customer-site-roaming-sgw01:
      ansible_host: 203.0.113.10
      gateway: 203.0.113.9
      netmask: 255.255.255.248       # sous-réseau /29
      host_vm_network: "498-public-servers"
      in_pool: False
      cdrs_enabled: True

smf:  # PGWs
  hosts:
    # PGW en itinérance avec IP publique
    customer-site-roaming-pgw01:
      ansible_host: 203.0.113.20
      gateway: 203.0.113.17
      netmask: 255.255.255.240       # sous-réseau /28
      host_vm_network: "497-public-services-LTE"
      in_pool: False
      ip_pools:
        - '100.64.24.0/22'
```

Exemple : DRA avec IP Publique

```yaml
dra:
  hosts:
    customer-site-dra01:
      ansible_host: 198.51.100.50
      gateway: 198.51.100.49
      netmask: 255.255.255.240       # sous-réseau /28
      host_vm_network: "497-public-services-LTE"
```

Exemple : ePDG avec IP Publique

```yaml
epdg:
  hosts:
    customer-site-epdg01:
      ansible_host: 198.51.100.51
      gateway: 198.51.100.49
      netmask: 255.255.255.240       # sous-réseau /28
      host_vm_network: "497-public-services-LTE"
```

Mélange d'IP Internes et Publiques

Un mélange d'IP internes et publiques au sein du même groupe de composants est courant. Par exemple :

- SGWs internes pour les sites locaux utilisant GTP
- SGWs publiques spécifiquement pour le trafic d'itinérance des opérateurs externes
- Le même PGW-C peut gérer à la fois les SGWs internes et externes

L'architecture d'OmniCore gère cela de manière transparente - il suffit de configurer chaque hôte avec son adressage IP approprié.

Introduction au déploiement Ansible chez Omnitouch

Vue d'ensemble

Omnitouch Network Services utilise Ansible comme plateforme d'automatisation de l'infrastructure pour déployer des solutions complètes de réseau cellulaire (4G/5G) de manière cohérente, répétable et automatisée. Ce document fournit un aperçu de la manière dont nous utilisons Ansible pour orchestrer des déploiements complexes dans le secteur des télécommunications.

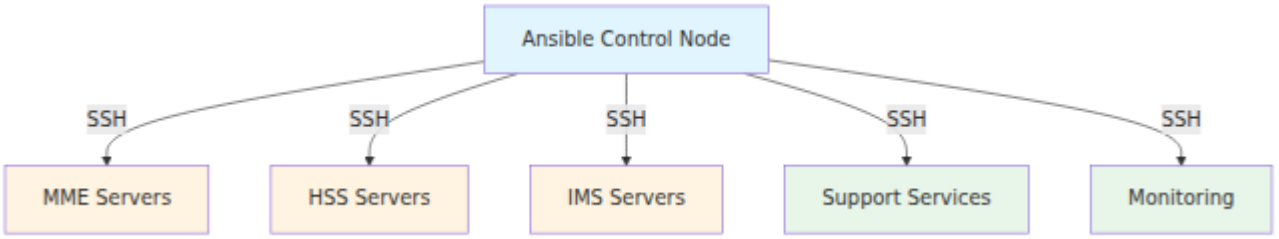
Qu'est-ce qu'Ansible ?

Ansible est un outil d'automatisation open-source qui vous permet de :

- Configurer des systèmes
- Déployer des logiciels
- Orchestrer des flux de travail complexes
- Gérer l'infrastructure en tant que code

Ansible utilise une approche déclarative - vous décrivez l'**état souhaité** de vos systèmes, et Ansible s'assure qu'ils atteignent cet état.

Comment Omnitouch utilise Ansible



Concepts clés

1. Inventaire (Fichiers d'hôtes)

Définit **quels** systèmes gérer. Chaque déploiement client a un fichier d'hôtes qui décrit :

- Toutes les machines virtuelles dans le réseau
- Leurs adresses IP
- Configuration réseau
- Paramètres spécifiques aux services

Les fichiers d'hôtes sont ce avec quoi vous allez travailler pour définir votre réseau.

Voir : Configuration du fichier d'hôtes

2. Rôles

Définit **comment** configurer chaque composant. Les rôles sont des unités réutilisables qui contiennent :

- Tâches (étapes à exécuter)
- Modèles (modèles de fichiers de configuration)
- Gestionnaires (actions déclenchées par des changements)
- Variables (valeurs de configuration par défaut)

Exemples de rôles pour les composants OmniCore : `omnihss`, `omnisgwc`, `omnipgwc`, `omnidra`, etc.

L'équipe ONS définit ces rôles. Vous pouvez les modifier. Cependant, il existe généralement des moyens plus propres d'apporter les ajustements nécessaires depuis votre fichier d'hôtes.

3. Playbooks

Orchestre **quand** et **où** les rôles sont appliqués :

```
- name: Setup Omnicore MME
  hosts: mme
  roles:
    - omnimme
```

Nous les utilisons essentiellement comme groupes pour les rôles.

En pratique, chaque playbook de service applique un seul rôle, et la configuration partagée est tirée séparément. Un bundle de services comme `services/epc.yml` utilise `import_playbook` pour d'abord exécuter `common.yml` puis chaque playbook par service :

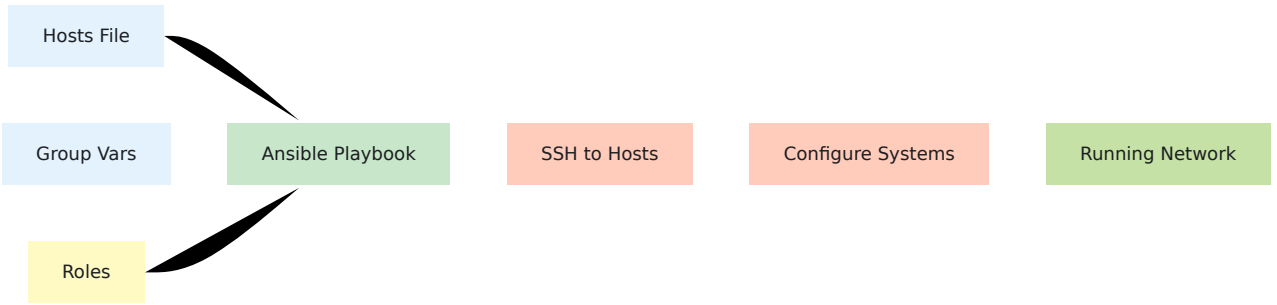
```
- name: Install Common
  import_playbook: common.yml

- name: Run MME Playbook
  import_playbook: omnimme.yml
```

4. Variables de groupe

Fournit une **configuration spécifique au client** qui remplace les valeurs par défaut des rôles. C'est ici que la personnalisation du client se fait sans modifier les rôles de base.

Voir : Variables de groupe et configuration

Architecture de déploiement



Le processus de déploiement

1. Définir l'infrastructure

Créez un fichier d'hôtes décrivant votre topologie réseau :

Remarque de planification : Avant de définir l'infrastructure, consultez la Norme de planification IP pour des conseils sur la segmentation du réseau, l'allocation des adresses IP et l'organisation des sous-réseaux.

Utilisateurs de Proxmox : Si vous déployez sur Proxmox, consultez Déploiement de VM/LXC Proxmox pour la provision de VM/conteneurs automatisée.

Voir : Configuration du fichier d'hôtes et Référence de configuration

```
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15
      mme_code: 1
```

2. Personnaliser la configuration

Définissez des variables spécifiques au client dans `group_vars` :

```yaml
plmn_id:
  mcc: '001'
  mnc: '01'
customer_name_short: customer
```

#ToDo - Ajouter un lien ici vers la référence de configuration pour la liste complète

3. Exécuter les playbooks

Déployez le réseau :

```
ansible-playbook -i hosts/customer/host_files/production.yml
services/epc.yml
```

4. Déploiement automatisé

Ansible va :

- Créer/provisionner des VM (si vous utilisez l'intégration Proxmox/VMware)
- Configurer le réseau
- Installer des paquets logiciels depuis le cache APT
- Déployer le code applicatif
- Configurer les services avec les paramètres du client
- Démarrer les services
- Valider le déploiement

Composants clés que nous déployons

OmniCore (Plateforme de cœur de paquet 4G/5G)

- **OmniHSS** - Serveur d'abonnés à domicile
- **OmniSGW** - Passerelle de service (plan de contrôle)
- **OmniPGW** - Passerelle de paquet (plan de contrôle)
- **OmniUPF** - Fonction de plan utilisateur
- **OmniDRA** - Agent de routage Diameter
- **OmniTWAG** - Passerelle d'accès WLAN de confiance

Voir : https://docs.omnitouch.com.au/docs/repos/OmniCore

OmniCall (Plateforme de voix et de messagerie)

- **OmniCall CSCF** - Fonction de contrôle de session d'appel (P-CSCF, I-CSCF, S-CSCF)
- **OmniTAS** - Serveur d'application IMS (services VoLTE/VoNR)
- **OmniMessage** - Centre SMS (SMS-C)
- **OmniMessage SMPP** - Support du protocole SMPP
- **OmniSS7** - Composants de signalisation SS7 (STP, HLR, CAMEL)
- **VisualVoicemail** - Fonctionnalité de messagerie vocale

Voir : https://docs.omnitouch.com.au/docs/repos/OmniCall

OmniCharge/OmniCRM

- **Plateforme CRM** - Gestion de la relation client, auto-inscription, facturation

Voir : https://docs.omnitouch.com.au/docs/repos/OmniCharge

Services de support

- **DNS** - Résolution DNS réseau
- **Serveur de licences** - Gestion des licences
- **Surveillance** - Prometheus, Grafana

Voir : Aperçu de l'architecture de déploiement

Gestion des paquets

Nous utilisons un modèle de distribution de paquets hybride :

Paquets APT précompilés

Tous les logiciels Omnitouch sont distribués sous forme de paquets Debian (`.deb` files) :

- Construits à partir de la source dans notre pipeline CI/CD
- Versionnés et testés
- Hébergés sur des dépôts de paquets

Système de cache APT

Les clients peuvent choisir entre :

1. **Cache APT local** - Miroir des paquets requis sur site pour un déploiement hors ligne
2. **Dépôt public** - Accès direct au dépôt de paquets hébergé par Omnitouch

Il existe également un `.deb` unifié `omnicore` (construit à partir de `packaging/`) qui regroupe les playbooks et les outils pour l'auto-provisionnement d'un déploiement.

Voir : Système de cache APT

Gestion des licences

Tous les composants logiciels Omnitouch nécessitent des licences valides gérées par un serveur de licences central :

- Les composants vérifient la validité de la licence au démarrage
- Les fonctionnalités sont activées/désactivées en fonction de la licence
- Le serveur de licences peut être local ou hébergé dans le cloud

Voir : Serveur de licences

Avantages de cette approche

Répétabilité

Les mêmes playbooks Ansible peuvent déployer :

- Laboratoires de développement
- Environnements de test
- Réseaux de production
- Sites clients

Cohérence

Chaque déploiement utilise les mêmes configurations testées, réduisant ainsi les erreurs humaines.

Contrôle de version

L'infrastructure est définie comme du code dans Git :

- Suivre tous les changements
- Réviser avant le déploiement
- Revenir en arrière si nécessaire

Personnalisation sans complexité

Les clients peuvent personnaliser leur déploiement via `group_vars` sans modifier les rôles principaux.

Déploiement rapide

Déployez un réseau cellulaire complet en quelques heures au lieu de jours ou de semaines.

Prise en main

Prérequis

Avant d'exécuter les playbooks Ansible, vous devez installer les dépendances requises. Ce projet utilise uv pour gérer l'environnement Python, donc **uv est requis**.

1. Installer uv

Si uv n'est pas déjà installé, installez-le en utilisant la méthode qui convient à votre nœud de contrôle. Consultez la documentation d'installation de uv. Par exemple :

```
pipx install uv                                    # via pipx
curl -LsSf https://astral.sh/uv/install.sh | sh    # installateur autonome
```

2. Installer les dépendances

Depuis la racine du dépôt, exécutez :

```
uv sync
```

Cela crée un `.venv/` isolé et installe Ansible ainsi que tous les paquets Python nécessaires (pinnés dans `uv.lock`) pour l'automatisation du déploiement

Omnitouch.

3. Activer l'environnement

Activez l'environnement virtuel et maintenez-le activé pour toutes les commandes Ansible :

```
source .venv/bin/activate      # Windows : .venv\Scripts\activate
```

4. Installer les collections Ansible

Installez les collections Ansible dont dépendent les playbooks (à partir de `requirements.yml`) :

```
ansible-galaxy collection install -r requirements.yml
```

5. Exécuter les commandes Ansible

Avec l'environnement activé, exécutez les commandes Ansible directement :

```
ansible --version
```

Remarque : Désactivez l'environnement lorsque vous avez terminé en exécutant `deactivate`.

Étapes de déploiement

1. Consultez la Configuration du fichier d'hôtes pour comprendre comment définir votre réseau
2. Apprenez à connaître les Variables de groupe pour la personnalisation
3. Comprenez le Système de cache APT pour la gestion des paquets
4. Consultez l'Architecture de déploiement pour voir comment tout s'assemble
5. Déployez !

Prochaines étapes

- Norme de planification IP - **Planifiez votre architecture réseau et votre allocation IP**
- Configuration du fichier d'hôtes - Apprenez à définir votre topologie réseau
- Système de cache APT - Comprenez la distribution des paquets
- Serveur de licences - Apprenez-en plus sur la gestion des licences
- Aperçu de l'architecture de déploiement - Voir l'ensemble du tableau
- Configuration des variables de groupe - Personnalisez votre déploiement
- Playbooks utilitaires - Outils opérationnels pour les vérifications de santé, les sauvegardes et la maintenance

Dépôt APT & Distribution de Paquets

Vue d'ensemble

Le système APT d'Omnitouch fournit une distribution de paquets pour tous les déploiements. Deux types de contenu sont servis :

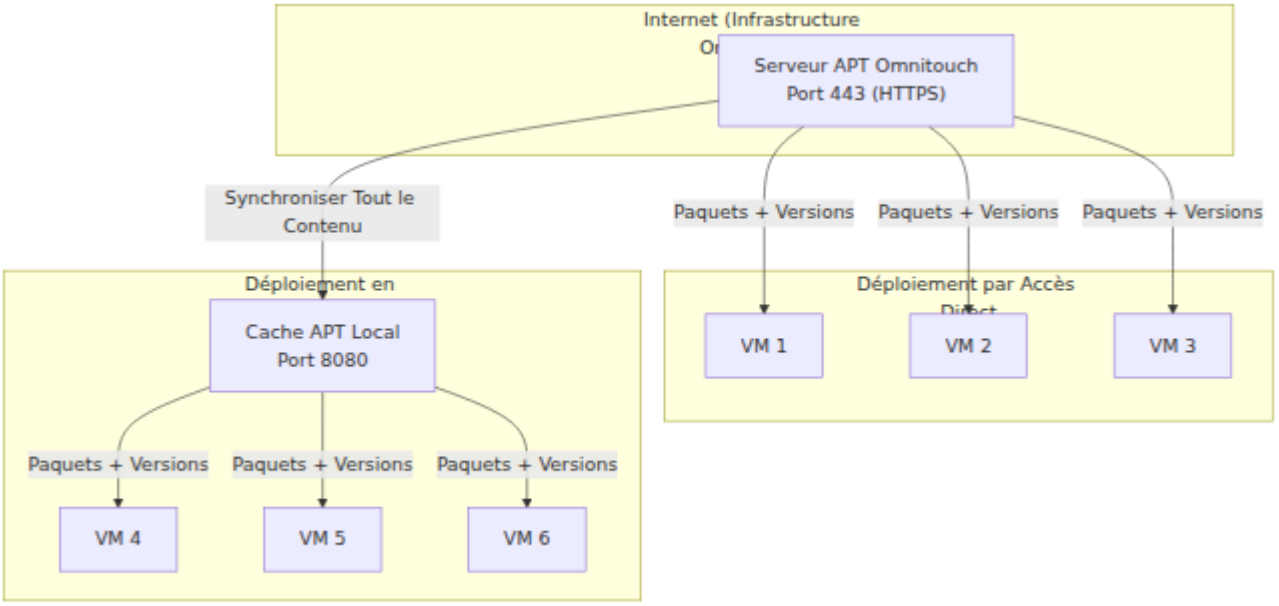
1. **Paquets APT** : paquets Debian installés via `apt install`
2. **Versions binaires** : binaires pré-construits téléchargés directement (exportateurs Prometheus, agents, etc.)

La page des paquets sur le serveur APT comprend également un onglet **Auto-configuration (OmniCore)**, qui fournit le paquet `omnicore` (servi à `/pool/main/o/omnicore/`) contenant l'ensemble complet d'outils d'automatisation Ansible pour les clients qui souhaitent déployer et gérer leur réseau eux-mêmes.

Deux modèles de déploiement sont pris en charge :

1. **Accès direct** : les VM tirent les paquets directement du serveur APT d'Omnitouch (par exemple, `apt.<region>.omnitou.ch`)
2. **Miroir de cache local** : un serveur local se synchronise avec Omnitouch et sert des paquets aux VM (pour des déploiements hors ligne/isolés)

Architecture



Contenu Servi

Le serveur APT héberge tout le contenu requis pour les déploiements :

Type de Contenu	Description	Chemin
Paquets Omnitouch	Paquets `.deb` construits sur mesure (omnihss, omnimme, etc.)	`/dists/<distro>/`
Ensemble d'Outils OmniCore	Le paquet `omnicore` : automatisation Ansible complète (rôles, playbooks, dépendances) pour déploiement en libre-service	`/pool/main/o/omnicore/`
Paquets Ubuntu	Paquets Ubuntu mis en cache avec toutes les dépendances	`/<distro>/pool/main/`
Versions GitHub	Binaires pré-construits (Prometheus, Grafana, Homer, etc.)	`/releases/<org>/<repo>/`
Archives Sources	Archives sources pour applications web (CGrateS_UI, speedtest)	`/repos/`
Paquets de Tiers	Galera, FRR, InfluxDB, KeyDB, etc.	`/releases/<vendor>/`

Variables de Configuration

`apt_repo` est la seule source de vérité pour la distribution des paquets. Définissez-la une fois dans votre inventaire. Tout le reste en découle. Vous ne configurez pas séparément l'URL de téléchargement binaire, le schéma, le port ou les identifiants.


apt_repo
(source de vérité
unique,
définie dans l'inventaire)
Dérivé
remote_apt_*
(roles/common/defaults/main.yml)
apt_download_base
(roles/common/tasks/facts.yml)
Ce qu'ils configurent
/etc/apt/sources.list.d/omnitouch.list
Téléchargements
binaires
/releases/, /repos/

Comment fonctionne la dérivation

Variable	Où définie	Rôle
`apt_repo`	Votre inventaire	Le seul bloc que vous configurez. Templaté directement dans la ligne source APT.
`remote_apt_*`	`roles/common/defaults/main.yml`	Dérivé de `apt_repo`. Dernier recours lorsque `apt_repo` est absent : `apt.us-fl.omnitou.ch` / `443` / `https`.
`apt_download_base`	`roles/common/tasks/facts.yml`	URL de base calculée unique que chaque rôle utilise pour les téléchargements binaires, afin que la ligne source APT et les téléchargements binaires soient toujours d'accord sur l'hôte, le schéma, le port

Variable	Où définie	Rôle
		et les identifiants.

Parce que les deux valeurs dérivées sont construites à partir du même bloc `apt_repo`, un inventaire normal **n'a besoin que de `apt_repo`**. Vous ne définissez pas `remote_apt_*` vous-même.

> **Exception : le serveur miroir de cache.** Un hôte dans le groupe `apt_cache_servers` est le seul endroit où `remote_apt_*` est encore défini explicitement. Son propre `apt_repo` pointe vers le *cache local*. Les valeurs `remote_apt_*` décrivent ensuite le serveur Omnitouch *amont* dont il se synchronise (voir Option 2).

Schéma et port

Le schéma suit automatiquement le port : **port `443` → `https`, tout autre port → `http`**. Lorsque `apt_repo.apt_repo_port` est omis :

- **Accès direct** par défaut à **`443` (HTTPS)**
- **Mode cache** par défaut à **`8080` (HTTP)**

Quand chaque mode est utilisé

Scénario	Ligne source APT	Téléchargements binaires (`apt_download_base`)
`use_apt_cache: true` (cache)	`http://<cache>:8080/...` (sans authentification)	`http://<cache>:8080/...` (sans authentification)
`use_apt_cache: false` (direct)	`https://<user>:<pass>@<server>/...`	`https://<user>:<pass>@<server>:443/...`

Option 1 : Accès Direct

Pour les déploiements avec connectivité Internet, les VM tirent les paquets directement du serveur APT d'Omnitouch.

Exigences Réseau

Liste blanche des IP sources : Votre adresse IP publique doit être sur liste blanche sur le serveur APT d'Omnitouch. Lors de la configuration, fournissez vos sous-réseaux sources à Omnitouch. En retour, vous recevrez :

- **Nom d'utilisateur** et **mot de passe** pour l'authentification HTTP Basic
- **FQDN** pour le serveur APT (par exemple, `apt.<region>.omnitou.ch`)

Exigences de Pare-feu : L'accès sortant aux plages IP suivantes d'Omnitouch doit être autorisé :

Réseau	Plage
IPv4	`144.79.167.0/24`
IPv4	`160.22.43.0/24`
IPv6	`2001:df3:dec0::/48`
ASN	`AS152894`

Services nécessitant un accès à l'infrastructure Omnitouch :

Service	Port	Protocole	But
Serveur APT	443	TCP	Téléchargements de paquets (HTTPS)
Serveur APT	53	TCP/UDP	Résolution DNS pour le FQDN APT
Serveur de Licences	123	UDP	Synchronisation horaire NTP pour validation de licence
Serveur de Licences	53	TCP/UDP	Résolution DNS pour validation de licence

Assurez-vous que le trafic HTTPS (TCP/443), NTP (UDP/123) et DNS (TCP+UDP/53) est autorisé vers les plages IP d'Omnitouch.

Configuration

```yaml
all:
  vars:
    use_apt_cache: false

    # Source unique de vérité. Configuration de téléchargement binaire (remote_apt_*) et
    # l'URL de base apt_download_base sont dérivées de cela automatiquement - vous ne devez PAS
    # les définir séparément.
    apt_repo:
      apt_server: "apt.<region>.omnitou.ch"   # FQDN fourni par Omnitouch
      apt_repo_username: "your-username"
      apt_repo_password: "your-password"
      # apt_repo_port: 443                     # optionnel ; par défaut à 443 (HTTPS)
```

Paramètres

Sources de Paquets APT (`apt_repo`)

Paramètre	Type	Requis	Par défaut	Description
`apt_repo.apt_server`	Chaîne	Oui	-	Nom d'hôte ou adresse IP du serveur APT
`apt_repo.apt_repo_username`	Chaîne	Oui	-	Nom d'utilisateur pour l'authentificat HTTP Basic
`apt_repo.apt_repo_password`	Chaîne	Oui	-	Mot de passe pour l'authentificat HTTP Basic
`apt_repo.apt_repo_port`	Entier	Non	`443`	Port du serveu Le schéma sui le port (`443` → HTTPS, sinon HTTP).

Téléchargements Binaires (`remote_apt_*`)

Ceci est dérivé automatiquement de `apt_repo`. Ne les définissez pas dans un inventaire d'accès direct. Ils existent uniquement pour que le serveur miroir de cache puisse pointer vers un amont différent (voir Option 2). Pour référence, ils par défaut comme suit :

Paramètre	Dérivé de	Sauvegarde
`remote_apt_server`	`apt_repo.apt_server`	`apt.us-fl.omnitou.ch`
`remote_apt_port`	`apt_repo.apt_repo_port`	`443`
`remote_apt_protocol`	port (`443` → `https`)	`https`
`remote_apt_user`	`apt_repo.apt_repo_username`	`""`
`remote_apt_password`	`apt_repo.apt_repo_password`	`""`

Général

Paramètre	Type	Requis	Par défaut	Description
`use_apt_cache`	Booléen	Oui	-	Doit être `false` pour un accès direct

Modèles d'URL (Accès Direct)

Sources de Paquets APT (configurées dans `/etc/apt/sources.list.d/omnitouch.list`) :

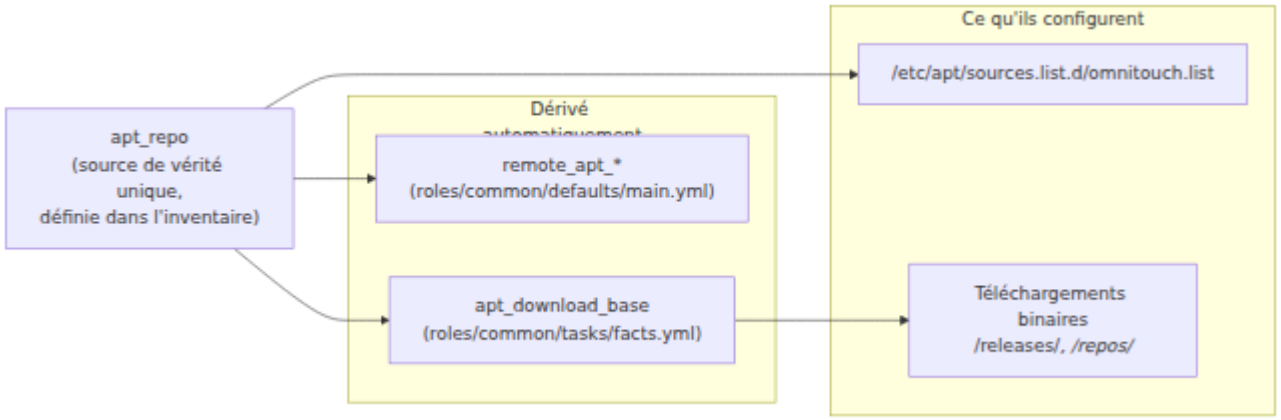
```
deb [trusted=yes] https://{username}:{password}@{apt_server}/
noble main
```

Téléchargements Binaires (construits à partir de `apt_download_base` par les tâches `get_url` d'Ansible) :

```
https://{username}:
{password}@{apt_server}/releases/prometheus/node_exporter/node_export
1.8.1.linux-amd64.tar.gz
```

> Le port `443` est omis de l'URL (c'est le défaut HTTPS). Un port non standard est ajouté comme `:<port>` et change le schéma en HTTP à moins qu'il ne soit `443`.

Comment Cela Fonctionne



Les VM s'authentifient avec l'authentification HTTP Basic pour les paquets APT et les téléchargements binaires. Les paquets système Ubuntu sont également servis depuis le serveur Omnitouch (pré-mis en cache), donc les VM n'ont pas besoin d'accéder aux miroirs Ubuntu.

Signature de Paquet (GPG)

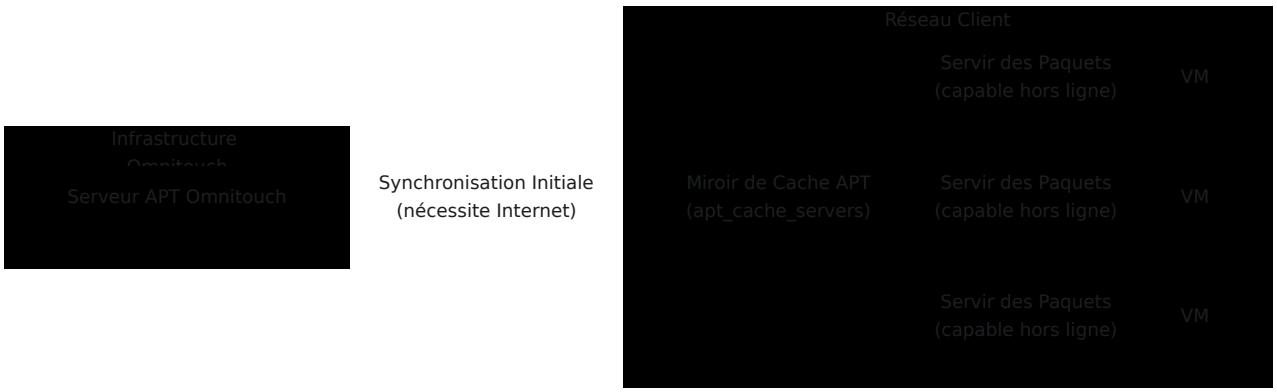
Le serveur APT d'Omnitouch génère une clé de signature de dépôt et sert la clé publique armée à `/omnitouch.gpg`. Pour les hôtes d'accès direct, le rôle `common` récupère automatiquement cette clé et l'installe dans `/etc/apt/trusted.gpg.d/omnitouch.gpg` afin que `apt-get update` vérifie les métadonnées du dépôt sans avertissements `NO_PUBKEY`. Cela se produit automatiquement ; aucune gestion manuelle des clés n'est requise.

> Le mode miroir de cache n'utilise pas la vérification GPG. Les hôtes qui tirent d'un cache local utilisent `[trusted=yes]` à la place (voir Option 2).

Option 2 : Miroir de Cache Local

Pour des déploiements hors ligne, isolés ou contraints en bande passante, déployez un cache APT local qui synchronise tout le contenu d'Omnitouch.

Architecture



Configuration

Définissez le serveur de cache dans votre fichier d'hôtes. C'est le seul endroit où `remote_apt_*` est défini explicitement. Ces variables décrivent le serveur Omnitouch **amont** dont le cache se synchronise :

```
apt_cache_servers:
  hosts:
    example-apt-cache:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # Le serveur de cache synchronise le contenu DU dépôt
Omnitouch amont
    remote_apt_server: "apt.<region>.omnitou.ch"
    remote_apt_port: 443
    remote_apt_protocol: "https"
    remote_apt_user: "your-username"
    remote_apt_password: "your-password"

all:
  vars:
    # use_apt_cache: true  # Auto-configuré lorsque le groupe
apt_cache_servers existe
    # apt_repo.apt_server: auto-dérivé à 192.168.1.100 (premier
serveur de cache)
```

Comment cela fonctionne :

- **Serveur de cache** (`192.168.1.100`) : Utilise les identifiants `remote_apt_*` pour synchroniser le contenu depuis le serveur Omnitouch amont via HTTPS.
- **Tous les autres hôtes** : Dérivent automatiquement `apt_repo.apt_server: "192.168.1.100"` et tirent du cache au port `8080` sans identifiants.

Paramètres

Sources de Paquets APT (`apt_repo`)

Paramètre	Type	Requis	Par défaut	Descripti
`apt_repo.apt_server`	Chaîne	Oui	Auto-dérivé	Adresse IP du serveur de ca local. Dérivée automatiquem du premier hô `apt_cache_se` si non spécifié
`apt_repo.apt_repo_username`	Chaîne	Non	-	Non requis lor l'utilisation du cache (aucune authentificatio nécessaire)
`apt_repo.apt_repo_password`	Chaîne	Non	-	Non requis lor l'utilisation du cache (aucune authentificatio nécessaire)
`apt_repo.apt_repo_port`	Entier	Non	`8080`	Port que les h utilisent pour atteindre le se de cache loca (Remplace le `apt_cache_po` obsolète.)

Synchronisation du Serveur de Cache (`remote_apt_*`)

Ces variables configurent comment le serveur de cache synchronise le contenu d'Omnitouch. C'est le seul scénario où `remote_apt_*` est défini manuellement :

Paramètre	Type	Requis	Par défaut	Description
`remote_apt_server`	Chaîne	Oui	-	Serveur APT Omnitouch amont à synchroniser
`remote_apt_port`	Entier	Non	`443`	Port du serveur APT Omnitouch amont
`remote_apt_protocol`	Chaîne	Non	`https`	Protocole pour la connexion de synchronisation
`remote_apt_user`	Chaîne	Oui	-	Identifiants pour la synchronisation depuis Omnitouch
`remote_apt_password`	Chaîne	Oui	-	Identifiants pour la synchronisation depuis Omnitouch

Général

Paramètre	Type	Requis	Par défaut	Description
`use_apt_cache`	Booléen	Non	`true`	Défini automatiquement sur `true` lorsque le groupe `apt_cache_servers` existe

> **Obsolète :** `apt_cache_port` n'est plus utilisé. Le port auquel les hôtes se connectent est maintenant défini via `apt_repo.apt_repo_port` (par défaut à `8080` lorsqu'il est omis).

Modèles d'URL (Mode Cache)

Sources de Paquets APT (configurées dans `/etc/apt/sources.list.d/omnitouch.list`) :

```
deb [trusted=yes] http://192.168.1.100:8080/noble noble main
```

Téléchargements Binaires (construits à partir de `apt_download_base`) :

```
http://192.168.1.100:8080/releases/prometheus/node_exporter/node_expo
1.8.1.linux-amd64.tar.gz
```

L'accès au cache n'a pas besoin d'identifiants. Il utilise la configuration APT `[trusted=yes]`.

Déploiement du Cache

1. **Provisionnez le serveur de cache** (VM ou conteneur LXC avec disque de 50 Go ou plus)
2. **Exécutez le playbook de configuration du cache :**

   ```
   ansible-playbook -i hosts/customer/production.yml
   services/apt_cache.yml
   ```

3. **Vérifiez le cache** en naviguant vers `http://192.168.1.100:8080/`

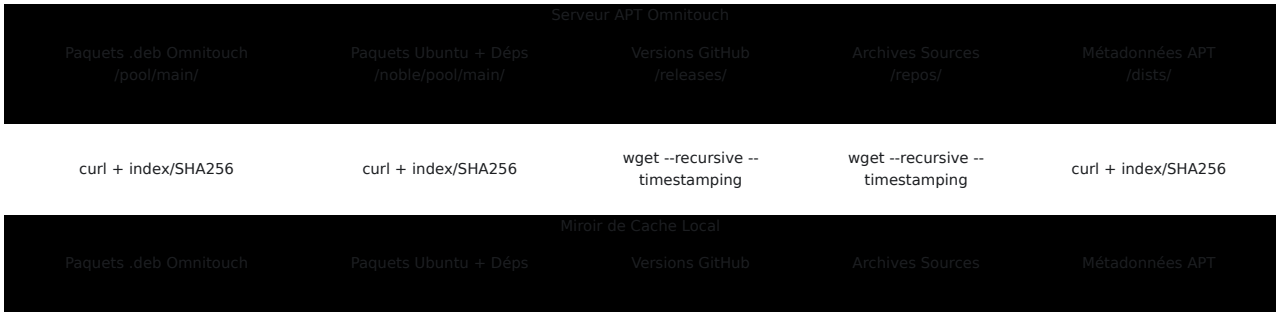
Ce qui est synchronisé

Le miroir de cache synchronise **tout le contenu** du serveur APT d'Omnitouch. Le mécanisme diffère selon le type de contenu :

- **Paquets APT** (`/dists/` + `/pool/` pour Omnitouch et Ubuntu) sont synchronisés avec `curl` piloté par l'index de dépôt Debian. Le miroir récupère le fichier `Release` amont (et court-circuite s'il est inchangé), analyse `Packages.gz` pour le nom de fichier/SHA256/taille autoritaire de

chaque paquet, et télécharge uniquement les fichiers manquants ou dont le SHA256 ne correspond pas à la copie locale. Chaque fichier téléchargé est vérifié par SHA256 avant d'être engagé.

- **Versions binaires et archives sources** (`/releases/`, `/repos/`) n'ont pas d'index de style Debian, donc ils tombent en arrière sur `wget --recursive --timestamping`, qui ne re-télécharge que les fichiers avec un `Last-Modified` amont plus récent.

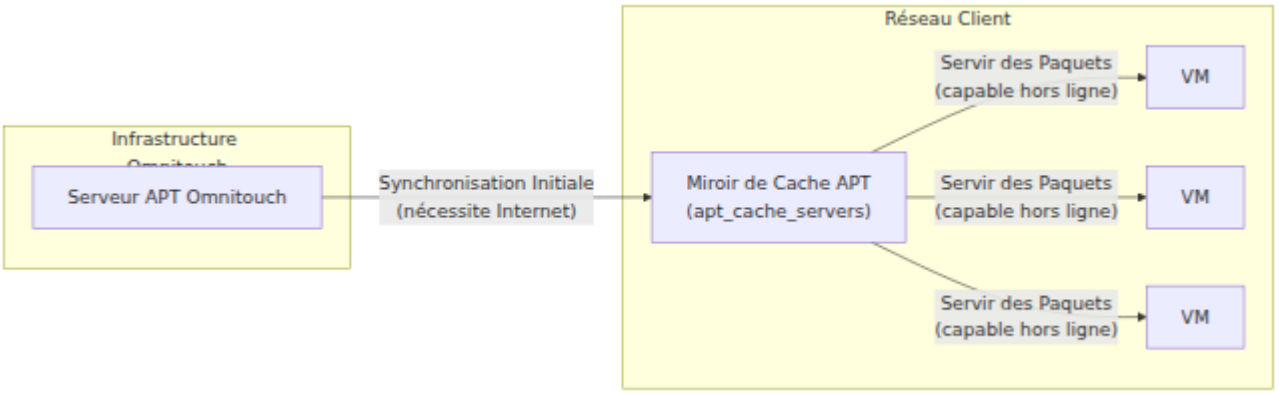


Répertoires de contenu synchronisés :

Chemin	Contenu
`/dists/<distro>/`	Métadonnées du dépôt APT (fichiers Packages, Release)
`/pool/main/`	Paquets .deb personnalisés d'Omnitouch
`/<distro>/pool/main/`	Paquets Ubuntu et toutes les dépendances
`/releases/`	Versions GitHub (Prometheus, Grafana, Zabbix, etc.)
`/repos/`	Archives sources (Erlang, Elixir, CGrateS_UI, etc.)

Après la synchronisation initiale, le cache peut servir tous les paquets sans connectivité Internet.

Comment Cela Fonctionne



Le miroir de cache s'authentifie avec l'authentification HTTP Basic pour tout le contenu. Les paquets APT sont synchronisés avec `curl` contre l'index de dépôt Debian (`Release` + `Packages.gz`), téléchargeant uniquement les fichiers dont le SHA256 diffère de la copie locale et vérifiant chacun après le téléchargement ; `/releases/` et `/repos/` tombent en arrière sur `wget --recursive --timestamping`. Dans tous les cas, les synchronisations suivantes ne récupèrent que les fichiers nouveaux ou modifiés.

Configuration Automatique

Lorsqu'un groupe `apt_cache_servers` existe dans votre inventaire, le système :

1. Définit `use_apt_cache: true` pour tous les hôtes (sauf si explicitement remplacé)
2. Dérive `apt_repo.apt_server` de l'IP `ansible_host` du premier serveur de cache

Exemple de Configuration Minimale

```yaml
apt_cache_servers:
  hosts:
    apt-cache-01:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # Le serveur de cache synchronise le contenu depuis le dépôt Omnitouch amont
    remote_apt_server: "apt.<region>.omnitou.ch"
    remote_apt_user: "your-username"
    remote_apt_password: "your-password"
```

Ce qui se passe automatiquement :

- Tous les hôtes (sauf le serveur de cache) obtiennent `use_apt_cache: true`
- Tous les hôtes (sauf le serveur de cache) obtiennent `apt_repo.apt_server: "192.168.1.100"`
- Tous les hôtes tirent de `http://192.168.1.100:8080/` sans identifiants
- Le serveur de cache synchronise les paquets depuis `https://your-username:your-password@apt.<region>.omnitou.ch/`

Remplacer le Comportement Automatique

Pour forcer l'accès direct même avec des serveurs de cache définis :

```yaml
all:
  vars:
    use_apt_cache: false  # Forcer l'accès direct même avec des serveurs de cache définis

    apt_repo:
      apt_server: "apt.<region>.omnitou.ch"
      apt_repo_username: "user"
      apt_repo_password: "pass"
```

Résumé de Configuration

Scénario 1 : Accès Direct au Serveur APT (Pas de Cache)

Tous les hôtes tirent les paquets directement du serveur de dépôt APT. Seul `apt_repo` est nécessaire. Les valeurs `remote_apt_*` en sont dérivées.

```yaml
all:
  vars:
    use_apt_cache: false

    # Source unique de vérité pour les sources APT et les
téléchargements binaires
    apt_repo:
      apt_server: "apt.<region>.omnitou.ch"
      apt_repo_username: "user"
      apt_repo_password: "pass"
```

Résultat : Tous les hôtes génèrent `deb [trusted=yes] https://user:pass@apt.<region>.omnitou.ch/ noble main`

Scénario 2 : Serveur de Cache APT Défini dans le Fichier d'Hôtes (Automatique)

Le serveur de cache est dans votre inventaire et sera déployé/synchronisé par Ansible.

```yaml
apt_cache_servers:
  hosts:
    cache-server:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # Le serveur de cache synchronise les paquets depuis le dépôt
authentifié amont
    remote_apt_server: "apt.<region>.omnitou.ch"
    remote_apt_port: 443
    remote_apt_protocol: "https"
    remote_apt_user: "user"
    remote_apt_password: "pass"

# Aucune configuration nécessaire dans all: vars :
# Tout est auto-dérivé du groupe apt_cache_servers
```

Résultat :

- **Serveur de cache** : Synchronise depuis `https://user:pass@apt.<region>.omnitou.ch/`
- **Tous les autres hôtes** : Génèrent `deb [trusted=yes] http://192.168.1.100:8080/noble noble main` (sans identifiants)

Scénario 3 : Cache APT DISTANT NON dans le Fichier d'Hôtes (Manuel)

Le serveur de cache existe ailleurs et est déjà configuré (non géré par votre Ansible).

```yaml
all:
  vars:
    use_apt_cache: true

    # Pointer tous les hôtes vers le serveur de cache externe
    apt_repo:
      apt_server: "192.168.1.100"  # IP du serveur de cache
externe
      apt_repo_port: 8080           # Le cache fonctionne
généralement sur le port 8080

# Aucun groupe apt_cache_servers nécessaire
# Aucun remote_apt_* nécessaire (le cache est déjà configuré
externement)
```

Résultat : Tous les hôtes génèrent `deb [trusted=yes] http://192.168.1.100:8080/noble noble main` (sans identifiants)

Exemple Complet

Voici un exemple complet fonctionnel. Il montre la configuration du serveur de cache avec plusieurs hôtes d'application :

```yaml
# Groupe Serveur de Cache APT
apt_cache_servers:
  hosts:
    example-apt-cache:
      ansible_host: 10.179.1.114
      gateway: 10.179.1.1
      host_vm_network: "vmbr0"
      num_cpus: 4
      memory_mb: 16384
      proxmoxLxcDiskSizeGb: 120
  vars:
    # Le serveur de cache synchronise les paquets depuis le dépôt
authentifié amont
    remote_apt_server: "apt.<region>.omnitou.ch"
    remote_apt_port: 443
    remote_apt_protocol: "https"
    remote_apt_user: "customer-username"
    remote_apt_password: "customer-secure-token"

# Serveurs d'Application
hss:
  hosts:
    customer-hss01:
      ansible_host: 10.179.2.140
      gateway: 10.179.2.1

mme:
  hosts:
    customer-mme01:
      ansible_host: 10.179.1.15
      gateway: 10.179.1.1

dns:
  hosts:
    customer-dns01:
      ansible_host: 10.179.2.177
      gateway: 10.179.2.1

# Configuration Globale
all:
  vars:
    # Auto-configuration (aucune configuration manuelle
nécessaire) :
```

```
    # - use_apt_cache: true (auto-activé lorsque apt_cache_servers
existe)
    # - apt_repo.apt_server: "10.179.1.114" (auto-dérivé du
serveur de cache)
```

Ce qui se passe lors du déploiement :

1. **Serveur de cache (`10.179.1.114`) :**
   - Utilise `remote_apt_*` de sa section `vars:`
   - Télécharge tous les paquets depuis `https://customer-username:customer-secure-token@apt.<region>.omnitou.ch/`
   - Sert les paquets sur le port 8080 via nginx
2. **Hôtes d'application (`customer-hss01`, `customer-mme01`, `customer-dns01`) :**
   - Détectent automatiquement que le groupe `apt_cache_servers` existe
   - Définissent automatiquement `use_apt_cache: true`
   - Dérivent automatiquement `apt_repo.apt_server: "10.179.1.114"`
   - Génèrent : `deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
   - Tirent tous les paquets depuis le serveur de cache (aucuns identifiants requis)

Mise à Jour du Cache

Pour synchroniser de nouveaux paquets ou mises à jour :

```
ansible-playbook -i hosts/customer/production.yml
services/apt_cache.yml
```

Cela synchronise de manière incrémentielle tout le contenu depuis le serveur APT d'Omnitouch :

- Nouvelles versions de paquets Omnitouch

- Nouveaux paquets Ubuntu et dépendances
- Nouvelles versions GitHub
- Archives sources mises à jour

Les re-synchronisations sont incrémentielles : les paquets APT sont comparés à l'index Debian amont par SHA256 (et sont complètement ignorés si le fichier `Release` amont est inchangé), tandis que `/releases/` et `/repos/` utilisent `wget --timestamping`. Les fichiers existants inchangés sont ignorés, rendant la re-synchronisation rapide.

Remarque : Le serveur APT d'Omnitouch est la seule source de vérité pour tous les paquets. Omnitouch construit et publie des paquets vers celui-ci ; l'exécution de `services/apt_cache.yml` sur les miroirs de cache synchronise alors tout ce qui est actuellement publié.

Dépannage

La Mise à Jour APT Échoue avec 401 Non Autorisé

Symptômes :

```
Échec de la récupération
https://10.179.1.115/noble/dists/noble/main/binary-amd64/Packages
401  Non Autorisé
```

Causes possibles :

- Configuration `apt_repo` définie dans `all: vars:` au lieu de `apt_cache_servers: vars:`
- Hôtes essayant d'accéder directement au dépôt authentifié au lieu du cache
- `apt_repo_username` ou `apt_repo_password` incorrects
- IP source non sur liste blanche sur le serveur APT d'Omnitouch

- Utilisation des identifiants de cache pour un accès direct ou vice versa

Résolution :

1. **Vérifiez la portée de la configuration** : Assurez-vous que `apt_repo` avec identifiants est défini dans `apt_cache_servers: vars:`, PAS dans `all: vars:`
2. **Vérifiez le mode cache** : Lors de l'utilisation du cache, les hôtes doivent se connecter au serveur de cache (port 8080), pas au dépôt amont
3. **Vérifiez les sources générées** : Sur l'hôte en échec, vérifiez `/etc/apt/sources.list.d/omnitouch.list`
   - **Correct (mode cache)** : `deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
   - **Incorrect (a des identifiants au mauvais endroit)** : `deb [trusted=yes] https://user:pass@10.179.1.115/noble noble main`
4. Vérifiez que les identifiants sont corrects pour votre mode de déploiement
5. Confirmez que votre IP publique est sur liste blanche avec Omnitouch (si vous utilisez l'accès direct)

Les Téléchargements Binaires Échouent (Node Exporter, Zabbix, etc.)

Symptômes : Le playbook Ansible échoue à télécharger des fichiers depuis le chemin `/releases/`

Causes possibles :

- Identifiants `apt_repo` incorrects (les téléchargements binaires dérivent leur authentification de `apt_repo` en mode direct)
- Sur un serveur miroir de cache, `remote_apt_user` / `remote_apt_password` incorrects
- Mauvais schéma ou port. Rappelez-vous que le port `443` implique HTTPS et tout autre port implique HTTP.

Résolution :

1. En mode direct, vérifiez que `apt_repo.apt_repo_username` / `apt_repo.apt_repo_password` correspondent à ceux fournis par Omnitouch
2. Sur un serveur miroir de cache, vérifiez les identifiants `remote_apt_*` amont
3. Confirmez que le `apt_download_base` dérivé utilise l'hôte/schéma/port attendus (vérifiez avec `ansible-playbook ... -vvv` ou en inspectant la tâche `get_url` échouée)

Le Serveur de Cache Ne Peut Pas Se Synchroniser

Symptômes : Le playbook du serveur de cache échoue à télécharger des paquets

Causes possibles :

- Le serveur de cache n'a pas accès à Internet
- Identifiants `remote_apt_*` incorrects
- Pare-feu bloquant les connexions sortantes vers Omnitouch

Résolution :

1. Vérifiez que le serveur de cache peut atteindre le serveur APT d'Omnitouch amont sur le port 443
2. Vérifiez les identifiants `remote_apt_*`
3. Passez en revue les règles de pare-feu pour l'accès sortant

---

Documentation Connexe

- Configuration du Fichier d'Hôtes : Configuration de l'inventaire et des variables
- Référence de Configuration : Référence complète des paramètres
- Architecture de Déploiement : Architecture globale du système
- Déploiement Proxmox : Déploiement du serveur de cache en tant que conteneur LXC

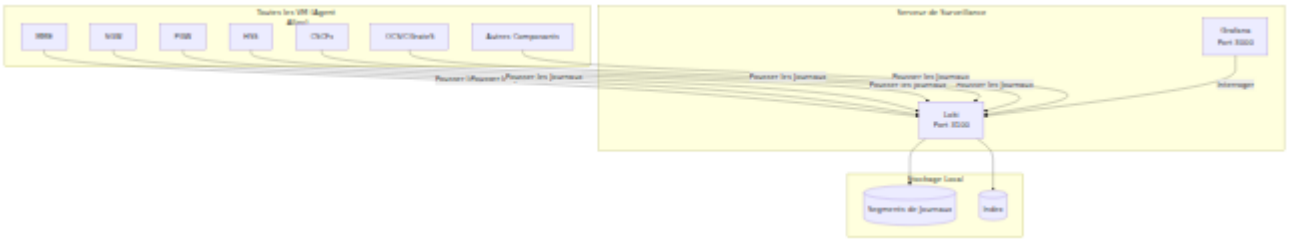
Journalisation Centralisée

Vue d'ensemble

OmniCore comprend un système de journalisation centralisé qui collecte les journaux de tous les composants et les rend consultables via Grafana. Le système utilise :

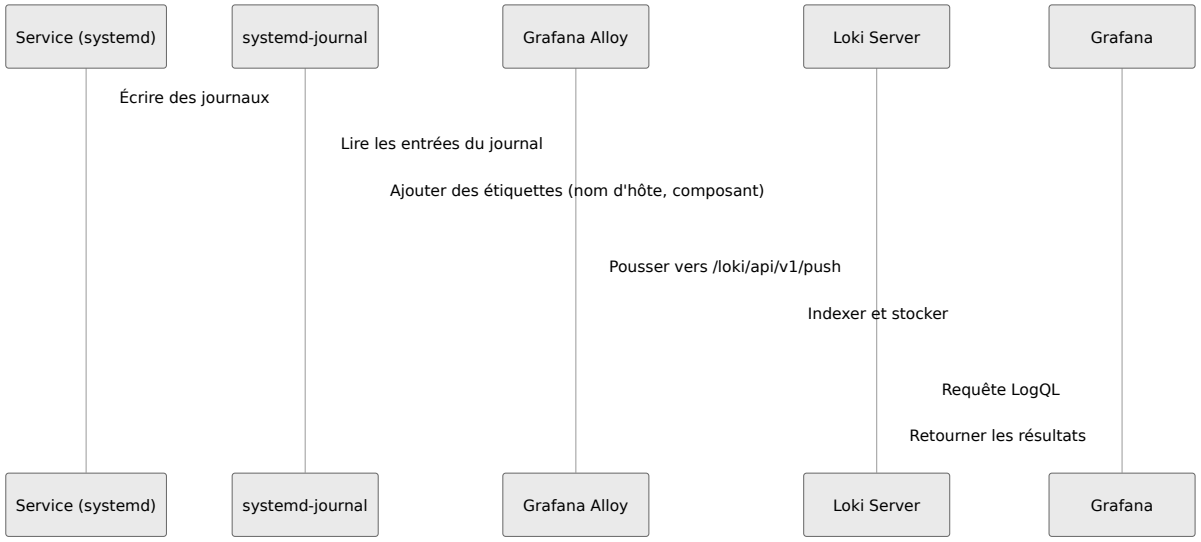
- **Grafana Alloy** : Agent de collecte de journaux déployé sur toutes les VM
- **Grafana Loki** : Serveur d'agrégation et de stockage des journaux sur les hôtes de surveillance
- **Tableaux de bord Grafana** : Tableaux de bord préconstruits pour chaque type de composant

Architecture



Comment ça fonctionne

Flux de Collecte des Journaux



Étiquettes de Journal

Chaque entrée de journal comprend ces étiquettes pour le filtrage et les requêtes :

Étiquette	Description	Exemple
`job`	Nom d'hôte de la VM source	`customer-mme01`
`hostname`	Identique à job (pour compatibilité)	`customer-mme01`
`component`	Type de composant du groupe d'inventaire	`mme`, `pcscf`, `ocs`
`unit`	Nom de l'unité systemd	`omnimme.service`
`level`	Gravité du journal	`info`, `warning`, `error`
`service`	Identifiant Syslog	`omnimme`

Configuration

Déploiement Automatique

La journalisation est automatiquement configurée lorsque :

1. Un groupe `monitoring` existe dans votre inventaire
2. Le rôle commun s'exécute sur les hôtes cibles

Aucune configuration supplémentaire n'est requise pour la fonctionnalité de base.

Exigences de l'Inventaire

```
monitoring:
  hosts:
    customer-monitoring01:
      ansible_host: 10.10.2.200
      gateway: 10.10.2.1
```

Lorsque le groupe `monitoring` est défini :

- **Hôtes de surveillance** : Exécutent le serveur Loki (reçoit et stocke les journaux)
- **Tous les autres hôtes** : Exécutent l'agent Alloy (collecte et envoie les journaux)

Configuration de Rétention

Loki est configuré avec une rétention basée sur le temps. Le compactor supprime les journaux plus anciens que la période de rétention :

Paramètre	Par défaut	Description
Rétention basée sur le temps	72 heures (3 jours)	Les journaux plus anciens que la période de rétention sont supprimés

Pour personnaliser la rétention, remplacez cette variable dans votre inventaire :

```
all:
  vars:
    loki_retention_period: "168h"  # par exemple 7 jours en heures
```

Configuration du Tampon Alloy

Alloy met en mémoire tampon les journaux localement dans un journal d'écriture anticipée (WAL) lorsque Loki est inaccessible. Les segments mis en mémoire tampon sont supprimés pour éviter une croissance indéfinie du disque :

Paramètre	Par défaut	Description
Âge maximum du segment WAL	1 heure	Les segments de journaux mis en mémoire tampon plus anciens qu'1 heure sont supprimés

Lorsque Loki est inaccessible pendant plus longtemps que l'âge du segment, les journaux mis en mémoire tampon les plus anciens sont supprimés.

Tableaux de Bord Grafana

Des tableaux de bord préconstruits sont automatiquement fournis pour chaque type de composant.

Tableaux de Bord Disponibles

Tableau de Bord	Emplacement	Description
Journaux CSCF	Journaux → Journaux CSCF	Journaux P-CSCF, S-CSCF, I-CSCF (OmniCSCF)
Journaux MME	Journaux → Journaux MME	Événements d'attachement/détachement MME et erreurs
Journaux SGW	Journaux → Journaux SGW	Événements de session et de bearer SGW
Journaux PGW	Journaux → Journaux PGW	Événements PDN et de session PGW
Journaux HSS	Journaux → Journaux HSS	Messages Diameter HSS et événements d'authentification
Journaux OmniMessage	Journaux → Journaux OmniMessage	Livraison SMS et événements SMPP
Journaux OCS/CGrateS	Journaux → Journaux OCS/CGrateS	Événements de facturation et trafic JSONRPC
Appels RPC CGrateS	Journaux → Appels RPC CGrateS	Rechercher et corréler les requêtes/réponses RPC avec latence

Fonctionnalités du Tableau de Bord

Chaque tableau de bord comprend :

- **Zone de recherche** : Recherche en texte libre à travers tous les journaux
- **Graphiques de taux de journaux** : Volume au fil du temps par hôte et gravité
- **Sections de composants** : Vues groupées (par exemple, P-CSCF, S-CSCF, I-CSCF)
- **Filtrage des erreurs** : Vues pré-filtrées pour les erreurs et les échecs
- **Suivi en direct** : Diffusion de journaux en temps réel (rafraîchissement de 10 secondes)

Utilisation des Tableaux de Bord

1. Accédez à Grafana (typiquement `http://<monitoring-host>:3000`)
2. Allez à **Tableaux de Bord** → **Journaux** → sélectionnez le composant
3. Utilisez la variable **Recherche** pour filtrer les journaux
4. Ajustez la plage horaire si nécessaire

Interrogation des Journaux

Bases de LogQL

Loki utilise LogQL pour les requêtes. Syntaxe de base :

```
{label="value"} |= "chaîne de recherche"
```

Requêtes Courantes

Tous les journaux d'un hôte spécifique :

```
{hostname="customer-mme01"}
```

Tous les journaux du composant MME :

```
{component="mme"}
```

Rechercher des erreurs à travers tous les CSCFs :

```
{component=~"pcscf|scscf|icscf"} |~ "(?i)error"
```

Filtrer par unité systemd :

```
{unit="omnicscf.service"}
```

Combiner des filtres :

```
{component="hss", hostname="customer-hss01"} |= "ULR" |=
"DIAMETER_SUCCESS"
```

Requêtes de Taux de Journaux

Volume de journaux par composant :

```
sum by (component) (rate({job=~".+"} [5m]))
```

Taux d'erreurs pour CSCF :

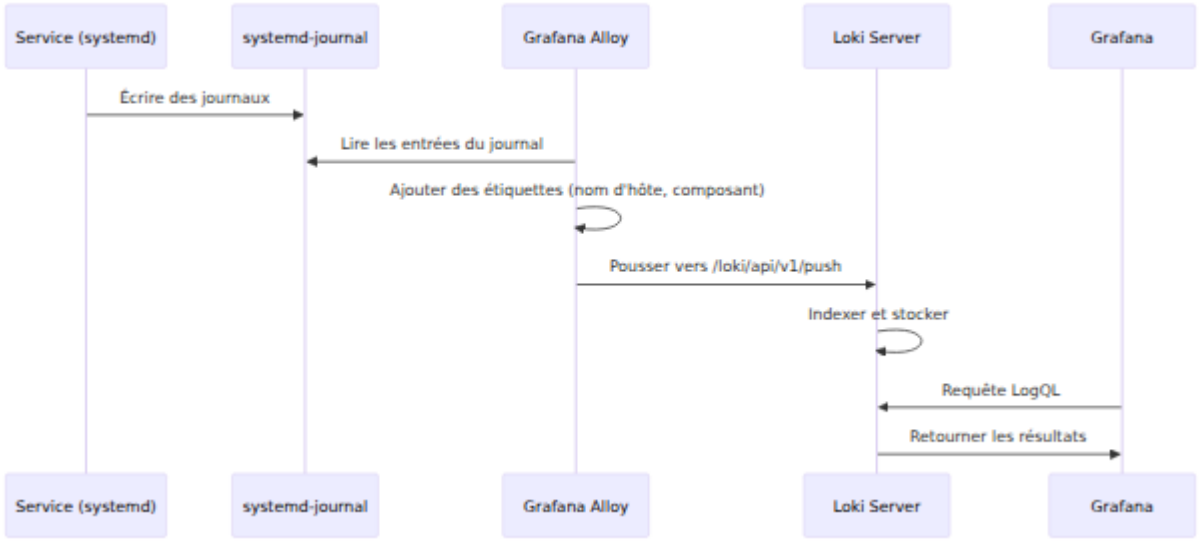
```
sum(rate({component=~"pcscf|scscf|icscf"} |~ "(?i)error" [5m]))
```

Journalisation JSONRPC CGrateS

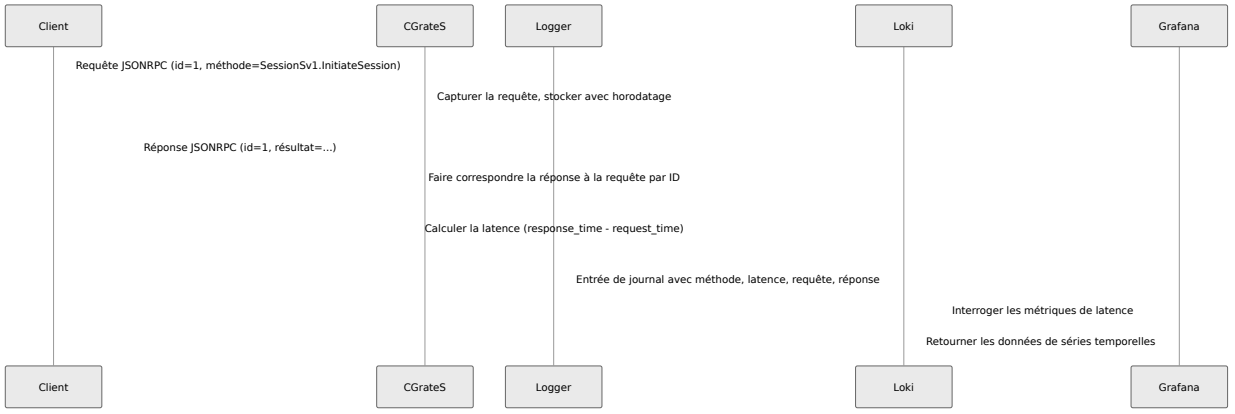
Pour les déploiements OCS/CGrateS, le trafic JSONRPC est capturé et enregistré avec corrélation des requêtes/réponses et suivi de latence pour l'analyse des performances et le débogage.

Le journaliseur JSONRPC est défini dans le rôle `cgrates` et son fichier journal (`/var/log/cgrates/jsonrpc.log`) est collecté par Alloy sur les groupes d'hôtes `ocs` et `cgrates`.

Architecture



Flux de Requête/Réponse



Ce qui est Capturé

Le trafic sur ces ports CGrateS est capturé :

Port	Nom du Service	Protocole	Description
2012	`rpc_json`	TCP	JSONRPC brut sur TCP
2080	`http`	HTTP	API HTTP (filtre `/metrics` et `/health`)

Le journaliseur filtre automatiquement :

- Les extractions de métriques Prometheus (`GET /metrics`)
- Les requêtes de vérification de santé (`GET /health`)
- Les réponses compressées en gzip (données binaires)

Format de Journal

Les journaux JSONRPC sont écrits dans `/var/log/cgrates/jsonrpc.log`. Chaque entrée de journal représente une **paire requête/réponse complétée** avec mesure de latence :

```
2026-03-01T10:30:45.123456 port=2012 service=rpc_json request_id=1
method=SessionSv1.InitiateSession latency_ms=2.45 status=ok error=
src=10.10.1.50:45678 dst=10.10.2.100:2012 request=
{"id":1,"method":"SessionSv1.InitiateSession",...} response=
{"id":1,"result":{...}}
```

Champs de Journal

Champ	Description	Exemple
`timestamp`	Horodatage ISO 8601 lorsque la réponse a été reçue	`2026-03-01T10:30:45.123456`
`port`	Port CGrateS auquel la requête a été faite	`2012`, `2080`
`service`	Nom du service pour le port	`rpc_json`, `http`
`request_id`	ID de requête JSONRPC pour corrélation	`1`, `42`, (vide pour nul)
`method`	Nom de la méthode JSONRPC	`SessionSv1.InitiateSession`
`latency_ms`	Latence aller-retour en millisecondes	`2.45`
`status`	Statut du résultat	`ok`, `error`
`error`	Message d'erreur si le statut est une erreur	`INSUFFICIENT_CREDIT`
`src`	IP:port source (client)	`10.10.1.50:45678`
`dst`	IP:port de destination (CGrateS)	`10.10.2.100:2012`
`request`	Charge utile complète de la requête JSONRPC (JSON compacté)	`{"id":1,"method":"..."}`

Champ	Description	Exemple
`response`	Charge utile complète de la réponse JSONRPC (JSON compacté)	`{"id":1,"result":{...}}`

Tableau de Bord Grafana

Le tableau de bord **Journaux OCS / CGrateS** comprend des panneaux dédiés pour l'analyse JSONRPC.

Latence JSONRPC par Méthode

Un graphique de séries temporelles montrant la latence pour chaque méthode JSONRPC au fil du temps. Utile pour identifier :

- Méthodes lentes qui peuvent nécessiter une optimisation
- Pics de latence indiquant des problèmes système
- Tendances de performance au fil du temps

La légende affiche la latence moyenne et maximale par méthode.

Trafic JSONRPC

Un panneau de journaux montrant toutes les paires requête/réponse JSONRPC avec :

- Horodatage
- Nom de la méthode
- Latence
- Charges utiles complètes de requête et de réponse (formatées)

Analyse par Méthode

Filtrer le trafic JSONRPC par une méthode spécifique en utilisant la variable **Méthode** en haut du tableau de bord. Entrez le nom de la méthode (par exemple, `SessionSv1.InitiateSession`) pour voir uniquement le trafic pour cette méthode.

Tableau de Bord des Appels RPC CGrateS

Le tableau de bord dédié **Appels RPC CGrateS** fournit des outils ciblés pour le dépannage des appels API et la corrélation des requêtes avec les réponses.

Cas d'utilisation : Trouver un Appel Échoué

Lorsqu'un utilisateur signale "J'ai essayé d'appeler 1234 et cela a échoué" :

1. Ouvrez **Tableaux de Bord → Journaux → Appels RPC CGrateS**
2. Dans le champ **Recherche**, entrez le numéro de téléphone ou l'ID de compte : `1234`
3. Réglez **Statut** sur **Erreur** pour voir uniquement les appels échoués
4. Le panneau **Recherche d'Appel RPC** montre tous les appels correspondants avec requête/réponse complètes

Chaque entrée de journal montre la charge utile complète de la requête (ce qui a été envoyé) et la charge utile de réponse (ce que CGrateS a retourné), facilitant l'identification de l'erreur exacte.

Variables du Tableau de Bord

Variable	Objectif	Exemple
Recherche	Recherche en texte libre dans les charges utiles de requête/réponse	`1234`, `Account123`, `INSUFFICIENT_CREDIT`
Méthode	Filtrer par nom de méthode RPC	`SessionSv1.InitiateSession`, `CDRsV1.ProcessEvent`
Statut	Filtrer par statut de résultat	Tous, Succès, Erreur

Panneaux du Tableau de Bord

Statistiques Résumées (ligne supérieure) :

- **Total des Appels RPC** : Nombre total d'appels dans la plage horaire
- **Erreurs** : Nombre d'appels échoués (codage couleur : vert=0, jaune=1-9, rouge=10+)
- **Latence Moyenne** : Temps moyen aller-retour (seuils codés par couleur)
- **Latence Maximale** : Appel ayant la latence la plus élevée

Latence RPC par Méthode : Séries temporelles montrant la latence pour chaque méthode. Survolez pour voir des valeurs spécifiques.

Recherche d'Appel RPC : Panneau de journal principal montrant les appels correspondants. Cliquez sur une entrée pour développer et voir le JSON complet de requête/réponse.

Appels RPC Échoués : Vue pré-filtrée montrant uniquement les appels `status=error`.

Répartition par Méthode : Graphiques à secteurs montrant la répartition des appels et la répartition des erreurs par méthode.

Interrogation des Journaux JSONRPC

Requêtes de Base

Tout le trafic JSONRPC :

```
{job="cgrates-jsonrpc"}
```

Filtrer par méthode :

```
{job="cgrates-jsonrpc"} |= "method=SessionSv1.InitiateSession"
```

Rechercher dans les charges utiles de requête/réponse :

```
{job="cgrates-jsonrpc"} |= "Account\":\"12345"
```

Erreurs uniquement :

```
{job="cgrates-jsonrpc"} |= "status=error"
```

Analyse de Latence

Extraire la latence en tant que métrique (pour les graphiques) :

```
max_over_time(
  {job="cgrates-jsonrpc"}
  |= "method="
  | regexp "method=(?P<method>[^ ]+) latency_ms=(?P<latency>[0-
9.]+)"
  | __error__=""
  | unwrap latency [$__auto]
) by (method)
```

Trouver des requêtes lentes (latence > 100ms) :

```
{job="cgrates-jsonrpc"}
| regexp "latency_ms=(?P<latency>[0-9.]+)"
| latency > 100
```

Requêtes Spécifiques par Méthode

Traitement des CDR :

```
{job="cgrates-jsonrpc"} |= "method=CDRsV1"
```

Gestion de session :

```
{job="cgrates-jsonrpc"} |~ "method=SessionSv1"
```

Opérations de compte :

```
{job="cgrates-jsonrpc"} |~ "method=ApierV"
```

Gestion du Service

Le journaliseur JSONRPC fonctionne en tant que service systemd sur les hôtes OCS/CGrateS.

Vérifier l'état du service :

```
systemctl status cgrates-jsonrpc-logger
```

Voir les journaux du service :

```
journalctl -u cgrates-jsonrpc-logger -f
```

Redémarrer le service :

```
systemctl restart cgrates-jsonrpc-logger
```

Dépannage

Aucun Journal JSONRPC N'apparaît

Symptômes : Le panneau Trafic JSONRPC ne montre aucune donnée

Causes possibles :

- Service de journaliseur non en cours d'exécution
- ngrep non installé
- Mismatch de l'interface réseau
- Alloy ne collecte pas le fichier journal

Résolution :

1. Vérifiez l'état du service de journaliseur :

   ```
   systemctl status cgrates-jsonrpc-logger
   journalctl -u cgrates-jsonrpc-logger -n 50
   ```

2. Vérifiez que ngrep est installé :

```
which ngrep
```

3. Vérifiez que le fichier journal est en cours d'écriture :

```
tail -f /var/log/cgrates/jsonrpc.log
```

4. Vérifiez qu'Alloy collecte le fichier :

```
journalctl -u alloy | grep jsonrpc
```

Le Graphique de Latence Ne Montre Aucune Donnée

Symptômes : Le panneau Latence JSONRPC par Méthode montre "Aucune donnée"

Causes possibles :

- Aucun trafic JSONRPC dans la plage horaire sélectionnée
- Mismatch de format de journal (regex non correspondant)

Résolution :

1. Vérifiez que des journaux existent pour la plage horaire :

```
{job="cgrates-jsonrpc"} |= "method="
```

2. Vérifiez que le champ `latency_ms` est présent dans les journaux
3. Élargissez la plage horaire

Compte de Requêtes En Attente Élevé

Symptômes : Les journaux de débogage montrent de nombreuses requêtes stockées mais peu de complétions enregistrées

Causes possibles :

- Requêtes sans réponses (client déconnecté)
- Réponse capturée avant la requête (ordre des paquets)

- Mismatch d'ID entre la requête et la réponse

Résolution :

1. Le journaliseur nettoie automatiquement les requêtes en attente de plus de 60 secondes
2. Vérifiez les problèmes de réseau entre le client et CGrateS
3. Vérifiez que CGrateS répond aux requêtes

Journaux de Fichiers

En plus des journaux du journal systemd, Alloy collecte des fichiers journaux spécifiques :

Journaux FreeSWITCH (Serveurs d'Applications)

Chemin	Étiquette de Job
`/var/log/freeswitch/freeswitch.log`	`freeswitch`
`/var/log/omnitas-freeswitch.log`	`freeswitch-structured`

Journaux Nginx (OmniCRM)

Chemin	Étiquette de Job	Étiquette de Type
`/var/log/nginx/access.log`	`nginx`	`access`
`/var/log/nginx/error.log`	`nginx`	`error`

JSONRPC CGrateS (OCS)

Chemin	Étiquette de Job
`/var/log/cgrates/jsonrpc.log`	`cgrates-jsonrpc`

Dépannage

Journaux Non Apparents dans Grafana

Symptômes : Aucun journal visible dans les tableaux de bord Loki

Causes possibles :

- Service Alloy non en cours d'exécution sur l'hôte source
- Service Loki non en cours d'exécution sur l'hôte de surveillance
- Connectivité réseau entre les hôtes bloquée
- Alloy ne peut pas atteindre Loki sur le port 3100

Résolution :

1. Vérifiez l'état d'Alloy sur l'hôte source :

```
systemctl status alloy
journalctl -u alloy -f
```

2. Vérifiez l'état de Loki sur l'hôte de surveillance :

```
systemctl status loki
journalctl -u loki -f
```

3. Testez la connectivité de l'hôte source à l'hôte de surveillance :

```
curl http://<monitoring-ip>:3100/ready
```

4. Vérifiez que le pare-feu autorise le TCP 3100 de tous les hôtes vers la surveillance

Utilisation Élevée du Disque par Alloy

Symptômes : `/var/lib/alloy` consomme un espace disque significatif

Causes possibles :

- Loki inaccessible pendant une période prolongée
- Tampon WAL se remplissant

Résolution :

1. Vérifiez la connectivité à Loki (voir ci-dessus)
2. Si Loki est hors service, les journaux sont mis en mémoire tampon localement dans le WAL
3. Une fois que Loki est accessible, les journaux mis en mémoire tampon sont envoyés automatiquement
4. Les segments mis en mémoire tampon plus anciens qu'1 heure sont supprimés (par conception)

Stockage Loki Plein

Symptômes : Loki cesse d'accepter des journaux, disque plein sur le serveur de surveillance

Causes possibles :

- Volume de journaux dépassant l'espace disque disponible avant que la rétention ne supprime les anciens journaux
- Compaction de rétention non exécutée

Résolution :

1. Vérifiez l'utilisation du stockage de Loki :

```
du -sh /var/lib/loki/
```

2. Vérifiez que le compactor fonctionne (vérifiez les journaux de Loki)
3. Déclenchez manuellement la compaction si nécessaire en redémarrant Loki
4. Envisagez d'augmenter l'allocation de disque ou de réduire la période de rétention

Étiquettes de Composant Manquantes

Symptômes : Les journaux apparaissent mais l'étiquette `component` est `infrastructure` au lieu de la valeur attendue

Causes possibles :

- Hôte non dans le groupe d'inventaire attendu
- Configuration d'Alloy non régénérée après un changement d'inventaire

Résolution :

1. Vérifiez que l'hôte est dans le bon groupe d'inventaire
2. Réexécutez Ansible pour régénérer la configuration d'Alloy :

```
ansible-playbook -i hosts/customer/hosts.yml services/all.yml --limit <hostname>
```

3. Redémarrez Alloy :

```
systemctl restart alloy
```

Ports de Service

Service	Port	Protocole	Description
Loki	3100	HTTP	Ingestion de journaux et API de requête
Loki	9096	gRPC	gRPC interne (non utilisé à l'extérieur)
Alloy	12345	HTTP	Métriques et UI d'Alloy
Grafana	3000	HTTP	Accès au tableau de bord

Documentation Connexe

- Surveillance & Observabilité : Grafana, Prometheus, tableaux de bord et alertes
- Architecture de Déploiement : Architecture système globale
- Configuration du Fichier d'Hôtes : Configuration de l'inventaire
- Référence de Configuration : Référence complète des paramètres

Noyau Commuté par Circuit (CS)

Le domaine commuté par circuit hérité (voix et SMS 2G/3G) est déployé par le playbook `services/cs.yml`. Il provisionne le BSC orienté radio, le commutateur logiciel OmniMSC et la pile de signalisation SS7/SIGTRAN.

Aperçu

`cs.yml` exécute trois rôles, chacun contre son propre groupe d'inventaire :

Composant	Rôle	Groupe d'inventaire	Paquets	De
BSC	`circuit_switched/bsc`	`bsc`	BSC, MGW	Co Sta Bas Pas Mu (pc tes val 2G
MSC	`circuit_switched/omnimsc`	`msc`	`omnimsc`	Co Mo Sw Ce
SS7	`omniss7`	`ss7`	`omniss7`	Pile SS (ST SM CA tes

```
- name: Setup MSC
  hosts: msc
  roles:
    - {role: circuit_switched/omnimsc, become: yes}
```

BSC (`circuit_switched/bsc`)

Installe un **BSC** et une **Passerelle Multimédia (MGW)** depuis le serveur Omnitouch APT pour les tests et la validation commutés par circuit. Le rôle configure les deux services.

- Génère les fichiers de configuration BSC et MGW à partir des modèles du rôle.
- Installe un remplacement systemd pour le service MGW qui **désactive la planification CPU en temps réel**. Il efface `CPUSchedulingPolicy`, `Priority` et `AmbientCapabilities`. Les machines virtuelles ne prennent pas en charge la planification en temps réel.
- Les gestionnaires redémarrent chaque service. Chaque gestionnaire **vérifie ensuite que le service atteint `active`** (6 tentatives, 5 s d'intervalle) avant de continuer.

MSC (`circuit_switched/omnimsc`)

Installe le paquet `omnimsc` et le configure de manière similaire à un NF 5GC :

- Résout `group_vars_omnicore` et dérive `license_server_api_urls` à partir du groupe `license_server`.
- Génère le `runtime.exs.j2` intégré vers `/etc/omnimsc/`. Si `omnimsc_template_config` est défini, il génère un modèle personnalisé à partir de `group_vars/` à la place.
- S'assure d'un répertoire de sortie CDR (`omnimsc_cdr_output_dir`, par défaut `/var/lib/omnimsc/cdr`).
- Génère un certificat TLS auto-signé pour l'interface web / API.
- Utilise le modèle de lien symbolique versionné + `runtime.exs`, redémarrant en cas de changement.

SS7 (`omniss7`)

Un seul rôle qui installe le paquet `omniss7` et génère **un modèle de configuration par rôle SS7**, sélectionné par la variable d'hôte `ss7_role` :

`ss7_role`	Modèle	Fonction
`stp`	`stp.j2`	Point de Transfert de Signal (routage SS7)
`hlr`	`hlr.j2`	Registre de Localisation à Domicile
`smsc`	`ipsmgw.j2`	SMSC / IP-SM-GW
`camel`	`camel.j2`	Contrôle de service CAMEL
`tester`	`tester.j2`	Agent de test/charge de signalisation

Sélecteurs supplémentaires :

- La définition de `ss7_protocol: m2pa` génère `m2pa.j2` (un point de terminaison M2PA) au lieu du modèle `tester`.
- La définition de `ss7_template_config` (un chemin sous `group_vars/`) remplace **tous** les modèles intégrés par une configuration spécifique à l'hôte.

Comme les autres composants Elixir, `omniss7` utilise le modèle de lien symbolique versionné + `runtime.exs` et reflète le lien symbolique dans `/opt/omniss7/releases/*`.

Documentation Connexe

- Playbooks de Service - Tous les playbooks de service et la hiérarchie de déploiement
- Noyau 5G - Le noyau autonome 5G et le rôle partagé `5gc_nf`
- Configuration des Variables de Groupe - Modèles de configuration personnalisés (par exemple, `ss7_template_config`, `omnimsc_template_config`)
- **Configuration d'OmniCore**: https://docs.omnitouch.com.au/docs/repos/OmniCore - Ce qui va dans les fichiers de configuration des composants

Référence de Configuration

Vue d'ensemble

Ce document fournit une référence complète pour la configuration des déploiements OmniCore via des fichiers d'hôtes. La configuration est principalement définie dans des fichiers d'inventaire d'hôtes avec des remplacements de `group_vars` minimaux nécessaires pour les déploiements modernes.

Pour la documentation spécifique au produit, voir :

- **OmniCore** : https://docs.omnitouch.com.au/docs/repos/OmniCore
- **OmniCall** : https://docs.omnitouch.com.au/docs/repos/OmniCall
- **OmniCharge** : https://docs.omnitouch.com.au/docs/repos/OmniCharge

Approche de Configuration

Les déploiements modernes d'OmniCore utilisent un modèle de configuration simplifié :

Principe Clé : La plupart des configurations sont définies directement dans le fichier d'hôtes. Les valeurs par défaut des rôles gèrent de nombreux paramètres. Cependant, certains composants sont configurés exclusivement à partir de `group_vars` et n'ont pas de valeur par défaut de rôle. OmniCRM est l'exemple notable (voir ci-dessous). Dans ces cas, une valeur manquante échoue l'exécution de manière bruyante plutôt que de revenir à une valeur par défaut.

> **Note de sécurité** : Plusieurs valeurs par défaut de rôle sont des espaces réservés non sécurisés destinés à un usage en laboratoire uniquement et **doivent** être remplacées pour tout déploiement réel. Les exemples incluent les mots de passe de base de données (par exemple, `sql_pass: password`, `omnitouch2024`) et les identifiants de dépôt APT (`omni`/`omni`). Toujours définir de véritables secrets uniques dans votre inventaire ou `group_vars` avant de déployer.

Planification Réseau

Avant de configurer les hôtes, consultez la Norme de Planification IP pour des conseils sur :

- Stratégies de segmentation réseau
- Allocation d'adresses IP
- Organisation de sous-réseaux
- Gestion des IP publiques

Paramètres Communs des Hôtes

#ToDo - Il suffit de dire de vérifier hosts-file-configuration.md pour cela

Drapeaux Spécifiques au Service

```
cdrs_enabled: True                  # Activer la génération de CDR
in_pool: False                      # Exclure du pool d'équilibrage
de charge
populate_crm: False                 # Peupler le CRM avec des données
initiales
```

Variables Globales (all:vars)

La section `all:vars` contient des paramètres à l'échelle du déploiement. Les déploiements modernes utilisent des variables globales minimales, avec de nombreux paramètres couverts par des valeurs par défaut de rôle. Notez les exceptions ci-dessous (OmniCRM, Grafana SMTP) où la configuration doit être fournie explicitement car aucune valeur par défaut de rôle n'existe.

Variables Globales Essentielles

Authentification & Accès

```
ansible_connection: ssh
ansible_user: root
ansible_password: password
ansible_become_password: password
```

Alternative : Utilisez des clés SSH au lieu de mots de passe :

```
ansible_ssh_private_key_file: '/path/to/key.pem'
```

Identité du Client

```
customer_name_short: omnitouch
customer_legal_name: "Customer Lab"
site_name: Customer-Site
region: AU
TZ: Australia/Melbourne
```

Configuration PLMN

```yaml
plmn_id:
  mcc: '001'               # Code de Pays Mobile (3 chiffres)
  mnc: '01'               # Code de Réseau Mobile (2-3 chiffres)
  mnc_longform: '001'     # MNC avec zéros en tête (toujours 3
chiffres)

diameter_realm: epc.mnc{{ plmn_id.mnc_longform }}.mcc{{
plmn_id.mcc }}.3gppnetwork.org
```

But : Identifie de manière unique votre réseau mobile. Utilisé pour la construction du domaine Diameter.

Noms de Réseau

```yaml
network_name_short: Omni
network_name_long: Omnitouch
tac_list: [10100,100]           # Liste TAC par défaut (peut être
remplacée par MME)
```

Affiché : Noms de réseau affichés sur les appareils UE dans Paramètres > Réseau Mobile.

Configuration DNS

```yaml
manage_resolv_conf: True         # Définir sur False pour empêcher
Ansible de gérer /etc/resolv.conf
```

Remarque : Lorsque `manage_resolv_conf` est défini sur `False`, Ansible ne remplacera pas `/etc/resolv.conf` sur cet hôte. Cela est utile pour les hôtes qui nécessitent une configuration DNS personnalisée ou qui sont gérés par des systèmes externes. Peut être défini par hôte dans l'inventaire ou globalement dans `all:vars`.

Configuration du Dépôt APT

`apt_repo` est la seule source de vérité. Définissez-le une fois. Les variables de téléchargement binaire (`remote_apt_*`) et l'URL `apt_download_base` en sont

dérivées automatiquement. Vous ne les définissez pas séparément dans un inventaire normal.

Valeurs par défaut automatiques : Lorsqu'un groupe `apt_cache_servers` est défini avec des hôtes :

- `use_apt_cache` par défaut à `True` (à moins d'être explicitement défini sur `False`)
- `apt_repo.apt_server` par défaut au premier IP du serveur de cache

```
use_apt_cache: False              # Accès direct au dépôt (True =
cache APT local)

apt_repo:
  apt_server: "apt.<region>.omnitou.ch"  # FQDN fourni par
Omnitouch (ou IP de cache)
  apt_repo_username: "your-username"     # Nécessaire pour un
accès direct
  apt_repo_password: "your-password"     # Nécessaire pour un
accès direct
  # apt_repo_port: 443                   # Optionnel ; par défaut
à 443 (HTTPS).
                                         # Le schéma suit le port
: 443 -> https, sinon http.
```

Le schéma et le port par défaut sont **HTTPS / 443** pour un accès direct et **HTTP / 8080** en mode cache. Seul un serveur miroir de cache (`apt_cache_servers` groupe) définit `remote_apt_*` manuellement, pour décrire le serveur Omnitouch en amont à partir duquel il se synchronise.

Voir : Système de Cache APT

Serveur de Licences

```
# Optionnel : dérivé automatiquement du groupe d'inventaire
`license_server` lorsqu'il n'est pas défini.
# Chaque hôte de ce groupe devient
https://<ansible_host>:8443/api.
license_server_api_urls: ["https://10.10.2.150:8443/api"]
license_enforced: false          # La valeur par défaut effective
est false (roles/omnihss/defaults/main.yml)
```

Remarque : Si un groupe `license_server` existe dans l'inventaire, `license_server_api_urls` est dérivé automatiquement de ses membres ; vous devez seulement le définir manuellement pour remplacer ce comportement.

Voir : Serveur de Licences

Paramètres MME

Le MME sélectionne les pairs S-GW et P-GW en utilisant une méthode configurable, définie par passerelle sous `omnimme`. La méthode peut être une seule valeur ou divisée par trafic `home`/`roaming`. Les valeurs valides sont `dns_lookup` (résoudre les pairs via DNS) et `random_peer` (choisir parmi les pairs configurés).

```
omnimme:
  sgw:
    selection_method: random_peer        # valeur unique, ou...
    # selection_method:
    #   home: random_peer
    #   roaming: dns_lookup
  pgw:
    selection_method:
      home: random_peer
      roaming: dns_lookup
```

Paramètres AMF

```
# tac_list entraîne également la liste des ta pris en charge par
l'AMF. Chaque TAC ici est
# annoncé aux gNB connectés lors de la configuration NG. Si non
défini, l'AMF par défaut
# au TAC 1 uniquement, ce qui rejette tout gNB ne diffusant pas le
TAC 1.
tac_list: [1, 3]

# Désactiver le chiffrement NAS (forcer NEA0). Par défaut à vrai
car omniamf
# >=20260727 négocie NEA2 mais ne peut pas décoder le NAS uplink
chiffré (Mode de Sécurité
# Complet et messages suivants), provoquant l'échec de chaque
enregistrement 5G.
# Définir sur faux une fois que le fournisseur confirme que le
déchiffrement uplink NEA2 fonctionne
# contre de vrais UE. L'intégrité NAS (NIA2) n'est pas affectée.
amf_force_null_ciphering: true
```

Paramètres SAEGW

```
mtu: 1400                       # Unité de Transmission Maximale
```

Paramètres OmniHSS

OmniHSS construit automatiquement sa liste de pairs Diameter à partir de l'inventaire. Il utilise les groupes `dra`, `mme`, `pgwc`, `scscf`, `icscf`, `pcscf` et `applicationserver` (les groupes utilisés dépendent de `packet_core_dra_support` / `ims_dra_support`). `exclude_diameter_peers` exclut des hôtes individuels de cette liste générée, pour les pairs qui sont dans l'inventaire mais ne devraient pas être des pairs Diameter de l'HSS.

```
omnihss:
  exclude_diameter_peers:          # optionnel ; par défaut à []
(rien exclu)
    - customer-as01
    - customer-as02
```

- Les entrées sont comparées à **`inventory_hostname`**, et non au FQDN ou à l'IP Diameter.
- L'exclusion s'applique à chaque groupe découvert automatiquement, donc une liste couvre un hôte peu importe dans quel groupe il se trouve.
- Les noms doivent exister dans l'inventaire. Le rôle vérifie cela avant le déploiement, donc une faute de frappe échoue l'exécution de manière bruyante plutôt que de laisser silencieusement le pair dans `/etc/omnihss/runtime.exs`.
- Définissez-le là où le reste du dictionnaire `omnihss` est défini (par hôte, ou sous `vars:` d'un groupe). Comme Ansible remplace plutôt que fusionne les dictionnaires, il doit se trouver dans le même bloc `omnihss` que les autres clés.
- Cela n'affecte que la liste des pairs Diameter. La liste des pairs SIP S-CSCF sous `config :hss, ims:` est sans rapport et n'est pas filtrée par cette clé.

> Le DRA a son propre mécanisme séparé : définir `dra_peer_exclude: True` sur un hôte l'exclut de la liste des pairs OmniDRA. Les deux sont indépendants. Exclure un hôte de l'HSS ne l'exclut pas du DRA, et vice versa.

Paramètres OmniCRM

La configuration d'OmniCRM est sourcée **exclusivement** à partir des `group_vars` du client (sous `hosts/`); aucune valeur par défaut n'est intégrée dans le rôle pour les secrets de site.

Chemin préféré (`omnicrm_deploy_mode: package`, la valeur par défaut du rôle) : installer les paquets génériques `omnicrm-api` / `omnicrm-ui` à partir de `apt_repo`, modéliser `crm_config.yaml` vers `/etc/omnicrm/crm_config.yaml`, synchroniser `deployment_assets/`, et `systemctl restart omnicrm-api`. Le branding UI provient du bloc `ui:` (runtime `GET /crm/public/ui-config`). Ce chemin nécessite **aucun** build yarn sur l'hôte et **aucun** `crm_env_override`.

Chemin hérité (`omnicrm_deploy_mode: source`) : archive git + venv hôte + build yarn ; `crm_env_override` est requis uniquement pour ce mode.

```
omnicrm_deploy_mode: package     # package (par défaut) | source
omnicrm_package_version: ""      # version optionnelle de pin apt
pour omnicrm-api/ui
crm_config.yaml                  # API + ui: config (group_vars ;
requis)
deployment_assets/               # logos/icônes optionnels pour
/etc/omnicrm/deployment_assets
crm_env_override: my-site.env    # requis uniquement pour le mode
source / fallback UI local
cell_sites: cellSites.txt        # optionnel : données sur les
sites cellulaires pour l'UI
message_templates: constants.js  # optionnel : modèles de messages
de diffusion cellulaire
site_data: site_data.json        # optionnel : limites
géographiques de diffusion cellulaire
```

Paramètres SMTP Grafana

Utilisés par le rôle de surveillance pour les e-mails d'alerte Grafana. `grafana_smtp_user` et `grafana_smtp_password` sont **requises** (pas de valeur par défaut : vous devez les définir) ; `grafana_smtp_host` par défaut à `in-v3.mailjet.com:587`. Voir `roles/monitoring/templates/grafana/grafana.ini`.

```
grafana_smtp_user: "smtp-user"           # REQUIS, pas de valeur
par défaut
grafana_smtp_password: "smtp-password"   # REQUIS, pas de valeur
par défaut
grafana_smtp_host: "in-v3.mailjet.com:587"  # optionnel, c'est la
valeur par défaut
```

Paramètres IMS

```
ims_dra_support: False           # Acheminer l'IMS via DRA
enable_homer: False              # Activer la capture SIP Homer
```

Configuration du Moniteur RAN

```yaml
use_nokia_monitor: True
use_casa_monitor: True
install_influxdb: True

influxdb_user: monitor
influxdb_password: "secure-password"
influxdb_organisation_name: omnitouch
influxdb_nokia_bucket_name: nokia-monitor
influxdb_casa_bucket_name: casa-monitor
influxdb_operator_token: "generated-token"
influxdb_url: http://127.0.0.1:8086

enable_pm_collection: False
enable_alarm_collection: False
enable_location_collection: False
enable_ran_status_collection: True
enable_nokia_rectifier_collection: False
collection_interval_in_seconds: 120

ran_monitor:
  sql:
    user: ran_monitor
    password: "secure-password"
    database_host: 127.0.0.1
    database_name: ran_monitor
  influxdb:
    address: 10.10.2.135
    port: 8086
  nokia:
    airscales:
    - address: 10.7.15.66
      name: site-Lab-Airscale
      port: 8080
      web_password: nemuuser
      web_username: Nemuadmin
```

Configuration du Pare-feu

```yaml
firewall:
  allowed_ssh_subnets:
    - '10.0.1.0/24'
    - '10.0.0.0/24'
  allowed_ue_voice_subnets:
    - '10.0.1.0/24'
  allowed_carrier_voice_subnets:
    - '10.0.1.0/24'
  allowed_signaling_subnets:
    - '10.0.1.0/24'
```

Serveurs DNS de Roaming

```yaml
roaming_dns_servers:
  wildcard: ['10.0.99.1']
  # DNS spécifiques aux opérateurs (basés sur PLMN)
  123456:                         # Exemple Opérateur 1
    - '10.10.2.197'
  654321:                         # Exemple Opérateur 2
    - '10.10.0.4'
```

Utilisateurs Locaux (Clés SSH)

```yaml
local_users:
  usera:
    name: Exemple Utilisateur A
    public_key: "ssh-rsa AAAAB3Nza..."
  userb:
    name: Exemple Utilisateur B
    public_key: "ssh-ed25519 AAAAC3..."
```

Configuration de l'Hyperviseur

Proxmox

Un site se déploie soit en tant que VMs soit en tant que conteneurs LXC, sélectionnés par entrée `proxmoxServers` via `deployment_type` (valeur par défaut `vm`). Toutes les entrées d'un site doivent être d'accord ; le mélange échoue la validation. Voir Déploiement Proxmox pour le guide complet.

Site VM

```
proxmoxServers:
  customer-prxmx01:
    # deployment_type omis → par défaut à "vm"
    proxmoxServerAddress: 10.10.0.100
    proxmoxServerPort: 8006
    proxmoxApiTokenName: AnsibleToken
    proxmoxApiTokenSecret: "token-secret"
    proxmoxNodeName: pve01
    proxmoxTemplateName: ubuntu-24.04-cloud-init-template
    proxmoxTemplateId: 9000
    proxmoxTemplateUser: omnitouch        # nom d'utilisateur
cloud-init optionnel ; par défaut au premier clé local_users
    proxmoxTemplatePassword: omnitouch    # mot de passe cloud-
init optionnel ; par défaut au nom d'utilisateur cloud-init
    storage: SSD_RAID0            # optionnel, stockage VM par
défaut
```

Site LXC

```yaml
proxmox_lxc_nameserver: "1.1.1.1"  # optionnel, injecté dans les
LXC à la création

proxmoxServers:
  customer-prxmx01:
    deployment_type: lxc
    proxmoxServerAddress: 10.10.0.100
    proxmoxServerPort: 8006
    proxmoxApiTokenName: AnsibleToken
    proxmoxApiTokenSecret: "token-secret"
    proxmoxNodeName: pve01
    proxmoxLxcOsTemplate: "local:vztmpl/ubuntu-24.04-
standard_24.04-2_amd64.tar.zst"
    proxmoxLxcDefaultStorage: SSD_RAID0   # optionnel, fallback
rootfs
```

Remplacements au niveau du groupe (les deux types)

```yaml
dns:
  vars:
    proxmox_interface: vmbr0              # requis : pont
    gateway: 10.10.0.1                    # requis
    netmask: 255.255.255.0                # requis
    vlanid: 100                           # optionnel
    proxmoxLxcCores: 2                    # optionnel (LXC
uniquement)
    proxmoxLxcMemoryMb: 4096              # optionnel (LXC
uniquement)
    proxmoxLxcDiskSizeGb: 30              # optionnel (LXC
uniquement)
    proxmoxLxcRootFsStorageName: SSD_RAID0 # optionnel (LXC
uniquement)
    host_vm_network: vmbr1                # remplacement de pont
optionnel
```

VMware vCenter

```yaml
vcenter_ip: "vcenter.example.com"
vcenter_username: "administrator@vsphere.local"
vcenter_password: "password"
vcenter_datacenter: "DC1"
vcenter_vm_template: ubuntu-24.04-model
vcenter_vm_disk_size: 50
vcenter_folder: "Omnicore"
host_vm_network: "Management"

vhosts:
  "10.0.0.23":
    vcenter_cluster_ip: 10.0.0.23
    vcenter_datastore: "datastore1 (3)"

netmask: 255.255.255.0
```

Documentation Connexe

- Norme de Planification IP - Architecture réseau et directives d'allocation IP
- Configuration du Fichier d'Hôtes - Comment structurer les fichiers d'hôtes
- Configuration des Variables de Groupe - Quand et comment utiliser group_vars
- Configuration de Netplan - IP secondaires et configuration multi-NIC
- Architecture de Déploiement - Comment les composants interagissent
- Système de Cache APT - Gestion des paquets
- Serveur de Licences - Configuration des licences

Documentation Produit

Pour des guides opérationnels détaillés et une configuration avancée :

- **Composants OmniCore** : https://docs.omnitouch.com.au/docs/repos/OmniCore

- **Composants OmniCall** : https://docs.omnitouch.com.au/docs/repos/OmniCall
- **OmniCharge/OmniCRM** : https://docs.omnitouch.com.au/docs/repos/OmniCharge

Vue d'ensemble de l'architecture de déploiement

Vue d'ensemble

Ce document fournit une vue complète de la façon dont le logiciel de réseau cellulaire des services Omnitouch est déployé à l'aide d'Ansible, montrant comment tous les composants s'assemblent pour créer un réseau 4G/5G fonctionnel.

Consultez la Norme de planification IP pour des informations détaillées sur le placement des composants, les directives d'attribution d'adresses IP et la gestion des IP publiques.

Exemple complet de déploiement

0. Provisionnement de l'infrastructure (Optionnel)

Pour les déploiements Proxmox, provisionnez les VMs/LXCs avant la configuration :

```
# Déployer des VMs et/ou des conteneurs LXC sur Proxmox (le
type_de_déploiement par hôte décide lequel)
ansible-playbook -i hosts/Customer/hosts.yml
util_playbooks/proxmox.yml
```

Voir : Déploiement de VM/LXC Proxmox

1. Définition de l'infrastructure (Fichier Hosts)

```yaml
# D��finir quoi déployer et où
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15

hss:
  hosts:
    customer-hss01:
      ansible_host: 10.10.2.140

# ... tous les autres composants
```

Voir : Configuration du fichier Hosts

2. Personnalisation (group_vars)

Le dossier `group_vars` stocke toutes les configurations de remplacement nécessaires au niveau d'un hôte, d'un site ou d'un réseau.

Par exemple, un dossier contient votre configuration SMSc OmniMessage. Les trunks SIP auxquels votre TAS se connecte se trouvent ici. Toute votre logique de routage Diameter se trouve ici aussi.

OmniCRM est également configuré ici. Ses paramètres sont uniquement `group_vars`. Son code source GitHub est cloné sur le contrôleur Ansible (`delegate_to: localhost`) avec les propres identifiants du contrôleur avant d'être synchronisé avec la cible, de sorte qu'aucune information d'identification n'atterrisse jamais sur l'hôte cible.

Voir : Configuration des variables de groupe

3. Distribution des paquets (Cache APT)

```yaml
# Configurer d'où obtenir les paquets
apt_repo:
  apt_server: "10.254.10.223"  # IP du serveur de cache ou serveur de repo direct
use_apt_cache: false  # true = utiliser le cache local, false = accès direct au repo
```

Voir : Système de cache APT

4. Configuration de la licence

```yaml
# Pointer les composants vers le serveur de licence
license_server_api_urls: ["https://10.10.2.150:8443/api"]
license_enforced: true
```

Voir : Serveur de licence

5. Exécuter le déploiement

Vous pouvez déployer des composants individuels. Par exemple, exécutez `services/twag.yml`. Le playbook `services/all.yml` gère tout. Utilisez `--limit=myhost` ou `--limit=mmee,sgw` pour limiter les hôtes sur lesquels vous travaillez.

```
# Déployer le réseau complet
ansible-playbook -i hosts/customer/host_files/production.yml services/all.yml

# Ou déployer des composants spécifiques
ansible-playbook -i hosts/customer/host_files/production.yml services/epc.yml
ansible-playbook -i hosts/customer/host_files/production.yml services/ims.yml
```

Modèles de déploiement

Il existe deux façons d'obtenir l'outil OmniCore sur un contrôleur et de l'exécuter.

Modèle A : Cloner le repo sur le contrôleur

Le modèle traditionnel : clonez ce dépôt sur votre contrôleur Ansible, gardez votre inventaire sous `hosts/`, et exécutez les playbooks directement depuis le checkout. C'est le modèle supposé par l'exemple de déploiement ci-dessus.

Modèle B : Auto-provisionnement `omnicore` .deb

OmniCore est également empaqueté en tant que paquet Debian `omnicore` autonome, de sorte qu'un contrôleur peut installer l'outil à partir d'apt plutôt que de cloner le repo.

- **Ce qu'il installe.** Le paquet (`name: omnicore`, voir `packaging/nfpm.yaml`) expédie les rôles Ansible, les playbooks, les collections *vendues*, un interpréteur Python empaqueté (`/opt/omnicore/python`) et une roue hors ligne vers `/opt/omnicore`. L'étape post-installation (`packaging/postinstall.sh`) construit un venv autonome (`/opt/omnicore/venv`) à partir de l'interpréteur empaqueté + de la roue et crée des liens symboliques des `ansible*` empaquetés sur `PATH` via `/usr/local/bin`, de sorte que le contrôleur exécute les versions exactes de `uv.lock` d'Ansible et de ses bibliothèques Python : pas d'Ansible de distribution, pas de Python système, pas de réseau. Seuls des outils système authentiques (`git`, `rsync`, `openssh-client`) sont tirés par apt. Le paquet est **amd64** et cible **Ubuntu 22.04+** (glibc ≥ 2.35).
- **Comment il est construit.** `packaging/build.sh` prépare le repo dans `/opt/omnicore` (excluant `hosts/`, `.git/`, les virtualenvs et les fichiers de construction), vend les collections Ansible à partir de `requirements.yml`, regroupe l'interpréteur autonome + une roue des dépendances Python verrouillées, exécute un scan secret en mode échec, puis construit le `.deb` avec nfpm.

- **Comment il est publié.** `.github/workflows/build-deb.yml` exécute `build.sh` à chaque push sur `main` et publie le `.deb` (plus un `_metadata.json` en complément) en tant que publication GitHub. Le script `discover_github_debs.py` du serveur apt découvre automatiquement cet actif de publication à travers l'organisation Omnitouch et l'ajoute à aptly. Aucun changement de rôle apt n'est nécessaire.

Flux client :

```
# 1. Installer l'outil à partir d'apt
apt-get install omnicore

# 2. Fournir votre propre inventaire
#    (le paquet n'en expédie pas un - il vit en dehors de
/opt/omnicore)
vi /etc/omnicore/inventory

# 3. Exécuter les playbooks à partir de l'outil installé
cd /opt/omnicore
ansible-playbook -i /etc/omnicore/inventory services/all.yml
```

Parce que l'inventaire vit à `/etc/omnicore/inventory` (séparé de l'outil installé `/opt/omnicore`), la mise à jour du paquet ne touche jamais votre définition d'infrastructure ou votre personnalisation.

Documentation connexe

- Introduction au déploiement Ansible - Premiers pas
- Playbooks de service - **Référence et hiérarchie des playbooks**
- Configuration du fichier Hosts - Définir l'infrastructure
- Norme de planification IP - **Architecture réseau et allocation IP**
- Configuration des variables de groupe - Personnalisation
- Système de cache APT - Gestion des paquets
- Serveur de licence - Gestion des licences
- Surveillance et observabilité - Grafana, Prometheus, alertes et tableaux de bord

- Journalisation centralisée - Collecte des journaux Loki et Alloy

Documentation produit

Pour des informations détaillées sur la configuration de chaque composant :

- **OmniCore** (Noyau de paquets 4G/5G) : https://docs.omnitouch.com.au/docs/repos/OmniCore
  - OmniHSS, OmniSGW, OmniPGW, OmniUPF, OmniDRA, OmniTWAG
- **OmniCall** (Voix et messagerie) : https://docs.omnitouch.com.au/docs/repos/OmniCall
  - OmniTAS, OmniCall CSCF, OmniMessage, OmniSS7, VisualVoicemail
- **OmniCharge/OmniCRM** (Facturation) : https://docs.omnitouch.com.au/docs/repos/OmniCharge
- **Documentation principale** : https://docs.omnitouch.com.au/

Pare-feu par rôle

Le pare-feu par rôle fournit à chaque hôte un ensemble de règles `iptables` et `ip6tables` qui est dérivé d'un modèle déclaratif unique du trafic du réseau, plutôt que de règles écrites à la main par hôte. Un fichier décrit chaque flux autorisé dans le cœur (qui peut atteindre quel rôle, sur quel protocole et port, et pourquoi). À partir de ce modèle, la plateforme produit deux choses qui s'accordent toujours :

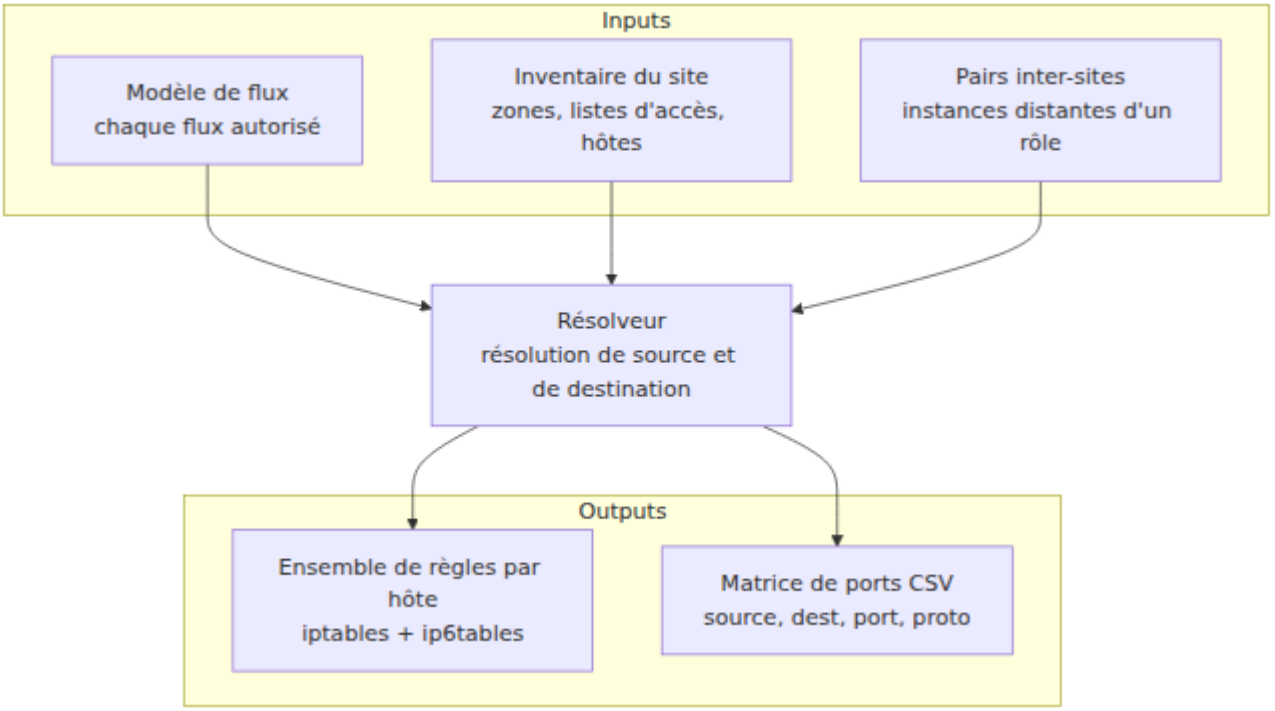
- l'**ensemble de règles appliqué** sur chaque hôte, et
- une **matrice de communication / port** (CSV) pour révision et pour remise à un pare-feu en amont.

Les règles fonctionnent dans l'un des deux modes : **surveillance** (journaliser chaque flux, ne rien bloquer) ou **application** (refus par défaut). Le blocage est délibéré et à l'épreuve des erreurs, et chaque application est protégée par un retour automatique sur l'hôte afin qu'un ensemble de règles erroné ne puisse pas verrouiller les opérateurs.

Table des matières

Aperçu de l'architecture

Un modèle unique plus l'inventaire propre à chaque site alimentent un résolveur. Le résolveur produit à la fois les règles sur l'hôte et la matrice de ports, de sorte que la documentation ne peut jamais décrire quelque chose de différent de ce qui est effectivement appliqué.



Le trafic abandonné et audité est enregistré via le pipeline d'observabilité standard :

Le modèle de flux

Le modèle de flux est la seule source de vérité pour **qui peut atteindre quoi**. Il contient deux parties : un ensemble de valeurs par défaut structurelles (qui ne portent aucune adresse de site), et la liste des flux. Un flux ne dit rien sur les numéros de port directement - il nomme un **service** exposé par le rôle de destination (voir Catalogue de services), et le port est recherché à partir de la déclaration de ce rôle :

```
- id: '13'                          # étiquette stable, apparaît
dans le commentaire de chaque règle
  dst: hss                          # groupe d'inventaire auquel la
règle est appliquée
  service: diameter                 # service nommé exposé par le
rôle hss (-> 3868/tcp+sctp)
  purpose: 'Diameter'               # description, affichée dans la
matrice
  sources: [{ zone: internal }]     # qui peut y accéder (liste de
jetons source)
```

Paramètre	Type	Requis	Par défaut	Description
`id`	Chaîne	Oui	-	Identifiant stable pour le flux. Apparaît dans le commentaire de chaque règle générée sous la forme `ans:<dst>:<id>` afin qu'une règle active puisse être retracée jusqu'au modèle.
`dst`	Chaîne	Oui	-	Le groupe d'inventaire sur lequel la règle s'applique, ou les pseudo-destinations `all` (chaque hôte) et `five_g_sbi` (les groupes de fonctions du réseau 5G). Un flux dont le groupe de destination n'est pas déployé sur un site ne produit aucune règle ni aucune ligne de matrice.
`service`	Chaîne	Oui*	-	Un service nommé déclaré par le rôle de destination (voir Catalogue de services). Les ports sont recherchés à partir de la déclaration du rôle, donc le flux ne réaffirme jamais un numéro de port.
`proto_ports`	Liste	Oui*	-	L'alternative littérale à `service`, pour les flux non liés à un seul rôle (par exemple, les flux `all`-host). Chaque entrée est

Paramètre	Type	Requis	Par défaut	Description
				`PORT/proto` (`tcp`, `udp`, `sctp`; le port peut être une plage `LO-HI`) ou un protocole nu (`icmp`, `esp`, `proto 89`).
`purpose`	Chaîne	Oui	-	Description humaine du flux. Affichée dans la colonne `service` de la matrice de ports.
`sources`	Liste	Oui	-	Les sources autorisées, sous forme de liste de jetons source. Voir Types de sources.

* Un flux fournit **soit** `service` **soit** `proto_ports`, pas les deux.

Types de sources

Chaque entrée dans la liste `sources` d'un flux est un jeton à clé unique qui est résolu par rapport à la configuration et à l'inventaire du site. Un jeton qui ne résout rien (par exemple, une liste d'accès vide) ne contribue à aucune règle : il n'est jamais émis en tant que substitut.

Jeton	Résout à
`{ zone: <name> }`	Les CIDR de cette zone de plan (`firewall.zones.<name>`).
`{ list: <name> }`	Une liste d'accès opérateur, par exemple `allowed_oam_subnets`.
`{ group: <name> }`	Les adresses hôtes du groupe d'inventaire, plus les pairs inter-sites de ce rôle. Voir Pairs inter-sites.
`{ ue_pools: true }`	Les pools UE / abonnés du site (`firewall.ue_pools`).
`{ dra_peers: true }`	Chaque adresse configurée sur la propre liste de pairs Diameter de l'hôte DRA.
`{ roaming: true }`	Plages de roaming (IR.21 / GRX-IPX). Actuellement, toute source : voir Précautions opérationnelles.
`{ literal: "..." }`	Une source en texte libre qui n'est pas dérivée de l'inventaire.

Catalogue de services

Les numéros de port sont **possédés par le rôle qui les ouvre**, et non par le modèle central. Chaque rôle qui expose des ports expédie un `firewall_services.yml` dans son répertoire de rôle, indexé par le(s) groupe(s) d'inventaire qu'il sert (un rôle tel que `cscf` sert plusieurs groupes, donc il déclare chacun). Cela garde la définition du port à côté du code qui l'ouvre :

```yaml
# roles/omnihss/firewall_services.yml
firewall_services:
  hss:
    diameter: [3868/tcp, 3868/sctp]   # S6a / Cx-Dx / Sh
    api_oam:  8080/tcp                # API HSS (OAM)
    api_nf:   8443/tcp                # API HSS (interrogée par
d'autres NFs)
    postgres: 5432/tcp                # réplication logique entre
pairs HSS
```

Le résolveur globule chaque `firewall_services.yml` de rôle dans un seul catalogue. Le `service:` d'un flux est recherché dans le catalogue pour le groupe `dst` de ce flux, donc le modèle exprime seulement la **relation** (qui peut atteindre le service `diameter` de `hss`) tandis que le **port** vit avec le rôle. Ajoutez ou déplacez un écouteur dans un rôle et mettez à jour son `firewall_services.yml`; rien dans le modèle central ne change.

Cette séparation a un filet de sécurité qu'un modèle uniquement central n'a pas : un **contrôle de dérive** compare ce que les rôles déclarent par rapport à ce que les hôtes écoutent réellement.

```
ansible-playbook -i hosts/<Deployment>/host_files/<Site>.yml \
    util_playbooks/firewall_services_audit.yml
```

Pour chaque hôte, il rapporte :

- **non déclaré** - un écouteur actif sur un port qu'aucun rôle n'a déclaré (soit l'ajouter au `firewall_services.yml` du rôle, soit c'est un écouteur inattendu), et
- **non écoutant** - un service déclaré qui n'est pas réellement actif.

Les hôtes dont les groupes ne déclarent aucun service sont ignorés, donc le contrôle est utile pendant que les rôles sont migrés vers le catalogue de manière incrémentielle plutôt que tous en même temps.

Écouteurs conditionnels. Certains ports ne sont ouverts que lorsqu'un autre rôle est déployé : le MME ouvre ses ports LCS (SBc-AP `29168/sctp`, SLs `9082/sctp`) uniquement lorsqu'un hôte `omnilcs` est dans l'inventaire. Une

déclaration statique ne peut pas exprimer cette condition, donc ces ports sont notés dans le `firewall_services.yml` du rôle mais ne sont pas déclarés. Sur un site qui exécute le rôle dépendant, le contrôle de dérive les signale comme non déclarés, ce qui est le bon signal pour les examiner et les autoriser.

Configuration du site

Chaque site déclare un bloc `firewall` sous `all.vars` dans son inventaire. Il n'y a pas de solutions de repli d'adresse : un site doit décrire son propre réseau. Tout ce qu'un site ne déclare pas est simplement vide, et aucune autre adresse d'environnement n'est jamais héritée.

L'exemple ci-dessous est entièrement peuplé afin que chaque champ montre une valeur représentative. Les adresses utilisent des plages de documentation et une disposition cohérente par plan (chaque plan est son propre sous-réseau au sein du superréseau `internal`), ce qui rend les zones utiles : elles permettent à un flux d'autoriser, par exemple, uniquement le plan IMS plutôt que l'ensemble du cœur.

```yaml
firewall:
  allowed_oam_subnets:           ['10.8.0.0/24']       # source de
gestion (SSH, UI, scrape, APIs de gestion)
  allowed_ran_subnets:           ['10.20.0.0/16']      # source de
signalisation RAN (S1AP, NGAP, GTP-U, Abis)
  allowed_ue_voice_subnets:      ['100.64.0.0/16']     # SIP UE
externe vers le P-CSCF
  allowed_carrier_voice_subnets: ['203.0.113.0/24']    # SIP
d'interconnexion des opérateurs
  zones:                                               # zones de
plan : sous-réseau réel de chaque plan
    internal:    ['10.8.0.0/16']                       #
superréseau de cœur de confiance
    mobile_core: ['10.8.1.0/24']                       #   NFs de
cœur mobile (EPC + 5GC + cœur CS)
    core_svc:    ['10.8.2.0/24']                       #   plan
des services de cœur (HSS, DRA, OCS, OAM)
    ims:         ['10.8.3.0/24']                       #   plan
IMS (CSCF, TAS)
    ue_svc:      ['10.8.4.0/24']                       #   plan
accessible par l'UE (P-CSCF, DNS, ePDG)
    core_v6:     ['2001:db8:0:1::/64']                 #
adresses IPv6 des NFs de cœur (pas des UEs)
  ue_pools: ['100.64.0.0/16', '2001:db8:64::/48']      # plages IP
UE / abonnés (v4 et v6)
  extra_rules:                                         # règles
d'acceptation manuelles (voir ci-dessous)
    - { proto: tcp, dport: '8443', source: '203.0.113.10/32',
comment: 'API partenaire' }
```

Seul `allowed_oam_subnets` est obligatoire. Chaque autre liste peut être laissée vide (`[]`) sur un site qui n'a pas cette fonction : un site sans RAN définit `allowed_ran_subnets: []`, un site sans cœur IPv6 définit `core_v6: []`, et un site sans pools UE définit `ue_pools: []`. Une liste vide produit simplement aucune règle pour les flux qui l'utilisent (voir la colonne `Requis` ci-dessous).

Paramètre	Type	Requis	Par défaut	Descripti
`allowed_oam_subnets`	Liste	Oui	-	Sous-réseau source de gestion. Rég SSH, l'interfa utilisateur, l scraping, et APIs de gest Si vide, le générateur s'arrête ave une erreur, une valeur v laisserait la gestion sans source auto et provoque un verrouilla sous application.
`allowed_ran_subnets`	Liste	Non	`[]`	Sous-réseau source pour signalisation RAN (S1AP, NGAP, GTP-U Abis). Peut é vide sur un sans RAN.
`allowed_ue_voice_subnets`	Liste	Non	`[]`	Sous-réseau source pour SIP UE exter vers le P-CS

Paramètre	Type	Requis	Par défaut	Descripti
`allowed_carrier_voice_subnets`	Liste	Non	`[]`	Sous-réseau source pour SIP d'interconne des opérate
`zones`	Carte	Oui	-	Zones de pl mappant chaque nom plan à son(s sous-réseau réel(s). Voir Paramètres zone.
`ue_pools`	Liste	Non	`[]`	Plages IP UE abonnés. Vi le site n'en a pas.
`extra_rules`	Liste	Non	`[]`	Règles d'acceptatio manuelles appliquées s chaque hôte Voir Paramè de règle manuelle.

Paramètres de zone

Chaque zone mappe un nom de plan à une liste de CIDR. Ne définissez que les plans que le site utilise ; laissez le reste vide.

Zone	Description
`internal`	Superréseau de cœur de confiance. Source de signalisation interne la plus large.
`mobile_core`	Fonctions de réseau de cœur mobile à travers les générations : EPC (MME, S/P-GW, UPF), 5GC (AMF, SMF, et le maillage SBI), et cœur CS (MSC, SS7/SIGTRAN).
`ims`	Plan IMS (I-CSCF, S-CSCF, TAS).
`ue_svc`	Plan des services accessibles par l'UE (P-CSCF, DNS, ePDG).
`core_svc`	Plan des services de cœur (HSS, DRA, OCS, OAM).
`core_v6`	Adresses de service de cœur IPv6. Vide si le site n'a pas de cœur IPv6.

Paramètres de règle manuelle

`extra_rules` (global, tous les hôtes) et les `firewall_extra_rules` par hôte ajoutent des règles d'acceptation brutes pour des cas que le modèle ne couvre pas. Une `source` contenant `:` est appliquée sur IPv6.

Paramètre	Type	Requis	Par défaut	Description
`proto`	Chaîne	Non	-	Protocole, par exemple `tcp`, `udp`, `sctp`.
`dport`	Chaîne	Non	-	Port ou plage de destination.
`source`	Chaîne	Non	-	CIDR de source. Un `:` dans la valeur applique la règle sur IPv6.
`comment`	Chaîne	Non	`override`	Étiquette enregistrée dans le commentaire de la règle.

Modes de fonctionnement

Le mode d'application est contrôlé par `firewall_mode`.

Mode	Comportement
`monitor`	Chaque règle d'acceptation est installée, la politique `INPUT` reste `ACCEPT`, et le trafic de passage est journalisé (comme `FW-AUDIT`) puis autorisé. Rien n'est bloqué. Utilisez cela pour valider le modèle par rapport au trafic réel avant de l'appliquer.
`enforce`	Le trafic de passage est journalisé (comme `FW-DROP`) puis bloqué, et les politiques `INPUT` et `FORWARD` sont définies sur `DROP`.

Le blocage est opt-in et à l'épreuve des erreurs : seule la valeur exacte `enforce` bloque jamais le trafic. Toute autre valeur, y compris une variable non définie ou une faute de frappe, est traitée comme une surveillance, donc un hôte ne peut pas commencer à bloquer le trafic par

accident. Cela s'applique à trois niveaux : la valeur par défaut de la plateforme est `monitor`, les sites déclarent `firewall_mode: monitor` dans leur inventaire, et le générateur traite tout ce qui n'est pas `enforce` comme une surveillance. L'application nécessite un changement délibéré.

```
all:
  vars:
    firewall_mode: monitor   # changer en 'enforce' uniquement
après que le journal d'audit confirme la couverture
```

`firewall_mode` peut être défini par site sous `all.vars`, ou remplacé par groupe ou par hôte via la priorité normale des variables.

Pairs inter-sites

Les rôles qui fédèrent à travers les sites (CSCF, TAS, OmniMessage, réplication HSS, OCS) listent leurs pairs distants dans le fichier de pairs du site (par exemple `prod_master_peers.yml`). Ces adresses distantes sont intégrées dans la résolution `{ group: <role> }`, donc un flux qui permet un rôle couvre automatiquement à la fois les instances sur le même site et celles inter-sites sous une politique identique. La même résolution est utilisée pour les règles et pour la matrice, donc les pairs apparaissent dans les deux.

Résolution d'adresses

Un hôte est résolu à **chaque adresse avec laquelle il est déclaré**, pas seulement à sa principale. Cela signifie `ansible_host` plus chaque `secondary_ips.<name>.ip_address` dans l'inventaire. Les nœuds de chemin de données et de signalisation lient des services à des NIC secondaires : le UPF répond au GTP-U N3 sur son adresse `n3`, et le STP reçoit et origine SIGTRAN sur son adresse publique. Les règles et la matrice couvrent toutes ces adresses, en tant que sources et en tant que destinations, donc un pair qui autorise un nœud par l'adresse qui lui est remise autorise l'adresse que le trafic utilise réellement.

Un hôte qui appartient à **plusieurs groupes d'inventaire** reçoit l'**union** des flux et des services déclarés de chaque groupe. Un nœud convergé expose tout pour tous les rôles qu'il exécute, et son ensemble d'écouteurs attendu (utilisé par le contrôle de dérive) est l'union de ses groupes.

Parce que la résolution utilise chaque adresse déclarée plutôt que de suivre quel service lie quel NIC, la matrice peut lister un service sur une adresse à laquelle il ne se lie pas réellement (par exemple, N3 GTP-U affiché sur l'adresse N6 du UPF). Cela est une sur-approximation délibérée : une matrice complète mais légèrement large est sûre à remettre à un pare-feu en amont, tandis que l'omission d'une adresse de chemin de données ou SIGTRAN réelle casse silencieusement le trafic. Si une précision par adresse est nécessaire plus tard, un service peut nommer le NIC auquel il se lie ; jusqu'à ce moment, la résolution reste factuelle et complète.

Matrice de communication / port

La matrice de ports est le modèle de flux résolu par rapport à l'inventaire d'un site, aplati à une ligne par source, hôte de destination, et port/protocole unique. C'est la forme à réviser ou à remettre à un pare-feu en amont. Générez-la avec :

```
ansible-playbook -i hosts/<Deployment>/host_files/<Site>.yml \
    util_playbooks/render_port_matrix.yml
```

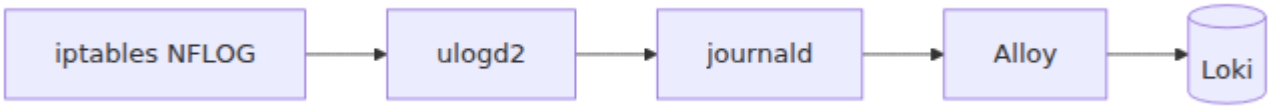
Cela écrit `generated/firewall-port-matrix-exploded.csv` (par site, non engagé). Les colonnes sont :

Colonne	Description
`id`	L'identifiant du flux.
`service`	Le but du flux.
`source`	Un CIDR ou une adresse source résolu.
`destination_role`	Le groupe d'inventaire de destination.
`destination_host`	Le nom de l'hôte de destination.
`destination_ip`	L'adresse de l'hôte de destination.
`port`	Le port ou la plage de destination (vide pour les entrées uniquement de protocole telles que ICMP ou ESP).
`protocol`	Le protocole.

Une ligne apparaît uniquement lorsque sa destination est déployée sur le site et que sa source se résout à une adresse réelle. Les rôles non déployés et les sources non résolues ne produisent aucune ligne.

Protection contre le verrouillage

L'application des règles est protégée par un retour automatique, contrôlé par `firewall_rollback_guard` (activé par défaut) et `firewall_rollback_timeout` :



1. L'ensemble de règles actif est instantané.
2. Un minuteur est armé sur l'hôte qui restaurera l'instantané. Il survit à la coupure de la session de gestion.
3. Les nouvelles règles sont appliquées.

4. Le contrôleur rétablit une nouvelle connexion et effectue un contrôle trivial. Si les nouvelles règles l'ont verrouillé, cela échoue et l'exécution s'arr��te pour cet hôte.
5. Si la connectivité est confirmée, le minuteur est désarmé. Si ce n'est pas le cas, le minuteur se déclenche et restaure l'ensemble de règles fonctionnel sans action de l'opérateur.

Paramètre	Type	Requis	Par défaut	Descriptio
`firewall_rollback_guard`	Booléen	Non	`true`	Active l'instantané, l minuteur et le contrôle de reconnexion autour de cha application.
`firewall_rollback_timeout`	Entier	Non	`120`	Secondes que minuteur atte avant de restaurer automatiquem l'instantané.

IPv6 et double pile

La famille d'adresses est déduite par CIDR : un `:` dans la valeur signifie IPv6, et la règle est appliquée sur `ip6tables`. Une zone ou une liste contenant uniquement des CIDR IPv4 produit donc aucune règle IPv6 pour cette source. Les sites sans cœur IPv6 laissent les zones et les pools IPv6 vides (par exemple `core_v6: []`). Sous application, la politique `INPUT` IPv6 est définie sur `DROP`, donc validez le trafic IPv6 en mode surveillance avant d'appliquer un hôte dont la seule entrée IPv6 légitime arrive par une source uniquement IPv4.

Journalisation

Le trafic audité et abandonné est enregistré avec NFLOG et transporté par le pipeline standard vers Loki. Trois préfixes sont utilisés :

Préfixe	Signification
`FW-FLOW` / `FW-FLOW6`	Une nouvelle connexion vue en mode surveillance. Construit la carte de flux en direct du trafic réellement utilisé.
`FW-AUDIT` / `FW-AUDIT6`	Trafic de passage en mode surveillance : serait bloqué sous application, mais est autorisé ici. Examinez-les avant d'appliquer.
`FW-DROP` / `FW-DROP6`	Trafic de passage effectivement bloqué sous application.

Exemples de requêtes Loki :

```
# Trafic qui serait bloqué une fois cet hôte appliqué
{job="firewall"} |= "FW-AUDIT"

# Trafic étant abandonné sous application, par hôte
sum by (host) (count_over_time({job="firewall"} |= "FW-DROP"
[5m]))
```

Précautions opérationnelles

- **Le roaming et les sources littérales sont actuellement toutes sources.** Les flux dont la source est `roaming` ou `literal` sont émis sans restriction de source (`0.0.0.0/0`). Traitez-les comme ouverts jusqu'à ce que les plages de roaming soient résolues en CIDR réels.
- **ICMP est accepté de toute source.** L'ensemble de règles généré permet ICMP sans condition, ce qui est plus large que la portée de gestion montrée pour le flux ping.

- **Ne pas appliquer un hôte de plan utilisateur sans règles de transfert.** Sous application, la politique `FORWARD` est définie sur `DROP`, et aucune règle de transfert n'est générée. Appliquer sur un UPF, PGW ou SGW sans ajouter de règles de transfert casserait le plan utilisateur. Validez d'abord.
- **La liste de gestion est grossière.** `allowed_oam_subnets` régit SSH, l'interface utilisateur, le scraping, et les APIs de gestion ensemble. Un site qui scrape depuis un réseau différent de celui dont il administre a besoin d'une entrée `extra_rules` pour les séparer.

Configuration des Variables de Groupe

Vue d'ensemble

Le répertoire `group_vars` est l'endroit où vous stockez des fichiers de configuration personnalisés qui remplacent les modèles par défaut.

C'est ici que résident vos configurations spécifiques au client : trunks SIP, règles de routage Diameter, logique de routage SMS, plans d'appel et toute autre personnalisation où vous ne souhaitez pas la configuration par défaut. Il se trouve dans `group_vars`.

Emplacement : `hosts/{Customer}/group_vars/`

Les rôles résolvent ce répertoire à l'exécution en tant que `group_vars_omnicore` - le répertoire `group_vars/` situé à côté du répertoire d'inventaire (`{{ inventory_dir.rsplit('/', 1)[0] }}/group_vars/`). Chaque fois qu'un chemin de fichier ci-dessous fait référence à `{{ group_vars_omnicore }}`, il pointe vers ce répertoire.

Comment ça fonctionne

Les rôles Ansible ont des modèles de configuration par défaut. Pour personnaliser pour un déploiement spécifique, placez vos fichiers personnalisés dans `group_vars` et faites-y référence dans votre fichier hosts.

```
Modèle par défaut du rôle → Remplacement group_vars (si spécifié)
→ Configuration déployée
```

Exemple 1 : Modèle de Configuration Personnalisé (OmniMessage)

Certains composants acceptent des modèles de configuration Jinja2 personnalisés.

Structure des Fichiers

```
hosts/Customer/
└── group_vars/
    └── smsc_controller.exs        # Votre modèle de configuration
personnalisé
```

Référence dans le Fichier Hosts

```yaml
omnimessage:
  hosts:
    customer-smsc-controller01:
      ansible_host: 10.10.3.219
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      smsc_template_config: smsc_controller.exs   # Référencez le
nom de votre modèle dans group_vars
```

Ce qui se passe :

1. Ansible trouve `smsc_template_config: smsc_controller.exs`
2. Cherche dans `hosts/Customer/group_vars/smsc_controller.exs`
3. Le modèle avec Jinja2 (peut utiliser `{{ inventory_hostname }}`, `{{ plmn_id.mcc }}`, etc.)
4. Déploie dans `/etc/omnimessage/runtime.exs`
5. Redémarre le service

Sans `smsc_template_config`, le modèle par défaut du rôle est utilisé.

Détails de configuration : Voir
https://docs.omnitouch.com.au/docs/repos/OmniCall

Exemple 2 : Collections de Fichiers de Configuration (Passerelles & Plans d'Appel OmniTAS)

Certains composants utilisent des répertoires de fichiers de configuration.

Structure des Fichiers

```
hosts/Customer/
└── group_vars/
    ├── gateways_prod/              # Configurations des
passerelles SIP
    │   ├── gateway_carrier1.xml
    │   ├── gateway_carrier2.xml
    │   └── gateway_emergency.xml
    ├── gateways_lab/               # Passerelles de laboratoire
    │   └── gateway_test.xml
    ├── dialplan/                   # Règles de routage d'appels
(par défaut)
    │   ├── mo_dialplan.xml         # Mobile Origine (sortant)
    │   ├── mt_dialplan.xml         # Mobile Terminé (entrant)
    │   └── emergency.xml
    └── dialplan_lab/               # Plans d'appel de laboratoire
        └── mo_dialplan.xml
```

Référence dans le Fichier Hosts

```
applicationserver:
  hosts:
    customer-tas01:
      ansible_host: 10.10.3.60
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      gateways_folder: "gateways_prod"   # Référencez votre
dossier de passerelles à utiliser sur cet hôte
      dialplan_folder: "dialplan"         # Optionnel - par défaut
à "dialplan" si non défini
```

Ce qui se passe :

1. Ansible trouve `gateways_folder: "gateways_prod"`
2. Copie tous les fichiers de `hosts/Customer/group_vars/gateways_prod/` vers `/etc/freeswitch/sip_profiles/`
3. Copie tous les fichiers de `hosts/Customer/group_vars/dialplan/` (ou le dossier spécifié par `dialplan_folder`) vers le répertoire des modèles OmniTAS
4. Les services chargent les configurations

Différents environnements : Utilisez différents dossiers par environnement :

- `gateways_folder: "gateways_lab"`
- `gateways_folder: "gateways_prod"`
- `gateways_folder: "gateways_customer_specific"`
- `dialplan_folder: "dialplan_lab"`
- `dialplan_folder: "dialplan_prod"`

Détails de configuration : Voir https://docs.omnitouch.com.au/docs/repos/OmniCall

---

Exemple 3 : Modèle de Configuration Personnalisé (OmniHSS)

Le Serveur d'Abonnés Domiciliaires accepte des modèles de configuration d'exécution personnalisés.

Structure des Fichiers

```
hosts/Customer/
└── group_vars/
    └── hss_runtime.exs.j2        # Votre modèle de configuration
HSS personnalisé
```

Référence dans le Fichier Hosts

```yaml
omnihss:
  hosts:
    customer-hss01:
      ansible_host: 10.10.3.50
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      hss_template_config: hss_runtime.exs.j2   # Référencez le
nom de votre modèle dans group_vars
```

Ce qui se passe :

1. Ansible trouve `hss_template_config: hss_runtime.exs.j2`
2. Cherche dans `hosts/Customer/group_vars/hss_runtime.exs.j2`
3. Le modèle avec Jinja2 (peut utiliser `{{ inventory_hostname }}`, `{{ plmn_id.mcc }}`, etc.)
4. Déploie dans `/etc/omnihss/runtime.exs`
5. Redémarre le service

Sans `hss_template_config`, le modèle par défaut du rôle est utilisé.

Détails de configuration : Voir https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Exemple 4 : Modèle de Configuration Personnalisé (OmniMME)

L'Entité de Gestion de Mobilité accepte des modèles de configuration d'exécution personnalisés.

Structure des Fichiers

```
hosts/Customer/
└── group_vars/
    └── mme_runtime.exs.j2        # Votre modèle de configuration
MME personnalisé
```

Référence dans le Fichier Hosts

```yaml
omnimme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.3.51
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      mme_template_config: mme_runtime.exs.j2   # Référencez le
nom de votre modèle dans group_vars
```

Ce qui se passe :

1. Ansible trouve `mme_template_config: mme_runtime.exs.j2`
2. Cherche dans `hosts/Customer/group_vars/mme_runtime.exs.j2`
3. Le modèle avec Jinja2 (peut utiliser `{{ inventory_hostname }}`, `{{ plmn_id.mcc }}`, etc.)

4. Déploie dans `/etc/omnimme/runtime.exs`
5. Redémarre le service

Sans `mme_template_config`, le modèle par défaut du rôle est utilisé.

Détails de configuration : Voir https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Exemple 5 : Fichiers de Configuration Requis (OmniCRM)

OmniCRM est source **exclusivement** à partir de group_vars - il n'y a pas de défaut de rôle sur lequel s'appuyer. La configuration de l'API n'est délibérément pas intégrée dans le rôle, de sorte qu'un fichier manquant échoue bruyamment l'exécution plutôt que d'expédier silencieusement les secrets d'un autre site.

Mode de déploiement préféré : paquets apt

Avec `omnicrm_deploy_mode: package` (défaut du rôle), Ansible installe des paquets génériques `omnicrm-api` et `omnicrm-ui` depuis `apt_repo`, modèle la configuration sous `/etc/omnicrm/`, et redémarre `omnicrm-api` via systemctl. L'interface utilisateur n'est **pas** reconstruite sur l'hôte ; le branding provient du bloc `ui:` dans `crm_config.yaml` (servi comme `GET /crm/public/ui-config`).

Structure des Fichiers

```
hosts/Customer/
└── group_vars/
    ├── crm_config.yaml          # API + ui: configuration
(requise)
    ├── deployment_assets/       # Logos/icônes optionnels →
/etc/omnicrm/deployment_assets
    │   └── logos/
    ├── customer_crm.env         # Optionnel ; uniquement pour
omnicrm_deploy_mode: source ou développement local de l'UI
    ├── cellSites.txt            # Données de site cellulaire
optionnelles
    ├── constants.js             # Modèles de messages de
diffusion cellulaire optionnels
    └── site_data.json           # Limites géographiques de
diffusion cellulaire optionnelles
```

Référence dans le Fichier Hosts

```
omnicrm:
  hosts:
    customer-crm01:
      ansible_host: 10.10.3.70
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      omnicrm_deploy_mode: package          # par défaut ;
utiliser source pour git+yarn hérité
      # omnicrm_package_version: "20260810...."  # version apt
optionnelle
      logo_light: logo-light.png            # copié dans
deployment_assets/logos/
      logo_dark: logo-dark.png
      login_background: login-bg.jpg        # →
deployment_assets/login/auth-one-bg.jpg (runtime)
      login_splash: splash.jpg              # →
deployment_assets/login/login-side.jpg (garde l'extension source)
      ssl_fqdn: portal.example.com
      # crm_env_override: customer_crm.env  # uniquement requis
pour le mode source
      cell_sites: cellSites.txt             # Optionnel
      message_templates: constants.js       # Optionnel
      site_data: site_data.json             # Optionnel
```

Ce qui se passe (mode paquet) :

1. Apt installe `omnicrm-api` et `omnicrm-ui`.
2. Modèle `{{ group_vars_omnicore }}/crm_config.yaml` dans `/etc/omnicrm/crm_config.yaml` (**requise**).
3. Synchronise `deployment_assets/` (et optionnel `logo_*` / `login_*`) vers `/etc/omnicrm/deployment_assets/`.
4. Redémarre `omnicrm-api` ; nginx sert l'UI depuis `/usr/share/omnicrm-ui`.

Le splash de connexion / arrière-plan d'authentification sont servis à l'exécution via `GET /crm/public/assets/login/…` (voir `ui.branding.login_splash` / `login_background` dans `crm_config.yaml`). Ils ne sont pas intégrés dans le `.deb` de l'UI. La copie garde l'extension source (`splash.jpg` → `login/login-side.jpg`) ; l'API sert ensuite quel que soit le fichier `login-side.*` existant.

Ce qui se passe (mode source, hérité) :

1. Modèle `crm_config.yaml` dans `/etc/omnicrm/OmniCRM-API/`.
2. Modèle `{{ crm_env_override }}` dans le fichier .env de l'UI (**requise** en mode source).
3. Construit l'UI sur l'hôte avec yarn.

Important : `crm_config.yaml` doit être présent. Pour le mode paquet, inclure un bloc `ui:` et définir `ui.assets_dir: '/etc/omnicrm/deployment_assets'`.

Détails de configuration : Voir https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Exemple 6 : Répertoire de Données de Provisionnement (OmniHSS)

Le service de provisionnement OmniHSS (`services/provision_omnihss.yml`) lit les données d'abonnés et de profils à partir d'un répertoire `omnihss/` dédié à l'intérieur de group_vars.

Structure des Fichiers

```
hosts/Customer/
└─── group_vars/
    └── omnihss/
        ├── subscribers.csv              # Abonnés à provisionner
        ├── apn_identifiers.json
        ├── apn_qos_profiles.json
        ├── apn_profiles.json
        ├── eir_rules.json
        ├── ims_profiles.json
        ├── epc_profiles.json
        ├── roaming_rules.json
        ├── roaming_profiles.json
        ├── flow_information_rules.json
        ├── charging_rules.json
        └── pcrf_profiles.json
```

Ce qui se passe :

1. Le service lit `{{ group_vars_omnicore }}/omnihss/subscribers.csv` pour la liste des abonnés à provisionner.
2. Il lit les fichiers JSON accompagnants du même répertoire `omnihss/` (`apn_identifiers.json`, `apn_qos_profiles.json`, `apn_profiles.json`, `eir_rules.json`, `ims_profiles.json`, `epc_profiles.json`, `roaming_rules.json`, `roaming_profiles.json`, `flow_information_rules.json`, `charging_rules.json`, `pcrf_profiles.json`) pour définir les données APN, QoS, IMS, EPC, itinérance, flux, facturation et PCRF appliquées à ces abonnés.

Détails de configuration : Voir https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Exemple 7 : Identifiants SMTP Grafana (Surveillance)

Le modèle `grafana.ini` du rôle de surveillance connecte les paramètres d'alerte par e-mail Grafana (SMTP) à partir de group_vars. L'utilisateur et le mot de passe n'ont **aucun défaut de rôle** - ils ne sont délibérément pas intégrés, de sorte qu'une valeur manquante échoue bruyamment l'exécution plutôt que d'expédier des identifiants.

Référence dans le Fichier Hosts

```
monitoring:
  hosts:
    customer-oam01:
      ansible_host: 10.10.3.135
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      grafana_smtp_user: "smtp-user"          # REQUIS - pas de
défaut
      grafana_smtp_password: "smtp-password"  # REQUIS - pas de
défaut
      grafana_smtp_host: "in-v3.mailjet.com:587"  # Optionnel -
par défaut à in-v3.mailjet.com:587
```

Ce qui se passe :

1. `grafana_smtp_user` et `grafana_smtp_password` sont modélisés dans la section `[smtp]` de `/etc/grafana/grafana.ini`. Les deux sont requis - il n'y a pas de solution de repli.
2. `grafana_smtp_host` est optionnel et par défaut à `in-v3.mailjet.com:587` si non défini.

Détails de configuration : Voir https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Exemple 8 : Clés HNET pour la Dé-concealement SUCI UDM

L'OmniUDM effectue le dé-concealement SUCI (TS 33.501 §6.12) : il récupère le SUPI à partir du SUCI dissimulé qu'un UE présente lors de l'enregistrement. Cela nécessite les clés **privées** du Réseau Domiciliaire qui s'associent aux clés **publiques** du Réseau Domiciliaire provisionnées sur les USIM. Ces clés privées résident dans un répertoire dédié `hnet_udm/` à l'intérieur de group_vars et sont déployées dans `/etc/omniudm/hnet/` sur l'hôte UDM par le rôle `5gc_nf`.

Structure des Fichiers

```
hosts/Customer/
└── group_vars/
    └── hnet_udm/
        ├── curve25519-1.key       # Clé privée X25519, PEM
(Identifiant de Clé Publique du Réseau Domiciliaire 1)
        ├── curve25519-1.pub       # Valeur publique brute de 32
octets (hex) - pour le provisionnement USIM
        ├── secp256r1-2.key        # Clé privée EC (prime256v1),
PEM (Identifiant de Clé Publique HNET 2)
        └── ...
```

Nommage des fichiers : `<curve>-<id>.key`, où `<id>` est l'Identifiant de Clé Publique du Réseau Domiciliaire que l'UDM associe au SUCI, et `<curve>` est `curve25519` (Profil A, X25519) ou `secp256r1` (Profil B, prime256v1). Le nom du fichier est la manière dont la clé est liée à son Identifiant de clé HNET. Il doit correspondre à l'ID programmé sur l'USIM.

Ce qui se passe :

1. Le rôle s'assure que `{{ group_vars_omnicore }}/hnet_udm/` existe sur le contrôleur Ansible (créé avec `become: false` : les clés restent sur le contrôleur et ne sont jamais générées sur la boîte distante).
2. Il scanne le répertoire à la recherche de fichiers `*.key`.
3. **Si aucun n'est trouvé, une clé par défaut `curve25519-1.key` (X25519) est générée automatiquement** avec `openssl genpkey`, de

sorte que le dé-concealement SUCI fonctionne dès le départ. Cela est idempotent (`creates:` garde) : une clé existante n'est jamais écrasée.

4. Pour chaque `curve25519-*.key`, la valeur publique brute correspondante de 32 octets est exportée dans un fichier `.pub` (hex) pour le provisionnement USIM. Les fichiers `.pub` restent dans l'inventaire et ne sont **pas** déployés sur l'UDM.
5. Tous les fichiers `*.key` sont copiés dans `/etc/omniudm/hnet/` sur l'UDM (mode `0600`, propriété root), et `omniudm` est redémarré.

> **Important :** Une clé générée automatiquement est aléatoire et ne dé-concealera que les SUCI des USIM programmées avec sa clé publique **correspondante**. Pour un laboratoire où vous contrôlez la programmation des SIM, prenez le `.pub` exporté et provisionnez-le comme la Clé Publique du Réseau Domiciliaire (ID 1) sur l'USIM. Si vos SIM utilisent le schéma nul (pas de dissimulation), aucune clé HNET n'est nécessaire et ce répertoire peut rester vide.

Pour exporter une clé publique à partir d'une clé privée existante manuellement :

```
openssl pkey -in curve25519-1.key -pubout -outform DER | tail -c
32 | xxd -p -c 32
```

Détails de configuration : Voir https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Exemple de Structure de

Répertoire dans le Monde Réel

```
hosts/Customer/
├── host_files/
│   └── production.yml          # Le fichier hosts fait référence
aux fichiers group_vars
└── group_vars/
    ├── smsc_controller.exs     # Modèle personnalisé OmniMessage
    ├── smsc_smpp.exs           # Modèle personnalisé SMPP
OmniMessage
    ├── tas_runtime.exs.j2      # Modèle personnalisé TAS
    ├── hss_runtime.exs.j2      # Modèle personnalisé HSS
    ├── mme_runtime.exs.j2      # Modèle personnalisé MME
    ├── dra_runtime.exs.j2      # Modèle personnalisé DRA
    ├── pgwc_runtime.exs.j2     # Modèle personnalisé PGW
    ├── dea_runtime.exs.j2      # Modèle personnalisé DEA
    ├── upf_config.yaml         # Configuration UPF
    ├── crm_config.yaml         # Configuration CRM
    ├── stp.j2                  # Modèle SS7 STP
    ├── hlr.j2                  # Modèle SS7 HLR
    ├── camel.j2                # Modèle SS7 CAMEL
    ├── ipsmgw.j2               # Modèle IP-SM-GW
    ├── omnicore_smsc_ims.yaml.j2  # Configuration SMSC IMS
    ├── pytap.yaml              # Configuration TAP3
    ├── sip_profiles/           # Passerelles SIP (dossier)
    │   └── gateway_otw.xml
    ├── dialplan/               # Règles de routage d'appels
(dossier)
    │   ├── mo_dialplan.xml     # Mobile Origine
    │   ├── mt_dialplan.xml     # Mobile Terminé
    │   └── mo_emergency.xml    # Routage d'urgence
    ├── omnihss/                # Données de provisionnement
OmniHSS (dossier)
    │   ├── subscribers.csv         # Abonnés à provisionner
    │   ├── apn_identifiers.json
    │   ├── apn_qos_profiles.json
    │   ├── apn_profiles.json
    │   ├── eir_rules.json
    │   ├── ims_profiles.json
    │   ├── epc_profiles.json
    │   ├── roaming_rules.json
    │   ├── roaming_profiles.json
```

```
    │   ├── flow_information_rules.json
    │   ├── charging_rules.json
    │   └── pcrf_profiles.json
    └── hnet_udm/               # Clés de dé-concealement SUCI UDM
(dossier)
        ├── curve25519-1.key        # Clé privée X25519
(Identifiant de Clé Publique HNET 1)
        └── curve25519-1.pub        # Valeur publique brute (hex)
pour le provisionnement USIM
```

Paramètres Communs Qui

Référencent group_vars

Paramètre	Composant	Références
`smsc_template_config`	omnimessage	Fichier modèle Jinja2 (ex. `smsc_controller.exs`)
`smsc_smpp_template_config`	omnimessage_smpp	Fichier modèle Jinja2 (ex. `smsc_smpp.exs`)
`gateways_folder`	applicationserver	Nom du dossier (ex. `gateways_prod`)
`dialplan_folder`	applicationserver	Nom du dossier (ex. `dialplan`) - par défaut à `dialplan` si non défini
`tas_runtime_template`	applicationserver	Fichier modèle Jinja2 (ex. `tas_runtime.exs.j2`) - par défaut à `tas_runtime.exs.j2` si non défini
`hss_template_config`	omnihss	Fichier modèle Jinja2 (ex. `hss_runtime.exs.j2`)
`mme_template_config`	omnimme	Fichier modèle Jinja2 (ex. `mme_runtime.exs.j2`)

Paramètre	Composant	Références
`dra_template_config`	dra	Fichier modèle Jinja2 (ex. `dra_runtime.exs.j2`)
`pgwc_template_config`	pgwc	Fichier modèle Jinja2 (ex. `pgwc_runtime.exs.j2`)
`frr_template_config`	omniupf	Fichier modèle Jinja2 (ex. `frr.conf.j2`)
Modèles SS7	ss7 (divers rôles)	Fichiers modèles Jinja2 (ex. `stp.j2`, `hlr.j2`, `camel.j2`)
`crm_config.yaml`	omnicrm	Fichier de configuration API - **requise**, source uniquement à partir de `{{ group_vars_omnicore }}/crm_config.yaml` (pas de défaut de rôle)
`crm_env_override`	omnicrm	Nom de fichier .env de l'UI (ex. `customer_crm.env`) - **requise**, pas de défaut de rôle
`cell_sites`	omnicrm	Fichier de données de site cellulaire optionnel (ex. `cellSites.txt`)

Paramètre	Composant	Références
`message_templates`	omnicrm	Modèles de messages de diffusion cellulaire optionnels (ex. `constants.js`)
`site_data`	omnicrm	Limites géographiques de diffusion cellulaire optionnelles (ex. `site_data.json`)
`grafana_smtp_user`	monitoring	Utilisateur SMTP Grafana - **requise**, pas de défaut de rôle
`grafana_smtp_password`	monitoring	Mot de passe SMTP Grafana - **requise**, pas de défaut de rôle
`grafana_smtp_host`	monitoring	Hôte SMTP Grafana optionnel - par défaut à `in-v3.mailjet.com:587`
Provisionnement OmniHSS	omnihss	Répertoire `omnihss/` avec `subscribers.csv` + fichiers JSON de profil
Clés HNET	5gc_nf (omniudm)	Répertoire `hnet_udm/` de clés privées `<curve>-<id>.key` pour le dé-concealement SUCI - génère automatiquement une

Paramètre	Composant	Références
		`curve25519-1.key` par défaut si vide
Configurations YAML	Divers composants	Fichiers de configuration directs (ex. `upf_config.yaml`, `crm_config.yaml`)

Points Clés

1. **group_vars contient des personnalisations** - Remplacements pour les configurations par défaut
2. **Référence par nom** - Utilisez des paramètres comme `smsc_template_config` ou `gateways_folder`
3. **Les modèles supportent Jinja2** - Accédez à n'importe quelle variable Ansible avec `{{ variable_name }}`
4. **Les dossiers déploient tout** - Tous les fichiers dans les dossiers référencés sont copiés
5. **Versionnez tout** - Commitez tous les group_vars dans Git

Quand Utiliser group_vars

✅ Utilisez group_vars pour :

- Modèles de configuration personnalisés pour les composants
- Définitions de passerelles SIP
- Plans d'appel de routage
- Règles de routage Diameter
- Paramètres spécifiques au client qui remplacent les valeurs par défaut

❌ Ne pas utiliser group_vars pour :

- Configuration de base des hôtes (IPs, noms d'hôtes) - Utilisez le fichier hosts
- Tests ponctuels - Utilisez des variables spécifiques aux hôtes dans le fichier hosts
- Changements temporaires - Modifiez sur la cible, commitez dans group_vars si permanent

---

Documentation Connexe

- Référence de Configuration - Tous les paramètres d'hôte et leur fonctionnement
- Configuration du Fichier Hosts - Comment structurer les fichiers hosts
- **Configuration OmniCall** : https://docs.omnitouch.com.au/docs/repos/OmniCall - Ce qui va dans les fichiers de configuration
- **Configuration OmniCore** : https://docs.omnitouch.com.au/docs/repos/OmniCore - Détails de configuration des composants

Configuration du Fichier Hosts

Vue d'ensemble

Le fichier hosts (également appelé fichier d'inventaire) est le document de configuration central qui définit l'ensemble de votre déploiement de réseau cellulaire. Il spécifie :

- Quelles fonctions réseau déployer
- Où elles s'exécutent (adresses IP, segments de réseau)
- Comment elles sont configurées (paramètres spécifiques au service)
- Paramètres spécifiques au client (PLMN, identifiants, fonctionnalités)

Emplacement du Fichier

Les fichiers hosts se trouvent sous le répertoire de niveau supérieur `hosts/` (défini comme l'`inventory` Ansible dans `ansible.cfg`), organisés par client :

```
hosts/
└── Customer_Name/
    └── host_files/
        ├── Production.yml
        ├── Staging.yml
        └── Customer_Lab.yml
```

Les noms de fichiers sont arbitraires. Il y a un fichier par déploiement, généralement nommé d'après le déploiement ou le site (par exemple `Production.yml`, `Staging.yml`) plutôt qu'un schéma fixe `production.yml`/`staging.yml`/`lab.yml`.

Exemple de Structure de Fichier Hosts

Cet exemple simplifié montre les sections clés :

```yaml
# Composants EPC
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15
      gateway: 10.10.1.1
      host_vm_network: "vmbr1"
      mme_code: 1
      network_name_short: Customer
      tac_list: [600, 601, 602]

sgw:
  hosts:
    customer-sgw01:
      ansible_host: 10.10.1.25
      gateway: 10.10.1.1
      cdrs_enabled: true

pgwc:
  hosts:
    customer-pgw01:
      ansible_host: 10.10.1.21
      gateway: 10.10.1.1
      ip_pools:
        - '100.64.16.0/24'

# Composants IMS
pcscf:
  hosts:
    customer-pcscf01:
      ansible_host: 10.10.4.165

# Services de Support
license_server:
  hosts:
    customer-licenseserver:
      ansible_host: 10.10.2.150

# Variables Globales
all:
  vars:
    ansible_connection: ssh
    ansible_password: password
```

```
    customer_name_short: customer
    plmn_id:
      mcc: '001'
      mnc: '01'
```

Paramètres Communs des Hôtes

Configuration Réseau

Chaque hôte inclut généralement :

```
pcscf:
  hosts:
    customer-pcscf01:
      ansible_host: 10.10.1.15      # Adresse IP pour l'accès SSH
      gateway: 10.10.1.1             # Passerelle par défaut
      host_vm_network: "vmbr1"       # nom de la NIC à utiliser
sur l'hyperviseur
```

Remarque : Pour des conseils sur la planification des adresses IP et les stratégies de segmentation de réseau, voir la Norme de Planification IP qui décrit l'architecture recommandée à quatre sous-réseaux pour les déploiements OmniCore.

Utilisateurs de Proxmox : Le paramètre `host_vm_network` spécifie quel pont utiliser. Voir Déploiement Proxmox VM/LXC pour un provisionnement automatisé.

Allocation des Ressources VM

Pour les services nécessitant des ressources spécifiques :

```
num_cpus: 4                     # Cœurs de CPU
memory_mb: 8192                 # RAM en mégaoctets
proxmoxLxcDiskSizeGb: 50      # Taille du disque en Go
```

Paramètres Spécifiques au Service

Chaque fonction réseau a ses propres paramètres. Exemples :

MME :

```
mme_code: 1                    # Identifiant MME (1-255)
mme_gid: 1                     # ID de groupe MME
network_name_short: Customer   # Nom du réseau (affiché sur les
téléphones)
network_name_long: Customer Network
tac_list: [600, 601, 602]     # Codes de zone de suivi
```

PGW :

```
ip_pools:                      # Pools IP pour les abonnés
  - '100.64.16.0/24'
  - '100.64.17.0/24'
combined_CP_UP: false          # Plan de contrôle/utilisateur
séparé
```

Pour une explication détaillée de ce que chaque variable contrôle, voir : Référence de Configuration

Serveur d'Application :

```
online_charging_enabled: true  # Activer l'intégration OCS
tas_branch: "main"             # Branche du logiciel à déployer
gateways_folder: "gateways_prod"  # Configuration du passerelle
SIP
```

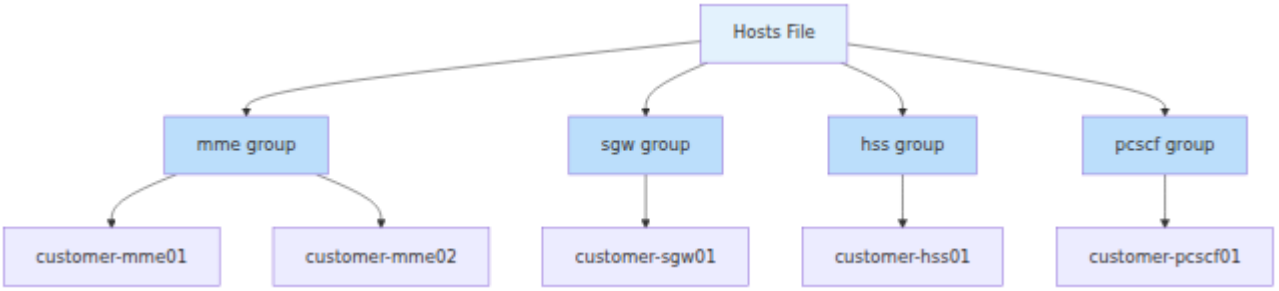
Section des Variables Globales

La section `all:vars` contient des paramètres qui s'appliquent à l'ensemble du déploiement :

```yaml
all:
  vars:
    # Authentification
    ansible_connection: ssh
    ansible_password: password
    ansible_become_password: password

    # Identité du Client
    customer_name_short: customer
    customer_legal_name: "Customer Inc."
    site_name: "Chicago DC1"
    region: US

    # Identifiant PLMN (Réseau Mobile)
    plmn_id:
      mcc: '001'              # Code de pays mobile
      mnc: '01'               # Code de réseau mobile
      mnc_longform: '001'     # MNC avec zéros en tête

    # Noms de Réseau
    network_name_short: Customer
    network_name_long: Customer Network

    # Dépôt APT
    # Remarque : Si le groupe apt_cache_servers est défini avec des hôtes,
    # use_apt_cache par défaut est vrai et apt_repo.apt_server
    # par défaut est l'adresse IP du premier serveur de cache automatiquement
    apt_repo:
      apt_server: "10.254.10.223"
      apt_repo_username: "customer"
      apt_repo_password: "secure-password"
    use_apt_cache: false

    # Fuseau Horaire
    TZ: America/Chicago
```

Comprendre les Groupes d'Hôtes

Ansible organise les hôtes en groupes qui correspondent à des rôles :



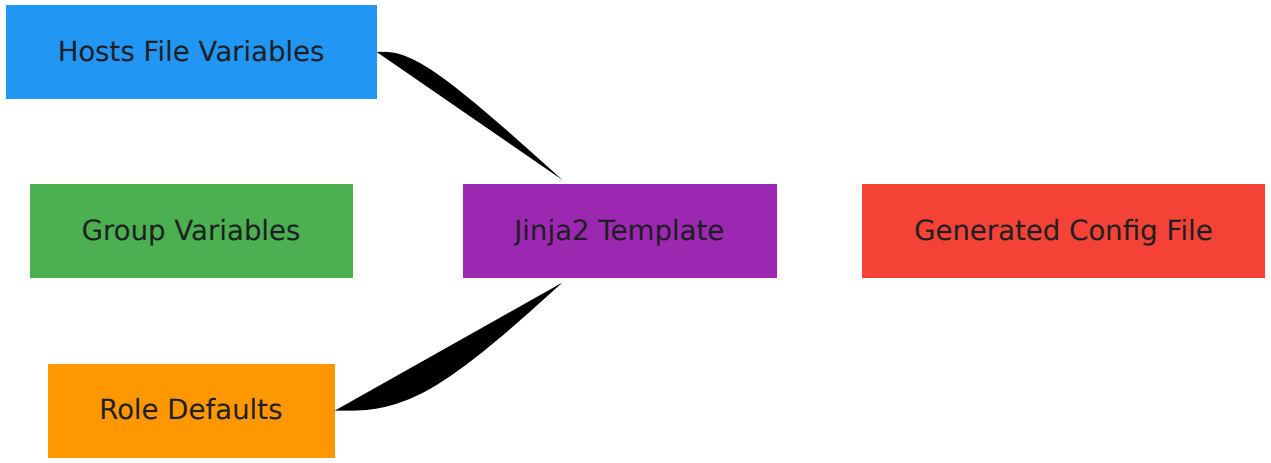
Lorsque vous exécutez un playbook ciblant `mme`, il s'applique à tous les hôtes de la section `mme:hosts:`.

Le diagramme ci-dessus montre seulement quelques groupes. Beaucoup d'autres existent, y compris `ocs`, `crm`, `smsc`, `omnimessage`, `dra`, `xcap`, `homer`, `cbc`, et les groupes 5GC `amf`, `smf`, et `upf`, parmi d'autres.

Configuration avec des Modèles Jinja2

Ansible utilise **le modèle Jinja2** pour générer des fichiers de configuration à partir des variables définies dans votre fichier hosts et group_vars.

Comment Fonctionne Jinja2



Exemple d'Utilisation de Modèle

Le fichier hosts définit :

```yaml
plmn_id:
  mcc: '001'
  mnc: '01'
customer_name_short: acme
```

Modèle Jinja2 (dans le rôle) :

```yaml
# mme_config.yml.j2
network:
  plmn:
    mcc: {{ plmn_id.mcc }}
    mnc: {{ plmn_id.mnc }}
  operator: {{ customer_name_short }}
  realm: epc.mnc{{ plmn_id.mnc_longform }}.mcc{{ plmn_id.mcc
}}.3gppnetwork.org
```

Fichier de configuration généré :

```yaml
network:
  plmn:
    mcc: 001
    mnc: 01
  operator: acme
  realm: epc.mnc001.mcc001.3gppnetwork.org
```

Modèles Jinja2 Communs

Accéder aux variables imbriquées :

```
{{ plmn_id.mcc }}
{{ apt_repo.apt_server }}
```

Logique conditionnelle :

```
{% if online_charging_enabled %}
  charging:
    enabled: true
    ocs_ip: {{ ocs_ip }}
{% endif %}
```

Boucles :

```
tracking_areas:
{% for tac in tac_list %}
  - {{ tac }}
{% endfor %}
```

Formatage :

```
# Zéro-padding à 3 chiffres
mnc{{ '%03d' | format(plmn_id.mnc|int) }}
```

Surcharge des Variables avec group_vars

Alors que le fichier hosts définit les infrastructures et les paramètres spécifiques aux hôtes, `group_vars` peut remplacer les valeurs par défaut pour des groupes d'hôtes.

Voir : Configuration des Variables de Groupe

Exemple Complet de Fichier Hosts

Cet exemple plus complet a des données sensibles obscurcies :

```yaml
# EPC Core
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15
      gateway: 10.10.1.1
      host_vm_network: "vmbr1"
      mme_code: 1
      mme_gid: 1
      network_name_short: Customer
      network_name_long: Customer Network
      tac_list: [600, 601, 602, 603]
      omnimme:
        sgw_selection_method: "random_peer"
        pgw_selection_method: "random_peer"

sgw:
  hosts:
    customer-sgw01:
      ansible_host: 10.10.1.25
      gateway: 10.10.1.1
      host_vm_network: "vmbr1"
      cdrs_enabled: true

pgwc:
  hosts:
    customer-pgw01:
      ansible_host: 10.10.1.21
      gateway: 10.10.1.1
      host_vm_network: "vmbr1"
      ip_pools:
        - '100.64.16.0/24'
      combined_CP_UP: false

hss:
  hosts:
    customer-hss01:
      ansible_host: 10.10.2.140
      gateway: 10.10.2.1
      host_vm_network: "vmbr2"

# IMS Core
pcscf:
```

```
  hosts:
    customer-pcscf01:
      ansible_host: 10.10.4.165
      gateway: 10.10.4.1
      host_vm_network: "vmbr4"

icscf:
  hosts:
    customer-icscf01:
      ansible_host: 10.10.3.55
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"

scscf:
  hosts:
    customer-scscf01:
      ansible_host: 10.10.3.45
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"

applicationserver:
  hosts:
    customer-as01:
      ansible_host: 10.10.3.60
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      online_charging_enabled: false
      gateways_folder: "gateways_prod"

# Services de Support
license_server:
  hosts:
    customer-licenseserver:
      ansible_host: 10.10.2.150
      gateway: 10.10.2.1
      host_vm_network: "vmbr2"

monitoring:
  hosts:
    customer-oam01:
      ansible_host: 10.10.2.135
      gateway: 10.10.2.1
      host_vm_network: "vmbr2"
      num_cpus: 4
```

```
      memory_mb: 8192

dns:
  hosts:
    customer-dns01:
      ansible_host: 10.10.2.177
      gateway: 10.10.2.1
      host_vm_network: "vmbr2"

# Variables Globales
all:
  vars:
    ansible_connection: ssh
    ansible_password: password
    ansible_become_password: password

    customer_name_short: customer
    customer_legal_name: "Customer Network Inc."
    site_name: "Primary DC"
    region: US
    TZ: America/Chicago

    # Configuration PLMN
    plmn_id:
      mcc: '001'
      mnc: '01'
      mnc_longform: '001'
    diameter_realm: epc.mnc{{ plmn_id.mnc_longform }}.mcc{{
plmn_id.mcc }}.3gppnetwork.org

    # Noms de Réseau
    network_name_short: Customer
    network_name_long: Customer Network
    tac_list: [600, 601]

    # Configuration APT
    apt_repo:
      apt_server: "10.254.10.223"
      apt_repo_username: "customer"
      apt_repo_password: "secure-password"
    use_apt_cache: false

    # Configuration de Chargement
    charging:
```

```
      data:
        online_charging:
          enabled: false
      voice:
        online_charging:
          enabled: true
          domain: "mnc{{ plmn_id.mnc_longform }}.mcc{{ plmn_id.mcc
}}.3gppnetwork.org"

    # Règles de Pare-feu
    firewall:
      allowed_ssh_subnets:
        - '10.0.0.0/8'
        - '192.168.0.0/16'
      allowed_ue_voice_subnets:
        - '10.0.0.0/8'
      allowed_signaling_subnets:
        - '10.0.0.0/8'

    # Configuration de l'Hyperviseur (Exemple de VM Proxmox)
    proxmoxServers:
      customer-prxmx01:
        # type_de_deploiement omis → par défaut à "vm"
        proxmoxServerAddress: 10.10.0.100
        proxmoxServerPort: 8006
        proxmoxApiTokenName: Customer
        proxmoxApiTokenSecret: "token-secret"
        proxmoxNodeName: pve01
        proxmoxTemplateName: ubuntu-24.04-cloud-init-template
        proxmoxTemplateId: 9000
        proxmoxTemplateUser: omnitouch        # nom d'utilisateur
cloud-init optionnel ; par défaut au premier clé local_users
        proxmoxTemplatePassword: omnitouch    # mot de passe
cloud-init optionnel ; par défaut au nom d'utilisateur cloud-init

    # Pour un site LXC, définissez deployment_type: lxc et
    # proxmoxLxcOsTemplate dans chaque entrée proxmoxServers à la
place.
```

Voir Déploiement Proxmox VM/LXC pour des détails complets sur la configuration et la mise en place de Proxmox.

Références de Documentation Produit

Pour une configuration détaillée de chaque composant, référez-vous à la documentation officielle du produit :

Composants OmniCore :

- **Documentation OmniCore** : https://docs.omnitouch.com.au/docs/repos/OmniCore
- **OmniHSS** - Serveur d'Abonnés Domiciliaires
- **OmniSGW** - Passerelle de Service (Plan de contrôle)
- **OmniPGW** - Passerelle de Paquet (Plan de contrôle)
- **OmniUPF** - Fonction de Plan Utilisateur
- **OmniDRA** - Agent de Routage Diameter
- **OmniTWAG** - Passerelle d'Accès WLAN de Confiance

Composants OmniCall :

- **Documentation OmniCall** : https://docs.omnitouch.com.au/docs/repos/OmniCall
- **OmniTAS** - Serveur d'Application IMS (VoLTE/VoNR)
- **OmniCall CSCF** - Fonctions de Contrôle de Session d'Appel
- **OmniMessage** - Centre SMS
- **OmniMessage SMPP** - Support du Protocole SMPP
- **OmniSS7** - Pile de Signalisation SS7
- **VisualVoicemail** - Messagerie Vocale

OmniCharge/OmniCRM :

- **Documentation OmniCharge** : https://docs.omnitouch.com.au/docs/repos/OmniCharge

Documentation Connexe

- Introduction au Déploiement Ansible - Processus de déploiement global
- Référence de Configuration - **Guide complet de toutes les variables de configuration**
- Configuration des Variables de Groupe - Surcharge des configurations par défaut
- Norme de Planification IP - **Architecture réseau et directives d'allocation IP**
- Configuration Netplan - **IPs secondaires et configuration réseau avancée**
- Système de Cache APT - Distribution de paquets
- Serveur de Licences - Gestion des licences
- Aperçu de l'Architecture de Déploiement - Vue complète du système

Prochaines Étapes

1. Créez votre fichier hosts basé sur ce modèle
2. Définissez votre PLMN et votre identité réseau
3. Configurez l'accès au dépôt APT
4. Configurez le serveur de licences
5. Personnalisez avec group_vars si nécessaire
6. Déployez avec des playbooks Ansible

Serveur de Licences

Aperçu

Le Serveur de Licences gère l'activation des fonctionnalités pour tous les composants Omnitouch. Chaque composant valide sa licence au démarrage et périodiquement pendant son fonctionnement.

Configuration

1. Définir dans le Fichier Hosts

```
license_server:
  hosts:
    customer-licenseserver:
      ansible_host: 10.10.2.150
      gateway: 10.10.2.1
      host_vm_network: "vmbr2"

all:
  vars:
    customer_legal_name: "Nom du Client"
    license_server_api_urls: ["https://10.10.2.150:8443/api"]
```

> **Remarque :** `license_enforced` est **limité à HSS**, pas un basculement d'application global. Il est uniquement défini pour le rôle `omnihss` (`roles/omnihss/defaults/main.yml`, où il est par défaut à `false`) et est actuellement seulement lu par le générateur HLD ; aucun autre rôle ne l'utilise. Définissez-le sous les variables de groupe omnihss si vous en avez besoin.

2. Fournir le Fichier de Licence

Placez `license.json` (fourni par Omnitouch) dans `hosts/Customer/group_vars/`

3. Déployer

```
ansible-playbook -i hosts/Customer/host_files/production.yml
services/license_server.yml
```

Les répertoires d'inventaire réels sont en majuscules et préfixés (par exemple, `hosts/Omnicore_Customer/`), et le fichier hôte est nommé d'après le site (par exemple, `hosts/Omnicore_Customer/host_files/Production.yml`).

Pour vérifier l'état de toutes les licences, accédez à https://license_server .

Suppression d'une Licence

Pour supprimer une licence, utilisez `util_playbooks/delete_license.yml`.

Exigences Réseau

Configuration du Pare-feu

Les pare-feu des sites clients doivent être configurés pour autoriser le trafic HTTPS (port 443) vers les serveurs de validation de licences Omnitouch :

Nom d'hôte	Objectif
1.time.omnitouch.com.au	Serveur de validation de licences 1
2.time.omnitouch.com.au	Serveur de validation de licences 2
3.time.omnitouch.com.au	Serveur de validation de licences 3

Règles sortantes requises :

- Protocole : HTTPS (TCP/443)
- Destination : les noms d'hôtes ci-dessus, résolus via DNS
- Direction : Sortant

> **Résoudre par DNS : ne pas coder en dur les IP.** Toujours autoriser ces hôtes par FQDN (ou par les adresses que votre pare-feu résout dynamiquement). Les IP sous-jacentes peuvent changer sans préavis. Les listes d'autorisation d'IP statiques ne sont pas prises en charge et finiront par casser la validation des licences.

Exigences DNS

Le serveur de licences nécessite une résolution DNS fonctionnelle pour communiquer avec l'infrastructure de validation de licences Omnitouch.

Configuration DNS requise :

- Le serveur de licences doit avoir accès à des serveurs DNS publics
- Configurez DNS pour utiliser l'un des suivants :
  - 1.1.1.1 (Cloudflare - prend en charge DNS sécurisé)
  - 8.8.8.8 (Google Public DNS)
- Ne pas utiliser de serveurs DNS internes/corporatifs pour le serveur de licences

Remarque : Les serveurs de licences Omnitouch utilisent DNS sécurisé (DoH/DoT). L'utilisation de serveurs DNS publics garantit une validation correcte de DNSSEC et empêche les problèmes d'interception DNS par des appareils de sécurité.

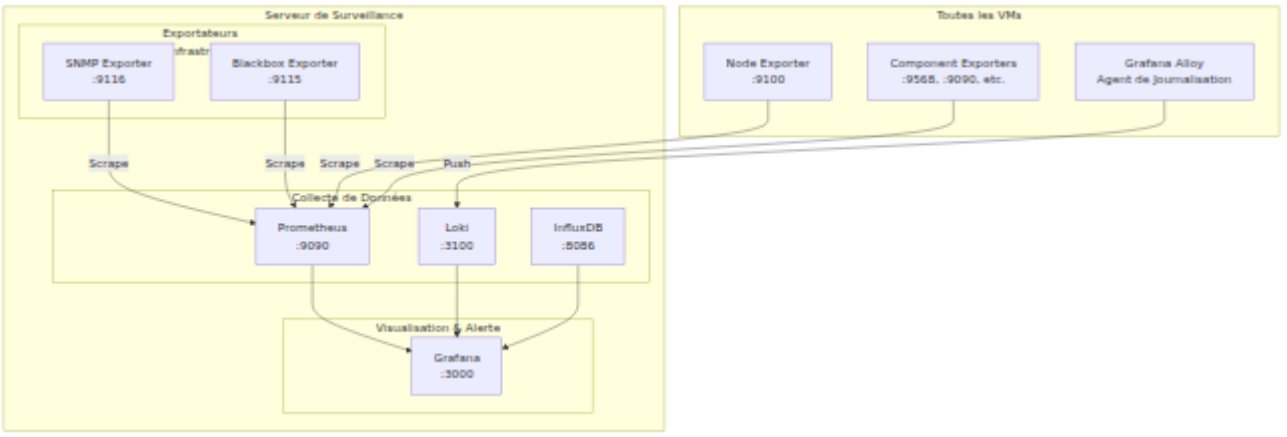
Documentation Connexe

- Référence de Configuration
- Configuration du Fichier Hosts

Surveillance & Observabilité

Aperçu

OmniCore comprend une pile de surveillance et d'observabilité complète qui fournit la collecte de métriques, l'agrégation des journaux, la visualisation et l'alerte. Le système est déployé automatiquement par le rôle de surveillance.



Composants

Composant	Objectif	Port	Rétention des Données
Prometheus	Stockage de métriques en séries temporelles	9090	30 jours ou 5 Go
Loki	Agrégation des journaux	3100	7 jours ou 50 Go
InfluxDB	Métriques RAN (Nokia)	8086	30 jours
Grafana	Visualisation et alerte	3000	N/A
Node Exporter	Métriques d'hôte	9100	N/A
SNMP Exporter	Métriques des dispositifs réseau	9116	N/A
Blackbox Exporter	Probes ICMP/HTTP	9115	N/A

Sources de Données

Prometheus

Prometheus scrape les métriques de tous les composants d'OmniCore toutes les 1 minute.

UID de la Source de Données: `omnicore_prometheus`

Cibles Scrappées

Nom de Job	Groupe d'Inventaire	Port	Métriques
`Node Exporter`	`all`	9100	CPU hôte, mémoire, disq réseau
`MMEs`	`mme`	9568	Comptes de sessions, ta d'attachement/détachen
`HSS`	`hss`	9568	Requêtes d'authentificat recherches d'abonnés
`SGW-C`	`sgw`	9568	Comptes de porteurs, messages GTP-C
`PGW-C`	`pgwc`	9090	Connexions PDN, allocat IP
`UPF Standalone`	`upf`	9090	Comptes de paquets, dé
`CSCF`	`pcscf`, `scscf`, `icscf`	9090	Enregistrements, transactions
`ApplicationServer FreeSWITCH`	`applicationserver`	9090	Comptes d'appels, utilisation de canaux
`DRA`	`dra`	9568	Décisions de routage, st des pairs
`OmniMessage`	`omnimessage`	9568	Livraison de SMS, sessio SMPP
`OmniSS7`	`omniss7`	8080	Métriques de passerelle SS7/SIGTRAN
`KeyDB`	`ocs`	9121	Utilisation de la mémoire opérations/sec

Nom de Job	Groupe d'Inventaire	Port	Métriques
`CGrateS`	`ocs`	2080	Tarification, facturation, statistiques CDR

Exportateurs d'Infrastructure

Exportateur	Cibles	Variable de Configuration
SNMP (MikroTik)	Routeurs/commutateurs	`mikrotik.hosts`
SNMP (iDRAC)	Serveurs Dell	`idrac.hosts`
SNMP (Synology)	Dispositifs NAS	`synology.hosts`
SNMP (SAF)	Liens micro-ondes	`saf.hosts`
SNMP (Cisco)	Commutateurs	`cisco.hosts`
Blackbox ICMP	Tous les hôtes + 8.8.8.8	Automatique
VMware	Clusters vCenter	`vcenter_ip`, `vcenter_password`
Proxmox	Clusters PVE	`proxmoxServers`

Loki

Loki reçoit des journaux des agents Grafana Alloy fonctionnant sur toutes les VMs.

UID de la Source de Données: `omnicore_loki`

Voir Journalisation Centralisée pour la configuration détaillée de Loki/Alloy.

Étiquettes de Journal

Étiquette	Description	Exemple
`hostname`	Nom d'hôte de la VM source	`customer-mme01`
`component`	Type de composant	`mme`, `cscf`, `ocs`
`unit`	Nom de l'unité systemd	`omnimme.service`
`level`	Gravité du journal	`info`, `error`

InfluxDB

InfluxDB stocke des données en séries temporelles pour des cas d'utilisation spécifiques nécessitant une rétention plus longue ou des modèles de requête différents.

UID de la Source de Données: `omnicore_influxdb`

Bases de Données

Base de Données	Objectif	Rétention
`nokia-monitor`	Métriques de performance RAN Nokia	30 jours
`dra`	Statistiques de routage DRA	365 jours
`Omnicharge_TAP3`	Métriques de fichiers TAP en itinérance	90 jours

Tableaux de Bord Grafana

Organisation des Tableaux de Bord

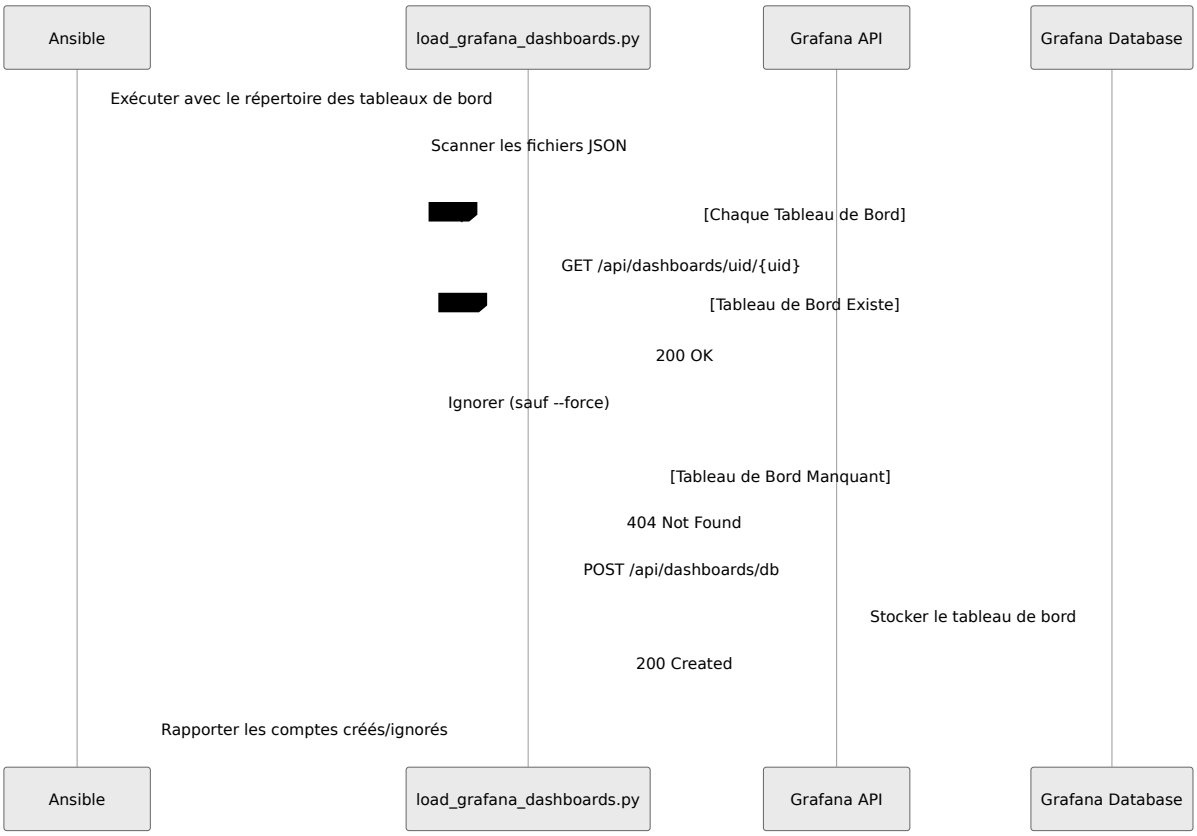
Les tableaux de bord sont organisés en dossiers par domaine fonctionnel :

Dossier	Contenu
BSS	Tableaux de bord CGrateS, KeyDB, de facturation
EPC	Tableaux de bord MME, SGW, PGW, UPF, HSS, DRA
IMS	Tableaux de bord CSCF, Serveur d'Application, OmniMessage
Infrastructure	Node Exporter, SNMP, surveillance réseau
RAN	Tableaux de bord de performance Nokia/OmniRAN
Logs	Visionneuses de journaux spécifiques aux composants

Chargement des Tableaux de Bord (Basé sur API)

Les tableaux de bord sont chargés via l'API Grafana plutôt que par un approvisionnement basé sur des fichiers. Cela permet aux tableaux de bord d'être :

- Édités via l'interface utilisateur de Grafana
- Modifiés via API
- Versionnés dans le dépôt Ansible



Chargement des Tableaux de Bord

Les tableaux de bord sont chargés automatiquement lors du déploiement Ansible. Pour recharger manuellement :

```
# Depuis le contrôleur Ansible
python3 roles/monitoring/files/load_grafana_dashboards.py \
    --url http://<monitoring-ip>:3000 \
    --user admin \
    --password <grafana_admin_password> \
    --dashboards-dir roles/monitoring/templates/grafana/dashboards

# Forcer la mise à jour des tableaux de bord existants
python3 roles/monitoring/files/load_grafana_dashboards.py \
    --url http://<monitoring-ip>:3000 \
    --user admin \
    --password <grafana_admin_password> \
    --dashboards-dir roles/monitoring/templates/grafana/dashboards
\
    --force
```

Fichiers Source des Tableaux de Bord

Les fichiers JSON des tableaux de bord sont stockés dans le dépôt Ansible :

```
roles/monitoring/templates/grafana/dashboards/
├── BSS/
│   ├── KeyDB_Cluster.json
│   ├── cgrates_mysql.json
│   └── cgrates_stats.json
├── EPC/
│   ├── MME_Dashboard.json
���   ├── OmniHSS.json
│   ├── OmniDRA.json
│   ├── SGW.json
│   ├── PGWs.json
│   └── ...
├── IMS/
│   ├── SMSc.json
│   ├── MMSc.json
│   └── ...
├── Infrastructure/
│   ├── Node_Exporter_Full.json
│   ├── MikroTik_Dashboard.json
│   └── ...
├── RAN/
│   ├── Nokia_Overview.json
│   ├── Nokia_Detailed.json
│   └── ...
└── Logs/
    ├── CSCF_Logs.json
    ├── MME_Logs.json
    └── ...
```

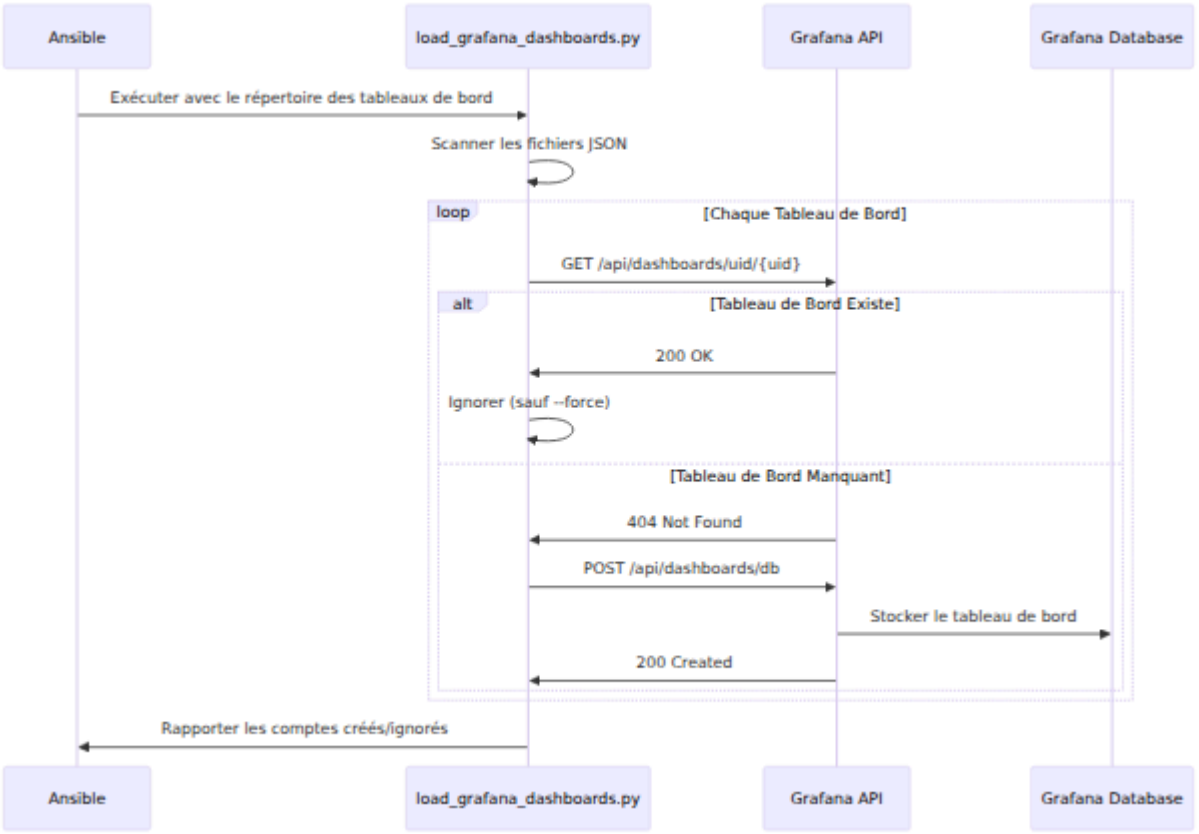
Sauvegarde des Tableaux de Bord

Exporter les tableaux de bord et alertes d'une instance Grafana en cours d'exécution vers le dépôt pour le contrôle de version :

```
# Depuis le répertoire roles/monitoring
python3 export_grafana.py --customer <CUSTOMER>
```

Cela exporte :

- Tous les tableaux de bord vers `roles/monitoring/templates/grafana/dashboards/{folder}/*.json` (partagé entre les clients)
- Règles d'alerte vers `hosts/Omnicore_{CUSTOMER}/group_vars/grafana/alerts/all_alert_rules.json`
- Points de contact vers `hosts/Omnicore_{CUSTOMER}/group_vars/grafana/alerts/contact_points.json`
- Politiques de notification vers `hosts/Omnicore_{CUSTOMER}/group_vars/grafana/alerts/notification_policies.json`



Comportement d'Exportation

- N'écrit que les fichiers qui ont changé (ignorant les champs volatils comme `version` et `id`)
- Préserve la structure des dossiers correspondant à l'organisation de Grafana
- Mappe les titres des tableaux de bord à des noms de fichiers cohérents

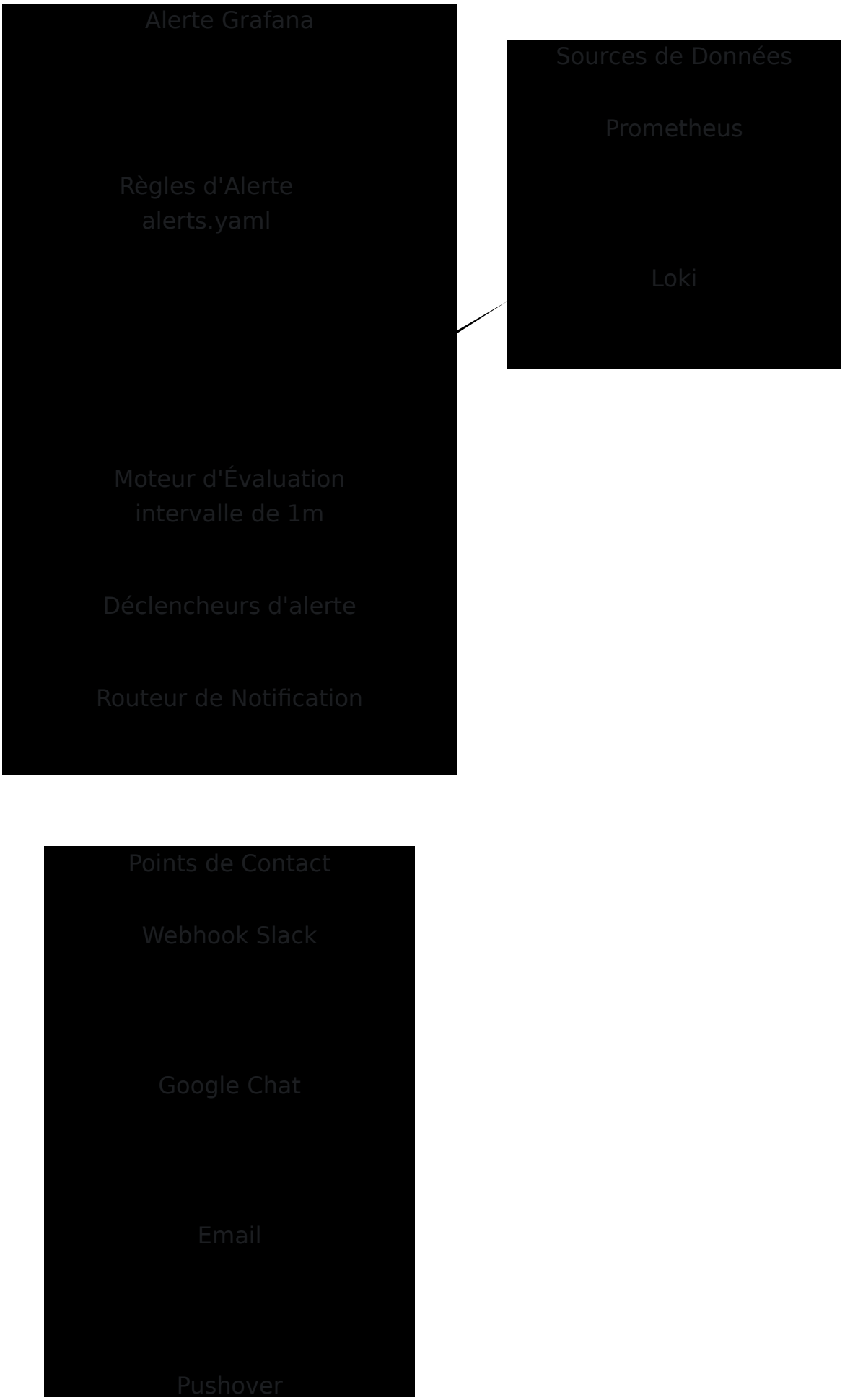
Tableaux de Bord Personnalisés

Les tableaux de bord spécifiques aux clients peuvent être placés dans `group_vars/grafana/dashboards/` et seront chargés aux côtés des tableaux de bord standard :

```
hosts/customer/group_vars/
└── grafana/
    └── dashboards/
        ├── Custom_Dashboard_1.json
        └── Custom_Dashboard_2.json
```

Alerte

Architecture


Alerte Grafana
Règles d'Alerte
alerts.yaml
Moteur d'Évaluation
intervalle de 1m
Déclencheurs d'alerte
Routeur de Notification
Sources de Données
Prometheus
Loki
Points de Contact
Webhook Slack
Google Chat
Email
Pushover

Chargement des Alertes

Les alertes sont chargées via l'API Grafana à partir des exports par client plutôt que d'un seul fichier YAML statique. Au moment du déploiement, le fichier d'approvisionnement d'alerte statique (`/etc/grafana/provisioning/alerting/alerts.yaml`) est supprimé afin que les alertes puissent être éditées via l'UI/API.

Le chemin principal charge les règles d'alerte, les points de contact et les politiques de notification par client à partir de `hosts/Omnicore_<CUSTOMER>/group_vars/grafana/alerts/` en utilisant `load_grafana_from_export.py` :

```
python3 roles/monitoring/files/load_grafana_from_export.py \
    --url http://<monitoring-ip>:3000 \
    --user admin \
    --password <grafana_admin_password> \
    --alerts-dir
hosts/Omnicore_<CUSTOMER>/group_vars/grafana/alerts \
    --dashboards-dir roles/monitoring/templates/grafana/dashboards
\
    --force
```

Si aucune alerte spécifique au client n'existe, le rôle revient à charger le modèle de rôle `alerts.yaml` via `load_grafana_alerts.py` :

```
python3 roles/monitoring/files/load_grafana_alerts.py \
    --url http://<monitoring-ip>:3000 \
    --user admin \
    --password <grafana_admin_password> \
    --alerts-file
roles/monitoring/templates/grafana/provisioning/alerting/alerts.yaml
```

Règles d'Alerte par Défaut

Alerte	Dossier	Condition	Pendant la Durée
Espace Disque 90% Utilisé	Infrastructure	Système de fichiers racine > 90%	5 minutes
CPU au-dessus de 90%	Infrastructure	Utilisation CPU > 90%	5 minutes
Charge Système supérieure à 90%	Infrastructure	Moyenne de charge > 90%	5 minutes
Hôte Hors Ligne	Infrastructure	Cible Prometheus injoignable	5 minutes
Abonnés MME Droppés 40%	EPC	Le nombre de sessions chute de 40%	30 secondes
0 Abonnés sur MME	EPC	Aucun abonné attaché	5 minutes
Abonnés IMS en baisse de 40%	IMS	Les enregistrements P-CSCF chutent de 40%	30 secondes
MOS Moyen en dessous de 4	IMS	Qualité vocale dégradée	5 minutes
Nombre d'eNodeB Droppé	RAN	eNB connectés diminués	1 minute
Délai de Réponse Diameter Élevé	EPC	Pic de latence P95	5 minutes

Structure des Règles d'Alerte

Règles d'alerte dans `alerts.yaml` :

```yaml
apiVersion: 1
groups:
  - orgId: 1
    name: Groupe d'Alerte
    folder: Infrastructure
    interval: 1m
    rules:
      - uid: alert-lowDiskSpace
        title: Espace Disque 90% Utilisé sur l'hôte
        condition: C
        data:
          - refId: A
            datasourceUid: omnicore_prometheus
            model:
              expr: |
                100 - ((node_filesystem_avail_bytes{job="Node
Exporter",mountpoint="/"} * 100)
                / node_filesystem_size_bytes{job="Node
Exporter",mountpoint="/"})
          # ... expressions de seuil
        noDataState: OK
        execErrState: Error
        for: 5m
        annotations:
          summary: L'espace disque a dépassé 90% sur l'hôte
        labels:
          type: resource
```

Configuration des Points de Contact

Les points de contact sont configurés via des variables d'inventaire :

```yaml
# Dans group_vars/all.yml ou vars spécifiques au client
monitoring_data:
  contactPoints:
    - orgId: 1
      name: default
      receivers:
        # Intégration Slack
        - uid: slack-alerts
          type: slack
          disableResolveMessage: false
          settings:
            url: "https://hooks.slack.com/services/xxx/yyy/zzz"

        # Intégration Google Chat
        - uid: gchat-alerts
          type: googlechat
          disableResolveMessage: false
          settings:
            url:
"https://chat.googleapis.com/v1/spaces/xxx/messages?key=yyy"
```

Les types de points de contact pris en charge incluent Slack, Google Chat, Email et Pushover (voir `roles/monitoring/templates/grafana/contact-points.yaml.j2`). Pushover est configuré via un bloc `notifications.pushover` (avec `enabled`, `api_token` et `user_key`).

Politiques de Notification

Les politiques de notification acheminent les alertes vers les points de contact appropriés :

```yaml
# templates/grafana/notification-policies.yaml.j2
apiVersion: 1

policies:
  - orgId: 1
    receiver: default
    group_by: ['grafana_folder', 'alertname']
    group_wait: 30s
    group_interval: 5m
    repeat_interval: 4h
```

Référence de Configuration

Configuration de Prometheus

Située à `/etc/prometheus/prometheus.yml` sur le serveur de surveillance.

Paramètre	Par Défaut	Description
`scrape_interval`	1m	À quelle fréquence scrapper les cibles
`evaluation_interval`	1m	À quelle fréquence évaluer les règles d'enregistrement
`scrape_timeout`	50s	Délai d'attente pour les requêtes de scrape

Configuration de Grafana

Située à `/etc/grafana/grafana.ini` sur le serveur de surveillance.

Paramètres clés configurés par le rôle de surveillance :

Paramètre	Valeur	Description
`allow_embedding`	true	Activer l'intégration dans des iframes
`provisioning`	`/etc/grafana/provisioning`	Répertoire d'approvisionnement

Le mot de passe admin n'est pas défini dans `grafana.ini` (la ligne `admin_password` est laissée commentée). Au lieu de cela, il est appliqué via la variable `grafana_admin_password`, qui est transmise aux tâches de chargement API (chargeurs de tableaux de bord/d'alerte) lors du déploiement.

Alerte par Email (SMTP)

L'alerte par email nécessite que les identifiants SMTP soient définis dans `group_vars` (voir `roles/monitoring/templates/grafana/grafana.ini`) :

Variable	Requise	Par Défaut	Description
`grafana_smtp_user`	Oui	*(aucun)*	Nom d'utilisateur SMTP. Pas de valeur par défaut. Si non défini, le déploiement échoue bruyamment plutôt que d'envoyer un secret.
`grafana_smtp_password`	Oui	*(aucun)*	Mot de passe SMTP. Pas de valeur par défaut. Si non défini, le déploiement échoue.
`grafana_smtp_host`	Non	`in-v3.mailjet.com:587`	Hôte:port du serveur SMTP.
`grafana_smtp_enabled`	Non	`true`	Indique si SMTP est activé.

L'`from_address` et le `from_name` sont dérivés de `customer_legal_name` (par exemple, `alerts+<customer_legal_name>@omnitouch.com.au` et `Omnitouch Alerts - <customer_legal_name>`).

Configuration de Loki

Voir Journalisation Centralisée pour les paramètres de rétention et de stockage de Loki.

Configuration de InfluxDB

Située à `/etc/influxdb/influxdb.conf` sur le serveur de surveillance.

Base de Données	Politique de Rétention	Durée
`nokia-monitor`	`autogen`	30 jours
`dra`	`autogen`	365 jours
`Omnicharge_TAP3`	`autogen`	90 jours

Ports de Service

Service	Port	Protocole	Description
Grafana	3000	HTTP	Interface et API du Tableau de Bord
Prometheus	9090	HTTP	Stockage et requête de métriques
Loki	3100	HTTP	Ingestion et requête de journaux
InfluxDB	8086	HTTP	API InfluxDB
Node Exporter	9100	HTTP	Métriques d'hôte
SNMP Exporter	9116	HTTP	Proxy de métriques SNMP
Blackbox Exporter	9115	HTTP	Probes de métriques
Alloy	12345	HTTP	Interface et métriques Alloy

Opérations Courantes

Visualiser les Métriques

1. Ouvrir Grafana à `http://<monitoring-ip>:3000`
2. Naviguer vers **Tableaux de Bord** et sélectionner le dossier approprié
3. Utiliser le sélecteur de plage horaire pour ajuster la fenêtre de visualisation

Rechercher des Journaux

1. Ouvrir Grafana à `http://<monitoring-ip>:3000`

2. Aller à **Explorer** et sélectionner la source de données **Loki**
3. Utiliser des requêtes LogQL :

```
# Tous les journaux d'un hôte spécifique
{hostname="customer-mme01"}

# Erreurs de tous les composants MME
{component="mme"} |~ "(?i)error"

# Rechercher un IMSI spécifique
{component="hss"} |= "123456789012345"
```

Créer des Alertes Personnalisées

Le flux de travail recommandé est de créer/éditer des alertes dans l'interface utilisateur de Grafana, puis de les exporter vers le répertoire `group_vars/grafana/alerts/` du client (voir Sauvegarde des Tableaux de Bord) afin qu'elles soient contrôlées par version et réappliquées lors du prochain déploiement.

Pour charger manuellement des alertes spécifiques au client :

```
python3 roles/monitoring/files/load_grafana_from_export.py \
    --url http://<monitoring-ip>:3000 \
    --user admin --password <password> \
    --alerts-dir
hosts/Omnicore_<CUSTOMER>/group_vars/grafana/alerts \
    --dashboards-dir roles/monitoring/templates/grafana/dashboards
\
    --force
```

Le modèle de rôle `roles/monitoring/templates/grafana/provisioning/alerting/alerts.yaml` n'est utilisé qu'en tant que solution de secours lorsqu'un client n'a pas d'alertes exportées.

Sauvegarder les Tableaux de Bord

Après avoir effectué des modifications dans l'interface utilisateur de Grafana, exporter vers le dépôt :

```
cd roles/monitoring
python3 export_grafana.py --customer <CUSTOMER> --url
http://<monitoring-ip>:3000
git add ../../hosts/Omnicore_<CUSTOMER>/group_vars/grafana/
git commit -m "Mettre à jour les alertes depuis la production"
```

L'argument `--customer` est requis. Ce script exporte les règles d'alerte, les points de contact et les politiques de notification vers le répertoire `group_vars/grafana/alerts/` du client ; il n'exporte plus les tableaux de bord, qui sont partagés entre les clients dans le répertoire de rôle (`roles/monitoring/templates/grafana/dashboards/`).

Dépannage

Cible Prometheus Hors Ligne

Symptômes: Le tableau de bord affiche "Aucune Donnée" ou l'état de la cible montre DOWN

Causes possibles :

- Service non en cours d'exécution sur l'hôte cible
- Pare-feu bloquant le port de l'exportateur
- Résolution de nom d'hôte incorrecte

Résolution :

1. Vérifier si le service est en cours d'exécution sur la cible :

   ```
   systemctl status <service>
   curl http://localhost:<port>/metrics
   ```

2. Vérifier la connectivité depuis le serveur de surveillance :

```
curl http://<target>:<port>/metrics
```

3. Vérifier la page des cibles de Prometheus : `http://<monitoring-ip>:9090/targets`

Tableau de Bord Ne Charge Pas

Symptômes: Le tableau de bord affiche un indicateur de chargement ou une erreur

Causes possibles :

- Échec de la connexion à la source de données
- Délai d'attente de requête
- Corruption JSON du tableau de bord

Résolution :

1. Tester la source de données dans Grafana : **Configuration** → **Sources de Données** → **Tester**
2. Vérifier les journaux de Grafana : `journalctl -u grafana-server -f`
3. Essayer de recharger le tableau de bord depuis le dépôt

Alertes Ne Se Déclenchent Pas

Symptômes: Condition remplie mais aucune notification reçue

Causes possibles :

- Alerte mise en pause
- Mauvaise configuration du point de contact
- Politique de notification non correspondante

Résolution :

1. Vérifier l'état de l'alerte dans Grafana : **Alerte** → **Règles d'Alerte**

2. Vérifier que le test du point de contact fonctionne : **Alerte → Points de Contact → Tester**
3. Examiner le routage de la politique de notification

Grafana Ne Peut Pas Se Connecter à la Source de Données

Symptômes: "Bad Gateway" ou délai d'attente de connexion dans Grafana

Causes possibles :

- Service Prometheus/Loki/InfluxDB hors ligne
- Pare-feu bloquant les connexions localhost
- Service lié à la mauvaise interface

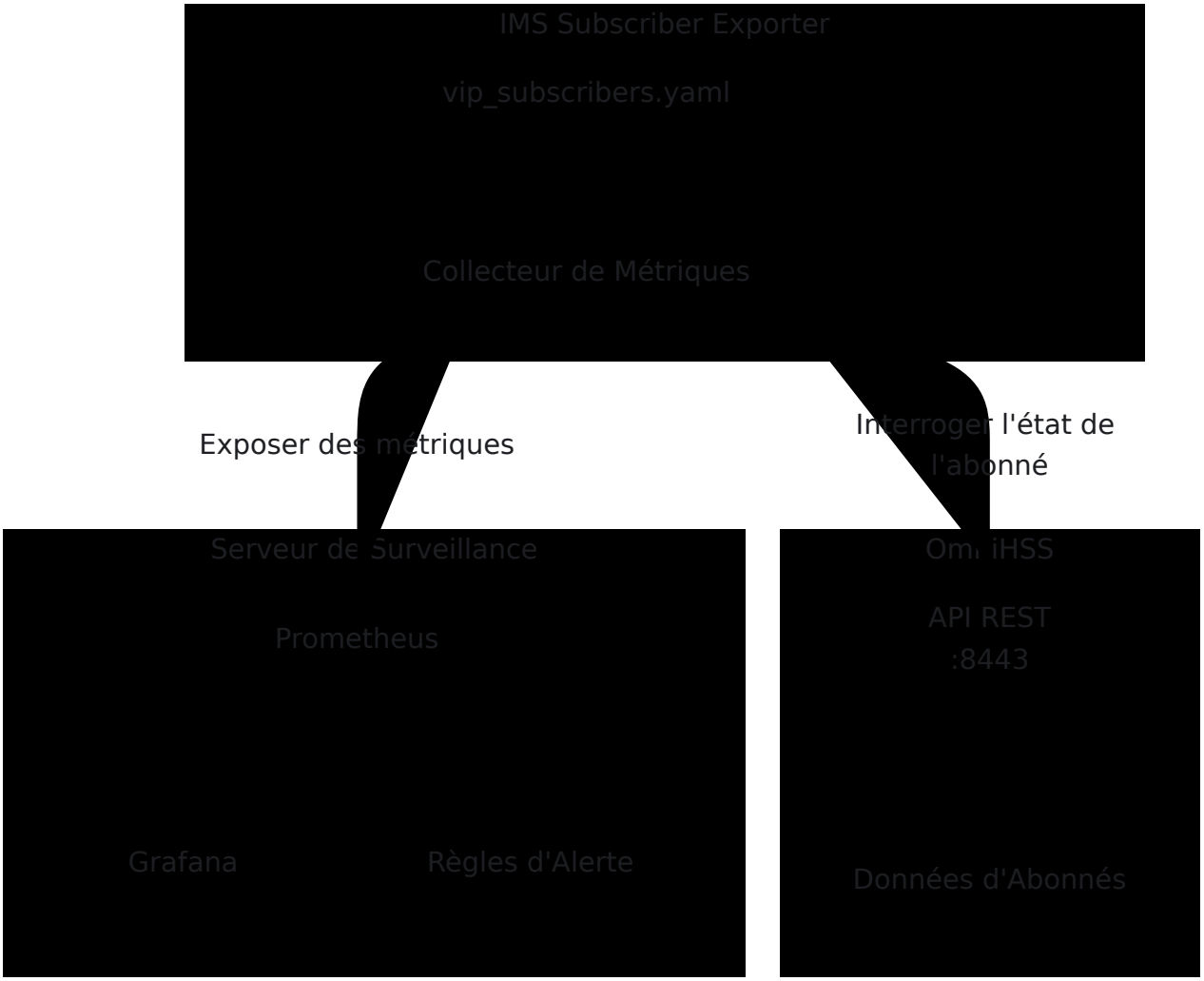
Résolution :

1. Vérifier l'état du service :
   ```
   systemctl status prometheus loki influxdb
   ```
2. Vérifier que le service écoute :
   ```
   ss -tlnp | grep -E '9090|3100|8086'
   ```
3. Tester la connectivité locale :
   ```
   curl http://localhost:9090/api/v1/status/config
   curl http://localhost:3100/ready
   ```

Surveillance des Abonnés VIP

La Surveillance des Abonnés VIP fournit un suivi en temps réel de l'état des abonnés critiques. Le système surveille l'enregistrement IMS, l'attachement EPC et la connectivité UE pour des abonnés prioritaires désignés tels que les services d'urgence, les hôpitaux et les infrastructures clés.

Architecture



Types de Service

Les abonnés sont classés par type de service attendu, ce qui détermine l'évaluation de la santé :

Type	IMS Requis	EPC Requis	Cas d'Utilisation
`full`	Oui	Oui	Abonnés mobiles standard avec voix et données
`voip_only`	Oui	Non	Dispositifs uniquement VoIP (téléphones IP, softphones)
`data_only`	Non	Oui	Dispositifs uniquement de données (routeurs, IoT, M2M)

Évaluation de la santé :

- **full**: Sain lorsque à la fois IMS enregistré (`assigned_scscf` non nul) ET EPC attaché (`last_seen_mme` non nul)
- **voip_only**: Sain lorsque IMS enregistré (`assigned_scscf` non nul)
- **data_only**: Sain lorsque EPC attaché (`last_seen_mme` non nul)

Configuration

Les abonnés VIP sont configurés dans les variables de groupe d'hôtes :

Fichier: `hosts/<customer>/group_vars/vip_subscribers.yaml`

```yaml
vip_subscriber_monitoring:
  hss_url: "https://10.80.12.140:8443"

  # Paramètres de surveillance
  scrape_interval_seconds: 30
  ping_timeout_seconds: 2
  ping_count: 2

  # Abonnés à surveiller
  subscribers:
    # Abonnés à service complet (IMS + EPC)
    - imsi: "001010000000001"
      label: "Services d'Urgence"
      type: "full"

    - imsi: "001010000000002"
      label: "Infrastructure Critique"
      type: "full"

    # Services uniquement de données (routeurs, IoT - aucun IMS
attendu)
    - imsi: "001010000000003"
      label: "Routeur de Site"
      type: "data_only"

    # Services uniquement VoIP (IMS attendu, aucun porteur de
données)
    - imsi: "001010000000004"
      label: "Téléphones IP"
      type: "voip_only"
```

Paramètres de Configuration

Paramètre	Type	Requis	Par Défaut	Descriptio
`hss_url`	Chaîne	Oui	-	URL de l'API REST OmniHSS (HTTPS
`scrape_interval_seconds`	Entier	Non	30	À quelle fréquenc interroger l'état d l'abonné
`ping_timeout_seconds`	Entier	Non	2	Délai d'attente p tests de ping IP U
`ping_count`	Entier	Non	2	Nombre de paqu ping à envoyer
`blackbox_exporter_url`	Chaîne	Non	-	URL de l'exportat blackbox distant les tests de ping exemple, `http://pcscf01:` Utilise une sonde distante au lieu d ping local.
`subscribers`	Liste	Oui	-	Liste des abonné surveiller

Paramètres de l'Abonné

Paramètre	Type	Requis	Par Défaut	Description
`imsi`	Chaîne	Oui	-	IMSI de l'abonné (15 chiffres)
`label`	Chaîne	Non	IMSI	Nom lisible par l'homme pour l'affichage
`type`	Chaîne	Non	`full`	Type de service : `full`, `voip_only` ou `data_only`

Remarque : Le MSISDN est automatiquement récupéré à partir de la réponse de l'API HSS et n'a pas besoin d'être configuré.

Métriques

L'exportateur expose des métriques sur le port 9550 à `/metrics`.

Métriques d'État

Métrique	Type	Description
`vip_subscriber_service_healthy`	Gauge	1 si l'abonné répond aux critères de santé pour son type de service, 0 sinon
`vip_subscriber_ims_registered`	Gauge	1 si l'abonné a un S-CSCF assigné, 0 sinon
`vip_subscriber_epc_registered`	Gauge	1 si l'abonné a une attache MME active, 0 sinon
`vip_subscriber_ue_ip_reachable`	Gauge	1 si l'IP UE répond au ping, 0 sinon
`vip_subscriber_hss_reachable`	Gauge	1 si l'API HSS a renvoyé des données valides, 0 sinon
`vip_subscriber_enabled`	Gauge	1 si le compte de l'abonné est activé dans HSS, 0 sinon

Métriques d'Âge

Métrique	Type	Description
`vip_subscriber_ims_registration_age_seconds`	Gauge	Secondes depuis le dernier enregistrement IMS (-1 si non enregistré)
`vip_subscriber_epc_registration_age_seconds`	Gauge	Secondes depuis la dernière mise à jour de localisation EPC (-1 si non attaché)

Métrique d'Info

Métrique	Type	Description
`vip_subscriber_info`	Gauge	Toujours 1, porte tous les états et informations comme étiquettes

Étiquettes sur toutes les métriques :

Étiquette	Description	Exemple
`imsi`	IMSI de l'abonné	`001010000000002`
`label`	Nom lisible par l'homme	`Infrastructure Critique`
`msisdn`	Numéro de téléphone (provenant de HSS)	`61400000002`
`type`	Type de service	`full`, `voip_only`, `data_only`

Étiquettes supplémentaires sur `vip_subscriber_info` :

Étiquette	Description	Exemple
`healthy`	État de santé du service	`1` ou `0`
`ims`	État d'enregistrement IMS	`1` ou `0`
`epc`	État d'attachement EPC	`1` ou `0`
`ping`	Connectivité IP UE	`1` ou `0`
`assigned_scscf`	Nom d'hôte S-CSCF	`scscf01.ims.example.com`
`last_seen_mme`	Nom d'hôte MME	`mme02.epc.example.com`
`ue_ip`	Adresse IP assignée à l'UE	`100.72.83.20`

Exemples de Requêtes

```
# Nombre d'abonnés VIP sains
count(vip_subscriber_service_healthy == 1)

# Nombre d'abonnés VIP non sains
count(vip_subscriber_service_healthy == 0)

# État d'enregistrement IMS pour les services VoIP
vip_subscriber_ims_registered{type=~"voip_only|full"}

# État d'attachement EPC pour les services de données
vip_subscriber_epc_registered{type=~"data_only|full"}

# Abonnés avec enregistrement IMS obsolète (>1 heure)
vip_subscriber_ims_registration_age_seconds > 3600
```

Tableau de Bord

Le tableau de bord **Abonnés VIP IMS** fournit une visibilité en temps réel sur l'état des abonnés VIP.

Emplacement: Grafana → IMS → Abonnés VIP IMS

URL du Tableau de Bord: `/d/ims-vip-subscribers/`

Panneaux

Panneau	Description
Services Sains	Nombre d'abonnés répondant aux critères de santé
Services Non Sains	Nombre d'abonnés échouant aux critères de santé
IMS Enregistrés	Nombre d'abonnés avec enregistrement IMS actif
EPC Enregistrés	Nombre d'abonnés avec attachement EPC actif
UE Joignable	Nombre d'abonnés avec IP UE pingable
Total Surveillés	Nombre total d'abonnés VIP configurés
État des Abonnés VIP	Tableau montrant tous les abonnés avec colonnes d'état
Santé du Service au Fil du Temps	Chronologie d'état montrant l'historique de santé

Alerte

Les règles d'alerte se déclenchent lorsque les services des abonnés VIP deviennent non sains.

Service d'Abonné VIP Non Sain

Alerte: `Service d'Abonné VIP Non Sain` **Gravité**: Critique **Pendant**: 1 minute **Condition**: `vip_subscriber_service_healthy == 0`

Annotations:

- **Résumé**: `L'abonné VIP {{ $labels.label }} est non sain`

- **Description**: Inclut le type de service et l'identifiant approprié :
  - Services VoIP : Nom et MSISDN
  - Services de données : Nom et IMSI
  - Services complets : Nom, MSISDN et IMSI

Exemples de descriptions d'alerte :

- `Le service Téléphones IP (voip_only) est HORS LIGNE. MSISDN: 61400000001`
- `Le service Routeur de Site (data_only) est HORS LIGNE. IMSI: 001010000000003`
- `Le service Infrastructure Critique (full) est HORS LIGNE. MSISDN: 61400000002 IMSI: 001010000000002`

Ajout de Nouveaux Abonnés VIP

1. Modifier le fichier de configuration :

```
# hosts/<customer>/group_vars/vip_subscribers.yaml
subscribers:
  - imsi: "123456789012345"
    label: "Nouvel Abonné VIP"
    type: "full"  # ou voip_only, data_only
```

2. Déployer la configuration mise à jour :

```
scp vip_subscribers.yaml <monitoring-server>:/tmp/
ssh <monitoring-server> "sudo cp /tmp/vip_subscribers.yaml
/etc/prometheus/vip_subscribers.yaml && sudo systemctl restart
ims-subscriber-exporter"
```

3. Vérifier que le nouvel abonné apparaît dans les métriques :

```
curl -s http://<monitoring-server>:9550/metrics | grep "Nouvel
Abonné VIP"
```

Documentation Connexe

- Journalisation Centralisée: Configuration détaillée de Loki et Alloy
- Architecture de Déploiement: Architecture globale du système
- Configuration du Fichier Hosts: Configuration de l'inventaire

Configuration de Netplan

Vue d'ensemble

OmniCore peut configurer automatiquement les interfaces réseau sur les VM déployées en utilisant netplan. Cela est utile pour :

- Configurer l'interface de gestion principale (eth0)
- Ajouter des interfaces secondaires pour des IP publiques, des connexions de peering ou du trafic dédié
- Configurer des routes statiques pour des destinations spécifiques

Activation de la configuration Netplan

Pour activer la configuration automatique de netplan pour un hôte, ajoutez la variable `netplan_config` pointant vers un modèle Jinja2 dans votre dossier `group_vars` :

```
dra:
  hosts:
    <hostname>:
      ansible_host: 10.0.1.100
      gateway: 10.0.1.1
      netplan_config: netplan.yaml.j2
```

Le modèle sera extrait de `hosts/<customer>/group_vars/netplan.yaml.j2`.

Référence de modèle

Il n'existe pas de `netplan.yaml.j2` partagé et canonique. Chaque client crée son propre `netplan.yaml.j2` dans `hosts/<customer>/group_vars/`, et ces modèles divergent - le nom de l'interface principale, le domaine de recherche DNS et la nomination des interfaces secondaires varient selon le client. Lors de l'intégration d'un nouveau client, créez un modèle par client qui correspond à la disposition des interfaces de ce client plutôt que de copier textuellement celui d'un autre client.

Ce qui suit est le modèle d'un client comme **exemple**. Il est illustratif, pas une spécification - d'autres clients diffèrent. Des commentaires sont ajoutés ici pour explication :

```
network:
  version: 2
  ethernets:
    # Interface principale - utilise ansible_host et gateway de
l'inventaire
    eth0:
      addresses:
      - "{{ ansible_host }}/{{ mask_cidr | default(24) }}"
      nameservers:
        addresses:
{% if 'dns' in group_names %}
        # Si cet hôte EST un serveur DNS, utilisez un DNS externe
pour éviter la dépendance circulaire
        - 8.8.8.8
{% else %}
        # Sinon, utilisez les serveurs DNS du groupe 'dns' dans
l'inventaire
{% for dns_host in groups['dns'] | default([]) %}
        - {{ hostvars[dns_host]['ansible_host'] }}
{% endfor %}
{% endif %}
        # Domaine de recherche DNS - spécifique au client (varie
par client ; certains n'en définissent pas)
        search:
        - customer.example
      routes:
      - to: "default"
        via: "{{ gateway }}"

{% if secondary_ips is defined %}
    # Interfaces secondaires - boucle à travers le dictionnaire
secondary_ips de l'inventaire
    # Cet exemple les nomme ens19, ens20, ens21... (18 +
loop.index). C'est
    # spécifique au modèle ; voir "Nommer les interfaces" ci-
dessous.
{% for nic_name, nic_config in secondary_ips.items() %}
    ens{{ 18 + loop.index }}:
      addresses:
      # Le masque provient du mask_cidr par NIC (par défaut à 24)
      - "{{ nic_config.ip_address }}/{{ nic_config.mask_cidr |
default(24) }}"
{% if nic_config.routes is defined %}
```

```
      # Routes statiques pour cette interface - chaque route
utilise la passerelle de cette interface
      routes:
{% for route in nic_config.routes %}
      - to: "{{ route }}"
        via: "{{ nic_config.gateway }}"
{% endfor %}
{% endif %}
{% endfor %}
{% endif %}
```

Points clés :

- `ansible_host` et `gateway` proviennent de l'entrée d'inventaire de l'hôte
- Les serveurs DNS sont récupérés dynamiquement à partir des hôtes du groupe `dns`
- Le domaine de recherche DNS est spécifique au client (varie ; certains clients l'omettent)
- La nomination des interfaces secondaires varie selon le client (certains utilisent `ens19`, `ens20`, ... ; voir ci-dessous)
- Chaque IP secondaire peut avoir son propre masque (`mask_cidr`), passerelle et routes statiques

Configuration de l'interface principale

L'interface principale (eth0) est configurée automatiquement en utilisant :

- `ansible_host` - L'adresse IP
- `gateway` - La passerelle par défaut
- `mask_cidr` - Masque réseau (par défaut à 24)

Les serveurs DNS sont automatiquement définis sur :

- Hôtes dans le groupe `dns` (utilise leurs IP `ansible_host`)
- Revertit à `8.8.8.8` si l'hôte est lui-même un serveur DNS

MTU de lien (`link_mtu`)

Le rôle `common` peut fixer le **link MTU** de l'interface principale indépendamment du modèle netplan, via la variable d'inventaire `link_mtu`. Lorsqu'elle est définie, elle génère un drop-in netplan (`/etc/netplan/999-link-mtu.yaml`) sur l'interface principale de l'hôte et l'applique. Cela existe parce que les VM proviennent du modèle VMware/Proxmox avec le MTU que le groupe de ports source a expédié (parfois jumbo/9000), ce qui ne correspond pas au reste du tissu et crée des trous noirs pour les SCTP/SIP surdimensionnés lorsque PMTUD est cassé.

Le drop-in est délibérément nommé `999-` afin qu'il soit trié **en dernier** : netplan fusionne tous les `/etc/netplan/*.yaml` dans l'ordre lexicographique avec le dernier fichier gagnant par clé, et le provisionnement peut laisser des fichiers concurrents qui définissent `mtu` sur la même interface (un `80-ansible-config.yaml` hérité et le `99-netcfg-vmware.yaml` du moteur VMware). Un drop-in numéroté plus bas serait silencieusement remplacé par ceux-ci.

```yaml
all:
  vars:
    link_mtu: 1500   # fixe le NIC principal de chaque hôte à 1500
```

Définissez-le dans `all.vars` pour l'appliquer à l'échelle du site, ou par hôte pour remplacer. Si `link_mtu` n'est pas défini, le rôle laisse le MTU de l'interface intact.

> **`link_mtu` n'est PAS `mtu`.** La variable `mtu` de niveau supérieur est le MTU de la charge utile GTP/tunnel (par exemple, 1400/1430/1300) consommé par les rôles de cœur de paquet, **pas** un MTU de lien. Définir le NIC physique sur ces valeurs serait incorrect. Utilisez toujours le `link_mtu` dédié pour le MTU de l'interface.

Interfaces secondaires

Pour les hôtes nécessitant des interfaces réseau supplémentaires (IP publiques, peering, etc.), utilisez la configuration `secondary_ips`.

Schéma

```
secondary_ips:
  <logical_name>:
    ip_address: <ip_address>
    mask_cidr: <prefix_length>   # Optionnel - masque par NIC, par
défaut à 24
    gateway: <gateway_ip>
    host_vm_network: <proxmox_bridge>
    vlanid: <vlan_id>
    nic_name: <interface_name>   # Optionnel - remplacement
explicite du nom de l'interface
                                 # (pris en charge par les modèles
qui le respectent)
    routes:                      # Optionnel - routes statiques
via cette interface
      - '<destination_cidr>'
      - '<destination_cidr>'
    table_routes:                # Optionnel - avancé, spécifique
au modèle
      - { to: '<cidr>', via: '<gateway>', table: <table_id> }
    routing_policy:              # Optionnel - avancé, spécifique
au modèle
      from: '<source_cidr>'
      via: '<gateway>'
      table: <table_id>
      priority: <priority>       # Optionnel, par défaut à 100
```

> La clé `mask_cidr` définit la longueur de préfixe pour cette interface secondaire. Elle est lue à l'intérieur de l'entrée IP secondaire (`nic_config.mask_cidr`), pas à partir d'un `mask_cidr` de niveau supérieur. Si omise, elle par défaut à `/24`.

> `table_routes` et `routing_policy` permettent le routage par politique/source mais ne sont rendus que par des modèles qui les implémentent. Le routage avancé est spécifique au modèle. Confirmez que le `netplan.yaml.j2` du client respecte ces clés avant de les utiliser.

Nommer les interfaces

La nomination des interfaces secondaires est spécifique au modèle - elle n'est pas standardisée entre les clients :

- Certains clients nomment les interfaces par index : `ens19`, `ens20`, `ens21`, ... (`18 + loop.index`). Cela correspond aux noms d'interface assignés par Proxmox lors de l'ajout de NIC supplémentaires à une VM.
- D'autres clients utilisent un `nic_config.nic_name` explicite lorsqu'il est défini, revenant à un nom basé sur l'index (par exemple, `ens(34 + loop.index)`) sinon.

Si un modèle respecte `nic_name`, vous pouvez fixer une interface secondaire à un nom spécifique en définissant `nic_name` sur cette entrée IP secondaire ; sinon, le schéma basé sur l'index du modèle s'applique.

Exemple de configuration

```yaml
dra:
  hosts:
    <hostname>:
      ansible_host: 10.0.1.100
      gateway: 10.0.1.1
      host_vm_network: "ovsbr1"
      vlanid: "100"
      netplan_config: netplan.yaml.j2
      secondary_ips:
        public_ip:
          ip_address: 192.0.2.50
          mask_cidr: 28
          gateway: 192.0.2.1
          host_vm_network: "vmbr0"
          vlanid: "200"
          routes:
            - '198.51.100.0/24'
            - '203.0.113.0/24'
        peering_ip:
          ip_address: 172.16.50.10
          gateway: 172.16.50.1
          host_vm_network: "ovsbr2"
          vlanid: "300"
          routes:
            - '172.17.0.0/16'
```

Sortie Netplan générée

Rendu avec le modèle d'exemple, la configuration ci-dessus génère (le domaine de recherche DNS et la nomination `ens19`/`ens20` sont spécifiques au client) :

```
network:
  version: 2
  ethernets:
    eth0:
      addresses:
      - "10.0.1.100/24"
      nameservers:
        addresses:
        - 10.0.1.53
        search:
        - customer.example
      routes:
      - to: "default"
        via: "10.0.1.1"
    ens19:
      addresses:
      - "192.0.2.50/28"
      routes:
      - to: "198.51.100.0/24"
        via: "192.0.2.1"
      - to: "203.0.113.0/24"
        via: "192.0.2.1"
    ens20:
      addresses:
      - "172.16.50.10/24"
      routes:
      - to: "172.17.0.0/16"
        via: "172.16.50.1"
```

L'interface `public_ip` se rend comme `/28` parce que son entrée a défini `mask_cidr: 28`, tandis que `peering_ip` revient à la valeur par défaut de `/24`.

Intégration Proxmox

Lors de l'utilisation du playbook `proxmox.yml`, des NIC secondaires sont automatiquement créés sur la VM :

1. **Nouvelles VM** : Les NIC secondaires sont ajoutées lors du provisionnement initial

2. **VM existantes** : Les NIC secondaires sont ajoutées et la VM est redémarrée pour appliquer les changements

La configuration Proxmox utilise :

- `host_vm_network` - Le pont auquel attacher le NIC
- `vlanid` - Tag VLAN pour l'interface

Comment ça fonctionne

1. Les configurations netplan existantes dans `/etc/netplan` sont sauvegardées (`*.bak`) puis supprimées pour éviter les conflits
2. Le modèle Jinja2 du client est rendu dans `/etc/netplan/01-netcfg.yaml` (mode `0600`)
3. `netplan try --timeout 30` applique la configuration avec un retour automatique si la connectivité est perdue
4. `wait_for_connection` confirme que l'hôte est resté accessible
5. `netplan apply` valide la configuration

Cas d'utilisation courants

Diameter Edge Agent (DEA) avec IP publique

```
<hostname>:
  ansible_host: 10.0.1.100           # IP de gestion interne
  gateway: 10.0.1.1
  netplan_config: netplan.yaml.j2
  secondary_ips:
    diameter_roaming:
      ip_address: 192.0.2.50         # IP publique pour les
partenaires de roaming
      gateway: 192.0.2.1
      host_vm_network: "vmbr0"
      vlanid: "200"
      routes:
        - '198.51.100.0/24'          # Réseau partenaire de
roaming
```

PGW avec interface S5/S8

```
<hostname>:
  ansible_host: 10.0.2.20            # IP interne
  gateway: 10.0.2.1
  netplan_config: netplan.yaml.j2
  secondary_ips:
    s5s8_interface:
      ip_address: 203.0.113.17       # IP publique S5/S8
      gateway: 203.0.113.1
      host_vm_network: "vmbr0"
      vlanid: "50"
```

Serveur multi-homé avec réseaux de gestion et de données séparés

```
<hostname>:
  ansible_host: 10.0.1.100           # Réseau de gestion
  gateway: 10.0.1.1
  netplan_config: netplan.yaml.j2
  secondary_ips:
    data_network:
      ip_address: 10.0.2.100         # Réseau de données
      gateway: 10.0.2.1
      host_vm_network: "ovsbr2"
      vlanid: "200"
    backup_network:
      ip_address: 10.0.3.100         # Réseau de sauvegarde
      gateway: 10.0.3.1
      host_vm_network: "ovsbr3"
      vlanid: "300"
```

Référencer les IP secondaires dans les modèles

Vous pouvez référencer les adresses IP secondaires dans d'autres modèles Jinja2 et fichiers de configuration.

Sur le même hôte

Lors de la configuration d'un service sur le même hôte qui a des IP secondaires, vous pouvez référencer directement ou utiliser `inventory_hostname` :

```
# Référence directe (la plus simple)
{{ secondary_ips.diameter_public_ip.ip_address }}

# Ou explicitement via inventory_hostname (même résultat)
{{ hostvars[inventory_hostname]['secondary_ips']
['diameter_public_ip']['ip_address'] }}

# Accéder à d'autres propriétés
{{ secondary_ips.diameter_public_ip.gateway }}
{{ secondary_ips.diameter_public_ip.vlanid }}
```

Depuis un autre hôte

Lorsque vous devez référencer une IP secondaire d'un *autre* hôte (par exemple, configurer une connexion de pair), utilisez `hostvars` avec le nom d'hôte cible :

```
# Référence le premier hôte dans le groupe dra
{{ hostvars[groups['dra'][0]]['secondary_ips']
['diameter_public_ip']['ip_address'] }}

# Boucle à travers tous les hôtes DRA et obtenez leurs IP
publiques
{% for host in groups['dra'] %}
{% if hostvars[host]['secondary_ips'] is defined %}
  - {{ hostvars[host]['secondary_ips']['diameter_public_ip']
['ip_address'] }}
{% endif %}
{% endfor %}
```

Exemple : Configuration de pair DRA

Configurer un pair diameter pour se lier à sa propre IP publique :

```
# Dans dra_config.yaml.j2 - utilisez inventory_hostname pour
l'hôte actuel
peers:
  - name: external_peer
    # Lier à l'IP publique diameter de cet hôte
    local_ip: {{ hostvars[inventory_hostname]['secondary_ips']
['diameter_public_ip']['ip_address'] }}
    remote_ip: 198.51.100.50
    port: 3868
```

Vérifier si des IP secondaires existent

Vérifiez toujours si la variable existe avant de l'utiliser :

```
{% if secondary_ips is defined and
secondary_ips.diameter_public_ip is defined %}
public_ip: {{ secondary_ips.diameter_public_ip.ip_address }}
{% else %}
public_ip: {{ ansible_host }}
{% endif %}
```

Dépannage

Vérifier les noms d'interface

SSH sur la VM et vérifier les noms d'interface :

```
ip link show
```

Sortie attendue pour une VM avec deux interfaces secondaires :

```
1: lo: <LOOPBACK,UP,LOWER_UP> ...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
3: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
4: ens20: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
```

Vérifier la configuration Netplan

```
cat /etc/netplan/01-netcfg.yaml
```

Appliquer Netplan manuellement

```
netplan apply
```

Déboguer Netplan

```
netplan --debug apply
```

Vérifier les routes

```
ip route show
```

Documentation connexe

- Configuration du fichier Hosts - Configuration de l'inventaire des hôtes
- Déploiement Proxmox VM/LXC - Provisionnement de VM
- Référence de configuration - Toutes les variables de configuration

Réplication OmniHSS & Opérations de Nœud

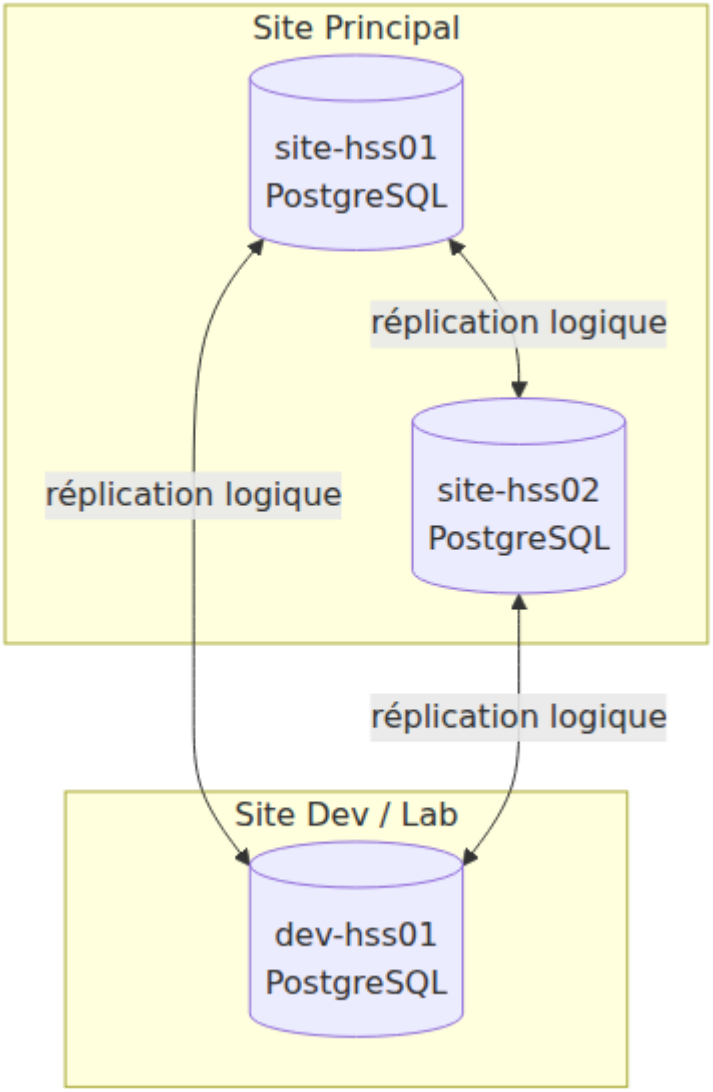
OmniHSS stocke ses données d'abonnés, d'authentification (AuC) et IMS dans une base de données PostgreSQL locale sur chaque nœud. **La réplication logique bidirectionnelle PostgreSQL** maintient plusieurs nœuds OmniHSS synchronisés. Cela forme un **maillage actif-actif** : chaque nœud est un primaire complet en lecture/écriture, et une écriture sur n'importe quel nœud se propage à tous les autres. Il n'y a pas de primaire/de secours. N'importe quel nœud peut servir le trafic S6a, Cx/Dx et Sh et accepter le provisionnement à tout moment.

Ce guide couvre le modèle de réplication. Il explique comment ajouter un nouveau nœud OmniHSS et le charger avec des données. Il explique également comment valider, réparer et supprimer des pairs.

Table des Matières

Aperçu de l'Architecture

Chaque nœud OmniHSS exécute sa propre instance PostgreSQL. Chaque nœud **publie** toutes ses tables d'application et **s'abonne** à la publication de chaque pair. `origin = none` empêche les boucles de réplication. Sur les tables d'état à changement rapide, le maillage résout les conflits d'écriture **dernier écrit gagne (LWW)** en comparant `updated_at`.

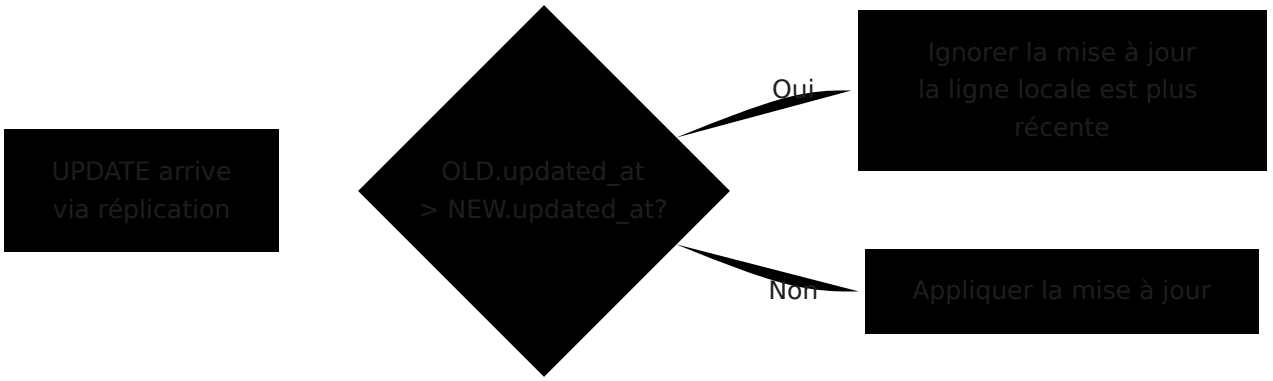


Chaque lien ci-dessus est deux abonnements unidirectionnels (un par direction). Pour un maillage de `N` nœuds, chaque nœud détient `N-1` abonnements. Les pairs consomment `N-1` des emplacements de réplication de sa publication.

Ce Qui Est Répliqué

Catégorie de table	Exemples	Gestion des conflits
Données maîtres / provisionnées	`subscriber`, `key_set` (AuC), `sim`, `msisdn`	Aucun besoin : provisionné d'un seul endroit à la fois
État dynamique	`subscriber_state`, `pdn_session`, `lte_call`, `ims_subscription`	Déclencheur dernier écrit gagne sur `updated_at`
Tenue de livres de migration	`schema_migrations`	**Exclu** de la réplication

La publication couvre chaque table dans le schéma `public` **sauf** `schema_migrations` (chaque nœud exécute ses propres migrations de manière indépendante). Lorsque deux nœuds mettent à jour la même ligne d'état en même temps, les déclencheurs LWW conservent le plus récent `updated_at` plutôt que la mise à jour qui est arrivée en dernier.



Comment les Pairs Sont Découverts

Lorsque la réplication est activée, le rôle OmniHSS construit automatiquement sa liste de pairs à partir de deux sources :

1. **Autres hôtes dans le groupe d'inventaire local `hss`** : les nœuds OmniHSS définis dans le *même* fichier d'inventaire/hôte (par exemple, un couple local `hss01` / `hss02`). Ceux-ci **n'ont pas** besoin d'être listés dans `connected_omnihss` ; l'appartenance au groupe suffit.
2. **La liste `connected_omnihss`** dans le fichier de pairs distants du site : nœuds OmniHSS définis dans un *autre* fichier hôte (autres sites). Les pairs inter-fichiers ne sont découverts qu'ici, car un inventaire n'a aucune connaissance du groupe `hss` d'un autre.

Le rôle filtre les pairs du groupe local pour les hôtes qui sont des participants à la réplication OmniHSS : ceux dont `omnihss.replication.enabled` est `true`. Un nœud dont le drapeau n'est pas `true` n'est pas pris en compte comme un pair de réplication.

Le rôle exclut automatiquement l'hôte actuel de sa propre liste de pairs, de sorte que le même bloc `connected_omnihss` peut être partagé par chaque site sans qu'un nœud essaie de se répliquer à partir de lui-même.

Nommage des Objets

Chaque nœud nomme ses objets de réplication de manière à ce qu'un maillage de n'importe quelle taille ne se heurte jamais :

Objet	Nom	Créé sur	Unique par
Publication	`<self>_pub`	ce nœud	nœud
Abonnement	`sub_from_<source>`	ce nœud (abonné)	(abonné)
Emplacement de réplication	`<subscriber>_from_<source>`	la source (éditeur)	paire (abonné, source)

Le nom de l'emplacement **doit** encoder l'abonné. Considérez ce qui se passe s'il n'encode que la source (le défaut de PostgreSQL, où l'emplacement hérite

du nom de l'abonnement). Deux nœuds s'abonnant à la même source essaieraient tous deux de créer un emplacement du même nom sur cette source. Le deuxième abonnement échouerait alors avec *"l'emplacement de réplication existe déjà"*. L'encodage de l'abonné donne à chaque lien un emplacement globalement unique.

Réconciliation et Sécurité `--limit`

Chaque exécution de rôle **réconcilie** les objets de réplication du nœud par rapport à l'ensemble actuel de pairs valides (groupe local `hss` + `connected_omnihss`, moins soi-même) :

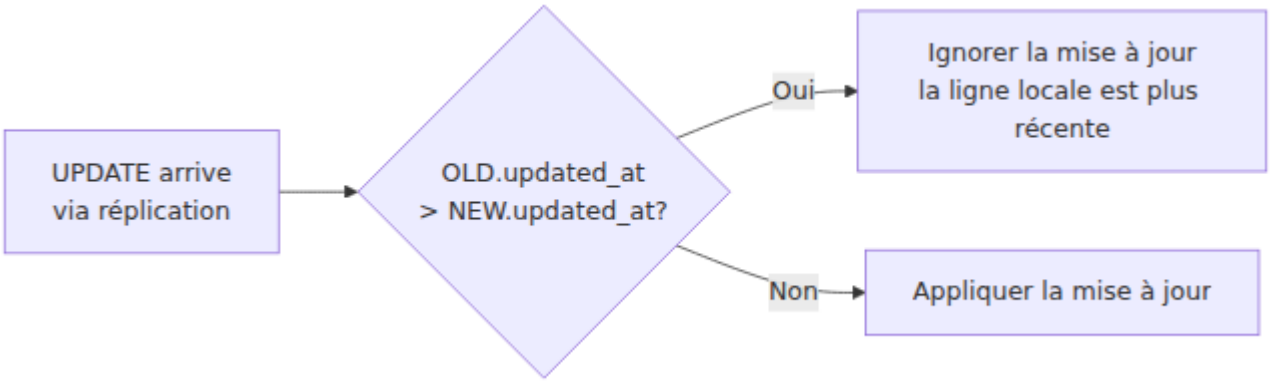
- Les publications autres que `<self>_pub` sont supprimées (nettoie automatiquement les publications obsolètes/renommées).
- Les abonnements dont le pair n'est plus dans l'ensemble valide sont désactivés, détachés de leur emplacement distant et supprimés.
- Les emplacements logiques inactifs qui ne nourrissent pas un pair valide actuel sont supprimés. Les emplacements actifs ne sont jamais touchés.

Cela fait de `connected_omnihss` la liste de maillage **authentique** : la prochaine exécution élimine un pair retiré de celle-ci (et des groupes `hss` pertinents) de chaque nœud. Deux propriétés de sécurité empêchent cela d'être destructeur :

- **`--limit` est sûr.** L'ensemble valide est dérivé de `groups['hss']` et `connected_omnihss`, qui reflètent toujours l'appartenance à l'inventaire *complet* indépendamment de `--limit`. L'exécution de `--limit hss01` ne réconcilie que `hss01`. Mais `hss01` voit toujours `hss02`/`hss03` comme valides et conserve leurs abonnements. `--limit` ne supprime jamais les pairs exclus.
- **Les ensembles vides ne suppriment jamais.** Les étapes de suppression sont complètement ignorées plutôt que d'effacer toute la réplication si l'ensemble de pairs valides est calculé comme vide (par exemple, un inventaire partiel avec un seul HSS et `connected_omnihss` non chargé).

> **Avertissement sur la liste authentique :** parce que la suppression est automatique, l'exécution supprime un pair inter-fichier qui fait réellement

> partie du maillage mais qui est absent de `connected_omnihss`. Gardez toujours `connected_omnihss` complet pour chaque pair inter-fichier.



Configuration

Activation de la Réplication

La réplication est contrôlée par déploiement par le drapeau `omnihss.replication.enabled`. Il est `false` par défaut. Réglez-le dans l'inventaire ou les group_vars du site :

```
omnihss:
  database_type: "postgres"
  database_host: "localhost"
  database_port: 5432
  database_name: "omnihss"
  database_username: "hss"
  database_password: "password"
  replication:
    enabled: true
```

Paramètre	Type	Requis	Par défaut	Descripti
`database_type`	Chaîne	Non	`postgres`	Backend de base données. La répli s'applique unique `postgres`.
`database_host`	Chaîne	Non	`localhost`	Hôte de la base d données. La répli est configurée uniquement lorsc base de données locale (`localhost`
`database_port`	Entier	Non	`5432`	Port PostgreSQL, également utilisé port par défaut p connexions de pa
`database_name`	Chaîne	Non	`omnihss`	Nom de la base d données. Doit correspondre à tr tous les pairs.
`database_username`	Chaîne	Oui	`hss`	Rôle de connexio avec l'attribut `REPLICATION` afir puisse conduire l réplication logiqu
`database_password`	Chaîne	Oui	`password`	Mot de passe pou de connexion. Do correspondre à tr tous les pairs (uti les chaînes de co des pairs).

Paramètre	Type	Requis	Par défaut	Descripti
`replication.enabled`	Booléen	Non	`false`	Interrupteur princ Lorsque `true`, Po est configuré pou réplication logiqu publications/abor sont créés.

Lorsque `replication.enabled` est `true`, le rôle applique les paramètres PostgreSQL suivants sur l'instance locale :

Paramètre	Valeur
`wal_level`	`logical`
`max_wal_senders`	`10`
`max_replication_slots`	`10`
`listen_addresses`	`*`
`max_slot_wal_keep_size`	`10GB` (remplaçable via `omnihss.replication.max_slot_wal_keep_size`)

Paramètre	Valeur

`pg_hba.conf` est mis à jour pour permettre l'IP de chaque pair (`/32`) pour les connexions `all` et `replication`.

Déclaration des Pairs Inter-Sites (`connected_omnihss`)

Les nœuds OmniHSS distants sont déclarés dans le fichier de pairs distants référencé par la variable `remote_peers_file` de chaque site (généralement `prod_master_peers.yml`) :

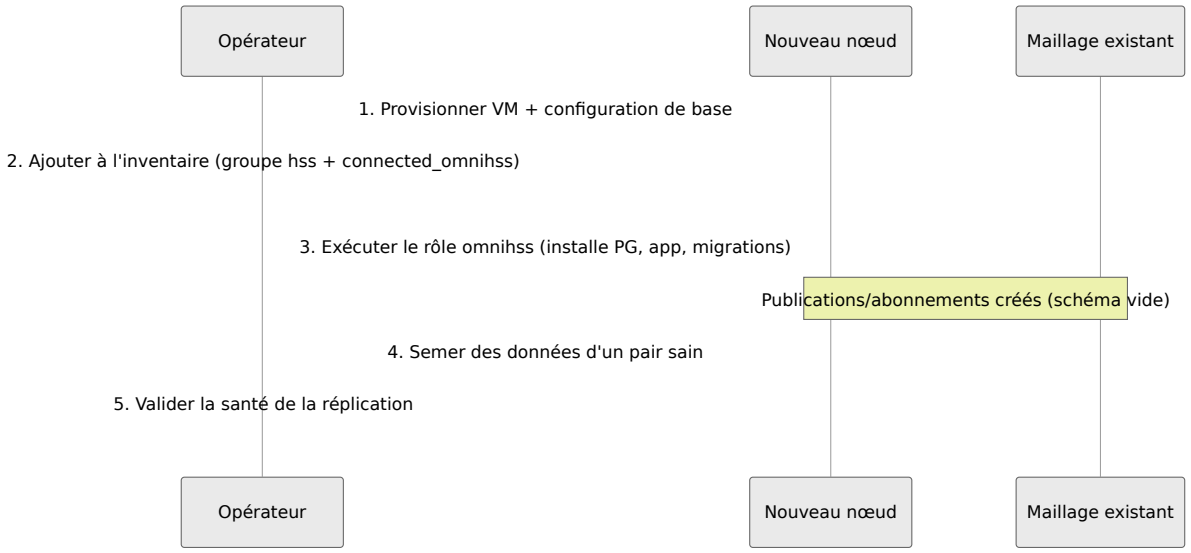
```yaml
connected_omnihss:
  - name: site-hss01
    host: 10.4.2.142
    port: 5432
  - name: site-hss02
    host: 10.4.2.143
    port: 5432
  - name: dev-hss01
    host: 10.4.10.142
    port: 5432
```

Paramètre	Type	Requis	Par défaut	Description
`name`	Chaîne	Oui	-	Le `inventory_hostname` du pair. Cela nomme l'abonnement (`sub_from_<name>`) et résout la publication du pair (`<name>_pub`), donc cela **doit** correspondre au nom d'hôte d'inventaire réel du pair.
`host`	Chaîne	Oui	-	L'adresse IP du pair, accessible depuis ce nœud via le réseau de signalisation.
`port`	Entier	Non	`5432`	Le port PostgreSQL du pair. Retombe sur `database_port`.

Parce que le fichier de pairs distants est chargé par chaque site, la liste complète `connected_omnihss` devrait contenir **chaque** nœud OmniHSS dans le maillage. Chaque site ignore automatiquement ses propres nœuds locaux (ils sont déjà couverts par le groupe local `hss`) et s'ignore lui-même.

Ajout d'un Nouveau Nœud OmniHSS

Ajouter un nœud a deux phases distinctes : **rejoindre le maillage** (afin que les changements circulent dans les deux sens) et **charger les données existantes** sur le nouveau nœud. La deuxième phase est importante car un nœud tout neuf commence avec une base de données vide. Voir Chargement d'un Nouveau Nœud Avec des Données.



Étape 1 : Provisionner l'hôte

Provisionnez la VM et le réseau de base comme pour tout nœud OmniCore (voir Déploiement Proxmox). Le nœud doit atteindre le port PostgreSQL de chaque pair (`5432` par défaut) via le réseau de signalisation, et les pairs doivent pouvoir l'atteindre.

Étape 2 : Ajouter le nœud à l'inventaire

Ajoutez le nouvel hôte au fichier hôte du site sous le groupe `hss:` :

```
hss:
  hosts:
    site-hss01:
      ansible_host: 10.4.2.142
      gateway: 10.4.2.1
      host_vm_network: "v401-omni-signalling"
    site-hss03:        # nouveau nœud
      ansible_host: 10.4.2.144
      gateway: 10.4.2.1
      host_vm_network: "v401-omni-signalling"
```

Ajoutez ensuite ce nœud à `connected_omnihss` dans le fichier de pairs distants afin que les **autres sites** en apprennent également l'existence :

```
connected_omnihss:
  - name: site-hss03
    host: 10.4.2.144
    port: 5432
  # ... entrées existantes ...
```

Étape 3 : Exécuter le rôle OmniHSS

Exécutez le playbook de service OmniHSS contre le nouvel hôte. Cela installe PostgreSQL et OmniHSS, exécute les migrations de base de données (créant un schéma **vide**), configure la réplication logique et crée la publication ainsi qu'un abonnement à chaque pair :

```
ansible-playbook -i hosts/Customer_Name/host_files/Production.yml
\
  services/omnihss.yml --limit site-hss03
```

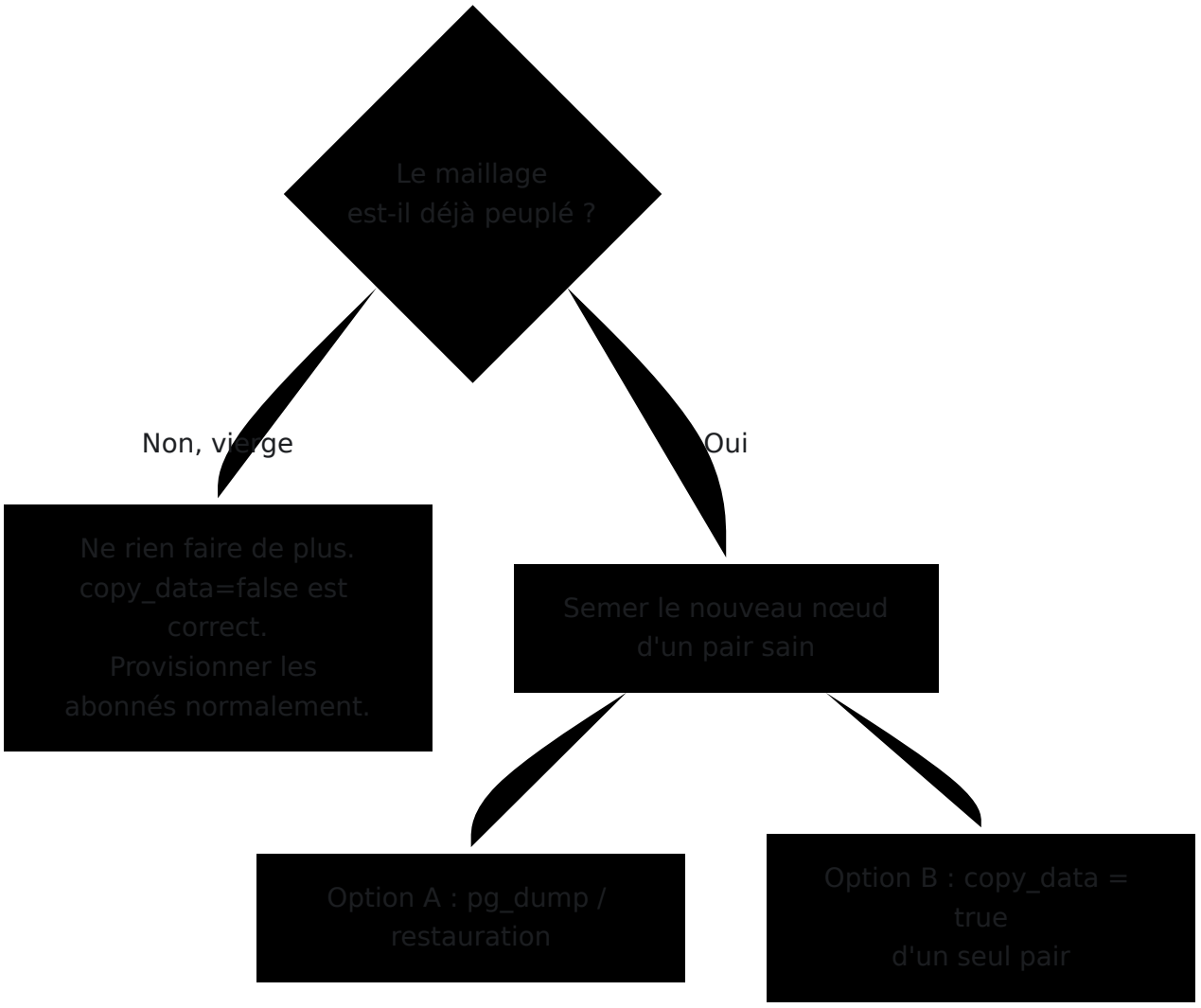
Rerun le rôle sur les pairs existants (ou laissez la prochaine exécution planifiée s'en occuper) afin qu'ils ouvrent `pg_hba.conf` pour le nouveau nœud et créent leur abonnement vers celui-ci. Les abonnements sont idempotents : le rôle détecte et ignore ceux qui existent déjà, et émet `ALTER SUBSCRIPTION ... REFRESH PUBLICATION` afin qu'il prenne en compte les nouvelles tables.

À ce stade, le nouveau nœud est membre du maillage et reçoit **toutes les futures écritures**. Mais il ne contient **pas encore** la base d'abonnés existante. Procédez au chargement de ses données.

Chargement d'un Nouveau Nœud Avec des Données

Le rôle crée les abonnements automatisés avec `copy_data = false`. Cela est correct pour un **maillage vierge** (chaque nœud commence vide en même temps : il n'y a rien à copier, et copier provoquerait des conflits de lignes en double). Mais cela signifie qu'un nœud rejoignant un **maillage déjà peuplé** reste vide sauf pour les changements qui se produisent après son adhésion.

Choisissez le chemin qui correspond à votre situation :



Maillage Vierge

Si vous mettez en place tous les nœuds OmniHSS ensemble sans données préexistantes, vous n'avez pas besoin de semer. Activez la réplication, exécutez le rôle sur tous les nœuds et provisionnez les abonnés normalement. Chaque écriture se réplique à tous les membres.

Option A : Semer via dump et restauration (recommandé)

Prenez un instantané d'un pair sain et restaurez-le sur le nouveau nœud. Ensuite, laissez la réplication transporter les changements en cours. Cela réutilise les outils de sauvegarde/restauration standard et est l'approche la plus prévisible, y compris pour de grands ensembles de données.

1. **Sauvegarder un pair sain.** Cela produit un `pg_dump` de la base de données OmniHSS (réalisé avec `--no-publications --no-subscriptions`, de sorte que les objets de réplication ne soient pas transférés) sur la machine de contrôle :

   ```
   ansible-playbook -i
   hosts/Customer_Name/host_files/Production.yml \
     services/backup.yml --limit site-hss01
   ```

   Le dump est enregistré sous `backups/<Site_Name>/hss_dump_<hostname>_<timestamp>.sql`.

2. **Restaurer sur le nouveau nœud.** Le playbook de restauration arrête OmniHSS, supprime et recrée la base de données, restaure le dump et redémarre le service. Il prend d'abord une sauvegarde avant restauration de la cible :

   ```
   ansible-playbook -i
   hosts/Customer_Name/host_files/Production.yml \
     util_playbooks/restore_hss.yml --limit site-hss03
   ```

Lorsque vous y êtes invité, fournissez le chemin vers le dump SQL de l'étape 1 et laissez le chemin de configuration vide (le nouveau nœud a déjà sa propre configuration provenant de l'exécution du rôle).

3. **Confirmer que la réplication est en cours.** Les abonnements existants du nouveau nœud (`copy_data = false`) appliquent maintenant les changements sur le snapshot restauré. Exécutez la validation ci-dessous.

> **Fenêtre de maintenance :** Il y a une petite fenêtre entre le dump et le moment où la réplication rattrape. Pendant ce temps, les changements de provisionnement sur les pairs ne sont pas encore sur le nouveau nœud. Les données des abonnés principaux changent rarement, et l'état dynamique (`subscriber_state`, `pdn_session`, `lte_call`, `ims_subscription`) se répare à partir du trafic en direct, donc une fenêtre de faible activité est suffisante. Ne provisionnez pas de nouveaux abonnés pendant le semis.

Option B : Semer avec le playbook de semis (recommandé)

Le playbook utilitaire `seed_hss_replication.yml` sème à partir d'un **seul** pair source choisi sans les pièges d'une copie brute en masse. Il recrée l'abonnement en tant que streaming uniquement (`copy_data = false`, pas de tablesync). Il remplit ensuite les lignes existantes de la **table maître** avec un chargement sûr des conflits. `subscriber_state` **fait** partie du remplissage : c'est des données structurelles 1:1 (chaque abonné référence une par FK, créée une fois lors du provisionnement). Le sauter laisse chaque abonné rempli orphelin, ce qui casse l'API et S6a sur la cible. Le streaming ne peut pas guérir cela, car la réplication logique supprime silencieusement les UPDATEs aux lignes qui n'existent pas. Le playbook ne fait délibérément **pas** de copie en masse des tables de session véritablement transitoires (`pdn_session`, `lte_call`). Un `COPY` de ces tables chaudes entre en collision (clé en double) avec des écritures concurrentes sur un système en direct et boucle de crash les tablesync. Elles se remplissent à partir de la réplication en streaming et du trafic en direct à la place.

Interactif (recommandé) : omettez la source. Le playbook trouve alors quels cibles sont vides, interroge les pairs pour ceux détenant des données, et vous invite à choisir :

```
ansible-playbook -i hosts/Customer_Name/host_files/Production.yml
\
  util_playbooks/seed_hss_replication.yml \
  --limit site-hss01,site-hss02
```

```
Pairs avec des données disponibles pour semer :
  1) dev-hss01 (10.4.10.142) - 838 abonnés
Entrez le NUMÉRO de la source à partir de laquelle semer
```

Explicite : nommez directement la source. Cela force un re-semencement même si la cible a déjà des données (utilisé pour la récupération) :

```
ansible-playbook -i hosts/Customer_Name/host_files/Production.yml
\
  util_playbooks/seed_hss_replication.yml \
  --limit site-hss01 \
  -e "seed_source_name=dev-hss01 seed_source_host=10.4.10.142"
```

Voir HSS Replication Seed pour la référence complète des paramètres et du comportement.

Toujours `--limit` aux nœuds cibles, et semez à partir d'exactement une source. Préférez Option A pour de très grands ensembles de données.

Les étapes manuelles équivalentes s'exécutent sur le nœud cible. Créez d'abord l'abonnement de streaming (notez le `slot_name` explicite par lien, voir Nommage des Objets). Ensuite, remplissez les tables maîtres de manière sûre contre les conflits tout en sautant les tables d'état chaudes :

```
-- 1. streamer toutes les tables, pas de copie en masse
DROP SUBSCRIPTION IF EXISTS sub_from_dev_hss01;
CREATE SUBSCRIPTION sub_from_dev_hss01
  CONNECTION 'host=10.4.10.142 port=5432 dbname=omnihss user=hss
password=<omnihss.database_password>'
  PUBLICATION dev_hss01_pub
  WITH (copy_data = false, origin = none,
        slot_name = 'site_hss01_from_dev_hss01');
```

```
# 2. remplir uniquement les tables maîtres (exécuter sur la cible
; -t chaque table sauf
#    schema_migrations, pdn_session, lte_call - subscriber_state
DOIT être
#    inclus, sinon chaque abonné rempli est orphelin : son
#    subscriber_state_id pointe vers une ligne qui n'arrive
jamais, cassant l'
#    API/S6a, et le streaming ne peut pas le guérir car la
réplication logique
#    supprime silencieusement les UPDATEs aux lignes manquantes).
#    Le chargement DOIT être estampillé avec une origine de
réplication : origin=none
#    les abonnements filtrent les changements estampillés à
l'éditeur, donc le remplissage est
#    invisible au maillage. Un chargement non estampillé est
décodé à partir de WAL comme n'importe
#    quelle écriture locale et renvoie les insertions aux pairs
qui détiennent déjà les lignes,
#    bloquant leurs travailleurs d'application par clé en double.
sudo -u postgres psql -d omnihss -c \
  "SELECT pg_replication_origin_create('omnihss_seed_load');"
{ echo "SELECT
pg_replication_origin_session_setup('omnihss_seed_load');"
  PGPASSWORD=<pw> pg_dump -h 10.4.10.142 -U hss -d omnihss \
    --data-only --inserts --on-conflict-do-nothing --disable-
triggers \
    -t subscriber -t subscriber_state -t key_set -t sim -t msisdn
... ; } \
  | sudo -u postgres psql -d omnihss
sudo -u postgres psql -d omnihss -c \
  "SELECT pg_replication_origin_drop('omnihss_seed_load');"
```

Validation de la Réplication

Le rôle exécute automatiquement un contrôle de santé après la configuration, et il peut également être exécuté de manière autonome. La validation vérifie que :

- Chaque abonnement a un processus de travail actif.
- Chaque slot de réplication logique est actif.

- Le retard WAL est inférieur à 100 Mo par slot.
- Le statut WAL de chaque slot est `reserved` ou `extended` (pas `lost`).

Pour vérifier la santé à la demande, relancez le rôle OmniHSS (la validation s'exécute à la fin), ou inspectez directement sur un nœud :

```
# Travailleurs d'abonnement - pid ne devrait pas être nul pour
chacun
sudo -u postgres psql -d omnihss -c \
  "SELECT subname, pid IS NOT NULL AS worker_alive FROM
pg_stat_subscription;"

# Slots de réplication - actif devrait être vrai, wal_status
réservé/étendu
sudo -u postgres psql -d omnihss -c \
  "SELECT slot_name, active, wal_status, \
     pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS lag \
   FROM pg_replication_slots WHERE slot_type = 'logical';"
```

Symptôme	Signification
`worker_alive = false`	Le travailleur d'abonnement est hors service : le pair est inaccessible ou les identifiants/la chaîne de connexion sont incorrects.
`active = false` (slot)	Aucun abonné ne consomme le slot : l'abonnement du pair est hors service ou a été supprimé.
Retard `lag` croissant	Le pair est lent ou le lien est dégradé. Le WAL s'accumule et consomme de l'espace disque.
`wal_status = lost`	Le slot a pris trop de retard et PostgreSQL a supprimé le WAL. Vous devez **re-semer** le pair (voir Charger un nouveau nœud avec des données).

Le rapport de contrôle de santé (`health_check.yml`) comprend également un panneau **HSS Postgres Replication** pour les nœuds avec réplication activée,

montrant l'état du travailleur de chaque abonnement et l'état actif de chaque slot et le statut WAL d'un coup d'œil.

Intégrité des données

L'application de la réplication logique n'impose pas de clés étrangères. Le travailleur d'application fonctionne avec `session_replication_role = replica`, ce qui supprime les déclencheurs RI internes qui mettent en œuvre des contraintes `FOREIGN KEY`. PostgreSQL **ne vérifie donc jamais** une ligne qui arrive par réplication par rapport à son parent. Les contraintes FK vous donnent une intégrité *par nœud*, pas une intégrité *à l'échelle du maillage*.

Dans un maillage actif-actif qui ouvre un trou qu'aucun nœud unique ne peut attraper :

- Le nœud A supprime `epc_profile` X. Son `ON DELETE RESTRICT` passe parce que A n'a pas d'abonné local référencé à X à cet instant.
- Le nœud B (ou une étape de provisionnement) pointe l'abonné 203 vers X. Cela passe parce que X existe toujours sur B.
- Les deux changements se répliquent et s'appliquent sans vérifications FK. Chaque nœud converge sur "203 → X" **et** "X supprimé" : une référence pendante que la FK de aucun nœud n'a jamais violée localement.

Un `subscriber.epc_profile_id` pendante rend l'HSS incapable de construire des données d'abonnement pour cet abonné. L'HSS répond alors `DIAMETER_ERROR_USER_UNKNOWN (5001)` sur ULR/AIR, même si la ligne de l'abonné existe et est activée. La fenêtre schéma-avant-données (ci-dessous) rend cela plus probable, car les transactions ignorées peuvent séparer une création/suppression de parent du repointage de l'enfant.

Détection (pas d'imposition)

L'imposition des FK sur le chemin d'application arrêterait le maillage sur toute anomalie, donc à la place nous **détectons et alarmons**. Chaque nœud installe une fonction `SECURITY DEFINER`, `omnihss_fk_orphan_count()`, qui compte chaque FK à colonne unique dont la valeur pointe vers un parent

manquant dans toutes les tables. `postgres_exporter` la sélectionne, et Grafana alarme lorsqu'elle est supérieure à zéro (voir Metrics).

Exécutez le balayage manuellement à tout moment :

```
sudo -u postgres psql -d omnihss -tAc "SELECT
omnihss_fk_orphan_count();"   # 0 = propre
```

Pour localiser les lignes problématiques pour une référence spécifique (exemple : abonnés pointant vers un profil EPC inexistant) :

```
SELECT s.imsi, s.epc_profile_id
FROM subscriber s
LEFT JOIN epc_profile e ON e.id = s.epc_profile_id
WHERE s.epc_profile_id IS NOT NULL AND e.id IS NULL;
```

Remédiez en repointant la ligne vers un parent valide (une `UPDATE` sur une ligne existante se réplique proprement à travers le maillage) ou en la supprimant.

Prévenir la dérive de schéma

Le déclencheur principal de l'incident qui a motivé cette section était une **migration de schéma appliquée sur un nœud pendant que ses pairs étaient encore sur l'ancien schéma**, puis de nouvelles données écrites contre les nouveaux champs. La source a publié des lignes que les pairs de l'ancien schéma ne pouvaient pas appliquer, causant des erreurs d'application et des **transactions ignorées**, et donc une divergence.

- **Migrez tout le maillage ensemble.** Appliquez les migrations à chaque nœud OmniHSS avant de provisionner contre les nouveaux champs. Ne pas écrire de données au nouveau schéma tant qu'un pair est en retard.
- **Surveillez la dérive.** Le nœud exporte `omnihss_schema_max_version` / `omnihss_schema_migration_count` par nœud. Comparez-les à travers le maillage. Un nœud en retard par rapport à ses pairs est une migration à moitié appliquée.

- **Surveillez les compteurs d'erreurs.** Un `omnihss_subscription_error_apply_error_count` / `_sync_error_count` en hausse signifie que le travailleur d'application échoue (et peut-être ignore). L'alerte "HSS Replication Apply/Sync Errors" se déclenche à ce sujet.

Suppression d'un pair

La suppression d'un pair supprime l'abonnement **de** ce pair et retire son entrée `pg_hba.conf`. Les données déjà répliquées restent ; seuls les futurs changements cessent de circuler. La suppression doit être confirmée explicitement.

Sur le nœud dont vous supprimez le pair *de* :

```
DROP SUBSCRIPTION IF EXISTS sub_from_<peer_name>;
```

Ensuite, retirez la ou les lignes du pair de `pg_hba.conf` et rechargez PostgreSQL :

```
sudo -u postgres psql -c 'SELECT pg_reload_conf();'
```

Le pair en cours de suppression détient toujours son abonnement vers ce nœud. Si le lien doit être complètement rompu, retirez cet abonnement là aussi. Ensuite, retirez le nœud de `connected_omnihss` et du groupe `hss` afin que la prochaine exécution du rôle ne le recrée pas.

Sauvegarde et restauration

La sauvegarde et la restauration d'OmniHSS utilisent des dumps PostgreSQL standard :

Opération	Playbook	Remarques
Sauvegarde	`services/backup.yml`	Produit `hss_dump_<host>_<ts>.s` (base de données) et `hss_<host>_<ts>.tar.gz` (la configuration `/etc/omnihss`) sous `backups/<Site_Name>/`. dump exclut les publications/abonnemen
Restauration	`util_playbooks/restore_hss.yml`	Demande le dump SQL et/ou l'archive de configuration. Arrête OmniHSS, prend une sauvegarde avant restauration, supprime/recrée la base données, restaure et redémarre.

Voir Utility Playbooks pour le flux de travail de sauvegarde/restauration plus large.

Métriques

Chaque nœud OmniHSS exécute `postgres_exporter`, exposant les métriques PostgreSQL (y compris la réplication) à Prometheus. Séries utiles pour la santé de la réplication :

Métrique : `pg_replication_slots_active` **Type** : Gauge **Description** : Si chaque slot de réplication logique a un consommateur actif. **Étiquettes** :

- `slot_name` : le slot de réplication (un par pair abonné)

Métrique : `pg_stat_replication_*` / retard de slot **Type** : Gauge **Description** : Retard de position WAL par slot. Il montre à quel point un pair consommateur est en retard.

Exemples de requêtes :

```
# Tout slot de réplication logique inactif (alerte si > 0)
count(pg_replication_slots_active == 0)

# Les abonnements qui devraient exister par rapport aux
travailleurs en cours d'exécution peuvent être vérifiés
# par rapport à la sortie du playbook de validation lors du
déploiement.
```

Métriques personnalisées OmniHSS

`postgres_exporter` sur chaque nœud exporte également des séries de santé et d'intégrité de réplication spécifiques à OmniHSS via un fichier de requête personnalisé (`roles/omnihss/files/postgres_exporter_queries.yaml`). Prometheus les scrute sous le travail `HSS Postgres`. Elles soutiennent les règles d'alerte dans le groupe **HSS Alerts** (`roles/monitoring/templates/grafana/rules/`) :

Métrique	Type	Si
`omnihss_subscription_worker_alive{subname}`	Gauge	1 si trav d'a ent fon
`omnihss_replication_slot_active{slot_name}`	Gauge	1 si con slot
`omnihss_replication_slot_wal_status_code{slot_name}`	Gauge	0=r 1= 2=r **3=**
`omnihss_replication_slot_lag_bytes{slot_name}`	Gauge	Oct rete slot
`omnihss_subscription_error_apply_error_count{subname}`	Counter	Erre cun trav d'a

Métrique	Type	Si
`omnihss_subscription_error_sync_error_count{subname}`	Counter	Erre cun syn de
`omnihss_integrity_fk_orphans`	Gauge	Ligı une étra per (de
`omnihss_schema_max_version` / `omnihss_schema_migration_count`	Gauge	Mai hau mig app con

Voir Monitoring & Observability pour le pipeline de métriques et les tableaux de bord.

Dépannage

Le nouveau nœud reste vide après avoir rejoint

Symptômes : Les travailleurs de réplication sont actifs et les slots sont actifs, mais le nouveau nœud n'a pas (ou peu) d'abonnés.

Causes possibles :

- Le nœud a rejoint un maillage déjà peuplé et n'a jamais été semé. Les abonnements utilisent `copy_data = false`, donc seuls les changements

post-adhésion arrivent.

Résolution :

1. Semez le nœud à partir d'un pair sain. Voir Charger un nouveau nœud avec des données.

Le travailleur d'abonnement ne fonctionne pas

Symptômes : `pid IS NULL` pour un abonnement dans `pg_stat_subscription`.

Causes possibles :

- Pair inaccessible sur le port PostgreSQL (pare-feu / routage).
- IP du pair manquante dans le `pg_hba.conf` de ce nœud.
- Mauvais identifiants dans la chaîne de connexion (`database_username` / `database_password` différents entre les nœuds).

Résolution :

1. Confirmez la connectivité TCP vers le pair sur `5432`.
2. Vérifiez que le pair apparaît dans `connected_omnihss` (ou le groupe local `hss`) et relancez le rôle pour que `pg_hba.conf` soit mis à jour.
3. Assurez-vous que `database_username`/`database_password` sont identiques sur tous les nœuds.

Le slot de réplication montre `wal_status = lost`

Symptômes : Le `wal_status` d'un slot est `lost` ; le pair correspondant est très en retard.

Causes possibles :

- Le pair était hors service ou inaccessible assez longtemps pour que le WAL requis soit supprimé.

Résolution :

1. Re-semez le pair affecté à partir d'un nœud sain. Voir Charger un nouveau nœud avec des données.

Erreurs de clé dupliquée lors de la copie initiale

Symptômes : Erreurs lors de l'application des changements juste après la création d'un abonnement avec `copy_data = true`.

Causes possibles :

- La copie initiale a été exécutée à partir de plus d'un pair, copiant les mêmes lignes deux fois.

Résolution :

1. Semez à partir de **exactement un** pair avec `copy_data = true` ; gardez tous les autres abonnements à `copy_data = false`.

Mises à jour d'état conflictuelles entre les sites

Symptômes : Une ligne d'état (par exemple `subscriber_state`) semble "flap" entre des valeurs définies à différents sites.

Causes possibles :

- Mises à jour concurrentes de la même ligne à deux sites.

Résolution : Cela est attendu et géré. Le déclencheur de dernier écrit gagne conserve la mise à jour avec le `updated_at` le plus récent. Confirmez que les horloges sont synchronisées (NTP) à travers tous les sites afin que l'ordre `updated_at` ait un sens.

Déploiement de VM/LXC Proxmox

La majorité de nos clients exécutent la pile OmniCore sur Proxmox. Un seul playbook, `util_playbooks/proxmox.yml`, provisionne à la fois des VMs QEMU et des conteneurs LXC en fonction d'un basculement `deployment_type`. Vous pouvez définir le basculement par site (sur l'entrée `proxmoxServers`) et le remplacer par hôte. Cela permet à un site de fonctionner principalement en tant que LXC tandis que des hôtes individuels (par exemple, le UPF) restent des VMs complètes.

Nous supportons toujours VMware, HyperV et le cloud (Vultr / AWS / GCP) pour les déploiements.

Voir aussi :

- Configuration du fichier Hosts : définir les VMs/LXCs à déployer
- Norme de planification IP
- Configuration de Netplan
- Architecture de déploiement

LXC vs VM

Conteneurs LXC :

- Léger, partage le noyau hôte
- Démarrage rapide, faible surcharge
- Isolation limitée
- Ne peut pas exécuter des noyaux ou des modules de noyau personnalisés
- **Pas adapté pour les déploiements EPC/IMS en production**
- **Ne peut pas exécuter UPF** (nécessite des modules de noyau / dispositifs TUN)

VMs (KVM) :

- Virtualisation complète avec noyau dédié
- Isolation complète, exécute des modules de noyau / mise en réseau personnalisée
- Surcharge de ressources plus élevée
- **Recommandé pour la production**
- **Requis pour les déploiements UPF**

Cas d'utilisation :

- **VMs** : sites de production, UPF, toutes les fonctions réseau
- **LXC** : environnements de laboratoire/test, services légers (apt-cache, surveillance, dns)

Choix du type de déploiement

`deployment_type` est résolu pour chaque hôte dans cet ordre :

1. **Par hôte** `deployment_type` (défini sous le nom d'hôte) : l'emporte s'il est présent.
2. **Par site** `deployment_type` sur l'entrée `proxmoxServers` que le round-robin sélectionne pour l'hôte.
3. **Par défaut** `vm` si aucun n'est défini.

Le modèle courant consiste à définir la valeur par défaut du site sur l'entrée `proxmoxServers` (par exemple, `deployment_type: lxc`) et à remplacer uniquement les hôtes qui doivent différer. L'exemple canonique est un laboratoire qui exécute tout en tant que LXC sauf le UPF, qui nécessite des modules de noyau / TUN et doit être une VM :

```yaml
all:
  vars:
    proxmoxServers:
      pve-node-01:
        deployment_type: lxc     # valeur par défaut du site → tout est LXC
        # ...

upf:
  hosts:
    site-upf01:
      ansible_host: 192.168.1.24
      deployment_type: vm         # remplacement par hôte → cet hôte est une VM
```

En tant que convention, gardez la valeur par défaut du site cohérente et ne remplacez que les exceptions. Lorsque vous dispersez des VM et des LXC sur de nombreux hôtes, le placement des nœuds dépend de l'index de l'hôte et est difficile à raisonner.

Le playbook échoue uniquement en cas de collisions de type avec des ressources existantes : si une ressource avec le nom d'hôte cible existe déjà sous le mauvais type (par exemple, un LXC existe mais l'hôte se résout à `deployment_type: vm`, ou vice versa), il s'arrête avec une erreur claire. Voir Type Guards.

Configuration de Proxmox

1. Créer un jeton API

```
# Interface Proxmox : Datacenter → Permissions → Jetons API
# Créer un jeton : root@pam!<TokenName>
# Copier le secret du jeton (affiché une seule fois)
# Décochez la case `Séparation des privilèges`
```

2a. Créer un modèle de VM Cloud-Init (sites VM uniquement)

Exécutez ce script sur l'hôte Proxmox. Il télécharge l'image cloud d'Ubuntu et crée un modèle propre. `proxmox.yml` injecte les identifiants cloud-init et les clés SSH au moment du clonage (voir les notes ci-dessous).

```
#!/bin/bash
set -e

TEMPLATE_ID=9000
IMAGE_URL="https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-amd64.img"
IMAGE="noble-server-cloudimg-amd64.img"

cd /var/lib/vz/template/iso
wget -N "$IMAGE_URL"

qm destroy $TEMPLATE_ID --purge 2>/dev/null || true

qm create $TEMPLATE_ID --name ubuntu-2404-template --memory 2048 --cores 2 --net0 virtio,bridge=vmbr0
qm importdisk $TEMPLATE_ID $IMAGE local-lvm
qm set $TEMPLATE_ID --scsihw virtio-scsi-pci --scsi0 local-lvm:vm-${TEMPLATE_ID}-disk-0
qm set $TEMPLATE_ID --ide2 local-lvm:cloudinit
qm set $TEMPLATE_ID --boot c --bootdisk scsi0
qm set $TEMPLATE_ID --vga std
qm set $TEMPLATE_ID --agent enabled=1
qm template $TEMPLATE_ID
```

Notes :

- Le modèle n'a pas d'utilisateurs ou de mots de passe codés en dur. `proxmox.yml` définit `ciuser`, `cipassword`, et `sshkeys` au moment du clonage.
- Priorité de l'utilisateur cloud-init : `proxmoxTemplateUser` (sur l'entrée `proxmoxServers` correspondante) → première clé `local_users`.
- Priorité du mot de passe cloud-init : `proxmoxTemplatePassword` (sur l'entrée `proxmoxServers` correspondante) → l'utilisateur cloud-init (c'est-à-dire la

valeur choisie ci-dessus).

- Les clés SSH sont injectées à partir de chaque entrée `local_users.*.public_key` (l'authentification par clé est le chemin de connexion prévu).
- Définissez `proxmoxTemplateUser` (et éventuellement `proxmoxTemplatePassword`) sur chaque entrée `proxmoxServers` chaque fois que le modèle a été préparé avec un utilisateur cloud-init qui n'est pas la première entrée `local_users`, afin que le bon compte soit sélectionné.
- `--vga std` garantit que la console web Proxmox fonctionne.
- Le drapeau `-N` sur wget ne télécharge que si le fichier est plus récent que la copie locale.

Alternative : Modèle manuel à partir de l'ISO

Si les images cloud ne sont pas disponibles ou si vous avez besoin d'une installation personnalisée :

Étape 1 : Créer une VM via l'interface Web

- Créer une nouvelle VM → ID de VM 9000, Nom : ubuntu-2404-template
- OS : Télécharger l'ISO d'Ubuntu Server ou utiliser un ISO existant
- Système : Par défaut (Contrôleur SCSI : VirtIO SCSI)
- Disques : 32 Go, Bus : SCSI
- CPU : 2 cœurs
- Mémoire : 2048 Mo
- Réseau : VirtIO, pont vmbr0
- Démarrer la VM et installer Ubuntu Server

Étape 2 : À l'intérieur de la VM - Nettoyer et préparer

```
# Installer cloud-init
sudo apt update
sudo apt install cloud-init qemu-guest-agent -y

# Nettoyer les données spécifiques à la machine
sudo cloud-init clean
sudo rm -f /etc/machine-id /var/lib/dbus/machine-id
sudo rm -f /etc/ssh/ssh_host_*
sudo truncate -s 0 /etc/hostname
sudo truncate -s 0 /etc/hosts

# Effacer l'historique bash et éteindre
history -c
sudo poweroff
```

Étape 3 : Ajouter Cloud-Init et convertir en modèle

- Sélectionner VM → Matériel → Ajouter → CloudInit Drive (sélectionner le stockage par exemple, local-lvm)
- Matériel → Options → Agent invité QEMU → Activer
- Clic droit sur VM → Convertir en modèle
- Ne **pas** définir l'utilisateur/mot de passe cloud-init sur le modèle. `proxmox.yml` injecte ces valeurs au moment du clonage à partir de `proxmoxTemplateUser` / `proxmoxTemplatePassword` (avec un retour en arrière sur `local_users` pour le nom d'utilisateur).

2b. Télécharger le modèle OS LXC (sites LXC uniquement)

```
# Sur le shell du nœud Proxmox :
pveam update
pveam download local ubuntu-24.04-standard_24.04-2_amd64.tar.zst
```

Le chemin du volume résultant (par exemple, `local:vztmpl/ubuntu-24.04-standard_24.04-2_amd64.tar.zst`) est ce que vous configurez comme `proxmoxLxcOsTemplate` ci-dessous.

Configuration du fichier Hosts

Déploiement de VM

```yaml
all:
  vars:
    # deployment_type omis → par défaut "vm"
    proxmoxServers:
      pve-node-01:
        proxmoxServerAddress: 192.168.1.100
        proxmoxServerPort: 8006
        proxmoxApiTokenName: ansible
        proxmoxApiTokenSecret: "your-token-secret-uuid"
        proxmoxNodeName: pve-node-01
        proxmoxTemplateName: ubuntu-2404-template
        proxmoxTemplateId: 9000
        proxmoxTemplateUser: omnitouch        # nom d'utilisateur
cloud-init optionnel ; par défaut à la première clé local_users
        proxmoxTemplatePassword: omnitouch    # mot de passe
cloud-init optionnel ; par défaut au nom d'utilisateur cloud-init
        storage: local-lvm  # optionnel
      pve-node-02:
        # ... configuration du deuxième nœud

    local_users:
      admin_user:
        name: Admin User
        public_key: "ssh-rsa AAAA..."

mme:
  hosts:
    site-mme01:
      ansible_host: 192.168.1.10
  vars:
    proxmox_interface: vmbr0
    gateway: 192.168.1.1
    mask_cidr: 24  # optionnel, par défaut 24
    vlanid: 100  # optionnel
```

Déploiement de LXC

```yaml
all:
  vars:
    proxmox_lxc_nameserver: "1.1.1.1"   # optionnel, par défaut
1.1.1.1
    proxmoxServers:
      pve-node-01:
        deployment_type: lxc
        proxmoxServerAddress: 192.168.1.100
        proxmoxServerPort: 8006
        proxmoxApiTokenName: ansible
        proxmoxApiTokenSecret: "your-token-secret-uuid"
        proxmoxNodeName: pve-node-01
        proxmoxLxcOsTemplate: "local:vztmpl/ubuntu-24.04-
standard_24.04-2_amd64.tar.zst"
        proxmoxLxcDefaultStorage: local-lvm  # optionnel, retour
en arrière rootfs
      pve-node-02:
        deployment_type: lxc
        # ... même forme

    local_users:
      admin_user:
        name: Admin User
        public_key: "ssh-rsa AAAA..."

dns:
  hosts:
    site-dns01:
      ansible_host: 192.168.1.20
  vars:
    proxmox_interface: vmbr0
    gateway: 192.168.1.1
    mask_cidr: 24                        # optionnel, par défaut
24
    vlanid: 100                         # optionnel
    proxmoxLxcCores: 2                  # remplacement optionnel
    proxmoxLxcMemoryMb: 4096            # remplacement optionnel
    proxmoxLxcDiskSizeGb: 30            # remplacement optionnel
    proxmoxLxcRootFsStorageName: local-lvm  # optionnel, remplace
le stockage par défaut proxmoxLxcDefaultStorage du serveur
```

Utilisation

Provisionner (VM ou LXC)

```
ansible-playbook -i hosts/Customer/site.yml
util_playbooks/proxmox.yml
```

Supprimer

```
ansible-playbook -i hosts/Customer/site.yml
util_playbooks/proxmox_delete.yml
```

Le playbook de suppression gère automatiquement à la fois les VMs et les LXCs, indépendamment de `deployment_type`.

Comportement du Playbook

Type Guards

- Si une VM avec le nom d'hôte cible existe mais `deployment_type: lxc` → le playbook échoue avec une erreur claire.
- Cas inverse (LXC existe, site configuré comme VM) → même chose.
- Groupe UPF sur un site LXC → avertissement uniquement (pas un échec dur, mais le UPF ne fonctionnera pas).

Nouvelle ressource

- Répartit les entrées `proxmoxServers` par round-robin selon l'index de l'hôte au sein du groupe, sauf si l'hôte définit `proxmox_server:` pour épingler une entrée spécifique (voir Distribution Across Nodes).
- Interroge l'API Proxmox pour le prochain VMID libre.
- **VM** : clone à partir de `proxmoxTemplateId`, définit l'utilisateur/mot de passe/clé SSH cloud-init (utilisateur cloud-init = `proxmoxTemplateUser` de

l'entrée serveur si défini, sinon la première clé `local_users` ; mot de passe cloud-init = `proxmoxTemplatePassword` si défini, sinon le nom d'utilisateur cloud-init), redimensionne `scsi0`, démarre.

- **LXC** : crée à partir de `proxmoxLxcOsTemplate`, rootfs sur `proxmoxLxcRootFsStorageName` (groupe) → `proxmoxLxcDefaultStorage` (serveur) → `storage` → `local-lvm` (retour final), injecte toutes les `local_users.*.public_key` via `ssh-public-keys`, démarre. Mot de passe root codé en dur à `0mn1Touch@`. Tous les LXCs sont non privilégiés (`unprivileged: 1`) avec le nesting activé (`features: nesting=1`).

Ressource existante

Les chemins de mise à jour pour les VM et LXC diffèrent l'état actuel de l'état désiré et appliquent les changements :

- Configuration réseau (pont, balise VLAN, IP pour LXC)
- Cœurs / mémoire
- Taille du disque (VM `scsi0` / LXC `rootfs`)
- NIC secondaires à partir de `secondary_ips` Redémarre la ressource si le réseau ou les ressources ont changé.

> **OmniUPF :** l'outil traite une entrée `secondary_ips` nommée `n3` (ou `data`/`gtp`) comme un NIC de plan utilisateur dédié. Le rôle `omniupf` lie N3/N9 + XDP à cela et réserve un CPU pour l'interface principale, de sorte qu'une charge de chemin de données lourde ne puisse pas affamer PFCP/API/SSH (qui restent sur l'IP principale).

Étiquetage

Chaque ressource est étiquetée avec ses noms de groupe Ansible + `site_name`.

Distribution à travers les nœuds

Par défaut, l'outil répartit les hôtes à travers les entrées `proxmoxServers` par round-robin, avec la même logique pour les VMs et les LXCs. L'index est la

position de l'hôte **au sein de son groupe de rôle**, pris modulo le nombre d'entrées `proxmoxServers` (dans l'ordre de déclaration) :

```
mme01 → pve-node-01 (0 % 3 = 0)
mme02 → pve-node-02 (1 % 3 = 1)
mme03 → pve-node-03 (2 % 3 = 2)
mme04 → pve-node-01 (3 % 3 = 0)
```

> **Ajouter un deuxième nœud redistribue tout.** Comme le placement est positionnel, passer d'une entrée `proxmoxServers` à deux change où se trouvent les groupes multi-hôtes existants (hôte-0 → entrée-0, hôte-1 → entrée-1, ...). Épinglez les hôtes que vous ne souhaitez pas déplacer (ci-dessous) avant de relancer contre une flotte mixte.

Épingler un hôte à un nœud spécifique

Définissez la clé optionnelle `proxmox_server:` au **nom d'une entrée `proxmoxServers`** pour forcer un hôte sur ce nœud indépendamment de son index de groupe. Lorsqu'elle n'est pas définie, la répartition par round-robin ci-dessus s'applique. Une valeur qui n'est pas une clé `proxmoxServers` échoue l'exécution avec une erreur qui liste les noms valides (donc une faute de frappe ne peut pas silencieusement faire un round-robin ailleurs).

```yaml
device_tester:
  hosts:
    site-device-tester01:
      ansible_host: 10.179.2.50
      proxmox_server: nicklab-prxmx01   # épingler à cette entrée proxmoxServers (→ son nœud)
      # Si le stockage disque de ce nœud diffère de celui du proxmoxLxcRootFsStorageName global,
      # remplacez-le ici aussi (le stockage est lu globalement, pas à partir de l'entrée serveur) :
      proxmoxLxcRootFsStorageName: local-lvm
```

`proxmox_server` peut également être défini au niveau des `vars:` de groupe pour épingler chaque hôte d'un groupe à un nœud.

Remplacements par hôte

`deployment_type`, `proxmox_server` (épinglage de nœud : voir Distribution Across Nodes), et n'importe lequel des réglages de groupe LXC (`proxmoxLxcCores`, `proxmoxLxcMemoryMb`, `proxmoxLxcDiskSizeGb`, `proxmoxLxcRootFsStorageName`, `proxmoxLxcOsTemplate`, `host_vm_network`) peuvent tous être définis par hôte sous le nom d'hôte.

```
dns:
  hosts:
    site-dns01:
      ansible_host: 192.168.1.20
      deployment_type: vm         # remplacement par hôte de la
valeur par défaut du site
      proxmoxLxcDiskSizeGb: 120   # remplacement par hôte
      host_vm_network: vmbr1      # remplacement de pont par hôte
  vars:
    proxmox_interface: vmbr0
    gateway: 192.168.1.1
    mask_cidr: 24  # optionnel, par défaut 24
```

Playbooks de Service

Les playbooks de service déploient et configurent l'infrastructure OmniCore. Situés dans le répertoire `services/`.

Hiérarchie des Playbooks

Les playbooks sont structurés de manière hiérarchique pour permettre des déploiements rapides et ciblés :

```
all.yml
├── setup_users.yml
├── apt_cache.yml          # uniquement lorsque le groupe
apt_cache_servers est défini
├── dns.yml
├── common.yml
├── license_server.yml
├── monitoring.yml
│   └── grafana.yml
├── device_tester.yml
├── epc.yml
│   ├── common.yml
│   ├── omnimme.yml
│   ├── omnisgwc.yml
│   ├── omnipgwc.yml
│   ├── upf.yml
│   ├── omnihss.yml
│   └── omnidra.yml
├── ims.yml
│   ├── pcscf.yml
│   ├── icscf.yml
│   ├── scscf.yml
│   ├── as.yml
│   ├── omnimessage.yml
│   └── omnisep.yml
├── omniepdg.yml
├── omnitwag.yml
├── omniss7.yml
├── ocs.yml
├── crm.yml
├── ran_monitor.yml
├── 5gc.yml
└── ../util_playbooks/health_check.yml
```

Exécutez uniquement ce dont vous avez besoin. Déployer un seul composant (par exemple, `omnimme.yml`) prend quelques secondes. Exécuter `all.yml` sur 50 hôtes prend plus de temps mais gère tout. Utilisez `--limit` pour restreindre davantage la portée.

Référence Rapide

Playbooks de Niveau Supérieur

Playbook	Portée	Description
`all.yml`	Réseau complet	Déploiement complet : utilisateurs, DNS, surveillance, testeur de périphériques, EPC, IMS, ePDG, TWAG, SS7, OCS, CRM, moniteur RAN, cœur 5G, et un dernier contrôle de santé (depuis `util_playbooks/`)
`epc.yml`	Cœur de Paquet	MME, SGW-C, PGW-C, UPF, HSS, DRA
`ims.yml`	Voix/IMS	P/I/S-CSCF, Serveur d'Application, Messagerie, OmniSEP (XCAP/Entitlements/BSF)
`ocs.yml`	Facturation	CGrateS, cluster KeyDB, synchronisation OCS
`monitoring.yml`	Observabilité	Prometheus, Grafana, exporters, HOMER

Playbooks d'Infrastructure

Playbook	Description
`common.yml`	Configuration de base du système d'exploitation, paquets, NTP, agents de journalisation
`setup_users.yml`	Comptes utilisateurs locaux et clés SSH
`dns.yml`	Déploiement du serveur DNS
`license_server.yml`	Serveur de licence OmniCore
`does_netplan_exist.yml`	Vérifiez si netplan est installé sur chaque hôte
`update_mtu.yml`	Définir et appliquer l'MTU de l'interface via netplan
`firewall.yml`	Règles iptables/nftables
`apt_cache.yml`	Miroir APT local (si activé)

Composants EPC

Playbook	Composant	Description
`omnimme.yml`	OmniMME	Entité de Gestion de Mobilité (4G)
`omnisgwc.yml`	OmniSGW-C	Plan de Contrôle de la Passerelle de Service
`omnipgwc.yml`	OmniPGW-C	Plan de Contrôle de la Passerelle PDN
`upf.yml` / `omniupf.yml`	OmniUPF	Fonction de Plan Utilisateur (SGW-U/PGW-U combiné)
`omnihss.yml` / `hss.yml`	OmniHSS	Serveur d'Abonnés à Domicile
`omnidra.yml`	OmniDRA	Agent de Routage Diameter
`omnitwag.yml`	OmniTWAG	Passerelle d'Accès Sans Fil de Confiance
`omniepdg.yml`	OmniEPDG	Passerelle de Données de Paquet Évoluée (Appels WiFi)
`gtp_proxy.yml`	Proxy GTP	Proxy de trafic GTP

Composants IMS

Playbook	Composant	Description
`pcscf.yml`	P-CSCF	Fonction de Contrôle de Session d'Appel Proxy
`icscf.yml`	I-CSCF	CSCF Interrogateur
`scscf.yml`	S-CSCF	CSCF de Service
`as.yml`	OmniTAS	Serveur d'Application de Téléphonie
`omnisep.yml`	OmniSEP	XCAP, Serveur de Droit, BSF, Messagerie Visuelle
`xcap.yml`	OmniSEP	Alias hérité de `omnisep.yml` (déploie le même rôle `omnisep`)
`omnimessage.yml`	OmniMessage / IMS SMSC	Contrôleur SMS-over-IP, SMPP, et le SMSC IMS (déployé via le rôle `cscf` avec `cscf_type: smsc_ims`)

Cœur 5G

Déployé via `5gc.yml`. Chaque fonction réseau est installée par le rôle partagé `5gc_nf` avec un `nf_package` par NF. Voir Cœur 5G pour les détails internes du rôle, la configuration par NF, le verrouillage de version, et les clés HNET/SUCI UDM.

Playbook	Composant	Description
`5gc.yml`	Cœur 5G	Déploie toutes les fonctions réseau 5G Standalone
	OmniNRF	Fonction de Répertoire Réseau
	OmniAMF	Fonction de Gestion d'Accès et de Mobilité
	OmniAUSF	Fonction de Serveur d'Authentification
	OmniBSF	Fonction de Support de Liaison
	OmniCHF	Fonction de Facturation
	OmniNSSF	Fonction de Sélection de Tranche Réseau
	OmniPCF	Fonction de Contrôle de Politique
	OmniSCP	Proxy de Communication de Service
	OmniSMF	Fonction de Gestion de Session
	OmniUDM	Gestion Unifiée des Données
	OmniUDR	Répertoire Unifié des Données

Commuté Circuits

Déployé via `cs.yml` (rôles sous `circuit_switched/`). Voir Cœur Commuté Circuits pour les détails des rôles BSC, MSC, et SS7.

Playbook	Composant	Description
`cs.yml`	Commuté Circuits	Déploie le domaine CS hérité
	BSC	Contrôleur de Station de Base (`circuit_switched/bsc`)
	OmniMSC	Centre de Commutation Mobile (`circuit_switched/omnimsc`)
	SS7	Pile SS7/SIGTRAN (`omniss7` rôle)

Services de Support

Playbook	Description
`ocs.yml`	Système de Facturation en Ligne (CGrateS + KeyDB)
`crm.yml`	Portail de gestion des clients (OmniCRM ; la configuration se trouve maintenant dans `group_vars/`)
`omniss7.yml`	Passerelle SS7/SIGTRAN
`homer.yml`	Capture de paquets SIP/Diameter
`grafana.yml`	Tableaux de bord Grafana et alertes
`ran_monitor.yml`	Intégration de surveillance RAN
`omniroam.yml`	Suite de roaming OmniRoam (TAP3/RAP UI, API Elixir, générateur)
`omnilcs.yml`	Services de Localisation OmniLCS et Diffusion Cellulaire
`device_tester.yml`	Banc d'essai de l'agent de téléphone OmniWeb (Appareils de Test)

> **Provisionnement Proxmox** (`proxmox.yml`, `proxmox_delete.yml`) se trouve sous `util_playbooks/`. Voir Playbooks Utilitaires et Déploiement de VM/LXC Proxmox.

Opérations

Playbook	Description
`backup.yml`	Sauvegarder les bases de données et les configurations
`reboot.yml`	Redémarrage contrôlé des hôtes
`shutdown.yml`	Arrêt en douceur
`apt_update.yml`	Mettre à jour les paquets
`apt_refresh_metadata.yml`	Rafraîchir les métadonnées du cache APT
`speedtest.yml`	Test de débit réseau

Playbooks de Restauration

Playbook	Description
`restore_applicationserver.yml`	Restaurer OmniTAS à partir de la sauvegarde
`restore_smsc.yml`	Restaurer le SMSC à partir de la sauvegarde : Contrôleur OmniMessage, passerelle SMPP, et nœuds IMS SMSc

> Remarque : `health_check.yml`, `restore_hss.yml`, et `restore_ocs.yml` ne sont **pas** dans `services/`. Ils se trouvent dans `util_playbooks/`. Voir Playbooks Utilitaires. `all.yml` exécute le contrôle de santé en tant que `../util_playbooks/health_check.yml`.

Utilisation

Déployer Tout

```
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml
```

Déployer un Sous-Système Spécifique

```
# Juste le cœur de paquet
ansible-playbook -i hosts/customer/host_files/production.yml
services/epc.yml

# Juste IMS/voix
ansible-playbook -i hosts/customer/host_files/production.yml
services/ims.yml
```

Déployer un Composant Unique

```
# Mettre à jour uniquement le MME
ansible-playbook -i hosts/customer/host_files/production.yml
services/omnimme.yml

# Mettre à jour uniquement la surveillance
ansible-playbook -i hosts/customer/host_files/production.yml
services/monitoring.yml
```

Limiter à des Hôtes Spécifiques

```
# Exécuter all.yml mais uniquement sur un hôte
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit mme01

# Exécuter sur plusieurs hôtes spécifiques
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit "mme01,hss01"

# Exécuter sur un groupe
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit mme
```

Documentation Connexe

- Architecture de Déploiement - Flux de travail global de déploiement
- Configuration du Fichier Hosts - Définir l'infrastructure
- Configuration des Variables de Groupe - Personnalisation
- Playbooks Utilitaires - Outils opérationnels (contrôle de santé, restauration, etc.)
- Surveillance & Observabilité - Grafana, Prometheus, alertes

Playbooks Utilitaires

Les playbooks utilitaires fournissent des outils opérationnels pour gérer l'infrastructure OmniCore déployée. Ces playbooks se trouvent dans le répertoire `util_playbooks/` et peuvent être exécutés indépendamment pour effectuer des tâches de maintenance et de dépannage courantes.

Référence Rapide

Playbook	Objectif
`proxmox.yml`	Provisionner des VM et/ou des conteneurs LXC sur Proxmox (selon `deployment_type`)
`proxmox_delete.yml`	Supprimer des VM/LXC sur Proxmox
`proxmox_restart_hung.yml`	Forcer le redémarrage des VM/LXC via l'API Proxmox lorsque SSH ne répond pas
`vmware.yml`	Provisionner des VM sur VMware vSphere
`vmware_delete.yml`	Supprimer des VM sur VMware vSphere
`health_check.yml`	Générer un rapport de santé complet pour tous les services
`restore_hss.yml`	Restaurer la base de données HSS et/ou la configuration à partir d'une sauvegarde
`seed_hss_replication.yml`	Initialiser un nœud OmniHSS vide à partir d'un pair peuplé (chargement initial des données de réplication)
`audit_hss_drift.yml`	Auditer le maillage OmniHSS pour dérive de données (nombre de lignes + résumé de contenu par table de référence sur tous les nœuds) ; en lecture seule
`reconcile_hss_drift.yml`	Réconcilier la dérive de données du maillage OmniHSS en remplissant les lignes manquantes sur les nœuds (exécution à sec par défaut)

Playbook	Objectif
`restore_ocs.yml`	Restaurer OCS KeyDB, MySQL et configuration à partir d'une sauvegarde
`ip_plan_generator.yml`	Générer de la documentation réseau avec des diagrammes Mermaid
`get_ports.yml`	Auditer les ports ouverts et les services à l'écoute sur tous les hôtes
`getLocalCapture.yml`	Récupérer les fichiers de capture de paquets des hôtes
`delete_local_user.yml`	Supprimer un compte utilisateur local de tous les hôtes
`clear_mnesia.yml`	Effacer la base de données Mnesia pour OmniSS7 / OmniMessage et redémarrer (destructif)
`delete_license.yml`	Supprimer le fichier de licence sur le serveur de licence et le redémarrer

Provisionnement de VM

Les playbooks de provisionnement de VM créent et gèrent des machines virtuelles sur des plateformes d'hyperviseur. Tous les playbooks prennent en charge `--limit` pour cibler des hôtes ou des groupes spécifiques.

Voir Déploiement Proxmox pour une configuration détaillée.

Proxmox

Fichiers : `util_playbooks/proxmox.yml`, `util_playbooks/proxmox_delete.yml`

`proxmox.yml` provisionne à la fois des VM et des conteneurs LXC ; qu'un hôte donné devienne une VM ou un LXC est décidé par son `deployment_type` résolu (surcharge par hôte → entrée `proxmoxServers` → défaut `vm`). Voir Déploiement Proxmox.

```
# Provisionner (VMs et/ou conteneurs LXC, selon deployment_type)
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox.yml

# Provisionner uniquement un groupe spécifique
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox.yml --limit mme

# Supprimer des VM/LXC
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox_delete.yml --limit old-host
```

Redémarrage des Hôtes Proxmox Bloqués

Fichier : `util_playbooks/proxmox_restart_hung.yml`

Vérifie la réactivité SSH de chaque hôte et force le redémarrage de toutes les VM ou conteneurs LXC non réactifs via l'API Proxmox. Les VM QEMU sont arrêtées brutalement et redémarrées ; les conteneurs LXC sont démarrés (ou redémarrés s'ils sont déjà en cours d'exécution). Les hôtes qui répondent à SSH sont ignorés.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox_restart_hung.yml

# Restreindre à un seul hôte ou groupe
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox_restart_hung.yml --limit hss01
```

VMware vSphere

Fichiers : `util_playbooks/vmware.yml`, `util_playbooks/vmware_delete.yml`

Les deux playbooks incluent les fichiers de tâche par VM sous `util_playbooks/vm/vmware/` (`create_individual_vm.yml` et `delete_individual_vm.yml`). `vmware_delete.yml` est un mince wrapper autour de la tâche de suppression ; `vmware.yml` attend également SSH après le provisionnement, puis supprime les fichiers netplan obsolètes et applique netplan sur la nouvelle VM.

Prérequis : Nécessite l'installation des dépendances Python (`uv sync`).

Utilisation :

```
# Provisionner des VM
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vmware.yml

# Provisionner uniquement un groupe spécifique
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vmware.yml --limit mme

# Supprimer des VM
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vmware_delete.yml --limit old-host
```

Variables Requises :

```yaml
all:
  vars:
    vcenter_ip: "vcenter.example.com"
    vcenter_username: "administrator@vsphere.local"
    vcenter_password: "password"
    vcenter_datacenter: "Datacenter"
    vcenter_folder: "OmniCore"
    vcenter_vm_template: "ubuntu-2404-template"
    vhosts:
      esxi-01:
        vcenter_cluster_ip: "192.168.1.10"
        vcenter_datastore: "datastore1"
```

Vultr

Fichiers : `util_playbooks/vm/vultr/vmProvisionVultr.yml`, `util_playbooks/vm/vultr/vmTeardownVultr.yml`, `util_playbooks/vm/vultr/vmCommonSetup.yml`

Provisionne et détruit des instances sur Vultr via l'API Vultr. `vmProvisionVultr.yml` crée des instances, `vmTeardownVultr.yml` les supprime, et `vmCommonSetup.yml` effectue la configuration de base après le provisionnement (DNS, redémarrage, etc.). Les trois sont des playbooks d'entrée de niveau supérieur plutôt que des inclusions de tâches. Ils lisent le jeton API à partir de la variable d'environnement `VULTR_API_TOKEN`, et le playbook de provisionnement fait référence à un `cloud-config.yaml` du même répertoire, donc exécutez-les depuis `util_playbooks/vm/vultr/`.

```
export VULTR_API_TOKEN=<token>
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vm/vultr/vmProvisionVultr.yml
```

Vérification de la Santé

Fichier : `util_playbooks/health_check.yml`

Génère un rapport de santé HTML complet couvrant tous les services OmniCore et OmniCall déployés.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/health_check.yml
```

Sortie : `/tmp/health_check_YYYY-MM-DD HH:MM:SS.html`

Informations Collectées

Composant	Données Collectées
Tous les services	État du service, version, temps de disponibilité
OmniHSS	État de la base de données, connexions de pairs Diameter
OmniDRA	Connexions de pairs Diameter et état
OmniTAS	Appels actifs, sessions, enregistrements, utilisation du CPU
OCS	État de la réplication KeyDB

Restauration HSS

Fichier : `util_playbooks/restore_hss.yml`

Restaure OmniHSS à partir de fichiers de sauvegarde. Prend en charge la restauration uniquement de la base de données, uniquement de la configuration, ou des deux.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/restore_hss.yml
```

Formats de Fichiers de Sauvegarde

Type	Modèle de Nom de Fichier	Contenu
Base de données	`hss_dump_<hostname>_<timestamp>.sql`	Dump PostgreSQL (pg_dump) de la base de données OmniHSS (par défaut `omnihss`)
Configuration	`hss_<hostname>_<timestamp>.tar.gz`	Archive du répertoire `/etc/omnihss`

L'état actuel de HSS (base de données et/ou configuration) est sauvegardé avant que la restauration ne soit exécutée.

Prompts

Invite	Requis	Description
Chemin du dump SQL HSS	Non	Chemin complet vers le fichier de dump SQL HSS (laisser vide pour ignorer la restauration de la base de données)
Chemin du tar.gz de configuration HSS	Non	Chemin complet vers le fichier tar.gz de configuration HSS (laisser vide pour ignorer la restauration de la configuration)
Confirmation	Oui	Tapez `yes` pour confirmer l'opération de restauration sur l'hôte(s) listé(s)

Semence de Réplication HSS

Fichier : `util_playbooks/seed_hss_replication.yml`

Effectue le chargement initial des données une seule fois pour un nœud OmniHSS rejoignant un maillage de réplication actif-actif déjà peuplé. Un nouveau nœud démarre vide (les abonnements sont créés avec `copy_data = false`) ; ce playbook recrée l'abonnement à un seul pair source choisi avec `copy_data = true`, de sorte que PostgreSQL copie les données existantes et continue ensuite le streaming sans interruption.

Il s'exécute dans l'un des deux modes :

Interactif (par défaut) : omettre les variables `seed_source_*`. Le playbook détecte quels cibles limitées sont vides. Il sonde ensuite les pairs valides pour ceux qui détiennent des données, les liste avec les comptes d'abonnés, et vous invite à choisir une source. Il ne sème que les cibles vides.

```
ansible-playbook -i hosts/customer/host_files/production.yml \
  util_playbooks/seed_hss_replication.yml \
  --limit site-hss01,site-hss02
```

```
Pairs avec des données disponibles pour semer à partir de :
  1) dev-hss01 (10.4.10.142) - 838 abonnés
Entrez le NUMÉRO de la source à partir de laquelle semer
```

Explicite : passez la source directement pour ignorer le menu. Cela **force** un re-semencement même si la cible détient déjà des données (utilisez pour la récupération).

```
ansible-playbook -i hosts/customer/host_files/production.yml \
  util_playbooks/seed_hss_replication.yml \
  --limit site-hss01 \
  -e "seed_source_name=dev-hss01 seed_source_host=10.4.10.142"
```

Paramètres

Paramètre	Requis	Par défaut	Description
`seed_source_name`	Non	(invité)	Nom d'hôte d'inventaire du pair source peuplé Utilisé pour nommer l'abonnement (`sub_from_<name>`) et résoudre sa publication (`<name>_pub`). Omettre pour le mode interactif.
`seed_source_host`	Non	(invité)	Adresse IP du pair source, accessible sur son port PostgreSQL. Omettre pour le mode interactif.
`seed_source_port`	Non	port `omnihss.database_port` de la cible (5432)	Port PostgreSQL sur le pair source.

> Les deux `seed_source_name` et `seed_source_host` doivent être fournis ensemble pour utiliser le mode explicite ; ne pas en fournir déclenche le mode interactif.

Comportement

1. Compte les abonnés sur chaque cible limitée pour déterminer lesquelles sont vides.

2. **Mode interactif :** sonde les pairs valides (groupe `hss` local + `connected_omnihss`) pour ceux détenant des données, les liste, et invite à faire un choix. **Mode explicite :** demande un `yes` pour confirmer.
3. S'assure que l'IP source est autorisée dans le `pg_hba.conf` de la cible.
4. Supprime et recrée l'abonnement à la source `WITH (copy_data = false)` (nom de slot par lien de `<target>_from_<source>`). Il envoie **toutes** les tables mais ne réalise pas de synchronisation de tables en masse.
5. Remplit les lignes de **table maître** existantes à partir de la source avec un chargement sûr en cas de conflit (`pg_dump --data-only --inserts --on-conflict-do-nothing --disable-triggers`). `subscriber_state` **est** inclus : chaque abonné référence un par FK, donc le sauter orphelin chaque abonné rempli. Les tables de session transitoires (`pdn_session`, `lte_call`) ne sont **pas** remplies.
6. Rapporte le nombre d'abonnés sur le nœud semé.

> **Pourquoi les tables de session sont ignorées :** ce sont des tables chaudes ; un `COPY` en masse d'elles sur un système en direct entre en collision (clé dupliquée) avec des écritures concurrentes et provoque des boucles de plantage pour le travail de synchronisation de tables. Elles se remplissent à partir de la réplication en streaming et du trafic en direct à la place. `subscriber_state` est différent : les lignes ne sont insérées qu'au moment du provisionnement (1:1 avec l'abonné) et ne sont mises à jour qu'ensuite, donc le remplissage en cas de conflit `--on-conflict-do-nothing` est sûr. Sans cela, les abonnés de la cible pointent vers des lignes d'état qui n'arrivent jamais, puisque la réplication logique abandonne silencieusement les mises à jour vers des lignes manquantes.

> **Important :** Toujours `--limit` aux nœuds cibles. En mode interactif, les cibles non vides sont automatiquement ignorées. Voir Réplication & Opérations de Nœud OmniHSS pour l'ensemble du flux de travail.

Audit de Dérive HSS

Fichiers : `util_playbooks/audit_hss_drift.yml` (autonome) et `roles/omnihss/tasks/audit_drift.yml` (l'implémentation partagée).

Rapporte la **dérive de données** à travers le maillage actif-actif OmniHSS. Les abonnements de réplication logique sont créés avec `copy_data = false`, donc les données qui ont divergé avant le lien de maillage (différences au moment de la semence) ou via une écriture directe/non répliquée ne sont jamais réconciliées et jamais alarmées. La réplication montre 0 retard pendant que les données diffèrent, et Prometheus par site ne peut pas le voir car le maillage s'étend sur deux sites.

L'audit se connecte à chaque membre du maillage (groupe `hss` local + pairs `connected_omnihss`) et, pour chaque table de référence/configuration, compare un nombre exact de lignes **et** un md5 indépendant de l'ordre du contenu complet. Il compare également `schema_migrations` (nombre + version max) pour détecter la dérive de schéma. Les tables d'état chaudes (`subscriber_state`, `pdn_session`, `lte_call`) sont exclues : elles diffèrent légitimement sous trafic en direct. Un contrôle de **santé** structurel est inclus en plus de la comparaison inter-nœuds : `__orphaned_subscribers__`, le compte par nœud d'abonnés dont la ligne `subscriber_state` est manquante. Un compte non nul sur **n'importe quel** nœud échoue l'audit même lorsque chaque nœud est d'accord. Une semence qui copie `subscriber` sans `subscriber_state` orphelin chaque nœud de manière identique, ce que la comparaison inter-nœuds pure ne peut jamais voir. La comparaison est centralisée sur un exécuteur qui atteint chaque Postgres de pair, donc une seule exécution couvre l'ensemble du maillage inter-site depuis n'importe quel inventaire, et elle est sécurisée par `--limit`.

S'exécute automatiquement après chaque déploiement HSS. À la fin du rôle `omnihss` (lorsque la réplication est activée), le même audit s'exécute comme un contrôle de cohérence post-déploiement, et par défaut **ÉCHOUE le play** si une dérive est trouvée, donc un maillage cassé est impossible à manquer lors d'un déploiement. Il est non interactif (pas de pause) : l'audit doit être sûr sous la stratégie `free` du déploiement agrégé `services/all.yml`, où une pause provoquerait une erreur. Dans ce déploiement agrégé, l'audit dans le rôle est supprimé et l'audit s'exécute plutôt une fois dans un play **linéaire** dédié à la fin de `services/deploy_nfs.yml`. Le comportement est contrôlé par deux variables de rôle :

Variable	Par défaut	Effet
`omnihss_audit_drift`	`true`	exécuter l'audit à la fin du rôle
`omnihss_audit_fail_on_drift`	`true`	échouer le play sur dérive ; définir `false` pour rapport uniquement (non interactif cron/CI)

L'audit est **en lecture seule** de toute façon. Le playbook autonome par défaut rapporte puis **échoue** sur dérive (afin qu'il puisse contrôler cron/CI) :

```
# à la demande
ansible-playbook -i hosts/<inv>/host_files/<site>.yml
util_playbooks/audit_hss_drift.yml
# rapport uniquement, toujours passer
ansible-playbook -i ... util_playbooks/audit_hss_drift.yml -e
audit_fail_on_drift=false
# désactiver l'audit automatique pour un seul déploiement
ansible-playbook -i ... services/omnihss.yml -e
omnihss_audit_drift=false
```

Réconciliation de Dérive HSS

Fichier : `util_playbooks/reconcile_hss_drift.yml`

Ramène les tables de référence divergentes en synchronisation en faisant en sorte que chaque nœud détienne l'**union** des lignes : pour chaque table dérivée, il remplit les lignes qu'un nœud manque à partir de ses pairs, en utilisant le même primitive sûr en cas de conflit que le playbook de semence (`pg_dump --data-only --inserts --on-conflict-do-nothing --disable-triggers`).

- **Non destructif** : les lignes ne sont jamais que des INSERTs (ignorant les conflits). Rien n'est supprimé ou mis à jour. Cela résout "un nœud manque des lignes" ; cela ne résout **pas** "même clé primaire, valeurs différentes" ou

ne supprime pas les lignes qui ne devraient pas exister. Cela nécessite une décision humaine.

- **Interactif par défaut** : il scanne, imprime le plan, et **demande** (`yes` pour continuer) avant d'écrire ; toute autre chose annule sans changements. Utilisez `-e dry_run=true` pour regarder uniquement, ou `-e assume_yes=true` pour des exécutions non interactives.
- Les écritures s'exécutent localement sur chaque cible en tant que superutilisateur postgres (pour `--disable-triggers`) ; les pairs ne sont jamais que lus, en tant qu'utilisateur de l'application.
- **Union symétrique, pas autoritaire en prod** : l'exécuter contre un site tire les lignes de *ce* site vers les autres. Cela ne traite **pas** la prod comme le maître, donc l'exécuter contre l'inventaire Lab propagerait les lignes Lab vers la prod. Si la prod doit être la source de vérité, dirigez-le depuis l'inventaire prod (ou demandez le mode autoritaire).
- **Inter-site :** un nœud ne peut être qu'un *cible* lorsqu'il est dans le play, donc exécutez ceci contre **chaque** inventaire de site pour couvrir chaque nœud. La rescane à la fin rapporte tout ce qui nécessite encore une exécution.

```
# interactif (par défaut) : demande avant d'écrire
ansible-playbook -i hosts/<inv>/host_files/<site>.yml
util_playbooks/reconcile_hss_drift.yml
# regarder uniquement, jamais écrire
ansible-playbook -i ... util_playbooks/reconcile_hss_drift.yml -e
dry_run=true
# application non interactive (CI)
ansible-playbook -i ... util_playbooks/reconcile_hss_drift.yml -e
assume_yes=true
```

> Exécutez `audit_hss_drift.yml` d'abord pour voir ce qui a dérivé, et à nouveau après la réconciliation (contre les deux inventaires) pour confirmer la convergence. Voir Réplication & Opérations de Nœud OmniHSS.

Restauration OCS

Fichier : `util_playbooks/restore_ocs.yml`

Restaure OCS (CGrateS) à partir de fichiers de sauvegarde. Gère la réplication KeyDB multi-maître en restaurant sur un nœud et en permettant à la réplication de se synchroniser avec les autres.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/restore_ocs.yml
```

Processus de Restauration

1. Arrête CGrateS et KeyDB sur tous les nœuds OCS
2. Efface les fichiers AOF pour éviter les conflits avec les données restaurées
3. Restaure KeyDB RDB sur le premier nœud, démarre KeyDB, laisse la réplication se synchroniser
4. Restaure MySQL StoreDB (optionnel)
5. Restaure la configuration `/etc/cgrates` (optionnel)
6. Démarre CGrateS sur tous les nœuds

Formats de Fichiers de Sauvegarde

Type	Modèle de Nom de Fichier	Contenu
Données KeyDB	`keydb_dump_<hostname>_<timestamp>.rdb`	Instantané RDB de KeyDB
MySQL StoreDB	`cgrates_dump_<hostname>_<timestamp>.sql`	Dump MySQL de la base de données `cgrates`
Configuration	`cgrates_<hostname>_<timestamp>.tar.gz`	Archive du répertoire `/etc/cgrates`

Prompts

Invite	Requis	Description
Chemin du dump RDB KeyDB	Oui	Chemin complet vers le fichier de sauvegarde RDB KeyDB
Chemin du dump SQL MySQL	Non	Chemin complet vers le dump SQL de CGrateS (sauter pour préserver le StoreDB actuel)
Chemin du tar.gz de configuration	Non	Chemin complet vers la sauvegarde de configuration (sauter pour préserver la configuration actuelle)

Générateur de Plan IP

Fichier : `util_playbooks/ip_plan_generator.yml`

Génère de la documentation réseau à partir de l'inventaire, y compris :

- Attributions d'IP des hôtes (NICs primaires et secondaires)
- Vue d'ensemble des segments réseau
- Diagrammes de connectivité des interfaces (Diameter, GTP, PFCP, SIP, SS7)

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/ip_plan_generator.yml
```

Fichiers de Sortie

Fichier	Format	Description
`/tmp/ip_plan_<customer>_<site>.md`	Markdown	Documentation textuelle
`/tmp/ip_plan_<customer>_<site>.html`	HTML	Diagramme interactif avec des couches filtrables

Audit des Ports

Fichier : `util_playbooks/get_ports.yml`

Audite tous les ports à l'écoute à travers le déploiement et génère de la documentation.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/get_ports.yml
```

Fichiers de Sortie

Fichier	Description
`/tmp/all_ports.csv`	CSV avec nom d'hôte, IP, protocole, port, service
`./open_ports.rst`	tableau reStructuredText pour la documentation Sphinx

Données Collectées

Champ	Description
Nom d'hôte	Nom d'hôte d'inventaire
IP	Adresse IP `ansible_host` de l'hôte
Version IP	IPv4 ou IPv6
Transport	TCP ou UDP
Port	Numéro de port à l'écoute
Service	Nom du processus

Récupération de Capture Locale

Fichier : `util_playbooks/getLocalCapture.yml`

Récupère les deux fichiers de capture de paquets les plus récents du répertoire `/etc/localcapture` de chaque hôte.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/getLocalCapture.yml
```

Sortie : `./localCapturePcaps/<hostname>/*.pcap`

Gestion des Utilisateurs

Fichier : `util_playbooks/delete_local_user.yml`

Supprime un compte utilisateur local de tous les hôtes de l'inventaire.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/delete_local_user.yml
```

Invite : Entrez le nom d'utilisateur à supprimer lorsqu'il est demandé.

Effacer la Base de Données Mnesia

Fichier : `util_playbooks/clear_mnesia.yml`

Arrête le service, supprime son répertoire de base de données Mnesia, et le redémarre. S'exécute contre le groupe `ss7` (OmniSS7) et le groupe `omnimessage` (OmniMessage).

> **Avertissement :** Ceci est destructif. La base de données Mnesia est définitivement supprimée et reconstruite vide au redémarrage.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/clear_mnesia.yml

# Limiter à un seul service ou hôte
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/clear_mnesia.yml --limit ss7
```

Supprimer la Licence

Fichier : `util_playbooks/delete_license.yml`

Supprime `/etc/license_server/license.json` sur le groupe `license_server` et redémarre le serveur de licence.

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/delete_license.yml
```

Exécution des Playbooks Utilitaires

Syntaxe de Base

```
ansible-playbook -i <inventory_file> util_playbooks/<playbook>.yml
```

Options Courantes

Option	Description
`-i <inventory>`	Spécifier le fichier d'inventaire
`--limit <hosts>`	Limiter à des hôtes ou groupes spécifiques
`-v` / `-vv` / `-vvv`	Augmenter la verbosité
`--check`	Exécution à sec (montrer ce qui changerait)
`--diff`	Montrer les différences de fichiers

Exemples

```
# Exécuter la vérification de santé en production
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/health_check.yml

# Restaurer HSS sur un hôte spécifique
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/restore_hss.yml --limit hss01

# Générer un plan IP avec sortie verbeuse
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/ip_plan_generator.yml -v
```