5G□心（5GC）

5G□立（SA）核心由`services/5gc.yml`剧本部署。一个共享角色`5gc_nf`安装每个网络功能（NF）。该角色针对每个NF进行了参数化。本文解释了该角色、它构建的NF以及每个NF读取的配置。

概述

OmniCore使用一个角色（**`5gc_nf`**）来处理所有十一种**5GC NF**，而不是每个NF一个角色。剧本每个NF调用一次该角色，并传递一个`nf_package`变量，选择Debian包、systemd服务和Jinja2配置模板。这使得安装、配置版本控制、许可和重启逻辑在每个NF中保持一致。

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

网络功能

每个NF都有自己的库存组、包、systemd服务和配置模板（`roles/5gc_nf/templates/<nf_package>.j2`）。

NF	`nf_package`	□□□	□□□□	□□
□□□□□□□	`omninrf`	`nrf`	`omninrf.j2`	□□□□ / NF□□□
□□□□□□□	`omniamf`	`amf`	`omniamf.j2`	N1/N2□□□□□□□□□□
□□□□□	`omniausf`	`ausf`	`omniausf.j2`	5G-AKA / EAP-AKA'□□
□□□□□□	`omnibsf`	`bsf`	`omnibsf.j2`	PDU□□□PCF□□
□□□□	`omnichf`	`chf`	`omnichf.j2`	□□□□□/□□□□
□□□□□□	`omnissf`	`nssf`	`omnissf.j2`	NSSAI / □□□□
□□□□□□	`omnipcf`	`pcf`	`omnipcf.j2`	□□□QoS□□
□□□□□□	`omniscp`	`scp`	`omniscp.j2`	SBI□□□□ / □□□□
□□□□□□	`omnismf`	`smf`	`omnismf.j2`	PDU□□□UPF□N4□□□
□□□□□□	`omniudm`	`udm`	`omniudm.j2`	□□□□□SUCI□□□
□□□□□□	`omniudr`	`udr`	`omniudr.j2`	UDM/PCF□□□□□□□

`5gc_nf`□□□□□□□

□□□□NF□□□□□□□□□□□□□□

1. □□**`group_vars_omnicore`**□□□□`<inventory>/group_vars/`□□□□□□□□□□□□□□□□□□□□
2. □□□□□□□□□□□□□□□`license_server_api_urls`□□□`license_server`□□□□□□`https://<host>:8443/api`□□
3. □□□□□□□□`lsb-release`□`curl`□`gpg`□`unzip`□`libglu1-mesa`□`libsctp1`□
4. □**Omnitouch APT**□□□□□**NF**□□□□□□□□□□□□□□□`nf_version`□□□□□□□□□□□□

5. □□□□□□□□□□□□□□□□□`templates/<nf_package>.j2`□□□`/etc/<nf_package>/runtime-<deb_version>.exs`□
6. □□□□`runtime.exs` → □□□□□□□□□□□□□□□□□□□□□□□`/opt/<nf_package>/releases/*`□□□□
7. □□**NF**□□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□Debian□□□□□□□`runtime-<deb_version>.exs`□□`runtime.exs`□□□□□□□□□□□□□□□□□□□

相关文档

- 服务剧本 - 所有服务剧本和部署层次结构
- 电路交换核心 - 传统CS域（BSC、MSC、SS7）
- 组变量配置 - 每个NF配置模板和HNET密钥
- 部署架构 - 各个域如何结合在一起
- **OmniCore**配置： https://docs.omnitouch.com.au/docs/repos/OmniCore - NF配置文件中的内容

OmniCore IP □□□□

□□

□□□□□□□ OmniCore □□□□□□ IP □□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□

网络段细分

1. 数据包核心网络 (第一个 /24)

目的：用户平面和核心控制平面元素

组件：

- OmniMME (移动管理实体)
- OmniSGW (服务网关)
- OmniPGW-C (PDN 网关控制平面)
- OmniUPF/PGW-U (用户平面功能 / PDN 网关用户平面)

示例：`10.179.1.0/24`

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

---

2. 信令网络 (第二个 /24)

目的：Diameter 信令、策略、计费和管理功能

组件：

- OmniHSS (家庭用户服务器)
- OmniCharge OCS (在线计费系统)
- OmniHSS PCRF (策略和计费规则功能)
- OmniDRA DRA (Diameter 路由代理)
- DNS 服务器（内部/基础设施 DNS - 与 IMS 公共段中的面向客户的 DNS 不同；两者都是故意的）
- TAP3/CDR 服务器
- 监控/OAM

- SIP 捕获
- 许可证服务器
- RAN 监控
- Omnitouch 警告链接 CBC (小区广播中心) - 如果已部署

示例: `10.179.2.0/24`

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

---

3. IMS 内部网络 (第三个 /24)

目的: IMS 核心信令和服务 (内部 SIP 信令)

组件:

- OmniCSCF S-CSCF (服务呼叫会话控制功能)
- OmniCSCF I-CSCF (查询呼叫会话控制功能)
- OmniTAS (电话应用服务器 / 应用服务器)
- OmniMessage (短信控制器, SMPP, IMS)

示例: `10.179.3.0/24`

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

---

4. IMS 公共网络（第四个 /24）

目的：面向用户的 IMS 服务、面向客户的 DNS 和 SS7/遗留互连

组件：

- OmniCSCF P-CSCF（代理呼叫会话控制功能）
- XCAP 服务器
- 可视语音邮件服务器
- 客户 DNS
- OmniSS7 STP (SS7 信令传输点)
- OmniSS7 HLR（家庭位置寄存器）- 用于 2G/3G
- OmniSS7 IP-SM-GW (MAP SMSc)
- OmniSS7 CAMEL 网关
- MSC（移动交换中心）

> 注意： SS7/HLR 和相关的遗留互连功能通常位于公共段，因为它们通过 SIGTRAN/M3UA 与外部 2G/3G 网络对等，而不是通过内部 IMS SIP。

示例： `10.179.4.0/24`

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

实施方法

OmniCore 支持两种主要方法来实现此网络分段：

方法 1：物理/虚拟网络接口（推荐用于生产）

为每个网络段使用单独的 NIC 或虚拟桥。这提供了最强的隔离，是生产部署的推荐方法。约定是根据其段命名每个桥（`v400-omni-packet-core`、`v401-omni-signalling`、`v402-omni-ims-internal`、`v403-omni-ims-public`），尽管在其他地方处理分段的单个共享桥（例如 `vmbr0`）也是有效的。

示例：

```yaml
# 数据包核心 - 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"

# 信令 - 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 内部 - 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 公共 - 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"
```

方法 2：基于 VLAN 的分段

使用单个物理接口并进行 VLAN 标记以分隔网络。这适用于较小的部署或当物理 NIC 有限时。

示例：

```yaml
# 所有组件共享一个桥 (ovsbr1)；每个段使用不同的 VLAN 标签。
# applicationserver (TAS) 是 IMS 内部组件 -> 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 和 dns 是信令组件 -> 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"
```

网络配置：

- 在物理交换机上配置 VLAN
- 在虚拟机监控程序级别适当地标记流量
- 在网关/防火墙处在 VLAN 之间路由

示例 **VLAN** 映射：

```
VLAN 400: 10.x.1.0/24 (□据包核心)
VLAN 401: 10.x.2.0/24 (信令)
VLAN 402: 10.x.3.0/24 (IMS 内部)
VLAN 403: 10.x.4.0/24 (IMS 公共)
```

使用公共 IP 地址

概述

许多 OmniCore 部署要求某些组件具有公共 IP 地址以实现外部连接，例如：

- **DRA** - 与外部运营商进行漫游直径信令
- 漫游 **SGW/PGW** - 来自漫游合作伙伴的 GTP 流量
- **ePDG** - 用于 WiFi 通话（来自 UE 的 IPsec 隧道）
- **SMSC** 网关 - 用于与外部 SMS 聚合器的 SMPP 连接
- **P-CSCF**（在某些部署中） - 用于直接 UE SIP 注册

如何分配公共 IP

公共 IP 的处理方式与内部 IP 在您的主机清单文件中完全相同。只需在 `ansible_host` 字段中指定公共 IP 地址，并提供适当的网关和子网掩码。

示例：具有公共 **IP** 的漫游 **SGW/PGW**

```yaml
sgw:
  hosts:
    # 内部 SGW 在私有网络上
    customer-site-sgw01:
      ansible_host: 10.4.1.25
      gateway: 10.4.1.1
      host_vm_network: "v400-omni-packet-core"

    # 具有公共 IP 的漫游 SGW
    customer-site-roaming-sgw01:
      ansible_host: 203.0.113.10
      gateway: 203.0.113.9
      netmask: 255.255.255.248       # /29 子网
      host_vm_network: "498-public-servers"
      in_pool: False
      cdrs_enabled: True

smf:  # PGWs
  hosts:
    # 具有公共 IP 的漫游 PGW
    customer-site-roaming-pgw01:
      ansible_host: 203.0.113.20
      gateway: 203.0.113.17
      netmask: 255.255.255.240       # /28 子网
      host_vm_network: "497-public-services-LTE"
      in_pool: False
      ip_pools:
        - '100.64.24.0/22'
```

示例：具有公共 **IP** 的 **DRA**

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

示例：具有公共 **IP** 的 **ePDG**

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

混合内部和公共 IP

在同一组件组中混合使用内部和公共 IP 是很常见的。例如：

- 用于本地站点的内部 SGW 使用 GTP
- 专门用于来自外部运营商的漫游流量的公共 SGW
- 同一个 PGW-C 可以管理内部和外部 SGW

OmniCore 的架构可以无缝处理这一点 - 只需为每个主机配置其适当的 IP 地址即可。

Omnitouch 中 Ansible 部署简介

概述

Omnitouch 网络服务使用 Ansible 作为其基础设施自动化平台，以一致、可重复和自动化的方式部署完整的蜂窝网络解决方案（4G/5G）。本文档提供了我们如何利用 Ansible 来协调复杂的电信部署的概述。

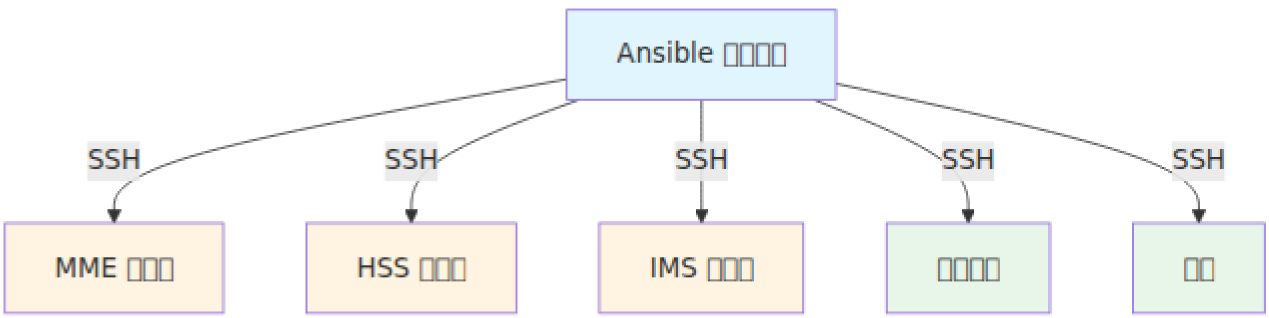
什么是 Ansible?

Ansible 是一个开源自动化工具，允许您:

- 配置系统
- 部署软件
- 协调复杂的工作流程
- 将基础设施作为代码进行管理

Ansible 使用声明性方法 - 您描述系统的 期望状态，Ansible 确保它们达到该状态。

Omnitouch 如何使用 Ansible



关键概念

1. 清单（主机文件）

定义 要管理的 系统。每个客户部署都有一个描述以下内容的主机文件：

- 网络中的所有虚拟机
- 它们的 IP 地址
- 网络配置
- 服务特定参数

主机文件是您将用于定义网络的内容。

参见：主机文件配置

2. 角色

定义 如何 配置每个组件。角色是可重用的单元，包含：

- 任务（要执行的步骤）
- 模板（配置文件模板）
- 处理程序（由更改触发的操作）
- 变量（默认配置值）

OmniCore 组件的示例角色：`omnihss`、`omnisgwc`、`omnipgwc`、`omnidra` 等

ONS 团队定义这些角色。您可以编辑它们。然而，通常有更简洁的方法可以从主机文件中进行任何必要的调整。

3. 剧本

协调 何时 和 在哪里 应用角色：

```yaml
- name: 设置 Omnicore MME
  hosts: mme
  roles:
    - omnimme
```

我们基本上将这些用作角色的分组。

在实践中，每个服务剧本应用一个单□的角色，共享设置单□拉取。像 `services/epc.yml` 这样的服务包使用 `import_playbook` 首先运行 `common.yml`，然后是每个服务的剧本：

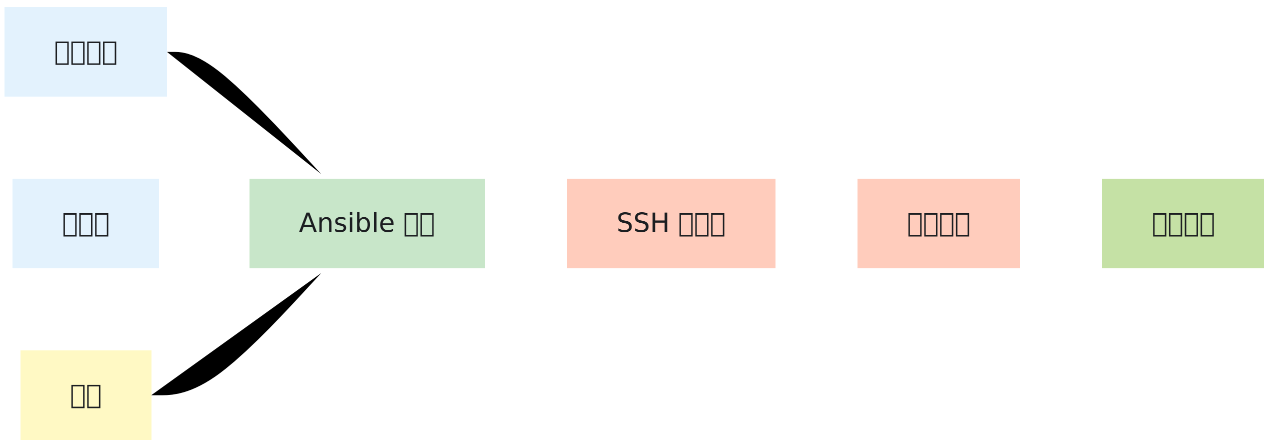
```yaml
- name: 安装公共组件
  import_playbook: common.yml

- name: 运行 MME 剧本
  import_playbook: omnimme.yml
```

4. 组变量

提供客户特定配置，覆盖角色默认值。这是客户自定义而不修改基础角色的地方。

参见：组变量和配置

部署架构



部署过程

1. 定义基础设施

创建一个描述您的网络拓扑的主机文件：

规划说明：在定义基础设施之前，请查看 IP 规划标准，以获取有关网络分段、IP 地址分配和子网组织的指导。

Proxmox 用户：如果在 Proxmox 上部署，请参见 Proxmox VM/LXC 部署 以获取自动化 VM/容器配置的信息。

参见：主机文件配置 和 配置参考

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

2. 自定义配置

在 `group_vars` 中设置客户特定变量：

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

#待办事项 - 在此处添加链接以获取完整列表的配置参考

3. 运行剧本

部署网络：

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

4. 自动化部署

Ansible 将：

- 创建/配置虚拟机（如果使用 Proxmox/VMware 集成）
- 配置网络
- 从 APT 缓存安装软件包
- 部署应用程序代码
- 使用客户设置配置服务
- 启动服务
- 验证部署

我们部署的关键组件

OmniCore（4G/5G 分组核心平台）

- **OmniHSS** - 家庭用户服务器
- **OmniSGW** - 服务网关（控制平面）
- **OmniPGW** - 分组网关（控制平面）
- **OmniUPF** - 用户面功能
- **OmniDRA** - Diameter 路由代理
- **OmniTWAG** - 受信 WLAN 访问网关

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

OmniCall（语音和消息平台）

- **OmniCall CSCF** - 呼叫会话控制功能（P-CSCF、I-CSCF、S-CSCF）
- **OmniTAS** - IMS 应用服务器（VoLTE/VoNR 服务）
- **OmniMessage** - 短信中心（SMS-C）
- **OmniMessage SMPP** - SMPP 协议支持
- **OmniSS7** - SS7 信令组件（STP、HLR、CAMEL）
- **VisualVoicemail** - 语音邮件功能

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

OmniCharge/OmniCRM

- **CRM** 平台 - 客户关系管理、自助注册、计费

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

支持服务

- **DNS** - 网络 DNS 解析
- 许可证服务器 - 许可证管理
- 监控 - Prometheus、Grafana

参见：部署架构概述

软件包管理

我们使用混合软件包分发模型：

预编译的 APT 软件包

所有 Omnitouch 软件以 Debian 软件包（`.deb` 文件）分发：

- 从我们的 CI/CD 管道中构建
- 版本化和测试
- 托管在软件包仓库中

APT 缓存系统

客户可以选择：

1. 本地 **APT** 缓存 - 现场所需软件包的镜像，用于离线部署
2. 公共仓库 - 直接访问 Omnitouch 托管的软件包仓库

还有一个统一的 `omnicore` `.deb`（从 `packaging/` 构建），将剧本和工具捆绑在一起，以便自我配置部署。

参见：APT 缓存系统

许可证管理

所有 Omnitouch 软件组件都需要通过中央许可证服务器管理的有效许可证：

- 组件在启动时检查许可证有效性
- 根据许可证启用/禁用功能
- 许可证服务器可以是本地或云托管的

参见：许可证服务器

□□□□□□□

□□□□

□□□ Ansible □□��□□□□

- □□□□□
- □□□□
- □□□□
- □□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□ Git □□□□

- □□□□□□
- □□□□□
- □□□□□□□

□□□□□□□

□□□□□□ `group_vars` □□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

入门

先决条件

在运行 Ansible 剧本之前，您需要安装所需的依赖项。该项目使用 uv 来管理 Python 环境，因此 需要 **uv**。

1. 安装 **uv**

如果尚未安装 uv，请使用适合您的控制节点的方法进行安装。请参见 uv 安装文档。例如：

```
pipx install uv                                      # 通过 pipx
curl -LsSf https://astral.sh/uv/install.sh | sh      # 独立安装程序
```

2. 安装依赖项

从仓库根目录运行：

```
uv sync
```

这将创建一个隔离的 `.venv/` 并安装 Ansible 以及 Omnitouch 部署自动化所需的所有 Python 包（在 `uv.lock` 中固定）。

3. 激活环境

激活虚拟环境，并在所有 Ansible 命令中保持激活状态：

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

4. 安装 **Ansible** 集合

安装剧本所依赖的 Ansible 集合（来自 `requirements.yml`）：

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

5. 运行 **Ansible** 命令

□□□□□□□□□□□□□□□ Ansible □□□

```
ansible --version
```

□□□ □□□□□□□□ `deactivate` □□□□□□□

□□□□

1. □□ □□□□□□□ □□□□□□□□□□□□□
2. □□ □□□ □□□□□□□
3. □□ APT □□□□ □□□□□□□□□
4. □□ □□□□ □□□□□□□□□□□□□□□□
5. □□□

□□□

- IP □□□□ - □□□□□□□□□□ **IP** □□
- □□□□□□□ - □□□□□□□□□□□□□
- APT □□□□ - □□□□□□□□
- □□□□□□□ - □□□□□□□□
- □□□□□□□ - □□□□□□□
- □□□□□ - □□□□□□□□□□□□□□□□□□□□
- □□□□ - □□□□□□□□□□□□□□□□□□□□

APT 仓库与软件包分发

概述

Omnitouch APT 系统为所有部署提供软件包分发。提供两种类型的内容：

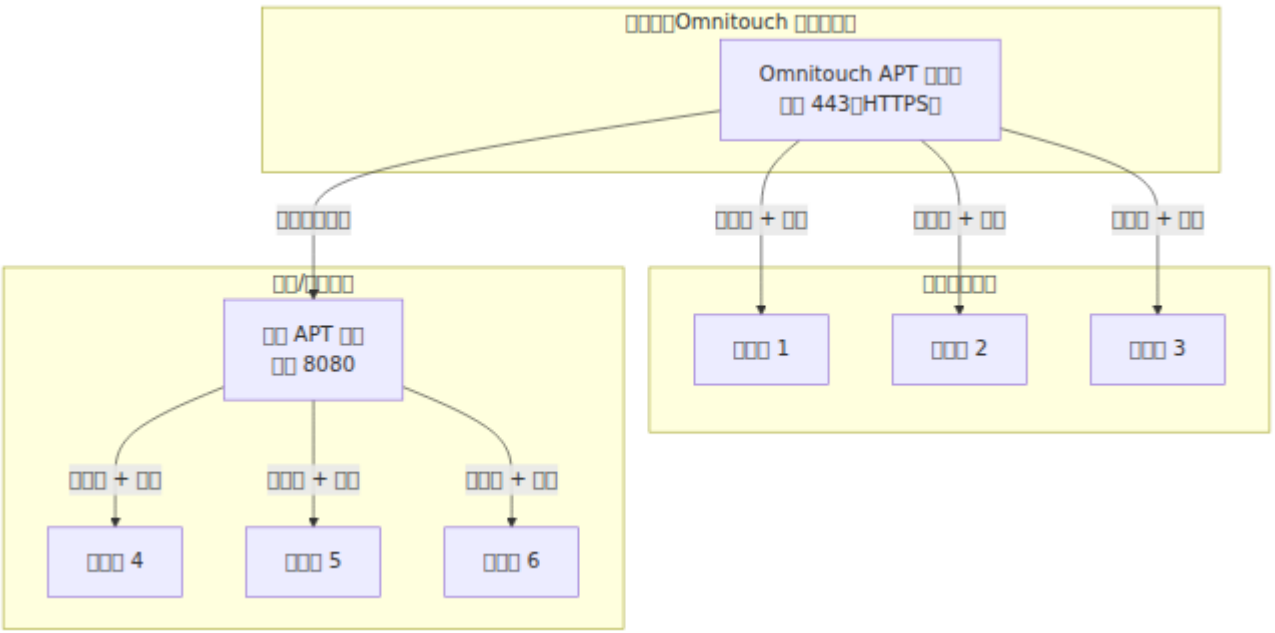
1. **APT** 软件包：通过 `apt install` 安装的 Debian 软件包
2. 二进制发布：直接下载的预构建二进制文件（Prometheus 导出器、代理等）

APT 服务器上的软件包页面还包括一个自我设置（**OmniCore**）选项卡，提供 `omnicore` 软件包（在 `/pool/main/o/omnicore/` 提供），其中包含完整的 Ansible 自动化工具包，供希望自行部署和管理网络的客户使用。

支持两种部署模型：

1. 直接访问：虚拟机直接从 Omnitouch APT 服务器拉取软件包（例如 `apt.<region>.omnitou.ch`）
2. 本地缓存镜像：本地服务器从 Omnitouch 同步并向虚拟机提供软件包（用于离线/隔离部署）

架构



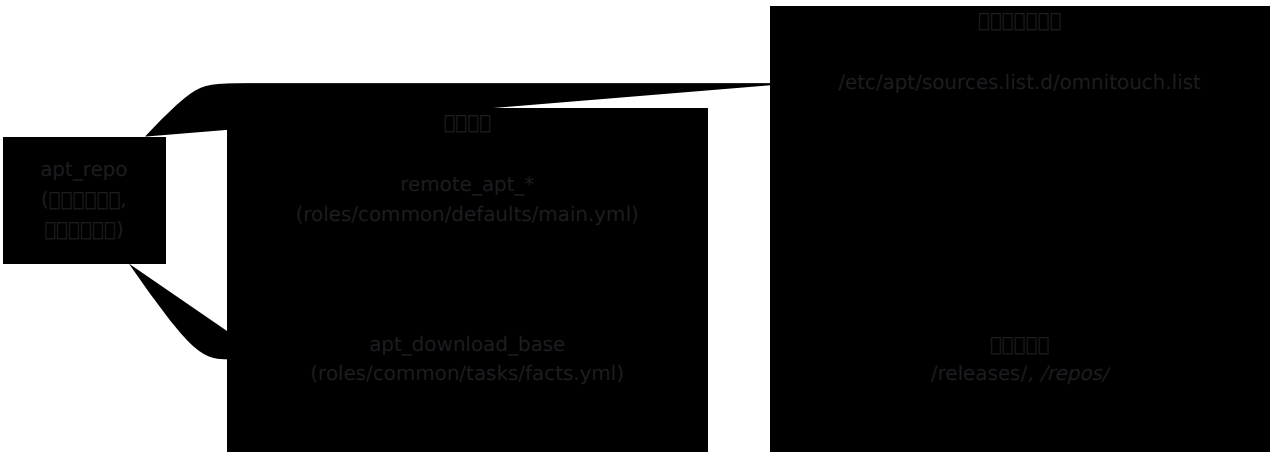
□□□□□

APT □□□□□□□□□□□□□□□□□

□□□□	□□	□□
Omnitouch □□	□□□□□□ `.deb` □□□□omnihss□ omnimme □□	`/dists/<distro>/`
OmniCore □□	`omnicore` □□□□□□□ Ansible □□□ □□□□□□□□□□□□□□□□□□	`/pool/main/o/omnicore/`
Ubuntu □□□	□□□□□□□□□□ Ubuntu □□□	`/<distro>/pool/main/`
GitHub □□	□□□□□□□□□□Prometheus□ Grafana□Homer □□	`/releases/<org>/<repo>/`
□□□□□□	Web □□□□□□□□□□CGrateS_UI□ speedtest□	`/repos/`
���□□□□	Galera□FRR□InfluxDB□KeyDB □	`/releases/<vendor>/`

□□□

`apt_repo` □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□



派生方式

变量	设置位置	角色
`apt_repo`	您的清单	您配置的唯一块。直接模板化到 APT 源行中。
`remote_apt_*`	`roles/common/defaults/main.yml`	从 `apt_repo` 派生。当 `apt_repo` 缺失时的最终回退：`apt.us-fl.omnitou.ch` / `443` / `https`。
`apt_download_base`	`roles/common/tasks/facts.yml`	每个角色用于二进制下载的单一计算基础 URL，因此 APT 源行和二进制下载始终在主机、方案、端口和凭据上保持一致。

因为两个派生值都是从同一个 `apt_repo` 块构建的，普通清单 只需要 **`apt_repo`**。您不需要自己设置 `remote_apt_*`。

> 例外：缓存镜像服务器。 `apt_cache_servers` 组中的主机是唯一一个 `remote_apt_*` 仍然显式设置的地方。它自己的 `apt_repo` 指向 *[illegible]*。然后 `remote_apt_*` 值描述它同步的 *[illegible]* Omnitouch 服务器（请参见 选项 2）。

方案和端口

方案自动跟随端口：端口 `443` → `https`，其他任何端口 → `http`。当省略 `apt_repo.apt_repo_port` 时：

- 直接访问 默认使用 `443`（**HTTPS**）
- 缓存模式 默认使用 `8080`（**HTTP**）

每种模式的使用场景

场景	**APT** 源行	二进制下载 （`apt_download_base`）
`use_apt_cache: true`（缓存）	`http://<cache>:8080/...` （无身份验证）	`http://<cache>:8080/...` （无身份验证）
`use_apt_cache: false`（直接）	`https://<user>:<pass>@<server>/...`	`https://<user>:<pass>@<server>:443/...`

选项 1：直接访问

对于具有互联网连接的部署，虚拟机直接从 Omnitouch APT 服务器拉取软件包。

网络要求

源 **IP** 白名单：您的公共 IP 地址必须在 Omnitouch APT 服务器上列入白名单。在设置过程中，向 Omnitouch 提供您的源子网。作为回报，您将收到：

- 用户名 和 密码 用于 HTTP 基本身份验证
- **APT** 服务器的 **FQDN**（例如 `apt.<region>.omnitou.ch`）

防火墙要求：必须允许对以下 Omnitouch IP 范围的出站访问：

网络	范围
IPv4	`144.79.167.0/24`
IPv4	`160.22.43.0/24`
IPv6	`2001:df3:dec0::/48`
ASN	`AS152894`

需要访问 **Omnitouch** 基础设施的服务:

服务	端口	协议	目的
APT 服务器	443	TCP	软件包下载（HTTPS）
APT 服务器	53	TCP/UDP	APT FQDN 的 DNS 解析
许可证服务器	123	UDP	用于许可证验证的 NTP 时间同步
许可证服务器	53	TCP/UDP	用于许可证验证的 DNS 解析

确保 HTTPS（TCP/443）、NTP（UDP/123）和 DNS（TCP+UDP/53）流量允许访问 Omnitouch IP 范围。

配置

```yaml
all:
  vars:
    use_apt_cache: false

    # 唯一真实来源。二进制下载配置（remote_apt_*）和
    # apt_download_base URL 是从此自动派生的 - 您不需要
    # 单独设置它们。
    apt_repo:
      apt_server: "apt.<region>.omnitou.ch"    # Omnitouch 提供的 FQDN
      apt_repo_username: "your-username"
      apt_repo_password: "your-password"
      # apt_repo_port: 443                     # 可选；默认为 443（HTTPS）
```

参数

APT 软件包源（`apt_repo`）

参数	类型	必需	默认	描述
`apt_repo.apt_server`	字符串	是	-	APT 服务器主机名或 IP 地址
`apt_repo.apt_repo_username`	字符串	是	-	HTTP 基本身份验证用户名
`apt_repo.apt_repo_password`	字符串	是	-	HTTP 基本身份验证密码
`apt_repo.apt_repo_port`	整数	否	`443`	服务器端口。方案跟随端口（`443` → HTTPS，其他则为 HTTP）。

二进制下载（`remote_apt_*`）

这些是从 `apt_repo` 自动派生的。在直接访问清单中不要设置它们。它们仅存在于缓存镜像服务器可以指向不同的上游（请参见选项 2）。作为参考，它们的默认值如下：

参数	从何派生	回退
`remote_apt_server`	`apt_repo.apt_server`	`apt.us-fl.omnitou.ch`
`remote_apt_port`	`apt_repo.apt_repo_port`	`443`
`remote_apt_protocol`	端口（`443` → `https`）	`https`
`remote_apt_user`	`apt_repo.apt_repo_username`	`""`
`remote_apt_password`	`apt_repo.apt_repo_password`	`""`

一般

参数	类型	必需	默认	描述
`use_apt_cache`	布尔	是	-	对于直接访问必须为 `false`

URL 模式（直接访问）

APT 软件包源（配置在 `/etc/apt/sources.list.d/omnitouch.list`）：

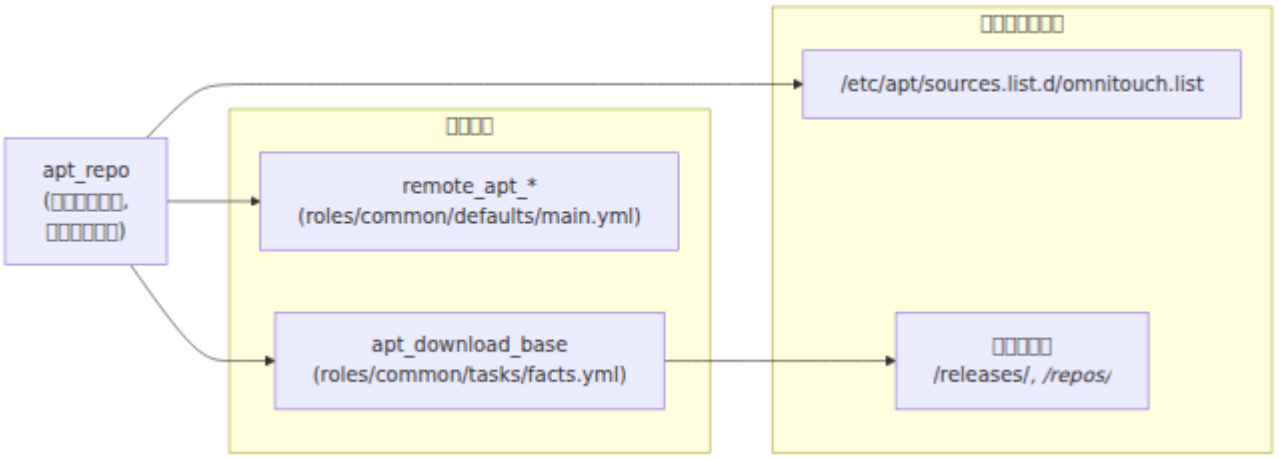
```
deb [trusted=yes] https://{username}:{password}@{apt_server}/
noble main
```

二进制下载（由 Ansible `get_url` 任务从 `apt_download_base` 构建）：

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

> 端口 `443` 在 URL 中被省略（它是 HTTPS 默认值）。非标准端口以 `:<port>` 附加，并将方案切换为 HTTP，除非它是 `443`。

工作原理



虚拟机通过 HTTP 基本身份验证对 APT 软件包和二进制下载进行身份验证。Ubuntu 系统软件包也从 Omnitouch 服务器提供（预缓存），因此虚拟机无需访问 Ubuntu 镜像。

软件包签名（**GPG**）

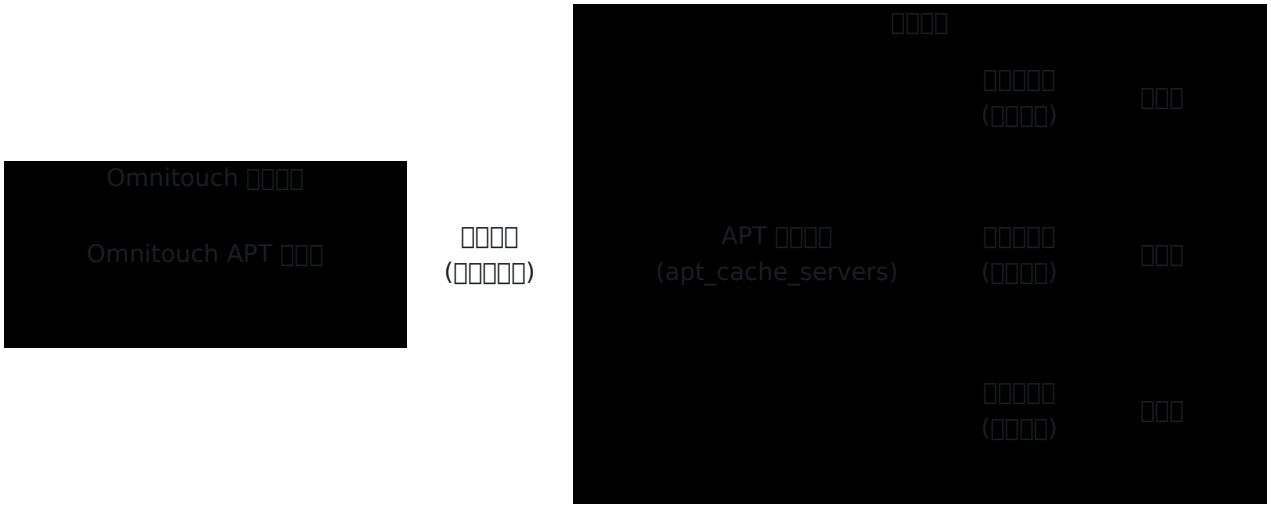
Omnitouch APT 服务器生成一个仓库签名密钥，并在 `/omnitouch.gpg` 提供装甲公钥。对于直接访问的主机，`common` 角色会自动获取该密钥并将其安装到 `/etc/apt/trusted.gpg.d/omnitouch.gpg` 中，以便 `apt-get update` 验证仓库元数据而不会出现 `NO_PUBKEY` 警告。这是自动发生的；无需手动密钥管理。

> 缓存镜像模式不使用 GPG 验证。从本地缓存拉取的主机使用 `[trusted=yes]` 代替（请参见选项 2）。

选项 **2**：本地缓存镜像

对于离线、隔离或带宽受限的部署，部署一个本地 APT 缓存，从 Omnitouch 同步所有内容。

架构



配置

在您的主机文件中定义缓存服务器。它是唯一一个显式设置 `remote_apt_*` 的地方。这些变量描述了缓存同步的 上游 Omnitouch 服务器:

```yaml
apt_cache_servers:
  hosts:
    example-apt-cache:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # 缓存服务器从上游 Omnitouch 仓库同步内容
    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  # 当 apt_cache_servers 组存在时自动设置
    # apt_repo.apt_server: 自动派生为 192.168.1.100 (第一个缓存服务器)
```

工作原理:

- 缓存服务器（`192.168.1.100`）：使用 `remote_apt_*` 凭据通过 HTTPS 从上游 Omnitouch 服务器同步内容。
- 所有其他主机：自动派生 `apt_repo.apt_server: "192.168.1.100"`，并在不需要凭据的情况下从缓存中拉取。

参数

APT 软件包源（`apt_repo`）

参数	类型	必需	默认	描述
`apt_repo.apt_server`	字符串	是	自动派生	本地缓存服务器 IP。如果未指定，则自动从第一个 `apt_cache_servers` 主机派生。
`apt_repo.apt_repo_username`	字符串	否	-	使用缓存时不需要（无需身份验证）
`apt_repo.apt_repo_password`	字符串	否	-	使用缓存时不需要（无需身份验证）
`apt_repo.apt_repo_port`	整数	否	`8080`	主机用于访问本地缓存服务器的端口。（替代已弃用的 `apt_cache_port`。）

缓存服务器同步（**`remote_apt_*`**）

这些变量配置缓存服务器如何从 Omnitouch 同步内容。这是唯一一个手动设置 `remote_apt_*` 的场景：

参数	类型	必需	默认	描述
`remote_apt_server`	字符串	是	-	上游 Omnitouch APT 服务器
`remote_apt_port`	整数	否	`443`	上游 Omnitouch APT 服务器端口
`remote_apt_protocol`	字符串	否	`https`	同步连接的协议
`remote_apt_user`	字符串	是	-	从 Omnitouch 同步的凭据
`remote_apt_password`	字符串	是	-	从 Omnitouch 同步的凭据

一般

参数	类型	必需	默认	描述
`use_apt_cache`	布尔	否	`true`	当 `apt_cache_servers` 组存在时自动设置为 `true`

> 已弃用：`apt_cache_port` 不再使用。主机连接的端口现在通过 `apt_repo.apt_repo_port` 设置（省略时默认为 `8080`）。

URL 模式（缓存模式）

APT 软件包源（配置在 `/etc/apt/sources.list.d/omnitouch.list`）：

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

二进制下载（由 `apt_download_base` 构建）：

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

缓存访问不需要凭据。它使用 `[trusted=yes]` APT 配置。

部署缓存

1. 准备缓存服务器（具有 50+ GB 磁盘的虚拟机或 LXC 容器）
2. 运行缓存设置剧本：

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

3. 通过访问 **`http://192.168.1.100:8080/`** 验证缓存

同步的内容

缓存镜像从 Omnitouch APT 服务器同步 所有内容。根据内容类型，机制有所不同：

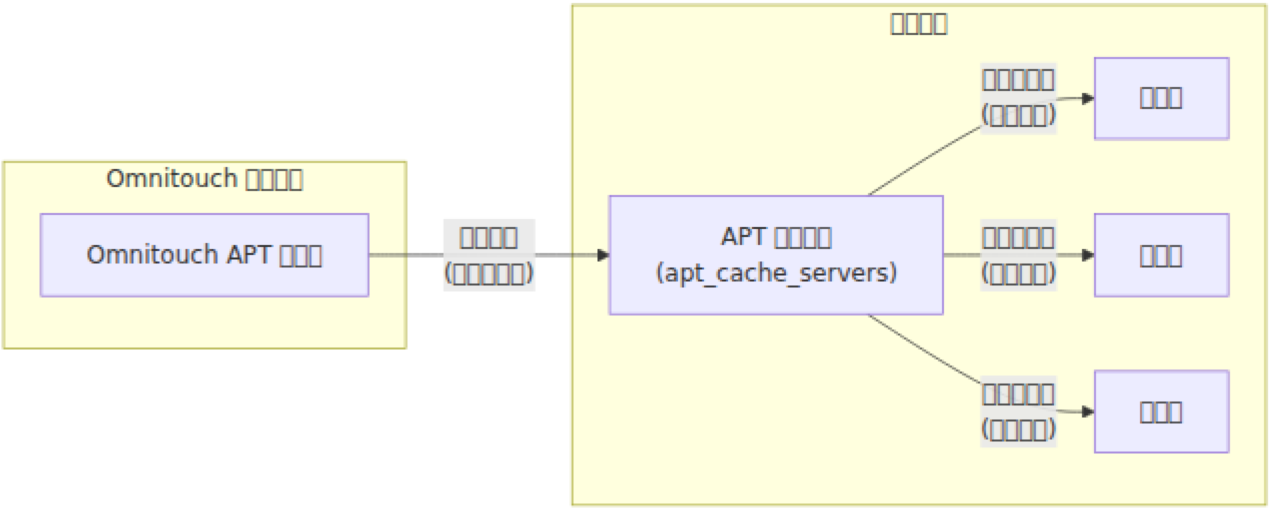
- **APT** 软件包（`/dists/` + `/pool/`，包括 Omnitouch 和 Ubuntu）通过 Debian 仓库索引使用 `curl` 进行同步。镜像获取上游的 `Release` 文件（如果未更改则短路），解析 `Packages.gz` 以获取每个软件包的权威文件名/SHA256/大小，并仅下载缺失的文件或其 SHA256 与本地副本不匹配的文件。每个下载的文件在提交之前都经过 SHA256 验证。
- 二进制发布和源代码压缩包（`/releases/`、`/repos/`）没有 Debian 风格的索引，因此退回到 `wget --recursive --timestamping`，仅重新下载具有更新 `Last-Modified` 的文件。

同步的内容目录：

路径	内容
`/dists/<distro>/`	APT 仓库元数据（软件包、发布文件）
`/pool/main/`	Omnitouch 自定义 .deb 软件包
`/<distro>/pool/main/`	Ubuntu 软件包及所有依赖
`/releases/`	GitHub 发布（Prometheus、Grafana、Zabbix 等）
`/repos/`	源代码压缩包（Erlang、Elixir、CGrateS_UI 等）

在初始同步后，缓存可以在没有互联网连接的情况下提供所有软件包。

工作原理



缓存镜像通过 HTTP 基本身份验证对所有内容进行身份验证。APT 软件包通过 `curl` 与 Debian 仓库索引（`Release` + `Packages.gz`）进行同步，仅下载 SHA256 与本地副本不同的文件，并在下载后验证每个文件；`/releases/` 和 `/repos/` 则退回到 `wget --recursive --timestamping`。无论哪种方式，后续同步仅获取新文件或更改的文件。

自动配置

当您的清单中存在 `apt_cache_servers` 组时，系统会自动：

1. 为所有主机设置 `use_apt_cache: true`（除非显式覆盖）
2. 从第一个缓存服务器的 `ansible_host` IP 派生 `apt_repo.apt_server`

最小配置示例

```yaml
apt_cache_servers:
  hosts:
    apt-cache-01:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # 缓存服务器从上游 Omnitouch 仓库同步内容
    remote_apt_server: "apt.<region>.omnitou.ch"
    remote_apt_user: "your-username"
    remote_apt_password: "your-password"
```

自动发生的事情：

- 所有主机（缓存服务器除外）获得 `use_apt_cache: true`
- 所有主机（缓存服务器除外）获得 `apt_repo.apt_server: "192.168.1.100"`
- 所有主机在不需要凭据的情况下从 `http://192.168.1.100:8080/` 拉取
- 缓存服务器从 `https://your-username:your-password@apt.<region>.omnitou.ch/` 同步软件包

覆盖自动行为

要强制直接访问，即使定义了缓存服务器：

```yaml
all:
  vars:
    use_apt_cache: false  # 强制直接访问，即使定义了缓存服务器

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

配置总结

场景 1：直接访问 APT 服务器（无缓存）

所有主机直接从 APT 仓库服务器拉取软件包。只需要 `apt_repo`°`remote_apt_*` 值是从中派生的。

```yaml
all:
  vars:
    use_apt_cache: false

    # APT 源和二进制下载的唯一真实来源
    apt_repo:
      apt_server: "apt.<region>.omnitou.ch"
      apt_repo_username: "user"
      apt_repo_password: "pass"
```

结果：所有主机生成 `deb [trusted=yes] https://user:pass@apt.<region>.omnitou.ch/ noble main`

场景 2：主机文件中定义 APT 缓存服务器（自动）

缓存服务器在您的清单中，并将由 Ansible 部署/同步。

```yaml
apt_cache_servers:
  hosts:
    cache-server:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # 缓存服务器从上游经过身份验证的仓库同步软件包
    remote_apt_server: "apt.<region>.omnitou.ch"
    remote_apt_port: 443
    remote_apt_protocol: "https"
    remote_apt_user: "user"
    remote_apt_password: "pass"

# 在 all: vars 中不需要配置:
# 所有内容自动从 apt_cache_servers 组派生
```

结果：

- 缓存服务器：从 `https://user:pass@apt.<region>.omnitou.ch/` 同步
- 所有其他主机：生成 `deb [trusted=yes] http://192.168.1.100:8080/noble noble main`（无凭据）

场景 3：主机文件中未定义远程 APT 缓存（手动）

缓存服务器存在于其他地方，并且已经设置（不由您的 Ansible 管理）。

```yaml
all:
  vars:
    use_apt_cache: true

    # 将所有主机指向外部缓存服务器
    apt_repo:
      apt_server: "192.168.1.100"  # 外部缓存服务器的 IP
      apt_repo_port: 8080           # 缓存通常在端口 8080 上运行

# 不需要 apt_cache_servers 组
# 不需要 remote_apt_* (缓存已在外部设置)
```

结果：所有主机生成 `deb [trusted=yes] http://192.168.1.100:8080/noble noble main`（无凭据）

完整示例

这是一个完整的工作示例。它显示了带有多个应用程序主机的缓存服务器配置：

```yaml
# APT 缓存服务器组
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:
    # 缓存服务器从上游经过身份验证的仓库同步软件包
    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"

# 应用服务器
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

# 全局配置
all:
  vars:
    # 自动配置（无需手动配置）：
```

```
    # - use_apt_cache: true□□ apt_cache_servers □□□□□□□□□
    # - apt_repo.apt_server: "10.179.1.114"□□□□□□□□□□□□□□
```

□□□□□□□□□□□

1. □□□□□□□**10.179.1.114**□□
   - □□□ `vars:` □□□□ `remote_apt_*`
   - □ `https://customer-username:customer-secure-token@apt.<region>.omnitou.ch/` □□□□□□□□
   - □□ nginx □□□ 8080 □□□□□
2. □□□□□**customer-hss01**□**customer-mme01**□**customer-dns01**□□
   - □□□□□ `apt_cache_servers` □□□
   - □□□□ `use_apt_cache: true`
   - □□□□ `apt_repo.apt_server: "10.179.1.114"`
   - □□□`deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
   - □□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□

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

□□□□□□□ Omnitouch APT □□□□□□□□□□□

- □□ Omnitouch □□□□□
- □□ Ubuntu □□□□□□
- □□ GitHub □□
- □□□□□□□□□□

重新同步是增量的：APT 软件包通过 SHA256 与上游 Debian 索引进行比较（如果上游 `Release` 文件未更改，则完全跳过），而 `/releases/` 和 `/repos/` 则使用 `wget --timestamping`。现有未更改的文件被跳过，使得重新同步快速。

注意： Omnitouch APT 服务器是所有软件包的唯一真实来源。Omnitouch 构建并发布软件包到它；在缓存镜像上运行 `services/apt_cache.yml` 然后同步当前发布的内容。

故障排除

APT 更新失败，401 未授权

症状：

```
无法获取 https://10.179.1.115/noble/dists/noble/main/binary-amd64/Packages  401  未授权
```

可能的原因：

- `apt_repo` 配置在 `all: vars:` 中定义，而不是 `apt_cache_servers: vars:` 中
- 主机尝试直接访问经过身份验证的仓库，而不是缓存
- `apt_repo_username` 或 `apt_repo_password` 不正确
- 源 IP 未在 Omnitouch APT 服务器上列入白名单
- 使用缓存凭据进行直接访问或反之亦然

解决方案：

1. 检查配置范围：确保带有凭据的 `apt_repo` 定义在 `apt_cache_servers: vars:` 中，而不是在 `all: vars:` 中
2. 验证缓存模式：使用缓存时，主机应连接到缓存服务器（端口 8080），而不是上游仓库
3. 检查生成的源：在失败的主机上，检查 `/etc/apt/sources.list.d/omnitouch.list`
    - 正确（缓存模式）：`deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
    - 不正确（凭据放在错误的位置）：`deb [trusted=yes] https://user:pass@10.179.1.115/noble noble main`

4. 验证凭据是否正确以适应您的部署模式
5. 确认您的公共 IP 是否在 Omnitouch 上列入白名单（如果使用直接访问）

二进制下载失败（节点导出器、**Zabbix** 等）

症状：Ansible 剧本在从 `/releases/` 路径下载文件时失败

可能的原因：

- `apt_repo` 凭据不正确（二进制下载在直接模式下从 `apt_repo` 派生其身份验证）
- 在缓存镜像服务器上，`remote_apt_user` / `remote_apt_password` 不正确
- 方案或端口错误。请记住，端口 `443` 表示 HTTPS，任何其他端口表示 HTTP。

解决方案：

1. 在直接模式下，验证 `apt_repo.apt_repo_username` / `apt_repo.apt_repo_password` 是否与 Omnitouch 提供的匹配
2. 在缓存镜像服务器上，验证 `remote_apt_*` 上游凭据
3. 确认派生的 `apt_download_base` 使用预期的主机/方案/端口（通过 `ansible-playbook ... -vvv` 检查或检查失败的 `get_url` 任务）

缓存服务器无法同步

症状：缓存服务器剧本无法下载软件包

可能的原因：

- 缓存服务器没有互联网访问
- `remote_apt_*` 凭据不正确
- 防火墙阻止对 Omnitouch 的出站连接

解决方案：

1. 验证缓存服务器是否可以在端口 443 上访问上游 Omnitouch APT 服务器
2. 检查 `remote_apt_*` 凭据
3. 检查防火墙规则以允许出站访问

相关文档

- 主机文件配置：清单和变量配置
- 配置参考：完整参数参考
- 部署架构：整体系统架构
- Proxmox 部署：将缓存服务器作为 LXC 容器部署

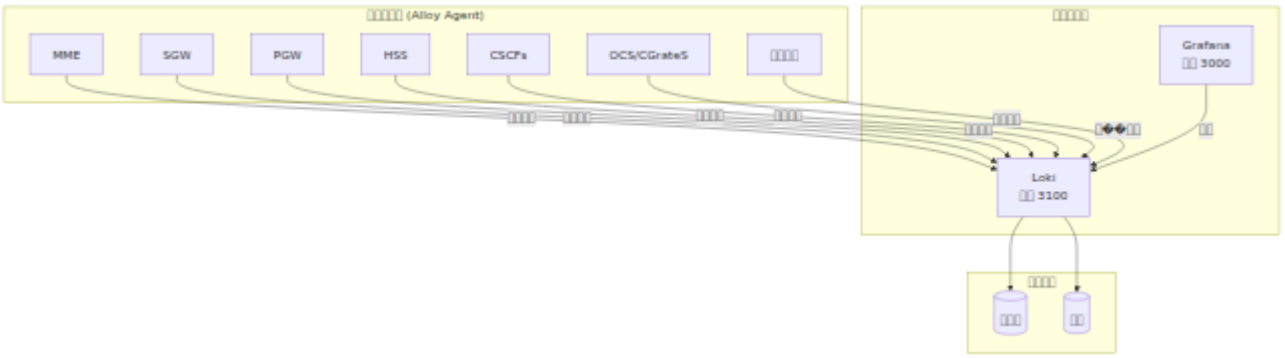
集中式日志记录

概述

OmniCore 包含一个集中式日志记录系统，该系统从所有组件收集日志，并通过 Grafana 进行搜索。该系统使用：

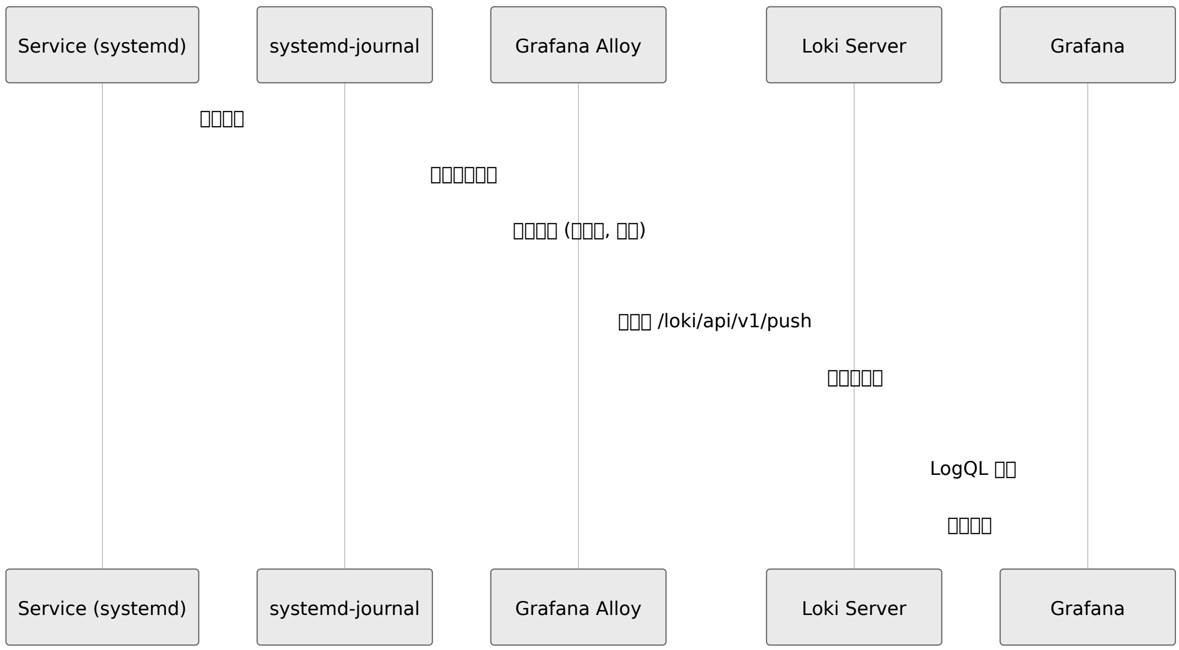
- **Grafana Alloy**：部署在所有虚拟机上的日志收集代理
- **Grafana Loki**：监控主机上的日志聚合和存储服务器
- **Grafana** 仪表板：为每种组件类型预构建的仪表板

架构



工作原理

日志收集流程



日志标签

每个日志条目都包含以下标签以便于过滤和查询：

标签	描述	示例
`job`	源虚拟机的主机名	`customer-mme01`
`hostname`	与 job 相同（为了兼容性）	`customer-mme01`
`component`	来自库存组的组件类型	`mme`, `pcscf`, `ocs`
`unit`	systemd 单元名称	`omnimme.service`
`level`	日志严重性	`info`, `warning`, `error`
`service`	Syslog 标识符	`omnimme`

□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□

1. □□□□□□□□□ `monitoring` □
2. □□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□

□□□□

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

□□□□ `monitoring` □□□

- □□□□□□□□ Loki □□□□□□□□□□□□□
- □□□□□□□□□□ Alloy □□□□□□□□□□□□

□□□□

Loki □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□	□□□	□□
□□□□□□□□	72 □□□3 □□	□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□

```
all:
  vars:
    loki_retention_period: "168h"  # 例如 7 天（以小时为单位）
```

Alloy 缓冲配置

当 Loki 无法访问时，Alloy 会在写前日志（WAL）中本地缓冲日志。缓冲段会被老化以防止无限制的磁盘增长：

设置	默认值	描述
WAL 最大段年龄	1 小时	超过 1 小时的缓冲日志段将被丢弃

当 Loki 无法访问超过段年龄时，最旧的缓冲日志将被丢弃。

Grafana 仪表板

为每种组件类型自动提供预构建的仪表板。

□□□□□

□□□	□□	□□
CSCF □□	□□ → CSCF □□	P-CSCF, S-CSCF, I-CSCF (OmniCSCF) □□
MME □□	□□ → MME □□	MME □□/□□□□□□□
SGW □□	□□ → SGW □□	SGW □□□□□□□
PGW □□	□□ → PGW □□	PGW PDN □□□□□
HSS □□	□□ → HSS □□	HSS Diameter □□□□□□□
OmniMessage □□	□□ → OmniMessage □□	□□□□□ SMPP □□
OCS/CGrateS □□	□□ → OCS/CGrateS □□	□□□□□ JSONRPC □□
CGrateS RPC □□	□□ → CGrateS RPC □□	□□□□□ RPC □□/□□□□□

□□□□□

□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□P-CSCF, S-CSCF, I-CSCF□
- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□10 □□□□

使用仪表板

1. 导航到 Grafana（通常是 `http://<monitoring-host>:3000`）
2. 转到 仪表板 → 日志 → 选择组件
3. 使用 搜索 变量过滤日志
4. 根据需要调整时间范围

查询日志

LogQL 基础

Loki 使用 LogQL 进行查询。基本语法：

```
{label="value"} |= "search string"
```

常见查询

来自特定主机的所有日志：

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

所有 **MME** 组件日志：

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

搜索所有 **CSCF** 的错误：

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

按 **systemd** 单元过滤：

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

组合过滤：

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

日志速率查询

每个组件的日志量：

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

CSCF 的错误率：

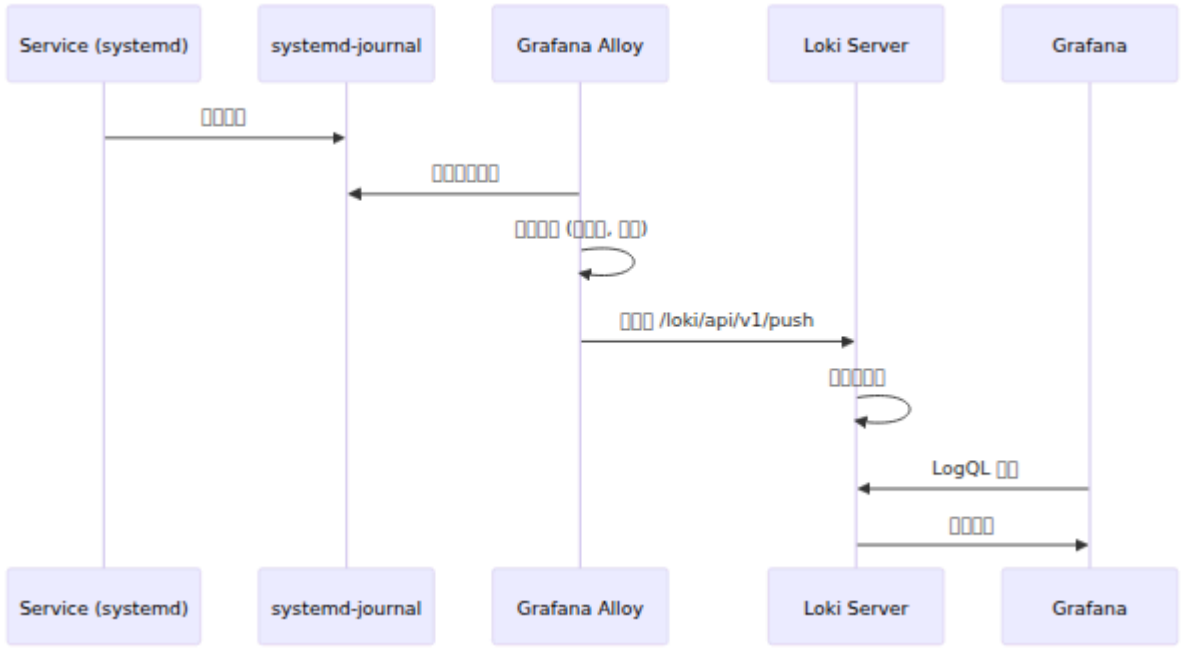
```
sum(rate({component=~"pcscf|scscf|icscf"} |~ "(?i)error" [5m]))
```

CGrateS JSONRPC 日志记录

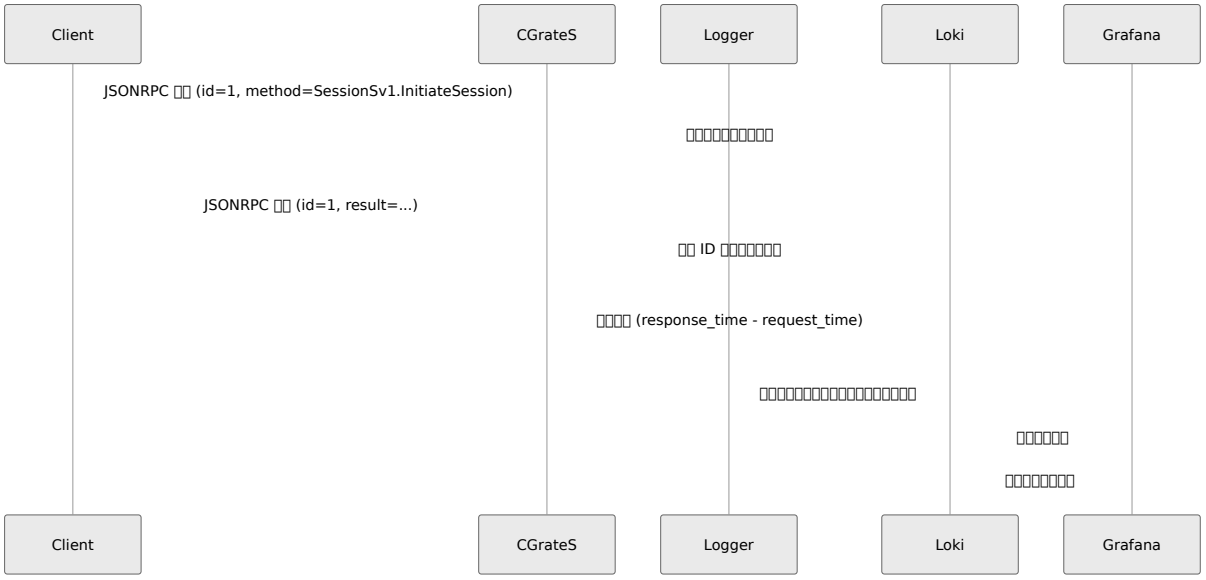
对于 OCS/CGrateS 部署，JSONRPC 流量被捕获并记录，具有请求/响应关联和延迟跟踪，以进行性能分析和调试。

JSONRPC 记录器在 `cgrates` 角色中定义，其日志文件 (`/var/log/cgrates/jsonrpc.log`) 由 Alloy 在 `ocs` 和 `cgrates` 主机组中收集。

□□


Service (systemd)
systemd-journal
Grafana Alloy
Loki Server
Grafana
□□□□
□□□□□□□
□□□□ (□□□, □□)
□□□ /loki/api/v1/push
□□□□□
LogQL □□
□□□□
Service (systemd)
systemd-journal
Grafana Alloy
Loki Server
Grafana

□□/□□□□


Client
CGrateS
Logger
Loki
Grafana
JSONRPC □□ (id=1, method=SessionSv1.InitiateSession)
□□□□□□□□□□□
JSONRPC □□ (id=1, result=...)
□□ ID □□□□□□□
□□□□ (response_time - request_time)
□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□
□□□□□□□□
Client
CGrateS
Logger
Loki
Grafana

□□□□

□□ CGrateS □□□□□□□□□□

端口	服务名称	协议	描述
2012	`rpc_json`	TCP	原始 JSONRPC 通过 TCP
2080	`http`	HTTP	HTTP API（过滤 `/metrics` 和 `/health`）

记录器会自动过滤掉：

- Prometheus 指标抓取（`GET /metrics`）
- 健康检查请求（`GET /health`）
- Gzip 压缩响应（二进制数据）

日志格式

JSONRPC 日志写入 `/var/log/cgrates/jsonrpc.log`。每个日志条目代表一个 已完成的请求/响应对，并包含延迟测量：

```
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":{...}}
```

日志字段

字段	描述	示例
`timestamp`	响应被接收时的 ISO 8601 时间戳	`2026-03-01T10:30:45.123456`
`port`	请求发送到的 CGrateS 端口	`2012`，`2080`
`service`	端口的服务名称	`rpc_json`，`http`
`request_id`	JSONRPC 请求 ID 以便关联	`1`，`42`，(空值表示 null)
`method`	JSONRPC 方法名称	`SessionSv1.InitiateSession`
`latency_ms`	毫秒级的往返延迟	`2.45`
`status`	结果状态	`ok`，`error`
`error`	如果状态为错误，则为错误消息	`INSUFFICIENT_CREDIT`
`src`	源 IP:端口（客户端）	`10.10.1.50:45678`
`dst`	目标 IP:端口（CGrateS）	`10.10.2.100:2012`
`request`	完整的 JSONRPC 请求负载（压缩 JSON）	`{"id":1,"method":"..."}`
`response`	完整的 JSONRPC 响应负载（压缩 JSON）	`{"id":1,"result":{...}}`

Grafana 仪表板

OCS / CGrateS 日志 仪表板包括用于 JSONRPC 分析的专用面板。

按方法的 **JSONRPC** 延迟

显示每个 JSONRPC 方法随时间变化的延迟的时间序列图。用于识别：

- □□□□□□□□□□□
- □□□□□□□□□□□□
- □□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

JSONRPC □□

□□□□ JSONRPC □□/□□□□□□□□□□□□□□

- □□□
- □□□□
- □□
- □□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□ □□ □□□□□□□□□□ JSONRPC □□□□□□□□□□□□□□□ `SessionSv1.InitiateSession`□□□□□□□□□□□□□

CGrateS RPC □□□□□

□□□ **CGrateS RPC** □□ □□□□□□□□□□□□□□□ API □□□□□□□□□□□□□□□

□□□□□□□□□□□

□□□□□“□□□□□□ 1234□□□□□”□□

1. □□ □□□ → □□ → **CGrateS RPC** □□
2. □ □□ □□□□□□□□□□□□□ ID□ `1234`
3. □ □□ □□□ □□ □□□□□□□□□
4. **RPC** □□□□ □□□□□□□□□□□□□□□□□□□/□□

□□□□□□□□□□□□□□□□□□□

□□	□□	□□
□□	□□□/□□□□□□□□□□□□□□□□□□	`1234`, `Account123`, `INSUFFICIENT_CREDIT`
□□	□ RPC □□□□□□□	`SessionSv1.InitiateSession`, `CDRsV1.ProcessEvent`
□□	□□□□□□□□	□□, □□, □□

□□□□□

□□□□□□□□□□□□

- □ **RPC** □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□

按方法过滤：

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

搜索请求/响应负载：

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

仅错误：

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

延迟分析

提取延迟作为指标（用于图表）：

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

查找慢请求（延迟 **> 100ms**）：

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

特定方法查询

CDR 处理：

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

会话管理：

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

账户操作：

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

服务管理

JSONRPC 记录器作为 systemd 服务在 OCS/CGrateS 主机上运行。

检查服务状态：

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

查看服务日志：

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

重启服务：

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

故障排除

没有 **JSONRPC** 日志出现

症状：JSONRPC 流量面板没有数据显示

可能原因：

- 记录器服务未运行
- ngrep 未安装
- 网络接口不匹配
- Alloy 未收集日志文件

解决方案：

1. 检查记录器服务状态：
   ```
   systemctl status cgrates-jsonrpc-logger
   journalctl -u cgrates-jsonrpc-logger -n 50
   ```
2. 验证 ngrep 是否已安装：
   ```
   which ngrep
   ```
3. 检查日志文件是否正在写入：
   ```
   tail -f /var/log/cgrates/jsonrpc.log
   ```
4. 验证 Alloy 是否在收集文件：
   ```
   journalctl -u alloy | grep jsonrpc
   ```

延迟图表没有数据显示

症状：JSONRPC 按方法的延迟面板显示“没有数据”

可能原因：

- 在选定的时间范围内没有 JSONRPC 流量
- 日志格式不匹配（正则表达式未匹配）

解决方案：

1. 验证在时间范围内是否存在日志：
   ```
   {job="cgrates-jsonrpc"} |= "method="
   ```

2. 检查日志中是否存在 `latency_ms` 字段
3. 扩大时间范围

高待处理请求计数

症状：调试日志显示许多存储的请求，但很少有记录的完成

可能原因：

- 没有响应的请求（客户端断开连接）
- 捕获响应在请求之前（数据包顺序）
- 请求和响应之间的请求 ID 不匹配

解决方案：

1. 记录器会自动清理超过 60 秒的待处理请求
2. 检查客户端与 CGrateS 之间的网络问题
3. 验证 CGrateS 是否在响应请求

文件日志

除了 systemd 日志，Alloy 还收集特定的日志文件：

FreeSWITCH 日志（应用服务器）

路径	作业标签
`/var/log/freeswitch/freeswitch.log`	`freeswitch`
`/var/log/omnitas-freeswitch.log`	`freeswitch-structured`

Nginx □□□OmniCRM□

□��□	□□□□	□��□□
`/var/log/nginx/access.log`	`nginx`	`access`
`/var/log/nginx/error.log`	`nginx`	`error`

CGrateS JSONRPC□OCS□

□□	□□□□
`/var/log/cgrates/jsonrpc.log`	`cgrates-jsonrpc`

□□□□

□□□□□□ Grafana □

□□□□ Loki □□□□□□□□□□□

□□□□□

- □□□□ Alloy □□□□□
- □□□□□ Loki □□□□□
- □□□□□□□□□□□□□
- Alloy □□□□□ 3100 □□□ Loki

□□□□□

1. □□□□□□□ Alloy □□□

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

2. □□□□□□□□□ Loki □□□

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

3. □□□□□□□□□□□□

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

4. □□□□□□□□□□□□□□□□□ TCP 3100

Alloy □□□□□□

□□□`/var/lib/alloy` □□□□□□□□□

□□□□□

- Loki □□□□□□□□
- WAL □□□□□

□□□□□

1. □□ Loki □□□□□□□□□
2. □□ Loki □□□□□□□□□□□□ WAL □□□□□
3. □□ Loki □□□□□□□□□□□□□□□□□

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

2. 验证压缩器是否在运行（检查 Loki 日志）
3. 如有必要，通过重启 Loki 手动触发压缩
4. 考虑增加磁盘分配或减少保留期

缺失组件标签

症状：日志出现，但 `component` 标签为 `infrastructure` 而不是预期值

可能原因：

- 主机不在预期的库存组中
- Alloy 配置在库存更改后未重新生成

解决方案：

1. 验证主机是否在正确的库存组中
2. 重新运行 Ansible 以重新生成 Alloy 配置：

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

3. 重启 Alloy：

   ```
   systemctl restart alloy
   ```

服务端口

服务	端口	协议	描述
Loki	3100	HTTP	日志摄取和查询 API
Loki	9096	gRPC	内部 gRPC（不用于外部）
Alloy	12345	HTTP	Alloy 指标和 UI
Grafana	3000	HTTP	仪表板访问

相关文档

- 监控与可观察性：Grafana、Prometheus、仪表板和警报
- 部署架构：整体系统架构
- 主机文件配置：库存配置
- 配置参考：完整参数参考

□□□□□□ (CS)

□□□□□□□□ (2G/3G □□□ SMS) □□□ `services/cs.yml` □□□□□□□□□□□□□□□□□ BSC□ OmniMSC □□□□□□ SS7/SIGTRAN □□□□

□□

`cs.yml` □□□□□□□□□□□□□□□□□□□□□□□□

□□	□□	□□□	□	□□
BSC	`circuit_switched/bsc`	`bsc`	BSC, MGW	□□□□□ + □□□□ (□□ 2G/3G □□□□□)
MSC	`circuit_switched/omnimsc`	`msc`	`omnimsc`	□□□□□□□□□□□
SS7	`omniss7`	`ss7`	`omniss7`	SS7/SIGTRAN □ (STP, HLR, SMSC, CAMEL, □□□)

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

BSC (`circuit_switched/bsc`)

□ Omnitouch APT □□□□□ **BSC** □ □□□□ **(MGW)** □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□□□ BSC □ MGW □□□□□
- □ MGW □□□□ systemd □□□□□□ **CPU** □□□□□□ `CPUSchedulingPolicy`□ `Priority` □ `AmbientCapabilities`□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□ **`active`**□6 □□□□□□□□ 5 □□□□□□□

MSC (`circuit_switched/omnimsc`)

安装 `omnimsc` 包并像 5GC NF 一样配置它：

- 解析 `group_vars_omnicore` 并从 `license_server` 组派生 `license_server_api_urls`。
- 渲染内置的 `runtime.exs.j2` 到 `/etc/omnimsc/`。如果设置了 `omnimsc_template_config`，则从 `group_vars/` 渲染自定义模板。
- 确保 CDR 输出目录 (`omnimsc_cdr_output_dir`，默认 `/var/lib/omnimsc/cdr`)。
- 为 Web UI / API 生成自签名 TLS 证书。
- 使用版本化配置 + `runtime.exs` 符号链接模式，在更改时重新启动。

SS7 (`omniss7`)

一个单一角色，安装 `omniss7` 包并为每个 SS7 角色渲染一个配置模板，由 `ss7_role` 主机变量选择：

`ss7_role`	模板	功能
`stp`	`stp.j2`	信号传输点 (SS7 路由)
`hlr`	`hlr.j2`	家庭位置寄存器
`smsc`	`ipsmgw.j2`	SMSC / IP-SM-GW
`camel`	`camel.j2`	CAMEL 服务控制
`tester`	`tester.j2`	信令测试/负载代理

附加选择器：

- 设置 `ss7_protocol: m2pa` 会渲染 `m2pa.j2`（一个 M2PA 端点），而不是 `tester` 模板。
- 设置 `ss7_template_config`（`group_vars/` 下的路径）会覆盖 所有 内置模板，使用主机特定的配置。

与其他 Elixir 组件一样，`omniss7` 使用版本化配置 + `runtime.exs` 符号链接模式，并将符号链接镜像到 `/opt/omniss7/releases/*`。

相关文档

- 服务剧本 - 所有服务剧本和部署层次结构
- 5G 核心 - 5G 独立核心和共享的 `5gc_nf` 角色
- 组变量配置 - 自定义配置模板（例如 `ss7_template_config`、`omnimsc_template_config`）
- **OmniCore** 配置: https://docs.omnitouch.com.au/docs/repos/OmniCore - 组件配置文件中的内容

配置参考

概述

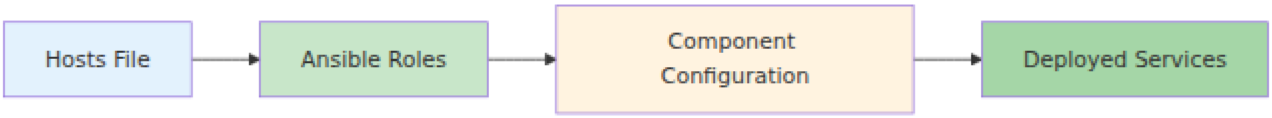
本文档提供了通过主机文件配置 OmniCore 部署的全面参考。配置主要在主机清单文件中定义，现代部署所需的组变量覆盖最小。

有关特定产品的文档，请参见：

- **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

配置方法

现代 OmniCore 部署使用简化的配置模型：



关键原则：大多数配置直接在主机文件中定义。角色默认值处理许多设置。然而，一些组件仅通过 `group_vars` 配置，并没有角色默认值。OmniCRM 是一个显著的例子（见下文）。在这些情况下，缺失的值会导致运行失败，而不是回退到默认值。

> 安全提示：几个角色默认值是不安全的占位符，仅用于实验室使用，必须 在任何实际部署中被覆盖。示例包括数据库密码（例如 `sql_pass: password`，`omnitouch2024`）和 APT 存储库凭据（`omni`/`omni`）。在部署之前，请始终在您的清单或组变量中设置真实、唯一的秘密。

网络规划

在配置主机之前，请查看 IP 规划标准 以获取以下方面的指导：

- 网络分段策略
- IP 地址分配
- 子网组织
- 公共 IP 处理

常见主机参数

#待办事项 - 只需查看 hosts-file-configuration.md

服务特定标志

```
cdrs_enabled: True               # 启用 CDR 生成
in_pool: False                   # 排除在负载均衡池外
populate_crm: False              # 用初始□据填充 CRM
```

全局变量（all:vars)

`all:vars` 部分包含部署范围的设置。现代部署使用最少的全局变量，许多设置由角色默认值覆盖。请注意以下例外（OmniCRM、Grafana SMTP），在这些情况下，必须显式提供配置，因为不存在角色默认值。

必要的全局变量

身份验证与访问

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

替代方案：使用 SSH 密钥而不是密码：

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

客户身份

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

PLMN [illegible]

```yaml
plmn_id:
  mcc: '001'              # [illegible]3 [illegible]
  mnc: '01'               # [illegible]2-3 [illegible]
  mnc_longform: '001'     # [illegible] MNC[illegible] 3 [illegible]

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

[illegible]

[illegible]

```yaml
network_name_short: Omni
network_name_long: Omnitouch
tac_list: [10100,100]              # [illegible] TAC [illegible] MME [illegible]
```

[illegible] UE [illegible] > [illegible]

DNS [illegible]

```yaml
manage_resolv_conf: True          # [illegible] False [illegible] Ansible [illegible]
/etc/resolv.conf
```

[illegible] `manage_resolv_conf` [illegible] `False` [illegible]Ansible [illegible] `/etc/resolv.conf`[illegible] DNS [illegible] `all:vars` [illegible]

APT [illegible]

apt_repo □□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□`remote_apt_*`□□ `apt_download_base` URL □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□ `apt_cache_servers` □□□

- `use_apt_cache` □□□□□□ `True`□□□□□□□□□ `False`□
- `apt_repo.apt_server` □□□□□□□□□□□□□□□□ IP

```
use_apt_cache: False              # □□□□□□□□□□True = □□ APT □□□□

apt_repo:
  apt_server: "apt.<region>.omnitou.ch"  # Omnitouch □□□ FQDN□□□□
IP□
  apt_repo_username: "your-username"     # □□□□□□□
  apt_repo_password: "your-password"     # □□□□□□□
  # apt_repo_port: 443                   # □□□□□□□ 443□HTTPS□□
                                         # □□□□□□□□443 -> https□□□□
□ http□
```

□□□□□□□□□ **HTTPS / 443** □□□□□□□□□□□□□□□□□ **HTTP / 8080**□□□□□□□□□□□□ □`apt_cache_servers` □□□□□□□ `remote_apt_*`□□□□□□□□□□□ Omnitouch □□□□

□□□APT □□□□

□□□□□□□

```
# □□□□□□□□□□□□□□□ `license_server` □□□□□□□
# □□□□□□□□□□□□□ https://<ansible_host>:8443/api□
license_server_api_urls: ["https://10.10.2.150:8443/api"]
license_enforced: false           # □□□□□□□ false
(roles/omnihss/defaults/main.yml)
```

□□□□□□□□□□□ `license_server` □□□ `license_server_api_urls` □□□□□□□□□□□□□□□□□□□□

MME □□□□□□□□□□ S-GW □ P-GW □□□□□□□□ `omnimme` □□□□□□□□□□□□□□□□□□□□ `home`/`roaming` □□□□□□□□□□ `dns_lookup`□□□ DNS □□□□□□□□ `random_peer`□□□□□□□□□□□□□

```
omnimme:
  sgw:
    selection_method: random_peer        # □□□□□...
    # selection_method:
    #   home: random_peer
    #   roaming: dns_lookup
  pgw:
    selection_method:
      home: random_peer
      roaming: dns_lookup
```

AMF □□

```
# tac_list □□□ AMF □ supported_ta_list□□□□□□ TAC □□
# □ NG □□□□□□□□□ gNB □□□□□□□□□□AMF □□□
# □ TAC 1□□□□□□□□□□□ TAC 1 □ gNB□
tac_list: [1, 3]

# □□ NAS □□□□□ NEA0□□□□□ true□□□ omniamf
# >=20260727 □□ NEA2□□□□□□□□□□□ NAS□□□
# □□□□□□□□□□□□□□□□ 5G □□□□□□□□□□□□□
# □□ NEA2 □□□□□ UE □□□□□□□□□□ false□NAS □□□□NIA2□□□□□□
amf_force_null_ciphering: true
```

SAEGW □□

```
mtu: 1400                          # □□□□□□
```

OmniHSS □□

OmniHSS □□□□□□□□□□□□□□□□□□□□□□ `dra`□`mme`□`pgwc`□`scscf`□`icscf`□`pcscf` □ `applicationserver` □□□□□□□□□ `packet_core_dra_support` / `ims_dra_support`□□`exclude_diameter_peers` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ HSS □□□□□□□

```
omnihss:
  exclude_diameter_peers:          # 可选；默认为 [] （不排除任何）
    - customer-as01
    - customer-as02
```

- 条目与 **`inventory_hostname`** 匹配，而不是直径 FQDN 或 IP。
- 排除适用于每个自动发现的组，因此一个列表覆盖主机，无论它位于哪个组中。
- 名称必须在清单中存在。角色在部署之前会验证这一点，因此拼写错误会导致运行失败，而不是静默地将对等体留在 `/etc/omnihss/runtime.exs` 中。
- 在设置 `omnihss` 字典的其他键的地方设置它（每个主机，或在组的 `vars:` 下）。由于 Ansible 替换而不是合并字典，它必须与其他键位于同一 `omnihss` 块中。
- 这仅影响直径对等体列表。`config :hss, ims:` 下的 S-CSCF SIP 对等体列表无关，并且不受此键的过滤。

> DRA 有其自己的□立机制：在主机上设置 `dra_peer_exclude: True` 会将其从 OmniDRA 对等体列表中移除。这两者是□立的。从 HSS 中排除主机并不会将其从 DRA 中排除，反之亦然。

OmniCRM 设置

OmniCRM 配置专门来自客户的 `group_vars`（在 `hosts/` 下）；角色中没有默认值用于站点秘密。

首选路径（`omnicrm_deploy_mode: package`，角色默认）：从 `apt_repo` 安装通用 `omnicrm-api` / `omnicrm-ui` 包，将 `crm_config.yaml` 模板化到 `/etc/omnicrm/crm_config.yaml`，同步 `deployment_assets/`，并 `systemctl restart omnicrm-api`。UI 品牌来自 `ui:` 块（运行时 `GET /crm/public/ui-config`）。此路径需要不在主机上进行 yarn 构建，并且不需要 `crm_env_override`。

遗留路径（`omnicrm_deploy_mode: source`）：git 存档 + 主机 venv + yarn 构建；仅在该模式下需要 `crm_env_override`。

```
omnicrm_deploy_mode: package      # package （默认） | source
omnicrm_package_version: ""       # 可选的 apt 版本固定用于 omnicrm-api/ui
crm_config.yaml                   # API + ui: 配置 （group_vars；必需）
deployment_assets/                # 可选的标志/图标用于
/etc/omnicrm/deployment_assets
crm_env_override: my-site.env     # 仅在源模式/本地 UI 回退时需要
cell_sites: cellSites.txt         # 可选：UI 的小区站点□据
message_templates: constants.js   # 可选：小区广播消息模板
site_data: site_data.json         # 可选：小区广播地理边界
```

Grafana SMTP 设置

由监控角色用于 Grafana 警报电子邮件。`grafana_smtp_user` 和 `grafana_smtp_password` 是必需的（没有默认值：您必须设置它们）；`grafana_smtp_host` 默认为 `in-v3.mailjet.com:587`。参见 `roles/monitoring/templates/grafana/grafana.ini`。

```yaml
grafana_smtp_user: "smtp-user"            # 必需，没有默认值
grafana_smtp_password: "smtp-password"    # 必需，没有默认值
grafana_smtp_host: "in-v3.mailjet.com:587"   # 可选，这是默认值
```

IMS 设置

```yaml
ims_dra_support: False              # 通过 DRA 路由 IMS
enable_homer: False                 # 启用 Homer SIP 捕获
```

RAN 监控配置

```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
```

□□□□□

```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'
```

漫游 **DNS** 服务器

```yaml
roaming_dns_servers:
  wildcard: ['10.0.99.1']
  # 运营商特定 DNS（基于 PLMN）
  123456:                              # 示例运营商 1
    - '10.10.2.197'
  654321:                              # ???例运营商 2
    - '10.10.0.4'
```

本地用户（**SSH** 密钥）

```yaml
local_users:
  usera:
    name: 示例用户 A
    public_key: "ssh-rsa AAAAB3Nza..."
  userb:
    name: 示例用户 B
    public_key: "ssh-ed25519 AAAAC3..."
```

虚拟化程序配置

Proxmox

一个站点作为 VM 或 LXC 容器部署，通过 `deployment_type`（默认 `vm`）在 `proxmoxServers` 条目中选择。站点中的所有条目必须一致；混合将导致验证失败。有关完整的操作步骤，请参见 Proxmox 部署。

VM 站点

```yaml
proxmoxServers:
  customer-prxmx01:
    # deployment_type 省略 → 默认为 "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        # 可选的 cloud-init 用户名；默认为第一个 local_users 键
    proxmoxTemplatePassword: omnitouch    # 可选的 cloud-init 密码；默认为 cloud-init 用户名
    storage: SSD_RAID0            # 可选，默认 VM 存储
```

LXC 站点

```yaml
proxmox_lxc_nameserver: "1.1.1.1"   # 可选，在创建时注入到 LXCs

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    # 可选，rootfs 回退
```

组级覆盖（两种类型）

```yaml
dns:
  vars:
    proxmox_interface: vmbr0                # 必需：桥接
    gateway: 10.10.0.1                      # 必需
    netmask: 255.255.255.0                  # 必需
    vlanid: 100                             # 可选
    proxmoxLxcCores: 2                      # 可选（仅 LXC）
    proxmoxLxcMemoryMb: 4096                # 可选（仅 LXC）
    proxmoxLxcDiskSizeGb: 30                # 可选（仅 LXC）
    proxmoxLxcRootFsStorageName: SSD_RAID0 # 可选（仅 LXC）
    host_vm_network: vmbr1                  # 可选桥接覆盖
```

VMware vCenter

```
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
```

相关文档

- IP 规划标准 - 网络架构和 IP 分配指南
- 主机文件配置 - 如何构建主机文件
- 组变量配置 - 何时以及如何使用 group_vars
- Netplan 配置 - 次要 IP 和多 NIC 设置
- 部署架构 - 组件如何交互
- APT 缓存系统 - 包管理
- 许可证服务器 - 许可证配置

产品文档

有关详细的操作指南和高级配置：

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

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

部署架构概述

概述

本文档提供了Omnitouch网络服务的蜂窝网络软件如何使用Ansible进行部署的完整视图，展示了所有组件如何结合在一起以创建一个可工作的4G/5G网络。

请参见IP规划标准以获取详细的组件放置、IP地址分配指南和公共IP处理。

完整部署示例

0. 基础设施提供（可选）

对于Proxmox部署，在配置之前提供VM/LXC：

```
# 在Proxmox上部署VM和/或LXC容器（每个主机的deployment_type决定选择哪个）
ansible-playbook -i hosts/Customer/hosts.yml
util_playbooks/proxmox.yml
```

请参见：Proxmox VM/LXC部署

1. □□□□□□□□□□□□□□

```
# □□□□□□□□□□□□□□
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15

hss:
  hosts:
    customer-hss01:
      ansible_host: 10.10.2.140

# ... □□□□□□□
```

□□□□□□□□□□

2. □□□□group_vars□

`group_vars`□□□□□□□□□□□□□

4. 许可证配置

```
# 指向许可证服务器的组件
license_server_api_urls: ["https://10.10.2.150:8443/api"]
license_enforced: true
```

请参见：许可证服务器

5. 执行部署

您可以部署单个组件。例如，运行`services/twag.yml`。`services/all.yml`剧本处理所有内容。使用`--limit=myhost`或`--limit=mmee,sgw`来限制您工作的主机。

```
# 部署完整网络
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml

# 或部署特定组件
ansible-playbook -i hosts/customer/host_files/production.yml
services/epc.yml
ansible-playbook -i hosts/customer/host_files/production.yml
services/ims.yml
```

部署模型

有两种方法可以将OmniCore工具包放到控制器上并运行它。

模型A：在控制器上克隆**repo**

传统模型：将此存储库克隆到您的Ansible控制器上，将您的库存保存在`hosts/`下，并直接从检出运行剧本。这是上述部署示例所假设的模型。

模型B：自我提供的`omnicore` **.deb**

OmniCore也被打包为一个自包含的`omnicore` Debian软件包，因此控制器可以从apt安装工具包，而不是克隆repo。

- □□□□□□□ □□□□□□`name: omnicore`□□□□`packaging/nfpm.yaml`□□□Ansible□□□□□□*vendored*□□□□□□□□Python□□□□`/opt/omnicore/python`□□□□□□wheelhouse□`/opt/omnicore`□□□□□□□`packaging/postinstall.sh`□□□□□□□□+ wheelhouse□□□□□□□□venv□`/opt/omnicore/venv`□□□□□`/usr/local/bin`□□□□`ansible*`□□□□□`PATH`□□□□□□□□□□□`uv.lock`□□□Ansible□□Python□□□□□□□ Ansible□□□□□Python□□□□□□□□□□□□□□□□□`git`□`rsync`□`openssh-client`□□apt□□□ □□□□□**amd64**□��□**Ubuntu 22.04+**□glibc ≥ 2.35□□
- □□□□□□□□ `packaging/build.sh`□repo□□□□`/opt/omnicore`□□□ `hosts/`□`.git/`□□□□□□□□□□□□□`requirements.yml`□□□Ansible□□□□□□□□□+□ □Python□□□wheelhouse□□□□□□□□□□□□□□□□□□nfpm□□`.deb`□
- □□□□□□□□ `.github/workflows/build-deb.yml`□□□□□□`main`□□□`build.sh`□ □`.deb`□□□`_metadata.json`□□□□□□□GitHub□□□□□apt□□□□ `discover_github_debs.py`□□□□□□□□□□□□□□□□□aptly□□□□□□apt□□□

□□□□□

```
# 1. □apt□□□□□
apt-get install omnicore

# 2. □□□□□□□□□
#    □□□□□□□□□□□ - □□□/opt/omnicore□□□
vi /etc/omnicore/inventory

# 3. □□□□□□□□□□□□□
cd /opt/omnicore
ansible-playbook -i /etc/omnicore/inventory services/all.yml
```

□□□□□□`/etc/omnicore/inventory`□□□□□□`/opt/omnicore`□□□□□□□□□□□□□□□□□□

- 组变量配置 - 自定义
- APT缓存系统 - 软件包管理
- 许可证服务器 - 许可证管理
- 监控与可观察性 - Grafana, Prometheus, 警报和仪表板
- 集中日志记录 - Loki和Alloy日志收集

产品文档

有关配置每个组件的详细信息:

- **OmniCore**（4G/5G分组核心）: https://docs.omnitouch.com.au/docs/repos/OmniCore
    - OmniHSS, OmniSGW, OmniPGW, OmniUPF, OmniDRA, OmniTWAG
- **OmniCall**（语音和消息）: https://docs.omnitouch.com.au/docs/repos/OmniCall
    - OmniTAS, OmniCall CSCF, OmniMessage, OmniSS7, VisualVoicemail
- **OmniCharge/OmniCRM**（计费）: https://docs.omnitouch.com.au/docs/repos/OmniCharge
- 主文档: https://docs.omnitouch.com.au/

[illegible]

[illegible] `iptables` [illegible] `ip6tables` [illegible]

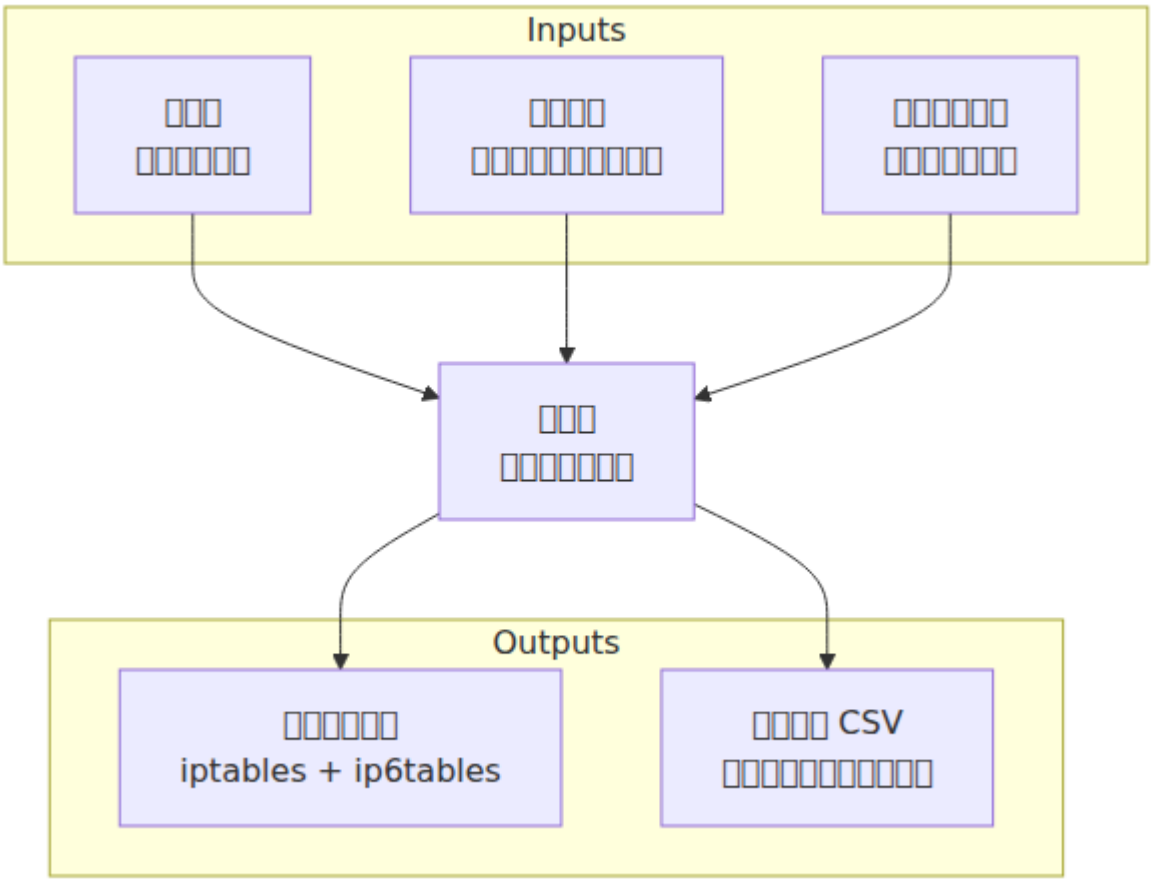
- [illegible]
- [illegible]/[illegible]CSV[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



 -

```
- id: '13'                          # 
  dst: hss                          # 
  service: diameter                 # 
3868/tcp+sctp)
  purpose: 'Diameter'               # 
  sources: [{ zone: internal }]     # 
```

参数	类型	必需	默认	描述
`id`	字符串	是	-	流的稳定标识符。出现在每个生成规则的注释中，格式为 `ans:<dst>:<id>`，以便可以追溯到模型。
`dst`	字符串	是	-	规则落在的库存组，或伪目的地 `all`（每个主机）和 `five_g_sbi`（5G 网络功能组）。目的地组未在站点上部署的流不会生成规则和矩阵行。
`service`	字符串	是*	-	由目标角色声明的命名服务（见 服务目录）。端口从角色的声明中查找，因此流永远不会重申端口号。
`proto_ports`	列表	是*	-	`service` 的字面替代，适用于未绑定到单个角色的流（例如 `all` 主机流）。每个条目为 `PORT/proto`（`tcp`、`udp`、`sctp`；端口可以是 `LO-HI` 范围）或裸协议（`icmp`、`esp`、`proto 89`）。
`purpose`	字符串	是	-	流的人类描述。显示在端口矩阵的 `service` 列中。
`sources`	列表	是	-	允许的源，作为源令牌的列表。见 源类型。

* 一个流提供 要么 `service` 要么 `proto_ports`，不能同时提供。

源类型

流的 `sources` 列表中的每个条目都是一个单键令牌，解析为站点的配置和库存。解析为空（例如空访问列表）的令牌不会贡献任何规则：它永远不会作为占位符发出。

令牌	解析为
`{ zone: <name> }`	该平面区域的 CIDR（`firewall.zones.<name>`）。
`{ list: <name> }`	操作员访问列表，例如 `allowed_oam_subnets`。
`{ group: <name> }`	库存组的主机地址，加上该角色的跨站点对等体。见 跨站点对等体。
`{ ue_pools: true }`	站点的 UE / 订阅者池（`firewall.ue_pools`）。
`{ dra_peers: true }`	DRA 主机自身直径对等体列表中配置的每个地址。
`{ roaming: true }`	漫游（IR.21 / GRX-IPX）范围。目前任何源：见 操作注意事项。
`{ literal: "..." }`	一个不从库存派生的自由文本源。

服务目录

端口号 由打开它们的角色拥有，而不是由中央模型拥有。每个暴露端口的角色在其角色目录中提供一个 `firewall_services.yml`，以其服务的库存组为键（例如 `cscf` 角色服务多个组，因此它声明每个组）。这将端口定义与打开它的代码保持在一起：

```
# roles/omnihss/firewall_services.yml
firewall_services:
  hss:
    diameter: [3868/tcp, 3868/sctp]   # S6a / Cx-Dx / Sh
    api_oam:  8080/tcp                # HSS API (OAM)
    api_nf:   8443/tcp                # HSS API (由其他 NF 查询)
    postgres: 5432/tcp                # HSS 对等体之间的逻辑复制
```

解析器将每个角色的 `firewall_services.yml` 聚合到一个目录中。流的 `service:` 在该流的 `dst` 组的目录中查找，因此模型仅表达 关系（谁可以到达 `hss` 的 `diameter` 服务），而 端口 与角色同在。添加或移动角色中的监听器并更新其 `firewall_services.yml`；中央模型中的内容不会改变。

这种拆分具有中央模型所缺乏的安全网：漂移检查 比较角色声明与实际在监听的主机。

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

□□□□□□□□□□□□

- □□□ - □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `firewall_services.yml`□□□□□□□□ □□□□□□□□□□□
- □□□ - □□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```yaml
firewall:
  allowed_oam_subnets:           ['10.8.0.0/24']       # 管理源（SSH、UI、抓取、管理 API）
  allowed_ran_subnets:           ['10.20.0.0/16']      # RAN 信令源（S1AP、NGAP、GTP-U、Abis）
  allowed_ue_voice_subnets:      ['100.64.0.0/16']     # 外部 UE SIP 进入 P-CSCF
  allowed_carrier_voice_subnets: ['203.0.113.0/24']    # 运营商互联 SIP
  zones:                                               # 平面区域：每个平面的真实子网
    internal:    ['10.8.0.0/16']                       # 受信任的核心超网
    mobile_core: ['10.8.1.0/24']                       # 移动核心 NF（EPC + 5GC + CS 核心）
    core_svc:    ['10.8.2.0/24']                       # 核心服务平面（HSS、DRA、OCS、OAM）
    ims:      ['10.8.3.0/24']                          # IMS 平面（CSCF、TAS）
    ue_svc:   ['10.8.4.0/24']                          # UE 可达平面（P-CSCF、DNS、ePDG）
    core_v6:  ['2001:db8:0:1::/64']                    # 核心 NF 的 IPv6 地址（非 UE）
  ue_pools: ['100.64.0.0/16', '2001:db8:64::/48']      # UE / 订阅者 IP 范围（v4 和 v6）
  extra_rules:                                         # 手动接受规则（见下文）
    - { proto: tcp, dport: '8443', source: '203.0.113.10/32', comment: '合作伙伴 API' }
```

只有 `allowed_oam_subnets` 是强制性的。每个其他列表可以在缺少该功能的站点上留空（`[]`）：没有 RAN 的站点设置 `allowed_ran_subnets: []`，没有 IPv6 核心的站点设置 `core_v6: []`，没有 UE 池的站点设置 `ue_pools: []`。空列表仅会为使用它的流生成没有规则（见下表的 `Required` 列）。

参数	类型	必需	默认	描述
`allowed_oam_subnets`	列表	是	-	管理源子网。管理 SSH、OmniWeb UI、Prometheus 抓取和管理 API。如果为空，生成器将停止并报错，因为空值将使管理没有允许的源，并在强制执行下导致锁定。
`allowed_ran_subnets`	列表	否	`[]`	RAN 信令的源子网（S1AP、NGAP、GTP-U、Abis）。在没有 RAN 的站点上可以为空。
`allowed_ue_voice_subnets`	列表	否	`[]`	外部 UE SIP 进入 P-CSCF 的源子网。
`allowed_carrier_voice_subnets`	列表	否	`[]`	运营商互联 SIP 的源子网。
`zones`	映射	是	-	平面区域，将每个平面名称映射到其真实子网。见 区域参数。
`ue_pools`	列表	否	`[]`	UE / 订阅者 IP 范围。如果站点没有则为空。
`extra_rules`	列表	否	`[]`	应用于每个主机的手动接受规则。见 手动规则参数。

区域参数

每个区域将平面名称映射到 CIDR 列表。仅设置站点使用的平面；其余保持为空。

区域	描述
`internal`	受信任的核心超网。最广泛的内部信令源。
`mobile_core`	跨代移动核心网络功能：EPC（MME、S/P-GW、UPF）、5GC（AMF、SMF 和 SBI 网格）和 CS 核心（MSC、SS7/SIGTRAN）。
`ims`	IMS 平面（I-CSCF、S-CSCF、TAS）。
`ue_svc`	UE 可达服务平面（P-CSCF、DNS、ePDG）。
`core_svc`	核心服务平面（HSS、DRA、OCS、OAM）。
`core_v6`	IPv6 核心服务地址。如果站点没有 IPv6 核心则为空。

手动规则参数

`extra_rules`（全局，所有主机）和每主机的 `firewall_extra_rules` 附加原始接受规则，以应对模型未覆盖的情况。包含 `:` 的 `source` 应用于 IPv6。

参数	类型	必需	默认	描述
`proto`	字符串	否	-	协议，例如 `tcp`、`udp`、`sctp`。
`dport`	字符串	否	-	目标端口或范围。
`source`	字符串	否	-	源 CIDR。值中的 `:` 应用于 IPv6。
`comment`	字符串	否	`override`	记录在规则注释中的标签。

操作模式

应用模式由 `firewall_mode` 控制。

□□	□□
`monitor`	□□□□□□□□□□□□ `INPUT` □□□□□□ `ACCEPT`□□□□□□□□□□□□□□□ `FW-AUDIT`□□□□□□□□□□□□□□□

SIGTRAN 地址会悄然中断流量。如果稍后需要逐地址精度，可以为服务命名其绑定的 NIC；在此之前，解析保持基于事实和完整。

通信/端口矩阵

端口矩阵是流模型针对站点库存解析的结果，扁平化为每个源、目标主机和单个端口/协议一行。它是审核或交给上游防火墙的形式。生成它的命令为：

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

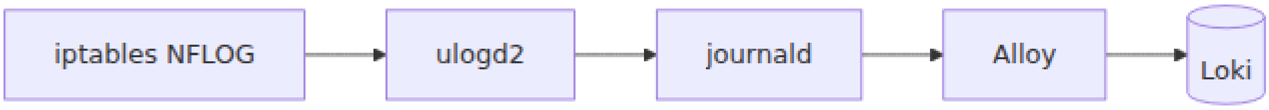
这将写入 `generated/firewall-port-matrix-exploded.csv`（按站点，不提交）。列包括：

列	描述
`id`	流标识符。
`service`	流的目的。
`source`	解析的源 CIDR 或地址。
`destination_role`	目标库存组。
`destination_host`	目标主机名称。
`destination_ip`	目标主机地址。
`port`	目标端口或范围（对于仅协议条目，如 ICMP 或 ESP，为空）。
`protocol`	协议。

仅当其目标在站点上部署且其源解析为真实地址时，行才会出现。未部署的角色和未解析的源不会生成行。

锁定保护

应用规则受到自动回滚的保护，由 `firewall_rollback_guard`（默认开启）和 `firewall_rollback_timeout` 控制：



1. [illegible]
2. [illegible]
3. [illegible]
4. [illegible]
5. [illegible]

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
`firewall_rollback_guard`	[illegible]	[illegible]	`true`	[illegible]
`firewall_rollback_timeout`	[illegible]	[illegible]	`120`	[illegible]

IPv6 [illegible]

[illegible] CIDR [illegible] `:` [illegible] IPv6[illegible] `ip6tables`[illegible] IPv4 CIDR [illegible] [illegible] IPv6 [illegible] IPv6 [illegible] IPv6 [illegible] `core_v6: []`[illegible]IPv6 [illegible] `INPUT` [illegible] `DROP`[illegible] IPv4 [illegible] IPv6 [illegible] IPv6 [illegible]

[illegible]

[illegible] NFLOG [illegible] Loki[illegible]

前缀	意义
`FW-FLOW` / `FW-FLOW6`	在监控模式下看到的新连接。构建实际使用的流量的实时流图。
`FW-AUDIT` / `FW-AUDIT6`	在监控模式下的通过流量：在强制执行下将被丢弃，但在这里被允许。在强制执行之前审核这些。
`FW-DROP` / `FW-DROP6`	在强制执行下实际丢弃的通过流量。

示例 Loki 查询：

```
# 一旦此主机被强制执行，将被阻止的流量
{job="firewall"} |= "FW-AUDIT"

# 在强制执行下被丢弃的流量，按主机
sum by (host) (count_over_time({job="firewall"} |= "FW-DROP"
[5m]))
```

操作注意事项

- 漫游和字面源当前为任何源。源为 `roaming` 或 `literal` 的流以没有源限制（`0.0.0.0/0`）发出。在漫游范围解析为真实 CIDR 之前，将它们视为开放。
- **ICMP** 从任何源被接受。生成的规则集无条件允许 ICMP，这比为 ping 流显示的管理范围更广。
- 在没有转发规则的情况下不要强制执行用户平面主机。在强制执行下，`FORWARD` 策略设置为 `DROP`，且未生成任何转发规则。在 UPF、PGW 或 SGW 上强制执行而不添加转发规则将破坏用户平面。请先验证。
- 管理列表较粗。`allowed_oam_subnets` 一起管理 SSH、UI、抓取和管理 API。需要从与其管理的网络不同的网络抓取的站点需要一个 `extra_rules` 条目来将它们分开。

□□□□□

□□

`group_vars` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□SIP □□□Diameter □□□□□SMS □□□□□□□□□□□□□□□□□□□□

在主机文件中引用

```
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   # 在 group_vars 中引用您的模板文件名
```

发生了什么：

1. Ansible 找到 `smsc_template_config: smsc_controller.exs`
2. 在 `hosts/Customer/group_vars/smsc_controller.exs` 中查找
3. 使用 Jinja2 进行模板化（可以使用 `{{ inventory_hostname }}`、`{{ plmn_id.mcc }}` 等）
4. 部署到 `/etc/omnimessage/runtime.exs`
5. 重启服务

没有 `smsc_template_config` 时，使用角色的默认模板。

配置详情：请参见 https://docs.omnitouch.com.au/docs/repos/OmniCall

示例 **2**：配置文件集合（**OmniTAS** 网关和拨号计划）

某些组件使用配置文件目录。

文件结构

```
hosts/Customer/
└── group_vars/
    ├── gateways_prod/                # SIP 网关配置
    │   ├── gateway_carrier1.xml
    │   ├── gateway_carrier2.xml
    │   └── gateway_emergency.xml
    ├── gateways_lab/                 # 实验室网关
    │   ├── gateway_test.xml
    ├── dialplan/                     # 呼叫路由规则（默认）
    │   ├── mo_dialplan.xml           # 移动发起（外拨）
    │   ├── mt_dialplan.xml           # 移动终止（来电）
    │   └── emergency.xml
    └── dialplan_lab/                 # 实验室拨号计划
        └── mo_dialplan.xml
```

在主机文件中引用

```yaml
applicationserver:
  hosts:
    customer-tas01:
      ansible_host: 10.10.3.60
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      gateways_folder: "gateways_prod"    # 引用您在此主机上使用的网关文件夹
      dialplan_folder: "dialplan"          # 可选 - 如果未设置，则默认为
"dialplan"
```

发生了什么：

1. Ansible 找到 `gateways_folder: "gateways_prod"`
2. 从 `hosts/Customer/group_vars/gateways_prod/` 复制所有文件到 `/etc/freeswitch/sip_profiles/`
3. 从 `hosts/Customer/group_vars/dialplan/`（或 `dialplan_folder` 指定的文件夹）复制所有文件到 OmniTAS 模板目录
4. 服务加载配置

不同环境：每个环境使用不同的文件夹：

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

配置详情：请参见 https://docs.omnitouch.com.au/docs/repos/OmniCall

示例 3：自定义配置模板（OmniHSS）

家庭用户服务器接受自定义运行时配置模板。

文件结构

```
hosts/Customer/
└── group_vars/
    └── hss_runtime.exs.j2          # 您的自定义 HSS 配置模板
```

在主机文件中引用

```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    # 在 group_vars 中引用您的模板文件名
```

发生了什么：

1. Ansible 找到 `hss_template_config: hss_runtime.exs.j2`
2. 在 `hosts/Customer/group_vars/hss_runtime.exs.j2` 中查找
3. 使用 Jinja2 进行模板化（可以使用 `{{ inventory_hostname }}`、`{{ plmn_id.mcc }}` 等）

4. □□□ `/etc/omnihss/runtime.exs`
5. □□□□

□□ `hss_template_config` □□□□□□□□□□□□□

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

□□ 4□□□□□□□□□OmniMME□

□□□□□□□□□□□□□□□□□□□□□

□□□□

```
hosts/Customer/
└── group_vars/
    └── mme_runtime.exs.j2        # □□□□□ MME ���□□□
```

□□□□□□□□

```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   # □ group_vars □□□
□□□□□□□□
```

□□□□□□

1. Ansible □□ `mme_template_config: mme_runtime.exs.j2`
2. □ `hosts/Customer/group_vars/mme_runtime.exs.j2` □□□
3. □□ Jinja2 □□□□□□□□□□ `{{ inventory_hostname }}`□`{{ plmn_id.mcc }}` □□
4. □□□ `/etc/omnimme/runtime.exs`
5. □□□□

没有 `mme_template_config` 时，使用角色的默认模板。

配置详情：请参见 https://docs.omnitouch.com.au/docs/repos/OmniCore

示例 **5**：必需的配置文件（**OmniCRM**）

OmniCRM 仅从 group_vars 中获取 - 没有角色默认值可供回退。API 配置故意不嵌入角色中，因此缺少文件会导致运行失败，而不是静默地传送另一个站点的机密。

首选部署模式：**apt** 包

使用 `omnicrm_deploy_mode: package`（角色默认），Ansible 从 `apt_repo` 安装通用的 `omnicrm-api` 和 `omnicrm-ui` 包，在 `/etc/omnicrm/` 下模板化配置，并通过 systemctl 重启 `omnicrm-api`。UI 不会在主机上重建；品牌来自 `crm_config.yaml` 中的 `ui:` 块（通过 `GET /crm/public/ui-config` 提供）。

文件结构

```
hosts/Customer/
└── group_vars/
    ├── crm_config.yaml          # API + ui: 配置（必需）
    ├── deployment_assets/       # 可选的徽标/图标 →
/etc/omnicrm/deployment_assets
    │   └── logos/
    ├── customer_crm.env         # 可选；仅用于 omnicrm_deploy_mode:
source 或本地 UI 开发
    ├── cellSites.txt            # 可选的基站□据
    ├── constants.js             # 可选的单元广播消息模板
    └── site_data.json           # 可选的单元广播地理边界
```

在主机文件中引用

```
omnicrm:
  hosts:
    customer-crm01:
      ansible_host: 10.10.3.70
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      omnicrm_deploy_mode: package           # 默认；使用 source 进行传统
git+yarn
      # omnicrm_package_version: "20260810...."  # 可选的 apt pin
      logo_light: logo-light.png             # 复制到
deployment_assets/logos/
      logo_dark: logo-dark.png
      login_background: login-bg.jpg         # →
deployment_assets/login/auth-one-bg.jpg (运行时)
      login_splash: splash.jpg               # →
deployment_assets/login/login-side.jpg (保持源扩展名)
      ssl_fqdn: portal.example.com
      # crm_env_override: customer_crm.env   # 仅在源模式下需要
      cell_sites: cellSites.txt              # 可选
      message_templates: constants.js        # 可选
      site_data: site_data.json              # 可选
```

发生了什么（包模式）：

1. Apt 安装 `omnicrm-api` 和 `omnicrm-ui`。
2. 将 `{{ group_vars_omnicore }}/crm_config.yaml` 模板化到 `/etc/omnicrm/crm_config.yaml`（必需）。
3. 同步 `deployment_assets/`（以及可选的 `logo_*` / `login_*`）到 `/etc/omnicrm/deployment_assets/`。
4. 重启 `omnicrm-api`；nginx 从 `/usr/share/omnicrm-ui` 提供 UI。

登录闪屏/认证背景通过 `GET /crm/public/assets/login/…` 在运行时提供（请参见 `crm_config.yaml` 中的 `ui.branding.login_splash` / `login_background`）。它们不会嵌入 UI `.deb`。复制保持源扩展名（`splash.jpg` → `login/login-side.jpg`）；API 然后提供任何存在的 `login-side.*` 文件。

发生了什么（源模式，传统）：

1. 将 `crm_config.yaml` 模板化到 `/etc/omnicrm/OmniCRM-API/`。

2. 将 `{{ crm_env_override }}` 模板化到 UI `.env` （在源模式下必需）。
3. 使用 yarn 在主机上构建 UI。

重要： `crm_config.yaml` 必须存在。对于包模式，包含一个 `ui:` 块并设置 `ui.assets_dir: '/etc/omnicrm/deployment_assets'`。

配置详情：请参见 https://docs.omnitouch.com.au/docs/repos/OmniCore

示例 6：配置数据目录（OmniHSS）

OmniHSS 配置服务（`services/provision_omnihss.yml`）从 group_vars 中专用的 `omnihss/` 目录读取用户和配置文件数据。

文件结构

```
hosts/Customer/
└── group_vars/
    └── omnihss/
        ├── subscribers.csv              # 要配置的用户
        ├── 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
```

发生了什么：

1. 服务读取 `{{ group_vars_omnicore }}/omnihss/subscribers.csv` 以获取要配置的用户列表。
2. 它从同一 `omnihss/` 目录读取附带的 JSON 文件（`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`□□□□□□□□□□□□ APN、QoS���IMS、EPC□□□□□□□□□ PCRF □□□

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

□□ 7：Grafana SMTP □□□□□□

□□□□□ `grafana.ini` □□□ group_vars □□□ Grafana □□□□□□□SMTP□□□□□□□□□□□□□□□□□□ - □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□

```
monitoring:
  hosts:
    customer-oam01:
      ansible_host: 10.10.3.135
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      grafana_smtp_user: "smtp-user"          # □□ - □□□□□
      grafana_smtp_password: "smtp-password"  # □□ - □□□□□
      grafana_smtp_host: "in-v3.mailjet.com:587"  # □□ - □□□ in-
v3.mailjet.com:587
```

□□□□□□

1. `grafana_smtp_user` □ `grafana_smtp_password` □□□□□ `/etc/grafana/grafana.ini` □ `[smtp]` □□□□□□□□□□□ - □□□□□
2. `grafana_smtp_host` □□□□□□□□□□□□□□□ `in-v3.mailjet.com:587`□

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

□□ 8□UDM SUCI □□□ HNET □□

OmniUDM □□ SUCI □□□TS 33.501 §6.12□□□□ UE □□□□□□□□□□ SUCI □□□ SUPI□□□□ □□ USIM □□□□□□□□□ □□ □□□□□□□□ □□□□□□□□□□□ group_vars □□□□ `hnet_udm/` □□□□□□□ `5gc_nf` □□□□□□ UDM □□□□ `/etc/omniudm/hnet/`□

□□□□

```
hosts/Customer/
└── group_vars/
    └── hnet_udm/
        ├── curve25519-1.key       # X25519 □□□PEM□□□□□□□□ ID 1□
        ├── curve25519-1.pub       # □□ 32 □□□□□□□□□□□□ - □□ USIM □□
        ├── secp256r1-2.key        # EC□prime256v1□□□□PEM□HNET □□
ID 2□
        └── ...
```

□□□□□□ `<curve>-<id>.key`□□□ `<id>` □ UDM □ SUCI □□□□□□□□□□□□□□□`<curve>` □ `curve25519`□□□ A□X25519□□ `secp256r1`□□□ B□prime256v1□□□□□□□□□□□ HNET □ □ ID □□□□□□□□□□□□□□□ USIM □□ ID □□□

□□□□□□□

1. □□□□ `{{ group_vars_omnicore }}/hnet_udm/` □ Ansible □□□□□□□□□□ `become: false` □□□□□□□□□□□□□□□

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

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

真实世界目录结构示例

```
hosts/Customer/
├── host_files/
│   └── production.yml          # 主机文件引用 group_vars 文件
└── group_vars/
    ├── smsc_controller.exs     # OmniMessage 自定义模板
    ├── smsc_smpp.exs           # OmniMessage SMPP 自定义模板
    ├── tas_runtime.exs.j2      # TAS 自定义模板
    ├── hss_runtime.exs.j2      # HSS 自定义模板
    ├── mme_runtime.exs.j2      # MME 自定义模板
    ├── dra_runtime.exs.j2      # DRA 自定义模板
    ├── pgwc_runtime.exs.j2     # PGW 自定义模板
    ├── dea_runtime.exs.j2      # DEA 自定义模板
    ├── upf_config.yaml         # UPF 配置
    ├── crm_config.yaml         # CRM 配置
    ├── stp.j2                  # SS7 STP 模板
    ├── hlr.j2                  # SS7 HLR 模板
    ├── camel.j2                # SS7 CAMEL 模板
    ├── ipsmgw.j2               # IP-SM-GW 模板
    ├── omnicore_smsc_ims.yaml.j2  # SMSC IMS 配置
    ├── pytap.yaml              # TAP3 配置
    ├── sip_profiles/           # SIP 网关（文件夹）
    │   └── gateway_otw.xml
    ├── dialplan/               # 呼叫路由规则（文件夹）
    │   ├── mo_dialplan.xml     # 移动发起
    │   ├── mt_dialplan.xml     # 移动终止
    │   └── mo_emergency.xml    # 紧急路由
    ├── omnihss/                # OmniHSS 配置□据（文件夹）
    │   ├── subscribers.csv         # 要配置的用户
    │   ├── 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/               # UDM SUCI 解密密钥（文件夹）
```

```
        ├── curve25519-1.key          # X25519 □□□HNET □□ ID 1□
        └── curve25519-1.pub          # □□□□□□□□□□□□□□ USIM □□
```

引用 **group_vars** 的常见参数

参数	组件	引用
`smsc_template_config`	omnimessage	Jinja2 模板文件（例如，`smsc_controller.exs[`
`smsc_smpp_template_config`	omnimessage_smpp	Jinja2 模板文件（例如，`smsc_smpp.exs`）
`gateways_folder`	applicationserver	文件夹名称（例如，`gateways_prod`）
`dialplan_folder`	applicationserver	文件夹名称（例如，`dialplan`） - 如果未设置，则默认为 `dialplan`
`tas_runtime_template`	applicationserver	Jinja2 模板文件（例如，`tas_runtime.exs.j2`） - 如果未设置，则默认为 `tas_runtime.exs.j2`
`hss_template_config`	omnihss	Jinja2 模板文件（例如，`hss_runtime.exs.j2`）
`mme_template_config`	omnimme	Jinja2 模板文件（例如，`mme_runtime.exs.j2`）
`dra_template_config`	dra	Jinja2 模板文件（例如，`dra_runtime.exs.j2`）
`pgwc_template_config`	pgwc	Jinja2 模板文件（例如，`pgwc_runtime.exs.j2[`

参数	组件	引用
`frr_template_config`	omniupf	Jinja2 模板文件（例如，`frr.conf.j2`）
SS7 模板	ss7（各种角色）	Jinja2 模板文件（例如，`stp.j2`、`hlr.j2`、`camel.j2`）
`crm_config.yaml`	omnicrm	API 配置文件 - 必需，仅从 `{{ group_vars_omnicore }}/crm_config.yaml` 获取（没有角色默认）
`crm_env_override`	omnicrm	UI .env 文件名（例如，`customer_crm.env`）- 需，没有角色默认
`cell_sites`	omnicrm	可选的基站数据文件（例如，`cellSites.txt`）
`message_templates`	omnicrm	可选的单元广播消息模板（例如，`constants.js`）
`site_data`	omnicrm	可选的单元广播地理边界（例如，`site_data.json`）
`grafana_smtp_user`	monitoring	Grafana SMTP 用户 - 必需，没有角色默认
`grafana_smtp_password`	monitoring	Grafana SMTP 密码 - 必需，没有角色默认
`grafana_smtp_host`	monitoring	可选的 Grafana SMTP 主机 - 默认为 `in-v3.mailjet.com:587`

参数	组件	引用
OmniHSS 配置	omnihss	`omnihss/` 目录，包含 `subscribers.csv` + 配 文件 JSON
HNET 密钥	5gc_nf（omniudm）	`hnet_udm/` 目录，包含用于 SUCI 解密的 `<curve>-<id>.key` 私钥 - 如果为空，则自 动生成默认的 `curve25519-1.key`
配置 YAML	各种组件	直接配置文件（例如，`upf_config.yaml`、`crm_config.yaml`）

关键点

1. **group_vars** 保存自定义 - 覆盖默认配置
2. 按名称引用 - 使用参数如 `smsc_template_config` 或 `gateways_folder`
3. 模板支持 **Jinja2** - 使用 `{{ variable_name }}` 访问任何 Ansible 变量
4. 文件夹部署所有内容 - 所有引用文件夹中的文件都会被复制
5. 版本控制所有内容 - 将所有 group_vars 提交到 Git

何时使用 **group_vars**

□ □□ **group_vars** 进行:

- 自定义组件配置模板
- SIP 网关定义
- 呼叫路由拨号计划
- Diameter 路由规则

- □□□□□□□□□□□□□□

□ □□□□ **group_vars** □□□

- □□□□□□□□IP□□□□□□ - □□□□□□□
- □□□□□□ - □□□□□□□□□□□□□□
- □□□□ - □□□□□□□□□□□□□□□□□□□□ group_vars

□□□□

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

主机文件配置

概述

主机文件（也称为清单文件）是定义整个蜂窝网络部署的核心配置文档。它指定：

- 部署哪些网络功能
- 它们运行的位置（IP 地址、网络段）
- 如何配置它们（特定于服务的参数）
- 客户特定的设置（PLMN、凭据、功能）

文件位置

主机文件位于顶级 `hosts/` 目录下（在 `ansible.cfg` 中设置为 Ansible 的 `inventory`），按客户组织：

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

文件名是任意的。每个部署有一个文件，通常以部署或站点命名（例如 `Production.yml`、`Staging.yml`），而不是固定的 `production.yml`/`staging.yml`/`lab.yml` 方案。

示例主机文件结构

这个简化的示例展示了关键部分：

```yaml
# EPC 节点
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'

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

# 许可证服务器
license_server:
  hosts:
    customer-licenseserver:
      ansible_host: 10.10.2.150

# 全局变量
all:
  vars:
    ansible_connection: ssh
    ansible_password: password
```

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

□□□□□□

□□□□

□□□□□□□□□□

```
pcscf:
  hosts:
    customer-pcscf01:
      ansible_host: 10.10.1.15      # SSH □□□ IP □□
      gateway: 10.10.1.1             # □□□□
      host_vm_network: "vmbr1"       # □□□□□□□□□□□□□ NIC □□
```

□□□ □□ IP □□□□□□□□□□□□□□□□□□□ IP □□□□□□□□□□□□ OmniCore □□□□□□□□□□□□

Proxmox □□□ `host_vm_network` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Proxmox VM/LXC □□□

VM □□□□

□□□□□□□□□□□□□

```
num_cpus: 4                     # CPU □□
memory_mb: 8192                 # □□□□□□□□ RAM
proxmoxLxcDiskSizeGb: 50       # □□□□□□ GB □□□□
```

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

MME:

```
mme_code: 1                    # MME 标识符（1-255）
mme_gid: 1                     # MME 组 ID
network_name_short: Customer   # 网络名称（在手机上显示）
network_name_long: Customer Network
tac_list: [600, 601, 602]     # 跟踪区域代码
```

PGW:

```
ip_pools:                      # 订阅者的 IP 池
  - '100.64.16.0/24'
  - '100.64.17.0/24'
combined_CP_UP: false          # 分离控制/用户平面
```

有关每个变量控制内容的详细说明，请参见：配置参考

应用服务器：

```
online_charging_enabled: true  # 启用 OCS 集成
tas_branch: "main"             # 要部署的软件分支
gateways_folder: "gateways_prod"  # SIP 网关配置
```

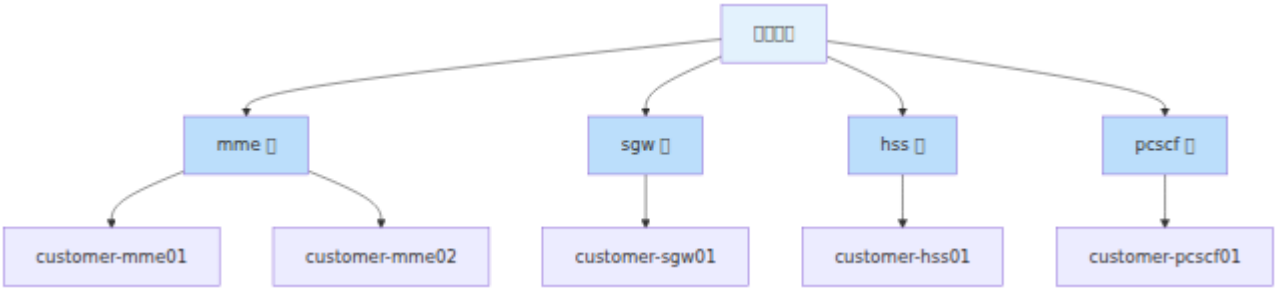
全局变量部分

`all:vars` 部分包含适用于整个部署的设置：

```yaml
all:
  vars:
    # 身份验证
    ansible_connection: ssh
    ansible_password: password
    ansible_become_password: password

    # 客户身份
    customer_name_short: customer
    customer_legal_name: "Customer Inc."
    site_name: "Chicago DC1"
    region: US

    # PLMN（移动网络）标识符
    plmn_id:
      mcc: '001'                # 移动国家代码
      mnc: '01'                 # 移动网络代码
      mnc_longform: '001'       # 零填充的 MNC

    # 网络名称
    network_name_short: Customer
    network_name_long: Customer Network

    # APT 存储库
    # 注意：如果定义了包含主机的 apt_cache_servers 组，
    # use_apt_cache 默认为 true，apt_repo.apt_server
    # 默认自动设置为第一个缓存服务器的 IP
    apt_repo:
      apt_server: "10.254.10.223"
      apt_repo_username: "customer"
      apt_repo_password: "secure-password"
    use_apt_cache: false

    # 时区
    TZ: America/Chicago
```

理解主机组

Ansible 将主机组织为与角色对应的组：



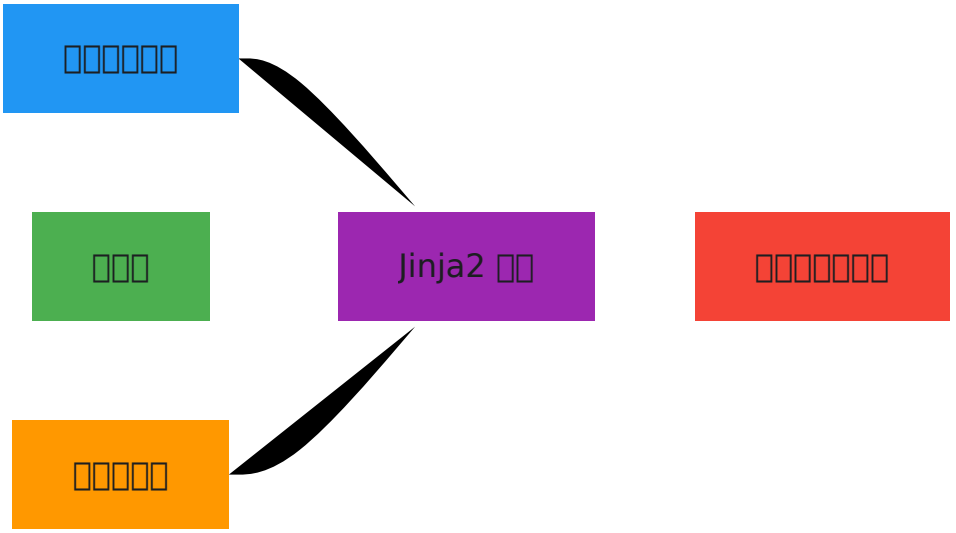
当您运行针对 `mme` 的剧本时，它会应用于 `mme:hosts:` 部分中的所有主机。

上面的图表仅显示了一小部分组。还有许多其他组，包括 `ocs`、`crm`、`smsc`、`omnimessage`、`dra`、`xcap`、`homer`、`cbc`，以及 5GC 组 `amf`、`smf` 和 `upf` 等。

使用 Jinja2 模板进行配置

Ansible 使用 **Jinja2** 模板从主机文件和 group_vars 中定义的变量生成配置文件。

Jinja2 的工作原理



示例模板用法

主机文件定义：

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

Jinja2 模板（在角色中）：

```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
```

生成的配置文件：

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

常见 **Jinja2** 模式

访问嵌套变量：

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

条件逻辑：

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

循环：

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

格式化：

```
# 零填充到 3 位□字
mnc{{ '%03d' | format(plmn_id.mnc|int) }}
```

使用 **group_vars** 重写变量

虽然主机文件定义了基础设施和特定于主机的设置，但 `group_vars` 可以覆盖主机组的默认值。

请参见：组变量配置

完整示例主机文件

这个更完整的示例隐藏了敏感数据：

```yaml
# EPC □□
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 □□
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"

# □□□□
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"

# □□□□
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

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

    # □□□□
    network_name_short: Customer
    network_name_long: Customer Network
    tac_list: [600, 601]

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

    # □□□□
    charging:
```

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

    # 防火墙规则
    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'

    # 虚拟化管理程序配置（Proxmox VM 示例）
    proxmoxServers:
      customer-prxmx01:
        # deployment_type 被省略 → 默认为 "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        # 可选的 cloud-init 用户
名；默认为第一个 local_users 键
        proxmoxTemplatePassword: omnitouch    # 可选的 cloud-init 密码；
默认为 cloud-init 用户名

    # 对于 LXC 站点，设置 deployment_type: lxc 并
    # 在每个 proxmoxServers 条目中设置 proxmoxLxcOsTemplate。
```

请参见 Proxmox VM/LXC 部署 获取完整的 Proxmox 设置和配置详细信息。

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

OmniCore □□□

- **OmniCore** □□□ https://docs.omnitouch.com.au/docs/repos/OmniCore
- **OmniHSS** - □□□□□□□□
- **OmniSGW** - □□□□□□□□□□□
- **OmniPGW** - □□□□□□□□□□□□
- **OmniUPF** - ���□□□□□
- **OmniDRA** - Diameter □□□□
- **OmniTWAG** - □□ WLAN □□□□

OmniCall □□□

- **OmniCall** □□□ https://docs.omnitouch.com.au/docs/repos/OmniCall
- **OmniTAS** - IMS □□□□□□VoLTE/VoNR□
- **OmniCall CSCF** - □□□□□□□□□
- **OmniMessage** - □□□□
- **OmniMessage SMPP** - SMPP □□□□
- **OmniSS7** - SS7 □□□
- **VisualVoicemail** - □□□□

OmniCharge/OmniCRM□

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

□□□□

- Ansible □□□□ - □□□□□□□
- □□□□ - □□□□□□□□□□□□
- □□□□□□ - □□□□□□□
- IP □□□□ - □□□□□ **IP** □□□□
- Netplan □□ - □□ **IP** □□□□□□□

- APT □□□□ - □□□□□
- □□□□□□ - □□□□□
- □□□□□□ - □□□□□□

□□□

1. □□□□□□□□□□□□□□
2. □□□□ PLMN □□□□□
3. □□ APT □□□□□
4. □□□□□□□□
5. □□□□□□ group_vars □□□□□
6. □□ Ansible □□□□□□

许可证服务器

概述

许可证服务器管理所有 Omnitouch 组件的功能激活。每个组件在启动时和操作期间定期验证其许可证。

设置

1. 在主机文件中定义

```
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: "客户名称"
    license_server_api_urls: ["https://10.10.2.150:8443/api"]
```

> 注意：`license_enforced` 是 **HSS** 范围，而不是全局强制切换。它仅在 `omnihss` 角色中定义（`roles/omnihss/defaults/main.yml`，默认值为 `false`），并且目前仅由 HLD 生成器读取；没有其他角色使用它。如果需要，请在 omnihss 组变量下设置它。

2. 提供许可证文件

将 `license.json`（由 Omnitouch 提供）放置在 `hosts/Customer/group_vars/` 中

3. 部署

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

真实的库存目录以大写字母开头并带有前缀（例如 `hosts/Omnicore_Customer/`），主机文件以站点命名（例如 `hosts/Omnicore_Customer/host_files/Production.yml`）。

要检查所有许可证的状态，请浏览到 https://license_server 。

移除许可证

要移除许可证，请使用 `util_playbooks/delete_license.yml`。

网络要求

防火墙配置

客户端站点的防火墙必须配置为允许 HTTPS（端口 443）流量到 Omnitouch 许可证验证服务器：

主机名	目的
1.time.omnitouch.com.au	许可证验证服务器 1
2.time.omnitouch.com.au	许可证验证服务器 2
3.time.omnitouch.com.au	许可证验证服务器 3

所需的出站规则：

- 协议：HTTPS (TCP/443)
- 目标：上述主机名，通过 DNS 解析
- 方向：出站

> 通过 **DNS** 解析：不要硬编码 **IP**。始终通过 FQDN（或防火墙动态解析的地址）允许这些主机。底层 IP 可能会在没有通知的情况下更改。静态 IP 允许列表不受支持，并最终会破坏许可证验证。

DNS 要求

许可证服务器需要功能正常的 DNS 解析以与 Omnitouch 许可证验证基础设施进行通信。

所需的 **DNS** 配置：

- 许可证服务器必须能够访问公共 DNS 服务器
- 配置 DNS 使用以下之一：
  - 1.1.1.1（Cloudflare - 支持安全 DNS）
  - 8.8.8.8（Google 公共 DNS）
- 不要为许可证服务器使用内部/公司 DNS 服务器

注意： Omnitouch 许可证服务器使用安全 DNS（DoH/DoT）。使用公共 DNS 服务器可确保正确的 DNSSEC 验证，并防止安全设备对 DNS 的拦截问题。

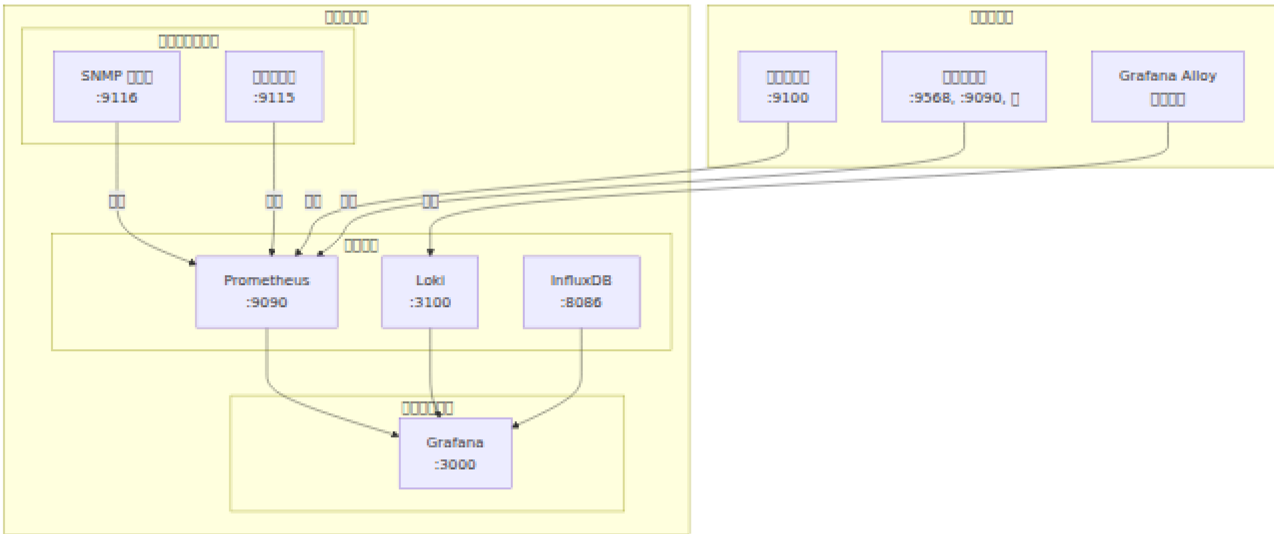
相关文档

- 配置参考
- 主机文件配置

[illegible]

[illegible]

OmniCore [illegible]



SNMP
:9116
:9115
:9100
:9568, :9090,
Grafana Alloy
Prometheus
:9090
Loki
:3100
InfluxDB
:8086
Grafana
:3000

组件

组件	目的	端口	数据保留
Prometheus	时间序列指标存储	9090	30 天或 5GB
Loki	日志聚合	3100	7 天或 50GB
InfluxDB	RAN 指标（诺基亚）	8086	30 天
Grafana	可视化和警报	3000	不适用
节点导出器	主机指标	9100	不适用
SNMP 导出器	网络设备指标	9116	不适用
黑盒导出器	ICMP/HTTP 探测	9115	不适用

数据源

Prometheus

Prometheus 每 1 分钟抓取所有 OmniCore 组件的指标。

数据源 **UID**: `omnicore_prometheus`

抓取目标

作业名称	库存组	端口	指标
`节点导出器`	`all`	9100	主机 CPU、内存、磁盘、网络
`MMEs`	`mme`	9568	会话计数、附加/分离速率
`HSS`	`hss`	9568	认证请求、订阅者查找
`SGW-C`	`sgw`	9568	承载计数、GTP-C 消息
`PGW-C`	`pgwc`	9090	PDN 连接、IP 分配
`UPF 独立`	`upf`	9090	数据包计数、吞吐量
`CSCF`	`pcscf`, `scscf`, `icscf`	9090	注册、事务
`应用服务器 FreeSWITCH`	`applicationserver`	9090	呼叫计数、通道使用
`DRA`	`dra`	9568	路由决策、对等状态
`OmniMessage`	`omnimessage`	9568	短信投递、SMPP 会话
`OmniSS7`	`omniss7`	8080	SS7/SIGTRAN 网关指标
`KeyDB`	`ocs`	9121	内存使用、操作/秒
`CGrateS`	`ocs`	2080	计费、收费、CDR 统计

基础设施导出器

导出器	目标	配置变量
SNMP (MikroTik)	路由器/交换机	`mikrotik.hosts`
SNMP (iDRAC)	戴尔服务器	`idrac.hosts`
SNMP (Synology)	NAS 设备	`synology.hosts`
SNMP (SAF)	微波链路	`saf.hosts`
SNMP (Cisco)	交换机	`cisco.hosts`
黑盒 ICMP	所有主机 + 8.8.8.8	自动
VMware	vCenter 集群	`vcenter_ip`, `vcenter_password`
Proxmox	PVE 集群	`proxmoxServers`

Loki

Loki 从在所有虚拟机上运行的 Grafana Alloy 代理接收日志。

数据源 **UID**: `omnicore_loki`

请参见 集中式日志记录 以获取详细的 Loki/Alloy 配置。

日志标签

□□	□□	□□
`hostname`	□□□□□□□□	`customer-mme01`
`component`	□□□□	`mme`, `cscf`, `ocs`
`unit`	systemd □□□□	`omnimme.service`
`level`	□□□□□	`info`, `error`

InfluxDB

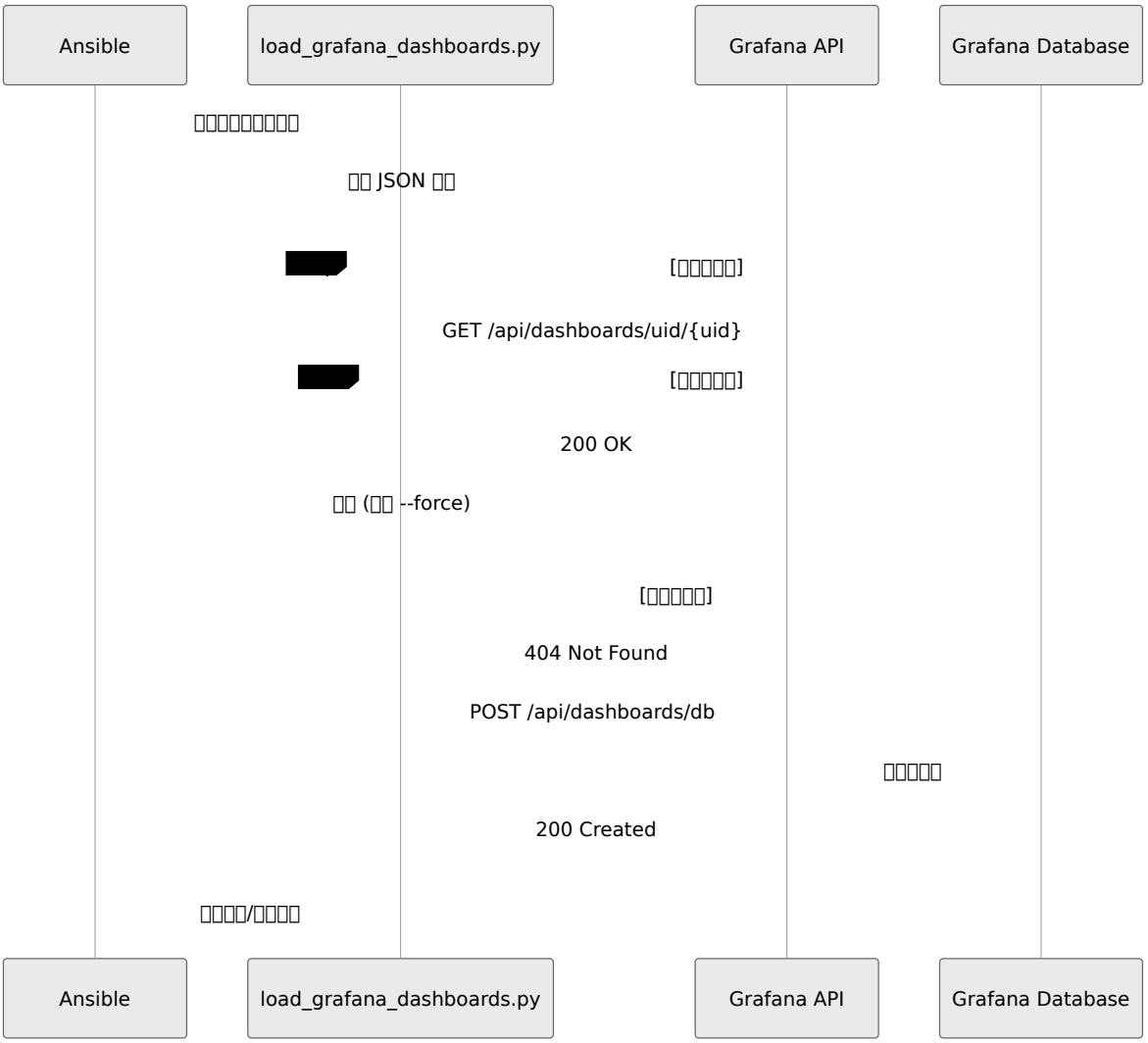
InfluxDB □□□□□□□□

文件夹	内容
BSS	CGrateS、KeyDB、计费仪表板
EPC	MME、SGW、PGW、UPF、HSS、DRA 仪表板
IMS	CSCF、应用服务器、OmniMessage 仪表板
基础设施	节点导出器、SNMP、网络监控
RAN	诺基亚/OmniRAN 性能仪表板
日志	组件特定的日志查看器

仪表板加载（基于 API)

仪表板通过 Grafana API 加载，而不是基于文件的配置。这允许仪表板被:

- 通过 Grafana UI 编辑
- 通过 API 修改
- 在 Ansible 存储库中进行版本控制


Ansible
load_grafana_dashboards.py
Grafana API
Grafana Database
□□□□□□□□□□□□
□□ JSON □□
[□□□□□□]
GET /api/dashboards/uid/{uid}
[□□□□□□]
200 OK
□□ (□□ --force)
[□□□□□□]
404 Not Found
POST /api/dashboards/db
□□□□□□
200 Created
□□□□/□□□□
Ansible
load_grafana_dashboards.py
Grafana API
Grafana Database

□□□□□

□□□□ Ansible □□□□□□□□□□□□□□□□□□□□□□

```
# 从 Ansible 控制器
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

# 强制更新现有仪表板
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
```

仪表板源文件

仪表板 JSON 文件存储在 Ansible 存储库中:

```
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
│   └── ...
├── 基础设施/
│   ├── Node_Exporter_Full.json
│   ├── MikroTik_Dashboard.json
│   └── ...
├── RAN/
│   ├── Nokia_Overview.json
│   ├── Nokia_Detailed.json
│   └── ...
└── 日志/
    ├── CSCF_Logs.json
    ├── MME_Logs.json
    └── ...
```

仪表板备份

从正在运行的 Grafana 实例导出仪表板和警报到存储库以进行版本控制：

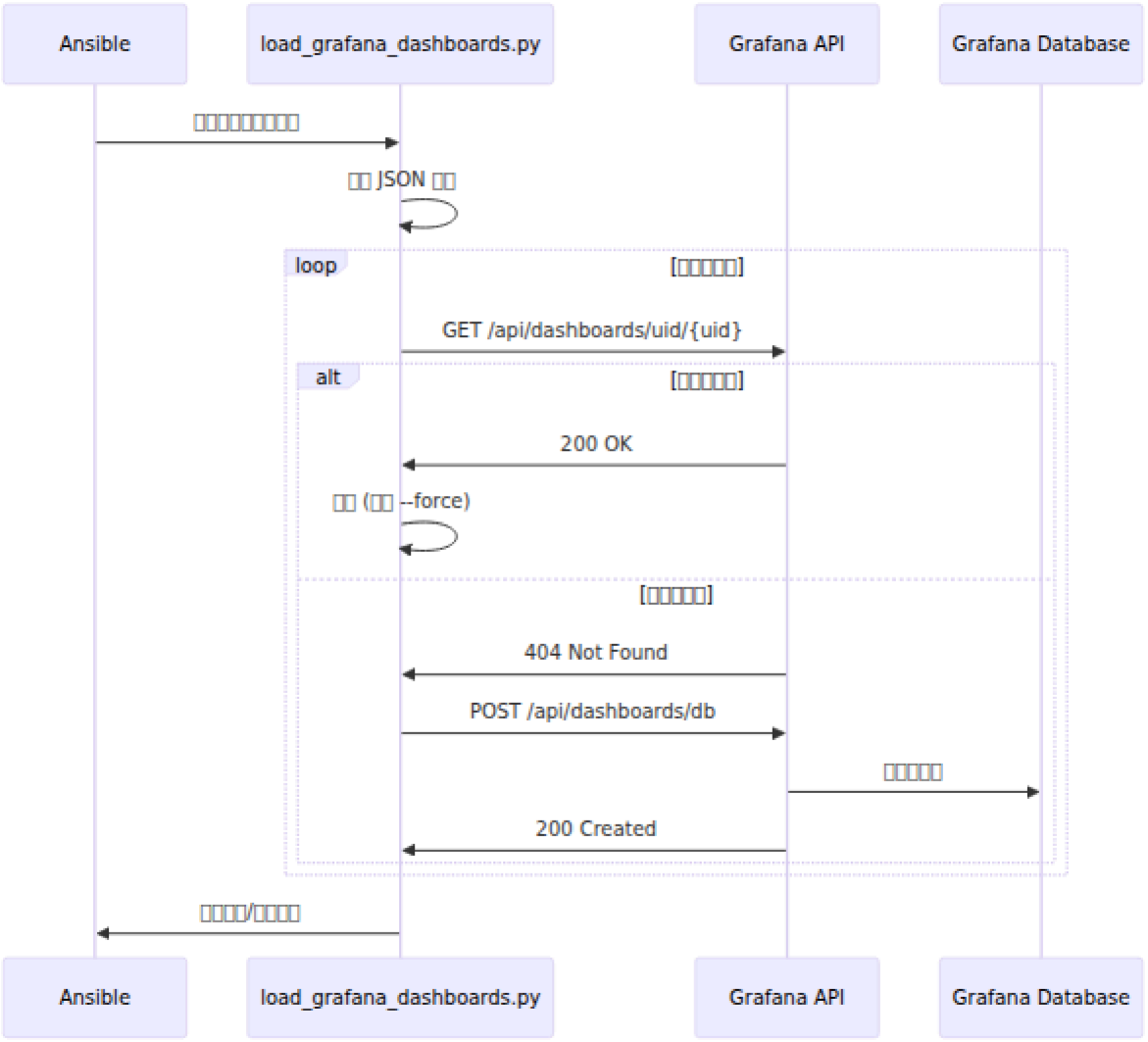
```
# 从 roles/monitoring 目录
python3 export_grafana.py --customer <CUSTOMER>
```

这将导出：

- 所有仪表板到 `roles/monitoring/templates/grafana/dashboards/{folder}/*.json`（跨客户

共享)

- 警报规则到 `hosts/Omnicore_{CUSTOMER}/group_vars/grafana/alerts/all_alert_rules.json`
- 联系点到 `hosts/Omnicore_{CUSTOMER}/group_vars/grafana/alerts/contact_points.json`
- 通知策略到 `hosts/Omnicore_{CUSTOMER}/group_vars/grafana/alerts/notification_policies.json`



导出行为

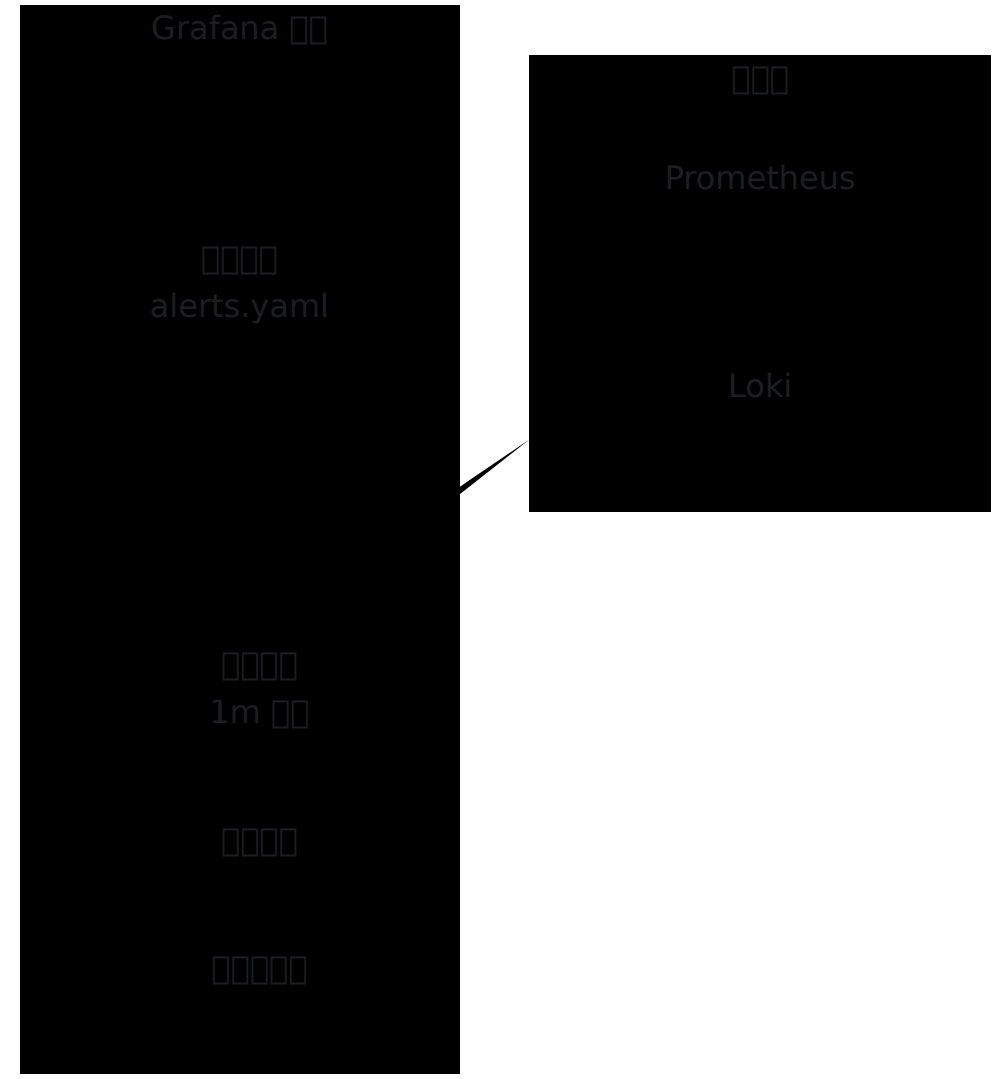
- 仅写入已更改的文件（忽略诸如 `version` 和 `id` 等易变字段）
- 保留与 Grafana 组织匹配的文件夹结构

- 将仪表板标题映射到一致的文件名

自定义仪表板

特定于客户的仪表板可以放置在 `group_vars/grafana/dashboards/` 中，并将与标准仪表板一起加载：

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


Grafana □□
□□□
Prometheus
□□□□
alerts.yaml
Loki
□□□□
1m □□
□□□□
□□□□□
□□□
Slack Webhook
Google Chat
□□□□
Pushover

警报加载

警报通过 Grafana API 从每个客户的导出加载，而不是单个静态 YAML 文件。在部署时，静态警报配置文件（`/etc/grafana/provisioning/alerting/alerts.yaml`）被删除，以便可以通过 UI/API 编辑警报。

主要路径从 `hosts/Omnicore_<CUSTOMER>/group_vars/grafana/alerts/` 加载每个客户的警报规则、联系点和通知策略，使用 `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
```

如果没有特定于客户的警报，则角色会回退到通过 `load_grafana_alerts.py` 加载角色模板 `alerts.yaml`：

```
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
```

默认警报规则

警报	文件夹	条件	持续时间
磁盘空间使用 **90%**	基础设施	根文件系统 > 90%	5 分钟
CPU 超过 **90%**	基础设施	CPU 使用 > 90%	5 分钟
系统负载超过 **90%**	基础设施	平均负载 > 90%	5 分钟
主机宕机	基础设施	Prometheus 目标不可达	5 分钟
MME 订阅者丢失 **40%**	EPC	会话计数下降 40%	30 秒
MME 上 **0** 个订阅者	EPC	无附加订阅者	5 分钟
IMS 订阅者下降 **40%**	IMS	P-CSCF 注册下降 40%	30 秒
平均 **MOS** 低于 **4**	IMS	语音质量下降	5 分钟
eNodeB 计数下降	RAN	连接的 eNB 减少	1 分钟
直径响应延迟高	EPC	P95 延迟峰值	5 分钟

警报规则结构

`alerts.yaml` 中的警报规则:

```yaml
apiVersion: 1
groups:
  - orgId: 1
    name: 警报组
    folder: 基础设施
    interval: 1m
    rules:
      - uid: alert-lowDiskSpace
        title: 主机上的磁盘空间使用 90%
        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="/"})
          # ... 阈值表达式
        noDataState: OK
        execErrState: Error
        for: 5m
        annotations:
          summary: 主机上的磁盘空间超过 90%
        labels:
          type: resource
```

联系点配置

联系点通过库存变量进行配置：

```yaml
# 在 group_vars/all.yml 或特定于客户的变量中
monitoring_data:
  contactPoints:
    - orgId: 1
      name: 默认
      receivers:
        # Slack 集成
        - uid: slack-alerts
          type: slack
          disableResolveMessage: false
          settings:
            url: "https://hooks.slack.com/services/xxx/yyy/zzz"

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

支持的联系点类型包括 Slack、Google Chat、电子邮件和 Pushover（请参见 `roles/monitoring/templates/grafana/contact-points.yaml.j2`）。Pushover 通过 `notifications.pushover` 块进行配置（包含 `enabled`、`api_token` 和 `user_key`）。

通知策略

通知策略将警报路由到适当的联系点：

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

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

□□□□

Prometheus □□

□□□□□□□□□ `/etc/prometheus/prometheus.yml`□

□□	□□	□□
`scrape_interval`	1m	□□□□□□□□
`evaluation_interval`	1m	□□□□□□□□□□
`scrape_timeout`	50s	□□□□□□□□

Grafana □□

□□□□□□□□□ `/etc/grafana/grafana.ini`□

□□□□□□□□□□□□□□

□□	□	□□
`allow_embedding`	true	□□ iframe □□
`provisioning`	`/etc/grafana/provisioning`	□□□□

□□□□□□□□ `grafana.ini` □□□□`admin_password` □□□□□□□□□□□□□□□□□□□□ `grafana_admin_password` □□□□□□□□□□□□□□□□□□□□ API □□□□□□□□□/□□□□□□□□

□□□□ **(SMTP)** □□

□□□□□□□□□ `group_vars` □□□ SMTP □□□□□□□ `roles/monitoring/templates/grafana/grafana.ini`□□

变量	必需	默认	描述
`grafana_smtp_user`	是	*(□)*	SMTP 用户名。无默认值。如果未设置，部署将显著失败，而不是发送秘密。
`grafana_smtp_password`	是	*(□)*	SMTP 密码。无默认值。如果未设置，部署将失败。
`grafana_smtp_host`	否	`in-v3.mailjet.com:587`	SMTP 服务器主机:端口。
`grafana_smtp_enabled`	否	`true`	是否启用 SMTP。

`from_address` 和 `from_name` 来源于 `customer_legal_name`（例如 `alerts+<customer_legal_name>@omnitouch.com.au` 和 `Omnitouch Alerts - <customer_legal_name>`）。

Loki 配置

请参见 集中式日志记录 以获取 Loki 的保留和存储设置。

InfluxDB 配置

位于监控服务器的 `/etc/influxdb/influxdb.conf`。

数据库	保留策略	持续时间
`nokia-monitor`	`autogen`	30 天
`dra`	`autogen`	365 天
`Omnicharge_TAP3`	`autogen`	90 天

服务端口

服务	端口	协议	描述
Grafana	3000	HTTP	仪表板 UI 和 API
Prometheus	9090	HTTP	指标存储和查询
Loki	3100	HTTP	日志摄取和查询
InfluxDB	8086	HTTP	InfluxDB API
节点导出器	9100	HTTP	主机指标
SNMP 导出器	9116	HTTP	SNMP 指标代理
黑盒导出器	9115	HTTP	探测指标
Alloy	12345	HTTP	Alloy UI 和指标

常见操作

查看指标

1. 打开 Grafana，地址为 `http://<monitoring-ip>:3000`
2. 导航到 仪表板 并选择适当的文件夹
3. 使用时间范围选择器调整视图窗口

搜索日志

1. 打开 Grafana，地址为 `http://<monitoring-ip>:3000`
2. 转到 探索 并选择 **Loki** 数据源
3. 使用 LogQL 查询:

```
# 从特定主机的所有日志
{hostname="customer-mme01"}

# 从所有 MME 组件的错误
{component="mme"} |~ "(?i)error"

# 搜索特定 IMSI
{component="hss"} |= "123456789012345"
```

创建自定义警报

推荐的工作流程是在 Grafana UI 中创建/编辑警报，然后将其导出到客户的 `group_vars/grafana/alerts/` 目录（请参见 仪表板备份），以便进行版本控制，并在下次部署时重新应用。

要手动加载特定于客户的警报：

```
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
```

角色模板 `roles/monitoring/templates/grafana/provisioning/alerting/alerts.yaml` 仅在客户没有导出警报时用作后备。

备份仪表板

在 Grafana UI 中进行更改后，导出到存储库：

```
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 "从生产更新警报"
```

`--customer` 参数是必需的。此脚本将警报规则、联系点和通知策略导出到客户的 `group_vars/grafana/alerts/` 目录；它不再导出仪表板，仪表板在角色目录中跨客户共享（`roles/monitoring/templates/grafana/dashboards/`）。

故障排除

Prometheus 目标宕机

症状：仪表板显示 "无数据" 或目标状态显示 DOWN

可能原因：

- 目标主机上的服务未运行
- 防火墙阻止导出器端口
- 主机名解析不正确

解决方案：

1. 检查目标上的服务是否在运行：

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

2. 验证从监控服务器的连接：

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

3. 检查 Prometheus 目标页面：`http://<monitoring-ip>:9090/targets`

仪表板未加载

症状：仪表板显示加载旋转图标或错误

可能原因：

- 数据源连接失败
- 查询超时
- 仪表板 JSON 损坏

解决方案：

1. 在 Grafana 中测试数据源：配置 → 数据源 → 测试
2. 检查 Grafana 日志：`journalctl -u grafana-server -f`
3. 尝试从存储库重新加载仪表板

警报未触发

症状：条件满足但未收到通知

可能原因：

- 警报已暂停
- 联系点配置错误
- 通知策略不匹配

解决方案：

1. 在 Grafana 中检查警报状态：警报 → 警报规则
2. 验证联系点测试是否有效：警报 → 联系点 → 测试
3. 审查通知策略路由

Grafana 无法连接到数据源

症状：Grafana 中显示 "错误网关" 或连接超时

可能原因：

- Prometheus/Loki/InfluxDB □□□□
- □□□□□□□□□□
- □□□□□□□□□□□

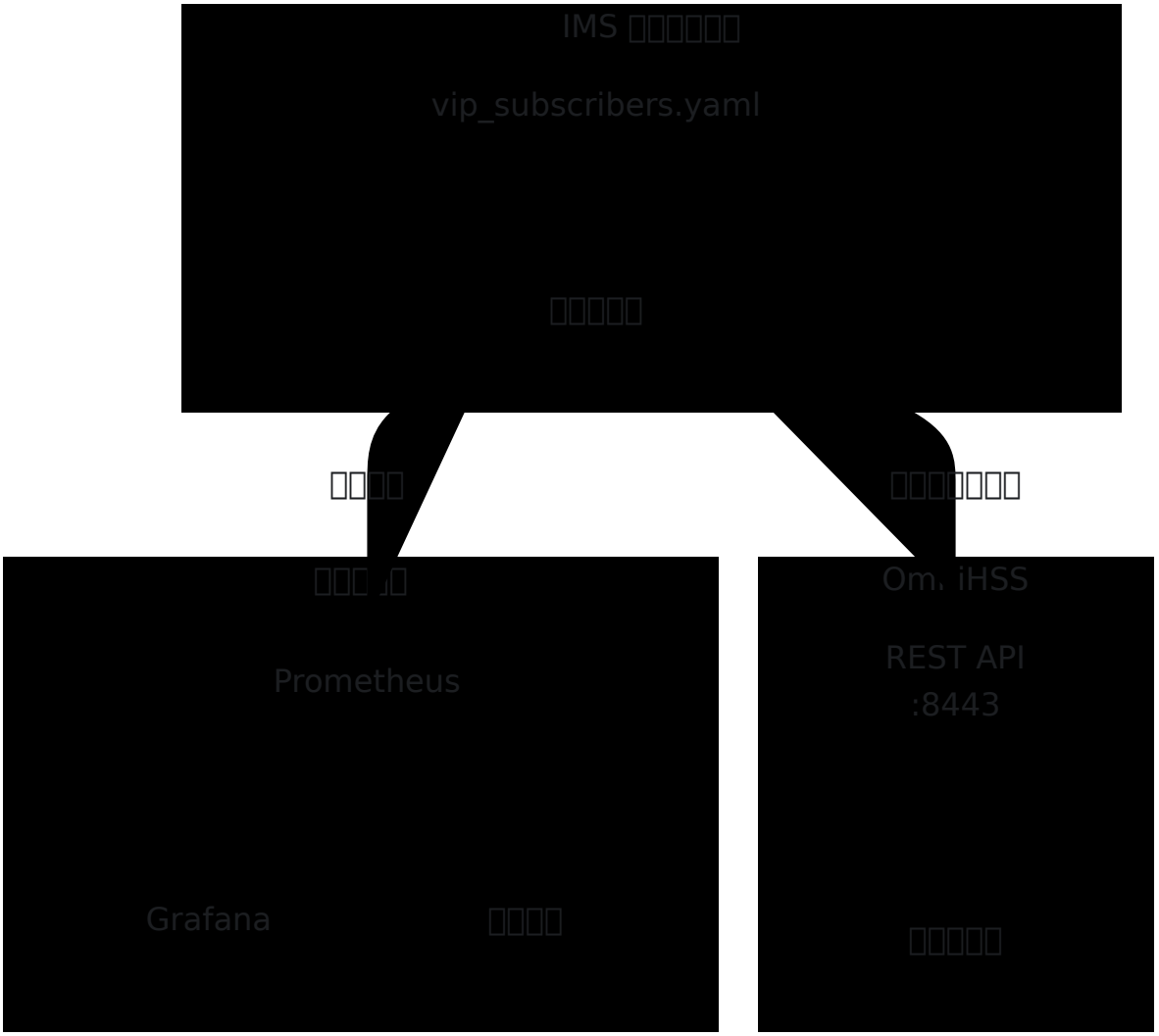
□□□□:

1. □□□□□□□□
   ```
   systemctl status prometheus loki influxdb
   ```
2. □□□□□□□□□□□
   ```
   ss -tlnp | grep -E '9090|3100|8086'
   ```
3. □□□□□□□□
   ```
   curl http://localhost:9090/api/v1/status/config
   curl http://localhost:3100/ready
   ```

VIP □□□□□

VIP □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ IMS □□□EPC □□□ UE □□□□□□□□□□□□□□□□□□□□

架构



服务类型

订阅者按预期服务类型分类，这决定了健康评估：

类型	**IMS** 必需	**EPC** 必需	用例
`full`	是	是	标准移动订阅者，具有语音和数据
`voip_only`	是	否	仅 VoIP 设备（IP 电话、软电话）
`data_only`	否	是	仅数据设备（路由器、物联网、M2M）

健康评估**:**

- **full**: 当 IMS 注册（`assigned_scscf` 不为 null）且 EPC 附加（`last_seen_mme` 不为 null）时健康
- **voip_only**: 当 IMS 注册（`assigned_scscf` 不为 null）时健康
- **data_only**: 当 EPC 附加（`last_seen_mme` 不为 null）时健康

配置

VIP 订阅者在主机组变量中配置：

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

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

  # 监控设置
  scrape_interval_seconds: 30
  ping_timeout_seconds: 2
  ping_count: 2

  # 要监控的订阅者
  subscribers:
    # 全服务订阅者 (IMS + EPC)
    - imsi: "001010000000001"
      label: "紧急服务"
      type: "full"

    - imsi: "001010000000002"
      label: "关键基础设施"
      type: "full"

    # 仅数据服务 (路由器、物联网 - 不期望 IMS)
    - imsi: "001010000000003"
      label: "站点路由器"
      type: "data_only"

    # 仅 VoIP 服务 (期望 IMS，无数据承载)
    - imsi: "001010000000004"
      label: "IP 电话"
      type: "voip_only"
```

配置参数

参数	类型	必需	默认	描述
`hss_url`	字符串	是	-	OmniHSS REST API URL (HTTPS)
`scrape_interval_seconds`	整数	否	30	查询订阅者状态的频率
`ping_timeout_seconds`	整数	否	2	UE IP ping 测试的超时
`ping_count`	整数	否	2	要发送的 ping 数据包数量
`blackbox_exporter_url`	字符串	否	-	用于 ping 测试的远程黑盒导出器 URL (例如 `http://pcscf01:9115`)。使用远程 ICMP 探测而不是本地 ping。
`subscribers`	列表	是	-	要监控的订阅者列表

订阅者参数

参数	类型	必需	默认	描述
`imsi`	字符串	是	-	订阅者 IMSI (15 位数字)
`label`	字符串	否	IMSI	用于显示的人类可读名称
`type`	字符串	否	`full`	服务类型：`full`、`voip_only` 或 `data_only`

注意：MSISDN 会从 HSS API 响应中自动获取，无需配置。

指标

导出器在端口 9550 的 `/metrics` 上暴露指标。

状态指标

指标	类型	描述
`vip_subscriber_service_healthy`	Gauge	如果订阅者满足其服务类型的健康标准则为 1，否则为 0
`vip_subscriber_ims_registered`	Gauge	如果订阅者已分配 S-CSCF，则为 1，否则为 0
`vip_subscriber_epc_registered`	Gauge	如果订阅者已活动 MME 附加，则为 1，否则为 0
`vip_subscriber_ue_ip_reachable`	Gauge	如果 UE IP 对 ping 响应，则为 1，否则为 0
`vip_subscriber_hss_reachable`	Gauge	如果 HSS API 返回有效数据，则为 1，否则为 0
`vip_subscriber_enabled`	Gauge	如果订阅者帐户在 HSS 中启用，则为 1，否则为 0

年龄指标

指标	类型	描述
`vip_subscriber_ims_registration_age_seconds`	Gauge	自上次 IMS 注册以来的秒数 (-1 如果未注册)
`vip_subscriber_epc_registration_age_seconds`	Gauge	自上次 EPC 更新位置以来的秒数 (-1 如果未附加)

信息指标

指标	类型	描述
`vip_subscriber_info`	Gauge	始终为 1，携带所有状态和信息作为标签

所有指标上的标签：

标签	描述	示例
`imsi`	订阅者 IMSI	`001010000000002`
`label`	人类可读名称	`关键基础设施`
`msisdn`	电话号码（来自 HSS）	`61400000002`
`type`	服务类型	`full`、`voip_only`、`data_only`

`vip_subscriber_info` 上的附加标签：

标签	描述	示例
`healthy`	服务健康状态	`1` 或 `0`
`ims`	IMS 注册状态	`1` 或 `0`
`epc`	EPC 附加状态	`1` 或 `0`
`ping`	UE IP 可达性	`1` 或 `0`
`assigned_scscf`	S-CSCF 主机名	`scscf01.ims.example.com`
`last_seen_mme`	MME 主机名	`mme02.epc.example.com`
`ue_ip`	UE 分配的 IP 地址	`100.72.83.20`

示例查询

```
# 健康 VIP 订阅者的计
count(vip_subscriber_service_healthy == 1)

# 不健康 VIP 订阅者的计
count(vip_subscriber_service_healthy == 0)

# VoIP 服务的 IMS 注册状态
vip_subscriber_ims_registered{type=~"voip_only|full"}

# 据服务的 EPC 附加状态
vip_subscriber_epc_registered{type=~"data_only|full"}

# IMS 注册过期的订阅者 (>1 小时)
vip_subscriber_ims_registration_age_seconds > 3600
```

仪表板

IMS VIP 订阅者 仪表板提供对 VIP 订阅者状态的实时可见性。

位置: Grafana → IMS → IMS VIP 订阅者

仪表板 **URL**: `/d/ims-vip-subscribers/`

面板

□□	□□
□□□□	□□□□□□□□□□□□□
□□□□□	□□□□□□□□□□□□□□
IMS □□	□□□□ IMS □□□□□□□□
EPC □□	□□□□ EPC □□□□□□□□
UE □□	□□□ ping □ UE IP □□□□□□
□□□	□□□ VIP □□□□□
VIP □□□□□	□□□□□□□□□□□□□□□
□□□□□□□□□	□□□□□□□□□□□□

□□

□ VIP □□□□□□□□□□□□□□□□□□□□□

VIP □□□□□□□□

□□: `VIP □□□□□□□□` □□□: □□ □□□□: 1 □□ □□: `vip_subscriber_service_healthy == 0`

□□:

- □□: `VIP □□□ {{ $labels.label }} □□□`
- □□: □□□□□□□□□□□□□□
  - VoIP □□: □□□ MSISDN
  - □□□□: □□□ IMSI
  - □□□: □□□MSISDN □ IMSI

□□□□□□**:**

- `IP 电话 (voip_only) 服务已宕机。MSISDN: 61400000001`
- `站点路由器 (data_only) 服务已宕机。IMSI: 001010000000003`
- `关键基础设施 (full) 服务已宕机。MSISDN: 61400000002 IMSI: 001010000000002`

添加新的 VIP 订阅者

1. 编辑配置文件：

```
# hosts/<customer>/group_vars/vip_subscribers.yaml
subscribers:
  - imsi: "123456789012345"
    label: "新 VIP 订阅者"
    type: "full"  # 或 voip_only, data_only
```

2. 部署更新的配置：

```
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. 验证新订阅者是否出现在指标中：

```
curl -s http://<monitoring-server>:9550/metrics | grep "新 VIP 订
阅者"
```

相关文档

- 集中式日志记录：详细的 Loki 和 Alloy 配置
- 部署架构：整体系统架构
- 主机文件配置：库存配置

Netplan 配置

概述

OmniCore 可以使用 netplan 自动配置部署的虚拟机上的网络接口。这对于以下情况非常有用：

- 设置主要管理接口 (eth0)
- 添加用于公共 IP、对等连接或专用流量的次要接口
- 为特定目的地配置静态路由

启用 **Netplan** 配置

要为主机启用自动 netplan 配置，请在 `group_vars` 文件夹中添加指向 Jinja2 模板的 `netplan_config` 变量：

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

模板将从 `hosts/<customer>/group_vars/netplan.yaml.j2` 中获取。

模板参考

没有共享的、规范的 `netplan.yaml.j2`。每个客户在 `hosts/<customer>/group_vars/` 中创建自己的 `netplan.yaml.j2`，这些模板各不相同 - 主要接口名称、DNS 搜索域和次要接口命名因客户而异。在为新客户入驻时，创建一个与该客户的接口布局匹配的客户专用模板，而不是逐字复制其他客户的模板。

以下是一个客户模板的 示例。它是说明性的，而不是规范 - 其他客户可能不同。这里添加了注释以作解释：

```
network:
  version: 2
  ethernets:
    # 主要接口 - 使用来自清单的 ansible_host 和 gateway
    eth0:
      addresses:
      - "{{ ansible_host }}/{{ mask_cidr | default(24) }}"
      nameservers:
        addresses:
{% if 'dns' in group_names %}
        # 如果该主机是 DNS 服务器，则使用外部 DNS 以避免循环依赖
        - 8.8.8.8
{% else %}
        # 否则，使用清单中 'dns' 组的 DNS 服务器
{% for dns_host in groups['dns'] | default([]) %}
        - {{ hostvars[dns_host]['ansible_host'] }}
{% endfor %}
{% endif %}
        # DNS 搜索域 - 客户特定（因客户而异；有些不设置）
        search:
        - customer.example
      routes:
      - to: "default"
        via: "{{ gateway }}"

{% if secondary_ips is defined %}
    # 次要接口 - 遍历来自清单的 secondary_ips 字典
    # 此示例将它们命名为 ens19、ens20、ens21...（18 + loop.index）。这是
    # 特定于模板的；请参见下面的“接口命名”。
{% for nic_name, nic_config in secondary_ips.items() %}
    ens{{ 18 + loop.index }}:
      addresses:
      # 掩码来自每个 NIC 的 mask_cidr（默认为 24）
      - "{{ nic_config.ip_address }}/{{ nic_config.mask_cidr | default(24) }}"
{% if nic_config.routes is defined %}
      # 此接口的静态路由 - 每个路由使用该接口的网关
      routes:
{% for route in nic_config.routes %}
      - to: "{{ route }}"
        via: "{{ nic_config.gateway }}"
{% endfor %}
{% endif %}
```

```
{% endfor %}
{% endif %}
```

关键点:

- `ansible_host` 和 `gateway` 来自主机的清单条目
- DNS 服务器动态从 `dns` 组中的主机中提取
- DNS `search` 域是客户特定的（因客户而异；有些客户省略它）
- 次要接口命名因客户而异（有些使用 `ens19`、`ens20`，…；见下文）
- 每个次要 IP 可以有自己的掩码（`mask_cidr`）、网关和静态路由

主要接口配置

主要接口 (eth0) 使用以下方式自动配置:

- `ansible_host` - IP 地址
- `gateway` - 默认网关
- `mask_cidr` - 网络掩码（默认为 24）

DNS 服务器自动设置为:

- `dns` 组中的主机（使用它们的 `ansible_host` IP）
- 如果主机本身是 DNS 服务器，则回退到 `8.8.8.8`

链接 **MTU (`link_mtu`)**

`common` 角色可以通过 `link_mtu` 清单变量独立于 netplan 模板固定主要接口的 链接 **MTU**。设置后，它会在主机的主要接口上渲染一个 netplan drop-in (`/etc/netplan/999-link-mtu.yaml`) 并应用它。这是因为虚拟机来自 VMware/Proxmox 模板，具有源端口组提供的任何 MTU（有时是 jumbo/9000），这与其余的网络不匹配，并在 PMTUD 破损时导致超大 SCTP/SIP 被黑洞。

该 drop-in 被故意命名为 `999-`，以便它排序为 最后：netplan 按字典顺序合并所有 `/etc/netplan/*.yaml`，并且每个键的最后文件优先，配置可以留下竞争文件，这些文件在同一接口上设置 `mtu`（一个遗留的 `80-ansible-config.yaml` 和 VMware 引擎的 `99-netcfg-vmware.yaml`）。较低编号的 drop-in 会被这些文件静默覆盖。

```
all:
  vars:
    link_mtu: 1500    # 将每个主机的主要 NIC 固定为 1500
```

将其设置在 `all.vars` 中以应用于整个站点，或按主机覆盖。如果未定义 `link_mtu`，该角色将保持接口 MTU 不变。

> **`link_mtu`** 不是 **`mtu`**。顶层 `mtu` 变量是 GTP/隧道有效载荷 MTU（例如 1400/1430/1300），由 packet-core 角色使用，而不是 链接 MTU。将物理 NIC 设置为这些值是错误的。始终使用专用的 `link_mtu` 作为接口 MTU。

次要接口

对于需要额外网络接口（公共 IP、对等等）的主机，请使用 `secondary_ips` 配置。

模式

```
secondary_ips:
  <logical_name>:
    ip_address: <ip_address>
    mask_cidr: <prefix_length>   # 可选 - 每个 NIC 的掩码，默认为 24
    gateway: <gateway_ip>
    host_vm_network: <proxmox_bridge>
    vlanid: <vlan_id>
    nic_name: <interface_name>   # 可选 - 显式接口名称覆盖
                                 # （由支持该功能的模板支持）
    routes:                      # 可选 - 通过此接口的静态路由
      - '<destination_cidr>'
      - '<destination_cidr>'
    table_routes:                # 可选 - 高级，特定于模板
      - { to: '<cidr>', via: '<gateway>', table: <table_id> }
    routing_policy:              # 可选 - 高级，特定于模板
      from: '<source_cidr>'
      via: '<gateway>'
      table: <table_id>
      priority: <priority>       # 可选，默认为 100
```

> `mask_cidr` 键设置该一个次要接口的前缀长度。它是从次要 IP 条目内部读取的（`nic_config.mask_cidr`），而不是从顶层 `mask_cidr`。如果省略，默认为 `/24`。

> `table_routes` 또 `routing_policy` □□□□/□□□

示例配置

```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'
```

生成的 **Netplan** 输出

使用示例模板呈现，上述配置生成（DNS `search` 域和 `ens19`/`ens20` 命名是客户特定的）：

```
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"
```

`public_ip` 接口呈现为 `/28`，因为其条目设置了 `mask_cidr: 28`，而 `peering_ip` 则回退到 `/24` 默认值。

Proxmox 集成

使用 `proxmox.yml` 剧本时，次要 NIC 会自动在虚拟机上创建：

1. 新虚拟机：在初始配置期间添加次要 NIC
2. 现有虚拟机：添加次要 NIC，并重启虚拟机以应用更改

Proxmox 配置使用：

- `host_vm_network` - 要将 NIC 附加到的桥接
- `vlanid` - 接口的 VLAN 标签

工作原理

1. `/etc/netplan` 中的现有 netplan 配置被备份 (`*.bak`)，然后删除以防止冲突
2. 客户的 Jinja2 模板渲染到 `/etc/netplan/01-netcfg.yaml`（模式 `0600`）
3. `netplan try --timeout 30` 应用配置，如果连接丢失则自动回滚
4. `wait_for_connection` 确认主机保持可达
5. `netplan apply` 提交配置

常见用例

具有公共 IP 的 Diameter Edge Agent (DEA)

```yaml
<hostname>:
  ansible_host: 10.0.1.100          # 内部管理 IP
  gateway: 10.0.1.1
  netplan_config: netplan.yaml.j2
  secondary_ips:
    diameter_roaming:
      ip_address: 192.0.2.50        # 用于漫游合作伙伴的公共 IP
      gateway: 192.0.2.1
      host_vm_network: "vmbr0"
      vlanid: "200"
      routes:
        - '198.51.100.0/24'         # 漫游合作伙伴网络
```

具有 S5/S8 接口的 PGW

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

具有独立管理和数据网络的多宿主服务器

```
<hostname>:
  ansible_host: 10.0.1.100             # 管理网络
  gateway: 10.0.1.1
  netplan_config: netplan.yaml.j2
  secondary_ips:
    data_network:
      ip_address: 10.0.2.100           # 据网络
      gateway: 10.0.2.1
      host_vm_network: "ovsbr2"
      vlanid: "200"
    backup_network:
      ip_address: 10.0.3.100           # 备份网络
      gateway: 10.0.3.1
      host_vm_network: "ovsbr3"
      vlanid: "300"
```

在模板中引用次要 IP

您可以在其他 Jinja2 模板和配置文件中引用次要 IP 地址。

在同一主机上

当在具有次要 IP 的同一主机上配置服务时，您可以直接引用或使用 `inventory_hostname`：

```
# 直接引用（最简单）
{{ secondary_ips.diameter_public_ip.ip_address }}

# 或通过 inventory_hostname 显式引用（结果相同）
{{ hostvars[inventory_hostname]['secondary_ips']
['diameter_public_ip']['ip_address'] }}

# 访问其他属性
{{ secondary_ips.diameter_public_ip.gateway }}
{{ secondary_ips.diameter_public_ip.vlanid }}
```

从另一主机

当您需要引用 *□□* 主机的次要 IP（例如，配置对等连接）时，请使用 `hostvars` 和目标主机名：

```
# 引用 dra 组中的第一主机
{{ hostvars[groups['dra'][0]]['secondary_ips']
['diameter_public_ip']['ip_address'] }}

# 遍历所有 DRA 主机并获取它们的公共 IP
{% for host in groups['dra'] %}
{% if hostvars[host]['secondary_ips'] is defined %}
  - {{ hostvars[host]['secondary_ips']['diameter_public_ip']
['ip_address'] }}
{% endif %}
{% endfor %}
```

示例：**DRA** 对等配置

配置一个直径对等体以绑定到其自己的公共 IP：

```
# 在 dra_config.yaml.j2 中 - 使用 inventory_hostname 作为当前主机
peers:
  - name: external_peer
    # 绑定到该主机的公共直径 IP
    local_ip: {{ hostvars[inventory_hostname]['secondary_ips']
['diameter_public_ip']['ip_address'] }}
    remote_ip: 198.51.100.50
    port: 3868
```

检查次要 IP 是否存在

在使用变量之前始终检查其是否存在：

```
{% 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 %}
```

故障排除

验证接口名称

SSH 到虚拟机并检查接口名称：

```
ip link show
```

具有两个次要接口的虚拟机的预期输出：

```
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> ...
```

检查 Netplan 配置

```
cat /etc/netplan/01-netcfg.yaml
```

手动应用 Netplan

```
netplan apply
```

调试 Netplan

```
netplan --debug apply
```

验证路由

```
ip route show
```

相关文档

- 主机文件配置 - 主机清单设置
- Proxmox VM/LXC 部署 - 虚拟机配置
- 配置参考 - 所有配置变量

OmniHSS 复制与节点操作

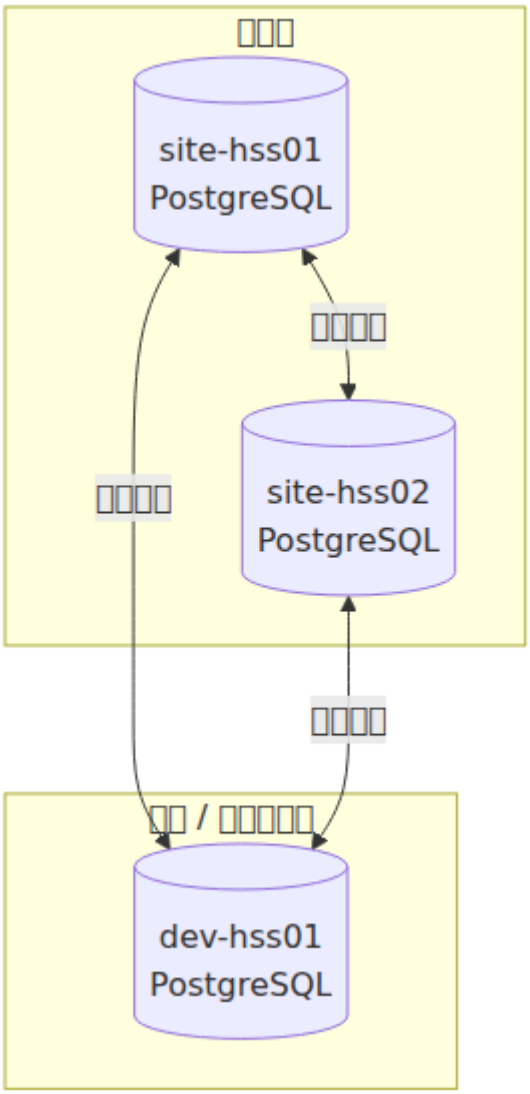
OmniHSS 在每个节点上将其订阅者、身份验证（AuC）和 IMS 数据存储在本地 PostgreSQL 数据库中。双向 **PostgreSQL 逻辑复制** 保持多个 OmniHSS 节点的同步。这形成了一个 **主动-主动网状结构**：每个节点都是完整的读/写主节点，任何节点上的写入都会传播到所有其他节点。没有主/备关系。任何节点都可以在任何时候提供 S6a、Cx/Dx 和 Sh 流量并接受配置。

本指南涵盖了复制模型。它解释了如何添加新的 OmniHSS 节点并加载数据。它还解释了如何验证、修复和移除对等节点。

目录

架构概述

每个 OmniHSS 节点运行其自己的 PostgreSQL 实例。每个节点 **发布** 所有应用程序表并 **订阅** 每个对等节点的发布。`origin = none` 防止复制循环。在快速变化的状态表上，网状结构通过比较 `updated_at` 解决写入冲突 **最后写入胜出**（**LWW**）。



上面的每个链接都是两个单向订阅（每个方向一个）。对于 `N` 个节点的网状结构，每个节点持有 `N-1` 个订阅。对等节点消耗其发布的 `N-1` 个复制槽。

复制内容

表类别	示例	冲突处理
主数据 / 配置数据	`subscriber`, `key_set` (AuC), `sim`, `msisdn`	无需处理：一次从一个地方配置
动态状态	`subscriber_state`, `pdn_session`, `lte_call`, `ims_subscription`	最后写入胜出触发器在 `updated_at` 上
迁移记账	`schema_migrations`	排除 复制

[illegible] `public` [illegible] `schema_migrations`[illegible]LWW [illegible] `updated_at`[illegible]

[illegible]

[illegible]OmniHSS [illegible]

1. [illegible] **`hss`** [illegible] *[illegible]* [illegible]/[illegible] OmniHSS [illegible] `hss01` / `hss02` [illegible] `connected_omnihss` [illegible]
2. [illegible] **`connected_omnihss`** [illegible] *[illegible]* [illegible] OmniHSS [illegible] `hss` [illegible]

[illegible] OmniHSS [illegible] `omnihss.replication.enabled` [illegible] `true` [illegible] `true` [illegible]

[illegible] `connected_omnihss` [illegible]

[illegible]

[illegible]

对象	名称	创建于	唯一于
发布	`<self>_pub`	本节点	节点
订阅	`sub_from_<source>`	本节点（订阅者）	（订阅者）
复制槽	`<subscriber>_from_<source>`	源（发布者）	（订阅者，源）对

槽名称 必须 编码订阅者。考虑如果它仅编码源会发生什么（PostgreSQL 的默认设置，其中槽继承订阅名称）。两个订阅同一源的节点都会尝试在该源上创建同名槽。第二个订阅将因 “*□□□□□□*” 而失败。编码订阅者为每个链接提供了全球唯一的槽。

统一与 `--limit` 安全性

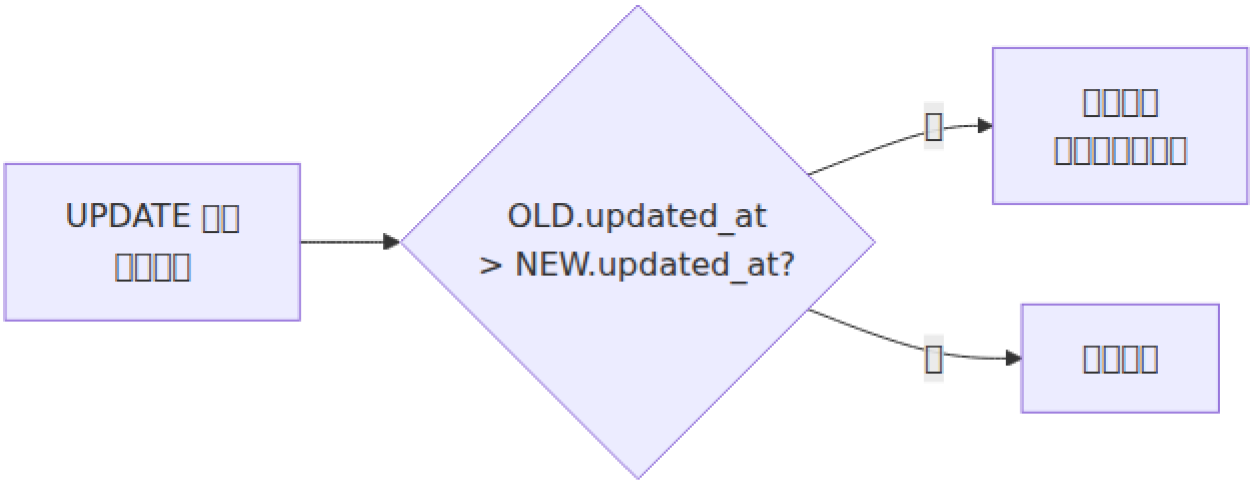
每个角色运行 统一 节点的复制对象与当前有效对等节点集（本地 `hss` 组 + `connected_omnihss`，减去自身）：

- 除 `<self>_pub` 之外的发布会被丢弃（自动清理过时/重命名的发布）。
- 对于不再在有效集中的对等节点的订阅会被禁用，与其远程槽断开并丢弃。
- 不再为当前有效对等节点提供服务的非活动逻辑槽会被丢弃。活动槽不会被触及。

这使得 `connected_omnihss` 成为 权威 网状列表：下一次运行会将从中移除的对等节点（以及相关的 `hss` 组）从每个节点中修剪掉。两个安全属性防止其具有破坏性：

- **`--limit`** 是安全的。有效集源自 `groups['hss']` 和 `connected_omnihss`，无论 `--limit` 如何，总是反映 *□□* 清单成员资格。运行 `--limit hss01` 仅统一 `hss01`。但 `hss01` 仍然将 `hss02`/`hss03` 视为有效并保持其订阅。`--limit` 从不修剪被排除的对等节点。
- 空集从不修剪。如果有效对等节点集计算为空，则完全跳过修剪步骤，而不是清除所有复制（例如，部分清单中只有一个 HSS，且 `connected_omnihss` 未加载）。

> 权威列表警告：由于修剪是自动的，运行会修剪缺失于 `connected_omnihss` 的跨文件对等节点，这些节点实际上是网状结构的一部分。始终确保 `connected_omnihss` 对每个跨文件对等节点保持完整。



配置

启用复制

复制由 `omnihss.replication.enabled` 标志控制。默认值为 `false`。在站点的清单或 group_vars 中设置:

```yaml
omnihss:
  database_type: "postgres"
  database_host: "localhost"
  database_port: 5432
  database_name: "omnihss"
  database_username: "hss"
  database_password: "password"
  replication:
    enabled: true
```

参数	类型	必需	默认	描述
`database_type`	字符串	否	`postgres`	数据库后端。复制仅适用于 `postgres`。
`database_host`	字符串	否	`localhost`	数据库主机。仅在数据库为本地（`localhost`）时配置复制。
`database_port`	整数	否	`5432`	PostgreSQL 端口，也用作对等连接的默认端口。
`database_name`	字符串	否	`omnihss`	数据库名称。必须在所有对等节点中匹配。
`database_username`	字符串	是	`hss`	登录角色。使用 `REPLICATION` 属性创建，以便能够驱动逻辑复制。
`database_password`	字符串	是	`password`	登录角色的密码。必须在所有对等节点中匹配（用于对等连接字符串）。
`replication.enabled`	布尔值	否	`false`	主开关。当为 `true` 时，PostgreSQL 被配置为逻辑复制，并创建发布/订阅。

当 `replication.enabled` 为 `true` 时，该角色会在本地实例上应用以下 PostgreSQL 设置：

设置	值
`wal_level`	`logical`
`max_wal_senders`	`10`
`max_replication_slots`	`10`
`listen_addresses`	`*`
`max_slot_wal_keep_size`	`10GB`（通过 `omnihss.replication.max_slot_wal_keep_size` 重写）

`pg_hba.conf` 被更新以允许每个对等节点的 IP（`/32`）用于 `all` 和 `replication` 连接。

声明跨站点对等节点（`connected_omnihss`）

远程 OmniHSS 节点在每个站点的 `remote_peers_file` 变量引用的远程对等节点文件中声明（通常为 `prod_master_peers.yml`）：

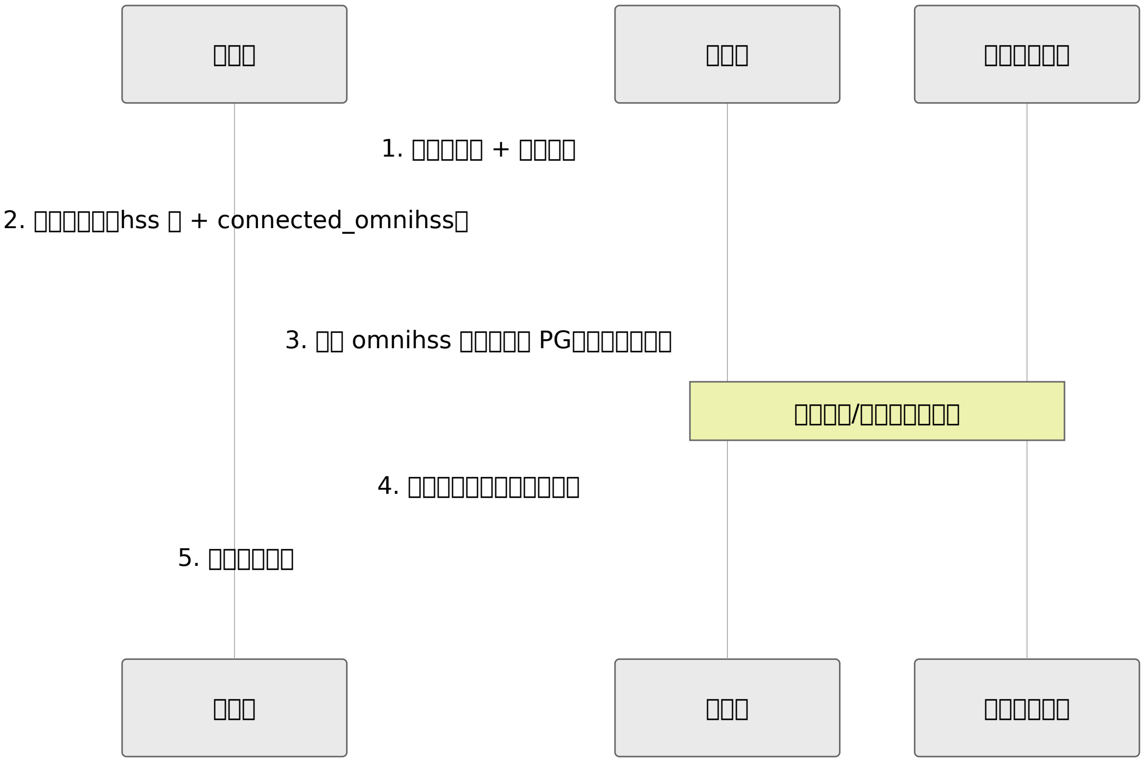
```
connected_omnihss:
  - name: site-hss01
    host: 10.4.2.142
    port: 5432
  - name: site-hss02
    host: 10.4.2.143
    port: 5432
  - name: dev-hss01
    host: 10.4.10.142
    port: 5432
```

参数	类型	必需	默认	描述
`name`	字符串	是	-	对等节点的 `inventory_hostname`。这命名订阅（`sub_from_<name>`）并解析对等节点的发布（`<name>_pub`），因此必须与对等节点的实际清单主机名匹配。
`host`	字符串	是	-	对等节点的 IP 地址，可以通过信令网络从此节点访问。
`port`	整数	否	`5432`	对等节点的 PostgreSQL 端口。回退到 `database_port`。

由于远程对等节点文件被每个站点加载，完整的 `connected_omnihss` 列表应包含网状结构中的每个 OmniHSS 节点。每个站点会自动跳过其本地节点（它们已经由本地 `hss` 组覆盖）并跳过自身。

添加新的 **OmniHSS** 节点

添加节点有两个不同的阶段：加入网状结构（使更改双向流动）和将现有数据加载到新节点。第二个阶段很重要，因为全新的节点从空数据库开始。请参见用数据加载新节点。



第 1 步：配置主机

按照任何 OmniCore 节点的方式配置虚拟机和基本网络（请参见 Proxmox 部署）。该节点必须能够通过信令网络访问每个对等节点的 PostgreSQL 端口（默认 `5432`），并且对等节点必须能够访问它。

第 2 步：将节点添加到清单

将新主机添加到站点 host_file 的 `hss:` 组中：

```yaml
hss:
  hosts:
    site-hss01:
      ansible_host: 10.4.2.142
      gateway: 10.4.2.1
      host_vm_network: "v401-omni-signalling"
    site-hss03:        # 新节点
      ansible_host: 10.4.2.144
      gateway: 10.4.2.1
      host_vm_network: "v401-omni-signalling"
```

然后将其添加到远程对等节点文件中的 `connected_omnihss` 中，以便 其他站点 也能了解它：

```
connected_omnihss:
  - name: site-hss03
    host: 10.4.2.144
    port: 5432
  # ... 现有条目 ...
```

第 3 步：运行 OmniHSS 角色

对新主机运行 OmniHSS 服务剧本。这将安装 PostgreSQL 和 OmniHSS，运行数据库迁移（创建 空 模式），配置逻辑复制，并为每个对等节点创建发布和订阅：

```
ansible-playbook -i hosts/Customer_Name/host_files/Production.yml
\
  services/omnihss.yml --limit site-hss03
```

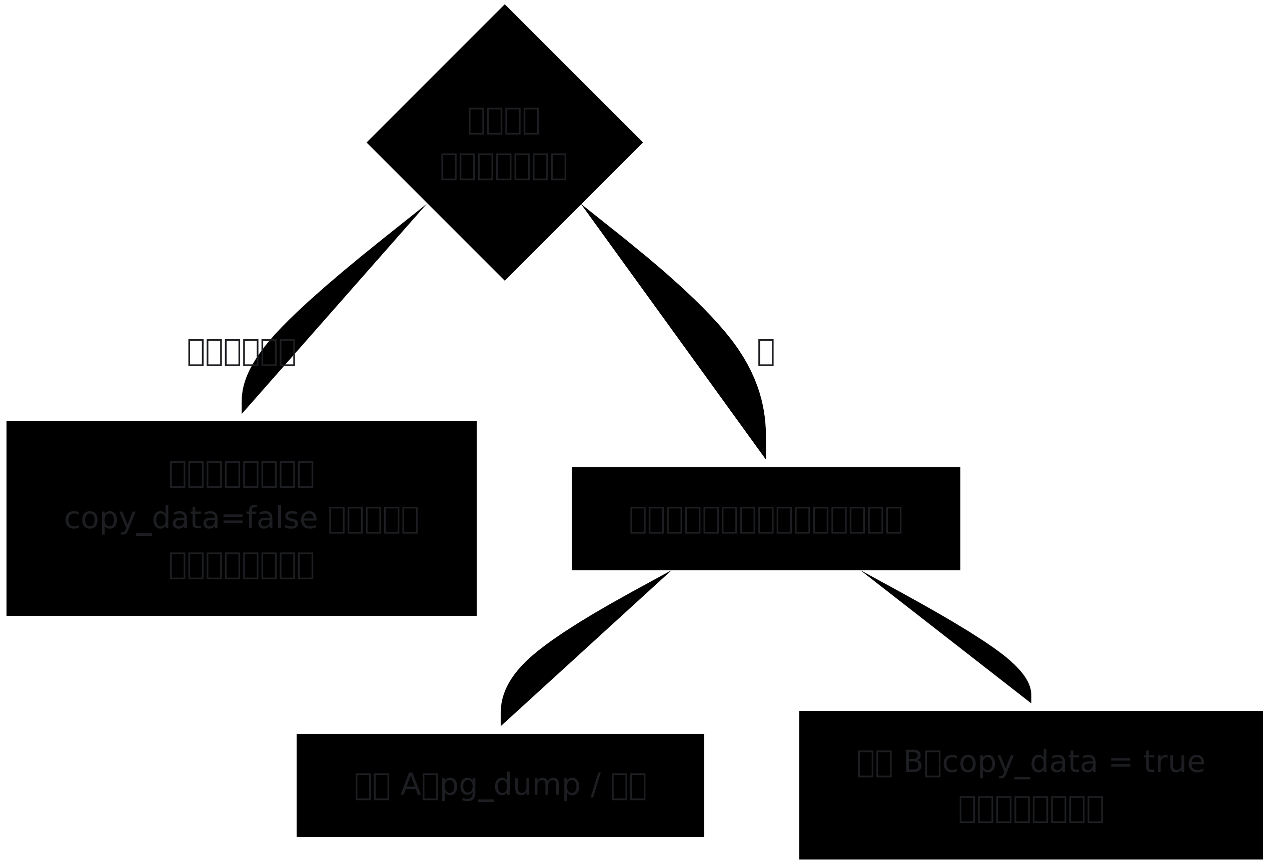
在现有对等节点上重新运行角色（或让下一个计划运行处理它），以便它们为新节点打开 `pg_hba.conf` 并创建回到它的订阅。订阅是幂等的：角色会检测并跳过现有的，并发出 `ALTER SUBSCRIPTION ... REFRESH PUBLICATION` 以便它能够获取任何新表。

此时，新节点已成为网状结构的成员，并接收 所有未来的写入。但它 尚未 包含现有的订阅者基础。继续加载其数据。

用数据加载新节点

该角色创建的自动化订阅使用 `copy_data = false`。这对于 绿色田野网状结构 是正确的（每个节点同时从空开始：没有任何内容可复制，复制会导致重复行冲突）。但这意味着加入 已经填充 的网状结构的节点保持空白，除非在它加入后发生更改。

选择与您的情况相匹配的路径：



绿色田野网状结构

如果您同时启动所有 OmniHSS 节点且没有预先存在的数据，则无需填充。启用复制，在所有节点上运行角色，并正常配置订阅者。每个写入都会复制到所有成员。

选项 A：通过转储和恢复填充（推荐）

获取健康对等节点的快照并将其恢复到新节点。然后让复制传递持续的更改。这重用了标准的备份/恢复工具，并且是最可预测的方法，包括对于大型数据集。

1. 备份健康对等节点。这会在控制机上生成 OmniHSS 数据库的 `pg_dump`（使用 `--no-publications --no-subscriptions`，因此不会携带复制对象）：

   ```
   ansible-playbook -i
   hosts/Customer_Name/host_files/Production.yml \
     services/backup.yml --limit site-hss01
   ```

   转储保存为 `backups/<Site_Name>/hss_dump_<hostname>_<timestamp>.sql`。

2. □□□□□□□ □□□□□□□ OmniHSS□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
ansible-playbook -i
hosts/Customer_Name/host_files/Production.yml \
  util_playbooks/restore_hss.yml --limit site-hss03
```

□□□□□□□□□ 1 □ SQL □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3. □□□□□□□□□ □□□□□□□□□`copy_data = false`□□□□□□□□□□□□□□□□□□□□ □□□

> □□□□□□ □□ □`subscriber_state`, `pdn_session`, `lte_call`, `ims_subscription`□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ B□□□□□□□□□□□□□□□

`seed_hss_replication.yml` □□□□□ □□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□`copy_data = false`□□□□□□□□□□□□□□□□ □□ □□`subscriber_state` □ □□□□□ □□□□□□ 1:1 □□□□□□□□□□ FK □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ API □ S6a□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ UPDATE□□□□□□□□□□□□□□□□□□□□□ □`pdn_session`, `lte_call`□□□□□□□ `COPY` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□

□□□□□□□□ □□

```
ansible-playbook -i hosts/Customer_Name/host_files/Production.yml
\
  util_playbooks/seed_hss_replication.yml \
  --limit site-hss01,site-hss02
```

```
□□□□□□□□□□□□□□□
  1) dev-hss01 (10.4.10.142) - 838 □□□□
□□□□□□□□□□
```

□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
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"
```

[illegible] HSS [illegible] [illegible]

[illegible] `--limit` [illegible] [illegible] A[illegible]

[illegible] `slot_name`[illegible] [illegible][illegible]

```
-- 1. [illegible]
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. 仅回填主表（在目标上运行；-t 每个表，除了
#    schema_migrations, pdn_session, lte_call - 必须包括
subscriber_state，
#    否则每个回填的订阅者将成为孤儿：其
#    subscriber_state_id 指向一个永远不会到达的行，破坏 API/S6a，并且流式传输无法修复它，因为逻辑复
制
#    默默丢弃对缺失行的 UPDATE）。
#    加载必须带有复制来源的标记：origin=none
#    订阅在发布者处过滤带标记的更改，因此回填对网状结构是不可见的。未标记的加载会像任何本地写入一样从 WAL 解码，并将插入回
显到已经持有行的对等节点，
#    导致重复键阻塞其应用工作进程。
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');"
```

验证复制

角色在设置后会自动运行健康检查，也可以单□运行。验证确保：

- 每个订阅都有一个活动的工作进程。
- 每个逻辑复制槽都是活动的。
- 每个槽的WAL延迟低于100 MB。
- 每个槽的WAL状态为`reserved`或`extended`（而不是`lost`）。

要按需检查健康状况，可以重新运行OmniHSS角色（验证在最后运行），或直接在节点上检查：

```
# 订阅工作进程 - 每个进程的pid应为非空
sudo -u postgres psql -d omnihss -c \
  "SELECT subname, pid IS NOT NULL AS worker_alive FROM
pg_stat_subscription;"

# 复制槽 - active 应为 true，wal_status 为 reserved/extended
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';"
```

症状	意义
`worker_alive = false`	订阅工作进程已停止：对等方不可达或凭据/连接字符串错误。
`active = false`（槽）	没有订阅者在消费该槽：对等方的订阅已停止或被删除。
增长的 `lag`	对等方速度慢或链接质量下降。WAL正在积累并消耗磁盘空间。
`wal_status = lost`	槽落后太多，PostgreSQL已删除WAL。您必须重新播种对等方（请参见 加载新节点数据）。

健康检查报告（`health_check.yml`）还包括一个**HSS Postgres** 复制面板，用于启用复制的节点，显示每个订阅的工作状态以及每个槽的活动状态和WAL状态一目了然。

数据完整性

逻辑复制应用不强制执行外键。应用工作进程以 `session_replication_role = replica` 运行，这会抑制实现 `FOREIGN KEY` 约束的内部RI触发器。因此，PostgreSQL 从不重新检查通过复制到达的行与其父行的关系。FK约束提供的是*□□□□*的完整性，而不是*□□*的完整性。

在一个主动-主动的网络中，打开了一个没有单个节点可以捕获的漏洞：

- 节点A删除了 `epc_profile` X。它的 `ON DELETE RESTRICT` 通过，因为A在那一刻没有本地订阅者引用X。
- 节点B（或一个配置步骤）将订阅者203指向X。这通过是因为X仍然存在于B上。

- □□□□□□□□□FK□□□□□□□□□□□□□□□□□□□□□□□□□“203 → X” □ “X□□□”□□□□□□□□□□□□□□□□□□□

移除对等方

移除对等方将从该对等方删除订阅，并移除其 `pg_hba.conf` 条目。已经复制的数据仍然存在；只有未来的更改停止流动。移除必须明确确认。

在您要移除对等方的节点上：

```
DROP SUBSCRIPTION IF EXISTS sub_from_<peer_name>;
```

然后从 `pg_hba.conf` 中移除对等方的行，并重新加载PostgreSQL：

```
sudo -u postgres psql -c 'SELECT pg_reload_conf();'
```

被移除的对等方仍然持有其回到此节点的订阅。如果链接应该完全断开，请在那里也移除该订阅。然后将节点从 `connected_omnihss` 和 `hss` 组中移除，以便下次角色运行时不会重新创建它。

备份与恢复

OmniHSS备份和恢复使用标准PostgreSQL转储：

操作	剧本	备注
备份	`services/backup.yml`	生成 `hss_dump_<host>_<ts>.sql`（数据库）和 `hss_<host>_<ts>.tar.gz`（`/etc/omnihss` 配置）在 `backups/<Site_Name>/` 下。转储排除了出版物/订阅。
恢复	`util_playbooks/restore_hss.yml`	提示输入SQL转储和/或配置档案。停止OmniHSS，进行预恢复备份，删除/重新创建数据库，恢复并重启。

请参见 实用剧本 以获取更广泛的备份/恢复工作流程。

指标

每个OmniHSS节点运行 `postgres_exporter`，将PostgreSQL指标（包括复制）暴露给Prometheus。用于复制健康的有用系列：

指标：`pg_replication_slots_active` 类型：Gauge 描述：每个逻辑复制槽是否有活动消费者。标签：

- `slot_name`：复制槽（每个订阅对等方一个）

指标：`pg_stat_replication_*` / 槽延迟 类型：Gauge 描述：每个槽的WAL位置延迟。它显示消费对等方落后了多少。

示例查询：

```
# 任何非活动的逻辑复制槽（如果 > 0 则警报）
count(pg_replication_slots_active == 0)

# 应存在的订阅与正在运行的工作进程可以在部署期间与验证剧本输出进行交叉检查。
```

OmniHSS自定义指标

每个节点上的 `postgres_exporter` 还通过自定义查询文件（`roles/omnihss/files/postgres_exporter_queries.yaml`）导出OmniHSS特定的复制健康和完整性系列。Prometheus在 `HSS Postgres` 作业下抓取这些数据。它们支持 **HSS**警报 组中的警报规则（`roles/monitoring/templates/grafana/rules/`）：

指标	类型	
`omnihss_subscription_worker_alive{subname}`	Gauge	如果[在运[
`omnihss_replication_slot_active{slot_name}`	Gauge	如果[则为1
`omnihss_replication_slot_wal_status_code{slot_name}`	Gauge	0=r 1= 2=u **3=**
`omnihss_replication_slot_lag_bytes{slot_name}`	Gauge	为该[
`omnihss_subscription_error_apply_error_count{subname}`	Counter	累计[
`omnihss_subscription_error_sync_error_count{subname}`	Counter	累计[
`omnihss_integrity_fk_orphans`	Gauge	具有[0）
`omnihss_schema_max_version` / `omnihss_schema_migration_count`	Gauge	应用[数

请参见 监控与可观察性 以获取指标管道和仪表板。

故障排除

新节点加入后保持空白

症状：复制工作进程处于活动状态且槽是活动的，但新节点没有（或很少）订阅者。

可能原因：

- 节点加入了一个已经填充的网络，并且从未播种。订阅使用 `copy_data = false`，因此只有加入后的更改会到达。

解决方案：

1. 从健康的对等方播种该节点。请参见 加载新节点数据。

订阅工作进程未运行

症状：在 `pg_stat_subscription` 中，某个订阅的 `pid IS NULL`。

可能原因：

- 对等方在PostgreSQL端口不可达（防火墙/路由）。
- 对等方的IP缺失在该节点的 `pg_hba.conf` 中。
- 连接字符串中的凭据错误（`database_username` / `database_password` 在节点之间不同）。

解决方案：

1. 确认TCP可以到达对等方的 `5432`。
2. 验证对等方出现在 `connected_omnihss`（或本地 `hss` 组）中，并重新运行角色以更新 `pg_hba.conf`。
3. 确保 `database_username`/`database_password` 在所有节点之间相同。

复制槽显示 `wal_status = lost`

症状：槽的 `wal_status` 为 `lost`；相应的对等方落后太多。

可能原因：

- 对等方长时间不可用或不可达，以至于所需的WAL被删除。

解决方案：

1. 从健康节点重新播种受影响的对等方。请参见 加载新节点数据。

初始复制期间的重复键错误

症状：在创建 `copy_data = true` 的订阅后立即应用更改时出现错误。

可能原因：

- 初始复制是从多个对等方运行的，导致相同的行被复制了两次。

解决方案：

1. 从恰好一个对等方播种，使用 `copy_data = true`；保持所有其他订阅为 `copy_data = false`。

站点之间的状态更新冲突

症状：状态行（例如 `subscriber_state`）似乎在不同站点设置的值之间“波动”。

可能原因：

- 对同一行在两个站点的并发更新。

解决方案：这是预期的并已处理。最后写入胜出的触发器保留最新的 `updated_at` 更新。确认所有站点的时钟已同步（NTP），以便 `updated_at` 排序是有意义的。

Proxmox VM/LXC □□

□□□□□□□□ Proxmox □□□ OmniCore □□□□□□□ `util_playbooks/proxmox.yml` □□ `deployment_type` □□□□□ QEMU □□□□ LXC □□□□□□□□□□□□□ `proxmoxServers` □□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ LXC □□□□□□□□□□□□ UPF□□□□□□□□□□□□

□□□□□□ VMware□HyperV □□□Vultr / AWS / GCP□□□□□□

□□□

- □□□□□□□□□□□□□□□□□□/LXC
- IP □□□□
- Netplan □□
- □□□□

LXC □ VM

LXC □□□

- □□□□□□□□□□□
- □□□□□□□□□
- □□□□
- □□□□□□□□□□□□□□□□
- □□□□□□ **EPC/IMS** □□
- □□□□ **UPF**□□□□□□□□/TUN □□□

□□□□**KVM**□□

- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□/□□□□□□
- □□□□□□□□
- □□□□□□□
- **UPF** □□□□

□□□

- 虚拟机：生产站点、UPF、所有网络功能
- **LXC**：实验室/测试环境、轻量级服务（apt-cache、监控、dns）

选择部署类型

`deployment_type` 按以下顺序为每个主机解析：

1. 每主机 `deployment_type`（在主机名下设置）：如果存在，则优先。
2. 每站点 `deployment_type` 在 `proxmoxServers` 条目中，轮询选择主机。
3. 默认 `vm` 如果两者都未设置。

常见模式是在 `proxmoxServers` 条目上设置站点默认值（例如 `deployment_type: lxc`），并仅覆盖必须不同的主机。典型示例是一个实验室，除了 UPF 之外，所有内容都作为 LXC 运行，而 UPF 需要内核模块/TUN，必须是虚拟机：

```
all:
  vars:
    proxmoxServers:
      pve-node-01:
        deployment_type: lxc     # 站点默认 → 一切都是 LXC
        # ...

upf:
  hosts:
    site-upf01:
      ansible_host: 192.168.1.24
      deployment_type: vm        # 每主机覆盖 → 此主机是虚拟机
```

作为一种约定，保持站点默认值一致，仅覆盖例外。当您在多个主机上分散虚拟机和 LXC 时，节点放置取决于主机索引，难以推理。

剧本仅在与现有资源的类型冲突时硬性失败：如果具有目标主机名的资源已经存在且类型错误（例如，存在 LXC 但主机解析为 `deployment_type: vm`，或反之），则会以清晰的错误中止。请参见 类型保护。

Proxmox 设置

1. 创建 API 令牌

```
# Proxmox UI: 据中心 → 权限 → API 令牌
# 创建令牌: root@pam!<TokenName>
# 复制令牌密钥（仅显示一次）
# 取消选中 `权限分离` 选项
```

2a. 创建 Cloud-Init 虚拟机模板（仅限虚拟机站点）

在 Proxmox 主机上运行此脚本。它下载 Ubuntu 的云镜像并创建一个干净的模板。`proxmox.yml` 在克隆时注入 cloud-init 凭据和 SSH 密钥（见下面的说明）。

```
#!/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
```

□□□

- □□□□□□□□□□□□□□□`proxmox.yml` □□□□□□□ `ciuser`□`cipassword` □ `sshkeys`□
- Cloud-init □□□□□□□`proxmoxTemplateUser`□□□□□□ `proxmoxServers` □□□□→ □□□ `local_users` □□
- Cloud-init □□□□□□□`proxmoxTemplatePassword`□□□□□□ `proxmoxServers` □□□□→ cloud-init □□□□□□□□□□□□□
- SSH □□□□□□□ `local_users.*.public_key` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□ `local_users` □□□ cloud-init □□□□□□□□□ `proxmoxTemplateUser` □□□ `proxmoxTemplatePassword`□□□□ `proxmoxServers` □□□□□□□□□□□□□□□□□
- `--vga std` □□ Proxmox Web □□□□□□□□□
- wget □ `-N` □□□□□□□□□□□□□□□□

□□□□□□ **ISO** □□□□□□

□□□□□□□□□□□□□□□□□□□□

□□ **1**□□□ **Web UI** □□□□□

- □□□□□□ → □□□ ID 9000□□□□ubuntu-2404-template
- □□□□□□□ Ubuntu Server ISO □□□□□ ISO
- □□□□□SCSI □□□□VirtIO SCSI□
- □□□32GB□□□□SCSI
- CPU□2 □□
- □□□2048 MB
- □□□VirtIO□□□ vmbr0
- □□□□□□□□ Ubuntu Server

□□ **2**□□□□□□□ **-** □□□□□

```
# □□ cloud-init
sudo apt update
sudo apt install cloud-init qemu-guest-agent -y

# □□□□□□□□□
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

# □□ bash □□□□□
history -c
sudo poweroff
```

□□ 3□□□ **Cloud-Init** □□□□□□

- □□□□□ → □□ → □□ → CloudInit □□□□□□□□□□ local-lvm□
- □□ → □□ → QEMU □□□□ → □□
- □□□□□□□ → □□□□□
- □□ □□□□□□ cloud-init □□/□□□`proxmox.yml` □□□□□ `proxmoxTemplateUser` / `proxmoxTemplatePassword` □□□□□□□ `local_users` □□□□□□□□□

2b. □□ LXC □□□□□□□□□ LXC □□□

```
# □ Proxmox □□ shell □□
pveam update
pveam download local ubuntu-24.04-standard_24.04-2_amd64.tar.zst
```

□□□□□□□□□ `local:vztmpl/ubuntu-24.04-standard_24.04-2_amd64.tar.zst`□□□ ��□□□□ `proxmoxLxcOsTemplate` □□□

主机文件配置

虚拟机部署

```yaml
all:
  vars:
    # deployment_type 省略 → 默认值为 "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        # 可选 cloud-init 用户名; 默认为第一个 local_users 键
        proxmoxTemplatePassword: omnitouch    # 可选 cloud-init 密码; 默认为 cloud-init 用户名
        storage: local-lvm  # 可选
      pve-node-02:
        # ... 第二个节点配置

    local_users:
      admin_user:
        name: 管理员用户
        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  # 可选，默认 24
    vlanid: 100  # 可选
```

LXC 部署

```yaml
all:
  vars:
    proxmox_lxc_nameserver: "1.1.1.1"    # 可选，默认 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  # 可选，rootfs 后备
      pve-node-02:
        deployment_type: lxc
        # ... 相同形状

    local_users:
      admin_user:
        name: 管理员用户
        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                          # 可选，默认 24
    vlanid: 100                            # 可选
    proxmoxLxcCores: 2                     # 可选覆盖
    proxmoxLxcMemoryMb: 4096               # 可选覆盖
    proxmoxLxcDiskSizeGb: 30               # 可选覆盖
    proxmoxLxcRootFsStorageName: local-lvm  # 可选，覆盖服务器
proxmoxLxcDefaultStorage
```

□□

□□□□□□□ LXC□

```
ansible-playbook -i hosts/Customer/site.yml
util_playbooks/proxmox.yml
```

□□

```
ansible-playbook -i hosts/Customer/site.yml
util_playbooks/proxmox_delete.yml
```

□□□□□□□□□□□□ LXC□□ `deployment_type` □□□

□□□□

□□□□

- □□□□□□□□□□□□□□□□□□ `deployment_type: lxc` → □□□□□□□□□□□□
- □□□□□LXC □□□□□□□□□□□□□→ □□□
- □ LXC □□□□ UPF □ → □□□□□□□□□□□ UPF □□□□□□□□□

□□□

- □□□□□□□□□ `proxmoxServers` □□□□□□□□□□□□□ `proxmox_server:` □□□□□□□□□ □□□□□□
- □□ Proxmox API □□□□□□□□□ VMID□
- □□□□□ `proxmoxTemplateId` □□□□□ cloud-init □□/□□/SSH □□□cloud-init □□ = □□ □□□□□□□□□□ `proxmoxTemplateUser`□□□□□□□ `local_users` □□cloud-init □□ = □□□□□□ `proxmoxTemplatePassword`□□□□ cloud-init □□□□□□□ `scsi0` □□□□□□
- **LXC**□□ `proxmoxLxcOsTemplate` □□□rootfs □ `proxmoxLxcRootFsStorageName` □□□→ `proxmoxLxcDefaultStorage`□□□□□→ `storage` → `local-lvm`□□□□□□□□□ `ssh-public-keys` □□□□ `local_users.*.public_key`□□□□□□□□□□□

`0mn1Touch@`。所有 LXC 都是非特权的（`unprivileged: 1`），并启用嵌套（`features: nesting=1`）。

现有资源

虚拟机和 LXC 更新路径比较当前与期望并应用更改:

- 网络配置（桥接、VLAN 标签、LXC 的 IP）
- 核心/内存
- 磁盘大小（虚拟机 `scsi0` / LXC `rootfs`）
- 来自 `secondary_ips` 的辅助 NIC 如果网络或资源发生更改，则重新启动资源。

> **OmniUPF**：工具将名为 `n3`（或 `data`/`gtp`）的 `secondary_ips` 条目视为 专用用户平面 NIC。 `omniupf` 角色将 N3/N9 + XDP 绑定到它，并为主接口保留一个 CPU，以便重负载数据路径不会耗尽 PFCP/API/SSH（这些仍然在主 IP 上）。

标记

每个资源都被标记为其 Ansible 组名称 + `site_name`。

跨节点分配

默认情况下，工具通过 轮询在 `proxmoxServers` 条目之间分散主机，对于虚拟机和 LXC 使用相同的逻辑。索引是主机在其角色组中的位置，取模 `proxmoxServers` 条目数量（按声明顺序）：

```
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)
```

> 添加第二个节点会重新分配所有内容。由于放置是位置性的，从一个 `proxmoxServers` 条目到两个会改变现有多主机组的位置（主机-0 → 条目-0，主机-1 → 条目-1，...）。在您对混合集群重新运行之前，固定您不希望移动的主机（如下所示）。

将主机固定到特定节点

将可选的 `proxmox_server:` 键设置为 **`proxmoxServers`** 条目的名称 以强制主机无论其组索引如何都放置在该节点上。当未设置时，上述轮询适用。不是 `proxmoxServers` 键的值会导致运行失败，并列出有效名称的错误（以便拼写错误不会默默地轮询到其他地方）。

```
device_tester:
  hosts:
    site-device-tester01:
      ansible_host: 10.179.2.50
      proxmox_server: nicklab-prxmx01    # 固定到此 proxmoxServers 条目（↪
其节点）
      # 如果该节点的磁盘存储与全局 proxmoxLxcRootFsStorageName 不同，
      # 也在此处覆盖（存储是全局读取的，而不是从服务器条目读取）：
      proxmoxLxcRootFsStorageName: local-lvm
```

`proxmox_server` 也可以在组 `vars:` 范围内设置，以将组中每个主机固定到一个节点。

每主机覆盖

`deployment_type`、`proxmox_server`（节点固定：见 跨节点分配）以及任何组级 LXC 调整（`proxmoxLxcCores`、`proxmoxLxcMemoryMb`、`proxmoxLxcDiskSizeGb`、`proxmoxLxcRootFsStorageName`、`proxmoxLxcOsTemplate`、`host_vm_network`）都可以在主机名下逐主机设置。

```
dns:
  hosts:
    site-dns01:
      ansible_host: 192.168.1.20
      deployment_type: vm          # 每主机覆盖站点默认值
      proxmoxLxcDiskSizeGb: 120    # 每主机覆盖
      host_vm_network: vmbr1       # 每主机桥接覆盖
  vars:
    proxmox_interface: vmbr0
    gateway: 192.168.1.1
    mask_cidr: 24   # 可选，默认 24
```

服务手册

服务手册用于部署和配置 OmniCore 基础设施。位于 `services/` 目录中。

手册层级

手册以层级结构组织，以便快速、针对性地部署：

```
all.yml
├── setup_users.yml
├── apt_cache.yml           # 仅在定义了 apt_cache_servers 组时
├── 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
```

仅运行您需要的内容。部署单个组件（例如 `omnimme.yml`）只需几秒钟。跨 50 个主机运行 `all.yml` 需要更长时间，但可以处理所有内容。使用 `--limit` 进一步缩小范围。

快速参考

顶级手册

手册	范围	描述
`all.yml`	全网络	完整部署：用户、DNS、监控、设备测试器、EPC、IMS、ePDG、TWAG、SS7、OCS、CRM、RAN 监控、5G 核心，以及最终的健康检查（来自 `util_playbooks/`）
`epc.yml`	数据包核心	MME、SGW-C、PGW-C、UPF、HSS、DRA
`ims.yml`	语音/IMS	P/I/S-CSCF、应用服务器、消息、OmniSEP（XCAP/权限/BSF）
`ocs.yml`	计费	CGrateS、KeyDB 集群、OCS 同步
`monitoring.yml`	可观察性	Prometheus、Grafana、出口程序、HOMER

基础设施手册

手册	描述
`common.yml`	基础操作系统配置、软件包、NTP、日志代理
`setup_users.yml`	本地用户账户和 SSH 密钥
`dns.yml`	DNS 服务器部署
`license_server.yml`	OmniCore 许可证服务器
`does_netplan_exist.yml`	检查每个主机上是否安装了 netplan
`update_mtu.yml`	通过 netplan 设置和应用接口 MTU
`firewall.yml`	iptables/nftables 规则
`apt_cache.yml`	本地 APT 镜像（如果启用）

EPC 组件

手册	组件	描述
`omnimme.yml`	OmniMME	移动管理实体 (4G)
`omnisgwc.yml`	OmniSGW-C	服务网关控制平面
`omnipgwc.yml`	OmniPGW-C	PDN 网关控制平面
`upf.yml` / `omniupf.yml`	OmniUPF	用户平面功能（合并 SGW-U/PGW-U）
`omnihss.yml` / `hss.yml`	OmniHSS	家庭用户服务器
`omnidra.yml`	OmniDRA	Diameter 路由代理
`omnitwag.yml`	OmniTWAG	可信无线接入网关
`omniepdg.yml`	OmniEPDG	进化数据包数据网关（WiFi 通话）
`gtp_proxy.yml`	GTP 代理	GTP 流量代理

IMS 组件

手册	组件	描述
`pcscf.yml`	P-CSCF	代理呼叫会话控制功能
`icscf.yml`	I-CSCF	查询 CSCF
`scscf.yml`	S-CSCF	服务 CSCF
`as.yml`	OmniTAS	电话应用服务器
`omnisep.yml`	OmniSEP	XCAP、权限服务器、BSF、可视语音邮件
`xcap.yml`	OmniSEP	`omnisep.yml` 的遗留别名（部署相同的 `omnisep` 角色）
`omnimessage.yml`	OmniMessage / IMS SMSC	SMS-over-IP 控制器、SMPP 和 IMS SMSC（通过 `cscf` 角色以 `cscf_type: smsc_ims` 部署）

5G 核心

通过 `5gc.yml` 部署。每个网络功能由共享的 `5gc_nf` 角色安装，并具有每个 NF 的 `nf_package`。有关角色内部、每个 NF 配置、版本固定和 UDM HNET/SUCI 密钥，请参见 5G Core。

手册	组件	描述
`5gc.yml`	5G 核心	部署所有 5G □立网络功能
	OmniNRF	网络存储库功能
	OmniAMF	接入和移动管理功能
	OmniAUSF	认证服务器功能
	OmniBSF	绑定支持功能
	OmniCHF	计费功能
	OmniNSSF	网络切片选择功能
	OmniPCF	策略控制功能
	OmniSCP	服务通信代理
	OmniSMF	会话管理功能
	OmniUDM	统一数据管理
	OmniUDR	统一数据存储

电路交换

通过 `cs.yml` 部署（角色位于 `circuit_switched/` 下）。有关 BSC、MSC 和 SS7 角色的详细信息，请参见 Circuit-Switched Core。

手册	组件	描述
`cs.yml`	电路交换	部署传统 CS 领域
	BSC	基站控制器（`circuit_switched/bsc`）
	OmniMSC	移动交换中心（`circuit_switched/omnimsc`）
	SS7	SS7/SIGTRAN 堆栈（`omniss7` 角色）

支持服务

手册	描述
`ocs.yml`	在线计费系统（CGrateS + KeyDB）
`crm.yml`	客户管理门户（OmniCRM；配置现在位于 `group_vars/` 中）
`omniss7.yml`	SS7/SIGTRAN 网关
`homer.yml`	SIP/Diameter 数据包捕获
`grafana.yml`	Grafana 仪表板和警报
`ran_monitor.yml`	RAN 监控集成
`omniroam.yml`	OmniRoam 漫游套件（TAP3/RAP UI、Elixir API、生成器）
`omnilcs.yml`	OmniLCS 位置服务和小区广播
`device_tester.yml`	OmniWeb 手机代理测试台（测试设备）

> **Proxmox** 配置（`proxmox.yml`，`proxmox_delete.yml`）位于 `util_playbooks/` 下。请参见 Utility Playbooks 和 Proxmox VM/LXC 部署。

操作

手册	描述
`backup.yml`	备份数据库和配置
`reboot.yml`	控制主机重启
`shutdown.yml`	优雅关机
`apt_update.yml`	更新软件包
`apt_refresh_metadata.yml`	刷新 APT 缓存元数据
`speedtest.yml`	网络吞吐量测试

恢复手册

手册	描述
`restore_applicationserver.yml`	从备份恢复 OmniTAS
`restore_smsc.yml`	从备份恢复 SMSC：OmniMessage 控制器、SMPP 网关和 IMS SMSc 节点

> 注意：`health_check.yml`、`restore_hss.yml` 和 `restore_ocs.yml` 不在 `services/` 中。它们位于 `util_playbooks/` 中。请参见 Utility Playbooks。`all.yml` 作为 `../util_playbooks/health_check.yml` 运行健康检查。

使用

部署所有内容

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

部署特定子系统

```
# 仅□据包核心
ansible-playbook -i hosts/customer/host_files/production.yml
services/epc.yml

# 仅 IMS/语音
ansible-playbook -i hosts/customer/host_files/production.yml
services/ims.yml
```

部署单个组件

```
# 仅更新 MME
ansible-playbook -i hosts/customer/host_files/production.yml
services/omnimme.yml

# 仅更新监控
ansible-playbook -i hosts/customer/host_files/production.yml
services/monitoring.yml
```

□□□□□□

```
# □□□□□□□□□□ all.yml
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit mme01

# □□□□□□□□□□□
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit "mme01,hss01"

# □□□□□□□□
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit mme
```

□□□□

- □□□□ - □□□□□□□□
- □□□□□□□ - □□□□□□
- □□□□□ - □□□
- □□□□□ - □□□□□□□□□□□□□□□□
- □□□□□□□□ - Grafana□Prometheus□□□

工具剧本

工具剧本提供了管理已部署 OmniCore 基础设施的操作工具。这些剧本位于 `util_playbooks/` 目录中，可以独立运行以执行常见的维护和故障排除任务。

□□□□

□□	□□
`proxmox.yml`	□ Proxmox □□□□□□□□/□ LXC □□□□□□ `deployment_type`□
`proxmox_delete.yml`	□□ Proxmox □□□□□□/LXC
`proxmox_restart_hung.yml`	□ SSH □□□□□□□□ Proxmox API □□□□□□□□/LXC
`vmware.yml`	□ VMware vSphere □□□□□□□
`vmware_delete.yml`	□□ VMware vSphere □□□□□□
`health_check.yml`	□□□□□□□□□□□□□□□□
`restore_hss.yml`	□□□□□□ HSS □□□□□/□□□□
`seed_hss_replication.yml`	□□□□□□□□□□□□□□□□□□ OmniHSS □□□□□□□□□□□□□□□
`audit_hss_drift.yml`	□□ OmniHSS □□□□□□□□□□□□□□□□□□□□ + □□□□□□□□□□
`reconcile_hss_drift.yml`	□□□□□□□□□□□□□□□□□□□□ OmniHSS □□□□□□□□□□□□□□□□□□
`restore_ocs.yml`	□□□□□□ OCS KeyDB□MySQL □□□□
`ip_plan_generator.yml`	□□□□□ Mermaid □□□□□□□
`get_ports.yml`	□□□□□□□□□□□□□□□□□□□□
`getLocalCapture.yml`	□□□□□□□□□□□□□□□
`delete_local_user.yml`	□□□□□□□□□□□□□□□□□□

剧本	目的
`clear_mnesia.yml`	清除 OmniSS7 / OmniMessage 的 Mnesia 数据库并重启（破坏性操作）
`delete_license.yml`	删除许可证服务器上的许可证文件并重启

虚拟机配置

虚拟机配置剧本在虚拟化平台上创建和管理虚拟机。所有剧本都支持 `--limit` 以针对特定主机或组。

有关详细配置，请参见 Proxmox 部署。

Proxmox

文件：`util_playbooks/proxmox.yml`，`util_playbooks/proxmox_delete.yml`

`proxmox.yml` 同时配置虚拟机和 LXC 容器；给定主机是成为虚拟机还是 LXC 由其解析的 `deployment_type` 决定（每主机覆盖 → `proxmoxServers` 条目 → 默认 `vm`）。请参见 Proxmox 部署。

```
# 配置（虚拟机和/或 LXC 容器，根据 deployment_type）
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox.yml

# 仅配置特定组
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox.yml --limit mme

# 删除虚拟机/LXC
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox_delete.yml --limit old-host
```

Proxmox 重启无响应主机

文件：`util_playbooks/proxmox_restart_hung.yml`

检查每个主机的 SSH 响应性，并通过 Proxmox API 强制重启任何无响应的虚拟机或 LXC 容器。QEMU 虚拟机被强制停止并重新启动；LXC 容器被启动（如果已经在运行则重启）。响应 SSH 的主机将被跳过。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox_restart_hung.yml

# 限制为单个主机或组
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/proxmox_restart_hung.yml --limit hss01
```

VMware vSphere

文件**:** `util_playbooks/vmware.yml`, `util_playbooks/vmware_delete.yml`

这两个剧本都包括位于 `util_playbooks/vm/vmware/` 下的每个虚拟机任务文件（`create_individual_vm.yml` 和 `delete_individual_vm.yml`）。`vmware_delete.yml` 是删除任务的薄包装；`vmware.yml` 还在配置后等待 SSH，然后删除过时的 netplan 文件并在新虚拟机上应用 netplan。

先决条件**:** 需要安装 Python 依赖项（`uv sync`）。

用法**:**

```
# 配置虚拟机
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vmware.yml

# 仅配置特定组
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vmware.yml --limit mme

# 删除虚拟机
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vmware_delete.yml --limit old-host
```

所需变量**:**

```
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

文件：`util_playbooks/vm/vultr/vmProvisionVultr.yml`，`util_playbooks/vm/vultr/vmTeardownVultr.yml`，`util_playbooks/vm/vultr/vmCommonSetup.yml`

通过 Vultr API 在 Vultr 上配置和拆除实例。`vmProvisionVultr.yml` 创建实例，`vmTeardownVultr.yml` 删除它们，`vmCommonSetup.yml` 执行配置后的基础设置（DNS、重启等）。这三个都是顶级入口剧本，而不是任务包含。它们从 `VULTR_API_TOKEN` 环境变量读取 API 令牌，配置剧本引用同一目录中的 `cloud-config.yaml`，因此从 `util_playbooks/vm/vultr/` 运行它们。

```
export VULTR_API_TOKEN=<token>
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/vm/vultr/vmProvisionVultr.yml
```

健康检查

文件：`util_playbooks/health_check.yml`

生成涵盖所有已部署 OmniCore 和 OmniCall 服务的全面 HTML 健康报告。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/health_check.yml
```

输出: `/tmp/health_check_YYYY-MM-DD HH:MM:SS.html`

收集的信息

组件	收集的数据
所有服务	服务状态、版本、正常运行时间
OmniHSS	数据库状态、Diameter 对等连接
OmniDRA	Diameter 对等连接和状态
OmniTAS	活动呼叫、会话、注册、CPU 使用率
OCS	KeyDB 复制状态

HSS 恢复

文件: `util_playbooks/restore_hss.yml`

从备份文件恢复 OmniHSS。支持仅恢复数据库、仅恢复配置或两者都恢复。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/restore_hss.yml
```

备份文件格式

类型	文件名模式	内容
数据库	`hss_dump_<hostname>_<timestamp>.sql`	OmniHSS 数据库的 PostgreSQL (pg_dump) 转储（默认 `omnihss`）
配置	`hss_<hostname>_<timestamp>.tar.gz`	`/etc/omnihss` 目录的归档

在恢复运行之前，当前 HSS 状态（数据库和/或配置）会被备份。

提示

提示	必需	描述
HSS SQL 转储路径	否	HSS SQL 转储文件的完整路径（留空以跳过数据库恢复）
HSS 配置 tar.gz 路径	否	HSS 配置 tar.gz 文件的完整路径（留空以跳过配置恢复）
确认	是	输入 `yes` 确认在列出的主机上执行恢复操作

HSS 复制种子

文件**:** `util_playbooks/seed_hss_replication.yml`

对加入已填充的主动-主动复制网格的 OmniHSS 节点执行一次性初始数据加载。新节点为空（订阅使用 `copy_data = false` 创建）；此剧本使用 `copy_data = true` 重新创建对单个选定源对等节点的订阅，因此 PostgreSQL 复制现有数据，然后继续流式传输而没有间隙。

它以两种模式之一运行：

交互式（默认）**:** 省略 `seed_source_*` 变量。剧本检测哪些有限目标是空的。然后它探测有效对等节点中持有数据的节点，列出它们及其订阅者计数，并提示您选择源。它仅对空目标进行种子。

```
ansible-playbook -i hosts/customer/host_files/production.yml \
  util_playbooks/seed_hss_replication.yml \
  --limit site-hss01,site-hss02
```

```
可用于种子的对等节点:
  1) dev-hss01 (10.4.10.142) - 838 个订阅者
输入要种子的源的编号
```

显式： 直接传递源以跳过菜单。这 强制 重新种子，即使目标已经持有数据（用于恢复）。

```
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"
```

参数

参数	必需	默认	描述
`seed_source_name`	否	（提示）	已填充源对等节点的清单主机名。用于命名订阅（`sub_from_<name>`）并解析其发布（`<name>_pub`）。在交互模式下省略。
`seed_source_host`	否	（提示）	源对等节点的 IP 地址，可在其 PostgreSQL 端口上访问。在交互模式下省略。
`seed_source_port`	否	目标的 `omnihss.database_port` (5432)	源对等节点的 PostgreSQL 端口。

> `seed_source_name` 和 `seed_source_host` 必须一起提供以使用显式模式；提供两者都不触发交互模式。

□□

1. □□□□□□□□□□□□□□□□□□□□□□□□□□□□
2. □□□□□: □□□□□□□□□□□□□ `hss` □ + `connected_omnihss`□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ □□: □□□□□ `yes` □□□□□
3. □□□ IP □□□□□ `pg_hba.conf` □□□□□□
4. □□□□□□□□□□□□□□ `WITH (copy_data = false)`□□□□□□□ `slot_name` □ `<target>_from_<source>`□□□□□□□□ □□ □□□□□□□□□□□□□□□
5. □□□□□□□ □□ □□□□□□□□□□□□□`pg_dump --data-only --inserts --on-conflict-do-nothing --disable-triggers`□□`subscriber_state` □ □□□□□□□□□□□□ FK □ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□`pdn_session`□`lte_call`□ □ □□□□□
6. □□□□□□□□□□□□□□□□□

> □□□□□□□□□: □□□□□□□□□□□□□□□□□□□□□□ `COPY` □□□□□□□□□□□□□□□□□□□□

在每次 **HSS** 部署后自动运行。在 `omnihss` 角色的末尾（启用复制时），相同的审计作为后部署一致性检查运行，默认情况下如果发现漂移则失败该剧本，因此在部署期间不可能错过破损的网格。它是非交互式的（没有暂停）：审计必须在聚合 `services/all.yml` 部署的 `free` 策略下安全，其中暂停会出错。在该聚合部署中，角色内审计被抑制，审计在 `services/deploy_nfs.yml` 末尾以专用 线性 剧本运行。行为由两个角色变量控制：

变量	默认	效果
`omnihss_audit_drift`	`true`	在角色结束时运行审计
`omnihss_audit_fail_on_drift`	`true`	在漂移时失败该剧本；设置为 `false` 仅报告（非交互式 cron/CI）

审计是 只读 的。□立剧本默认情况下在发现漂移时报告然后 失败（因此可以控制 cron/CI）：

```
# 按需
ansible-playbook -i hosts/<inv>/host_files/<site>.yml
util_playbooks/audit_hss_drift.yml
# 仅报告，始终通过
ansible-playbook -i ... util_playbooks/audit_hss_drift.yml -e
audit_fail_on_drift=false
# 禁用单次部署的自动审计
ansible-playbook -i ... services/omnihss.yml -e
omnihss_audit_drift=false
```

HSS 漂移调和

文件**:** `util_playbooks/reconcile_hss_drift.yml`

通过使每个节点持有 联合 的行来将不同的参考表重新同步：对于每个漂移的表，它从其对等节点回填节点缺失的行，使用与种子剧本相同的冲突安全原语（`pg_dump --data-only --inserts --on-conflict-do-nothing --disable-triggers`）。

- 非破坏性：行仅在 INSERT 时添加（跳过冲突）。没有任何内容被删除或更新。它解决“节点缺失行”；它 不 解决“相同主键，不同值”或删除不应存在的行。这些需要人工决策。
- 默认交互式：它扫描、打印计划，并 询问（输入 `yes` 继续）后再写入；其他任何内容都会中止且不做更改。使用 `-e dry_run=true` 仅查看，或 `-e assume_yes=true` 进行非交互式运行。

- □□□□□□□□□□ postgres □□□□□□□□□□□□□□□ `--disable-triggers`□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ // □□□□□□□□□□□□□□□□□□□□

6. 在所有节点上启动 CGrateS

备份文件格式

类型	文件名模式	内容
KeyDB 数据库	`keydb_dump_<hostname>_<timestamp>.rdb`	KeyDB RDB 快照
MySQL StoreDB	`cgrates_dump_<hostname>_<timestamp>.sql`	`cgrates` 数据库的 MySQL 转储
配置	`cgrates_<hostname>_<timestamp>.tar.gz`	`/etc/cgrates` 目录的归档

提示

提示	必需	描述
KeyDB RDB 转储路径	是	KeyDB RDB 备份文件的完整路径
MySQL SQL 转储路径	否	CGrateS SQL 转储的完整路径（跳过以保留当前 StoreDB）
配置 tar.gz 路径	否	配置备份的完整路径（跳过以保留当前配置）

IP 计划生成器

文件**:** `util_playbooks/ip_plan_generator.yml`

从清单生成网络文档，包括：

- 主机 IP 分配（主 NIC 和次 NIC）
- 网络段概述
- 接口连接图（Diameter、GTP、PFCP、SIP、SS7）

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/ip_plan_generator.yml
```

输出文件

文件	格式	描述
`/tmp/ip_plan_<customer>_<site>.md`	Markdown	基于文本的文档
`/tmp/ip_plan_<customer>_<site>.html`	HTML	带有可过滤层的交互式图

端口审计

文件：`util_playbooks/get_ports.yml`

审计部署中的所有监听端口并生成文档。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/get_ports.yml
```

输出文件

文件	描述
`/tmp/all_ports.csv`	包含主机名、IP、协议、端口、服务的 CSV 文件
`./open_ports.rst`	用于 Sphinx 文档的 reStructuredText 表

收集的数据

字段	描述
主机名	清单主机名
IP	主机的 `ansible_host` IP 地址
IP 版本	IPv4 或 IPv6
传输	TCP 或 UDP
端口	监听端口号
服务	进程名称

本地捕获检索

文件：`util_playbooks/getLocalCapture.yml`

从每个主机的 `/etc/localcapture` 目录检索两个最近的数据包捕获文件。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/getLocalCapture.yml
```

输出：`./localCapturePcaps/<hostname>/*.pcap`

用户管理

文件：`util_playbooks/delete_local_user.yml`

从清单中的所有主机中删除本地用户帐户。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/delete_local_user.yml
```

提示：当提示时输入要删除的用户名。

清除 **Mnesia** 数据库

文件：`util_playbooks/clear_mnesia.yml`

停止服务，删除其 Mnesia 数据库目录并重启。针对 `ss7` 组（OmniSS7）和 `omnimessage` 组（OmniMessage）运行。

> 警告：这是破坏性的。Mnesia 数据库在重启时会被永久删除并重新构建为空。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/clear_mnesia.yml

# 限制为单个服务或主机
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/clear_mnesia.yml --limit ss7
```

删除许可证

文件：`util_playbooks/delete_license.yml`

删除 `license_server` 组上的 `/etc/license_server/license.json` 并重启许可证服务器。

```
ansible-playbook -i hosts/customer/host_files/production.yml
util_playbooks/delete_license.yml
```

运行工具剧本

基本语法

```
ansible-playbook -i <inventory_file> util_playbooks/<playbook>.yml
```

常用选项

选项	描述
`-i <inventory>`	指定清单文件
`--limit <hosts>`	限制为特定主机或组
`-v` / `-vv` / `-vvv`	增加详细程度
`--check`	干运行（显示将会更改的内容）
`--diff`	显示文件差异

示例

```
# 在生产环境中运行健康检查
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/health_check.yml

# 在特定主机上恢复 HSS
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/restore_hss.yml --limit hss01

# 生成 IP 计划并输出详细信息
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/ip_plan_generator.yml -v
```