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。

```
- 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`	库存组	配置模板	角色
网络存储库功能	`omninrf`	`nrf`	`omninrf.j2`	服务发现 / NF注册
接入与移动管理	`omniamf`	`amf`	`omniamf.j2`	N1/N2信令、注册、连接管理
认证服务器	`omniausf`	`ausf`	`omniausf.j2`	5G-AKA / EAP-AKA'认证
绑定支持功能	`omnibsf`	`bsf`	`omnibsf.j2`	PDU会话的PCF绑定
计费功能	`omnichf`	`chf`	`omnichf.j2`	融合的在线/离线计费
网络切片选择	`omnissf`	`nssf`	`omnissf.j2`	NSSAI / 切片选择
策略控制功能	`omnipcf`	`pcf`	`omnipcf.j2`	策略和QoS规则
服务通信代理	`omniscp`	`scp`	`omniscp.j2`	SBI消息路由 / 间接通信
会话管理功能	`omnismf`	`smf`	`omnismf.j2`	PDU会话和UPF（N4）控制
统一数据管理	`omniudm`	`udm`	`omniudm.j2`	订阅数据、SUCI去隐蔽
统一数据存储	`omniudr`	`udr`	`omniudr.j2`	UDM/PCF的后端数据存储

`5gc_nf`角色的工作原理

对于每个NF，该角色执行相同的顺序：

1. 解析`group_vars_omnicore`：设置为`<inventory>/group_vars/`，因此配置查找相对于活动库存解析。
2. 发现许可服务器：如果未设置`license_server_api_urls`，则从`license_server`库存组派生（`https://<host>:8443/api`）。
3. 安装先决条件：`lsb-release`、`curl`、`gpg`、`unzip`、`libglu1-mesa`、`libsctp1`。
4. 从**Omnitouch APT**存储库安装**NF**包。默认安装最新版本；使用`nf_version`固定版本（见版本固定）。

5. 模板化运行时配置（版本化）：将`templates/<nf_package>.j2`渲染到`/etc/<nf_package>/runtime-<deb_version>.exs`。
6. 符号链接`runtime.exs` → 活动版本的配置，并将该符号链接镜像到每个`/opt/<nf_package>/releases/*`目录中。
7. 重启**NF**（处理程序），每当活动配置更改时。

版本固定和回滚

配置是按Debian版本编写的（`runtime-<deb_version>.exs`），`runtime.exs`是指向活动版本的符号链接。先前的配置保留在磁盘上，因此回滚只需重新安装旧的包版本：

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

省略`nf_version`以跟踪最新包。

UDM HNET密钥（SUCI去隐蔽）

UDM（`nf_package: omniudm`）有额外步骤以支持SUCI去隐蔽（TS 33.501 §6.12）。去隐蔽从UE在注册时呈现的隐蔽SUCI中恢复SUPI。它需要与USIM上的家庭网络公钥配对的家庭网络私钥。

- 密钥存储在Ansible控制器的`group_vars/hnet_udm/`中，命名为`<curve>-<id>.key`（PEM），其中`<curve>`为`curve25519`（Profile A, X25519）或`secp256r1`（Profile B, prime256v1），`<id>`为家网络公钥标识符。
- 如果目录为空，则自动生成默认的`curve25519-1.key`，以便去隐蔽开箱即用。此步骤是幂等的。现有密钥不会被覆盖。
- 对于每个`curve25519-*.key`，匹配的原始32字节公值将以（十六进制）导出到`.pub`文件中，用于USIM配置。`.pub`文件保留在库存中；只有`*.key`文件部署到UDM上的`/etc/omniudm/hnet/`（模式`0600`，root）。

> 自动生成的密钥是随机的，仅从使用其匹配公钥编程的USIM中去隐蔽SUCIs。获取导出的`.pub`并将其配置为USIM上的家庭网络公钥。使用空方案的SIM无需HNET密钥，可以将目录留空。

详细信息和文件布局示例：组变量配置：示例8。

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 规划标准

概述

本文档概述了 OmniCore 部署的标准 IP 规划方法。该架构需要 四个内部子网 以适当地分隔网络功能，以确保安全性、性能和操作清晰度。

IP 分配要求

标准分配：四个 /24 子网

每个 OmniCore 部署需要四个不同的子网用于内部网络：

1. 数据包□心网络 - 第一个 /24
2. 信令网络 - 第二个 /24
3. **IMS** 内部网络 - 第三个 /24
4. **IMS** 公共网络 - 第四个 /24

重要提示：这些是建议，而不是要求

本文档中描述的子网分配是组织 OmniCore 部署的 推荐最佳实践。然而，该架构是 完全灵活的：

- 所有主机在一个子网中：如果符合您的部署需求，可以将所有组件放置在一个子网中
- 每种主机类型在其自己的子网中：可以为每种组件类型创建单□的子网（一个用于 MME，一个用于 HSS，等等）
- 自定义分组：可以根据您的具体要求将���机组织到任何子网结构中
- 混合内部和公共 **IP**：某些主机可以使用内部（RFC 1918）地址，而其他主机则使用公共 IP，所有这些都在同一部署中

推荐的四子网方法提供了最佳的 安全隔离、流量管理和操作清晰度，这就是我们建议在生产部署中使用它的原因。然而，您应该根据您的具体网络拓扑、可用地址空间和操作要求调整 IP 计划。

网络段细分

1. 数据包□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`

```
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`

```
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□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
# □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：

```
# 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：

```
- 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：

```
- 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□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

这种方法的好处

可重复性

相同的 Ansible 剧本��以部署:

- 开发实验室
- 测试环境
- 生产网络
- 客户站点

一致性

每次部署都使用相同的经过测试的配置，减少人为错误。

版本控制

基础设施作为代码在 Git 中定义:

- 跟踪所有更改
- 部署前审查
- 如有需要可回滚

无复杂性的定制

客户可以通过 `group_vars` 自定义其部署，而无需修改核心角色。

快速部署

在几小时内而不是几天或几周内部署完整的蜂窝网络。

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

在环境激活的情况下，直接运行 Ansible 命令：

```
ansible --version
```

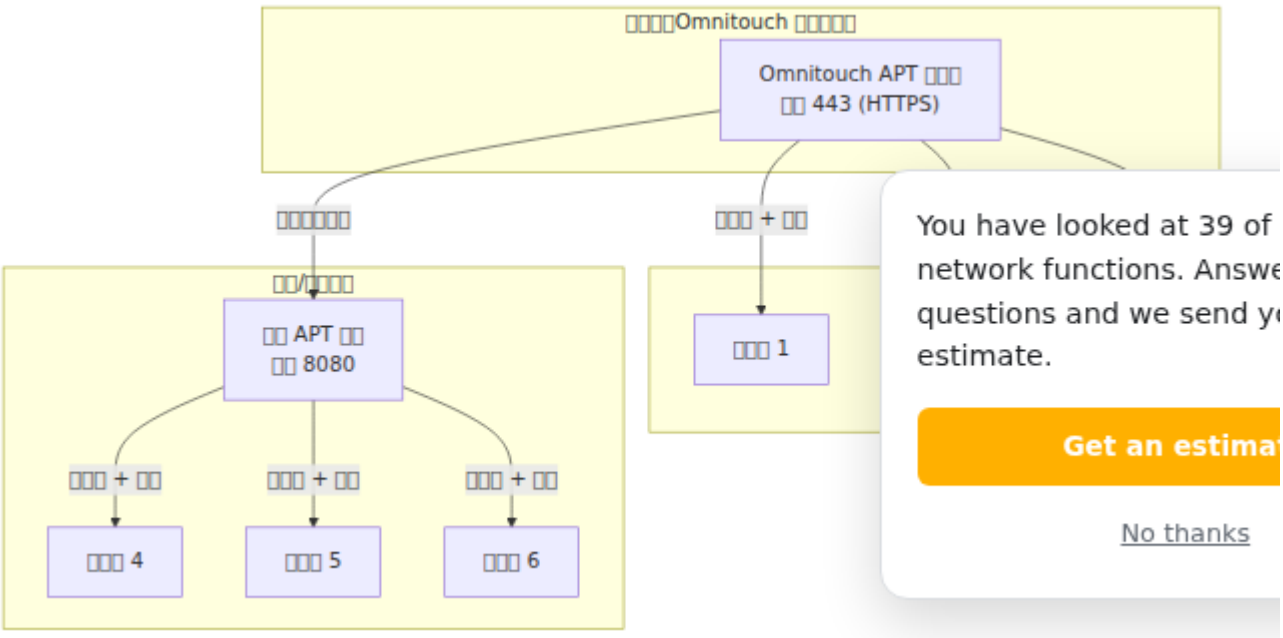
注意：完成后通过运行 `deactivate` 来停用环境。

部署步骤

1. 查看 主机文件配置 以了解如何定义您的网络
2. 了解 组变量 以进行自定义
3. 理解 APT 缓存系统 以进行软件包管理
4. 查看 部署架构 以了解所有内容如何结合在一起
5. 部署！

下一步

- IP 规划标准 - 规划您的网络架构和 IP 分配
- 主机文件配置 - 了解如何定义您的网络拓扑
- APT 缓存系统 - 理解软件包分发
- 许可证服务器 - 了解许可证管理
- 部署架构概述 - 查看完整图景
- 组变量配置 - 自定��您的部署
- 实用剧本 - 用于健康检查、备份和维护的操作工具



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（位于 `/pool/main/o/omnicore/`），其中包含完整的 Ansible 自动化工具包，供希望自行部署和管理其网络的客户使用。

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

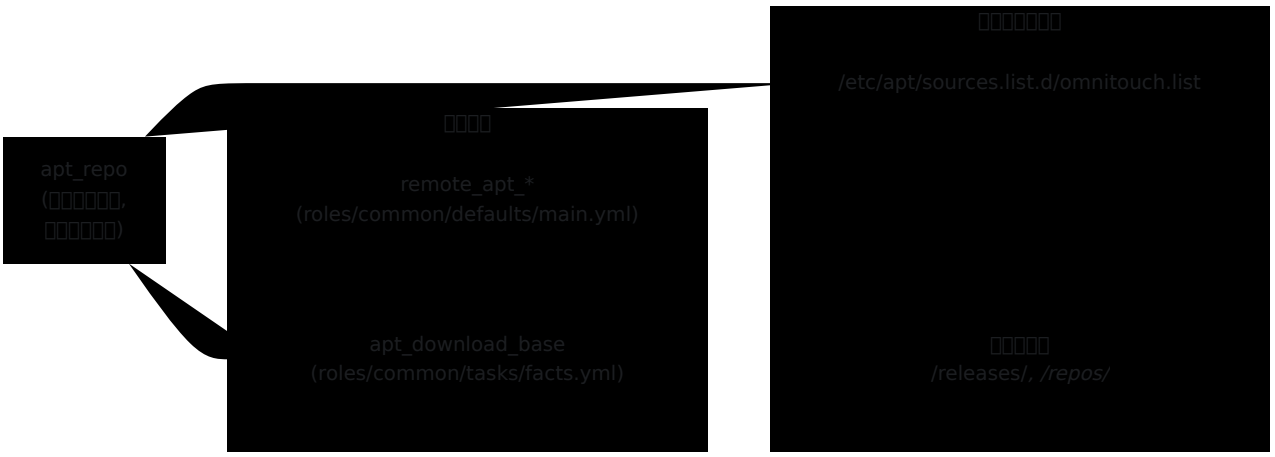
提供的内容

APT 服务器托管所有部署所需的内容：

内容类型	描述	路径
Omnitouch 软件包	自定义构建的 `.deb` 软件包（omnihss、omnimme 等）	`/dists/<distro>/`
OmniCore 工具包	`omnicore` 软件包：完整的 Ansible 自动化（角色、剧本、依赖）以供自助部署	`/pool/main/o/omnicore/`
Ubuntu 软件包	带有所有依赖项的缓存 Ubuntu 软件包	`/<distro>/pool/main/`
GitHub 发布	预构建的二进制文件（Prometheus、Grafana、VoIPmonitor 等）	`/releases/<org>/<repo>/`
源代码压缩包	Web 应用程序的源代码档案（CGrateS_UI、speedtest）	`/repos/`
第三方软件包	Galera、FRR、InfluxDB、KeyDB 等	`/releases/<vendor>/`

配置变量

`apt_repo` 是软件包分发的单一真实来源。在您的清单中定义一次。其他所有内容均由此派生。您不需要单□配置二进制下载 URL、方案、端口或凭据。



派生工作原理

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` 缺失时的最终后备：`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 源行和二制下载始终在主机、方案、端口和凭据上保持一致。

由于这两个派生值都是从同一个 `apt_repo` 块构建的，正常清单 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仍然显式设置 `remote_apt_*` 的地方。它自己的 `apt_repo` 指向 □□□□。然后 `remote_apt_*` 值描述它从中同步的 □□ Omnitouch 服务器（请参见 选项 2）。

方案和端口

方案会自动跟随端口：端口 `443` → `https`，任何其他端口 → `http`。当省略 `apt_repo.apt_repo_port` 时：

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

每种模式的使用时机

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。在设置期间，将您的源子网提供给 Omnitouch。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	许可证验证的 NTP 时间同步
许可证服务器	53	TCP/UDP	许可证验证的 DNS 解析

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

配置

```yaml
all:
  vars:
    use_apt_cache: false

    # 单一真实来源。二进制下载配置（remote_apt_*）和
    # apt_download_base URL 自动从此派生 - 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。请勿在直接访问清单中设置它们。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	后备
`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`

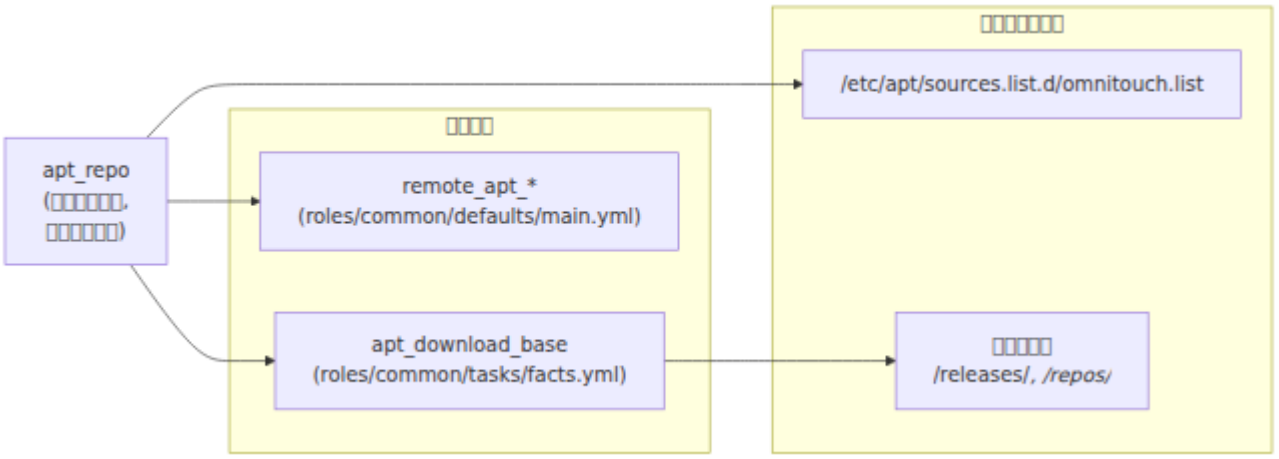
URL 模式（直接访问）

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
```



> URL 中省略了端口 `443`（这是 HTTPS 默认值）。非标准端口作为 `:<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���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。

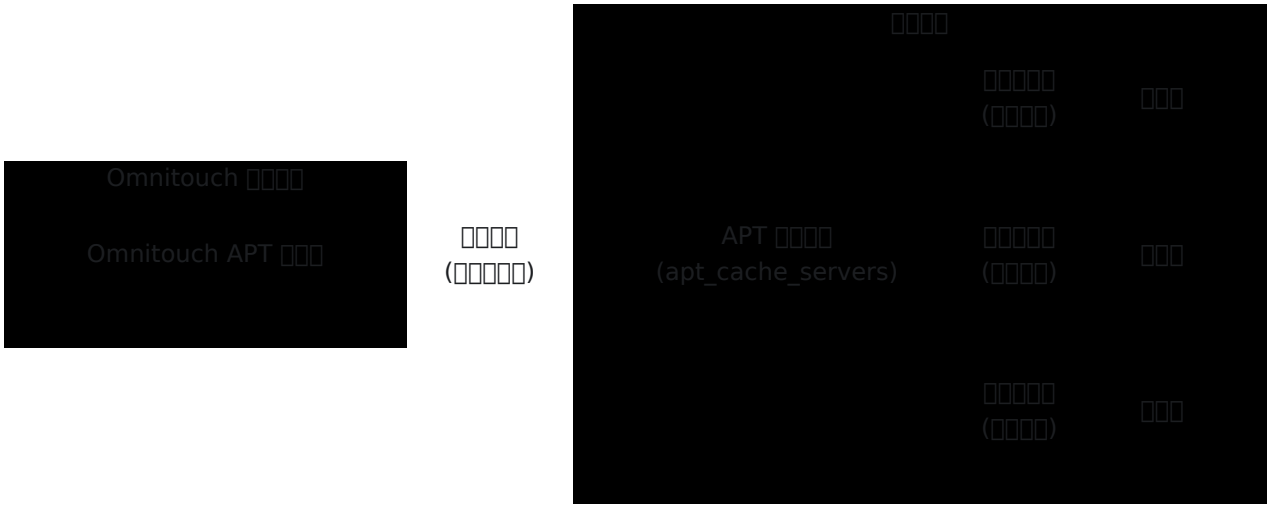
软件包签名（**GPG**）

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；无需手动管理密钥。

> 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从中同步的 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"`，并在没有凭据的情况下从缓存中拉取，端口为 `8080`。

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。（替换已弃用的 `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. 配置缓存服务器（虚拟机或具有 50+ GB 磁盘的 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。不同类型内容的机制有所不同：

- **APT** 软件包（`/dists/` + `/pool/`，包括 Omnitouch 和 Ubuntu）通过 `curl` 通过 Debian 仓库索引同步。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，并仅下载缺失或其 SHA256 与本地副本不匹配的文件。每个下��的文件在提交之前都会进行 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，因此回退到 `wget --recursive --timestamping`，仅重新下载具有更新的 `Last-Modified` 上游的文件。

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）

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

缓存镜像使用 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`）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/` 则回退到 `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）都获得 `use_apt_cache: true`
- 所有主机（缓存服务器除外）都获得 `apt_repo.apt_server: "192.168.1.100"`
- 所有主机在没有凭据的情况下从 `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���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"
```

配置摘要

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_*` 值从中派生。

```yaml
all:
  vars:
    use_apt_cache: false

    # APT 源和二进制下载的单一真实来源
    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：在主机文件中定义 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 中不需要任何配置:
# 一切都从 apt_cache_servers 组自动派生
```

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: 远程 APT 缓存不在主机文件中 (手动)

缓存服务器存在于其他地方并且已经设置好 (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           # 缓存通常运行在 8080 端口

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

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。它展示了多个应用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（当 apt_cache_servers 存在时自动启用）
    # - apt_repo.apt_server: "10.179.1.114"（自动从缓存服务器推导）
```

部署期间发生的事情：

1. 缓存服务器（**`10.179.1.114`**）：
   - 使用其 `vars:` 部分中的 `remote_apt_*`
   - 从 `https://customer-username:customer-secure-token@apt.<region>.omnitou.ch/` 下载所有软件包
   - 通过 nginx 在 8080 端口提供软件包
2. 应用主机（**`customer-hss01`**、**`customer-mme01`**、**`customer-dns01`**）：
   - 自动检测到 `apt_cache_servers` 组存在
   - 自动设置 `use_apt_cache: true`
   - 自动推导 `apt_repo.apt_server: "10.179.1.114"`
   - 生成：`deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
   - 从缓存服务器拉取所有软件包（无需凭据）

更新缓存

要同步新软件包或更新：

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

这将增量同步来自 Omnitouch APT 服务器的所有内容：

- 新的 Omnitouch 软件包版本
- 新的 Ubuntu 软件包和依赖项
- 新的 GitHub 发布
- 更新的源 tarball

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（如果上游的 `Release` 文件未更改，则完全跳过），而 `/releases/` 和 `/repos/` 使用 `wget --timestamping`。现有未更改的文件被跳过，使重新同步快速。

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此；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，而不是在 `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` 在 `apt_cache_servers: vars:` 中定义，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（8080 端口），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 已在 Omnitouch 上列入白名单（如果使用直接访问）

二进制下载失败（Node Exporter、Zabbix 等）

症状：Ansible playbook 在下载 `/releases/` 路径中的文件时失败

可能的原因：

- `apt_repo` 凭据不正确（二进制下载从直接模式的 `apt_repo` 中获取其身份验证）
- 在缓存服务器上，`remote_apt_user` / `remote_apt_password` 不正确
- 错误的方案或端口。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. 确认推导的 `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 playbook 无法下载软件包

可能的原因：

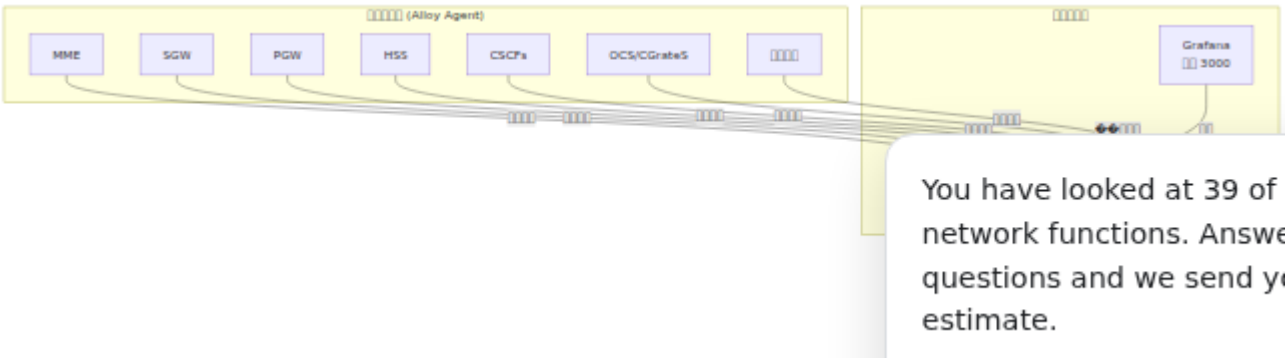
- 缓存服务器没有互联网访问
- `remote_apt_*` 凭据不正确
- 防火墙阻止与 Omnitouch 的出站连接

解决方案：

1. 验证缓存服务器可以在 443 端口访问上游 Omnitouch APT 服务器
2. 检查 `remote_apt_*` 凭据
3. 审查出站访问的防火墙规则

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



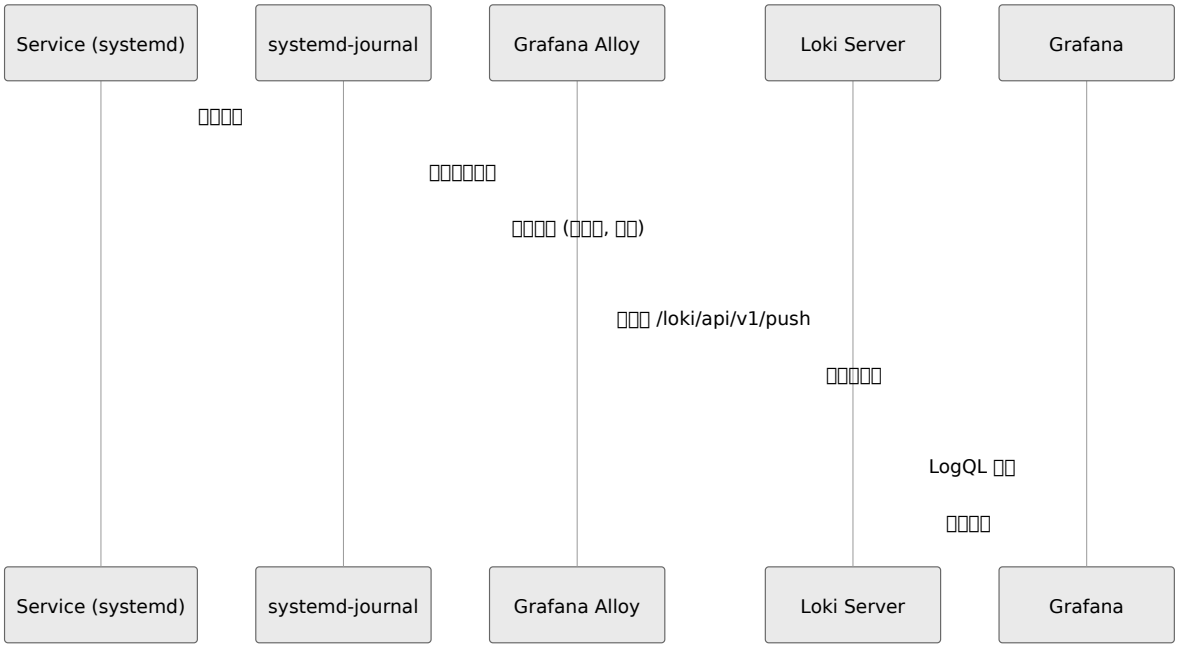
集中式日志记录

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 使其可搜索。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

Service (systemd)
systemd-journal
Grafana Alloy
Loki Server
Grafana
写入日志
读取日志条目
添加标签（主机名，组件）
推送到 /loki/api/v1/push
索引和存储
LogQL 查询
返回结果
Service (systemd)
systemd-journal
Grafana Alloy
Loki Server
Grafana

日志标签

每个日志条目包括以下标签以供过滤和查询：

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`

配置

自动部署

当满足以下条件时，日志记录会自动配置：

1. 在您的库存中存在 `monitoring` 组
2. 通用角色在���标主机上运行

基本功能不需要额外配置。

库存要求

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

当定义 `monitoring` 组时：

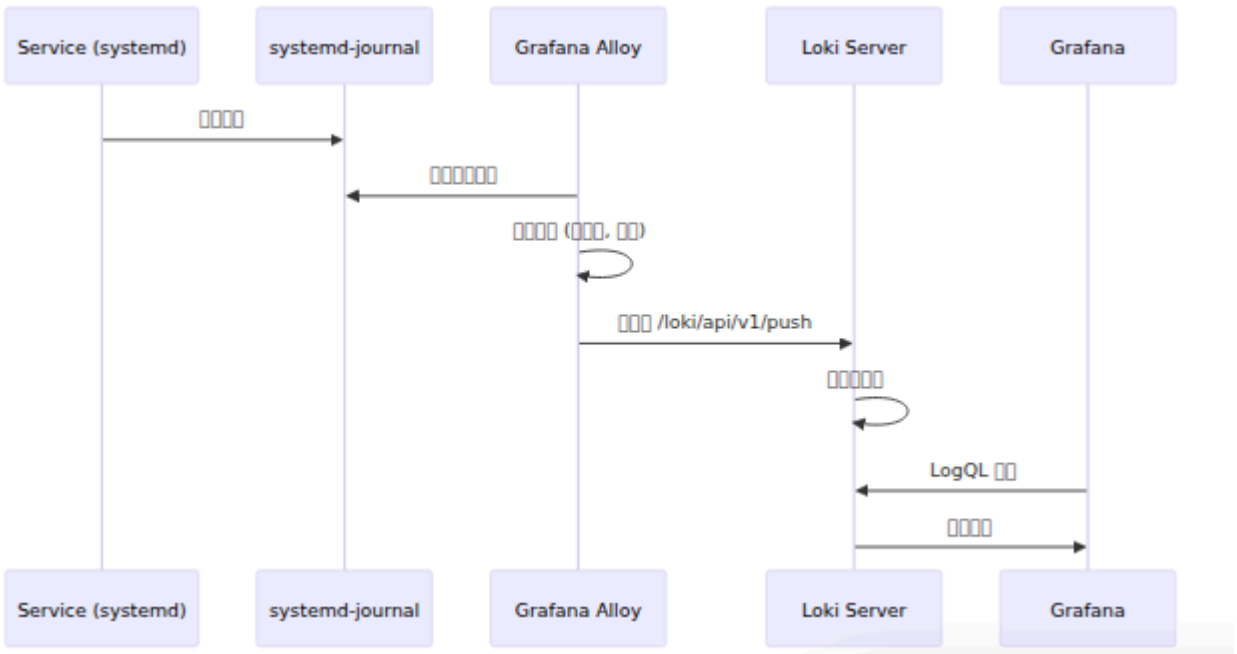
- 监控主机：运行 Loki 服务器，接收并存储日志。它们还运行 Alloy，收集主机自身的日志并终止 网络设备 syslog 流。
- 所有其他主机：运行 Alloy 代理，收集日志并将其发送到 Loki。

保留配置

Loki 配置为基于时间的保留。压缩器会删除超过保留期的日志：

设置	默认值	描述
基于时间的保留	72 小时（3 天）	超过保留期的日志将被删除

要自定义保留，请在您的库存中覆盖此变量：



```yaml
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。为了防止磁盘无限增长，缓冲段会被逐出：

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。

网络设备 Syslog

路由器、交换机和其他设备无法运行 Alloy。这些设备将 BSD syslog 发送到监控主机。Alloy 接收该流并将其写入相同的 Loki，标签为 `job="network_syslog"`。设备日志然后共享主机日志的搜索、时间范围和保留。

该监听器是标准监控构建的一部分。站点不设置标志以接收设备 syslog。Alloy 根据数据报的源地址为每个设备命名，站点库存在供应商 `devices:` 映射中持有。

有关收集器参数、RouterOS 和其他发送方的设备配置、标签引用和故障排除步骤，请参见 网络设备 Syslog。

Grafana 仪表板

为每种组件类型自动提供预构建的仪表板。

可用仪表板

仪表板	位置	描述
CSCF 日志	日志 → CSCF 日志	P-CSCF, S-CSCF, I-CSCF (OmniCSCF) 日志
MME 日志	日志 → MME 日志	MME 附加/分离事件和错误
SGW 日志	日志 ��� SGW 日志	SGW 会话和承载事件
PGW 日志	日志 → PGW 日志	PGW PDN 和会话事件
HSS 日志	日志 → HSS 日志	HSS Diameter 消息和认证事件
OmniMessage 日志	日志 → OmniMessage 日志	短信投递和 SMPP 事件
OCS/CGrateS 日志	日志 → OCS/CGrateS 日志	计费事件和 JSONRPC 流量
CGrateS RPC 调用	日志 → CGrateS RPC 调用	搜索和关联 RPC 请求/响应及延迟
网络设备 Syslog	日志 → 网络设备 Syslog	MikroTik / 网络设备 syslog，按设备和严重性

仪表板功能

每个仪表板包括：

- 搜索框：在所有日志中进行自由文本搜索
- 日志速率图：按主机和严重性显示的时间段内的日志量
- 组件部分：分组视图（例如，P-CSCF, S-CSCF, I-CSCF）
- 错误过滤：预过滤的错误和故障视图
- 实时尾部：实时日志流（10 秒刷新）

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** 中的错误：

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

按 **systemd** 单元过滤：

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

组合过滤器：

```
{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]))
```

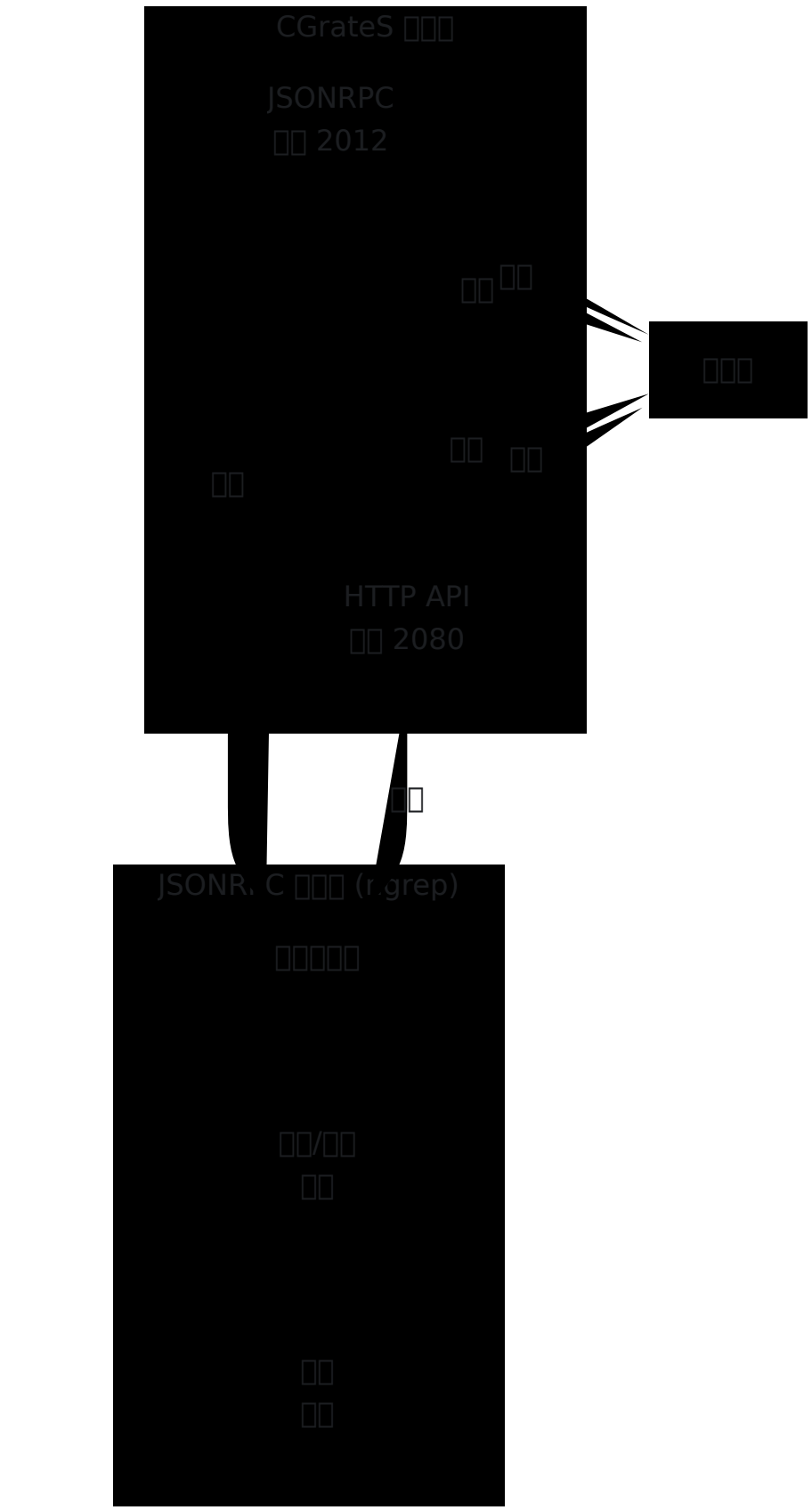
CSCF 的错误率：

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

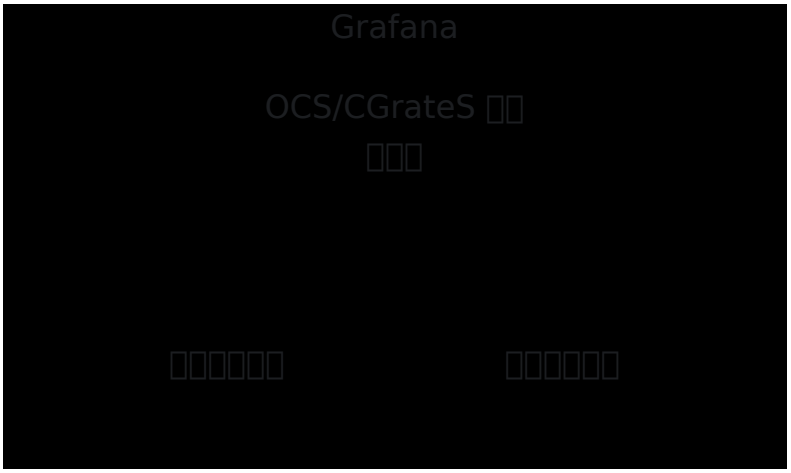
CGrateS JSONRPC 日志记录

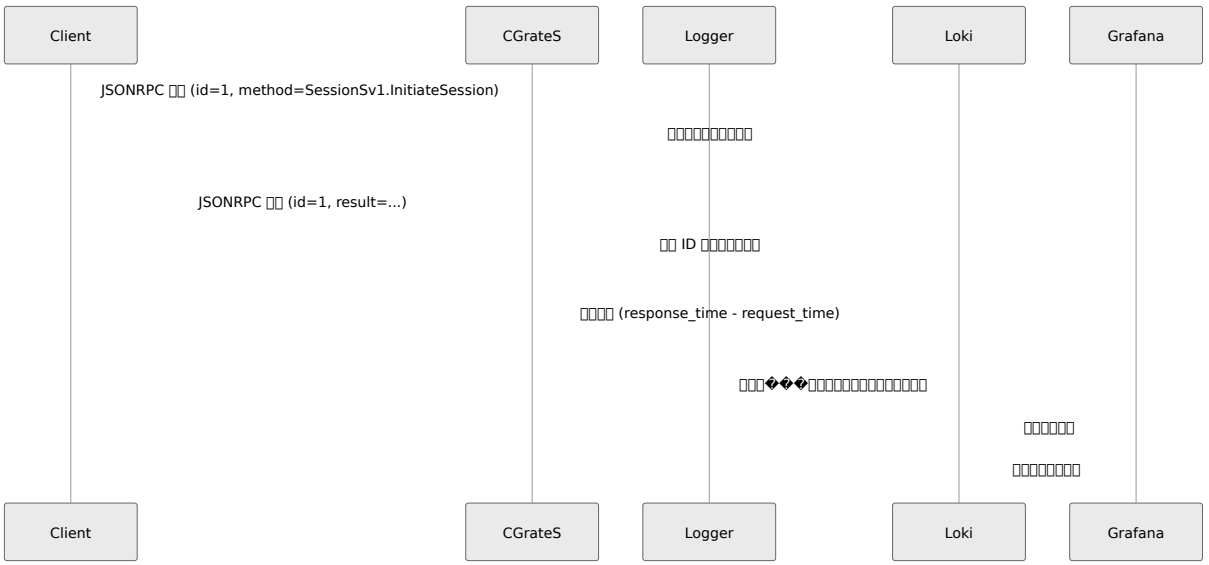
对于 OCS/CGrateS 部署，捕获 JSONRPC 流量并记录请求/响应关联和延迟跟踪，以进行性能分析和调试。

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


CGrateS □□□
JSONRPC
□□ 2012
□□ □□
□□□
□□ □□
□□
HTTP API
□□ 2080
□□
JSONR. C □□□ (r.grep)
□□□□□
□□/□□
□□
□□
□□
□□□□
/var/log/cgrates/jsonrpc.log







Grafana Alloy
Loki

Grafana
OCS/CGrateS □□
□□□
□□□□□□
□□□□□□

请求/响应流程

Client
CGrateS
Logger
Loki
Grafana
JSONRPC 请求 (id=1, method=SessionSv1.InitiateSession)
捕获请求，存储时间戳
JSONRPC 响应 (id=1, result=...)
通过 ID 匹配响应到请求
计算延迟 (response_time - request_time)
日志条���，包含方法、延迟、请求、响应
查询延迟指标
返回时间序列数据
Client
CGrateS
Logger
Loki
Grafana

捕获的内容

这些 CGrateS 端口上的流量被捕获:

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`。每个日志条目表示一个 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`	接收到响应时的 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 以进行关联	`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 仪表板包含用于 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：

- 可能需要优化的慢方法
- 表示系统问题的延迟峰值
- 随时间变化的性能趋势

图例显示每个方法的平均和最大延迟。

JSONRPC 流量

显示所有 JSONRPC 请求/响应对的日志面板，包括：

- 时间戳
- 方法名称
- 延迟
- 完整的请求和响应负载（美化）

方法深入分析

使用仪表板顶部的 方法 变量按特定方法过滤 JSONRPC 流量。输入方法名称（例如，`SessionSv1.InitiateSession`）以仅查看该方法的流量。

CGrateS RPC 调用仪表板

专用的 **CGrateS RPC** 调用 仪表板提供了专注于故障排除 API 调用和关联请求与响应的工具。

用例：查找失败的调用

当用户报告“我尝试拨打 1234，但失败了”时：

1. 打开 仪表板 → 日志 → **CGrateS RPC** 调用
2. 在 搜索 字段中输入电话号码或账户 ID：`1234`
3. 将 状态 设置为 错误 以仅查看失败的调用
4. **RPC** 调用搜索 面板显示所有匹配的调用及完整的请求/响应

每个日志条目显示完整的请求负载（发送的内容）和响应负载（CGrateS 返回的内容），使识别确切错误变得容易。

仪表板变量

变量	目的	示例
搜索	在请求/响应负载中进行自由文本搜索	`1234`, `Account123`, `INSUFFICIENT_CREDIT`
方法	按 RPC 方法名称过滤	`SessionSv1.InitiateSession`, `CDRsV1.ProcessEvent`
状态	按结果状态过滤	全部, 成功, 错误

仪表板面板

摘要统计（顶部行）：

- 总 **RPC** 调用：时间范围内所有调用的计数
- 错误：失败调用的计数（颜色编码：绿色=0，黄色=1-9，红色=10+）
- 平均延迟：平均往返时间（颜色编码阈值）
- 最大延迟：最高延迟调用

按方法的 **RPC** 延迟：时间序列显示每个方法的延迟。悬停以查看特定值。

RPC 调用搜索：主日志面板显示匹配的调用。单击任何条目以展开并查看完整的请求/响应 JSON。

失败的 **RPC** 调用：预过滤视图，仅显示 `status=error` 的调用。

方法分解：饼图显示按方法的调用分布和错误分布。

查询 **JSONRPC** 日志

基本查询

所有 **JSONRPC** 流量：

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

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��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 是否在收集该文件：

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

延迟图表显示无数据

症状：JSONRPC 按方法的延迟面板显示“无数据”

可能原因：

- 在所选时间范围内没有 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，但记录的完成很少

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 是否响应请求

文件日志

除了 systemd 日志外，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 日志（OmniCRM）

路径	作业标签	类型标签
`/var/log/nginx/access.log`	`nginx`	`access`
`/var/log/nginx/error.log`	`nginx`	`error`

CGrateS JSONRPC（OCS）

路径	作业标签
`/var/log/cgrates/jsonrpc.log`	`cgrates-jsonrpc`

故障排除

日志未在 Grafana 中出现

症状：在 Loki 仪表板中没有可见日志

可能原因：

- 源主机上未运行 Alloy 服务
- 监控主机上未运行 Loki 服务
- 主机之间的网络连接被阻止
- Alloy 无法通过端口 3100 访问 Loki

解决方案：

1. 检查源主机上的 Alloy 状态：

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

2. 检查监控主机上的 Loki 状态：

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

3. 测试从源到监控的连接：

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

4. 验证防火墙允许所有主机到监控的 TCP 3100

Alloy 磁盘使用率高

症状：`/var/lib/alloy` 消耗了大量磁盘空间

可能原因：

- Loki 在较长时间内无法访问
- WAL 缓冲区填满

解决方案：

1. 检查 Loki 连接性（见上文）
2. 如果 Loki 关闭，日志将本地缓冲在 WAL 中
3. 一旦 Loki 可访问，缓冲日志将自动发送
4. 超过 1 小时的缓冲段将被丢弃（按设计）

Loki 存储已满

症状：Loki 停止接收日志，监控服务器磁盘已满

可能原因：

- 日志量在保留删除旧日志之前超过可用磁盘
- 保留压缩未运行

解决方案：

1. 检查 Loki 存储使用情况：

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

2. 验证压缩器是否在运行（检查 Loki 日志）
3. 如有需要，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

缺少组件标签

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

电路交换□心 (CS)

传统的电路交换域 (2G/3G 语音和短信) 由 `services/cs.yml` 剧本部署。它配置了面向无线的 BSC、□立的媒体网关、OmniMSC 软交换机，以及 SS7/SIGTRAN 信令栈。

概述

`cs.yml` 运行这些角色，每个角色对应其自己的库存组：

组件	角色	库存组	软件包	描述
BSC	`circuit_switched/bsc`	`bsc`	BSC, MGW	基站控制器 + 媒体网关 (用于 2G/3G 测试和验证)
MGW	`circuit_switched/mgw`	`mgw` (不在 `bsc` 中的主机)	`osmo-mgw`	适用于没有 BSC 的站点的□立媒体网关
MSC	`circuit_switched/omnimsc`	`msc`	`omnimsc`	移动交换中心软交换机
SS7	`omniss7`	`ss7`	`omniss7`	SS7/SIGTRAN 栈 (STP, HLR, SMSC, CAMEL, 测试器)

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

BSC (`circuit_switched/bsc`)

从 Omnitouch APT 服务器安装 **BSC** 和 媒体网关 **(MGW)** 以进行电路交换测试和验证。该角色配置这两项服务。

- 从角色的模板渲染 BSC 和 MGW 配置文件。
- 为 MGW 服务安装 systemd 覆盖，禁用实时 **CPU** 调度。它清除 `CPUSchedulingPolicy`、`Priority` 和 `AmbientCapabilities`。虚拟机不支持实时调度。
- 处理程序重新启动每个服务。每个处理程序然后 验证服务是否达到 **`active`** (6 次重试，每次间隔 5 秒) 然后继续。

MGW (`circuit_switched/mgw`)

从 Omnitouch APT 服务器安装独立的 媒体网关 (`osmo-mgw`)。在有 MSC 但没有 BSC 的站点使用。MSC 在 MGCP 上将每个 CS 呼叫的媒体锚定在 MGW 上。

- 渲染 MGW 配置。MGCP 绑定到 `mgw_bind_ip` 上的 `mgw_port`。RTP 绑定到 `mgw_rtp_bind_ip` 并使用 `mgw_rtp_port_range`。两个地址默认为主机的 `ansible_host`。端口默认为 2427，范围为 `4002 16001`。
- 安装与 BSC 角色相同的 systemd 覆盖。该覆盖禁用实时 CPU 调度，虚拟机和 LXC 不支持此功能。
- 处理程序重新启动服务，然后运行 `verify_service` 崩溃循环检查。
- 也在 `bsc` 组中的主机已经从 BSC 角色运行 MGW。剧本在这样的主机上跳过此角色。

MSC (`circuit_switched/omnimsc`)

安装 `omnimsc` 软件包，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`。
- 将内置的 `runtime.exs.j2` 渲染到 `/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`，默认为 `/var/lib/omnimsc/cdr`)。
- 为 Web UI / API 生成自签名的 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。

- 按以下顺序构建 MGCP 网关列表：

  i. `omnimsc_mgcp_gateways`，如果设置（显式覆盖）。
  ii. 每个 `mgw` 组中的主机一个网关（地址 `ansible_host`，端口 `mgw_port`，默认为 2427）。
  iii. 每个 `bsc` 组中的主机一个网关（BSC 角色共置一个 MGW）。

  如果这些都不存在，角色在安装任何内容之前失败。使用的媒体网关是 `omnimsc_media_gateway`（默认为 `MGW-01`，列表中的第一个网关）。

SS7（`omniss7`）

一个单一角色，安装 `omniss7` 软件包并为每个 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`（在 `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/*`。

CAMEL 计费

具有 CAMEL 数据的订阅者的 CS 呼叫通过 CAMEL 网关计费，通过 TAS 用于 Ro 的相同 CGrateS OCS 和租户（`ocs_tenant`）：

主机变量	主机	功能
`camel_gsmscf_global_title`	`ss7_role: camel`	CAMEL 网关的 gsmSCF 全局标题。`stp.j2` 为其添加 GT 路由，而 `camel.j2` 将其用作 CAP 应答的呼叫 GT。
`hlr_camel_subscriptions`	`ss7_role: hlr`	一组带有 `o_csi` 和/或 `t_csi` 的 `imsi_prefix` 规则（`service_key`、`gsmscf_address`、`default_call_handling`：`release_call` 或 `continue_call`），以及一个可选的 `mo_sms_csi`（`service_key`、`gsmscf_address`、`default_sms_handling`：`release_transaction` 或 `continue_transaction`）。HLR 将它们发送到 VLR 的 InsertSubscriberData。MSC 为呼叫发送 InitialDP，或为 MO SMS 发送 InitialDPSMS，至 `gsmscf_address`。

`camel.j2` 将网关指向第一个 `ocs` 主机（`:2080/jsonrpc`，防火墙流 31b）。它以 10 秒为单位计费呼叫，正如 TAS 所做，并将每个 MO SMS 计费为一个 `*sms` 单位。当两个变量都设置时，ran_simulator 角色设置 `camel_origin_host` 场景变量，从而启用 CAMEL 计费预设。

USSD（`support/ussd_app`）

移动发起的 USSD 从 MSC 通过 MAP `processUnstructuredSS-Request` 发送到 HLR。当库存中有 `ussd` 组时，`hlr.j2` 启用 omniss7 USSD 网关，并设置一个通用路由（`*`）到 `http://<first ussd host>:8380/ussd`。没有该组时，网关未配置。

`support/ussd_app` 角色（剧本 `services/ussd.yml`）在 `ussd` 主机上安装一个小的回调应用，作为 `ussd-app` systemd 服务。它为每个请求记录一行到 journald。

拨号 / 回复	答复
`*100#`	结束：`您的余额为 $12.34`
`*102*<digits>#`	结束：`您发送了 <digits>`
`*101#`	继续：`1. 余额` / `2. 数据` / `3. 退出`
回复 `1`、`2`、`3`	结束：`余额：$12.34`、`数据：剩余 1.5 GB`、`再见`
其他任何回复	结束：`无效选项`
其他任何代码	结束：`未知代码 <code>`

端口（`ussd_app_port`，8380）属于该角色。`roles/omniss7` ���角色默认值中读取它，防火墙在角色的 `firewall_services.yml` 中声明它。剧本将上述每种情况发布到应用程序，并在答案错误时失败。

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，一些组件仅从 `group_vars` 配置，并且没有角色默认值。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，仅供实验室使用，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、唯一的机密。

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                      # 排除在负载均衡池之外
dra_peer_exclude: True              # 从生成的 Diameter 对等体列表中排除
gtpc_weight: 100                    # SGW/PGW 在其优先级层中的份额（MME 对等体选择）
gtpc_priority: 100                  # SGW/PGW 优先级层，较低者优先
```

有关 `dra_peer_exclude` 的详细信息，请参见 Diameter 对等体排除。

`gtpc_weight` 和 `gtpc_priority` 在 MME 构建其静态 GTP-C 对等体列表时从每个 `sgwc` / `pgwc` 主机读取。两者默认为 `100`，因此一个对□□体可以在不更改其他主机的情况下在池中加权或分层。MME 选择数值上 □□ 的 `gtpc_priority` 层，并按 `gtpc_weight` 在其中分配流量（RFC 2782）；更高的层处于待命状态，仅在没有对等体留在较低层时才接收流量。相对 MME 容量以相同方式设置，使用 MME 主机上的 `relative_capacity`（默认 `100`，范围 0-255）。

全局变量 (all:vars)

`all:vars` 部分包含部署范围的设置。现代部署使用最小的全局变量，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，因为没有角色默认值。

必要的全局变量

认证与访问

```
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

```
customer_name_short: omnitouch
customer_legal_name: "客户实验室"
site_name: 客户站点
region: AU
TZ: Australia/Melbourne
```

PLMN 配置

```
plmn_id:
  mcc: '001'                  # 移动国家代码（3 位□字）
  mnc: '01'                   # 移动网络代码（2-3 位□字）
  mnc_longform: '001'         # ��填充的 MNC（始��� 3 位□字）

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

目的：唯一标识您的移动网络。用于 Diameter 领域构建。

网络名称

```
network_name_short: Omni
network_name_long: Omnitouch
tac_list: [10100,100]              # 默认 TAC 列表（可以按 MME 覆盖）
```

显示：在 UE 设备的设置 > 移动网络中显示的网络名称。

DNS 配置

```
manage_resolv_conf: True           # 设置为 False 以防止 Ansible 管理
/etc/resolv.conf
```

注意：当 `manage_resolv_conf` 设置为 `False` 时，Ansible 不会覆盖该主机上的 `/etc/resolv.conf`。这对于需要自定义 DNS 配置或由外部系统管理的主机非常有用。可以在清单中按主机设置或在 `all:vars` 中全局设置。

APT 存储库配置

`apt_repo` 是唯一真实来源。只需定义一次。二进制下载变量（`remote_apt_*`）和 `apt_download_base` URL 会自动从中派生。在正常清单中，您不需要单□设置它们。

自动默认值：当定义了带有主机的 `apt_cache_servers` 组时：

- `use_apt_cache` 自动默认为 `True`（除非显式设置为 `False`）
- `apt_repo.apt_server` 自动默认为第一个缓存服务器的 IP

```
use_apt_cache: False                # 直接访问存储库（True = 本地 APT 缓存）

apt_repo:
  apt_server: "apt.<region>.omnitou.ch"  # Omnitouch 提供的 FQDN（或缓存 IP）
  apt_repo_username: "your-username"     # 直接访问所需
  apt_repo_password: "your-password"     # 直接访问所需
  # apt_repo_port: 443                   # 可选；默认为 443（HTTPS）。
                                         # 方案遵循端口：443 -> https，其他为 http。
```

方案和端口默认为 **HTTPS / 443** 以进行直接访问，在缓存模式下为 **HTTP / 8080**。只有缓存镜像服务器（`apt_cache_servers` 组）手动设置 `remote_apt_*`，以描述其同步的上游 Omnitouch 服务器。

请参见：APT 缓存系统

许可证服务器

```
# 可选：在未设置时从 `license_server` 清单组自动派生。
# 该组中的每个主机都变为 https://<ansible_host>:8443/api。
license_server_api_urls: ["https://10.10.2.150:8443/api"]
```

注意：如果清单中存在 `license_server` 组，则 `license_server_api_urls` 会自动从其成员派生；您只需手动设置它以覆盖该行为。

请参见：许可证服务器

MME 设置

MME 使用可配置的方法选择 S-GW 和 P-GW 对等体，该方法在 `omnimme` 下为每个网关设置。该方法可以是单个值或按 `home`/`roaming` 流量拆分。有效值为 `dns_lookup`（通过 DNS 解析对等体）和 `random_peer`（从配置的对等体中选择）。

```
omnimme:
  sgw:
    selection_method: random_peer          # 单个值，或...
    # selection_method:
    #   home: random_peer
    #   roaming: dns_lookup
  pgw:
    selection_method:
      home: random_peer
      roaming: dns_lookup
```

AMF 设置

```
# tac_list 还驱动 AMF 的 supported_ta_list。这里的每个 TAC 都会在 NG 设置期间向连接的 gNB 广播。如果未设置，AMF 默认为仅 TAC 1，这会拒绝任何未广播 TAC 1 的 gNB。
tac_list: [1, 3]

# 禁用 NAS 加密（强制 NEA0）。默认为 true，因为 omniamf >=20260727 协商 NEA2，但无法解码加密的上行 NAS（安全模式完成和后续消息），导致每个 5G 注册失败。一旦供应商确认上行 NEA2 解密在真实 UE 上有效，请设置为 false。NAS 完整性（NIA2）不受影响。
amf_force_null_ciphering: true
```

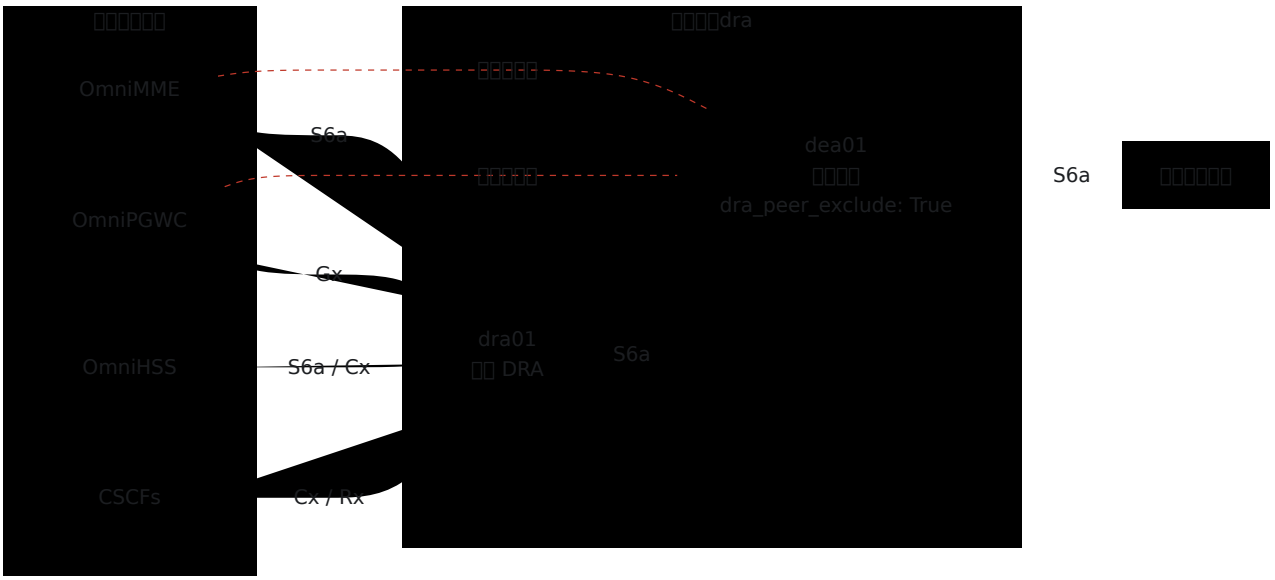
SAEGW 设置

```
mtu: 1400                                  # 最大传输单元
```

OmniHSS 设置

OmniHSS 自动从清单中构建其 Diameter 对等体列表。它使用 `dra`、`mme`、`pgwc`、`scscf`、`icscf`、`pcscf` 和 `applicationserver` 组（使用哪些取决于 `packet_core_dra_support` /

`ims_dra_support`□□`exclude_diameter_peers` □□□□



配置

在主机条目本身上设置标志，紧挨着 `ansible_host`。

```yaml
dra:
  hosts:
    dra01:                    # 内部 DRA。每个内部网络功能的对等体。
      ansible_host: 10.4.2.150
    dea01:                    # 边缘代理。永远不是内部网络功能的对等体。
      ansible_host: 203.0.113.10
      packet_core_dra_support: False
      ims_dra_support: False
      dra_peer_exclude: True
```

参数	类型	必需	默认	描述
`ansible_host`	字符串	是	-	主机的管理 IP 地址。也是生成的对等体列表中的对等体 IP 地址。
`dra_peer_exclude`	布尔值	否	`False`	当为 `True` 时，其他每个主机在从清单组构建 Diameter 对等体列表时跳过此主机。
`packet_core_dra_support`	布尔值	否	`False`	当为 `True` 时，主机与 DRA 进行数据包核心接[（S6a、Gx）的对等体。当为 `False` 时，主[直接与 HSS 和 PCRF 进行对等体。在 DEA 上，将其设置为 `False`，因为 DEA 从 `dra_module_advanced_routing` 获取路由。
`ims_dra_support`	布尔值	否	`False`	当为 `True` 时，主机与 DRA 进行 IMS 接口（Cx、Rx、Sh）的对等体。在 DEA 上，将其[置为 `False`。

范围

该标志在两个方向上工作。网络功能跳过 `dra` 组的排除机，DRA 跳过任何其他组的排除主机。

过滤 **`dra`** 组的网络功能：

角色	对等体列表
`omnimme`	S6a 和 T6a 对等体
`omnipgwc`	Gx 和 Gy 对等体
`omnihss`	S6a、Cx 和 Sh 对等体
`omniepdg`	SWm 和 S6b 对等体
`omnitwag`	S6b 对等体
`omniscef`	T6a 对等体
`omnisep`	Sh 对等体
`omnilcs`	SLg 和 SLh 对等体
`cscf`	P-CSCF、I-CSCF 和 S-CSCF 的 Cx、Rx 和 Dx 对等体
TAS 和 SMSC（站点 `group_vars`）	Sh 和 Ro 对等体

DRA 过滤的组：`hss`、`mme`、`pgwc`、`epdg`、`scef`、`twag`、`ocs`、`omnilcs`、`omnimessage`、`pcscf`、`icscf`、`scscf`、`xcap` 和 `applicationserver`。

行为规则

- 在要排除的主机上设置标志。不要在构建对等体列表的主机上设置它。
- 该标志仅适用于生成的对等体列表。显式静态对等体永远不会被过滤。这涵盖了 DRA 上的 `dra_peers` 和网络功能上的 `diameter_static_peers`。
- 该标志不会更改防火墙规则、Prometheus 目标或 IP 计划。被排除的主机保持其网络访问和监控。
- 该标志在下次角色运行时生效。在设置标志后，运行每个受影响网络功能的角色。该主机上部署的 `runtime.exs` 在那之前保持旧的对等体。

缺少排除的症状

如果内部网络功能连接到 DEA，请首先检查此标志。症状包括：

- MME 或 PGW-C 打开到 DEA 的公共 IP 地址的 SCTP 连接。
- DEA 记录未授权的对等体连接尝试，因为 `allow_undefined_peers_to_connect` 为 `False`。
- 网络功能报告对等体处于永久重连循环中。

要纠正此问题，请在 DEA 主机上设置 `dra_peer_exclude: True`。然后再次运行网络功能的角色。然后确认部署的 `runtime.exs` 中的对等体列表仅包含内部 DRA。

OmniCRM 设置

OmniCRM 配置仅来源于客户的 `group_vars`（在 `hosts/` 下）；站点机密没有默认值被嵌入角色中。

`omnicrm` 角色从源代码部署 OmniCRMv2：控制器提供 OmniCRMv2 检出的 tarball，主机构建 venv 和 UI，角色将 `crm_config.yaml` 模板化到 `/etc/omnicrm/crm_config.yaml`，同步 `deployment_assets/`，并在 nginx 后面运行 `omnicrm-v2-*` 单元。UI 品牌来自 `ui:` 块（运行时 `GET /api/public/ui-config`）。

```
crm_config.yaml                    # API + ui: 配置（group_vars；必需）
deployment_assets/                 # 可选的徽标/图标，用于
/etc/omnicrm/deployment_assets
```

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`。请参见 `roles/monitoring/templates/grafana/grafana.ini`。

```
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

```
ims_dra_support: False             # 通过 DRA 路由 IMS
sipcapture_client_enabled: False # 将 SIP/RTP 发送到 VoIPmonitor 捕获框
（siptrace 组）
```

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
```

防火墙配置

由 `services/firewall.yml` 应用。完整的参数集、平面区域和监控/强制模式在 每角色防火墙 中有文档；只有 `allowed_oam_subnets` 是强制的。

```yaml
firewall:
  allowed_oam_subnets:              # 管理源（SSH、UI、抓取、管理 API）
    - '10.0.1.0/24'
    - '10.0.0.0/24'
  allowed_ran_subnets: []           # S1AP / NGAP / GTP-U / Abis
  allowed_ue_voice_subnets:         # 外部 UE SIP 进入 P-CSCF
    - '10.0.1.0/24'
  allowed_carrier_voice_subnets:    # 运营商互联 SIP
    - '10.0.1.0/24'
  zones:                            # 平面区域 — 参见 docs/firewall.md
    internal: ['10.0.0.0/8']
  ue_pools: []
```

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:                           # 示例运营商 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..."
```

虚拟机管理程序配置

Proxmox

一个站点作为 VM 或 LXC 容器部署，通过 `deployment_type`（默认为 `vm`）在每个 `proxmoxServers` 条目中选择。站点中的所有条目必须一致；混合会导致验证失败。请参见 Proxmox 部署 以获取完整的操作步骤。

VM 站点

```
proxmoxServers:
  customer-prxmx01:
    # deployment_type 被省略 → 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 LXC 中

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 - 次级 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。

有关详细组件放置、IP地址分配指南和公共IP处理，请参见IP规划标准。

完整部署示例

0. 基础设施准备（可选）

对于Proxmox部署，在配置之前准备虚拟机/LXC:

```
# 在Proxmox上部署虚拟机和/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
```

参见: Proxmox VM/LXC部署

1. 基础设施定义（主机文件）

```yaml
# 定义要部署的内容及其位置
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15

hss:
  hosts:
    customer-hss01:
      ansible_host: 10.10.2.140

# ... 所有其他组件
```

参见：主机文件配置

2. 自定义（group_vars）

`group_vars`文件夹存储在主机、站点或网络级别所需的任何配置覆盖。

例如，一个文件夹保存您的OmniMessage SMSc配置。您的TAS连接的SIP中继在这里。您的所有Diameter路由逻辑也在这里。

OmniCRM也在这里配置。它的设置仅限于`group_vars`。在同步到目标之前，其GitHub源在Ansible控制器上被克隆（`delegate_to: localhost`），使用控制器自己的凭据，因此没有凭据会落到目标主机上。

参见：组变量配置

3. 包分发（APT缓存）

```yaml
# 配置获取软件包的位置
apt_repo:
  apt_server: "10.254.10.223"  # 缓存服务器IP或直接repo服务器
use_apt_cache: false  # true = 使用本地缓存，false = 直接访问repo
```

参见：APT缓存系统

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"]
```

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工具包安装到控制器并运行它。

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，将您的清单保存在`hosts/`下，并直接从检出运行剧本。这是上述部署示例所假定的模型。

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。

- 它安装了什么。该软件包（`name: omnicore`，参见`packaging/nfpm.yaml`）提供Ansible角色、剧本、*vendored*集合、捆绑的Python解释器（`/opt/omnicore/python`）和一个离线wheelhouse到`/opt/omnicore`。后安装步骤（`packaging/postinstall.sh`）从捆绑的解释器+wheelhouse构建一个自包含的venv（`/opt/omnicore/venv`），并通过`/usr/local/bin`将捆绑的`ansible*`符号链接到`PATH`，因此控制器运行确切的`uv.lock`版本的Ansible及其Python库：没有发行版Ansible，没有系统Python，没有网络。只有真正的系统工具（`git`、`rsync`、`openssh-client`）通过apt引入。该软���包是**amd64**，并针对**Ubuntu 22.04+**（glibc ≥ 2.35）。
- 它是如何构建的。`packaging/build.sh`将repo分阶段到`/opt/omnicore`（不包括`hosts/`、`.git/`、虚拟环境和构建文件），从`requirements.yml`中提供Ansible集合，捆绑□立解释器+锁定Python依赖的wheelhouse，运行失败关闭的秘密扫描，然后使用nfpm构建`.deb`。
- 它是如何发布的。`.github/workflows/build-deb.yml`在每次推送到`main`时运行`build.sh`并将`.deb`（加上一个`_metadata.json`侧车）作为GitHub发布发布。apt服务器的`discover_github_debs.py`自动发现该发布资产并将其添加到aptly。无需更改apt角色。

客户流程：

```
# 1. 从apt安装工具包
apt-get install omnicore

# 2. 提供您自己的清单
#     （该软件包不提供清单 - 它位于/opt/omnicore之外）
vi /etc/omnicore/inventory

# 3. 从已安装的工具包运行剧本
cd /opt/omnicore
ansible-playbook -i /etc/omnicore/inventory services/all.yml
```

由于清单位于`/etc/omnicore/inventory`（与已安装的`/opt/omnicore`工具包分开），升级软件包永远不会影响您的基础设施定义或自定义。

相关文档

- Ansible部署简介 - 入门指南
- 服务剧本 - 剧本参考和层次结构
- 主机文件配置 - 定义基础设施
- IP规划标准 - 网络架构和IP分配

- 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
- 集中式日志记录 - Loki和Alloy日志收集

产品文档

有关每个组件配置的详细信息：

- **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
- 主要文档：https://docs.omnitouch.com.au/

按角色防火墙

按角色防火墙为每个主机提供一个 `iptables` 和 `ip6tables` 规则集，该规则集 源自网络流量的单一声明性模型，而不是逐个主机手动编写的规则。一个文件描述了核心中每个允许的流（谁 可以到达哪个角色，使用哪个协议和端口，以及原因）。基于该模型，平台生成两个始终相互一致的内容：

- 强制执行的规则集 应用于每个主机，以及
- 通信/端口矩阵（CSV），供审查和交给上游 防火墙。

规则以两种模式之一运行：监控（记录每个流，不丢弃任何内容）或 强制执行 （默认拒绝）。阻止是故意的和安全的，每次应用都受到主机上自动回滚的保护，因此错误的规则集无法将操作员锁定在外。

目录

- 架构概述
- 流模型
- 源类型
- 服务目录
- 站点配置
- 操作模式
- 跨站点对等体
- 地址解析
- 通信/端口矩阵
- 锁定保护
- IPv6 和双栈
- 日志记录
- 操作注意事项

架构概述

一个单一模型加上每个站点自己的库存供给一个解析器。解析器生成 主机上的规则和端口矩阵，因此文档永远无法描述与实际强制执行的内容不同的内容。

丢弃和审核的流量通过标准可观察性管道记录：

流模型

流模型是 谁可以到达什么 的单一真实来源。它包含两个部分：一组结构默认值（不携带站点地址）和流的列表。流并不直接说明端口号 - 它命名一个由目标角色暴露的 服务（见 服务目录），端口从该角色的声明中查找：

```
- id: '13'                          # 稳定标签，在每个规则的注释中出现
  dst: hss                          # 规则应用于的库存组
  service: diameter                 # 由 hss 角色暴露的命名服务 (->
3868/tcp+sctp)
  purpose: 'Diameter'               # 描述，在矩阵中显示
  sources: [{ zone: internal }]     # 谁可以到达它（源令牌列表）
```

参数	类型	必需	默认	描述
`id`	字符串	是	-	流的稳定标识符。在每个生成的规则的注释中作为 `ans:<dst>:<id>` 出现，因此可以追溯到模型的实时规则。
`dst`	字符串	是	-	规则落入的库存组，或伪目标 `all`（每个主机）和 `five_g_sbi`（5G 网络功能组）。目标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*	-	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，因此流不会重申端口号。
`proto_ports`	列表	是*	-	`service` 的字面替代，用`���`不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。在端口矩阵的 `service` 列中显示。
`sources`	列表	是	-	允许的源，作为源令牌的列表。见 源类型。

* 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 主机自己 Diameter 对等列表中配置的每个地址。
`{ net_devices: true }`	每个受管理网络设备地址（`all.vars.<vendor>.devices`，`mikrotik.devices`）。
`{ roaming: true }`	漫游（IR.21 / GRX-IPX）范围。目前任何源：见 操作注意事项。
`{ literal: "..." }`	一个不来自库存的自由文本源。

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` 归入一个目录。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 与角色一起存在。在角色中添加或移动监听器并更新其 `firewall_services.yml`；中央模型中的内容不会更改。

这种分离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监听的主机。

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

对于每个主机，它报告：

- 未声明 - 在没有角色声明的端口上存在的实时监听器（要么将其添加到角色的 `firewall_services.yml`，要么它是意外的监听器），以及
- 未监听 - 已声明的服务实际上未启动。

声明没有服务的组的主机将被跳过，因此在角色逐步迁移到目录而不是一次性迁移时，此检查是有用的。

条件监听器。某些端口仅在部署另一个角色时打开：MME 仅在库存中存在 `omnilcs` 主机时打开其 LCS 端口（SBc-AP `29168/sctp`，SLs `9082/sctp`）。静态声明无法表达该条件，因此这些端口在角色的 `firewall_services.yml` 中被标记但未声明。在运行依赖角色的站点上，漂移检查将其报告为未声明，这是审查和允许它们的正确提示。

站点配置

每个站点在其库存的 `all.vars` 下声明一个 `firewall` 块。没有地址回退：站点必须描述自己的网络。站点未声明的任何内容都将是空的，其他环境的地址永远不会被继承。

下面的示例是完全填充的，以便每个字段显示一个代表值。地址使用文档范围和一致的每平面布局（每个平面是 `internal` 超网内的自己的子网），这使得区域有用：它们允许一个流仅允许 IMS 平面而不是整个核心。

```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。其他每个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，因为空值会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 ��。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；将其余部分留空。

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）以及 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，以处理模型未涵盖的情况。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	-	���标端口或范围。
`source`	字符串	否	-	源 CIDR。值中的 `:` 将规则应用于 IPv6。
`comment`	字符串	否	`override`	在规则注释中记录的标签。

操作模式

应用模式由 `firewall_mode` 控制。

模式	行为
`monitor`	每个接受规则都被安装，`INPUT` 策略保持为 `ACCEPT`，并记录流量（作为 `FW-AUDIT`），然后允许。没有任何内容被丢弃。使用此模式验证模型与实际流量的匹配，然后再进行强制执行。
`enforce`	流量被记录（作为 `FW-DROP`），然后丢弃，`INPUT` 和 `FORWARD` 策略设置为 `DROP`。

阻止是选择性和安全的：只有确切的值 **`enforce`** 才会丢弃流量。任何其他值，包括未设置的变量或拼写错误，都被视为监控，因此主机无法意外开始丢弃流量。这在三个层面上保持：平台默认值为 `monitor`，站点在其库存中声明 `firewall_mode: monitor`，生成器将任何不是 `enforce` 的内容视为监控。强制执行需要有意的更改。

```yaml
all:
  vars:
    firewall_mode: monitor    # 仅在审核日志确认覆盖后更改为 'enforce'
```

`firewall_mode` 可以在 `all.vars` 下为每个站点设置，或通过正常的变量优先级在每个组或每个主机上覆盖。

跨站点对等体

跨站点联邦的角色（CSCF, TAS, OmniMessage, HSS 复制, OCS）在站点的对等体文件中列出 它们的远程对等体（例如 `prod_master_peers.yml`）。这些远程地址被折叠到 `{ group: <role> }` 解析中，因此允许一个角色的流自动涵盖同一站点和跨站点的实例，适用相同的策略。相同的解析用于规则和矩阵，因此对等体在两者中都出现。

地址解析

主机解析为 每个声明的地址，而不仅仅是其主要地址。这意味着 `ansible_host` 加上库存中的每个 `secondary_ips.<name>.ip_address`。数据路径和信令节点将服务绑定到辅助 NIC: UPF 在其 `n3` 地址上回答 N3 GTP-U，而 STP 在其公共地址上接收和发起 SIGTRAN。规则和矩阵涵盖所有这些地址，作为源和目标，因此允许通过提供给它的地址来允许节点的对等体实际上是在允许流量使用的地址。

属于 多个库存组 的主机接收 每个组的流和声明服务的并集。一个融合节点为其运行的所有角色暴露所有内容，其预期的监听器集（由漂移检查使用）是其组的并集。

由于解析使用每个声明的地址，而不是跟踪哪个服务绑定到哪个 NIC，因此矩阵可以在它实际上不绑定的地址上列出服务（例如 N3 GTP-U 显示在 UPF 的 N6 地址上）。这是故意的过度近似：一个完整但稍微宽泛的矩阵是安全的，可以交给上游防火墙，而省略真实的数据路径或

SIGTRAN 地址则会悄悄中断流量。如果以后需要按地址精度，可以命名绑定的 NIC；在此之前，解析保持基于事实和完整。

通信 / 端口矩阵

端口矩阵是流模型与站点库存解析后的结果，展平为每个源、目标主机和单个端口/协议的一行。它是审查或交给上游防火墙的形式。生成它的命令是：

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

这会写入 `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`	解析后的源 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：



Get an estima

1. 实时规则集被快照。
2. 在主机上启动一个计时器，该计时器将在会话被切断时恢复快照。
3. 应用新规则。
4. 控制器重新建立连接并运行简单检查。如果新规则将其锁定，则此检查失败，运行停止。
5. 如果确认连接，则解除计时器。如果未确认，计时器触发并恢复工作规则集，无需操作员操作。

参数	类型	必需	默认	描述
`firewall_rollback_guard`	布尔值	否	`true`	启用每次应用的快照、计时器和重新连接检查。
`firewall_rollback_timeout`	整数	否	`120`	计时器在自动恢复快照之前等待的秒数。

IPv6 和双栈

地址族根据 CIDR 推断：值中的 `:` 表示 IPv6，规则在 `ip6tables` 上���用。仅包含 IPv4 CIDR 的区域或列表因此不会为该源生成 IPv6 规则。没有 IPv6 核心的站点将 IPv6 区域和池留空（例如 `core_v6: []`）。在强制执行下，IPv6 的 `INPUT` 策略设置为 `DROP`，因此在强制执行仅通过 IPv4 源到达的唯一合法 IPv6 流量之前，请在监控模式下验证 IPv6 流量。

日志记录

审核和丢弃的流量被记录并通过标准管道传送到 Loki。使用三个前缀：

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`	监控模式下的流量：在强制执行下将被丢弃，但在这里允许。在强制执行之前审查这些。
`FW-DROP` / `FW-DROP6`	在强制执行下实际丢弃的流量。

日志目标

记录通过 `NFLOG` 写入，并由 `ulogd2` 读取，该角色作为硬性要求安装，连同 `iptables-persistent`。NFLOG 仅在消费者运行时可读 - 没有 `ulogd2`，内核将丢弃每条记录，而监控模式在看似正常工作时实际上证明无效：规则已安装，计数器上升，但从未生成日志。该失败是静默的，这就是为什么该软件包是强制性的，而不是尽力而为。

这两个软件包都在 Omnitouch apt 仓库中提供（`ubuntu_packages.networking`），因此无法安装它们的主机有一个损坏的仓库，播放在此停止。

Alloy 日志管道将其标记为 `unit="ulogd2.service"` 和 `service="ulogd"`。注意没有 `job="firewall"` 标签 - `job` 带有主机的库存名称，因此选择 `unit` 以跨越每个主机。

示例 Loki 查询：

```
# 一旦这些主机被强制执行，将被阻止的流量
{unit="ulogd2.service"} |= "FW-AUDIT"

# 按主机被强制丢弃的流量
sum by (hostname) (count_over_time({unit="ulogd2.service"} |= "FW-
DROP" [5m]))

# 仅一个主机
{unit="ulogd2.service", hostname="omni-tampa-sbc01"} |= "FW-DROP"
```

在主机本身上：

```
journalctl -t ulogd -f | grep FW-AUDIT
```

操作注意事项

- 漫游和字面源当前为任何源。源为 `roaming` 或 `literal` 的流量在没有源限制的情况下发出 (`0.0.0.0/0`)。在漫游范围解析为真实 CIDR 之前，将其视为开放。
- **ICMP** 被接受自任何源。生成的规则集无条件允许 ICMP，这比为 ping 流量显示的管理范围更广。
- 不要在没有转发规则的情况下强制用户平面主机。在强制下，`FORWARD` 策略设置为 `DROP`，并且没有生成转发规则。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。
- 管理列表较粗糙。`allowed_oam_subnets` 管理 SSH、UI、抓取和管理 API。一个从不同网络抓取而不是从其管理的网络需要一个 `extra_rules` 条目来将它们分开。

组变量配置

概述

`group_vars` 目录是存储自定义配置文件的地方，这些文件会覆盖默认模板。

这里存放着您的客户特定配置：SIP 中继、Diameter 路由规则、SMS 路由逻辑、拨号计划，以及您不希望使用默认配置的任何其他自定义设置。它位于 `group_vars` 中。

位置：`hosts/{Customer}/group_vars/`

角色在运行时将此目录解析为 `group_vars_omnicore` - `group_vars/` 目录与清单目录相邻（`{{ inventory_dir.rsplit('/', 1)[0] }}/group_vars/`）。无论下面的文件路径如何引用 `{{ group_vars_omnicore }}`，它都指向此目录。

工作原理

Ansible 角色有默认的配置模板。要为特定部署进行自定义，请将您的自定义文件放入 `group_vars` 并在您的主机文件中引用它们。

```
角色默认模板 → group_vars 覆盖（如果指定） → 部署配置
```

示例 1：自定义配置模板（OmniMessage）

某些组件接受自定义 Jinja2 配置模板。

文件结构

```
hosts/Customer/
└── group_vars/
    └── smsc_controller.exs          # 您的自定义配置模板
```

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. 用 Jinja2 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，将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（接听）
    │   └── 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，默认为
"dialplan"
```

发生了什么：

1. Ansible 找到 `gateways_folder: "gateways_prod"`
2. 将所有文件从 `hosts/Customer/group_vars/gateways_prod/` 复制到 `/etc/freeswitch/sip_profiles/`
3. 将所有文件从 `hosts/Customer/group_vars/dialplan/`（或 `dialplan_folder` 指定的文件夹）复制到 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. 用 Jinja2 模板化（可以使用 `{{ inventory_hostname }}`、`{{ plmn_id.mcc }}` 等）

4. 部署到 `/etc/omnihss/runtime.exs`
5. 重启服务

没有 `hss_template_config` 时，将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 4：自定义配置模板（OmniMME）

移动管理实体接受自定义运行时配置模板。

文件结构

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

在主机文件中引用

```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    # 在 group_vars 中引用
您的模板文件名
```

发生了什么：

1. Ansible 找到 `mme_template_config: mme_runtime.exs.j2`
2. 在 `hosts/Customer/group_vars/mme_runtime.exs.j2` 中查找
3. 用 Jinja2 模板化（可以使用 `{{ inventory_hostname }}`、`{{ plmn_id.mcc }}` 等）
4. 部署到 `/etc/omnimme/runtime.exs`
5. 重启服务

没有 `mme_template_config` 时，将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 专门从 group_vars 源获取 - 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。

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
    └── omnicrm_plays/            # 可选的站点配置剧本，复制到检出中
```

在主机文件中引用

```yaml
crm:
  hosts:
    customer-crm01:
      ansible_host: 10.10.3.70
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      ssl_fqdn: portal.example.com
```

发生了什么：

1. 将 OmniCRMv2 检出发送到 `/opt/omnicrm-v2`，构建其虚拟环境，并创建 PostgreSQL 角色和数据库。
2. 将 `{{ group_vars_omnicore }}/crm_config.yaml` 模板化到 `/etc/omnicrm/crm_config.yaml`（必需）。
3. 将 `deployment_assets/` 同步到 `/etc/omnicrm/deployment_assets/`。
4. 迁移，在主机上构建 UI，并启动 `omnicrm-v2-*` 单元，后面跟着 nginx。

重要：`crm_config.yaml` 必须存在，并包含一个 `ui:` 块，`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 -


```yaml
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` 角色部署到 UDM 主机上的 `/etc/omniudm/hnet/`。

文件结构

```
hosts/Customer/
└── group_vars/
    └── hnet_udm/
        ├── curve25519-1.key        # X25519 私钥，PEM（家庭网络公钥 ID 1）
        ├── curve25519-1.pub        # 原始 32 字节公值（十六进制） - 用于 USIM 配置
        ├── secp256r1-2.key         # EC（prime256v1）私钥，PEM（HNET 公钥
ID 2）
        └── ...
```

文件命名：`<curve>-<id>.key`，其中 `<id>` 是家庭网络公钥标识符，UDM 根据 SUCI 进行匹配，`<curve>` 是 `curve25519`（配置 A，X25519）或 `secp256r1`（配置 B，prime256v1）。文件名是密钥与其 HNET 密钥 ID 绑定的方式。它必须��编程到 USIM 上的 ID 匹配。

发生了什么：

1. 角色确保 `{{ group_vars_omnicore }}/hnet_udm/` 在 Ansible 控制器上存在（使用 `become: false` 创建：密钥保留在控制器上，永远不会在远程主机上生成）。
2. 它扫描目录中的 `*.key` 文件。
3. 如果未找到任何文件，将自动生成一个默认的 **`curve25519-1.key`**（**X25519**），以便 SUCI 解密可以开箱即用。这是幂等的（`creates:` 保护）：现有密钥不会被覆盖。
4. 对于每个 `curve25519-*.key`，匹配的原始 32 字节公值将导出到 `.pub` 文件（十六进制）以供 USIM 配置。`.pub` 文件保留在清单中，并不部署到 UDM。
5. 所有 `*.key` 文件都被复制到 UDM 上的 `/etc/omniudm/hnet/`（模式 `0600`，根拥有），并重启 `omniudm`。

> 重要：自动生成的密钥是随机的，只能解密与其匹配公钥编程的 USIM 的 SUCI。对于您控制 SIM 编程的实验室，请获取导出的 `.pub` 并将其配置为 USIM 上的家庭网络公钥（ID 1）。如果您的 SIM 使用空方案（无隐蔽），则不需要 HNET 密钥，此目录可以保持为空。

要手动从现有私钥导出公钥：

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

参考 **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`）
`mmsc_runtime_template`	mmsc	Jinja2 模板文件（例如，`mmsc_runtime.exs`）-[
`gateways_folder`	applicationserver	文件夹名称（例如，`gateways_p`
`dialplan_folder`	applicationserver	文件夹名称（例如，`dialplan`）· 设置，默认为 `dialplan`
`tas_runtime_template`	applicationserver	Jinja2 模板文件（例如，`tas_runtime.exs.j2`）设置，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`）

参数	组件	引用
`pgwc_template_config`	pgwc	Jinja2 模板文件（例如，`pgwc_runtime.exs.j2[`
`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` 源 有角色默认值）
`grafana_smtp_user`	monitoring	Grafana SMTP 用户 - 必需， 默认值
`grafana_smtp_password`	monitoring	Grafana SMTP 密码 - 必需， 默认值
`grafana_smtp_host`	monitoring	可选的 Grafana SMTP 主机 - `in-v3.mailjet.com:58`
OmniHSS 配置	omnihss	`omnihss/` 目录，包含 `subscribers.csv` + 配[JSON
HNET 密钥	5gc_nf（omniudm）	`hnet_udm/` 目录中的 `<curv` `<id>.key` 私钥，用于 SUCI [果为空，自动生成默认的 `curve255` `1.key`
`secrets.yml`	common/secrets （omnihss 及其 API 客户	Ansible Vault 文件，包含 AP 密码，通过

参数	组件	引用
	端）	`scripts/vault_secret` 管理 - 参见 客户机密
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[illegible]自定义设置 - 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
- 覆盖默认值的客户特定设置

□ □□□□ **group_vars** 进行:

- 基本主机配置（IP、主机名） - 使用主机文件
- 一次性测试 - 在主机文件中使用特定于主机的变量
- 临时更改 - 在目标上编辑，如果是永久的则提交到 group_vars

相关文档

- 配置参考 - 所有主机参数及其作用
- 主机文件配置 - 如何构建主机文件
- 客户机密 - 加密的 `secrets.yml` 和如何管理它
- **OmniCall** 配置: https://docs.omnitouch.com.au/docs/repos/OmniCall - 配置文件中包含的内容
- **OmniCore** 配置: https://docs.omnitouch.com.au/docs/repos/OmniCore - 组件配置详细信息

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中央配置文档。它指定：

- 要部署的网络功能
- 它们运行的位置（IP 地址、网络段）
- 它们的配置方式（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'
```

常见主机参数

网络配置

每个主机通常包括：

```
pcscf:
  hosts:
    customer-pcscf01:
      ansible_host: 10.10.1.15       # SSH 访问的 IP 地址
      gateway: 10.10.1.1              # 默认网关
      host_vm_network: "vmbr1"        # 在虚拟化管理程序上使用的 NIC 名称
```

注意：有关 IP 地址规划和网络分段策略的指导，请参见 IP 规划标准，其中概述了 OmniCore 部署的推荐四个子网架构。

Proxmox 用户：`host_vm_network` 参数指定使用哪个桥接。有关自动化配置的详���信息，请参见 Proxmox VM/LXC 部署。

VM 资源分配

对于需要特定资源的服务：

```
num_cpus: 4                    # CPU 核心
memory_mb: 8192                # 内存（以兆字节为单位）
proxmoxLxcDiskSizeGb: 50       # 磁盘大小（以 GB 为单位）
```

特定于服务的参数

每个网络功能都有其自己的参数。示例：

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                # 分开控制/用户平面
```

有关每个变量控制的详细说明，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
```

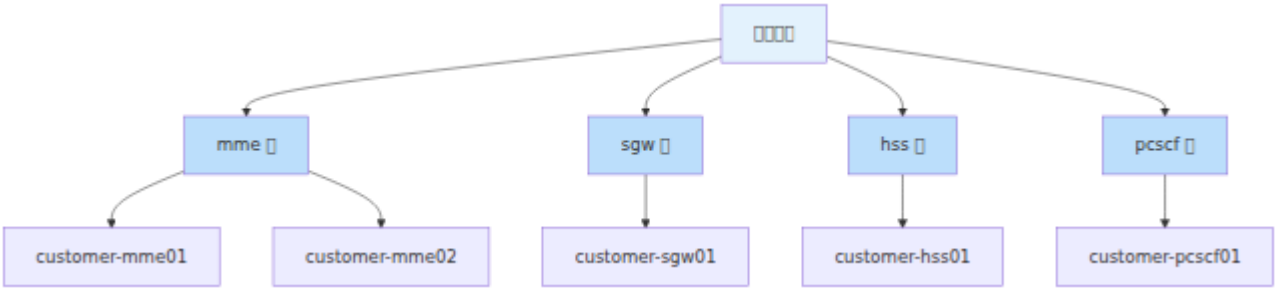
全局变量部分

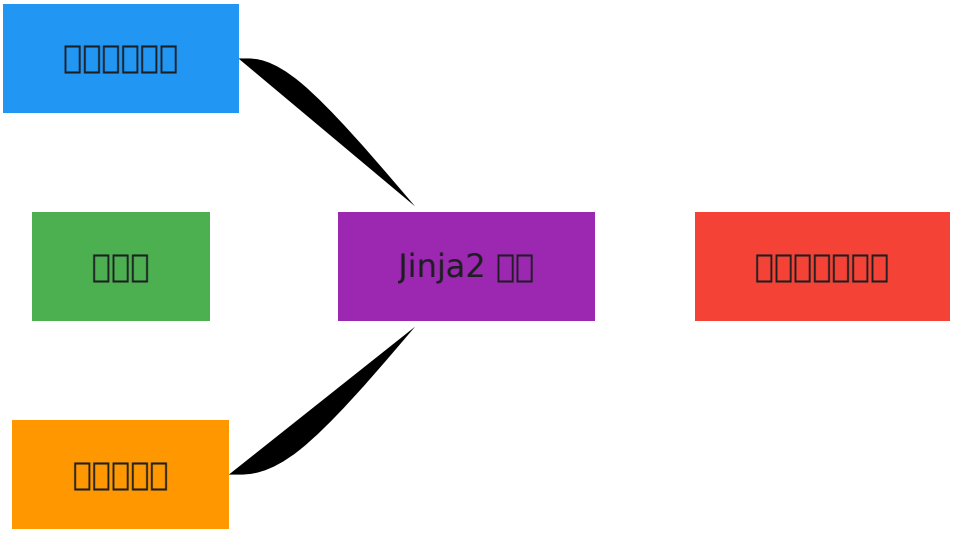
`all:vars` 部分包含适用于整个部署的设置：

```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：如果定义了带有主机的 apt_cache_servers 组，
    # use_apt_cache 默认为 true，apt_repo.apt_server
    # 默认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 将主机组织成与角色相对应的组：





当您运行一个针对 `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少数组。还有许多其他组，包括 `ocs`、`crm`、`smsc`、`omnimessage`、`mmsc`、`dra`、`xcap`、`siptrace`、`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
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` 可以重写主机组的默认值。

参见：组变量配置

完整示例主机文件

这个更完整的示例已将敏感数据遮蔽：

```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 — 有关完整参□集，请参见 docs/firewall.md
    firewall:
      allowed_oam_subnets:
        - '10.0.0.0/8'
        - '192.168.0.0/16'
      allowed_ue_voice_subnets:
        - '10.0.0.0/8'
      zones:
        internal: ['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。
```

有关完整的 Proxmox 设置和配置详细信息，请参见 Proxmox VM/LXC 部署。

产品文档参考

有关每个组件的详细配置，请参阅官方产品文档：

OmniCore 组件：

- **OmniCore** 文档：https://docs.omnitouch.com.au/docs/repos/OmniCore
- **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** - 用户平面功能
- **OmniDRA** - Diameter 路由代理
- **OmniTWAG** - 受信 WLAN 访问网关

OmniCall 组件：

- **OmniCall** 文档：https://docs.omnitouch.com.au/docs/repos/OmniCall
- **OmniTAS** - IMS 应用服务器（VoLTE/VoNR）
- **OmniCall CSCF** - 呼叫会话控制功能
- **OmniMessage** - 短信中心
- **OmniMessage SMPP** - SMPP 协议支持
- **OmniSS7** - SS7 信令栈
- **VisualVoicemail** - 语音邮件

OmniCharge/OmniCRM：

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

相关文档

- Ansible 部署简介 - 整体部署过程
- 配置参考 - 所有配置变量的完整指南
- 组变量配置 - 重写默认配置
- IP 规划标准 - 网络架构和 **IP** 分配指南
- Netplan 配置 - 辅助 **IP** 和高级网络配置

- APT 缓存系统 - 软件包分发
- 许可证服务器 - 许可证管理
- 部署架构概述 - 完整系统视图

下一步

1. 根据此模板创建您的主机文件
2. 定义您的 PLMN 和网络身份
3. 配置 APT 存储库访问
4. 设置许可证服务器
5. 根据需要使用 group_vars 进行自定义
6. 使用 Ansible 剧本进行部署

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运行期间定期验证其许可证。

设置

1. 在 **Hosts** 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"]
```

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
```

真实的库存目录使用大写字母并带有前缀（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，请访问 https://license_server 。

删除许可证

要删除许可证，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到 Omnitouch 许可证验证服务器的 HTTPS（端口 443）流量：

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)
- 目的地：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。基础 IP 可能会在没有通知的情况下更改。不支持静态 IP 允许列表，最终会导致许可证验证失败。

DNS 要求

许可证服务器需要功能正常的 DNS 解析，以便与 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
- Hosts 文件配置

监控与可观察性

概述

OmniCore 包含一个全面的监控和可观察性栈，提供指标收集、日志聚合、可视化和警报。该系统由监控角色自动部署。

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
Node Exporter	主机指标	9100	不适用
SNMP Exporter	网络设备指标	9116	不适用
Blackbox Exporter	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
`Node Exporter`	`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、用户查找
`SGW-C`	`sgw`	9568	承载计数、GTP-C 消息
`PGW-C`	`pgwc`	9090	PDN 连接、IP 分配
`UPF Standalone`	`upf`	9090	数据包计数、吞吐量
`CSCF`	`pcscf`, `scscf`, `icscf`	9090	注册、事务
`ApplicationServer 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
`OmniMessage MMSc`	`mmsc`	9568	MM1 请求、MMS 通知、MM4 转发和入站、在网/离网决策
`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`
Blackbox 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 从运行在所有虚拟机上的 Grafana Alloy 代理接收日志。

数据源 **UID**: `omnicore_loki`

有关 Loki/Alloy 配置的详细信息，请参见 集中式日志记录。

日志标签

标签	描述	示例
`hostname`	源虚拟机主机名	`customer-mme01`
`component`	组件类型	`mme`, `cscf`, `ocs`
`unit`	systemd 单元名称	`omnimme.service`
`level`	日志严重性	`info`, `error`

InfluxDB

InfluxDB 存储特定用例的时间序列数据，这些用例需要更长的保留时间或不同的查询模式。

数据源 **UID**: `omnicore_influxdb`

数据库

数据库	目的	保留
`nokia-monitor`	诺基亚 RAN 性能指标	30 天
`dra`	DRA 路由统计	365 天
`Omnicharge_TAP3`	漫游 TAP 文件指标	90 天

Grafana 仪表板

仪表板组织

仪表板按功能区域组织到文件夹中：

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	Node Exporter、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

仪表板配置（磁盘上的文件）

该存储库是仪表板的唯一来源。Ansible 将角色的仪表板和客户的 `group_vars/grafana/dashboards` 复制到 Grafana 主机上的 `/var/lib/grafana/omnicore-dashboards/`。`/etc/grafana/provisioning/dashboards/` 中的提供程序文件使 Grafana 加载它们。

- 仪表板在 Grafana UI 中为只读。
- 更改的文件将在 30 秒内更新仪表板。
- 删除的文件将删除仪表板。
- 每个目录变成同名的文件夹。
- 重新加载后，剧本将删除任何未从磁盘来的仪表板。
- 客户文件将替换同一路径下的角色文件。

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/
│   ├── OmniMME.json
│   ├── OmniHSS.json
│   ├── OmniDRA.json
│   ├── OmniPGWC.json
│   ├── OmniUPF.json
│   └── ...
├── IMS/
│   ├── IMS_Status.json
│   ├── OmniMessage_(SMS-C).json
│   └── ...
├── Infrastructure/
│   ├── Node_Exporter_Full.json
│   ├── MikroTik_Dashboard.json
│   └── ...
├── RAN/
│   ├── Nokia_Overview.json
│   ├── Nokia_Detailed.json
│   └── ...
└── Logs/
    └── Logs.json
```

仪表板备份

从运行中的 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
```

警报

架构

警报加载

警报规则来自文件。该存储库是规则的唯一来源。

1. `load_grafana_from_export.py` 合并角色的规则、客户的规则和站点覆盖。它写入一个 Grafana 提供文件。
2. Ansible 将该文件复制到 `/etc/grafana/provisioning/alerting/omnicore-rules.yaml` 并重新加载警报。
3. 规则在 Grafana UI 中为只读。

删除：加载器删除主机上所有未通过文件提供的规则。它在新文件的 `deleteRules` 下列出每个从磁盘消失的文件规则。因此，删除规则文件将在下次运行时删除该规则。

联系点和通知策略仍通过 API 从 `hosts/Omnicore_<CUSTOMER>/group_vars/grafana/alerts/` 加载。

默认警报规则

警报	文件夹	条件	持续时间
磁盘空间使用率 **90%**	基础设施	根文件系统 > 90%	5 分钟
CPU 使用率超过 **90%**	基础设施	CPU 使用率 > 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 用户数下降 **40%**	EPC	会话计数下降 40%	30 秒
MME 上 **0** 个用户	EPC	无附加用户	5 分钟
IMS 用户数下降 **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 数量下降	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率 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: default
      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: default
    group_by: ['grafana_folder', 'alertname']
    group_wait: 30s
    group_interval: 5m
    repeat_interval: 4h
```

配置参考

Prometheus 配置

位于监控服务器的 `/etc/prometheus/prometheus.yml`。

参数	默认值	描述
`scrape_interval`	1m	抓取目标的频率
`evaluation_interval`	1m	评估记录规则的频率
`scrape_timeout`	50s	抓取请求的超时

Grafana 配置

位于监控服务器的 `/etc/grafana/grafana.ini`。

监控角色配置的关键设置：

设置	值	描述
`allow_embedding`	true	启用 iframe 嵌入
`provisioning`	`/etc/grafana/provisioning`	提供目录

管理员密码未在 `grafana.ini` 中设置（`admin_password` 行保持注释状态）。相反，它通过 `grafana_admin_password` 变量应用，该变量在部署期间传递给 API 加载任务（仪表板/警报加载器）。

电子邮件（**SMTP**）警��

电子邮件警报需要在 `group_vars` 中设置 SMTP 凭据（请参见 `roles/monitoring/templates/grafana/grafana.ini`）：

变量	必需	默认值	描述
`grafana_smtp_user`	是	*(无)*	SMTP 用户名。无默认值。如果未设置，部署将失败。
`grafana_smtp_password`	是	*(无)*	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

有关 Loki 保留和存储设置，请参见 集中式日志记录。

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
Node Exporter	9100	HTTP	主机指标
SNMP Exporter	9116	HTTP	SNMP 指标代理
Blackbox Exporter	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:

```
# 从特定主机获取所有日志
{hostname="customer-mme01"}

# 从所有 MME 组件获取错误
{component="mme"} |~ "(?i)error"

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

创建自定义警报

将规则文件添加到客户的 `group_vars/grafana/alerts/` 目录，或添加到角色的 `templates/grafana/rules/` 目录以供每个站点使用。运行监控剧本。在 Grafana UI 中创建的规则将在下次运行时被删除。

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 "从生产中更新警报"
```

`--customer` 参数是必需的。该脚本将警报规则、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

可能的原因：

- 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正在运行：

```
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

可能的原因：

- 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

可能的原因：

- 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

可能的原因：

- Prometheus/Loki/InfluxDB 服务宕机
- 防火墙阻止本地连接
- 服务绑定到错误的接口

解决方案：

1. 检查服务状态：

```
systemctl status prometheus loki influxdb
```

2. 验证服务是否在监听：

```
ss -tlnp | grep -E '9090|3100|8086'
```

3. 测试本地连接：

```
curl http://localhost:9090/api/v1/status/config
curl http://localhost:3100/ready
```

VIP 订阅者监控

VIP 订阅者监控跟踪所选订阅者（紧急服务、医院、关键客户路由器和 IP 电话）的服务：IMS 注册、EPC 附加和 UE 可达性。该列表由运营团队在 **OmniWeb → 监控 → VIP 订阅者** 中按站点管理；此存储库中的内容无需更改即可添加或删除订阅者。

架构

OmniWeb 向站点的 HSS 请求每个监控的订阅者，通过探测主机的黑箱导出器 ping UE 地址，该主机路由到 UE 池（从库存的 `pcscf`、`pgwc`、`sgwc` 和 `monitoring` 组中选择），并公开结果。Prometheus 将其作为作业 `vip_subscribers` 抓取。

服务类型

类型	**IMS** 必需	**EPC** 必需	用例
语音 + 数据 (`full`)	是	是	移动终端
仅语音 (`voip_only`)	是	否	仅 VoIP 设备、无 SIM 的 IMS 账户（标识符 = SIP 用户名）
仅数据 (`data_only`)	否	是	路由器、物联网、M2M

`vip_subscriber_service_healthy` 在满足类型要求且账户在 HSS 中启用时为 1 。ping 不会改变判决。

配置

库存中没有任何内容。Prometheus 在 OAM 主机上以作业 `vip_subscribers` 抓取 OmniWeb 服务；端点名称为 subscribers，因此 OmniWeb nginx 仅允许回环调用访问 `/api/vip-subscribers/metrics` 。

探测主机、每次检查的 ping 次数、超时和重新检查间隔在页面（探测面板）上设置，按站点进行配置。

指标

指标名称与早期的每站点出口保持不变，因此仪表板和警报规则可以继续使用。

□□	□□
`vip_subscriber_service_healthy`	□□□□

0 □□□□

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：

```
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□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。

```yaml
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 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

```yaml
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` 또는 `routing_policy` 변수를 사용/수정하려면 템플릿 파일(예: 사용자 정의 `netplan.yaml.j2`)을 수정해야 합니다.

참고 사항

- 네트워크 인터페이스 이름 - 인터페이스 이름 지정

- 기본적으로 VM 생성 시 첫 번째 NIC는 `ens19`, 두 번째 NIC는 `ens20`, 세 번째 NIC는 `ens21`, 그 이후는 `18 + loop.index` 규칙을 따릅니다. 이는 Proxmox 가상 머신에서 NIC 인터페이스 이름 지정 방식과 일치합니다.
- 사용자 정의 인터페이스 이름을 사용하려면 `nic_config.nic_name` 변수를 설정하여 `ens(34 + loop.index)` 형식을 따르세요.

네트워크 구성 `nic_name` 변수를 사용하여 IP 주소를 할당 `nic_name` 변수를 사용하여 네트워크 구성을 참조합니다.

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)

```
<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

网络设备 Syslog

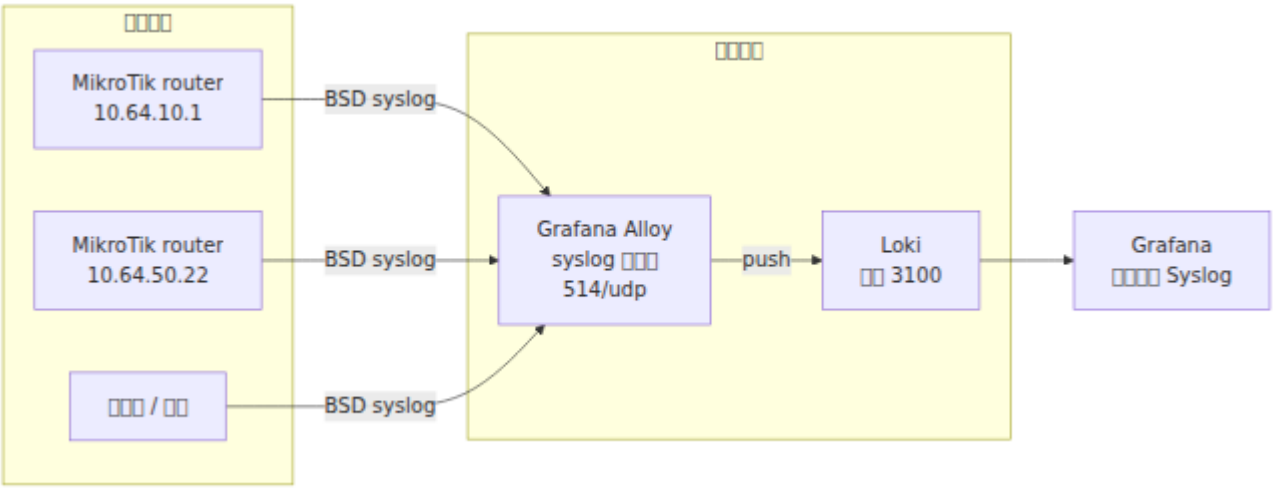
路由器、交换机和其他网络设备无法运行 Grafana Alloy 代理。这些设备发送 syslog。每个监控主机在 Alloy 内部运行一个 syslog 监听器。监听器将消息写入本地 Loki。设备日志与核心网络日志并排存放，并共享相同的搜索、时间范围和保留策略。

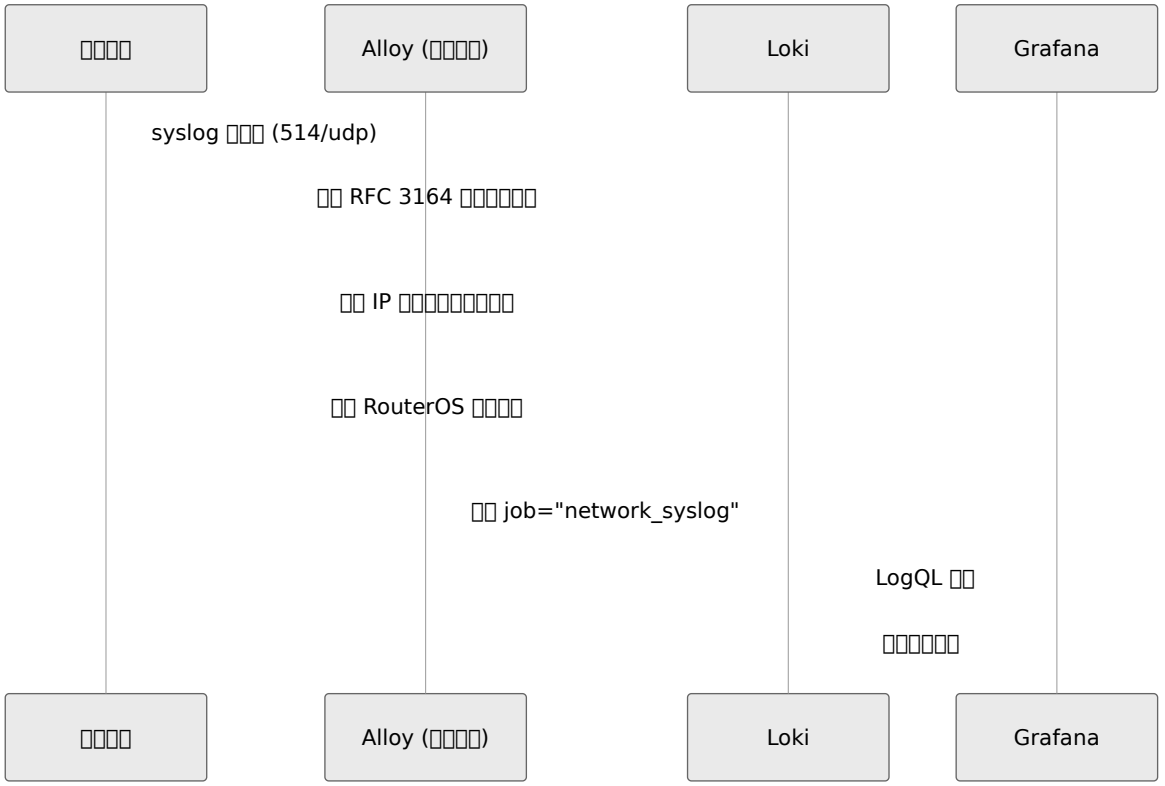
目录

- 架构
- 设备识别方式
- 收集器配置
- 设备清单
- 日志标签
- MikroTik RouterOS
- 其他 Syslog 发送者
- 防火墙
- 仪表板
- 查询日志
- 故障排除
- 参考文献

架构

收集器是标准监控构建的一部分。每个监控主机打开监听器。站点不设置标志以接收设备 syslog。





□□□□
MikroTik router
10.64.10.1
MikroTik router
10.64.50.22
□□□ / □□
BSD syslog
BSD syslog
BSD syslog
□□□□
Grafana Alloy
syslog □□□
514/udp
push
Loki
□□ 3100
Grafana
□□□□ Syslog

监听器接受 BSD syslog，这是 RFC 3164 定义的格式。Alloy 解析优先级、时间戳和消息。然后，Alloy 添加下一节描述的标签。

网络设备
Alloy (监控主机)
Loki
Grafana
syslog 数据报 (514/udp)
解析 RFC 3164 优先级和消息
将源 IP 映射到清单设备名称
提取 RouterOS 主题列表
推送 job="network_syslog"
LogQL 查询
匹配的日志行
网络设备
Alloy (监控主机)
Loki
Grafana

设备识别方式

Alloy 根据数据报的源地址为每个设备命名。Alloy 将该地址与站点清单中的 `devices:` 映射进行比较。匹配时设置 `device` 和 `vendor` 标签。

这种设计产生两个结果：

- 即使设备在消息头中发送错误身份、空身份或重复身份，设备标签也是正确的。
- 从清单中未列出的地址发送的设备仅保留 `device_ip` 标签。将地址添加到清单中以为该设备命名。

如果设备有多个接口，请确保它从清单中列出的地址发送。设置设备上的源地址。

收集器配置

收集器参数位于 `roles/common/defaults/main.yml` 中。在站点清单中的 `all.vars` 下覆盖参数。

```yaml
all:
  vars:
    network_syslog_port: 514          # 监听端口
    network_syslog_protocol: udp      # 传输协议
    network_syslog_vendors:           # 命名发送者的供应商块
      - mikrotik
```

参数	类型	必需	默认	描述
`network_syslog_port`	整数	否	`514`	监听器在监控主机上绑定的端口。低于 1024 的端口是特权端口。请参见下面的说明。
`network_syslog_protocol`	字符串	否	`udp`	监听器的传输协议。使用 `udp` 或 `tcp`。大多数网络设备发送 UDP。
`network_syslog_vendors`	列表	否	`[mikrotik]`	清单供应商块，其 `devices:` 映射命名 syslog 发送者。每个列出的供应商贡献设备名称标签。

Alloy 作为非特权的 `alloy` 用户运行。因此，低于 1024 的端口需要一个能力。当 `network_syslog_port` 低于 1024 时，监控主机上的 Alloy 单元自动接收 `CAP_NET_BIND_SERVICE`。高于 1024 的端口不接收该能力。

使用 `common` 角色应用对这些参数的更改：

```
ansible-playbook -i hosts/<Customer>/host_files/<Site>.yml
services/common.yml \
  --limit <monitoring-host> --tags alloy
```

监控主机为此监听器运行 Alloy。因此，监控主机还将其自己的日志、SSH 和审计日志发送到其本地 Loki。

设备清单

收集器读取与 SNMP 导出器使用的相同供应商映射。每个设备添加一次。该条目随后为设备提供日志标签、SNMP 目标和防火墙规则。

```yaml
all:
  vars:
    mikrotik:
      devices:
        nf-nids-rtr-core01: 10.64.10.1
        nf-flgstf-rtr01: 10.64.50.22
      snmp_community: <community-string>
```

参数	类型	必需	默认	描述
`devices`	映射	是	-	设备名称到 IP 地址。名称成为 `device` 标签。地址标识发送者并打开防火墙。
`snmp_community`	字符串	否	`public`	Prometheus SNMP 导出器的 SNMP 社区。syslog 收集器不使用它。

日志标签

设备 syslog 在 `job="network_syslog"` 下到达。以下标签适用。

标签	描述	示例
`job`	设备 syslog 的固定值。	`network_syslog`
`component`	设备 syslog 的固定值。	`network_device`
`device`	来自清单的设备名称。Alloy 匹配发送者地址。	`nf-nids-rtr-core01`
`device_ip`	数据报的源地址。	`10.64.10.1`
`vendor`	包含设备的清单供应商块。	`mikrotik`
`level`	Syslog 严重性。	`informational`, `warning`, `err`, `crit`
`facility`	Syslog 设施。	`daemon`
`topics`	消息头中的完整 RouterOS 主题列表。	`system,info,account`
`topic`	该列表中的第一个主题。用于常见过滤。	`system`, `firewall`, `dhcp`
`syslog_host`	syslog 头的主机字段，按发送者所写。	`system,info,account`

RouterOS 不发送自己的身份。它在主机名的位置写入其主题列表，因此解析器将该列表存储为 syslog 主机。收集器从该字段读取 `topics` 和 `topic`。写入真实主机名的发送者仅设置 `syslog_host`，并不创建主题标签。

MikroTik RouterOS

从 RouterOS 终端配置每个设备。设备将其内置的 `remote` 日志操作指向监控主机。然后，设备需要为每个主题设置一个日志规则。

```
/system logging action set [find name=remote] target=remote
remote=<monitoring-host> remote-port=514 remote-log-format=bsd-
syslog
/system logging add topics=error action=remote
/system logging add topics=warning action=remote
/system logging add topics=critical action=remote
/system logging add topics=info action=remote
```

属性	值	描述
`target`	`remote`	将日志发送到远程 syslog 服务器，而不是内存或磁盘。
`remote`	监控主机地址	syslog 流的目的地。
`remote-port`	`514`	目的端口。与 `network_syslog_port` 匹配。
`remote-log-format`	`bsd-syslog`	仅适用于 RouterOS 7。发送 RFC 3164。请参见下面的版本表。
`src-address`	清单中的设备地址	可选。当设备有多个接口时设置，以便源地址与清单条目匹配。

RouterOS 版本的消息格式

收集器解析 RFC 3164，因此设备必须发送该格式。控制格式的属性在主要版本之间发生了变化。使用 `/system resource print` 检查版本。

版本	属性	默认	操作
RouterOS 7	`remote-log-format`	`default`	设置 `remote-log-format=bsd-syslog`。`default` 值发送 MikroTik 帧，没有 syslog 优先级头，收集器无法解析���RouterOS 6 的 `bsd-syslog` 属性在这里不存在，该命令会因 `bad parameter bsd-syslog` 而失败。
RouterOS 6	`bsd-syslog`	`no`	设置 `bsd-syslog=yes`。默认值发送裸消息，收集器无法解析。

```
# RouterOS 7
/system logging action set [find name=remote] remote-log-
format=bsd-syslog

# RouterOS 6
/system logging action set [find name=remote] bsd-syslog=yes
```

数据包捕获显示了差异。错误格式在 `tcpdump` 中解码为 `(invalid)`，并且有效负载以二进制字节开头，然后是消息文本。正确格式解码为 `SYSLOG, Facility local0, Severity info`。

RouterOS 主题

主题	描述
`critical`	停止设备上服务的故障。
`error`	失败的操作，例如失败的 DHCP 租约或失败的隧道。
`warning`	需要注意但不停止服务的条件。
`info`	正常事件。此主题承载 `account` 事件，即用户登录、失败的登录和配置更改。

从 `error`、`warning` 和 `critical` 开始，以减少流量。当您想要登录和配置更改审计时，添加 `info`。

检查设备上的结果：

```
/system logging print
/system logging action print
```

其他 Syslog 发送者

任何支持 BSD syslog 的设备都可以使用收集器。适用三个条件：

1. 设备发送 BSD syslog，如 RFC 3164 中定义的那样。收集器不解析 RFC 5424。

2. 设备将 UDP 发送到监控主机的 514 端口，除非站点更改了 `network_syslog_protocol` 或 `network_syslog_port`。
3. 设备从 `devices:` 映射列出的地址发送。防火墙允许该地址，并且该地址为设备提供名称。

将新供应商添加到 `network_syslog_vendors` 以标记其设备。防火墙已经接受每个供应商的设备。

Cisco IOS

```
logging host <monitoring-host>
logging trap informational
logging source-interface <management-interface>
```

Linux rsyslog

```
*.* @<monitoring-host>:514
```

默认的 rsyslog 模板是 RFC 3164。保持该模板。不要选择 `RSYSLOG_SyslogProtocol23Format`，因为该模板发送 RFC 5424。

具有 **Web** 界面的设备

设置远程 syslog 服务器和端口。如果设备提供格式选择，请选择 BSD 或传统格式。

防火墙

流模型携带规则。流 `4b` 允许 `514/udp` 到 `monitoring` 组。源来自 `{ net_devices: true }` 令牌，该令牌解析每个 `all.vars.<vendor>.devices` 映射中的每个地址。

当站点在 `enforce` 模式下运行防火墙时，清单中未列出的设备将被丢弃。将设备添加到清单中，然后应用防火墙：

```
ansible-playbook -i hosts/<Customer>/host_files/<Site>.yml
services/firewall.yml \
  --limit <monitoring-host>
```

有关流模型和模式，请参阅 每角色防火墙 指南。

仪表板

Grafana 在 **Logs** 文件夹中保存 网络设备 **Syslog** 仪表板。

面板	描述
网络设备日志搜索	跨每个设备的自由文本搜索。
按设备的日志速率	每个设备随时间变化的日志量。零处的平坦线显示设备停止发送。
按严重性分类的日志速率	每个严重性随时间变化的日志量。
网络设备日志	选定设备和搜索字符串的所有设备日志。
错误、警告和严重	严重性为 `warning` 及以上。
登录、身份验证失败和配置更改	审计的帐户和系统事件。
链接、OSPF/BGP 和 DHCP 事件	接口、路由和租约事件。

变量	描述
`Site (logs)`	Loki 数据源。选择站点。
`Device`	设备下拉菜单。列表来自 Loki 中的 `device` 标签值。选择一个设备、多个设备或 `All`。
`Search`	日志面板的自由文本过滤器。

查询日志

```
# 来自一个设备的每条消息
{job="network_syslog", device="nf-nids-rtr-core01"}

# 严重性为 warning 及以上，跨整个环境
{job="network_syslog", level=~"emerg|alert|crit|err|warning"}

# 登录和配置更改
{job="network_syslog"} |~ `(?i)(logged in|login failure|changed
by)`

# 一个 RouterOS 主题
{job="network_syslog", topic="firewall"}

# 每个设备的消息速率
sum by (device) (rate({job="network_syslog"} [5m]))

# 在过去一小时内停止发送的设备
sum by (device) (count_over_time({job="network_syslog"} [1h]))
```

故障排除

Loki 中没有设备日志

症状：查询 `{job="network_syslog"}` 没有返回结果。

可能原因：

- Alloy 没有在监控主机上运行。
- 监听器未绑定端口。
- 设备未发送。
- 防火墙丢弃数据报。

解决方案：

1. 使用 `systemctl status alloy` 检查监控主机上的代理状态。

2. 使用 `ss -lunp | grep 514` 检查 Alloy 是否拥有该端口。
3. 使用 `tcpdump -n -i any udp port 514` 捕获流量。流量证明设备正在发送，并且防火墙允许数据报。
4. 如果没有流量，请检查设备上的日志操作。

Alloy 无法启动

症状：`alloy` 服务循环重启。日志显示绑定错误或端口 514 的权限错误。

可能原因：

- 单元没有 `CAP_NET_BIND_SERVICE`。
- 另一个进程占用了该端口，例如本地 `rsyslogd` 的 UDP 监听器。

解决方案：

1. 使用 `journalctl -u alloy -n 50` 阅读错误。
2. 使用 `systemctl cat alloy` 确认能力。
3. 使用 `ss -lunp | grep 514` 查找其他监听器。停止该监听器，或将 `network_syslog_port` 设置为一个空闲端口，并配置设备以匹配。

数据报到达但 Loki 不存储任何内容

症状：`tcpdump` 显示在端口 514 上的流量。Loki 没有 `network_syslog` 流。

可能原因：

- 设备未发送 BSD syslog。RouterOS 7 设备需要 `remote-log-format=bsd-syslog`。RouterOS 6 设备需要 `bsd-syslog=yes`。
- 设备发送 RFC 5424。

解决方案：

1. 使用 `tcpdump -n -i any -A udp port 514` 捕获流量。`tcpdump` 标记为 `(invalid)` 的消息没有 syslog 优先级头，因此格式错误。
2. 为设备的版本设置格式属性。请参见 RouterOS 版本的消息格式。
3. 在另一台设备上，选择 BSD 或传统 syslog 格式。
4. 使用 `journalctl -u alloy | grep -i syslog` 检查 Alloy 日志中的解析错误。

设备标签为空

症状：日志行携带 `device_ip` 但没有 `device` 标签。

可能原因：

- 发送者地址不在 `devices:` 映射中。
- 设备从第二个接口发送，因此源地址与清单地址不同。

解决方案：

1. 从 `device_ip` 标签读���源地址。
2. 将该地址添加到供应商 `devices:` 映射中，或在设备上设置源地址。
3. 使用 `common` 角色应用更改，并在站点强制防火墙时应用防火墙播放。

一台设备停止发送

症状：按设备的日志速率 面板对一台设备没有数据。其他设备继续发送。

可能原因：

- 操作员在设备上删除或禁用了日志规则。
- 设备失去了到监控主机的路由。
- 清单地址已更改，因此防火墙丢弃新的源地址。

解决方案：

1. 使用 `/system logging print` 检查设备上的规则。
2. 从设备 ping 监控主机。
3. 确认设备的当前地址与清单条目匹配。

参考文献

参考	标题
RFC 3164	BSD syslog 协议
RFC 5424	Syslog 协议。收集器不解析此格式。
集中式日志记录	主机日志管道、保留和仪表板
每角色防火墙	流模型、源令牌和模式
监控与可观察性	Prometheus、Grafana 和指标

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-主动网格：每个节点都是完整的读/写主节点，任何节点上的写操作都会传播到所有其他节点。没有主/备用节点。任何节点都可以在任何时间处理 S6a、Cx/Dx 和 Sh 流量并接受配置。

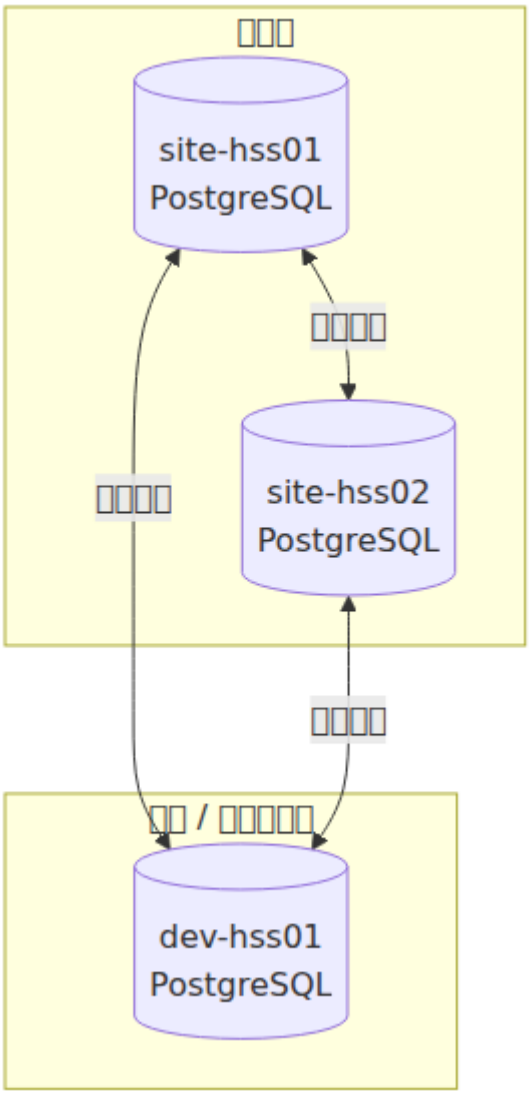
本指南涵盖了复制模型。它解释了如何添加新的 OmniHSS 节点并加载数据。它还解释了如何验证、修复和移除对等节点。

目录

- 架构概述
- 如何发现对等节点
- 配置
- 添加新的 OmniHSS 节点
- 用数据加载新节点
- 验证复制
- 数据完整性
- 移除对等节点
- 备份与恢复
- 指标
- 故障排除

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，网格通过比较 `updated_at` 解决写冲突 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` 个节点的网格，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/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

发布涵盖 `public` 模式中的每个表 除了 `schema_migrations`（每个节点□立运行其迁移）。当两个节点同时更新相同的状态行时，LWW 触发器保留最新的 `updated_at`，而不是最后到达的更新。

如何发现对等节点

当启用复制时，OmniHSS 角色会自动从两个来源构建其对等节点列表：

1. 本地 **`hss`** 清单组中的其他主机：在 □□ 清单/主机文件中定义的 OmniHSS 节点（例如本地的 `hss01` / `hss02` 配对）。这些节点 不 需要在 `connected_omnihss` 中列出；组成员资格就足够了。
2. 站点的远程对等节点文件中的 **`connected_omnihss`** 列表：在 □□ 主机文件中定义的 OmniHSS 节点（其他站点）。跨文件对等节点只能在这里发现，因为一个清单对另一个的 `hss` 组没有知识。

该角色将本地组对等节点过滤为 OmniHSS 复制参与者：那些 `omnihss.replication.enabled` 为 `true` 的节点。标志不是 `true` 的节点不会被视为复制对等节点。

该角色自动将当前主机排除在其对等节点列表之外，因此相同的 `connected_omnihss` 块可以被每个站点共享，而不会出现节点试图从自身复制的情况。

对象命名

每个节点为其复制对象命名，以确保任何大小的网格永远不会发生冲突：

对象	名称	创建于	每个唯一
发布	`<self>_pub`	此节点	节点
订阅	`sub_from_<source>`	此节点（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）。两个节点订阅相同的源将都尝试在该源上创建相同名称的槽。第二个订阅将失败，出现 *"□□□□□□"* 的错误。编码订阅者为每个链接提供了全球唯一的槽。

和解与 `--limit` 安全性

每个角色运行 和解 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` 外的发布将被丢弃（自动清理过时/重命名的发布）。
- 对等节点不再在有效集中的订阅将被禁用、与其远程槽分离并丢弃。
- 不向当前有效对等节点提供的非活动逻辑槽将被丢弃。活动槽永远不会被触及。

这使得 `connected_omnihss` 成为 权威 ��格列表：下一次运行将从每个节点中修剪被移除的对等节点（以及相关的 `hss` 组）。两个安全属性防止其破坏性：

- **`--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` 仅和解 `hss01`。但 `hss01` 仍然将 `hss02`/`hss03` 视为有效并保持其订阅。`--limit` 永远不会修剪被排除的对等节点。
- 空集永远不会修剪。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，运行会修剪一个跨文件对等节点，该节点实际上是网格的一部分，但缺失于 `connected_omnihss`。始终保持 `connected_omnihss` 完整，以涵盖每个跨文件对等节点。

配置

启用复制

复制由 `omnihss.replication.enabled` 标志每个部署控制。默认值为 `false`。在站点的清单或 group_vars 中设置它：

```
omnihss:
  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_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。创建时具有 `REPLICATION` 属性，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	否	`false`	主开关。当 `true` 时，PostgreSQL 被配置为逻辑复制，并创建发布/订阅。

当 `replication.enabled` 为 `true` 时，角色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` 覆盖）

`pg_hba.conf` 被更新，以允许每个对等节点的 IP（`/32`）进行 `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`）：

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

参数	类型	必需	默认	描述
`name`	字符串	是	-	对等节点的 `inventory_hostname`。这命名了订阅（`sub_from_<name>`）和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，因此完整的 `connected_omnihss` 列表应包含 每个 网格中的 OmniHSS 节点。每个站点自动跳过其本地节点（它们已由本地 `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：加入网格（以便更改双向流动）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。

操作员
新节点
现有网格
1. 配置虚拟机 + 基本配置
2. 添加到清单（hss 组 + connected_omnihss）
3. 运行 omnihss 角色（安装 PG、应用程序、迁移）
创建发布/订阅（空模式）
4. 从健康的对等节点播种数据
5. 验证复制健康
操作员
新节点
现有网格

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的 host_file 下的 `hss:` 组：

```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��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` 以便它获取任何新表。

此时，新节点已成为网格的成员，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

该角色创建自动订阅时 `copy_data = false`。这对于 绿地网格 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 已填充 网格的节点在加入之前保持为空，除了之后发生的更改。

选择与您的情况匹配的路径：

绿地网格

如果您同时启动所有 OmniHSS 节点而没有预先存在的数据，则不需要播种。启用复制，在所有节点上运行角色，并正常配置订阅者。每次写入都会复制到所有成员。

选项 **A**：通过转储和恢复播种（推荐）

对健康的对等节点进行快照并将其恢复到新节点。然后让复制携带持续的更改。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. 备份健康的对等节点。这会在控制机上生成 OmniHSS 数据库的 `pg_dump`（使用 `--no-publications --no-subscriptions`，因此复制对象不会被携带）：

   ```
   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. 恢复到新节点。恢复剧本停止 OmniHSS，删除并重新创建数据库，恢复转储，并重新启动服务。它首先对目标进行预恢复备份：

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

添加 `-e skip_pre_restore_backup=true` 以跳过预恢复转储——当目标是新节点或已经为空的节点时非常有用，或者当转储较慢且其当前内容已知无用时。这样不会保存任何内容，因此节点的当前数据无法恢复。

当提示时，提供步骤 1 中 SQL 转储的路径，并将配置路径留空（新节点已经从角色运行中获得了自己的配置）。

3. 确认复制正在流动。新节点的现有订阅（`copy_data = false`）现在在恢复的快照上应用更改。运行下面的 验证。

> 维护窗口：转储与复制赶上之间存在一个小窗口。在此期间，对等节点上的配置更改尚未在新节点上。主订阅者数据变化不频繁，动态状态（`subscriber_state`, `pdn_session`, `lte_call`, `ims_subscription`）会从实时流量自我修复，因此低活动窗口是足够的。在播种期间不要配置新的订阅者。

选项 B：使用播种剧本播种（推荐）

`seed_hss_replication.yml` 实用剧本从 一个 选择的源对等节点播种，而没有原始批量复制的陷阱。它将订阅重新创建为仅流式（`copy_data = false`，没有表同步）。然后，它以安全的方式回填现有的 主表 行。`subscriber_state` 是 回填的一部分：它是结构性的 1:1 数据（每个订阅者通过外键引用一次，在配置时创建）。跳过它会使每个回填的订阅者孤立，这会破坏目标上的 API 和 S6a。流式复制无法修复，因为逻辑复制会默默地丢弃对不存在行的更新。该剧本故意不批量复制真正瞬态的会话表（`pdn_session`, `lte_call`）。这热表的 `COPY` 会与时系统上的并发写入发生冲突（重复键），并导致表同步崩溃循环。它们会从流式复制和实时流量中重新填充。

交互式（推荐）：省略源。该剧本会查找哪些目标为空，探测对等节点以获取持有数据的节点，并提示您选择：

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

```
可用据的对等节点:
  1) dev-hss01 (10.4.10.142) - 838 个订阅者
输入要播种的源的编号
```

明确：直接命名源。这会强制重新播种，即使目标已经有数据（用于恢复）：

```
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"
```

请参见 HSS 复制播种 以获取完整的参数和行为参考。

`seed_hss_replication.yml` 还在播种后 连接完整网格：它在每个节点上运行复制和解（发布、订阅、槽）并验证它。最好在没有 **`--limit`** 的情况下运行它——空节点会自动检测为播种目标，填充节点为源，并且每个节点都在一次通过中连接。`--limit` 会缩小播种���连接，因此单个 `--limit` 节点会使网格半连接；仅在有针对性的重新播种时使用它，然后重新运行没有 `--limit` 以连接其余部分。从确切的一个源播种。对于非常大的数据集，优先选择 选项 A。

在目标节点上运行的等效手动步骤。首先创建流式订阅（注意每个链接的 `slot_name`，请参见 对象命名）。然后在跳过热状态表的同时安全地回填主表：

```
-- 1. 流式传输所有表，不进行批量复制
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会孤立：其
#    subscriber_state_id 指向一个永远不会到达的行，破坏 API/S6a，
#     流式复制无法修复，因为逻辑复制
#     默默地丢弃对缺失行的更新）。
#     负载必须带有复制来源的戳记：origin=none
#     订阅在发布者处过滤带有戳记的更改，因此回填对网格是不可见的。未戳记的负载
#     从 WAL 解码，就像任何本地写入一样，并将插入回显到已经持有行的对等节点，
#     使其应用工作者停滞。
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

该角色在设置后会自动运行健康检查，也可以单□运行。验证断言：

- 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应该是非空的
sudo -u postgres psql -d omnihss -c \
  "SELECT subname, pid IS NOT NULL AS worker_alive FROM
pg_stat_subscription;"

# 复制槽 - active应该为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）	没有订阅者正在消费该槽：对等方的订阅已停止或被删除。
增长的`lag`	对等方运行缓慢或连接质量下降。WAL正在积累并消耗磁盘空间。
`wal_status = lost`	槽落后太多，PostgreSQL删除了WAL。您必须重新播种对等方（请参见用数据加载新节点）。

健康检查报告（`health_check.yml`）还包括一个**HSS Postgres** 复制面板，适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。

数据完整性

逻辑复制应用不强制外键。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-主动的网格中，打开了一个漏洞，没有单个节点可以捕获：

- 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。

- 两个更改都复制并应用而没有FK检查。每个节点都收敛到“203 → X”和“X已删除”：一个悬空的引用，没有节点的FK在本地违反。

悬空的`subscriber.epc_profile_id`���HSS无法为该订阅者构建订阅数据。HSS随后在ULR/AIR上回答`DIAMETER_ERROR_USER_UNKNOWN (5001)`，尽管订阅者行存在且已启用。数据前的模式窗口（如下）使这种情况更可能，因为跳过的事务可能会将父项的创建/删除与子项的重新指向分开。

检测（而非强制）

在应用路径上强制FK会在任何异常时停止网格，因此我们检测并报警。每个节点安装一个`SECURITY DEFINER`函数，`omnihss_fk_orphan_count()`，该函数计算每个指向缺失父项的单列FK。`postgres_exporter`选择它，当它大于零时，Grafana会报警（请参见指标）。

随时手动运行清理：

```
sudo -u postgres psql -d omnihss -tAc "SELECT
omnihss_fk_orphan_count();"   # 0 = 干净
```

要定位特定引用的违规行（示例：指向不存在的EPC配置文件的订阅者）：

```
SELECT s.imsi, s.epc_profile_id
FROM subscriber s
LEFT JOIN epc_profile e ON e.id = s.epc_profile_id
WHERE s.epc_profile_id IS NOT NULL AND e.id IS NULL;
```

通过将行重新指向有效的父项进行修复（对现有行的`UPDATE`在网格中干净复制）或删除它。

防止模式漂移

促使本节���事件的根本触发因素是在一个节点上应用的模式迁移，而其对等方仍在旧模式上，然后针对新字段写入新数据。源发布的行使旧模式的对等方无法应用，导致应用错误和跳过的事务，因此导致分歧。

- 一起迁移整个网格。在针对新字段进行配置之前，对每个OmniHSS节点应用迁移。在任何对等方落后时，不要写入新模式数据。
- 监控漂移。节点导出`omnihss_schema_max_version` / `omnihss_schema_migration_count`。在网格中进行比较。落后于其对等方的节点是半应用的迁移。
- 监控错误计数器。上升的`omnihss_subscription_error_apply_error_count` / `_sync_error_count`意味着应用工作进程失败（并可能跳过）。此时会触发“HSS复制应用/同步错误”警报。

移除对等方

移除对等方将从该对等方中删除订阅，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`中删除对等方的行，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���该节点的订阅。如果链接应该完全拆除，也请在那里移除该订阅。然后从`connected_omnihss`和`hss`组中移除该节点，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的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。转储不包括发布/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，将节点从网格中拆除（删除其订阅/发布/槽并禁用对等方的订阅），删除/重新创建数据库，恢复，迁移并重启。它拒绝同时针对多个节点。

恢复与网格

在清除之前，恢复将节点从主动-主动网格中移除。`DROP DATABASE`在存在订阅或逻辑槽时失败。当启用复制时（`omnihss.replication.enabled`），恢复按以下顺序重建网格：

1. 发布。剧本在恢复的节点上创建发布。它尚未将恢复的节点订阅到其对等方。
2. 重新连接并播种本地对等方。剧本从每个本地`hss`对等方重新创建到恢复节点的订阅，`copy_data = false`。[illegible]后用`pg_dump --data-only --inserts --on-conflict-do-nothing`填充每个空对等方。它跳过`schema_migrations`、`pdn_session`和`lte_call`，如`seed_hss_replication.yml`所示。持有数据的对等方保留其数据并仅恢复流。
3. 重新订阅。剧本运行角色的`replication.yml`。这会创建从恢复节点到其对等方的订阅。

仅在步骤2之后执行步骤3。如果恢复的节点在播种之前订阅了对等方，则播种行会流回到它。这会导致`duplicate key`错误和应用工作进程崩溃循环。

不要在对等方上使用`TRUNCATE`或`COPY`。逻辑复制会复制`TRUNCATE`，因此它会清除恢复节点和其余的网格。

所有本地对等方必须可达。如果某个对等方不可达，恢复将失败。

恢复不会播种跨站点（`connected_omnihss`）对等方。使用`seed_hss_replication.yml`播种它们。恢复后，运行`util_playbooks/audit_hss_drift.yml`以确保网格收敛。

请参见实用剧本以获取更广泛的备份/恢复工作流程。

将节点恢复到主动-主动网格中

`restore_hss.yml`在恢复的节点上拆除复制：在删除和重新创建数据库之前，它删除该节点的订阅、发布和槽。因此，恢复后该节点没有复制对象 — 网格必须重新连接，仅重新连接[illegible]是不够的（恢复的节点仍然需要其自己的发布）。两个规则：

1. 仅恢复一个节点。同时恢复两个或多个节点会破坏网格（`restore_hss.yml`对此发出警告）。恢复一个节点，然后从它播种其余节点。
2. 在连接复制之前，所有节点必须运行相同的**OmniHSS**版本。逻辑复制携带数据，而不是模式。`CREATE SUBSCRIPTION`检查发布者的发布中的每个表是否已经存在于订阅者上，因此在较新版本的节点（其发布包含后续迁移添加的表，例如`ims_subscription`）无法被仍在旧模式上的对等方订阅 — 订阅会因`relation "public.<table>" does not exist`而失败，该方向永远不会出现。恢复转储的模式也必须与已部署版本匹配，否则恢复的节点将超前于其对等方。

从头开始重建一个主动-主动对：

```
# 1. 配置两个虚拟机，然后在两个节点上以相同的版本安装OmniHSS + 模式（每个节点上相同、空的、迁移的模式）。
ansible-playbook -i hosts/<inv>/host_files/<site>.yml omnihss.yml

# 2. 仅在一个节点上恢复据（转储模式必须与步骤1的版本匹配）。
#    这会清除该节点的复制对象；步骤3重建网格。
ansible-playbook -i hosts/<inv>/host_files/<site>.yml \
  util_playbooks/restore_hss.yml --limit <site>-hss01

# 3. 播种其他节点并连接完整的双向网格 — 不要使用 --limit。
#    空节点从恢复的节点回填；在每个节点上（重新）创建发布、订阅和槽，然后进行验证。
ansible-playbook -i hosts/<inv>/host_files/<site>.yml \
  util_playbooks/seed_hss_replication.yml
```

请勿以`seed_hss_replication.yml --limit <peer>`结束：`--limit`也限制了连接，因此它只创建对等方的订阅，而使恢复的节点没有其发布 — 一个半连接的单向网格。`validate_replication.yml`（自动运行）现在断言每个节点都订阅每个对等方，因此半连接的结果会大声失败，而不是报告健康。

指标

每个OmniHSS节点运行`postgres_exporter`，将PostgreSQL指标（包括复制）暴露给Prometheus。有关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)

# 应该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`作业下抓取这些指标。它们支持**HSS**警报组中的警报规则（`roles/monitoring/templates/grafana/rules/`）：

□□	□□	□□
`omnihss_subscription_worker_alive{subname}`	Gauge	□□□ □□□ □□□ □□□ □□ □□□ □1
`omnihss_replication_slot_active{slot_name}`	Gauge	□□□ □□□ □□□ □□□ □□1
`omnihss_replication_slot_wal_status_code{slot_name}`	Gauge	0=[□ 1=[□ 2=[□□ **3=**[□
`omnihss_replication_slot_lag_bytes{slot_name}`	Gauge	□□□ □□□ WA □□
`omnihss_subscription_error_apply_error_count{subname}`	Counter	□□□ □□□ □□□ □
`omnihss_subscription_error_sync_error_count{subname}`	Counter	□□□ □□□

指标	类型	意义
		误
`omnihss_integrity_fk_orphans`	Gauge	悬空键的（应为0
`omnihss_schema_max_version` / `omnihss_schema_migration_count`	Gauge	应用移的水位计数

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

新节点在加入后保持空白

症状：复制工作进程处于活动状态，槽处于活动状态，但新节点没有（或很少）订阅者。

可能的原因：

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

解决方案：

1. 从健康的对等方播种该节点。请参见用数据加载新节点。

订阅工作进程未运行

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

可能的原因：

- 对等方在PostgreSQL端口不可达（防火墙/路由）。

- 对等方的IP在该节点的`pg_hba.conf`中缺失。
- 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. 确保TCP可以到达对等方的`5432`端口。
2. 验证对等方是否出现在`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：某个槽的`wal_status`为`lost`；相应的对等方落后很多。

可能的原因：

- 对等方停机或不可达的时间过长，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。请参见用数据加载新节点。

初始复制期间的重复键错误

症状：在创建带有`copy_data = true`的订阅后，应用更改时出现错误。

可能的原因：

- 初始复制是从多个对等方运行的，复制了相同的行两次。

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”。

可能的原因：

- 对同一行在两个站点的并发更新。

解决方案：这是预期的并且可以处理。最后写入获胜触发器保留最新的`updated_at`更新。确认所有站点[illegible]时钟已同步（NTP），以便`updated_at`排序有意义。

Proxmox VM/LXC 部署

我们大多数客户在 Proxmox 上运行 OmniCore 堆栈。单个剧本 `util_playbooks/proxmox.yml` 根据 `deployment_type` 切换来配置 QEMU 虚拟机和 LXC 容器。您可以在每个站点（在 `proxmoxServers` 条目上）设置切换，并在每个主机上覆盖它。这使得一个站点可以大部分作为 LXC 运行，而个别主机（例如 UPF）保持为完整的虚拟机。

我们仍然支持 VMware、HyperV 和云（Vultr / AWS / GCP）进行部署。

另见：

- 主机文件配置：定义要部署的虚拟机/LXC
- IP 规划标准
- Netplan 配置
- 部署架构

LXC 与 VM

LXC 容器：

- 轻量级，共享主机内核
- 启动快速，开销低
- 隔离有限
- 不能运行自定义内核或内核模块
- 不适合生产 **EPC/IMS** 部署
- 不能运行 **UPF**（需要内核模块/TUN 设备）

虚拟机（**KVM**）：

- 完全虚拟化，具有���用内核
- 完全隔离，运行内核模块/自定义网络
- 更高的资源开销
- 推荐用于生产
- **UPF** 部署所需

用例：

- 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
```

说明:

- 模板没有硬编码的用户或密码。`proxmox.yml` 在克隆时设置 `ciuser`、`cipassword` 和 `sshkeys`。
- Cloud-init 用户优先级: `proxmoxTemplateUser`（在匹配的 `proxmoxServers` 条目上）→ 第一个 `local_users` 键。
- Cloud-init 密码优先级: `proxmoxTemplatePassword`（在匹配的 `proxmoxServers` 条目上）→ cloud-init 用户（即上面选择的值）。
- SSH 密钥通过每个 `local_users.*.public_key` 条目注入（基于密钥的身份验证是预期的登录路径）。
- 每当模板是使用不是第一个 `local_users` 条目的 cloud-init 用户准备的，设置 `proxmoxTemplateUser`（可选 `proxmoxTemplatePassword`）在每个 `proxmoxServers` 条目上，以便选择正确的帐户。
- `--vga std` 确保 Proxmox Web 控制台正常工作。
- wget 的 `-N` 标志仅在本地副本更新时下载。

替代方案: 从 **ISO** 手动创建模板

如果云镜像不可用或您需要自定义安装:

步骤 **1**: 通过 **Web UI** 创建虚拟机

- 创建新虚拟机 → 虚拟机 ID 9000, 名称: ubuntu-2404-template
- 操作系统: 上传 Ubuntu Server ISO 或使用现有 ISO
- 系��: 默认（SCSI 控制器: VirtIO SCSI）
- 磁盘: 32GB, 总线: SCSI
- CPU: 2 核心
- 内存: 2048 MB
- 网络: VirtIO, 桥接 vmbr0
- 启动虚拟机并安装 Ubuntu Server

步骤 **2**: 在虚拟机内部 **-** 清理和准备

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

# 清理机器特定据
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

# 清除 bash 历史并关闭
history -c
sudo poweroff
```

步骤 **3**：添加 **Cloud-Init** 并转换为模板

- 选择虚拟机 → 硬件 → 添加 → CloudInit 驱动器（选择存储，例如 local-lvm）
- 硬件 → 选项 → QEMU 客户代理 → 启用
- 右键单击虚拟机 → 转换为模板
- 不要 在模板上设置 cloud-init 用户/密码。`proxmox.yml` 在克隆时从 `proxmoxTemplateUser` / `proxmoxTemplatePassword` 注入这些（使用 `local_users` 作为用户名的后备）。

2b. 下载 LXC 操作系统模板（仅限 LXC 站点）

```
# 在 Proxmox 节点 shell 中:
pveam update
pveam download local ubuntu-24.04-standard_24.04-2_amd64.tar.zst
```

生成的卷路径（例如 `local:vztmpl/ubuntu-24.04-standard_24.04-2_amd64.tar.zst`）是您 ﷺ下面配置为 `proxmoxLxcOsTemplate` 的内容。

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
```

使用

部署（虚拟机或 **LXC**）

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

删除

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

删除剧本自动处理虚拟机和 LXC，无论 `deployment_type` 如何。

剧本行为

类型保护

- 如果存在具有目标主机名的虚拟机，但 `deployment_type: lxc` → 剧本以清晰的错误失败。
- 反向情况（LXC 存在，站点配置为虚拟机）→ 同样。
- 在 LXC 站点上的 UPF 组 → 仅警告（不是硬失败，但 UPF 将无法正常工作）。

新资源

- 在组内按主机索引在 `proxmoxServers` 条目之间轮询，除非主机设置 `proxmox_server:` 以固定特定条目（见 跨节点分配）。
- 查询 Proxmox API 以获取下一个可用的 VMID。
- 虚拟机：从 `proxmoxTemplateId` 克隆，设置 cloud-init 用户/密码/SSH 密钥（cloud-init 用户 = 如果设置则为服务器条目的 `proxmoxTemplateUser`，否则为第一个 `local_users` 键；cloud-init 密码 = 如果设置则为 `proxmoxTemplatePassword`，否则为 cloud-init 用户名），调整 `scsi0` 大小，启动。
- **LXC**：从 `proxmoxLxcOsTemplate` 创建，rootfs 在 `proxmoxLxcRootFsStorageName`（组）→ `proxmoxLxcDefaultStorage`（服务器）→ `storage` → `local-lvm`（最终后备），通过 `ssh-public-keys` 注入所有 `local_users.*.public_key`，启动。根密码硬编码为

`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）。

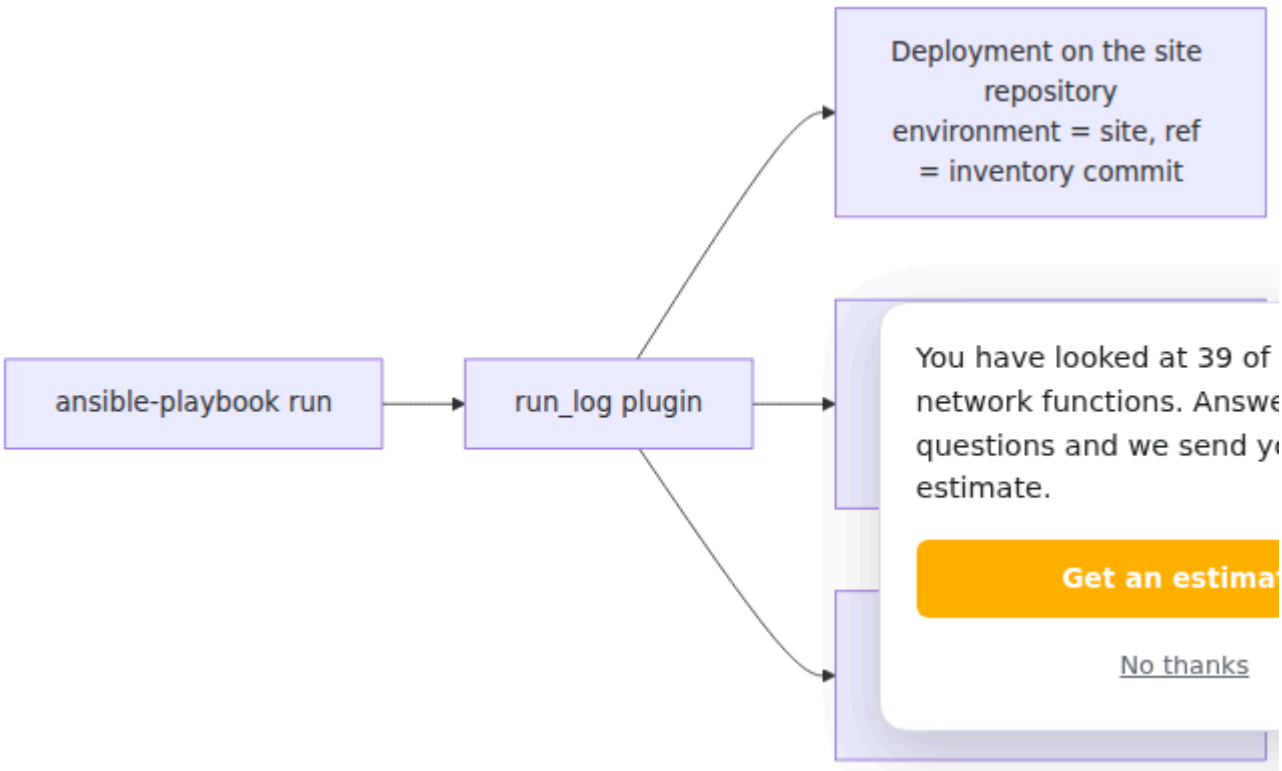
```yaml
device_tester:
  hosts:
    site-device-tester01:
      ansible_host: 10.179.2.50
      proxmox_server: nicklab-prxmx01    # 固定到此 proxmoxServers 条目（→ 其节点）
      # 如果该节点的磁盘存储与全局 proxmoxLxcRootFsStorageName 不同，
      # 也在此处覆盖（存储是全局读取的，而不是从服务器条目读取）：
      proxmoxLxcRootFsStorageName: local-lvm
```

`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。

```yaml
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
```



Playbook 运行记录

`run_log` 回调插件记录每次在 GitHub 上运行的 `ansible-playbook`。它在 git 检出和 apt 安装的工具包中的 `ansible.cfg` 中启用。操作员像以前一样运行 playbook:

```
ansible-playbook -i hosts/YourCustomer/host_files/production.yml
services/firewall.yml
```

在运行结束时，插件打印本地运行目录、部署和日志链接。

记录内容

记录保存在站点仓库中：持有通过 `-i` 提供的清单文件的 git 检出的 `origin`。站点名称是没有扩展名的清单文件名。

记录	内容
部署	环境 = site。引用 = 清单提交。状态 = `success` 或 `failure`。有效载荷 = 命令、操作员、机器、开始时间、持续时间、结果、失败的主机、Ansible 版本，以及两个检出的版本：分支和提交，或来自 apt 安装的 `omnicore` 包版本，以及未提交文件的数量。
提交评论	运行的完整日志、play 回顾和两个检出的未提交更改。部署状态链接到此。长日志分成多个评论。
Slack 消息	结果、操作员、命令、两个提交，以及指向部署、提交和日志的链接。发布到 `monitoring_data.contactPoints` 中的第一个 `slack` 接收器，Grafana 通过该接收器发送警报。
本地副本	`~/.local/state/omnicore/runs/<id>/`：`ansible.log`、`run.json`、`playbooks.diff`、`inventory.diff`。

要求

- 清单文件位于 `origin` 在 GitHub 上的 git 检出中，并且清单提交已推送。
- 安装并登录了 `gh`（GitHub CLI），并具有对站点仓库的访问权限。
- 在 `ansible.cfg` 中设置了 `log_path`。

如果未满足某个要求，插件将在运行结束时打印 `run_log:` 错误。playbook 结果不会改变。

□□□□

□□□

名称	类型	需要	描述
`omnihss_api_keys`	映射	`omnihss_api_auth: rbac`（默认值）	每个客户端一个 OmniHSS REST API 密钥。每个密钥是至少 32 个字符的随机机密。两个客户端不得共享一个密钥。请参见 OmniHSS API Keys。

其他角色在采用该文件时会在本指南的各自部分记录它们读取的名称。

管理机密

每次更改都使用 `scripts/vault_secrets.py`。不要使用 `ansible-vault edit` 编辑文件：该工具将明文保存在管道中，并且从不写入磁盘。

```
scripts/vault_secrets.py hosts/<customer> list [vault options]
scripts/vault_secrets.py hosts/<customer> set NAME [--random-hex N] [--map K1,K2,...] [--force] [vault options]
```

参数	描述
`hosts/<customer>`	客户库存目录。必须包含 `host_files/`。
`list`	打印机密名称。对于映射，也打印其键。绝不要打印值。
`set NAME`	将 `NAME` 添加到文件中。如果文件不存在，则创建该文件。
`--random-hex N`	将值设置为随机字符串，长度为 `N` 字节，以 `2N` 个十六进制字符表示。`N` 必须至少为 16。
`--map K1,K2,...`	与 `--random-hex` 一起使用，将值设置为每个键一个随机字符串的映射。
`--force`	替换已存在的 `NAME`。如果没有此选项，工具将拒绝。
vault options	传递给 `ansible-vault`，例如 `--ask-vault-pass` 或 `--vault-password-file FILE`。

如果未提供 `--random-hex`，工具将从标准输入读取一行并将其用作值。

使用 `--ask-vault-pass` 时，工具会询问密码一次。当它创建新文件时，会询问两次以确认。

示例

为新客户创建文件和 OmniHSS API 密钥：

```
scripts/vault_secrets.py hosts/ExampleCustomer set
omnihss_api_keys --random-hex 32 \
    --map omniweb,omnihlr,omnicrm,omniudr,omnieir,ran_simulator --
ask-vault-pass
```

从您已有的值中添加机密：

```
echo 'the-api-key' | scripts/vault_secrets.py
hosts/ExampleCustomer set example_api_key --ask-vault-pass
```

查看客户拥有的机密：

```
scripts/vault_secrets.py hosts/ExampleCustomer list --ask-vault-
pass
```

轮换 OmniHSS API 密钥：

```
scripts/vault_secrets.py hosts/ExampleCustomer set
omnihss_api_keys --random-hex 32 \
    --map omniweb,omnihlr,omnicrm,omniudr,omnieir,ran_simulator --
force --ask-vault-pass
```

然后运行 HSS 和每个 API 客户端的 playbook，以便所有人都获得新密钥。

运行 **Playbooks**

每个到达具有机密的角色的 playbook 都需要 vault 密码。可以通过两种方式提供：

方法	命令	用途
提示	`ansible-playbook ... --ask-vault-pass`	交互式运行
密码文件	`ansible-playbook ... --vault-password-file ~/.ansible/<customer>_vault_pass`	无人值守运行，CI

每个客户的 vault 密码是共享密码管理器中的 OmniCore 条目。将密码文件保存在外部，权限为 `0600`。

将 `secrets.yml` 提交到客户库存仓库。提交是安全的：该文件是密文。在推送之前请验证：

```
head -c 14 hosts/ExampleCustomer/group_vars/secrets.yml   # prints $ANSIBLE_VAULT
```

OmniHSS API 密钥

OmniHSS 使用每个客户端一个密钥来保护其 REST API。每个客户端都有一个固定的 OmniHSS 角色，因此被盗的密钥仅授予该客户端的权限。

密钥	客户端	**OmniHSS** 角色
`omniweb`	OmniWeb	`oam`
`omnihlr`	OmniSS7 HLR 互通	`hlr_interworking`
`omnicrm`	OmniCRM	`crm_provisioning`
`omniudr`	OmniUDR (5G 核心)	`udr`
`omnieir`	OmniEIR (5G 核心)	`eir_lookup`
`ran_simulator`	RAN 模拟器	`ran_simulator`

`omnihss` 角色将所有六个密钥渲染到 HSS 配置中。每个客户端角色仅渲染其自己的密钥。密钥必须至少为 32 个字符，并且两个客户端不得共享一个密钥。如果违反任何规则，则运行失败。

参数	类型	必需	默认	描述
`omnihss_api_auth`	字符串	否	`rbac`	`rbac`：每个客户端从 `omnihss_api_keys` 发送其密钥，且需要 `secrets.yml`。`disabled`：OmniHSS 不需要密钥即可为每个调用者提供服务，无需机密文件。将其设置在主机文件中的 `all.vars` 下。

故障排除

运行失败，显示 "必须存在并且已使用 **ansible-vault** 加密"

症状：角色在 "断言机密文件存在并且已加密" 停止。

可能原因：

- 客户尚未拥有 `group_vars/secrets.yml`。
- 文件以明文形式写入。

解决方案：

1. 运行错误消息中显示的 `set` 命令。它会创建加密的文件。
2. 如果存在明文文件，请删除它。然后使用工具重新创建它。

运行失败，显示 "无法解密"

症状：角色在 "解释机密加载失败" 停止。

可能原因：

- playbook 在没有 `--ask-vault-pass` 或 `--vault-password-file` 的情况下运行。
- 密码与该客户的 OmniCore vault 条目不匹配。

解决方案：

1. 使用 vault 密码再次运行 playbook。

2. 使用 `ansible-vault view hosts/<customer>/group_vars/secrets.yml > /dev/null && echo VAULT-OK` 检查密码。

客户端从 OmniHSS API 收到 401

症状：OmniWeb、OmniCRM 或其他客户端在密钥轮换后从 HSS 日志中记录 `401 Unauthorized`。

可能原因：

- HSS 获得了新密钥，但客户端没有，或反之亦然。

解决方案：

1. 使用相同的 `secrets.yml` 运行 HSS playbook 和客户端的 playbook。
2. 确认两个主机都已重新启动其服务。

相关文档

- 组变量配置 - 其他每个客户的文件位于 `group_vars/`
- 配置参考 - 主机参数
- 概述

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

手册采用层级结构，以便快速、针对性地进行部署：

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

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

快速参考

顶层手册

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`	分组核心	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

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 设置并应用接口 MTU
`firewall.yml`	iptables/nftables 规则
`apt_cache.yml`	本地 APT ���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	进化分组数据网关（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 / MMSc	SMS-over-IP 控制器、SMPP、IMS SMSC（通过 `cscf` 角色以 `cscf_type: smsc_ims` 部署），以及独立的 MMSc（`mmsc` 组，`mmsc` 角色）

5G 核心

通过 `5gc.yml` 部署。每个网络功能由共享的 `5gc_nf` 角色与每个 NF 的 `nf_package` 安装。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 独立网络功能
	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、MGW、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 域
	BSC	基站控制器（`circuit_switched/bsc`）
	MGW	□立媒体网关（`circuit_switched/mgw`）
	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
`voipmonitor.yml`	SIP/RTP 数据包捕获（VoIPmonitor）在 `siptrace` 捕获箱上，以及选择性薄客户端收集器
`grafana.yml`	Grafana 仪表板和警报
`ran_monitor.yml`	RAN 监控集成
`ussd.yml`	HLR 的 USSD 网关的 USSD 回调应用程序（`ussd` 组）。请参见 Circuit-Switched
`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** 供应（`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

```
# 仅分组核心
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
```

限制特定主机

```
# 在一台主机上运行 all.yml
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit mme01

# 在多台特定主机上运行
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit "mme01,hss01"

# 在一个组上运行
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml --limit mme
```

相关文档

- 部署架构 - 整体部署工作流
- 主机文件配置 - 定义基础设施
- 组变量配置 - 自定义
- 实用手册 - 操作工具（健康检查、恢复等）
- 监控与可观察性 - Grafana、Prometheus、警报

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□立运行以执行常见的维护和故障排除任务。

快速参考

剧本	目的
`proxmox.yml`	在 Proxmox 上配置虚拟机和/或 LXC 容器（根据 `deployment_type`）
`proxmox_delete.yml`	删除 Proxmox 上的虚拟机/LXC
`proxmox_restart_hung.yml`	当 SSH 无响应时，通过 Proxmox API 强制重启虚拟机/LXC
`vmware.yml`	在 VMware vSphere 上配置虚拟机
`vmware_delete.yml`	删除 VMware vSphere 上的虚拟机
`health_check.yml`	生成所有服务的全面健康报告
`restore_hss.yml`	从备份恢复 HSS 数据库和/或配置
`seed_hss_replication.yml`	从已填充的对等节点种子一个空的 OmniHSS 节点（初始复制数据加载）
`audit_hss_drift.yml`	审计 OmniHSS 网格的数据漂移（每个参考表的行数 + 内容摘要）；只读
`reconcile_hss_drift.yml`	通过跨���点联合回填缺失行来调和 OmniHSS 网格数据漂移（默认干运行）
`restore_ocs.yml`	从备份恢复 OCS KeyDB、MySQL 和配置
`ip_plan_generator.yml`	生成带有 Mermaid 图的网络文档
`get_ports.yml`	审计所有主机的开放端口和监听服务
`getLocalCapture.yml`	从主机检索数据包捕获文件
`delete_local_user.yml`	从所有主机中删除本地用户帐户

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：

```yaml
all:
  vars:
    vcenter_ip: "vcenter.example.com"
    vcenter_username: "administrator@vsphere.local"
    vcenter_password: "password"
    vcenter_datacenter: "Datacenter"
    vcenter_folder: "OmniCore"
    vcenter_vm_template: "ubuntu-2404-template"
    vhosts:
      esxi-01:
        vcenter_cluster_ip: "192.168.1.10"
        vcenter_datastore: "datastore1"
```

Vultr

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。

行为

1. 计算每个有限目标上的订阅者，以确定哪些是空的。
2. 交互模式：探测有效对等节点（本地 `hss` 组 + `connected_omnihss`）中持有数据的节点，列出它们并提示选择。显式模式：提示输入 `yes` 以确认。
3. 确保源 IP 在目标的 `pg_hba.conf` 中被允许。
4. 删除并重新创建对源的订阅 `WITH (copy_data = false)`（每个链接的 `slot_name` 为 `<target>_from_<source>`）。它流式传输 所有 表，但不执行批量表同步。
5. 从源回填现有 主表 行，使用冲突安全加载（`pg_dump --data-only --inserts --on-conflict-do-nothing --disable-triggers`）。`subscriber_state` 是 包含的：每个订阅者通过 FK 引用一个，因此跳过它会使每个回填的订阅者孤立。瞬态会话表（`pdn_session`、`lte_call`）不 被回填。
6. 报告已种子节点上的订阅者计数。

> 为什么跳过会话表：它们是热表；在实时系统上对它们进行批量 `COPY` 会与并发写入发生冲突（重复键），并使表同步工作者崩溃循环。它们会从流复制和实时流量中重新填充。`subscriber_state` 是不同的：行仅在配置时插入（与订阅者 1:1），之后仅更新，因此 `--on-conflict-do-nothing` 的回填是安全的。没有它，目标的订阅者指向的状态行永远不会到达，因为逻辑复制会静默丢弃对缺失行的更新[illegible]

> 重要：始终对目标节点使用 `--limit`。在交互模式下，非空目标会自动跳过。有关完整工作流程，请参见 OmniHSS 复制与节点操作。

HSS 漂移审计

文件：`util_playbooks/audit_hss_drift.yml`（[illegible]立）和 `roles/omnihss/tasks/audit_drift.yml`（共享实现）。

报告 OmniHSS 主动-主动网格中的 数据漂移。逻辑复制订阅使用 `copy_data = false` 创建，因此在网格链接之前（种子时间差异）或通过直接/未复制写入的数据不会被调和，也不会报警。在数据不同的情况下，复制显示 0 延迟，而每个站点的 Prometheus 无法看到，因为网格跨越两个站点。

审计连接到每个网格成员（本地 `hss` 组 + `connected_omnihss` 对等节点），并且对于每个参考/配置表，比较确切的行数 和完整内容的无序 md5。它还比较 `schema_migrations`（计数 + 最大版本）以捕获模式漂移。热状态表（`subscriber_state`、`pdn_session`、`lte_call`）被排除在外：它们在实时流量下合法不同。除了跨节点比较之外，还包括一个结构 健康检查：`__orphaned_subscribers__`，每个节点缺少 `subscriber_state` 行的订阅者计数。任何节点上的非零计数都会使审计失败，即使每个节点都[illegible]意。一个复制 `subscriber` 而不带 `subscriber_state` 的种子会使每个节点相同地孤立，而纯跨节点比较永远无法看到。比较集中在一个运行者上，该运行者访问每个对等节点的 Postgres，因此一次运行覆盖整个跨站点网格，并且是 `--limit` 安全的。

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。

- 写入在每个目标上以 postgres 超级用户身份本地运行（用于 `--disable-triggers`）；对等节点仅作为应用用户读取。
- 对称联合，而非生产权威：在某个站点运行时，会将 □ 站点的行□□□到其他站点。它 不 将生产视为主控，因此在实验室清单上运行会将实验室行传播到生产。如果生产必须是真实来源，则从生产清单驱动（或请求权威模式）。
- 跨站点：节点只能在剧本中作为 □□，因此针对 每个 站点的清单运行此操作，以覆盖每个节点。最后的重新扫描报告仍需运行的任何内容。

```
# 交互式（默认）：写入前询问
ansible-playbook -i hosts/<inv>/host_files/<site>.yml
util_playbooks/reconcile_hss_drift.yml
# 仅查看，从不写入
ansible-playbook -i ... util_playbooks/reconcile_hss_drift.yml -e
dry_run=true
# 非交互式应用（CI）
ansible-playbook -i ... util_playbooks/reconcile_hss_drift.yml -e
assume_yes=true
```

> 首先运行 `audit_hss_drift.yml` 以查看漂移情况，然后在调和后再次运行（针对两个清单）以确认收敛。有关更多信息，请参见 OmniHSS 复制与节点操作。

OCS 恢复

文件：`util_playbooks/restore_ocs.yml`

从备份文件恢复 OCS (CGrateS)。通过恢复到一个节点并允许复制同步到其他节点来处理多主 KeyDB 复制。

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

恢复过程

1. 停止所有 OCS 节点上的 CGrateS 和 KeyDB
2. 清除 AOF 文件以防止与恢复数据发生冲突
3. 将 KeyDB RDB 恢复到第一个节点，启动 KeyDB，让复制同步
4. 恢复 MySQL StoreDB（可选）
5. 恢复 `/etc/cgrates` 配置（可选）

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
```