Omnitouch □ Ansible 部署介绍

概述

Omnitouch □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□络解决方案（4G/5G）。本文档提供了我们如何利用 Ansible 来协调复杂的通信部署的概述。

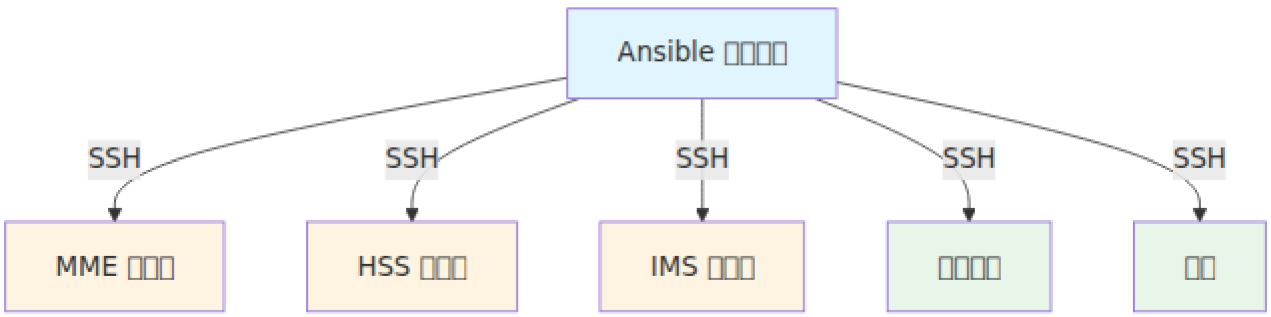
什么是 **Ansible**?

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□ 系统。每���客户部署都有一个描述以下内容的主机文件：

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

主机文件是您将用于定义□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

这些由 ONS 团队定义，虽然您可以编辑它们，但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: □署 EPC 核心
  hosts: mme
  roles:
    - common
    - omnimme
```

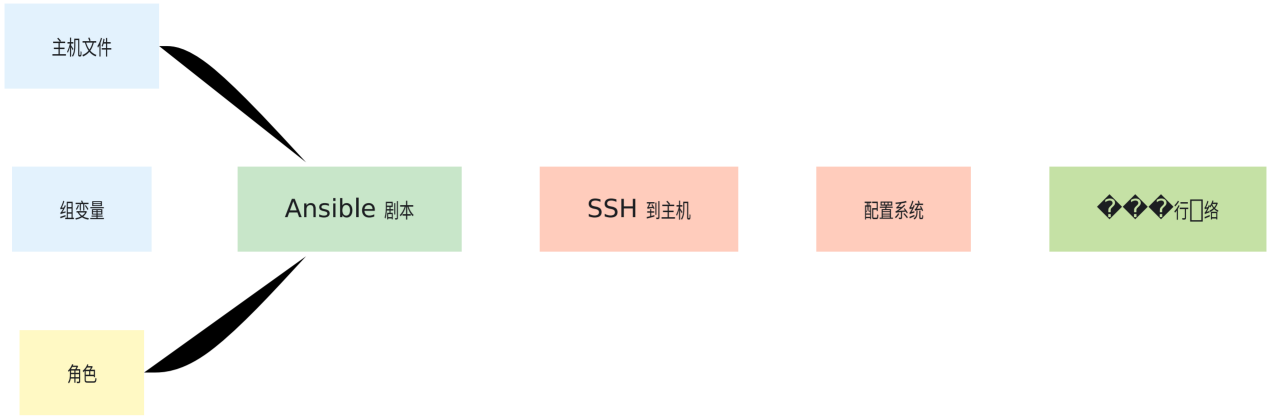
我们基本上将这些用作角色的分组。

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

创建一个描述您的络拓扑的主机文件：

规划注意事项：在定义基础设施之前，请查看 IP 规划标准，以获取有关络分段、IP 地址分配和子组织的指导。

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。

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 - 在此处添加配置参考的完整列表链接

3. 运行剧本

部署□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□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□关键组件

OmniCore（4G/5G 分组核心平台）

- **OmniHSS** - 家庭用户服务器
- **OmniSGW** - 服务□关（控制平面）
- **OmniPGW** - 分组□关（控制平面）
- **OmniUPF** - 用户平面功能

- **OmniDRA** - Diameter 路由代理
- **OmniTWAG** - 受信 WLAN 访问□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** - □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

包管理

我们使用混合包分发模型：

预编译 APT 包

所 Omnitouch 软件作为 Debian 包（`.deb` 文件）分发：

- 在我们的 CI/CD 管道中从源代码构建
- 版本控制和测试
- 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

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

参见：许可证服务器

这种方法好处

可重复性

相同的 Ansible 剧本可以部署：

- 开发实验室
- 测试环境
- 生产络

- 客户站点

一致性

每次部署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
- 部署前进行审核
- 如有需要可回滚

无复杂性□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

在几个小时内部署完整的蜂窝□络，而不是几天或几周。

入门

先决条件

在���行 Ansible 剧本之前，您需要设置 Python 虚拟环境并安装所需的依赖项。

1. 创建 **Python** 虚拟环境

为 Ansible 部署创建一个隔离的 Python 环境：

```
python3 -m venv .venv
```

2. 激活虚拟环境

激活虚拟环境：

```
source .venv/bin/activate
```

在 Windows 上，使用：

```
.venv\Scripts\activate
```

3. 安装所需软件包

从 requirements.txt 文件中安装所有依赖项：

```
pip install -r requirements.txt
```

这将安装 Ansible 及所有必要的 Python 包，以实现 Omnitouch 部署自动化。

注意：在运行 Ansible 命令时，请保持虚拟环境处于激活状态。完成后，您可以通过运行 `deactivate` 来停用它。

部署步骤

1. 查看 主机文件配置，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 以进行包管理
4. 查看 部署架构，了解所有内容如何结合在一起
5. 部署！

下一步

- IP 规划标准 - 规划您网络架构和 **IP** 分配
- 主机文件配置 - 学习如何定义您的网络拓扑
- APT 缓存系统 - 理解包分发
- 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 - 自定义您的部署

- 实用剧本 - 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

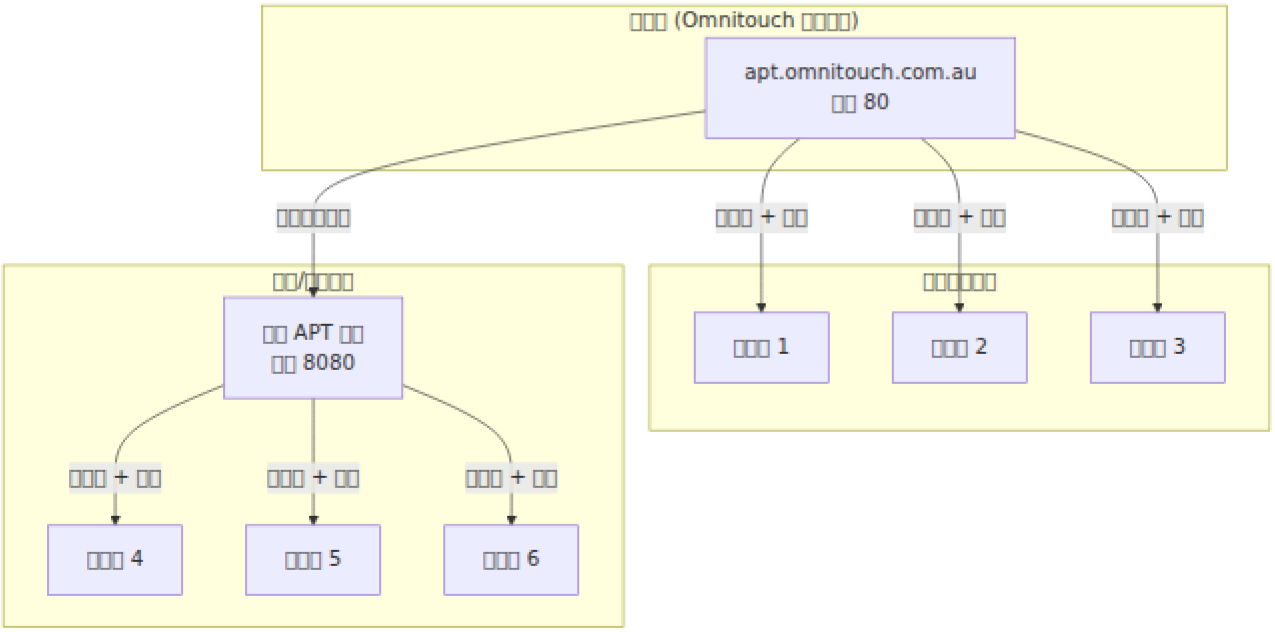
Omnitouch APT 系统为所有部署提供软件包分发。提供两种类型的内容：

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）

支持两种部署模型：

1. 直接访问 — 虚拟机直接从 `apt.omnitouch.com.au` 拉取软件包
2. 本地缓存镜像 — 本地服务器从 Omnitouch 同步并为虚拟机提供软件包（用于离线/隔离部署）

架构



提供□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>/`
Ubuntu 软件包	缓存的 Ubuntu 软件包及其所有依赖项	`/<distro>/pool/main/`
GitHub 发布	预构建的二进制文件（Prometheus、Grafana、Homer 等）	`/releases/<org>/<repo>/`
源代码压缩包	Web 应用的源代码档案（CGrateS_UI、speedtest）	`/repos/`
第三方软件包	Galera、FRR、InfluxDB、KeyDB 等	`/releases/<vendor>/`

配置变量

两个独立的变量集控制软件包分发。理解它们的用途对于正确配置至关重要。

变量用途

变量集	目	用于
`apt_repo`	配置 APT 软件包源	`/etc/apt/sources.list` 和 `/etc/apt/sources.list.d/*.list`
`remote_apt_*`	配置二进制下载 URL	从 `/releases/` 路径下载文件（Node Exporter、Zabbix、Nagios 等）

每个变量集使用时机

场景	**APT 源（`apt_repo`）**	二进制下载（**`remote_apt_*`**）
`use_apt_cache: true`	使用 `apt_repo.apt_server`	使用 `apt_repo.apt_server`
`use_apt_cache: false`	使用带凭据的 `apt_repo.*`	使用带凭据的 `remote_apt_*`

当 **`use_apt_cache: false`** 时，两个变量集都是必需。

选项 1：直接访问

对于具有互联连接的部署，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
- **FQDN** 用于 APT 服务器

防火墙要求：必须允许对以下 Omnitouch IP 范围的出站访问：

网	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** 基础设施服务：

服务	端口	协议	目
APT 服务器	80	TCP	软件包下载
APT 服务器	53	TCP/UDP	`apt.omnitouch.com.au` 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 HTTP (TCP/80)、NTP (UDP/123) 和 DNS (TCP+UDP/53) 流量被允许到 Omnitouch IP 范围。

配置

```yaml
all:
  vars:
    use_apt_cache: false

    # APT 件包源配置
    # 配置 /etc/apt/sources.list 以用于 apt install 命令
    apt_repo:
      apt_server: "apt.omnitouch.com.au"
      apt_repo_username: "your-username"
      apt_repo_password: "your-password"

    # 二进制下载配置
    # 用于从 /releases/ 路径下载文件
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_port: 80
    remote_apt_protocol: "http"
    remote_apt_user: "your-username"
    remote_apt_password: "your-password"
```

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	-	APT 源的 HTTP 基本身份验证用户名
`apt_repo.apt_repo_password`	字符串	是	-	APT 源的 HTTP 基本身份验证密码

二进制下载（`remote_apt_*`）

参数	类型	必需	默认	描述
`remote_apt_server`	字符串	是	-	二进制下载的服务器主机名或 IP
`remote_apt_port`	整数	否	`80`	二进制下载的服务器端口
`remote_apt_protocol`	字符串	否	`http`	协议（`http` 或 `https`）
`remote_apt_user`	字符串	是	-	下载的 HTTP 基本身份验证用户名
`remote_apt_password`	字符串	是	-	下载的 HTTP 基本身份验证密码

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）

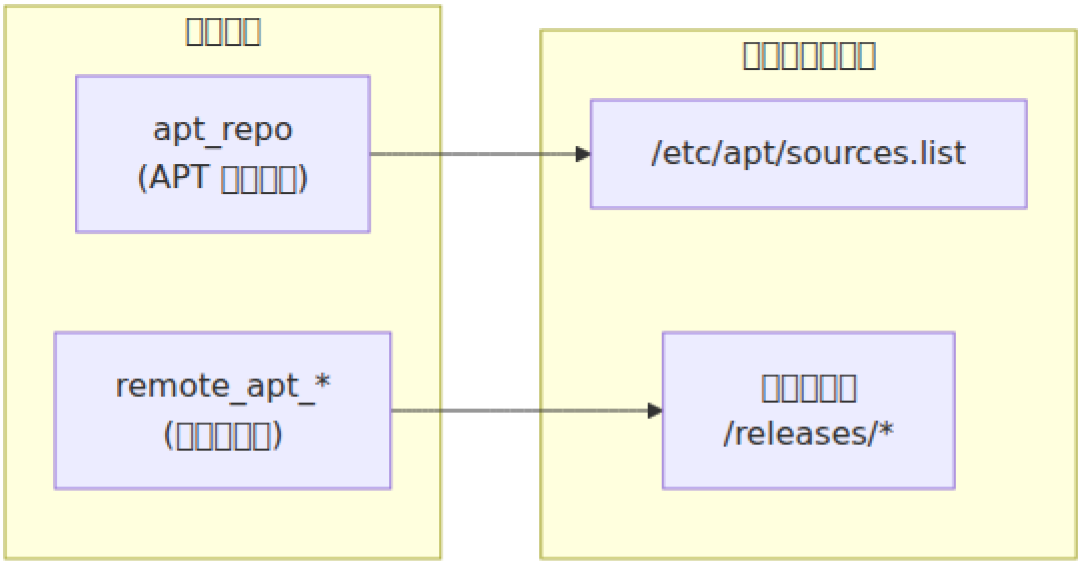
APT 软件包源（配置在 `/etc/apt/sources.list`）：

```
deb [trusted=yes] http://{apt_repo_username}:
{apt_repo_password}@{apt_server}/ noble main
```

二进制下载（由 Ansible `get_url` 任务使用）：

```
http://{remote_apt_user}:
{remote_apt_password}@{remote_apt_server}:
{remote_apt_port}/releases/prometheus/node_exporter/node_exporter-
1.8.1.linux-amd64.tar.gz
```

工作原理

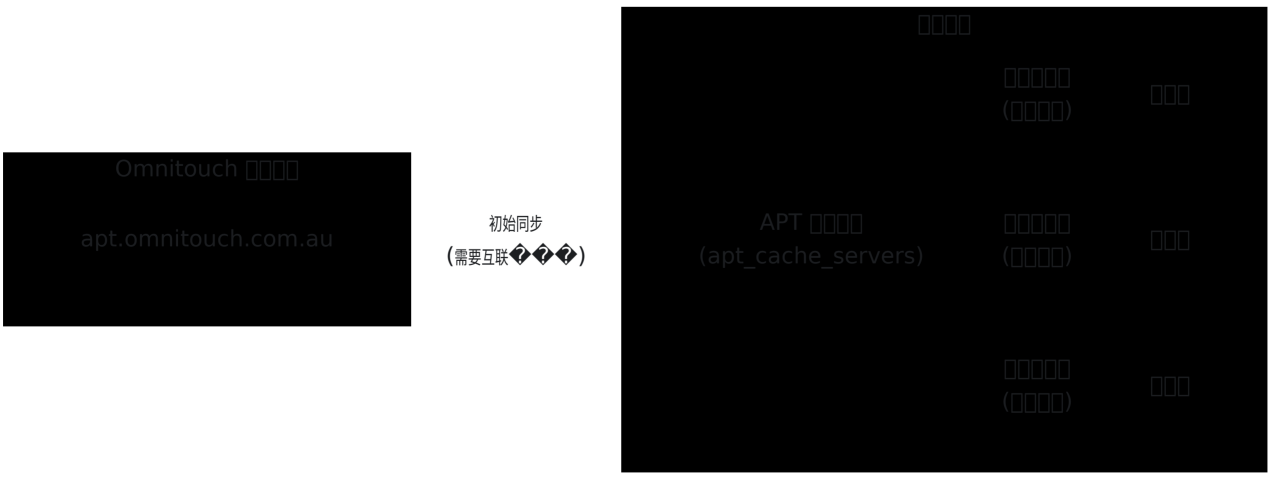


虚拟机通过 HTTP 基本身份验证进行 APT 软件包和二进制下载的身份验证。Ubuntu 系统软件包也从 Omnitouch 服务器提供（预缓存），因此虚拟机无需访问 Ubuntu 镜像。

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 hosts 文件中定义缓存服务器及其仓库配置：

```yaml
apt_cache_servers:
  hosts:
    customer-apt-cache:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # 缓存服务器从经过身份验证的仓库同步件包
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_port: 80
    remote_apt_protocol: "http"
    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: 自动推导为 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_*` 凭据从 `apt.omnitouch.com.au:80` 同步软件包
- 所有其他主机：自动推导 `apt_repo.apt_server: "192.168.1.100"`，并在不需要凭据的情况下从端口 `8080` 拉取缓存

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	自动推导	本地缓存服务器 IP。如果未指定，则自动从第一个 `apt_cache_servers` 主机推导。
`apt_repo.apt_repo_username`	字符串	否	-	使用缓存时不需要（不需要身份验证）
`apt_repo.apt_repo_password`	字符串	否	-	使用缓存时不需要（不需要身份验证）

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：

参数	类型	必需	默认	描述
`remote_apt_server`	字符串	是	-	从中同步的 Omnitouch APT 服务器
`remote_apt_port`	整数	否	`80`	Omnitouch APT 服务器端口
`remote_apt_protocol`	字符串	否	`http`	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`
`apt_cache_port`	整数	否	`8080`	本地缓存服务器监听的端口

URL 模式（缓存模式）

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

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

二进制下载（由 Ansible `get_url` 任务使用）：

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

访问缓存时不需要凭据——它使用 `[trusted=yes]` APT 配置。

部署缓存

1. 准备缓存服务器（虚拟机或 LXC 容器，磁盘至少 50 GB）
2. 运行缓存设置剧本：

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

3. 通过浏览 **`http://192.168.1.100:8080/`** 验证缓存

同步□内容

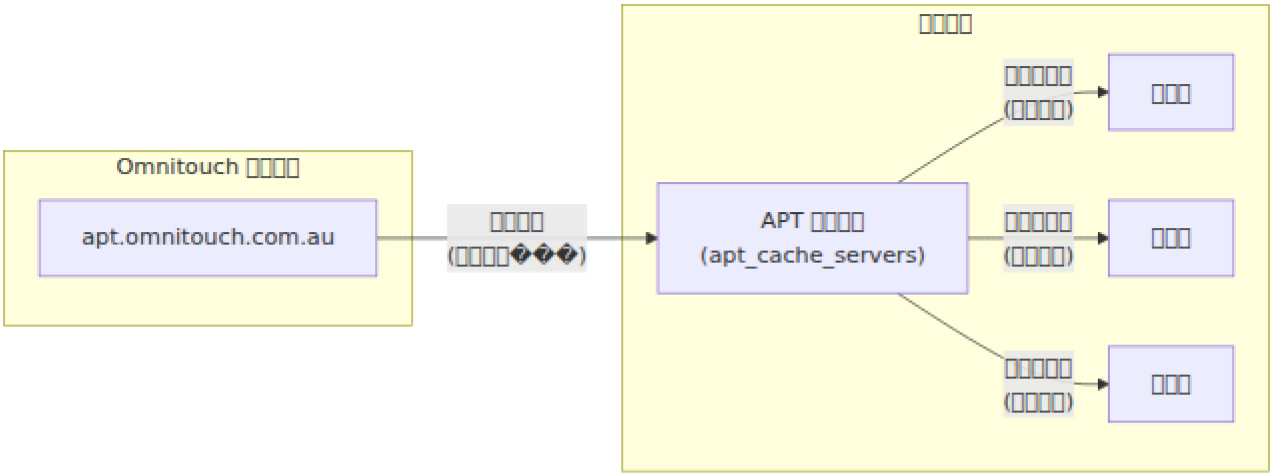
缓存镜像使用递归 wget 下载从 Omnitouch APT 服务器同步 所有内容:

同步□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 软件包及其所有依赖项
`/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□连接的情况下提供所有软件包。

工作原理



缓存镜像使用 `wget --recursive` 和 HTTP 基本身份验证从 Omnitouch APT 服务器下载所有内容。后续同步仅下载新/更改的文件（时间戳）。

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`（除非明确覆盖）
2. 从第一个缓存服务器的 `ansible_host` IP 推导 `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 Omnitouch 仓库同步内容
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_user: "your-username"
    remote_apt_password: "your-password"
```

自动发生□事情：

- 所有主机（除了缓存服务器）都获得 `use_apt_cache: true`
- 所有主机（除了缓存服务器）都获得 `apt_repo.apt_server: "192.168.1.100"`
- 所有主机从 `http://192.168.1.100:8080/` 拉取，无需凭据
- 缓存服务器从 `http://your-username:your-password@apt.omnitouch.com.au/` 同步软件包

覆盖自动行为

要强制直接访问，即使定义了缓存服务器：

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

    apt_repo:
      apt_server: "apt.omnitouch.com.au"
      apt_repo_username: "user"
      apt_repo_password: "pass"

    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_user: "user"
    remote_apt_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。

```yaml
all:
  vars:
    use_apt_cache: false

    # APT 件包源 - 所有主机使用
    apt_repo:
      apt_server: "apt.omnitouch.com.au"
      apt_repo_username: "user"
      apt_repo_password: "pass"

    # 二进制下载 - 所有主机使用
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_port: 80
    remote_apt_protocol: "http"
    remote_apt_user: "user"
    remote_apt_password: "pass"
```

结果：所有主机生成 `deb [trusted=yes] http://user:pass@apt.omnitouch.com.au/ noble main`

场景 2：在 hosts 文件中定义 APT 缓存服务器（自动）

缓存服务器在您的清单中，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:
    # 缓存服务器从经过身份验证的仓库同步件包
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_port: 80
    remote_apt_protocol: "http"
    remote_apt_user: "user"
    remote_apt_password: "pass"

# 在 all: vars: 中无需配置：
# 所有内容自动从 apt_cache_servers 组推导
```

结果：

- 缓存服务器：从 `http://user:pass@apt.omnitouch.com.au:80/` 同步
- 所有其他主机：生成 `deb [trusted=yes] http://192.168.1.100:8080/noble noble main`（无需凭据）

场景 3：在 hosts 文件中未定义远程 APT 缓存（手动）

缓存服务器存在于其他地方，并且已设置（不由您的 Ansible 管理）。

```
all:
  vars:
    use_apt_cache: true

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

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

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

完整示例

以下是一个完整的工作示例，展示了缓存服务器配置与多个应用主机：

```yaml
# APT 缓存服务器组
apt_cache_servers:
  hosts:
    customer-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:
    # 缓存服务器从经过身份验证的仓库同步□件包
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_port: 80
    remote_apt_protocol: "http"
    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□事情：

1. 缓存服务器 (`10.179.1.114`)：
   - 使用其 `vars:` 部分中的 `remote_apt_*`
   - 从 `http://customer-username:customer-secure-token@apt.omnitouch.com.au:80/` 下载所有软件包
   - 通过 nginx 在端口 8080 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
```

这会增量同步 Omnitouch APT 服务器中的所有内容：

- 新的 Omnitouch 软件包版本
- 新的 Ubuntu 软件包及其依赖项
- 新的 GitHub 发布
- 更新的源代码压缩包

同步使用 `wget --timestamping`，因此跳过现有未更改的文件，使重新同步速度更快。

注意：Omnitouch APT 服务器（`apt.omnitouch.com.au`）是所有软件包的唯一真实来源。首先在 APT 服务器上运行 `services/apt.yml` 以构建/更新软件包，然后在缓存镜像上运行 `services/apt_cache.yml` 以同步。

故障排除

APT 更新失败，401 未授权

症状：

```
无法获取 http://10.179.1.115:80/noble/dists/noble/main/binary-amd64/Packages  401  未授权
```

可能原因：

- `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
- 在直接访问中使用缓存凭据或反之亦然

解决方案：

1. 检查配置范围：确保带凭据的 `apt_repo` 定义在 `apt_cache_servers: vars:` 中，而不是在 `all: vars:` 中
2. 验证缓存模式：使用缓存时，主机应连接到缓存服务器（端口 8080），而不是仓库（端口 80）
3. 检查生成源：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（凭据放错地方）：`deb [trusted=yes] http://user:pass@10.179.1.115:80/noble noble main`
4. 验证凭据是否正确以适应您的部署模式
5. 确认您的公共 IP 是否在 Omnitouch 上列入白名单（如果使用直接访问）

二进制下载失败（Node Exporter、Zabbix 等）

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

可能原因：

- `remote_apt_*` 变量未配置
- `remote_apt_user` 或 `remote_apt_password` 不正确
- 当 `use_apt_cache: false` 时缺少 `remote_apt_server`

解决方案：

1. 确保所有 `remote_apt_*` 变量已定义
2. 验证凭据是否与 Omnitouch 提供的匹配
3. 检查 `remote_apt_server` 是否指向正确的主机

缓存服务器无法同步

症状：缓存服务器剧本无法下载软件包

可能原因：

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

解决方案：

1. 验证缓存服务器是否可以在端口 80 上访问 `apt.omnitouch.com.au`
2. 检查 `remote_apt_*` 凭据
3. 检查防火墙规则以确保出站访问

相关文档

- 主机文件配置 — 清单和变量配置
- 配置参考 — 完整参数参考

- 部署架构 — 整体系统架构
- Proxmox 部署 — 将缓存服务器部署为 LXC 容器

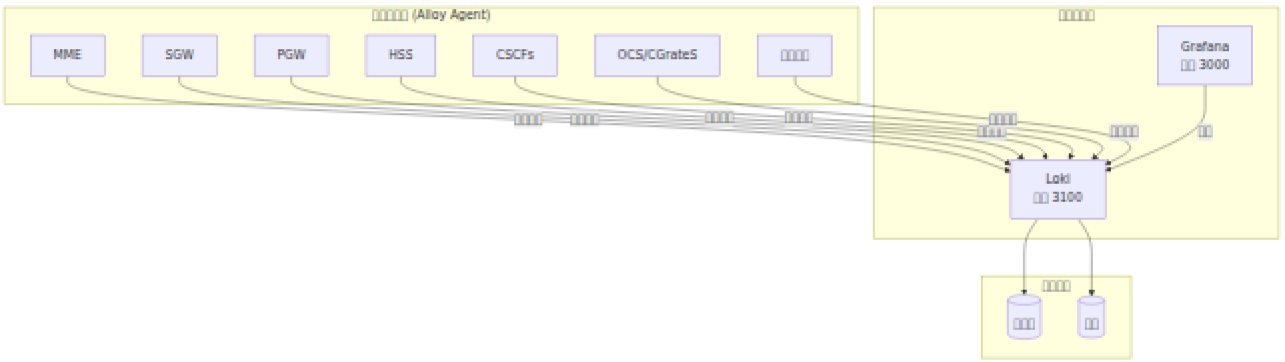
集中日志记录

概述

OmniCore 包含一个集中日志记录系统，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：

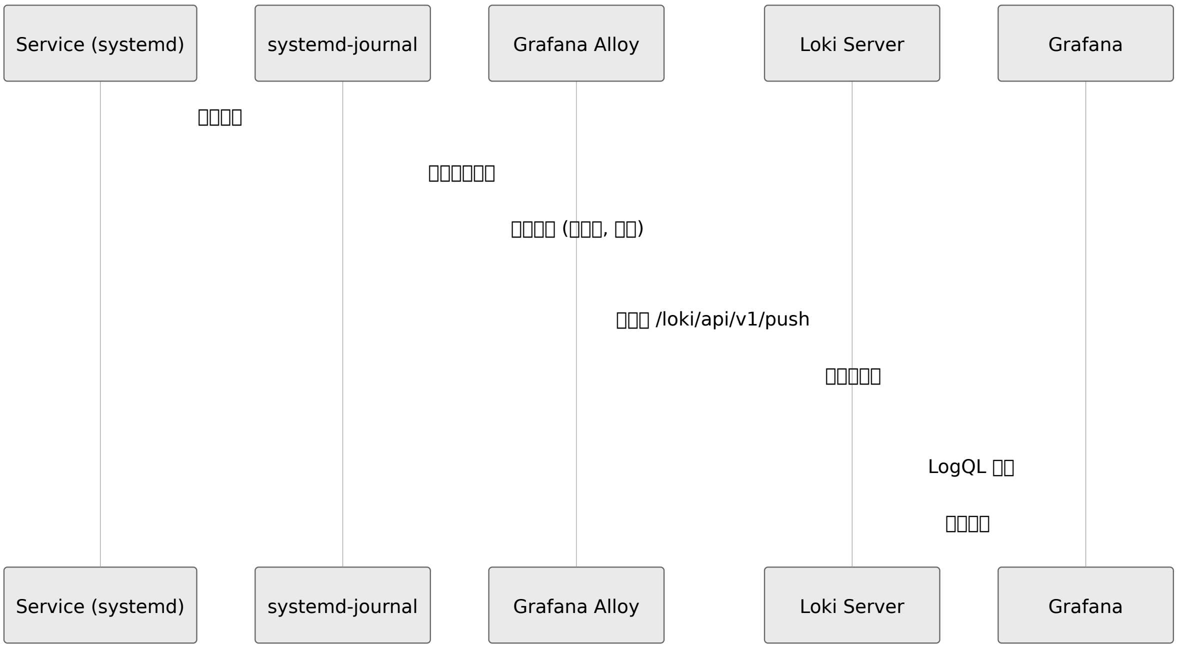
- **Grafana Alloy** — 部署在所有虚拟机上的日志收集代理
- **Grafana Loki** — 监控主机上的日志聚合和存储服务器
- **Grafana Dashboards** — 针对每种组件类型的预构建仪表板

架构



□□□□

□□□□□□



Service (systemd)
systemd-journal
Grafana Alloy
Loki Server
Grafana

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

库存组	组件标签	收集□服务
`mme`	`mme`	omnimme, open5gs
`sgw`	`sgw`	omnisgwc, open5gs
`pgw`	`pgw`	omnipgwc, open5gs
`upf`	`upf`	omniupf, frr
`hss`	`hss`	omnihss, pyhss, mysql
`dra`	`dra`	omnidra, freediam
`pcscf`, `scscf`, `icscf`	`cscf`	kamailio, rtpengine
`applicationserver`	`applicationserver`	freeswitch, omnitas
`ocs`	`ocs`	cgrates, ocs, pcef, keydb
`cgrates`	`cgrates`	cgrates, mysql
`omnimessage`	`omnimessage`	omnimessage, kamailio
`dns`	`dns`	named, bind
`omnicrm`	`omnicrm`	omnicrm, nginx, mysql
`monitoring`	`monitoring`	prometheus, grafana, loki, nginx

日志标签

每个日志条目包含以下标签以便过滤和查询：

标签	描述	示例
`job`	源虚拟机的主机名	`customer-mme01`
`hostname`	与 job 相同（兼容性）	`customer-mme01`
`component`	来自库存组的组件类型	`mme`, `cscf`, `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. 库存中存在 `monitoring` 组
2. 公共角色在目标主机上运行

基本功能不需要额外配置。

库存要求

```yaml
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（接收和存储日志）
- 所有其他主机：运行 Alloy 代理（收集和发送日志）

保留配置

Loki 配置了双重保留限制：

设置	默认值	描述
基于时间的保留	7 天	超过 7 天的日志将被删除
基于大小的保留	50 GB	当存储超过 50 GB 时，删除最旧的日志

首先达到的限制将触发日志删除。

要自定义保留，请在库存中覆盖以下变量：

```
all:
  vars:
    loki_retention_period: "168h"  # 7 天的小时数
```

Alloy 缓冲配置

当 Loki 无法访问时，Alloy 会在本地缓冲日志。缓冲区有限制以防止磁盘耗尽：

设置	默认值	描述
WAL 最大大小	500 MB	缓冲日志的最大磁盘空间
WAL 最大段年龄	1 小时	超过 1 小时的最旧缓冲日志将被丢弃

当缓冲区满时，最旧的日志将被丢弃以腾出空间给新条目。

Grafana 仪表板

针对每种组件类型的预构建仪表板会自动提供。

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 (Kamailio) 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 ���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

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 **CSCFs** 中的错误：

```
{component="cscf"} |~ "(?i)error"
```

按 **systemd** 单元过滤：

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

组合过滤器：

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

日志速率查询

每个组件日志量：

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

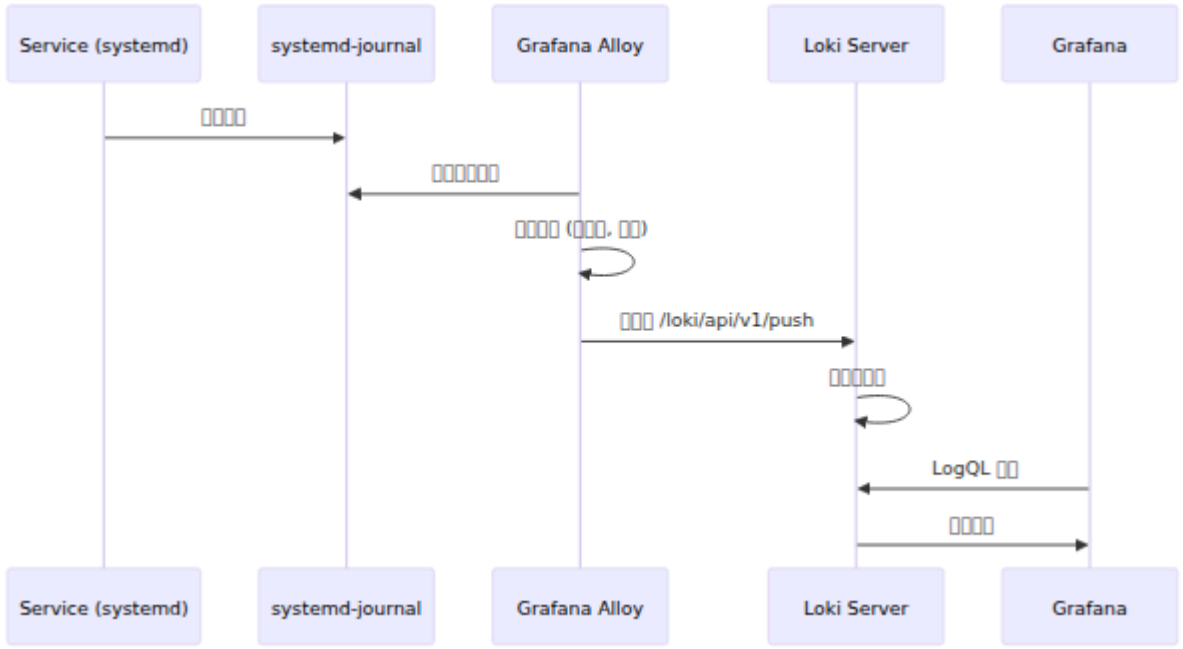
CSCF 错误率：

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

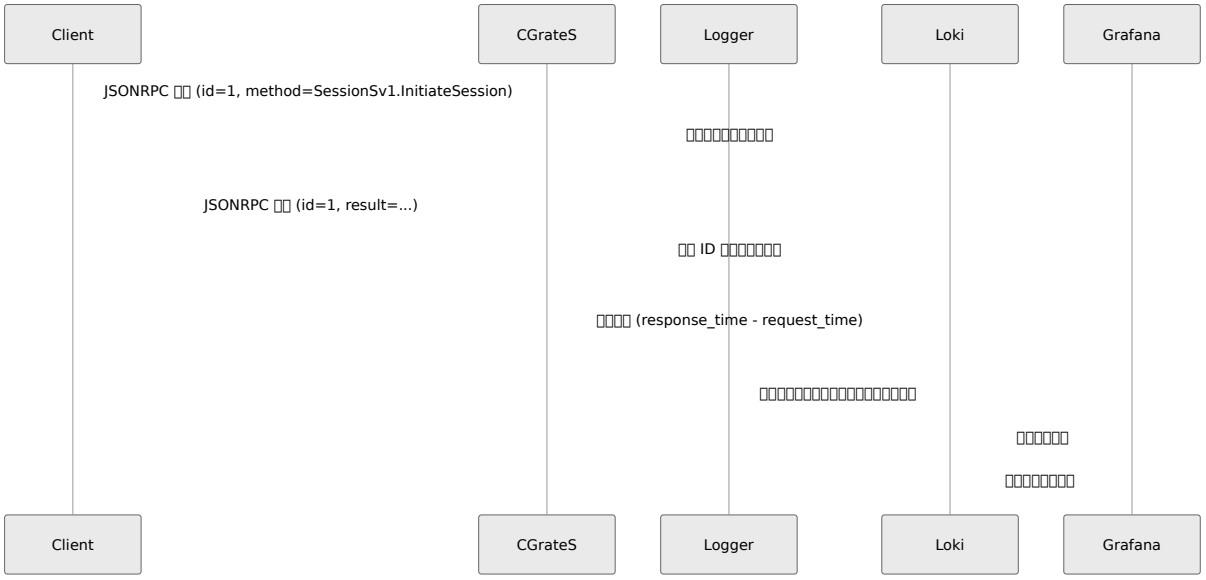
CGrateS JSONRPC 日志记录

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

□□


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

□□/□□□□


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

□□□□

□□ CGrateS □□□□□□□□□□

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

记录器会自动过滤掉：

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

日志格式

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

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

日志字段

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

Grafana 仪表板

OCS / CGrateS 日志 仪表板包括专用面板用于 JSONRPC 分析。

按方法 **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
- 指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，包含：

- 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 仪表板提供了集中工具用于故障排除 API 调用并关联请求与响应。

用例：查找失败□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请求/响应

每个日志条目显示���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简单。

仪表板变量

变量	目	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	所有，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 **RPC** 延迟：时间序列显示每个方法的延迟。悬停以查看具体值。

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 **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（用于图形）：

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

查找慢请求（延迟 **> 100ms**）：

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

特定方法查询

CDR 处理：

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

会话管理：

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

账户操作：

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

服务管理

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

检查服务状态：

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

查看服务日志：

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

重启服务：

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

故障排除

没有 **JSONRPC** 日志出现

症状：JSONRPC 流量面板没有数据显示

可能原因：

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

解决方案：

1. 检查记录器服务状态：

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

2. 验证 ngrep 是否已安装：

```
which ngrep
```

3. 检查日志文件是否正在写入：

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

4. 验证 Alloy 是否正在收集该文件：

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

延迟图表没有数据显示

症状：按方法的 JSONRPC 延迟面板显示“没有数据”

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，但很少有记录的完成

可能原因：

- 没有响应的请求（客户端断开连接）
- 在请求之前捕获响应（数据包顺序）
- ��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. 检查客户端和 CGrateS 之间的□络问题
3. 验证 CGrateS 是否对请求作出响应

文件日志

除了 systemd 日志，Alloy 还收集特定的日志文件：

FreeSWITCH 日志（应用服务器）

路径	作业标签
`/var/log/freeswitch/*.log`	`freeswitch`

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

日志未出现在 Grafana 中

症状：在 Loki 仪表板中没有日志可见

可能原因：

- 源主机上 Alloy 服务未运行
- 监控主机上 Loki 服务未运行
- 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 高磁盘使用率

症状：`/var/lib/alloy` 消耗大量磁盘空间

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 停止工作，日志会在本地缓冲最多 500 MB
3. 一旦 Loki 可访问，缓冲的日志会自动发送
4. 如果缓冲区已满，最旧的日志将被丢弃（设计如此）

Loki 存储已满

症状：Loki 停止接受日志，监控服务器磁盘已满

可能原因：

- 日志量超过 50 GB 的保留限制
- 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. 验证压缩器是否正在运行（检查 Loki 日志）
3. 如果需要，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. ��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

配置参考

概述

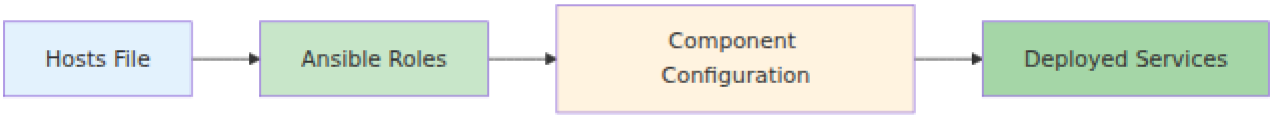
本文档提供了通过主机文件配置 OmniCore 部署的全面参考。配置主要在主机清单文件中定义，对于现代部署，所需的 group_vars 覆盖最小。

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。角色默认值处理大多数设置，group_vars 仅用于特定自定义。

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：

- 络分段策略
- IP 地址分配
- 子组织
- 公共 IP 处理

常见主机参数

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

服务特定标志

```
cdrs_enabled: True                  # 启用 CDR 生成
in_pool: False                      # 排除在负载均衡池外
online_charging_enabled: False      # 启用 OCS 集成
recording: True                     # 启用通话录音 (AS)
populate_crm: False                 # 用初始数据填充 CRM
```

全局变量 (all:vars)

`all:vars` 部分包含部署范围的设置。现代部署使用最小的全局变量，大多数配置在角色默认值中。

必需全局变量

身份验证与访问

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

替代方案：使用 SSH 密钥而不是密码：

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

客户身份

```
customer_name_short: omnitouch
customer_legal_name: "YKTN Lab"
site_name: YKTN
region: AU
TZ: Australia/Melbourne
```

PLMN 配置

```
plmn_id:
  mcc: '001'                # 移动国家代码 (3 位)
  mnc: '01'                 # 移动网络代码 (2-3 位)
  mnc_longform: '001'       # 零填充的 MNC (始终 3 位)

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

目□: 唯一标识您的移动□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: □络名称在 UE 设备的设置 > 移动□络中显示。

DNS 配置

```
netplan_DNS: False                # 使用 systemd-resolved 而不是 netplan
DNS
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 将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 仓库配置

自动默认值：当定义了 `apt_cache_servers` 组并包含主机时：

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

```
# 手动配置（如果存在 apt_cache_servers 组则可选）
use_apt_cache: True                # 使用本地 APT 缓存而不是直接访问仓库

apt_repo:
  apt_server: "10.10.1.114"        # APT 缓存服务器或仓库服务器
  # 仅在 use_apt_cache: False 时需要凭据
  # apt_repo_username: "omni"
  # apt_repo_password: "omni"

# 二进制下载和缓存同步配置
# 用于：(1) 当 use_apt_cache: false 时从 /releases/ 下载二进制文件
#       (2) 当 use_apt_cache: true 时从 Omnitouch 同步缓存服务器
remote_apt_server: "apt.omnitouch.com.au"
remote_apt_user: "omni"
remote_apt_password: "omni"
```

请参见：APT 缓存系统

许可证服务器

```
license_server_api_urls: ["https://10.10.2.150:8443/api"]
license_enforced: true
```

请参见：许可证服务器

MME 设置

```
mme_dns: False                     # 启用 MME DNS 解析
```

SAEGW 设置

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

IMS 设置

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

RAN 监控配置

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

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

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

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

□□□□□

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

漫游 **DNS** 服务器

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

□□□□□□□□□

Proxmox

```
proxmoxServers:
  customer-prxmx01:
    proxmoxServerAddress: 10.10.0.100
    proxmoxServerPort: 8006
    proxmoxRootPassword: password
    proxmoxApiTokenName: AnsibleToken
    proxmoxApiTokenSecret: "token-secret"
    proxmoxTemplateName: ubuntu-24.04-cloud-init-template
    proxmoxTemplateId: 9000
    proxmoxNodeName: pve01

# □□ Proxmox □□
proxmoxServerAddress: 10.10.0.100
proxmoxServerPort: 8006
proxmoxNodeName: 'pve01'
proxmoxLxcOsTemplate: 'local:vztmpl/ubuntu-24.04-standard_24.04-
2_amd64.tar.zst'
proxmoxApiTokenName: DocsTest
proxmoxLxcCores: 8
proxmoxLxcDiskSizeGb: 20
proxmoxLxcMemoryMb: 64000
proxmoxLxcRootFsStorageName: SSD_RAID0
proxmoxLxcBridgeName: vmbr0
proxmoxTemplateName: "ubuntu-24.04-cloud-init-template"
proxmoxStorage: SSD_RAID0
vLabNetmask: 24
PROXMOX_API_TOKEN: "token-secret"
vlabRootPassword: password
vLabPublicKey: "ssh-rsa AAAAB3..."
mask_cidr: 24
```

VMware vCenter

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

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

netmask: 255.255.255.0
```

相关文档

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

产品文档

有关详细的操作指南和高级配置：

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

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

部署架构概述

概述

本文档提供了Omnitouch□络服务的蜂窝□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□络。

请参阅IP规划标准以获取详细的组件放置、IP地址分配指南和公共IP处理。

完整部署示例

0. 基础设施准备（可选）

对于Proxmox部署，在配置之前准备虚拟机/LXC：

```
# 在Proxmox上□署虚拟机
ansible-playbook -i hosts/Customer/hosts.yml
util_playbooks/proxmox.yml

# 或者□署LXC容器（仅限实验/测试）
ansible-playbook -i hosts/Customer/hosts.yml
util_playbooks/proxmox_lxc.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`文件夹是我们可以存储在主机、站点或网络级别所需的任何配置覆盖的地方。

例如，您将有一个包含您的OmniMessage SMSc配置的文件夹，您的TAS连接的SIP中继将在这里，所有的Diameter路由逻辑等等。

参见：组变量配置

3. 软件包分发（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或直接的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"]
license_enforced: true
```

参见：许可证服务器

5. 执行部署

可以通过运行`services/twag.yml`来单独部署组件，例如，但`services/all.yml`将处理所有内容，您可以使用`--limit=myhost`或`--limit=mmee,sgw`等来限制我们正在处理的主机。

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

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

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
- 集中日志记录 - Loki和Alloy日志收集

产品文档

有关配置每个组件的详细信息：

- **OmniCore**（4G/5G数据核心）：
  https://docs.omnitouch.com.au/docs/repos/OmniCore
  - OmniHSS, OmniSGW, OmniPGW, OmniUPF, OmniDRA, OmniTWAG
- **OmniCall**（语音与消息）：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/

组变量配置

概述

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

这里是您特定客户的配置所在 - 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内容 - 它位于 `group_vars` 中。

位置: `hosts/{Customer}/group_vars/`

工作原理

Ansible 角色有默认的配置模板。要针对特定部署进行自定义，请将您的自定义文件放在 `group_vars` 中，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） → □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. 使用 Jinja2 进行模板化（可以使用 `{{ inventory_hostname }}`、`{{ plmn_id.mcc }}` 等）
4. 部署到 `/etc/omnimessage/runtime.exs`
5. 重启服务

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

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

示例 2：配置文件集合（OmniTAS 网关和拨号计划）

某些组件使用配置文件目录。

文件结构

```
hosts/Customer/
└── group_vars/
    ├── gateways_prod/                # SIP 网关配置
    │   ├── gateway_carrier1.xml
    │   ├── gateway_carrier2.xml
    │   └── gateway_emergency.xml
    ├── gateways_lab/                 # 实验室网关
    │   └── gateway_test.xml
    ├── dialplan/                     # 呼叫路由规则（默认）
    │   ├── mo_dialplan.xml           # 移动发起（外拨）
    │   ├── mt_dialplan.xml           # 移动终止（来电）
    │   └── emergency.xml
    └── dialplan_lab/                 # ��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"    # 引用您要在此主机上使用的网关文件夹
      dialplan_folder: "dialplan"          # 可选 - 如果未设置，则默认为
"dialplan"
```

发生了什么：

1. Ansible 找到 `gateways_folder: "gateways_prod"`
2. 将 `hosts/Customer/group_vars/gateways_prod/` 中的所有文件复制到 `/etc/freeswitch/sip_profiles/`
3. 将 `hosts/Customer/group_vars/dialplan/`（或 `dialplan_folder` 指定的文件夹）中的所有文件复制到 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. 使用 Jinja2 进行模板化（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，使用角色中的默认模板。

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. 使用 Jinja2 进行模板化（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，使用角色中的默认模板。

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

真实世界目录结构示例

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

常见参数引用 group_vars

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

参数	组件	引用
SS7 模板	ss7（各种角色）	Jinja2 模板文件（例如，`stp.j2`、`hlr.j2`、`camel.j2`）
配置 YAML	各种组件	直接配置文件（例如，`upf_config.yaml`、`crm_config.yaml`）

关键点

1. **group_vars** 保存自定义 - 覆盖默认配置
2. 按名称引用 - 使用参数如 `smsc_template_config` 或 `gateways_folder`
3. 模板支持 **Jinja2** - 使用 `{{ variable_name }}` 访问任何 Ansible 变量
4. 文件夹部署所有内容 - 引用文件夹中的所有文件都会被复制
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** 来**:**

- 自定义组件配置模板
- SIP □关定义
- 呼叫路由拨号计划
- Diameter 路由规则
- 覆盖默认设置的客户特定设置

□ □□□□ **group_vars** 来**:**

- 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，如果是永久性则提交到 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
- **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 - 组件配置详情

□□□□

□□

□□

□□□□□□

□□□□□□□□□□ HTML □□□□□□□ OmniCore □□□□□□□□□□□□□□□□□□□□□□□□□

□□ **`services/all.yml`** □□□□□□□□□□□

□□□□

□□□□

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

□□

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

□□□□□□□□□□□□□□□

□□□□

HTML □□□□□

主机信息

- 主机名和 **IP** 地址
- 网络/子网（来自 `host_vm_network` 变量，如果未设置则为 N/A）
- **CPU**（vCPU 数量）
- **RAM**（总内存和可用内存）
- 磁盘（根分区的总空间和可用空间及百分比）
- 操作系统（发行版和版本）

服务状态

- 服务状态（活动/非活动，带颜色指示）
- 正常运行[illegible]间
- 版本/发布信息

HSS Diameter 对等体

- 数据库连接状态（已连接/未连接）
- **Diameter** 对等体连接（IP、源主机、状态）
- 从 HSS 指标端点获取（端口 9568）

其他常用工具

基础系统配置

公共角色（`services/common.yml`）

- 将基础系统配置应用于所有主机
- 设置存储库、SSH 密钥、时区、NTP
- 配置[illegible]络和系统加固
- 在部署服务之前运行此命令

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

设置用户（`services/setup_users.yml`）

- 在所有主机上创建和配置用户帐户
- 管理 SSH 密钥和 sudo 权限
- 确保用户设置一致

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

重启（`services/reboot.yml`）

- 优雅地重启所有目标主机
- 等待系统重新上线（5 分钟超时）
- 在内核更新或配置更改后非常有用

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

操作工具

IP 计划生成器（`util_playbooks/ip_plan_generator.yml`）

- 生成 IP 地址分配的 HTML 报告
- 显示来自主机文件的完整□络拓扑
- 对于文档和故障排除非常有用

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

HSS 备份（`util_playbooks/hss_backup.yml`）

- 备份 HSS 数据库表
- 将 MySQL 转储复制到本地 Ansible 机器
- 交互式提示备份路径

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

获取本地捕获 (`util_playbooks/getLocalCapture.yml`)

- 从所有主机获取两个最新的包捕获文件
- 从 `/etc/localcapture/` 检索 pcap 文件
- 对于调试连接问题非常有用

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

更新 **MTU** (`util_playbooks/updateMtu.yml`)

- 更新□络接口 MTU 设置
- 通过 netplan 应用更改
- 对于巨型帧配置非常有用

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

相关文档

- 主 README - 概述和入门
- Ansible 部署简介 - 运行剧本
- 主机文件配置 - 配置您的清单
- 部署架构 - 完整系统概述
- APT 缓存系统 - 包管理

主机文件配置

概述

主机文件（也称为清单文件）是定义您整个蜂窝□络部署的中央配置文档。它指定：

- 要部署的□络功能
- 它们运行的位置（IP 地址、□络段）
- 它们的配置方式（服务特定参数）
- 客户特定设置（PLMN、凭据、功能）

文件位置

主机文件按客户和环境组织：

```
services/hosts/
└── Customer_Name/
    └── host_files/
        ├── production.yml
        ├── staging.yml
        └── lab.yml
```

示例主机文件结构

以下是一个简化的示例，展示了关键部分：

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

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

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

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

# □□□□
license_server:
  hosts:
    customer-licenseserver:
      ansible_host: 10.10.2.150

# □□□□
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，该标准概述了 OmniCore 部署推荐的四个子网架构。

Proxmox 用户：`host_vm_network` 参数指定要使用的桥接。有关自动化配置的详细信息，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）
```

服务特定参数

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                # 分离控制/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□件分支
gateways_folder: "gateways_prod"   # SIP 网关配置
```

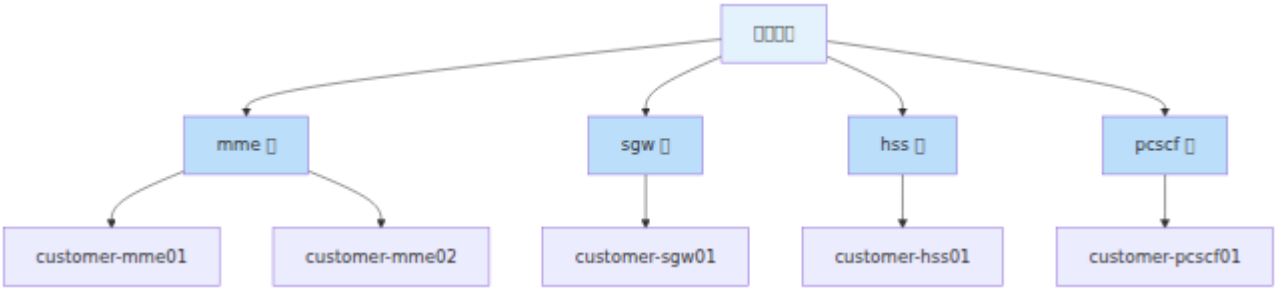
全局变量部分

`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 apt_cache_servers 组已定义主机，
    # use_apt_cache 默认为 true，apt_repo.apt_server
    # 默认为第一个缓存服务器的 IP 自动
    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：

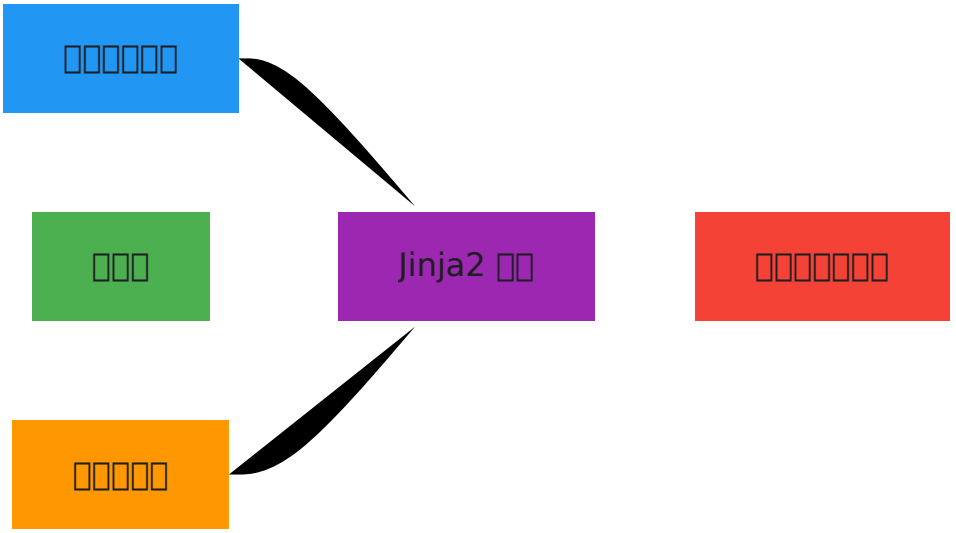


□□□□□□ `mme` □□□□□□□□□□ `mme:hosts:` □□□□□□□□□

□□ Jinja2 □□□□□□

Ansible □□ **Jinja2** □□ □□□□□□□□ group_vars □□□□□□□□□□□□□□

Jinja2 □□□□□



□□□□□□

□□□□□□□

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

Jinja2 □□□□□□□□□

```
# 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□配置文件:

```
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 位数字
mnc{{ '%03d' | format(plmn_id.mnc|int) }}
```

使用 **group_vars** 重写变量

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

请参见：组变量配置

完整示例主机文件

以下是一个更完整的示例（敏感数据已模糊处理）：

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

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

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

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

# IMS □□
pcscf:
```

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

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

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

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

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

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

```
      memory_mb: 8192

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

# □□□□
all:
  vars:
    ansible_connection: ssh
    ansible_password: password
    ansible_become_password: password

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

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

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

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

    # □□□□
    charging:
```

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

    # 防火墙规则
    firewall:
      allowed_ssh_subnets:
        - '10.0.0.0/8'
        - '192.168.0.0/16'
      allowed_ue_voice_subnets:
        - '10.0.0.0/8'
      allowed_signaling_subnets:
        - '10.0.0.0/8'

    # 虚拟化管理程序配置（Proxmox 示例）
    proxmoxServers:
      customer-prxmx01:
        proxmoxServerAddress: 10.10.0.100
        proxmoxServerPort: 8006
        proxmoxApiTokenName: Customer
        proxmoxApiTokenSecret: "token-secret"
        proxmoxTemplateName: ubuntu-24.04-cloud-init-template
        proxmoxNodeName: pve01
```

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

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□关（控制平面）

- **OmniPGW** - 包□关（控制平面）
- **OmniUPF** - 用户平面功能
- **OmniDRA** - Diameter 路由代理
- **OmniTWAG** - 受信 WLAN 接入□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□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��� **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□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

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，以适当分隔□络功能，以确保安全性、性能和操作清晰性。

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□用于内部□络：

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

重要提示：这些是建议，而非要求

本文档中描述的子□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单个子□中
- 每种主机类型在其自己□子网中：可以为每种组件类型创建单独的子□（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主机组织成任何子□结构
- 混合内部和公共 **IP**：某些主机可以使用内部（RFC 1918）地址，而其他主机使用公共 IP，所有这些都在同一部署中

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

网络段划分

1. 数据包核心网络（第一个 /24）

目□：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□关）
- OmniPGW-C（PDN □关控制平面）
- OmniUPF/PGW-U（用户平面功能 / PDN □关用户平面）

示例：`10.179.1.0/24`

```
mme:
  hosts:
    omni-site-mme01:
      ansible_host: 10.179.1.15
      gateway: 10.179.1.1
      host_vm_network: "vmbr1"
```

---

2. 信令网络（第二个 /24）

目□：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）
- OminiHSS PCRF（策略和计费规则功能）
- OmniDRA DRA（Diameter 路由代理）
- DNS 服务器
- 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） - 如果部署
- APT 缓���服务器 - 如果部署

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: "vmbr2"
```

---

3. IMS 内部网络（第三个 /24）

目□：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（询问呼叫会话控制功能）
- OmniTAS（电话应用服务器 / 应用服务器）
- OmniMessage（短信控制器，SMPP，IMS）
- OmniSS7 STP（SS7 信令转发点）
- OmniSS7 HLR（家庭位置注册） - 用于 2G/3G
- OmniSS7 IP-SM-GW（MAP SMSc）
- OmniSS7 CAMEL □关

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: "vmbr3"
```

---

4. UE 公共网络（第四个 /24）

目□：面向用户的服务，如 IMS 和 DNS

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

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: "vmbr4"
```

实施方法

OmniCore 支持两���主要方法来实现此□络分段：

方法 1：物理/虚拟网络接口（推荐用于生产）

为每个□络段使用单独的 NIC 或虚拟桥。这提供了最强的隔离，是生产部署的推荐方法。

示例：

```
# 数据包核心 - vmbr1
mme:
  hosts:
    omni-lab07-mme01:
      ansible_host: 10.179.1.15
      gateway: 10.179.1.1
      host_vm_network: "vmbr1"

# 信令 - vmbr2
hss:
  hosts:
    omni-lab07-hss01:
      ansible_host: 10.179.2.140
      gateway: 10.179.2.1
      host_vm_network: "vmbr2"

# IMS 内部 - vmbr3
icscf:
  hosts:
    omni-lab07-icscf01:
      ansible_host: 10.179.3.55
      gateway: 10.179.3.1
      host_vm_network: "vmbr3"

# UE 公共 - vmbr4
pcscf:
  hosts:
    omni-lab07-pcscf01:
      ansible_host: 10.179.4.165
      gateway: 10.179.4.1
      host_vm_network: "vmbr4"
```

方法 2：基于 VLAN 分段

使用单个物理接口并通过 VLAN 标签来分隔络。这适用于较小的部署或当物理 NIC 有限时。

示例：

```
# 所有组件使用 vmbr12，具有不同的 VLAN
applicationserver:
  hosts:
    ons-lab08sbc01:
      ansible_host: 10.178.2.213
      gateway: 10.178.2.1
      host_vm_network: "ovsbr1"
      vlanid: "402"

dra:
  hosts:
    ons-lab08dra01:
      ansible_host: 10.178.2.211
      gateway: 10.178.2.1
      host_vm_network: "ovsbr1"
      vlanid: "402"

dns:
  hosts:
    ons-lab08dns01:
      ansible_host: 10.178.2.178
      gateway: 10.178.2.1
      host_vm_network: "ovsbr1"
      vlanid: "402"
```

网络配置：

- 在物理交换机上配置 VLAN
- 在虚拟化管理程序级别适当地标记流量
- 在□关/防火墙处在 VLAN 之间路由

示例 **VLAN** 映射：

```
VLAN 10: 10.x.1.0/24 (数据包核心)
VLAN 20: 10.x.2.0/24 (信令)
VLAN 30: 10.x.3.0/24 (IMS 内□)
VLAN 40: 10.x.4.0/24 (UE 公共)
```

使用公共 IP 地址

概述

许多 OmniCore 部署需要某些组件具有公共 IP 地址以实现外部连接，例如：

- **DRA** - 用于与外部运营商的漫游直径信令
- 漫游 **SGW/PGW** - 用于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的处理方式 完全相同。只需在 `ansible_host` 字段中指定公共 IP 地址，以及适当的□关和子□掩码。

示例：具有公共 **IP** □漫游 **SGW/PGW**

```yaml
sgw:
  hosts:
    # 内部 SGW 在私有网络上
    opt-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
    opt-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
    opt-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 □ DRA**

```yaml
dra:
  hosts:
    opt-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 □ ePDG**

```
epdg:
  hosts:
    opt-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
- 专门用于外部运营商漫游流量的公共 SGW
- 同一 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。

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。每个组件在启动时以及在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"]
    license_enforced: true
```

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
```

您可以通过访问 https://license_server 检查所有许可证的状态。

网络要求

防火墙配置

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

主机名	**IP** 地址	目
time.omnitouch.com.au	160.22.43.18	许可证验证服务器 1
time.omnitouch.com.au	160.22.43.66	许可证验证服务器 2
time.omnitouch.com.au	160.22.43.114	许可证验证服务器 3

所需出站规则：

- 协议：HTTPS (TCP/443)
- 目的地：160.22.43.18, 160.22.43.66, 160.22.43.114
- 方向：出站

DNS 要求

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

所需 **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。

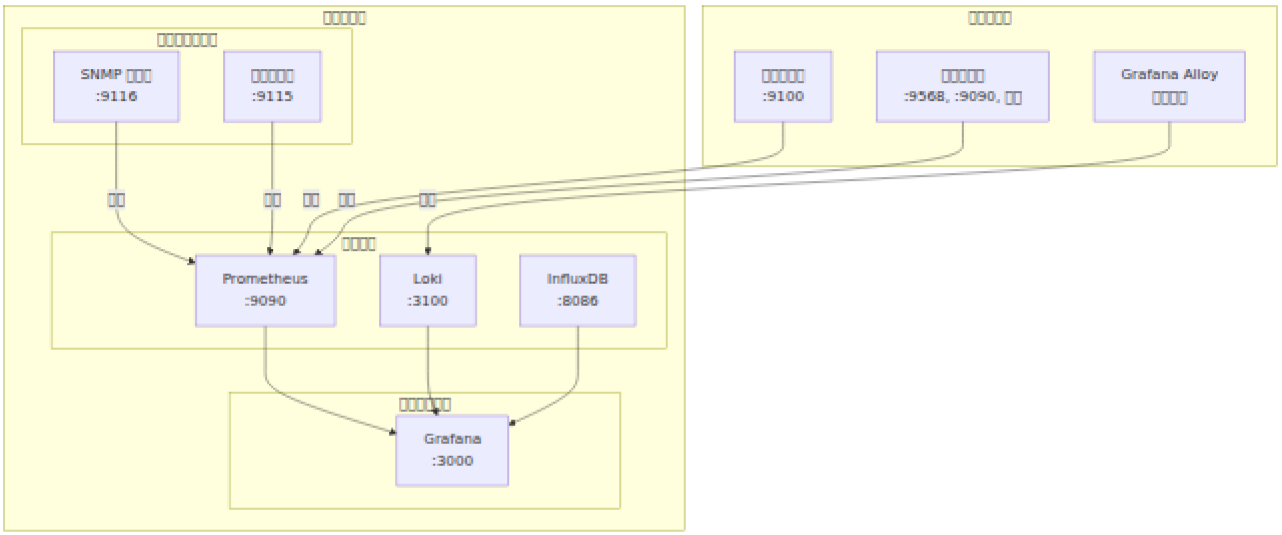
相关文档

- 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堆栈，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□	端口	数据保留
Prometheus	时间序列指标存储	9090	15 天（默认）
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	N/A
节点导出器	主机指标	9100	N/A
SNMP 导出器	□络设备指标	9116	N/A
黑箱导出器	ICMP/HTTP 探测	9115	N/A

数据源

Prometheus

Prometheus 每 1 分钟抓取所有 OmniCore 组件的指标。

数据源 **UID**: `omnicore_prometheus`

抓取目标

作业名称	库存组	端口	指标
节点导出器	all	9100	主机 CPU、内存、磁盘、□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 独立	upf	9090	数据包计数、吞吐量
Kamailio CSCF	pcscf, scscf, icscf	9090	注册、事务
应用服务器 FreeSWITCH	applicationserver	9090	呼叫计数、通道使用
DRA	dra	9568	路由决策、对等状态
OmniMessage	omnimessage	9568	短信投递、SMPP 会话
OmniSS7	omniss7	8080	SS7/SIGTRAN □关指标
KeyDB	ocs	9121	内存使用、操作/秒
CGrateS	ocs	2080	计费、收费、CDR 统计

基础设施导出器

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

Loki

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

数据源 **UID**: `omnicore_loki`

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期或不同的查询模式。

数据源 **UID**: `omnicore_influxdb`

数据库

数据库	目	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

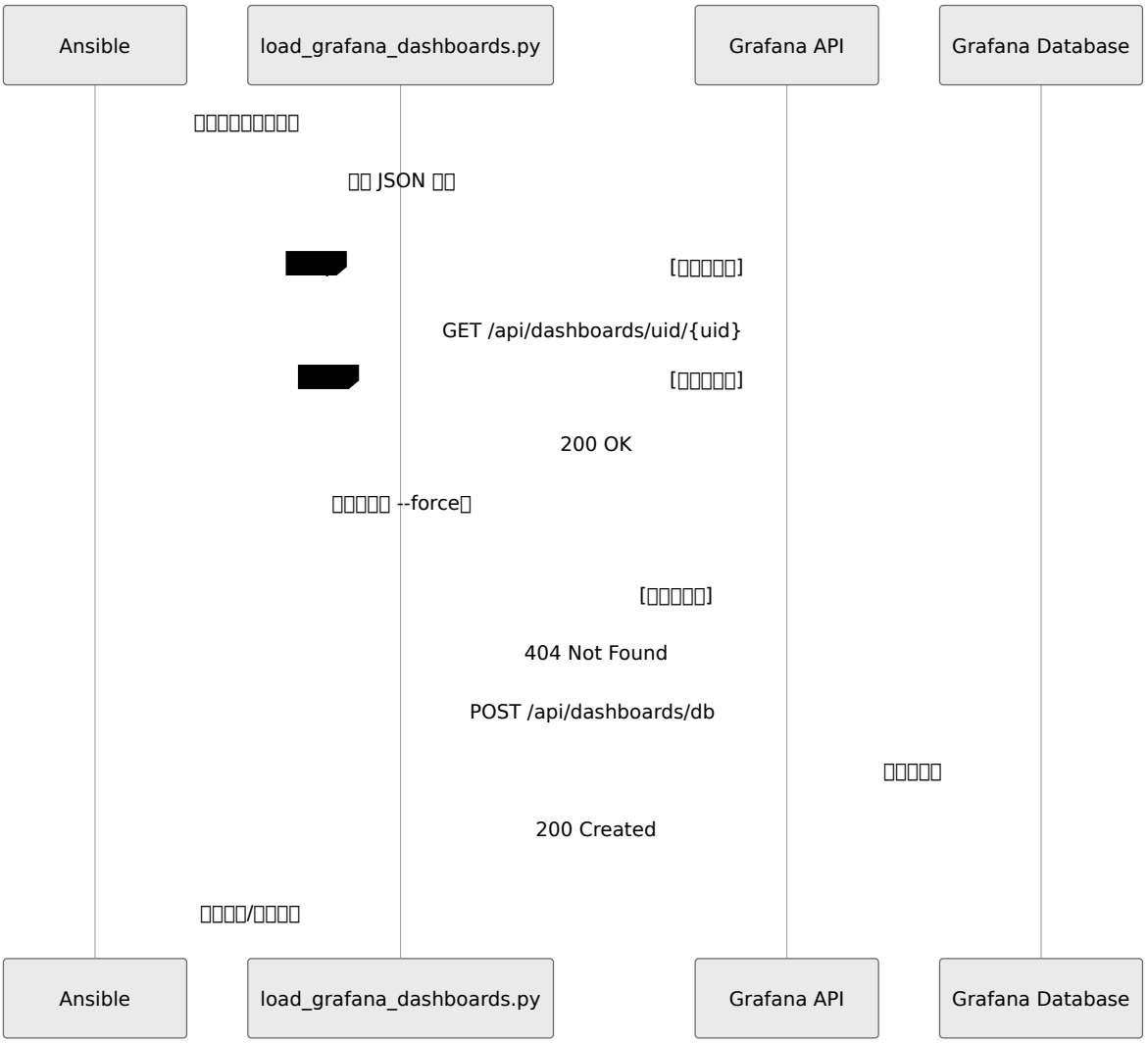
仪表板按功能区域组织到文件夹中：

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	节点导出器、SNMP、□络监控
RAN	诺基亚/OmniRAN 性能仪表板
日志	组件特定的日志查看器

仪表板加载（基于 **API**）

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

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



[redacted]

[redacted] Ansible [redacted]

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

# 强制更新现有仪表板
python3 roles/monitoring/files/load_grafana_dashboards.py \
    --url http://<monitoring-ip>:3000 \
    --user admin \
    --password <grafana_admin_password> \
    --dashboards-dir roles/monitoring/templates/grafana/dashboards
\
    --force
```

仪表板源文件

仪表板 JSON 文件存储在 Ansible 仓库中:

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

仪表板备份

从运行中的 Grafana 实例导出仪表板和警报到仓库以进行版本控制：

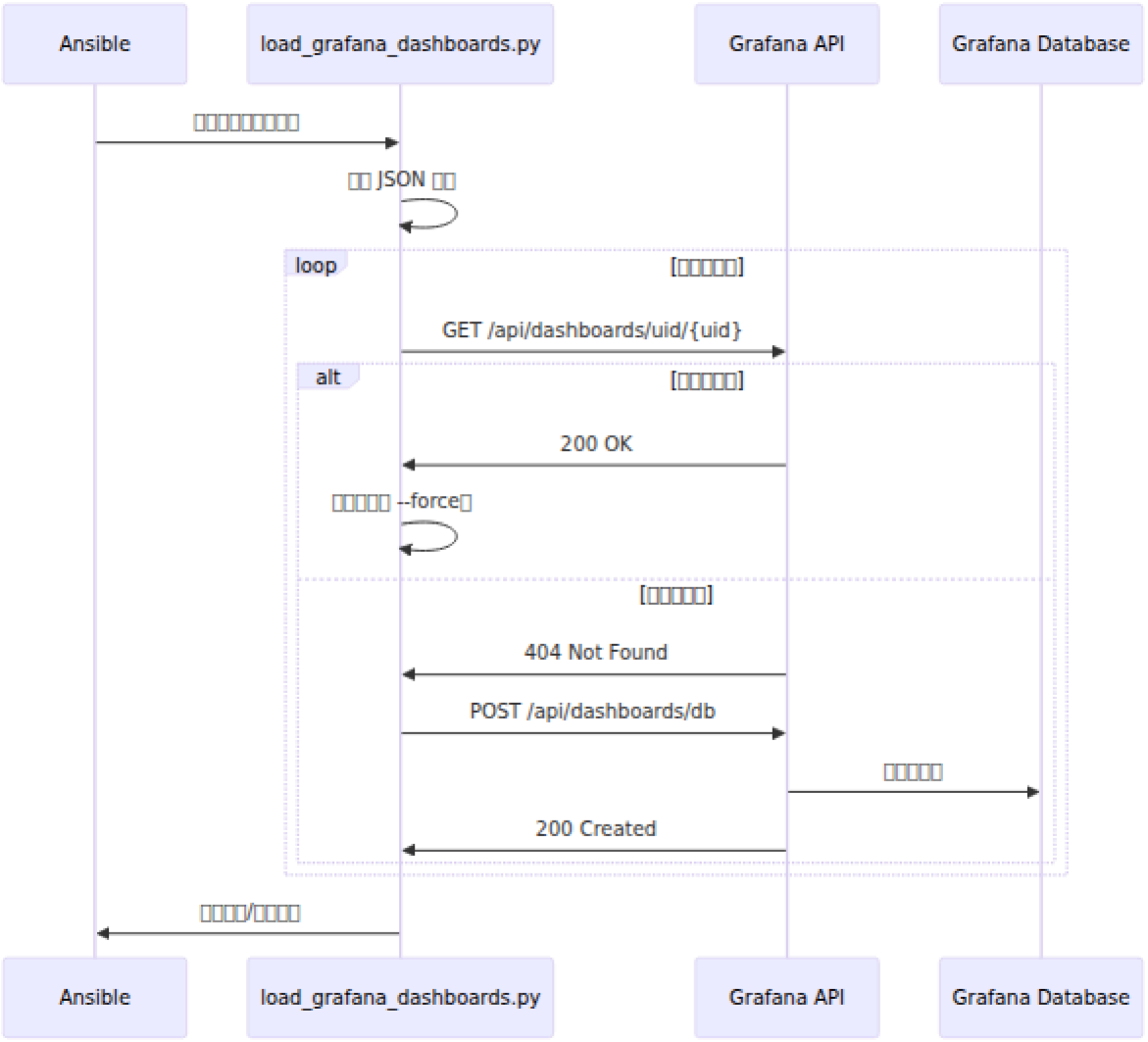
```
# 从 roles/monitoring 目录
python3 export_grafana.py --customer <CUSTOMER>
```

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

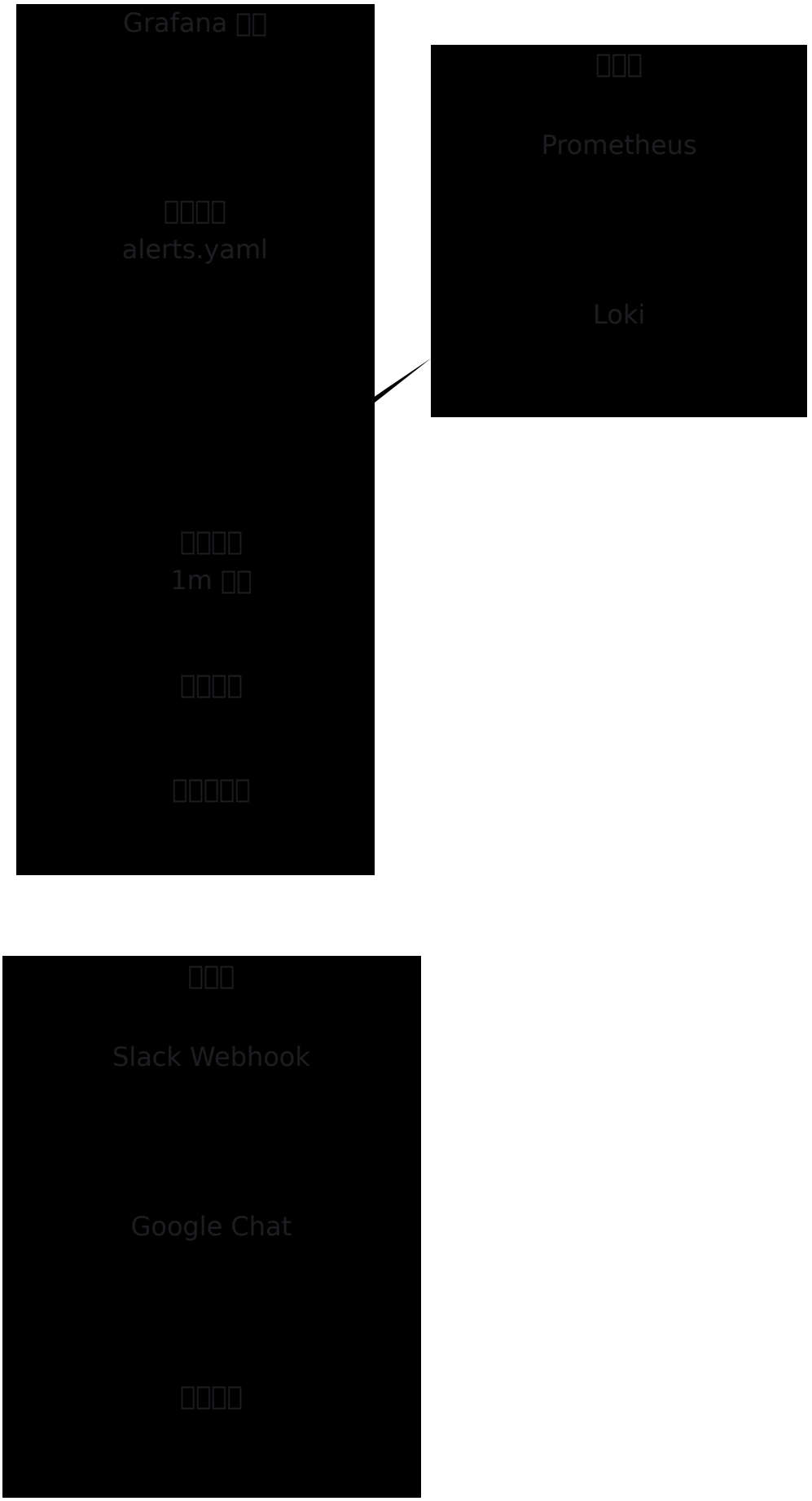
- 仅写入已更改的文件（忽略诸如 `version` 和 `id` 等易变字段）
- 保留与 Grafana 组织匹配的文件夹结构

- 将仪表板标题映射到一致的文件名

自定义仪表板

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

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


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

警报加载

警报在 YAML 中定义，并通过 API 加载，类似于仪表板：

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

默认警报规则

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

警报规则结构

`alerts.yaml` 中的警报规则：

```yaml
apiVersion: 1
groups:
  - orgId: 1
    name: 警报组
    folder: 基础设施
    interval: 1m
    rules:
      - uid: alert-lowDiskSpace
        title: 主机上的磁盘空间 90% 使用
        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 或客户特定变量中
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"
```

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: ['alertname', 'instance']
    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	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
`admin_password`	`grafana_admin_password` var	管理员用户密码
`allow_embedding`	true	允许在 iframe 中嵌入
`provisioning`	`/etc/grafana/provisioning`	配置目录

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
节点导出器	9100	HTTP	主机指标
SNMP 导出器	9116	HTTP	SNMP 指标代理
黑箱导出器	9115	HTTP	探测指标
Alloy	12345	HTTP	Alloy UI 和指标

常见操作

查看指标

1. 打开 Grafana，地址为 `http://<monitoring-ip>:3000`
2. 导航到 仪表板 并选择相应的文件夹
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：

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

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

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

创建自定义警报

1. 在 `roles/monitoring/templates/grafana/provisioning/alerting/alerts.yaml` 中创建警报规则
2. 运行监控剧本或手动加载：

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

备份仪表板

在 Grafana UI 中进行更改后，导出到仓库：

```
cd roles/monitoring
python3 export_grafana.py --url http://<monitoring-ip>:3000
git add templates/grafana/
git commit -m "从生产更新仪表板"
```

故障排除

Prometheus 目标宕机

症状：仪表板显示“无数据”或目标状态显示 DOWN

可能原因：

- 目标主机上的服务未运行
- 防火墙阻止导出器端口
- 主机名解析不正确

解决方案：

1. 检查目标上的服务是否在运行：

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

2. 验证从监控服务器的连接性：

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

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

仪表板未加载

症状：仪表板显示加载旋转图标或错误

可能原因：

- 数据源连接失败
- 查询超时
- 仪表板 JSON 损坏

解决方案：

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

警报未触发

症状：条件满足但未收到通知

可能原因：

- 警报已暂停
- 联系点配置错误
- 通知策略不匹配

解决方案：

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

Grafana 无法连接到数据源

症状：Grafana 中出现“错误关”或连接超时

可能原因：

- 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'
```

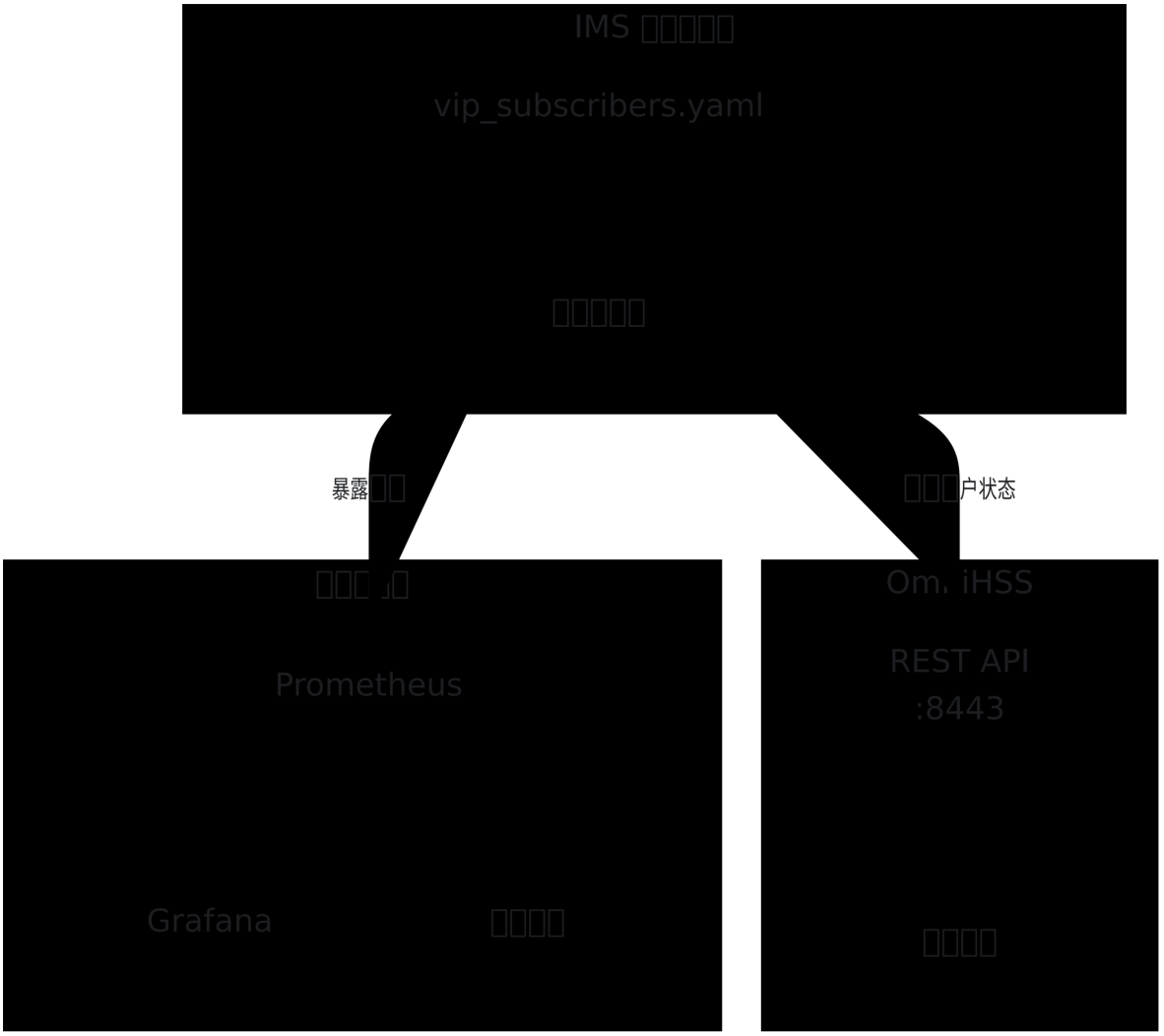
3. 测试本地连接性：

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

VIP 用户监控

VIP 用户监控提供对关键用户的实时状态跟踪。该系统监控 IMS 注册、EPC 附加和 UE 可达性，适用于指定的高优先级用户，如紧急服务、医院和关键基础设施。

架构



服务类型

用户按预期服务类型分类，这决定了健康评估：

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

健康评估：

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

配置

VIP 用户在主机组变量中配置：

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

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

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

  # 要监控的用户
  subscribers:
    # 全服务用户（IMS + EPC）
    - imsi: "313380930011949"
      label: "海上救援"
      type: "full"

    - imsi: "313380930011948"
      label: "警察"
      type: "full"

    # 仅数据服务（路由器、物联网 - 不期望 IMS）
    - imsi: "313380930010064"
      label: "医院路由器"
      type: "data_only"

    # 仅 VoIP 服务（期望 IMS，无数据承载）
    - imsi: "896468419011262"
      label: "医院电话"
      type: "voip_only"
```

配置参数

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

用户参数

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

注意：MSISDN 会从 HSS API 响应中自动检索，无需配置。

指标

导出器在端口 9550 的 `/metrics` 上暴露指标。

状态指标

指标	类型	描述
`vip_subscriber_service_healthy`	Gauge	如果用户满足其服务类型的健康标准则为 1，否则为 0
`vip_subscriber_ims_registered`	Gauge	如果用户已分配 S-CSCF，则为 1，否则为 0
`vip_subscriber_epc_registered`	Gauge	如果用户具有活动 MME 附加，则为 1，否则为 0
`vip_subscriber_ue_ip_reachable`	Gauge	如果 UE IP 对 ping 响应，则为 1，否则为 0
`vip_subscriber_hss_reachable`	Gauge	如果 HSS API 返回有效数据，则为 1，否则为 0
`vip_subscriber_enabled`	Gauge	如果用户帐户在 HSS 中启用，则为 1，否则为 0

年龄指标

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

信息指标

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

所有指标上□标签：

标签	描述	示例
`imsi`	用户 IMSI	`313380930011948`
`label`	人类可读名称	`警察`
`msisdn`	电话号码（来自 HSS）	`24724748250`
`type`	服务类型	`full`、`voip_only`、`data_only`

`vip_subscriber_info` 上□附加标签：

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

示例查询

```
# 健康 VIP 用户的计数
count(vip_subscriber_service_healthy == 1)

# 不健康 VIP 用户的计数
count(vip_subscriber_service_healthy == 0)

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

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

# IMS 注册过期的用户（>1 小时）
vip_subscriber_ims_registration_age_seconds > 3600
```

仪表板

IMS VIP 用户 仪表板提供 VIP 用户状态的实时可见性。

位置: Grafana → IMS → IMS VIP 用户

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

面板

面板	描述
服务健康	满足健康标准的用户计数
服务不健康	未满足健康标准的用户计数
IMS 注册	具有活动 IMS 注册的用户计数
EPC 注册	具有活动 EPC 附加的用户计数
UE 可达	具有可 ping 的 UE IP 的用户计数
总监控	配置的 VIP 用户总数
VIP 用户状态	显示所有用户及其状态的表格
服务健康随时间变化	显示健康历史的状态时间线

警报

当 VIP 用户服务变得不健康时，���报规则会触发。

VIP 用户服务不健康

警报：`VIP 用户服务不健康` 严重性：关键 持续时间：1 分钟 条件：`vip_subscriber_service_healthy == 0`

注释：

- 摘要：`VIP 用户 {{ $labels.label }} 不健康`
- 描述：包括服务类型和适当的标识符：
  - VoIP 服务：名称和 MSISDN
  - 数据服务：名称和 IMSI
  - 全服务：名称、MSISDN 和 IMSI

示例警报描述：

- 医院电话（voip_only）服务已宕机。MSISDN: 24724766000
- 医院路由器（data_only）服务已宕机。IMSI: 313380930010064
- 警察（full）服务已宕机。MSISDN: 24724748250 IMSI: 313380930011948

添加新 VIP 用户

1. 编辑配置文件：

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

2. 部署更新的配置：

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

3. 验证新用户是否出现在指标中：

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

相关文档

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

Netplan 配置

概述

OmniCore 可以使用 netplan 自动配置已部署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)
- 为公共 IP、对等连接或专用流量添加辅助接口
- 为特定目的地配置静态路由

启用 Netplan 配置

要启用主机的自动 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

以下是完整的 `netplan.yaml.j2` 模板，包含解释每个部分的注释：

```
network:
  version: 2
  ethernets:
    # 主接口 - 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 %}
        # 如果此主机是 DNS 服务器，则使用外部 DNS 以避免循环依赖
        - 8.8.8.8
{% else %}
        # 否则，使用来自清单中 'dns' 组的 DNS 服务器
{% for dns_host in groups['dns'] | default([]) %}
        - {{ hostvars[dns_host]['ansible_host'] }}
{% endfor %}
{% endif %}
        search:
        - slice
      routes:
      - to: "default"
        via: "{{ gateway }}"

{% if secondary_ips is defined %}
    # 辅助接口 - 循环遍历来自清单的 secondary_ips 字典
    # 接口命名: ens19, ens20, ens21... (18 + loop.index)
{% for nic_name, nic_config in secondary_ips.items() %}
    ens{{ 18 + loop.index }}:
      addresses:
      - "{{ nic_config.ip_address }}/{{ mask_cidr | default(24)
}}"
{% if nic_config.routes is defined %}
      # 此接口的静态路由 - 每个路由使用此接口的网关
      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
- 辅助接口命名为 `ens19`、`ens20` 等，以匹配 Proxmox NIC 命名
- 每个辅助 IP 可以有自己的□关和静态路由

主接口配置

主接口 (eth0) 自动配置使用：

- `ansible_host` - IP 地址
- `gateway` - 默认□关
- `mask_cidr` - □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`

辅助接口

对于需要额外□络接口的主机（公共 IP、对等连接等），使用 `secondary_ips` 配置。

结构

```
secondary_ips:
  <logical_name>:
    ip_address: <ip_address>
    gateway: <gateway_ip>
    host_vm_network: <proxmox_bridge>
    vlanid: <vlan_id>
    routes:                    # 可选 - 通过此接口的静态路由
      - '<destination_cidr>'
      - '<destination_cidr>'
```

接口命名

辅助接口使用 Ubuntu 的可预测命名方案自动命名：

- 第一个辅助接口：`ens19`
- 第二个辅助接口：`ens20`
- 第三个辅助接口：`ens21`
- 依此类推...

这与 Proxmox 在向虚拟机添加额外 NIC 时分配的接口名称相匹配。

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
          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 Netplan 输出

上述配置生成：

```
network:
  version: 2
  ethernets:
    eth0:
      addresses:
      - "10.0.1.100/24"
      nameservers:
        addresses:
        - 10.0.1.53
        search:
        - slice
      routes:
      - to: "default"
        via: "10.0.1.1"
    ens19:
      addresses:
      - "192.0.2.50/24"
      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"
```

Proxmox 集成

在使用 `proxmox.yml` 剧本时，辅助 NIC 会在虚拟机上自动创建：

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

Proxmox 配置使用:

- `host_vm_network` - 连接 NIC 的桥接
- `vlanid` - 接口的 VLAN 标签

工作原理

1. 从主机文件传递变量到 Jinja2 模板
2. 模板渲染到 `/etc/netplan/01-netcfg.yaml`
3. 删除任何现有的 netplan 配置以防止冲突
4. `netplan apply` 激活配置
5. 使用 `ip addr show` 验证 IP 地址

常见用例

带公共 IP □直径边缘代理（DEA）

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

带 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"
```

多宿主服务器，管理和数据网络分开

```
<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           # 数据网络
      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"
```

在模板中引用辅助 IP

您可以在其他 Jinja2 模板和配置文件中引用辅助 IP 地址。

在同一主机上

当在具有辅助 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 明确引用（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 □□ 主机的辅助 IP（例如，配置对等连接）时，使用 `hostvars` 和目标主机名：

```
# 引用 dra 组中的第一台主机
{{ hostvars[groups['dra'][0]]['secondary_ips']
['diameter_public_ip']['ip_address'] }}

# 循环遍历所有 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
    # 绑定到此主机的公共直径 IP
    local_ip: {{ hostvars[inventory_hostname]['secondary_ips']
['diameter_public_ip']['ip_address'] }}
    remote_ip: 198.51.100.50
    port: 3868
```

检查辅助 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 登录到虚拟机并检查接口名称：

```
ip link show
```

对于具有两个辅助接口的虚拟机，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

Proxmox VM/LXC 部署

我们大多数客户在 Proxmox 上运行 OmniCore 堆栈，本指南详细解释了如何使用 `proxmox` 剧本来设置他们的环境。

我们仍然继续支持 VMware、HyperV 和云（目前为 Vultr / AWS / GCP）进行部署。

另见：

- 主机文件配置 - 定义要部署的虚拟机
- IP 规划标准 - IP 地址分配指南
- Netplan 配置 - 辅助 IP 和多□卡设置
- 部署架构 - 完整的部署工作流程

LXC 与 VM

LXC 容器：

- 轻量级，分享主机内核
- 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，具有专用内核
- 完整隔离
- 可以运行内核模块和自定义□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 、有络功能
- **LXC**：实验/测试环境、轻量级服务（apt-cache、监控）

Proxmox 设置

1. 创建 API 令牌

```
# 在 Proxmox UI：数据中心 → 权限 → API 令牌
# 创建令牌：root@pam!<TokenName>
# 复制令牌密钥（仅显示一次）
```

2. 创建 Cloud-Init 虚拟机模板（仅适用于虚拟机）

在 Proxmox 主机上运行此脚本。它下载 Ubuntu 的云镜像并创建一个带有 cloud-init 用户凭据的模板。

```
#!/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"

echo "=== 下载 Ubuntu 云镜像 ==="
cd /var/lib/vz/template/iso
wget -N "$IMAGE_URL"

echo "=== 清理旧模板 ==="
qm destroy $TEMPLATE_ID --purge 2>/dev/null || true

echo "=== 启用片段存储 ==="
pvesm set local --content images,vztmpl,iso,backup,snippets

echo "=== 创建 cloud-init 用户数据 ==="
mkdir -p /var/lib/vz/snippets
cat > /var/lib/vz/snippets/user-data.yml << 'USERDATA'
#cloud-config
ssh_pwauth: true
users:
  - name: omnitouch
    plain_text_passwd: password
    lock_passwd: false
    shell: /bin/bash
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: sudo
USERDATA

echo "=== 创建模板虚拟机 ==="
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 set $TEMPLATE_ID --cicustom user=local:snippets/user-data.yml
```

```
qm template $TEMPLATE_ID

echo "=== 模板 $TEMPLATE_ID 创建成功 ==="
```

注意：

- 模板提供了一个后备登录：`omnitouch` / `password`（如果 cloud-init 失败，供控制台访问）
- 通过 Ansible 克隆时，凭据将从主机文件中的 `local_users` 中覆盖：
  - 用户名：`local_users` 中第一个用户的密钥
  - 密码：第一个用户的 `password` 字段（如果未设置，默认为 'password'）
  - SSH 密钥：第一个用户的 `public_key` 字段
- `--vga std` 确保 Proxmox Web 控制台正常工作
- wget 的 `-N` 标志仅在本地副本较新时下载

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
- 系统：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 **-** 清理和准备

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

# 清理机器特定数据
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）
- Cloud-Init → 用户：`omnitouch`，密码：`password`
- 硬件 → 选项 → QEMU 客户端代理 → 启用
- 右键单击虚拟机 → 转换为模板

3. 下载 LXC 模板（仅适用于 LXC）

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

主机文件配置

用于虚拟机部署（proxmox.yml）

```yaml
all:
  vars:
    proxmoxServers:
      pve-node-01:
        proxmoxServerAddress: 192.168.1.100
        proxmoxServerPort: 8006
        proxmoxRootPassword: YourPassword
        proxmoxApiTokenName: ansible
        proxmoxApiTokenSecret: "your-token-secret-uuid"
        proxmoxTemplateName: ubuntu-2404-template
        proxmoxTemplateId: 9000
        proxmoxNodeName: pve-node-01
        storage: local-lvm  # 可选
      pve-node-02:
        # ... 第二个节点配置

    # 用户凭据 - 第一个用户用于虚拟机 cloud-init
    local_users:
      admin_user:
        name: Admin User
        public_key: "ssh-rsa AAAA..."
        password: "optional-password"  # 如果未设置，默认为 'password'

mme:
  hosts:
    site-mme01:
      ansible_host: 192.168.1.10
      gateway: 192.168.1.1
      vlanid: "100"  # 可选
```

用于 LXC 部署 (proxmox_lxc.yml)

```yaml
all:
  vars:
    proxmoxServerAddress: 192.168.1.100
    proxmoxServerPort: 8006
    proxmoxNodeName: ['pve-node-01', 'pve-node-02']  # 单个或列表
    proxmoxApiTokenName: ansible
    PROXMOX_API_TOKEN: "your-token-secret-uuid"
    proxmoxLxcOsTemplate: 'local:vztmpl/ubuntu-24.04-
standard_24.04-2_amd64.tar.zst'
    proxmoxLxcCores: 2
    proxmoxLxcMemoryMb: 4096
    proxmoxLxcDiskSizeGb: 30
    proxmoxLxcRootFsStorageName: local-lvm
    mask_cidr: 24
    host_vm_network: vmbr0

    # 用户凭据 - 第一个用户用于初始虚拟机/LXC 访问
    local_users:
      admin_user:
        name: Admin User
        public_key: "ssh-rsa AAAA..."
        password: "optional-password"  # 如果未设置，默认为 'password'

apt_cache_servers:
  hosts:
    site-cache:
      ansible_host: 192.168.1.20
      gateway: 192.168.1.1
      vlanid: "100"  # 可选
      proxmoxLxcDiskSizeGb: 120  # 每个主机覆盖
```

使用

部署虚拟机

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

部署 LXC 容器

```
ansible-playbook -i hosts/Customer/hosts.yml
util_playbooks/proxmox_lxc.yml
```

删除虚拟机/LXC

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

行为

proxmox.yml

- 检查 Proxmox 中是否已存在同名虚拟机
- 使用轮询法在节点之间分配虚拟机
- 从模板克隆
- 配置静态 IP、标签和 cloud-init
- 从第一个 `local_users` 条目设置 **cloud-init** 用户凭据
- 支持 VLAN 标签

proxmox_lxc.yml

- 检查容器是否按名称或 IP 不存在

- 使用轮询法在节点之间分配 LXC
- 创建具有静态 IP 的容器
- 自动创建第一个 `local_users` 帐户，具有 **sudo** 访问权限和 **SSH** 密钥
- 配置□络的 netplan
- 自动启动容器
- 排除 UPF 主机

proxmox_delete.yml

- 停止���删除与库存主机名匹配的虚拟机/LXC
- 在所有配置的节点中搜索
- 20 秒后强制停止

虚拟机/LXC 分配与标签

轮询分配

虚拟机和 LXC 会使用轮询（模）逻辑自动分配到 Proxmox 节点：

示例：**3** 个虚拟机和 **5** 个 **MME**：

```
mme01 → pve-node-01 (index 0 % 3 = 0)
mme02 → pve-node-02 (index 1 % 3 = 1)
mme03 → pve-node-03 (index 2 % 3 = 2)
mme04 → pve-node-01 (index 3 % 3 = 0)
mme05 → pve-node-02 (index 4 % 3 = 1)
```

工作原理：

1. 剧本识别主机的角色组（例如，`mme`、`sgw`、`hss`）
2. 计算该组内的主机索引（从 0 开始）
3. 使用模运算：`host_index % number_of_nodes`
4. 根据结果选择虚拟机

配置：

```
# 对于虚拟机 (proxmox.yml) - 定义多个服务器
proxmoxServers:
  pve-node-01: { ... }
  pve-node-02: { ... }
  pve-node-03: { ... }

# 对于 LXC (proxmox_lxc.yml) - 列出多个节点
proxmoxNodeName: ['pve-node-01', 'pve-node-02', 'pve-node-03']
```

自动标签

虚拟机和 LXC 会自动标记为：

- 角色/组名：主机所属的所有 Ansible 组
- 站点名称：`site_name` 变量

示例：

```
site_name: "melbourne-prod"

mme:
  hosts:
    melbourne-mme01: { ... }
```

结果：虚拟机/LXC 被标记为：`mme`，`melbourne-prod`

标签在 Proxmox UI 中可见，并且对过滤/组织非常有用。

每主机覆盖

在特定主机上覆盖默认值：

```yaml
hosts:
  high-spec-host:
    ansible_host: 192.168.1.50
    gateway: 192.168.1.1
    proxmoxLxcCores: 8         # 覆盖核心数
    proxmoxLxcMemoryMb: 16384  # 覆盖内存
    proxmoxLxcDiskSizeGb: 100  # 覆盖磁盘
```

服务剧本

服务剧本部署和配置 OmniCore 基础设施。位于 `services/` 目录中。

剧本层次结构

剧本按层次结构组织，以实现快速、针对性的部署：

```
all.yml
├── setup_users.yml
├── apt_cache.yml
├── dns.yml
├── common.yml
├── license_server.yml
├── monitoring.yml
│   └── grafana.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
│   ├── smsc.yml
│   └── omnisep.yml
├── omniepdg.yml
├── omniss7.yml
├── ocs.yml
├── crm.yml
├── ran_monitor.yml
└── health_check.yml
```

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

快速参考

顶级剧本

剧本	范围	描述
`all.yml`	全□络	完整部署：用户、DNS、监控、EPC、IMS、OCS、CRM
`epc.yml`	数据包核心	MME、SGW-C、PGW-C、UPF、HSS、DRA
`ims.yml`	语音/IMS	P/I/S-CSCF、应用服务器、XCAP、消息传递
`ocs.yml`	计费	CGrateS、KeyDB 集群、OCS 同步
`monitoring.yml`	可观察性	Prometheus、Grafana、出口程序、HOMER

基础设施剧本

剧本	描述
`common.yml`	基础操作系统配置、软件包、NTP、日志代理
`setup_users.yml`	本地用户帐户和 SSH 密钥
`dns.yml`	DNS 服务器部署
`license_server.yml`	OmniCore 许可证服务器
`netplan.yml`	络接口配置
`firewall.yml`	iptables/nftables 规则
`apt_cache.yml`	本地 APT 镜像（如果启用）

EPC 组件

剧本	组件	描述
`omnimme.yml`	OmniMME	移动管理实体 (4G)
`omnisgwc.yml`	OmniSGW-C	服务□关控制平面
`omnipgwc.yml`	OmniPGW-C	PDN □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	受信任的无线接入□关
`omniepdg.yml`	OmniEPDG	进化数据包数据□关（WiFi 通话）
`gtp_proxy.yml`	GTP 代理	GTP 流量代理

IMS 组件

剧本	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、授权服务器、BSF、视觉语音邮件
`omnimessage.yml`	OmniMessage	短信通过 IP 控制器
`smsc.yml`	SMSC	短消息服务中心

支持服务

剧本	描述
`ocs.yml`	在线计费系统 (CGrateS + KeyDB)
`crm.yml`	客户管理门户
`omniss7.yml`	SS7/SIGTRAN 关
`homer.yml`	SIP/Diameter 数据包捕获
`grafana.yml`	Grafana 仪表板和警报
`promtail.yml`	日志传输到 Loki
`ran_monitor.yml`	RAN 监控集成

虚拟机配置

剧本	描述
`proxmox.yml`	在 Proxmox VE 上创建虚拟机
`proxmox_lxc.yml`	创建 LXC 容器（实验室/测试）
`proxmox_delete.yml`	删除 Proxmox 虚拟机/LXC

操作

剧本	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`	络吞吐量测试

恢复剧本

剧本	描述
`restore_applicationserver.yml`	从备份恢复 OmniTAS
`restore_omnimessage_controller.yml`	从备份恢复 OmniMessage
`restore_smsc.yml`	从备份恢复 SMSC

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

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

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

部署单个组件

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

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

限制特定主机

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

# 在多个特定主机上运行
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
- 实用剧本 - 操作工具（健康检查、恢复等）
- 监控与可观察性 - Grafana、Prometheus、警报

实用程序剧本

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

快速参考

剧本	目□
`proxmox.yml`	在 Proxmox 上配置虚拟机
`proxmox_delete.yml`	在 Proxmox 上删除虚拟机/LXC
`proxmox_lxc.yml`	在 Proxmox 上配置 LXC 容器
`vmware.yml`	在 VMware vSphere 上配置虚拟机
`vmware_delete.yml`	在 VMware vSphere 上删除虚拟机
`health_check.yml`	生成所有服务的综合健康报告
`restore_hss.yml`	从备份恢复 HSS 数据库和/或配置
`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□络文档
`get_ports.yml`	审计所有主机上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`	从所有主机删除本地用户帐户

□□□□□

□□□□

```
# 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"
```

健康检查

文件: `util_playbooks/health_check.yml`

生成涵盖所有已部署的 OmniCore 和 OmniCall 服务的综合 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□信息

组件	收集□数据
所有服务	服务状态、版本、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、仅恢复配置或同时恢复两者。

```
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 MySQL 转储
配置	`hss_<hostname>_<timestamp>.tar.gz`	`/etc/omnihss` 目录的归档

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冲突
3. 将 KeyDB RDB 恢复到第一个节点，启动 KeyDB，允许复制同步
4. 恢复 MySQL StoreDB（可选）
5. 恢复 `/etc/cgrates` 配置（可选）
6. 启动所有节点上的 CGrateS

备份文件格式

类型	文件名模式	内容
KeyDB DataDB	`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□络文档，包括：

- 主机 IP 分配（主 NIC 和次 NIC）
- □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
`./open_ports.rst`	用于 Sphinx 文档的 reStructuredText 表

收集□数据

字段	描�?
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` 目录检索两个最新的数据包捕获文件。

```
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：在提示时输入要删除的用户名。

运行实用程序剧本

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（显示将要更改的内容）
`--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
```