Omnitouch □ Ansible 部署介绍

概述

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

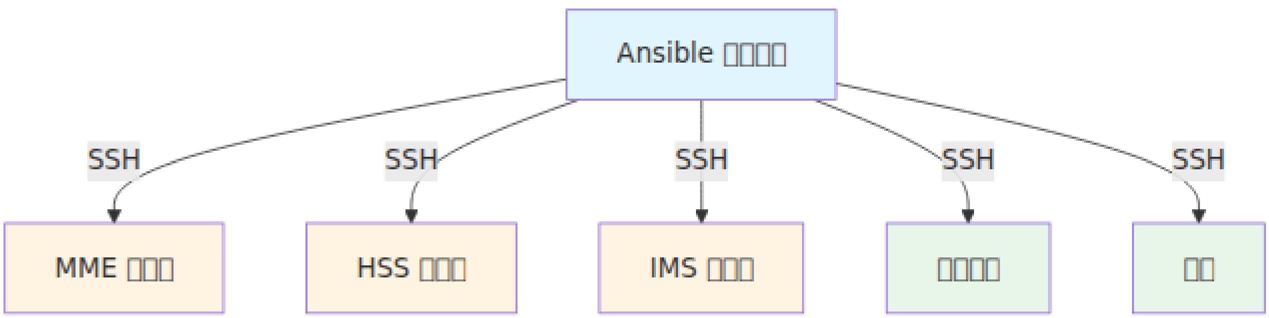
什么是 **Ansible**？

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）

□□ □□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□□□□□
- □□□ IP □□
- □□□□
- □□□□□□

□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□

2. □□

□□ □□ □□□□□□□□□□□□□□□□□□□□□□□□

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

OmniCore □□□□□□□□□`omnihss`□`omnisgwc`□`omnipgwc`□`omnidra` □

□□□ ONS □□□

3. □□

□□ □□ □ □□□ □□□□□

```
- name: □□ EPC □□
  hosts: mme
  roles:
    - common
    - omnimme
```

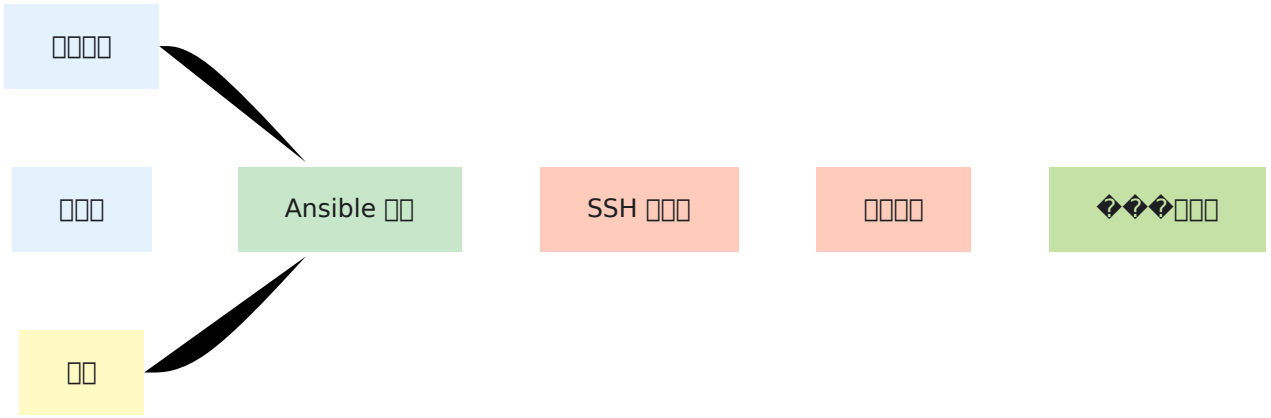
□□□□□□□□□□□□□□□□□

4. □□□

□□ □□□

□□□□□□□□□

□□□□



□□□□

1. □□□□□□

□□□□□□□□□□□□□□□□□□□□

□□□□□□□□ □□□□□□□□□□□□□□□ IP □□□□□□□□□□□□□□□□IP □□□□□□□□□□□□□□□

Proxmox □□□ □□□ Proxmox □□□□□□□□ Proxmox VM/LXC □□ □□□□□□□ VM/□□□□□

□□□□□□□□□ □ □□□□

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

2. □□□□□

□ `group_vars` □□□□□□□□□□

```yaml
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 访问网关

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

OmniCall（语音和消息平台）

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

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

OmniCharge/OmniCRM

- **CRM 平台** - 客户关系管理、自助注册、计费

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

支持服务

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

参见：部署架构概述

包管理

我们使用混合包分发模型：

□□□□ APT □

□��□ Omnitouch □□□□ Debian □□ `.deb` □□□□□□

- □□□□ CI/CD □□□□□□□□□
- □□□□□□□□
- □□□□□□□□□

APT □□□□

□□□□□□□

1. □□ **APT** □□ - □□□□□□□□□□□□□□□□□□
2. □□□□ - □□□□ Omnitouch □□□□□□□□

□□□APT □□□□

□□□□

□□ Omnitouch □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□□□□□□□□□□
- □□□□□□□□/□□□□
- □□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

□□□□

□□□ Ansible □□□□□□□

- □□□□□
- □□□□
- □□□□

- □□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□ Git □□□□

- □□□□□□□
- □□□□□□□□
- □□□□□□□□

□□□□□□□

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

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

□□□□

□□□ Ansible □□□□□□□□□□ Python □□□□□□□□□□□□□□□□

1. □□ Python □□□□

□ Ansible □□□□□□□□□□ Python □□□

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

2. □□□□□□

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□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□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

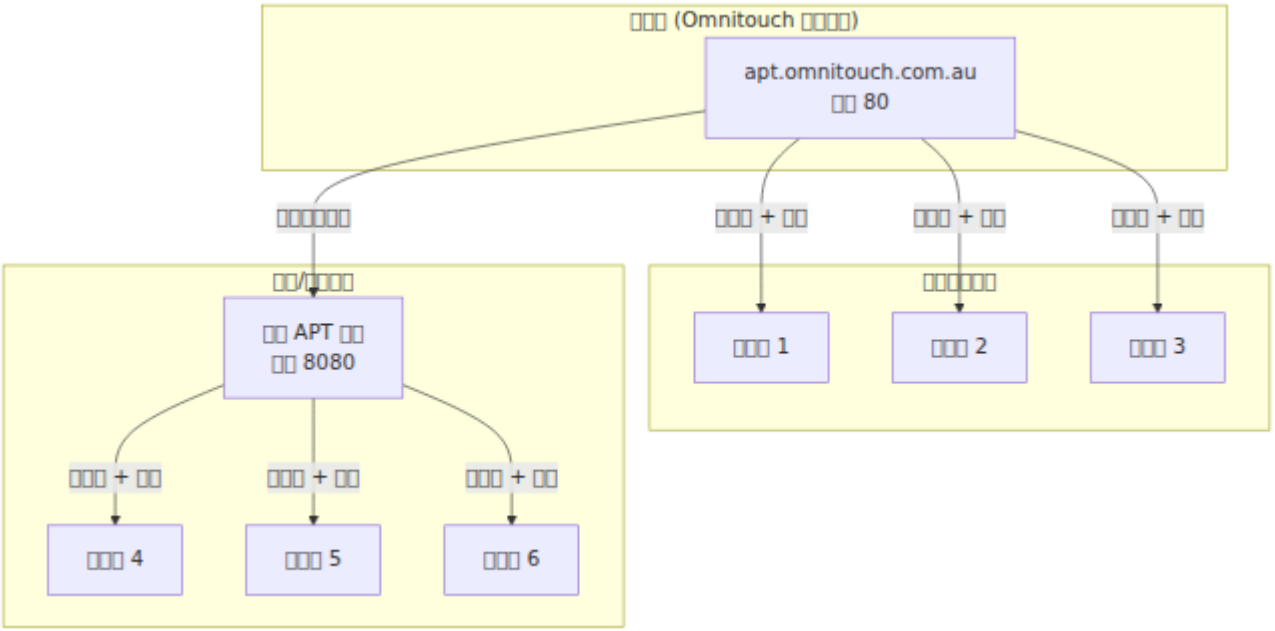
Omnitouch APT 系统为所有部署提供软件包分发。提供两种类型的内容：

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

□□□□□□□□□□□□□□□ Omnitouch IP □□□□□□□□□

□���	□□
IPv4	`144.79.167.0/24`
IPv4	`160.22.43.0/24`
IPv6	`2001:df3:dec0::/48`
ASN	`AS152894`

□□□□ **Omnitouch** □□□□□□□□□

□□	□□	□□	□□
APT □□□	80	TCP	□□□□□
APT □□□	53	TCP/UDP	`apt.omnitouch.com.au` □ DNS □□
□□□□□□	123	UDP	□□□□□□ NTP □□□□
□□□□□□	53	TCP/UDP	□□□□□□ DNS □□

□□ HTTP (TCP/80)□NTP (UDP/123) □ DNS (TCP+UDP/53) □□□□□□ Omnitouch IP □□□

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）

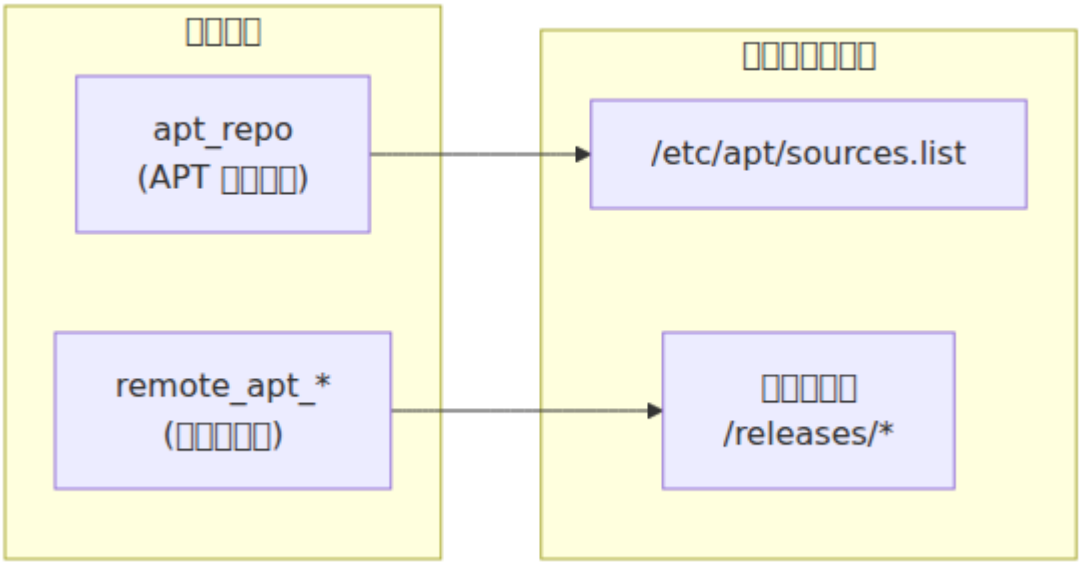
APT 软件包源（配置在 `/etc/apt/sources.list`）：

```
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
```

□□□□

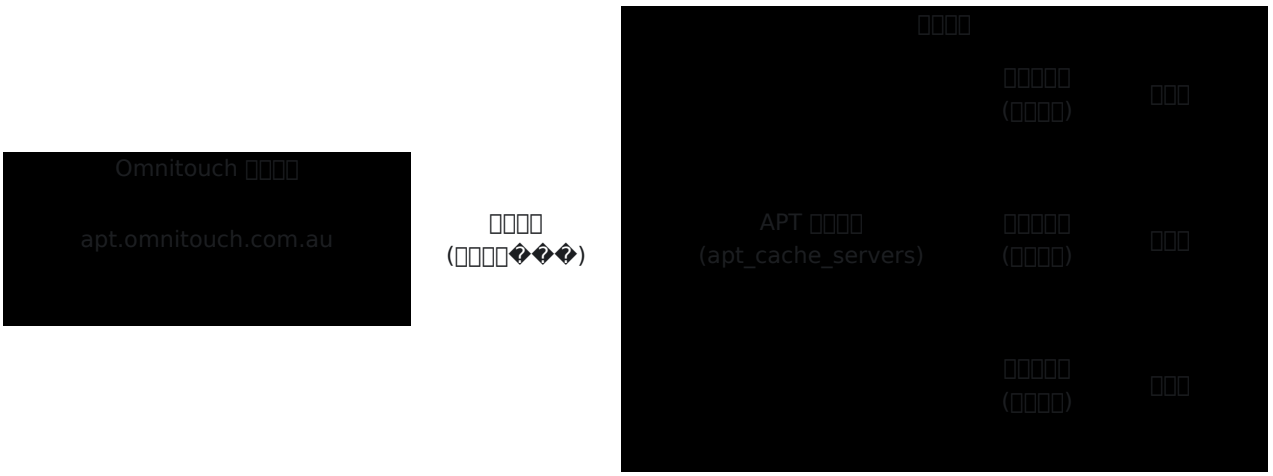


□□□□□ HTTP □□□□□□□□ APT □□□□□□□□□□□□□□□□Ubuntu □□□□□□□ Omnitouch □□□□□ □□□□□□□□□□□□□□□ Ubuntu □□□

□□ 2□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□ APT □□□□ Omnitouch □□□□□□□

□□



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`)

□□	□□	□□	□□	□□
`apt_repo.apt_server`	□□□□	□	□□□□	□□□□

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

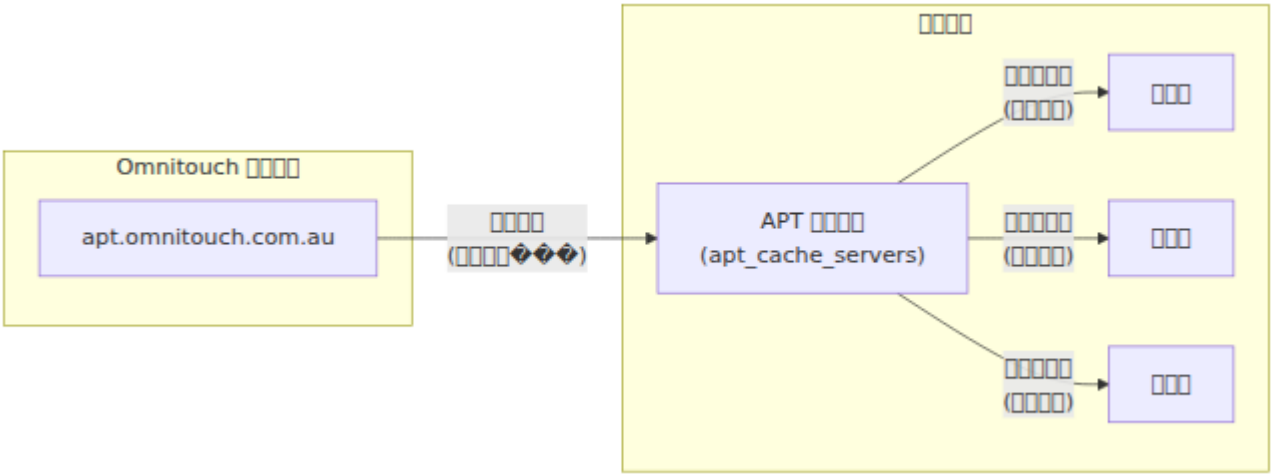
缓存镜像使用递归 wget 下载从 Omnitouch APT 服务器同步 所有内容：

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。

□□□□



□□□□□□□ `wget --recursive` □ HTTP □□□□□□□□ Omnitouch APT □□□□□□□□□□□□□□□□□□□□□□/□□□□□□□□□□□□□

□□□□

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

1. □□□□□□□□ `use_apt_cache: true`□□□□□□□□□
2. □□□□□□□□□□□ `ansible_host` IP □□ `apt_repo.apt_server`

□□□□□□□

```yaml
apt_cache_servers:
  hosts:
    apt-cache-01:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # □□□□□□□ Omnitouch □□□□□□□
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_user: "your-username"
    remote_apt_password: "your-password"
```

□□□□□□□□□

- 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）。

```yaml
all:
  vars:
    use_apt_cache: true

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

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

□□□□□□□□□□□

1. □□□□□ (`10.179.1.114`)□
   - □□□ `vars:` □□□□ `remote_apt_*`
   - □ `http://customer-username:customer-secure-token@apt.omnitouch.com.au:80/` □□□□□□□
   - □□ nginx □□□ 8080 □□□□□
2. □□□□ (`customer-hss01`□`customer-mme01`□`customer-dns01`)□
   - □□□□□ `apt_cache_servers` □□□
   - □□□□□ `use_apt_cache: true`
   - □□□□ `apt_repo.apt_server: "10.179.1.114"`
   - □□□`deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
   - □□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□

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

□□□□□□ Omnitouch APT □□□□□□□□□□

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

□□□□ `wget --timestamping`□□□□□□□□□□□□□□□□□□□□□□□□□□□□

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）

□□□□□□□□Node Exporter□Zabbix □□

□□□Ansible □□□□ `/releases/` □□□□□□□□□

□□□□□□

- `remote_apt_*` □□□□□
- `remote_apt_user` □ `remote_apt_password` □□□
- □ `use_apt_cache: false` □□□ `remote_apt_server`

□□□□□

1. □□□□ `remote_apt_*` □□□□□
2. □□□□□□□ Omnitouch □□□□□
3. □□ `remote_apt_server` □□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□

□□□□□□

- □□□□□□□□□□□□
- `remote_apt_*` □□□□□
- □□□□□□□ Omnitouch □□□□□

□□□□□

1. □□□□□□□□□□□□□□ 80 □□□ `apt.omnitouch.com.au`
2. □□ `remote_apt_*` □□
3. □□□□□□□□□□□□□□

□□□

- □□□□□□ — □□□□□□□
- □□□□ — □□□□□□

- 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

□□□□

□□

□□□□□□□□□□□□□□□ OmniCore □□□□□□□□□□□□□□□□

常见主机参数

#待办事项 - 仅需查看 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。现代部署使用最少的全局变量，大多数配置在角色默认值中。

必需□全局变量

认证与访问

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

替代方案：使用 SSH 密钥而不是密码：

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

客户身份

```
customer_name_short: omnitouch
customer_legal_name: "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 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。用于直径领域构建。

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 重写)
```

显示: 在 UE 设备的设置 > 移动□络中显示的□络名称。

DNS 配置

```
netplan_DNS: False               # 使用 systemd-resolved 而不是 netplan DNS
```

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"
```

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
```

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 □□□□

```
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

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

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

netmask: 255.255.255.0
```

相关文档

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

产品文档

有关详细的操作指南和高级配置：

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

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

部署架构概述

概述

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

有关详细组件放置、IP地址分配指南和公共IP处理，请参见IP规划标准。

完整部署示例

0. 基础设施供应（可选）

对于Proxmox部署，在配置之前供应虚拟机/LXC：

```
# 在Proxmox上部署虚拟机
ansible-playbook -i hosts/Customer/hosts.yml services/proxmox.yml

# 或部署LXC容器（仅限实验/测试）
ansible-playbook -i hosts/Customer/hosts.yml services/proxmox_lxc.yml
```

参见：Proxmox VM/LXC部署

1. 基础设施定义（主机文件）

```
# 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，您会有一个包含您的OmniMessage SMSc配置的文件夹，您的TAS连接的SIP中继将位于这里，所有的Diameter路由逻辑等等。

参见：组变量配置

3. 包分发（APT缓存）

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

参见：APT缓存系统

4. 许可证配置

```
# 指向许可证服务器的组件
license_server_api_urls: ["https://10.10.2.150:8443/api"]
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`来部署单个组件，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。

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

# 或□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
- 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 - 包管理
- 许可证服务器 - 许可证管理

产品文档

有关配置每个组件的详细信息：

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

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。

这里存放着您特定客户的配置 - SIP 中继、Diameter 路由规则、SMS 路由逻辑、拨号计划以及您不想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。要为特定部署进行自定义，请将您的自定义文件放入 `group_vars` 并在您的主机文件中引用它们。

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

示例 **1:** 自定义配置模板 **(OmniMessage)**

某些组件接受自定义 Jinja2 配置模板。

文件结构

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

在主机文件中引用

```
omnimessage:
  hosts:
    customer-smsc-controller01:
      ansible_host: 10.10.3.219
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      smsc_template_config: smsc_controller.exs    # 在 group_vars 中
引用您的模板文件名
```

发生了什么：

1. Ansible 找到 `smsc_template_config: smsc_controller.exs`
2. 在 `hosts/Customer/group_vars/smsc_controller.exs` 中查找
3. 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，将使用角色的默认模板。

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

示例 2：配置文件集合（OmniTAS 网关和拨号计划）

某些组件使用配置文件目录。

文件结构

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

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/` 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"`

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

```
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，将使用角色的默认模板。

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

示例 4：自定义配置模板（OmniMME）

移动管理实体接受自定义运行时配置模板。

文件结构

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

在主机文件中引用

```yaml
omnimme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.3.51
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      mme_template_config: mme_runtime.exs.j2    # 在 group_vars 中引用
您的模板文件名
```

发生了什么：

1. Ansible 找到 `mme_template_config: mme_runtime.exs.j2`
2. 在 `hosts/Customer/group_vars/mme_runtime.exs.j2` 中查找
3. 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，将使用角色的默认模板。

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

真实世界目录结构示例

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

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`）
拨号计划（自动）	applicationserver	`dialplan/` 路由 XML 文件[
`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
- 覆盖默认值的客户特定设置

□ □□□□ **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

实用剧本

概述

该存储库包含多个实用剧本，用于维护、监控和操作任务。这些剧本补充了主要的部署剧本，提供日常管理功能。

健康检查工具

健康检查工具生成一个 HTML 报告，显示所有 OmniCore 组件的系统健康、服务状态、正常运行时间和版本信息。

作为 **`services/all.yml`** 剧本□一部分自动运行。

使用方法

手动运行

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

输出

报告生成在 `/tmp/health_check_YYYY-MM-DD HH:MM:SS.html`

在任何□页浏览器中打开以查看。

报告内容

HTML 报告显示:

□□□□

- □□□□ **IP** □□
- □□/□□□□□ `host_vm_network` □□□□□□□□□□ N/A□
- **CPU**□vCPU □□□
- **RAM**□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□

□□□□

- □□□□□□□□/□□□□□□□□□□□
- □□□□□□□□□
- □□/□□□□

HSS Diameter □□□

- □□□□□□□□□□□□□/□□□□
- **Diameter** □□□□□□□IP□□□□□□□□
- □ HSS □□□□□□□□□□ 9568□

□□□□□□

□□□□□□

□□□□ (`services/common.yml`)

- □□□□□□□□□□□□□□□□
- □□□□□□□SSH □□□□□□□NTP
- □□□□□□□□□□
- □□□□□□□□□□□□□

```
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`)

- 更新网络接口 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）是定义您整个蜂窝网络部署的中央配置文档。它指定：

- 要部署的网络功能
- 它们运行的位置（IP 地址、网络段）
- 它们的配置方式（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 节点
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15
      gateway: 10.10.1.1
      host_vm_network: "vmbr1"
      mme_code: 1
      network_name_short: Customer
      tac_list: [600, 601, 602]

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

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

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

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

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

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

常见主机参数

网络配置

每个主机通常包括：

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

注意：有关 IP 地址规划和网络分段策略的指导，请参见 IP 规划标准，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
```

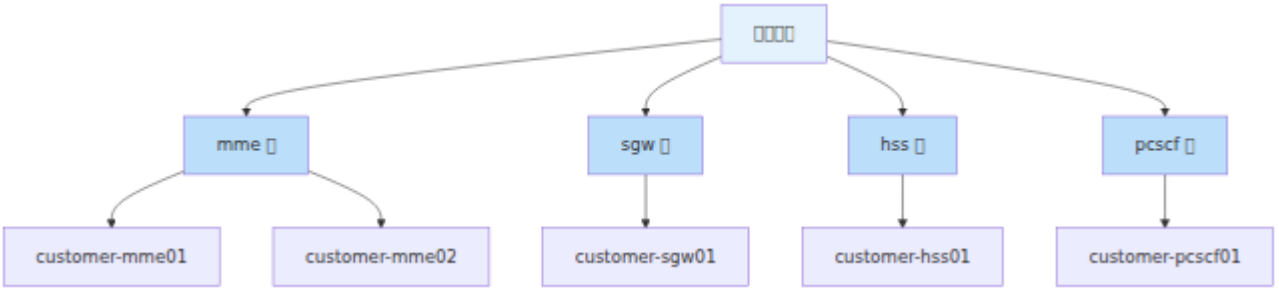
全局变量部分

`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：

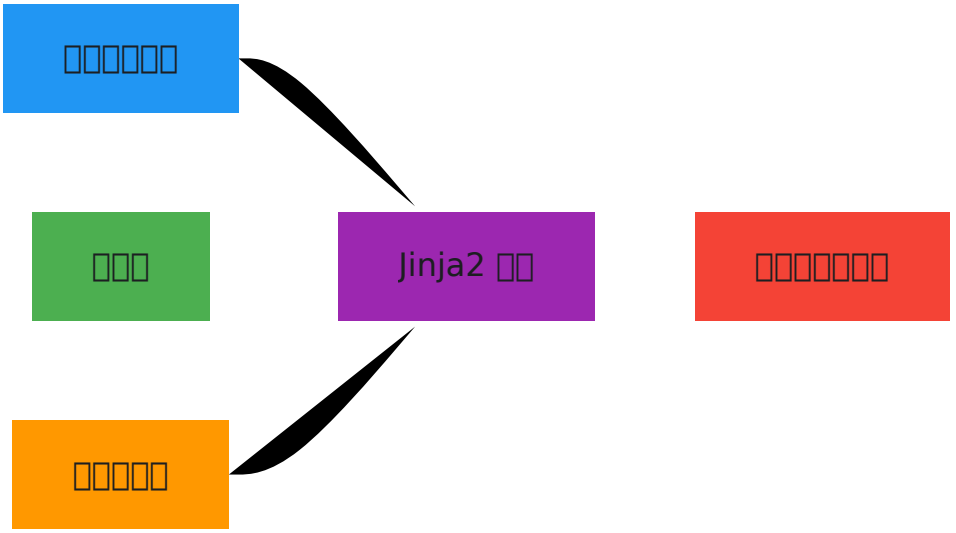


当您运行针对 `mme` 的剧本时，它会应用于 `mme:hosts:` 部分中的所有主机。

使用 **Jinja2** 模板进行配置

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

Jinja2 工作原理



示例模板用法

主机文件定义：

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

Jinja2 模板（在角色中）：

```
# 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** - □□□□□□□□□□
- **OmniUPF** - □□□□□□□
- **OmniDRA** - Diameter □□□□
- **OmniTWAG** - □□ WLAN □□□□

OmniCall □□□

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

OmniCharge/OmniCRM□

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

□□□□

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

□□□

1. □□□□□□□□□□□□□□
2. □□□□ PLMN □□□□□

3. 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，以适当分隔网络功能，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网用于内部网络：

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

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

- 所有主机在一个子网中：如果符合您的部署需求，可以将所有组件放置在单个子网中
- 每种主机类型在其自己的子网中：可以为每种组件类型创建单独的子网（一个用于 MME，一个用于 HSS，等等）
- 自定义分组：可以根据您的具体要求将主机组织成任何子网结构
- 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

推荐的四子网方法提供了最佳的 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网络拓扑、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`

```yaml
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 □□
- □□□□□□□
- RAN □□
- Omnitouch □□□□ CBC□□□□□□□□□ - □□□□
- APT □���□□□ - □□□□

□□□ `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 □□□□□□□□ /24□

□□□ IMS □□□□□□□□□□ SIP □□□

□□□

- OmniCSCF S-CSCF□□□□□□□□□□□□□
- OmniCSCF I-CSCF□□□□□□□□□□□□□
- OmniTAS□□□□□□□□ / □□□□□□
- OmniMessage□□□□□□□SMPP□IMS□
- OmniSS7 STP□SS7 □□□□□□
- OmniSS7 HLR□□□□□□□□ - □□ 2G/3G
- OmniSS7 IP-SM-GW□MAP SMSc□
- OmniSS7 CAMEL □□

□□□ `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 □□□□□□□□ /24□

□□□ □□□□□□□□□ IMS □ DNS

□□□

- OmniCSCF P-CSCF□□□□□□□□□□□□
- XCAP □□□
- □□□□□□□□□
- □□ DNS

□□□ `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"
```

□□□□

OmniCore □□□□□□□□□□□□□□□

□□ 1□□□/□□□□□□□□□□□□□□□

□□□□□□□□□□ NIC □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

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，以及适当的网关和子网掩码。

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

```yaml
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

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：

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

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

模板参考

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 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 可以有自己的网关和静态路由

主接口配置

主接口 (eth0) 自动配置使用：

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

DNS 服务器自动设置为：

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

辅助接口

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

结构

```yaml
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，将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，本指南详细说明了如何使用 `proxmox` play 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 - 次要 IP 和多网卡设置
- 部署架构 - 完整的部署工作流程

LXC 与 VM

LXC 容器：

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

虚拟机 **(KVM)**：

- 完整虚拟化，具有专用内核
- 完全隔离
- 可以运行内核模块和自定义网络
- 更高的资源开销
- 推荐用于生产
- **UPF** 部署所需

使用案例：

- 虚拟机：生产站点、UPF、所有网络功能
- **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 失败时用于控制台访问）
- 通过 Ansible 克隆时，凭据会从主机文件中的 `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` 标志仅在比本地副本更新时下载

替代方案：从 **ISO** 手动创建模板

如果云镜像不可用或需要自定义安装：

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**：在虚拟机内部 **-** 清理和准备

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

# 清理机器特定数据
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 历史记录并关机
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

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
```

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 services/proxmox.yml
```

部署 **LXC** 容器

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

删除虚拟机/**LXC**

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

行为

proxmox.yml

- 检查 Proxmox 中是否已存在同名虚拟机
- 使用轮询方式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
- 使用轮询方式在节点之间分配 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
- 配置网络的 netplan
- 自动启动容器
- 排除 UPF 主机

proxmox_delete.yml

- □□□□

```
# □□□□□□ (proxmox.yml) - □□□□□□□□
proxmoxServers:
  pve-node-01: { ... }
  pve-node-02: { ... }
  pve-node-03: { ... }

# □□ LXC (proxmox_lxc.yml) - □□□□□□
proxmoxNodeName: ['pve-node-01', 'pve-node-02', 'pve-node-03']
```

□□□□

□□□□ LXC □□□□□□□

- □□/□□□□□□□□□□□ Ansible □
- □□□□□`site_name` □□

□□□

```
site_name: "melbourne-prod"

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

□□□ □□□/LXC □□□□□`mme`□`melbourne-prod`

□���□ Proxmox UI □□□□□□□□□/□□□

□□□□□

□□□□□□□□□□□□

```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
```

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□
`health_check.yml`	为所有服务生成全面的健康报告
`restore_hss.yml`	从备份恢复 HSS 数据库和/或配置
`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
`updateMtu.yml`	将□络接口的 MTU 设置为 9000（巨型帧）
`systemctl status.yml`	检查 EPC 组件上的服务状态

健康检查

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

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

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

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

收集□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、仅恢复配置或两者都恢复。

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

备份文件格式

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

IP 计划生成器

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

从清单生成网络文档，包括：

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

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

输出文件

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

端口审计

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

审计部署中的所有监听端口并生成文档。

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

输出文件

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

收集□数据

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

本地捕获检索

文件：`util_playbooks/getLocalCapture.yml`

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

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

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

用户管理

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

从清单中的所有主机删除本地用户帐户。

```
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。

MTU 配置

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

将所有主机的 `ens160` 接口的 MTU 设置为 9000（巨型帧）。

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

注意**:** 此剧本是为 `ens160` 接口硬编码的。如果您的环境使用不同的接口名称，请修改剧本。

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（显示将会更改的内容）
`--diff`	显示文件差异

示例

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

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

# 生成带有详细输出的 IP 计划
ansible-playbook -i hosts/acme/host_files/production.yml util_playbooks/ip_plan_generator.yml -v
```