Introduction to Ansible Deployment at Omnitouch

Overview

Omnitouch Network Services uses Ansible as its infrastructure automation platform to deploy complete cellular network solutions (4G/5G) in a consistent, repeatable, and automated manner. This document provides an overview of how we leverage Ansible to orchestrate complex telecom deployments.

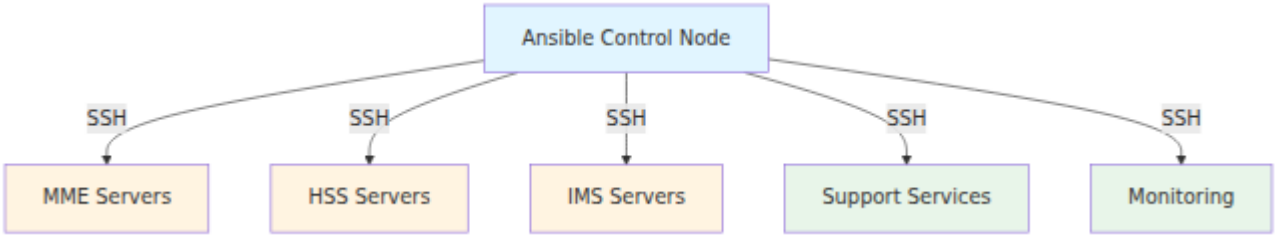
What is Ansible?

Ansible is an open-source automation tool that allows you to:

- Configure systems
- Deploy software
- Orchestrate complex workflows
- Manage infrastructure as code

Ansible uses a declarative approach - you describe the **desired state** of your systems, and Ansible ensures they reach that state.

How Omnitouch Uses Ansible



Key Concepts

1. Inventory (Hosts Files)

Defines **what** systems to manage. Each customer deployment has a hosts file that describes:

- All virtual machines in the network
- Their IP addresses
- Network configuration
- Service-specific parameters

Host files are what you will be working with to define your network.

See: Hosts File Configuration

2. Roles

Defines **how** to configure each component. Roles are reusable units that contain:

- Tasks (steps to execute)
- Templates (configuration file templates)
- Handlers (actions triggered by changes)
- Variables (default configuration values)

Example roles for OmniCore components: `omnihss`, `omnisgwc`, `omnipgwc`, `omnidra`, etc

These are defined by the ONS team, while you can edit them, there's generally cleaner ways to make any tweaks you might need from within your hosts file.

3. Playbooks

Orchestrates **when** and **where** roles are applied:

```
- name: Deploy EPC Core
  hosts: mme
  roles:
    - common
    - omnimme
```

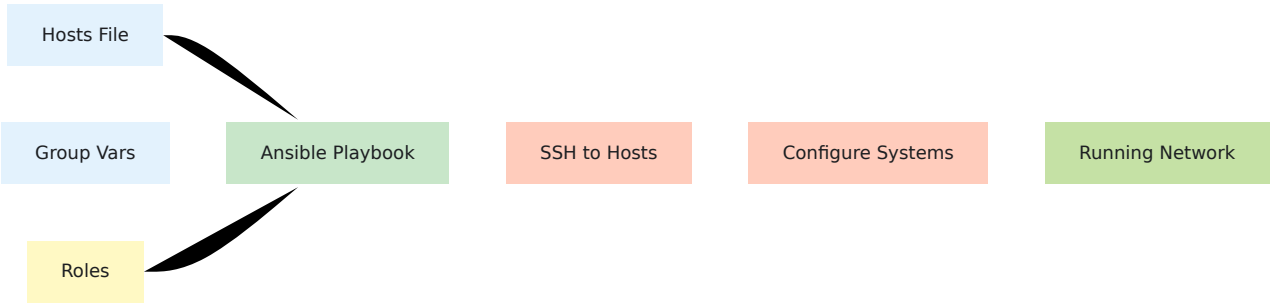
We use these essentially as groups for the roles.

4. Group Variables

Provides **customer-specific configuration** that overrides role defaults. This is where customer customization happens without modifying the base roles.

See: Group Variables and Configuration

Deployment Architecture



The Deployment Process

1. Define Infrastructure

Create a hosts file describing your network topology:

Planning Note: Before defining infrastructure, review the IP Planning Standard for guidance on network segmentation, IP address allocation, and subnet organization.

Proxmox Users: If deploying on Proxmox, see Proxmox VM/LXC Deployment for automated VM/container provisioning.

See: Hosts File Configuration and Configuration Reference

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

2. Customize Configuration

Set customer-specific variables in `group_vars`:

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

#ToDo - Add link here to conifg reference for complete list

3. Run Playbooks

Deploy the network:

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

4. Automated Deployment

Ansible will:

- Create/provision VMs (if using Proxmox/VMware integration)
- Configure networking
- Install software packages from APT cache
- Deploy application code
- Configure services with customer settings

- Start services
- Validate deployment

Key Components We Deploy

OmniCore (4G/5G Packet Core Platform)

- **OmniHSS** - Home Subscriber Server
- **OmniSGW** - Serving Gateway (Control plane)
- **OmniPGW** - Packet Gateway (Control plane)
- **OmniUPF** - User Plane Function
- **OmniDRA** - Diameter Routing Agent
- **OmniTWAG** - Trusted WLAN Access Gateway

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

OmniCall (Voice & Messaging Platform)

- **OmniCall CSCF** - Call Session Control Function (P-CSCF, I-CSCF, S-CSCF)
- **OmniTAS** - IMS Application Server (VoLTE/VoNR services)
- **OmniMessage** - SMS Center (SMS-C)
- **OmniMessage SMPP** - SMPP protocol support
- **OmniSS7** - SS7 signaling components (STP, HLR, CAMEL)
- **VisualVoicemail** - Voicemail functionality

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

OmniCharge/OmniCRM

- **CRM Platform** - Customer relationship management, self-signup, billing

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

Support Services

- **DNS** - Network DNS resolution
- **License Server** - License management
- **Monitoring** - Prometheus, Grafana

See: Deployment Architecture Overview

Package Management

We use a hybrid package distribution model:

Pre-compiled APT Packages

All Omnitouch software is distributed as Debian packages (`.deb` files):

- Built from source in our CI/CD pipeline
- Versioned and tested
- Hosted on package repositories

APT Cache System

Customers can choose between:

1. **Local APT Cache** - Mirror of required packages on-site for offline deployment
2. **Public Repository** - Direct access to Omnitouch's hosted package repository

See: APT Cache System

License Management

All Omnitouch software components require valid licenses managed through a central license server:

- Components check license validity on startup
- Features are enabled/disabled based on license
- License server can be local or cloud-hosted

See: License Server

Benefits of This Approach

Repeatability

The same Ansible playbooks can deploy:

- Development labs
- Testing environments
- Production networks
- Customer sites

Consistency

Every deployment uses the same tested configurations, reducing human error.

Version Control

Infrastructure is defined as code in Git:

- Track all changes
- Review before deployment
- Roll back if needed

Customization Without Complexity

Customers can customize their deployment through `group_vars` without modifying core roles.

Rapid Deployment

Deploy a complete cellular network in hours instead of days or weeks.

Getting Started

Prerequisites

Before running Ansible playbooks, you need to set up a Python virtual environment and install the required dependencies.

1. Create a Python Virtual Environment

Create an isolated Python environment for the Ansible deployment:

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

2. Activate the Virtual Environment

Activate the virtual environment:

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

On Windows, use:

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

3. Install Required Packages

Install all dependencies from the requirements.txt file:

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

This will install Ansible and all necessary Python packages for Omnitouch deployment automation.

Note: Keep the virtual environment activated whenever running Ansible commands. You can deactivate it when finished by running `deactivate`.

Deployment Steps

1. Review the Hosts File Configuration to understand how to define your network
2. Learn about Group Variables for customization
3. Understand the APT Cache System for package management
4. Review the Deployment Architecture to see how everything fits together
5. Deploy!

Next Steps

- IP Planning Standard - **Plan your network architecture and IP allocation**
- Hosts File Configuration - Learn how to define your network topology
- APT Cache System - Understand package distribution
- License Server - Learn about license management
- Deployment Architecture Overview - See the complete picture
- Group Variables Configuration - Customize your deployment
- Utility Playbooks - Operational tools for health checks, backups, and maintenance

APT Repository & Package Distribution

Overview

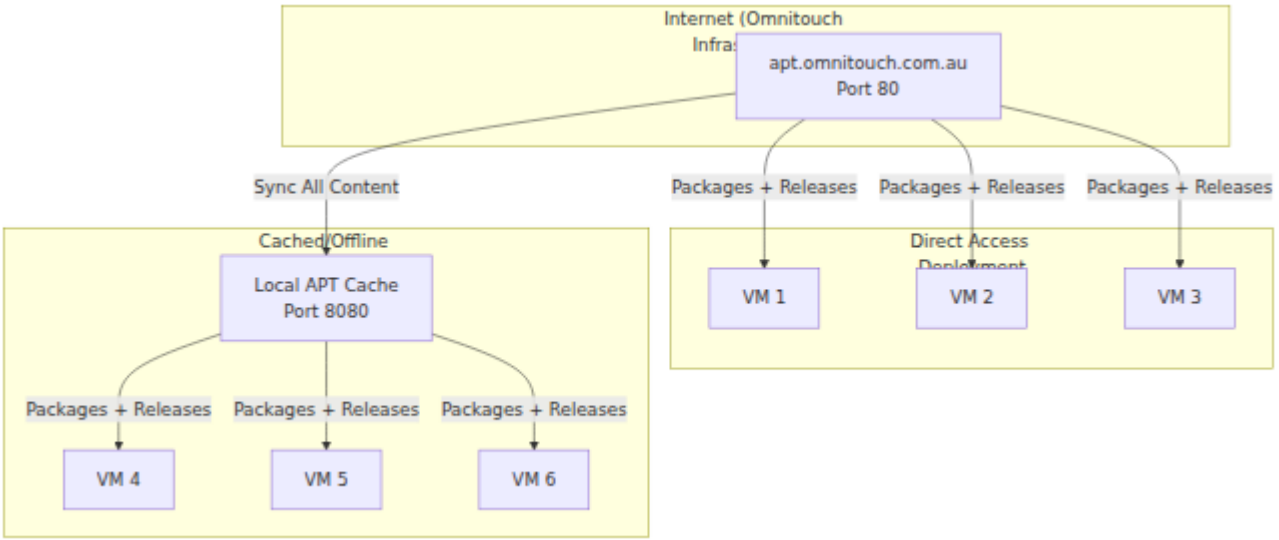
The Omnitouch APT system provides package distribution for all deployments. Two types of content are served:

1. **APT Packages** — Debian packages installed via `apt install`
2. **Binary Releases** — Pre-built binaries downloaded directly (Prometheus exporters, agents, etc.)

Two deployment models are supported:

1. **Direct Access** — VMs pull packages directly from `apt.omnitouch.com.au`
2. **Local Cache Mirror** — A local server syncs from Omnitouch and serves packages to VMs (for offline/airgapped deployments)

Architecture



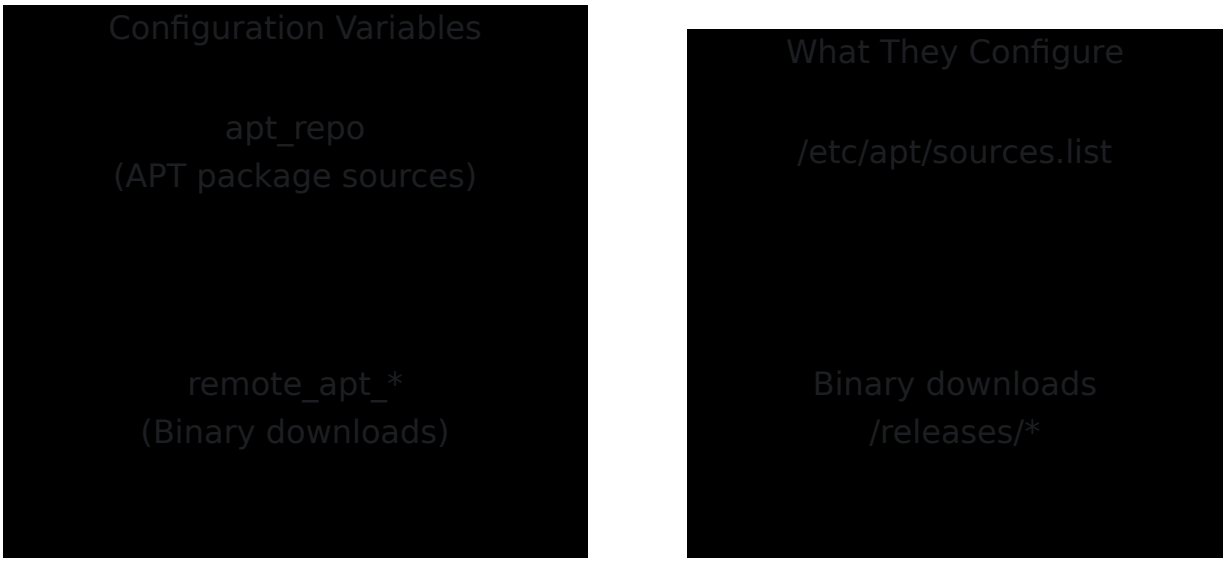
Content Served

The APT server hosts all content required for deployments:

Content Type	Description	Path
Omnitouch Packages	Custom-built `.deb` packages (omnihss, omnimme, etc.)	`/dists/<distro>/`
Ubuntu Packages	Cached Ubuntu packages with all dependencies	`/<distro>/pool/main/`
GitHub Releases	Pre-built binaries (Prometheus, Grafana, Homer, etc.)	`/releases/<org>/<repo>/`
Source Tarballs	Source archives for web apps (CGrateS_UI, speedtest)	`/repos/`
Third-Party Packages	Galera, FRR, InfluxDB, KeyDB, etc.	`/releases/<vendor>/`

Configuration Variables

Two separate variable sets control package distribution. Understanding their purposes is essential for correct configuration.



Variable Purposes

Variable Set	Purpose	Used For
`apt_repo`	Configures APT package sources	`/etc/apt/sources.list` and `/etc/apt/sources.list.d/*.list`
`remote_apt_*`	Configures binary download URLs	Downloading files from `/releases/` path (Node Exporter, Zabbix, Nagios, etc.)

When Each Variable Set Is Used

Scenario	APT Sources (`apt_repo`)	Binary Downloads (`remote_apt_*`)
`use_apt_cache: true`	Uses `apt_repo.apt_server`	Uses `apt_repo.apt_server`
`use_apt_cache: false`	Uses `apt_repo.*` with credentials	Uses `remote_apt_*` with credentials

When `use_apt_cache: false`, both variable sets are required.

Option 1: Direct Access

For deployments with internet connectivity, VMs pull packages directly from the Omnitouch APT server.

Network Requirements

Source IP Whitelisting: Your public IP address must be whitelisted on the Omnitouch APT server. During setup, provide your source subnets to Omnitouch. In return, you will receive:

- **Username** and **password** for HTTP Basic Auth
- **FQDN** for the APT server

Firewall Requirements: Outbound access to the following Omnitouch IP ranges must be allowed:

Network	Range
IPv4	`144.79.167.0/24`
IPv4	`160.22.43.0/24`
IPv6	`2001:df3:dec0::/48`
ASN	`AS152894`

Services requiring access to Omnitouch infrastructure:

Service	Port	Protocol	Purpose
APT Server	80	TCP	Package downloads
APT Server	53	TCP/UDP	DNS resolution for `apt.omnitouch.com.au`
License Server	123	UDP	NTP time synchronization for license validation
License Server	53	TCP/UDP	DNS resolution for license validation

Ensure HTTP (TCP/80), NTP (UDP/123), and DNS (TCP+UDP/53) traffic is allowed to the Omnitouch IP ranges.

Configuration

```yaml
all:
  vars:
    use_apt_cache: false

    # APT package sources configuration
    # Configures /etc/apt/sources.list for apt install commands
    apt_repo:
      apt_server: "apt.omnitouch.com.au"
      apt_repo_username: "your-username"
      apt_repo_password: "your-password"

    # Binary downloads configuration
    # Used for downloading files from /releases/ path
    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"
```

Parameters

APT Package Sources (`apt_repo`)

Parameter	Type	Required	Default	Descriptio
`apt_repo.apt_server`	String	Yes	-	APT server hostname o IP address
`apt_repo.apt_repo_username`	String	Yes	-	HTTP Basic Auth username for APT sources
`apt_repo.apt_repo_password`	String	Yes	-	HTTP Basic Auth password fo APT source

Binary Downloads (`remote_apt_*`)

Parameter	Type	Required	Default	Description
`remote_apt_server`	String	Yes	-	Server hostname or IP for binary downloads
`remote_apt_port`	Integer	No	`80`	Server port for binary downloads
`remote_apt_protocol`	String	No	`http`	Protocol (`http` or `https`)
`remote_apt_user`	String	Yes	-	HTTP Basic Auth username for downloads
`remote_apt_password`	String	Yes	-	HTTP Basic Auth password for downloads

General

Parameter	Type	Required	Default	Description
`use_apt_cache`	Boolean	Yes	-	Must be `false` for direct access

URL Patterns (Direct Access)

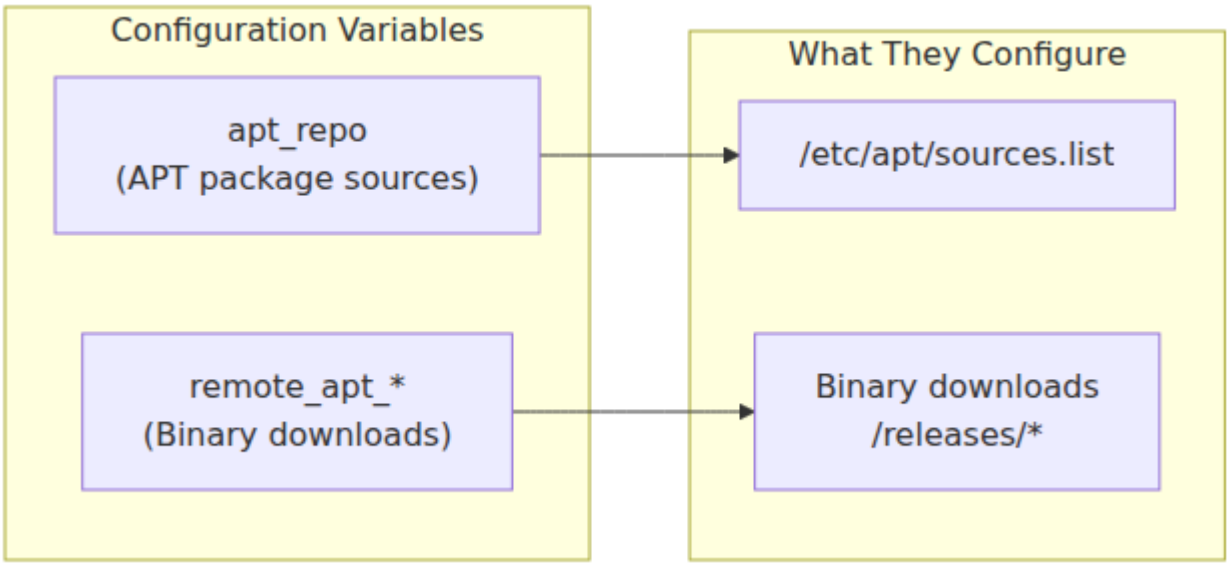
APT Package Sources (configured in `/etc/apt/sources.list`):

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

Binary Downloads (used by Ansible `get_url` tasks):

```
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
```

How It Works

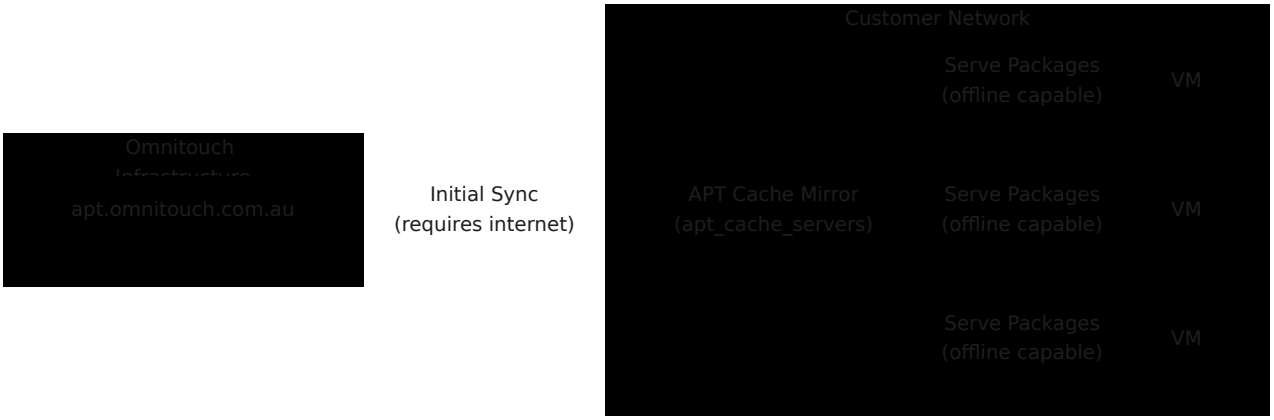


VMs authenticate with HTTP Basic Auth for both APT packages and binary downloads. Ubuntu system packages are also served from the Omnitouch server (pre-cached), so VMs do not need access to Ubuntu mirrors.

Option 2: Local Cache Mirror

For offline, airgapped, or bandwidth-constrained deployments, deploy a local APT cache that syncs all content from Omnitouch.

Architecture



Configuration

Define the cache server in your hosts file with its repository configuration:

```
apt_cache_servers:
  hosts:
    customer-apt-cache:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # Cache server syncs packages from authenticated repository
    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  # Auto-set when apt_cache_servers group
exists
    # apt_repo.apt_server: auto-derived to 192.168.1.100 (first
cache server)
```

How it works:

- **Cache server** (`192.168.1.100`): Uses `remote_apt_*` credentials to sync packages from `apt.omnitouch.com.au:80`

- **All other hosts**: Automatically derive `apt_repo.apt_server: "192.168.1.100"` and pull from cache at port `8080` without credentials

Parameters

APT Package Sources (`apt_repo`)

Parameter	Type	Required	Default	Descrip
`apt_repo.apt_server`	String	Yes	Auto-derived	Local cache IP. Automat derived fro `apt_cache_` host if not specified.
`apt_repo.apt_repo_username`	String	No	-	Not require using cache auth neede
`apt_repo.apt_repo_password`	String	No	-	Not require using cache auth neede

Cache Server Sync (`remote_apt_*`)

These variables configure how the cache server syncs content from Omnitouch:

Parameter	Type	Required	Default	Description
`remote_apt_server`	String	Yes	-	Omnitouch APT server to sync from
`remote_apt_port`	Integer	No	`80`	Omnitouch APT server port
`remote_apt_protocol`	String	No	`http`	Protocol for sync connection
`remote_apt_user`	String	Yes	-	Credentials for syncing from Omnitouch
`remote_apt_password`	String	Yes	-	Credentials for syncing from Omnitouch

General

Parameter	Type	Required	Default	Description
`use_apt_cache`	Boolean	No	`true`	Automatically set to `true` when `apt_cache_servers` group exists
`apt_cache_port`	Integer	No	`8080`	Port the local cache server listens on

URL Patterns (Cache Mode)

APT Package Sources (configured in `/etc/apt/sources.list`):

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

Binary Downloads (used by Ansible `get_url` tasks):

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

No credentials required for cache access—it uses `[trusted=yes]` APT configuration.

Deploying the Cache

1. **Provision the cache server** (VM or LXC container with 50+ GB disk)
2. **Run the cache setup playbook:**

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

3. **Verify the cache** by browsing to `http://192.168.1.100:8080/`

What Gets Synced

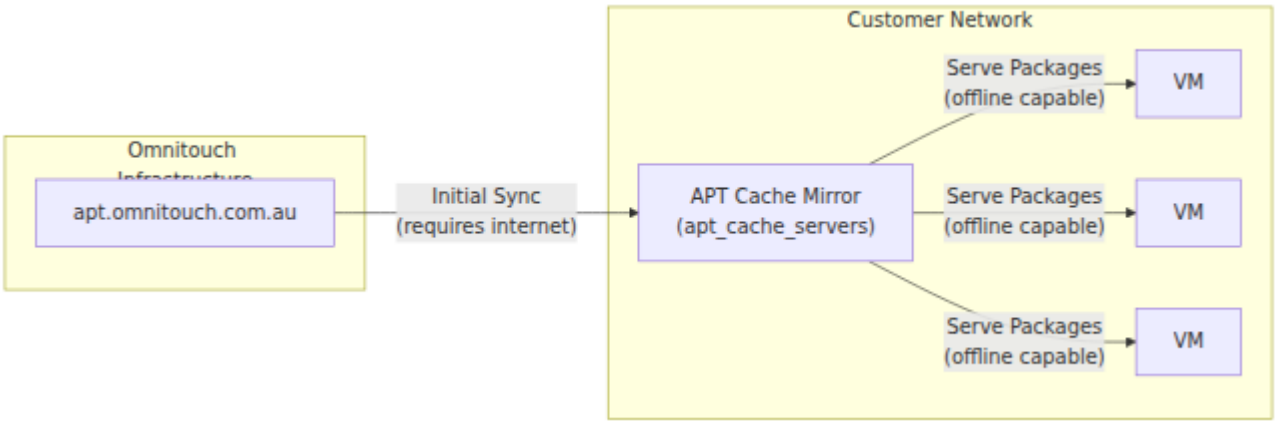
The cache mirror syncs **all content** from the Omnitouch APT server using recursive wget download:

Content directories synced:

Path	Content
`/dists/<distro>/`	APT repository metadata (Packages, Release files)
`/pool/main/`	Omnitouch custom .deb packages
`/<distro>/pool/main/`	Ubuntu packages and all dependencies
`/releases/`	GitHub releases (Prometheus, Grafana, Zabbix, etc.)
`/repos/`	Source tarballs (Erlang, Elixir, CGrateS_UI, etc.)

After initial sync, the cache can serve all packages without internet connectivity.

How It Works



The cache mirror uses `wget --recursive` with HTTP Basic Auth to download all content from the Omnitouch APT server. Subsequent syncs only download new/changed files (timestamping).

Automatic Configuration

When an `apt_cache_servers` group exists in your inventory, the system automatically:

1. Sets `use_apt_cache: true` for all hosts (unless explicitly overridden)
2. Derives `apt_repo.apt_server` from the first cache server's `ansible_host` IP

Minimal Configuration Example

```yaml
apt_cache_servers:
  hosts:
    apt-cache-01:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # Cache server syncs content from Omnitouch repository
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_user: "your-username"
    remote_apt_password: "your-password"
```

What happens automatically:

- All hosts (except cache server) get `use_apt_cache: true`
- All hosts (except cache server) get `apt_repo.apt_server: "192.168.1.100"`
- All hosts pull from `http://192.168.1.100:8080/` without credentials
- Cache server syncs packages from `http://your-username:your-password@apt.omnitouch.com.au/`

Override Automatic Behavior

To force direct access even with cache servers defined:

```yaml
all:
  vars:
    use_apt_cache: false  # Force direct access even with cache
servers defined

    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"
```

Configuration Summary

Scenario 1: Direct Access to APT Server (No Cache)

All hosts pull packages directly from the APT repository server.

```yaml
all:
  vars:
    use_apt_cache: false

    # APT package sources - used by all hosts
    apt_repo:
      apt_server: "apt.omnitouch.com.au"
      apt_repo_username: "user"
      apt_repo_password: "pass"

    # Binary downloads - used by all hosts
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_port: 80
    remote_apt_protocol: "http"
    remote_apt_user: "user"
    remote_apt_password: "pass"
```

Result: All hosts generate `deb [trusted=yes] http://user:pass@apt.omnitouch.com.au/ noble main`

Scenario 2: APT Cache Server Defined in Hosts File (Automatic)

Cache server is in your inventory and will be deployed/synced by Ansible.

```
apt_cache_servers:
  hosts:
    cache-server:
      ansible_host: 192.168.1.100
      gateway: 192.168.1.1
  vars:
    # Cache server syncs packages from authenticated repository
    remote_apt_server: "apt.omnitouch.com.au"
    remote_apt_port: 80
    remote_apt_protocol: "http"
    remote_apt_user: "user"
    remote_apt_password: "pass"

# No configuration needed in all: vars:
# Everything auto-derived from apt_cache_servers group
```

Result:

- **Cache server**: Syncs from `http://user:pass@apt.omnitouch.com.au:80/`
- **All other hosts**: Generate `deb [trusted=yes] http://192.168.1.100:8080/noble noble main` (no credentials)

Scenario 3: Remote APT Cache NOT in Hosts File (Manual)

Cache server exists elsewhere and is already set up (not managed by your Ansible).

```yaml
all:
  vars:
    use_apt_cache: true

    # Point all hosts to the external cache server
    apt_repo:
      apt_server: "192.168.1.100"  # IP of external cache server
      apt_repo_port: 8080           # Cache typically runs on port
8080

# No apt_cache_servers group needed
# No remote_apt_* needed (cache is already set up externally)
```

Result: All hosts generate `deb [trusted=yes] http://192.168.1.100:8080/noble noble main` (no credentials)

Complete Example

Here's a complete working example showing cache server configuration with multiple application hosts:

```yaml
# APT Cache Server Group
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:
    # Cache server syncs packages from authenticated repository
    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"

# Application Servers
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

# Global Configuration
all:
  vars:
    # Auto-configuration (no manual config needed):
    # - use_apt_cache: true (auto-enabled when apt_cache_servers
exists)
```

```
    # - apt_repo.apt_server: "10.179.1.114" (auto-derived from
cache server)
```

What happens during deployment:

1. **Cache server (`10.179.1.114`):**
   - Uses `remote_apt_*` from its `vars:` section
   - Downloads all packages from `http://customer-username:customer-secure-token@apt.omnitouch.com.au:80/`
   - Serves packages on port 8080 via nginx
2. **Application hosts (`customer-hss01`, `customer-mme01`, `customer-dns01`):**
   - Auto-detect `apt_cache_servers` group exists
   - Auto-set `use_apt_cache: true`
   - Auto-derive `apt_repo.apt_server: "10.179.1.114"`
   - Generate: `deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
   - Pull all packages from cache server (no credentials required)

Updating the Cache

To sync new packages or updates:

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

This incrementally syncs all content from the Omnitouch APT server:

- New Omnitouch package versions
- New Ubuntu packages and dependencies
- New GitHub releases
- Updated source tarballs

The sync uses `wget --timestamping`, so existing unchanged files are skipped, making re-sync fast.

Note: The Omnitouch APT server (`apt.omnitouch.com.au`) is the single source of truth for all packages. Run `services/apt.yml` on the apt server first to build/update packages, then run `services/apt_cache.yml` on cache mirrors to sync.

Troubleshooting

APT Update Fails with 401 Unauthorized

Symptoms:

```
Failed to fetch
http://10.179.1.115:80/noble/dists/noble/main/binary-
amd64/Packages  401  Unauthorized
```

Possible causes:

- `apt_repo` configuration defined in `all: vars:` instead of `apt_cache_servers: vars:`
- Hosts trying to access authenticated repository directly instead of cache
- Incorrect `apt_repo_username` or `apt_repo_password`
- Source IP not whitelisted on Omnitouch APT server
- Using cache credentials for direct access or vice versa

Resolution:

1. **Check configuration scope**: Ensure `apt_repo` with credentials is defined in `apt_cache_servers: vars:`, NOT in `all: vars:`
2. **Verify cache mode**: When using cache, hosts should connect to cache server (port 8080), not repository (port 80)
3. **Check generated sources**: On failing host, check `/etc/apt/sources.list.d/omnitouch.list`

    - **Correct (cache mode)**: `deb [trusted=yes] http://10.179.1.114:8080/noble noble main`
    - **Incorrect (has credentials in wrong place)**: `deb [trusted=yes] http://user:pass@10.179.1.115:80/noble noble main`
4. Verify credentials are correct for your deployment mode
5. Confirm your public IP is whitelisted with Omnitouch (if using direct access)

Binary Downloads Fail (Node Exporter, Zabbix, etc.)

Symptoms: Ansible playbook fails downloading files from `/releases/` path

Possible causes:

- `remote_apt_*` variables not configured
- Incorrect `remote_apt_user` or `remote_apt_password`
- Missing `remote_apt_server` when `use_apt_cache: false`

Resolution:

1. Ensure all `remote_apt_*` variables are defined
2. Verify credentials match those provided by Omnitouch
3. Check that `remote_apt_server` points to correct host

Cache Server Cannot Sync

Symptoms: Cache server playbook fails to download packages

Possible causes:

- Cache server has no internet access
- `remote_apt_*` credentials incorrect
- Firewall blocking outbound connections to Omnitouch

Resolution:

1. Verify cache server can reach `apt.omnitouch.com.au` on port 80

2. Check `remote_apt_*` credentials
3. Review firewall rules for outbound access

Related Documentation

- Hosts File Configuration — Inventory and variable configuration
- Configuration Reference — Complete parameter reference
- Deployment Architecture — Overall system architecture
- Proxmox Deployment — Deploying cache server as LXC container

Configuration Reference

Overview

This document provides a comprehensive reference for configuring OmniCore deployments through hosts files. Configuration is primarily defined in host inventory files with minimal group_vars overrides needed for modern deployments.

For product-specific documentation, see:

- **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

Configuration Approach

Modern OmniCore deployments use a simplified configuration model:

Key Principle: Most configuration is defined directly in the hosts file. Role defaults handle the majority of settings, with group_vars used only for specific customizations.

Network Planning

Before configuring hosts, review the IP Planning Standard for guidance on:

- Network segmentation strategies
- IP address allocation

- Subnet organization
- Public IP handling

Common Host Parameters

#ToDo - Just say to check hosts-file-configuration.md for this

Service-Specific Flags

```
cdrs_enabled: True                # Enable CDR generation
in_pool: False                    # Exclude from load balancing
pool
online_charging_enabled: False    # Enable OCS integration
recording: True                   # Enable call recording (AS)
populate_crm: False               # Populate CRM with initial data
```

Global Variables (all:vars)

The `all:vars` section contains deployment-wide settings. Modern deployments use minimal global variables with most configuration in role defaults.

Essential Global Variables

Authentication & Access

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

Alternative: Use SSH keys instead of passwords:

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

Customer Identity

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

PLMN Configuration

```
plmn_id:
  mcc: '001'               # Mobile Country Code (3 digits)
  mnc: '01'               # Mobile Network Code (2-3 digits)
  mnc_longform: '001'     # Zero-padded MNC (always 3 digits)

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

Purpose: Uniquely identifies your mobile network. Used for Diameter realm construction.

Network Names

```
network_name_short: Omni
network_name_long: Omnitouch
tac_list: [10100,100]            # Default TAC list (can override
per-MME)
```

Displayed: Network names shown on UE devices in Settings > Mobile Network.

DNS Configuration

```
netplan_DNS: False               # Use systemd-resolved instead of
netplan DNS
```

APT Repository Configuration

Automatic Defaults: When an `apt_cache_servers` group is defined with hosts:

- `use_apt_cache` automatically defaults to `True` (unless explicitly set to `False`)
- `apt_repo.apt_server` automatically defaults to the first cache server's IP

```
# Manual configuration (optional if apt_cache_servers group
exists)
use_apt_cache: True              # Use local APT cache vs direct
repo access

apt_repo:
  apt_server: "10.10.1.114"      # APT cache server or repo server
  # Credentials only needed when use_apt_cache: False
  # apt_repo_username: "omni"
  # apt_repo_password: "omni"

# Binary downloads and cache sync configuration
# Used for: (1) downloading binaries from /releases/ when
use_apt_cache: false
#           (2) cache server syncing from Omnitouch when
use_apt_cache: true
remote_apt_server: "apt.omnitouch.com.au"
remote_apt_user: "omni"
remote_apt_password: "omni"
```

See: APT Cache System

License Server

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

See: License Server

MME Settings

```
mme_dns: False                    # Enable MME DNS resolution
```

SAEGW Settings

```
mtu: 1400                         # Maximum Transmission Unit
```

IMS Settings

```
ims_dra_support: False            # Route IMS through DRA
enable_homer: False               # Enable Homer SIP capture
```

RAN Monitor Configuration

```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
```

Firewall Configuration

```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'
```

Roaming DNS Servers

```yaml
roaming_dns_servers:
  wildcard: ['10.0.99.1']
  # Carrier-specific DNS (PLMN-based)
  123456:                          # Example Carrier 1
    - '10.10.2.197'
  654321:                          # Example Carrier 2
    - '10.10.0.4'
```

Local Users (SSH Keys)

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

Hypervisor Configuration

Proxmox

```yaml
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

# Default Proxmox settings
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
```

Related Documentation

- IP Planning Standard - Network architecture and IP allocation guidelines
- Hosts File Configuration - How to structure hosts files
- Group Variables Configuration - When and how to use group_vars
- Netplan Configuration - Secondary IPs and multi-NIC setup
- Deployment Architecture - How components interact
- APT Cache System - Package management
- License Server - License configuration

Product Documentation

For detailed operational guides and advanced configuration:

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

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

Deployment Architecture Overview

Overview

This document provides a complete view of how Omnitouch Network Services' cellular network software is deployed using Ansible, showing how all components fit together to create a working 4G/5G network.

See the IP Planning Standard for detailed component placement, IP address assignment guidelines, and public IP handling.

Complete Deployment Example

0. Infrastructure Provisioning (Optional)

For Proxmox deployments, provision VMs/LXCs before configuration:

```
# Deploy VMs on Proxmox
ansible-playbook -i hosts/Customer/hosts.yml services/proxmox.yml

# Or deploy LXC containers (lab/test only)
ansible-playbook -i hosts/Customer/hosts.yml
services/proxmox_lxc.yml
```

See: Proxmox VM/LXC Deployment

1. Infrastructure Definition (Hosts File)

```yaml
# Define what to deploy and where
mme:
  hosts:
    customer-mme01:
      ansible_host: 10.10.1.15

hss:
  hosts:
    customer-hss01:
      ansible_host: 10.10.2.140

# ... all other components
```

See: Hosts File Configuration

2. Customization (group_vars)

The `group_vars` folder is where we can store any config overrides needed at a host, site or network level.

For example you'd have a folder with your OmniMessage SMSc config, the SIP trunks your TAS connects to would live here, all your Diameter Routing logic, etc, etc.

See: Group Variables Configuration

3. Package Distribution (APT Cache)

```yaml
# Configure where to get packages
apt_repo:
  apt_server: "10.254.10.223"  # Cache server IP or direct repo
server
use_apt_cache: false  # true = use local cache, false = direct
repo access
```

See: APT Cache System

4. License Configuration

```yaml
# Point components to license server
license_server_api_urls: ["https://10.10.2.150:8443/api"]
license_enforced: true
```

See: License Server

5. Execute Deployment

Individual components can be deployed by running `services/twag.yml` for example, but the `services/all.yml` will handle everything, and you can use `--limit=myhost` or `--limit=mmee,sgw`, etc, to limit the hosts we're working on.

```
# Deploy complete network
ansible-playbook -i hosts/customer/host_files/production.yml
services/all.yml

# Or deploy specific components
ansible-playbook -i hosts/customer/host_files/production.yml
services/epc.yml
ansible-playbook -i hosts/customer/host_files/production.yml
services/ims.yml
```

Related Documentation

- Introduction to Ansible Deployment - Getting started
- Hosts File Configuration - Defining infrastructure
- IP Planning Standard - **Network architecture and IP allocation**
- Group Variables Configuration - Customization
- APT Cache System - Package management
- License Server - License management

Product Documentation

For detailed information on configuring each component:

- **OmniCore** (4G/5G Packet Core): https://docs.omnitouch.com.au/docs/repos/OmniCore
  - OmniHSS, OmniSGW, OmniPGW, OmniUPF, OmniDRA, OmniTWAG
- **OmniCall** (Voice & Messaging): https://docs.omnitouch.com.au/docs/repos/OmniCall
  - OmniTAS, OmniCall CSCF, OmniMessage, OmniSS7, VisualVoicemail
- **OmniCharge/OmniCRM** (Billing): https://docs.omnitouch.com.au/docs/repos/OmniCharge
- **Main Documentation**: https://docs.omnitouch.com.au/

Group Variables Configuration

Overview

The `group_vars` directory is where you store custom configuration files that override default templates.

This is where your customer-specific configurations live - SIP trunks, Diameter routing rules, SMS routing logic, dialplans, and any other customizations where you don't want the default config - It lives in `group_vars`.

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

How It Works

Ansible roles have default configuration templates. To customize for a specific deployment, place your custom files in `group_vars` and reference them in your hosts file.

```
Role Default Template → group_vars Override (if specified) →
Deployed Config
```

Example 1: Custom Configuration Template (OmniMessage)

Some components accept custom Jinja2 configuration templates.

File Structure

```
hosts/Customer/
└── group_vars/
    └── smsc_controller.exs        # Your custom config template
```

Reference in Hosts File

```yaml
omnimessage:
  hosts:
    customer-smsc-controller01:
      ansible_host: 10.10.3.219
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      smsc_template_config: smsc_controller.exs   # Reference your
template filename in group_vars
```

What happens:

1. Ansible finds `smsc_template_config: smsc_controller.exs`
2. Looks in `hosts/Customer/group_vars/smsc_controller.exs`
3. Templates it with Jinja2 (can use `{{ inventory_hostname }}`, `{{ plmn_id.mcc }}`, etc.)
4. Deploys to `/etc/omnimessage/runtime.exs`
5. Restarts the service

Without `smsc_template_config`, the default template from the role is used.

Configuration details: See https://docs.omnitouch.com.au/docs/repos/OmniCall

---

Example 2: Configuration File Collections (OmniTAS Gateways & Dialplans)

Some components use directories of configuration files.

File Structure

```
hosts/Customer/
└── group_vars/
    ├── gateways_prod/              # SIP gateway configs
    │   ├── gateway_carrier1.xml
    │   ├── gateway_carrier2.xml
    │   └── gateway_emergency.xml
    ├── gateways_lab/               # Lab gateways
    │   └── gateway_test.xml
    └── dialplan/                   # Call routing rules
        ├── mo_dialplan.xml         # Mobile Originated (outgoing)
        ├── mt_dialplan.xml         # Mobile Terminated (incoming)
        └── emergency.xml
```

Reference in Hosts File

```yaml
applicationserver:
  hosts:
    customer-tas01:
      ansible_host: 10.10.3.60
      gateway: 10.10.3.1
      host_vm_network: "vmbr3"
      gateways_folder: "gateways_prod"   # Reference your gateway
folder to use on this host
```

What happens:

1. Ansible finds `gateways_folder: "gateways_prod"`

2. Copies all files from `hosts/Customer/group_vars/gateways_prod/` to `/etc/freeswitch/sip_profiles/`
3. Copies all files from `hosts/Customer/group_vars/dialplan/` to OmniTAS templates directory
4. Services load the configurations

Different environments: Use different folders per environment:

- `gateways_folder: "gateways_lab"`
- `gateways_folder: "gateways_prod"`
- `gateways_folder: "gateways_customer_specific"`

Configuration details: See https://docs.omnitouch.com.au/docs/repos/OmniCall

---

Example 3: Custom Configuration Template (OmniHSS)

The Home Subscriber Server accepts custom runtime configuration templates.

File Structure

```
hosts/Customer/
└── group_vars/
    └── hss_runtime.exs.j2        # Your custom HSS config
template
```

Reference in Hosts File

```
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   # Reference your
template filename in group_vars
```

What happens:

1. Ansible finds `hss_template_config: hss_runtime.exs.j2`
2. Looks in `hosts/Customer/group_vars/hss_runtime.exs.j2`
3. Templates it with Jinja2 (can use `{{ inventory_hostname }}`, `{{ plmn_id.mcc }}`, etc.)
4. Deploys to `/etc/omnihss/runtime.exs`
5. Restarts the service

Without `hss_template_config`, the default template from the role is used.

Configuration details: See https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Example 4: Custom Configuration Template (OmniMME)

The Mobility Management Entity accepts custom runtime configuration templates.

File Structure

```
hosts/Customer/
└── group_vars/
    └── mme_runtime.exs.j2        # Your custom MME config
template
```

Reference in Hosts File

```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   # Reference your
template filename in group_vars
```

What happens:

1. Ansible finds `mme_template_config: mme_runtime.exs.j2`
2. Looks in `hosts/Customer/group_vars/mme_runtime.exs.j2`
3. Templates it with Jinja2 (can use `{{ inventory_hostname }}`, `{{ plmn_id.mcc }}`, etc.)
4. Deploys to `/etc/omnimme/runtime.exs`
5. Restarts the service

Without `mme_template_config`, the default template from the role is used.

Configuration details: See https://docs.omnitouch.com.au/docs/repos/OmniCore

---

Real-World Directory Structure Example

```
hosts/Customer/
├── host_files/
│   └── production.yml          # Hosts file references group_vars
files
└── group_vars/
    ├── smsc_controller.exs     # OmniMessage custom template
    ├── smsc_smpp.exs           # OmniMessage SMPP custom template
    ├── tas_runtime.exs.j2      # TAS custom template
    ├── hss_runtime.exs.j2      # HSS custom template
    ├── mme_runtime.exs.j2      # MME custom template
    ├── dra_runtime.exs.j2      # DRA custom template
    ├── pgwc_runtime.exs.j2     # PGW custom template
    ├── dea_runtime.exs.j2      # DEA custom template
    ├── upf_config.yaml         # UPF configuration
    ├── crm_config.yaml         # CRM configuration
    ├── stp.j2                  # SS7 STP template
    ├── hlr.j2                  # SS7 HLR template
    ├── camel.j2                # SS7 CAMEL template
    ├── ipsmgw.j2               # IP-SM-GW template
    ├── omnicore_smsc_ims.yaml.j2  # SMSC IMS config
    ├── pytap.yaml              # TAP3 configuration
    ├── sip_profiles/           # SIP gateways (folder)
    │   └── gateway_otw.xml
    └── dialplan/               # Call routing rules (folder)
        ├── mo_dialplan.xml     # Mobile Originated
        ├── mt_dialplan.xml     # Mobile Terminated
        └── mo_emergency.xml    # Emergency routing
```

Common Parameters That

Reference group_vars

Parameter	Component	References
`smsc_template_config`	omnimessage	Jinja2 template file (e.g., `smsc_controller.exs`)
`smsc_smpp_template_config`	omnimessage_smpp	Jinja2 template file (e.g., `smsc_smpp.exs`)
`gateways_folder`	applicationserver	Folder name (e.g., `sip_profiles`)
Dialplans (automatic)	applicationserver	`dialplan/` folder of routing XMLs
`tas_template_config`	applicationserver	Jinja2 template file (e.g., `tas_runtime.exs.j2`)
`hss_template_config`	omnihss	Jinja2 template file (e.g., `hss_runtime.exs.j2`)
`mme_template_config`	omnimme	Jinja2 template file (e.g., `mme_runtime.exs.j2`)
`dra_template_config`	dra	Jinja2 template file (e.g., `dra_runtime.exs.j2`)
`pgwc_template_config`	pgwc	Jinja2 template file (e.g., `pgwc_runtime.exs.j2`)

Parameter	Component	References
`frr_template_config`	omniupf	Jinja2 template file (e.g., `frr.conf.j2`)
SS7 templates	ss7 (various roles)	Jinja2 template files (e.g., `stp.j2`, `hlr.j2`, `camel.j2`)
Config YAMLs	Various components	Direct config files (e.g., `upf_config.yaml`, `crm_config.yaml`)

Key Points

1. **group_vars holds customizations** - Overrides for default configurations
2. **Reference by name** - Use parameters like `smsc_template_config` or `gateways_folder`
3. **Templates support Jinja2** - Access any Ansible variable with `{{ variable_name }}`
4. **Folders deploy everything** - All files in referenced folders are copied
5. **Version control everything** - Commit all group_vars to Git

When to Use group_vars

✅ Use group_vars for:

- Custom component configuration templates
- SIP gateway definitions
- Call routing dialplans
- Diameter routing rules
- Customer-specific settings that override defaults

❌ Don't use group_vars for:

- Basic host configuration (IPs, hostnames) - Use hosts file
- One-off testing - Use host-specific vars in hosts file
- Temporary changes - Edit on target, commit to group_vars if permanent

---

Related Documentation

- Configuration Reference - All host parameters and what they do
- Hosts File Configuration - How to structure hosts files
- **OmniCall Configuration**: https://docs.omnitouch.com.au/docs/repos/OmniCall - What goes in the config files
- **OmniCore Configuration**: https://docs.omnitouch.com.au/docs/repos/OmniCore - Component configuration details

Utility Playbooks

Overview

This repository includes several utility playbooks for maintenance, monitoring, and operational tasks. These complement the main deployment playbooks with day-to-day management capabilities.

Health Check Utility

The Health Check utility generates an HTML report showing system health, service status, uptime, and version information across all OmniCore components.

Runs automatically as part of `services/all.yml` playbook.

Usage

Manual Run

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

Output

Report is generated at `/tmp/health_check_YYYY-MM-DD HH:MM:SS.html`

Open in any web browser to view.

Report Contents

The HTML report displays:

Host Information

- **Host name and IP address**
- **Network/Subnet** (from `host_vm_network` variable, or N/A if not set)
- **CPU** (vCPU count)
- **RAM** (total and free memory)
- **Disk** (root partition total and free space with percentage)
- **OS** (distribution and version)

Service Status

- **Service status** (active/inactive with color indicators)
- **Uptime**
- **Version/release information**

HSS Diameter Peers

- **Database connection status** (connected/disconnected)
- **Diameter peer connections** (IP, origin host, status)
- Fetched from HSS metrics endpoint (port 9568)

Other Common Utilities

Base System Configuration

Common Role (`services/common.yml`)

- Applies base system configuration to all hosts
- Sets up repositories, SSH keys, timezone, NTP
- Configures networking and system hardening
- Run this before deploying services

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

Setup Users (`services/setup_users.yml`)

- Creates and configures user accounts across all hosts
- Manages SSH keys and sudo privileges
- Ensures consistent user setup

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

Reboot (`services/reboot.yml`)

- Gracefully reboots all targeted hosts
- Waits for systems to come back online (5 minute timeout)
- Useful after kernel updates or configuration changes

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

Operational Utilities

IP Plan Generator (`util_playbooks/ip_plan_generator.yml`)

- Generates HTML report of IP address assignments
- Shows complete network topology from hosts file
- Useful for documentation and troubleshooting

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

HSS Backup (`util_playbooks/hss_backup.yml`)

- Backs up HSS database tables
- Copies MySQL dump to local Ansible machine
- Interactive prompts for backup path

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

Get Local Capture (`util_playbooks/getLocalCapture.yml`)

- Fetches the two most recent packet capture files from all hosts
- Retrieves pcap files from `/etc/localcapture/`
- Useful for debugging connectivity issues

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

Update MTU (`util_playbooks/updateMtu.yml`)

- Updates network interface MTU settings
- Applies changes via netplan
- Useful for jumbo frame configuration

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

Related Documentation

- Main README - Overview and getting started
- Introduction to Ansible Deployment - Running playbooks
- Hosts File Configuration - Configure your inventory
- Deployment Architecture - Complete system overview
- APT Cache System - Package management

Hosts File Configuration

Overview

The hosts file (also called inventory file) is the central configuration document that defines your entire cellular network deployment. It specifies:

- What network functions to deploy
- Where they run (IP addresses, network segments)
- How they're configured (service-specific parameters)
- Customer-specific settings (PLMN, credentials, features)

File Location

Hosts files are organized by customer and environment:

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

Example Hosts File Structure

Here's a simplified example showing the key sections:

```yaml
# EPC Components
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 Components
pcscf:
  hosts:
    customer-pcscf01:
      ansible_host: 10.10.4.165

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

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

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

Common Host Parameters

Network Configuration

Every host typically includes:

```
pcscf:
  hosts:
    customer-pcscf01:
      ansible_host: 10.10.1.15      # IP address for SSH access
      gateway: 10.10.1.1             # Default gateway
      host_vm_network: "vmbr1"       # name of NIC to use on
Hypervisor
```

Note: For guidance on IP address planning and network segmentation strategies, see the IP Planning Standard which outlines the recommended four-subnet architecture for OmniCore deployments.

Proxmox Users: The `host_vm_network` parameter specifies which bridge to use. See Proxmox VM/LXC Deployment for automated provisioning.

VM Resource Allocation

For services needing specific resources:

```
num_cpus: 4                     # CPU cores
memory_mb: 8192                 # RAM in megabytes
proxmoxLxcDiskSizeGb: 50      # Disk size in GB
```

Service-Specific Parameters

Each network function has its own parameters. Examples:

MME:

```yaml
mme_code: 1                      # MME identifier (1-255)
mme_gid: 1                       # MME Group ID
network_name_short: Customer     # Network name (shown on phones)
network_name_long: Customer Network
tac_list: [600, 601, 602]      # Tracking Area Codes
```

PGW:

```yaml
ip_pools:                        # IP pools for subscribers
  - '100.64.16.0/24'
  - '100.64.17.0/24'
combined_CP_UP: false            # Separate control/user plane
```

For detailed explanation of what each variable controls, see: Configuration Reference

Application Server:

```yaml
online_charging_enabled: true  # Enable OCS integration
tas_branch: "main"             # Software branch to deploy
gateways_folder: "gateways_prod"  # SIP gateway configuration
```

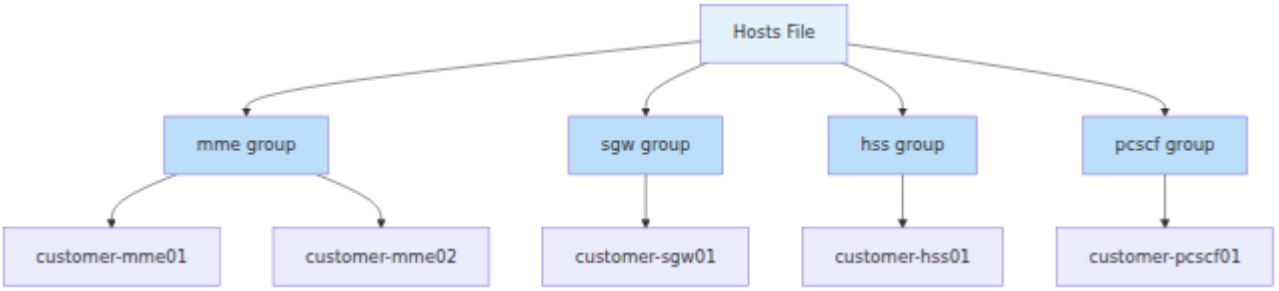
Global Variables Section

The `all:vars` section contains settings that apply to the entire deployment:

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

    # Customer Identity
    customer_name_short: customer
    customer_legal_name: "Customer Inc."
    site_name: "Chicago DC1"
    region: US

    # PLMN (Mobile Network) Identifier
    plmn_id:
      mcc: '001'              # Mobile Country Code
      mnc: '01'               # Mobile Network Code
      mnc_longform: '001'     # Zero-padded MNC

    # Network Names
    network_name_short: Customer
    network_name_long: Customer Network

    # APT Repository
    # Note: If apt_cache_servers group is defined with hosts,
    # use_apt_cache defaults to true and apt_repo.apt_server
    # defaults to the first cache server's IP automatically
    apt_repo:
      apt_server: "10.254.10.223"
      apt_repo_username: "customer"
      apt_repo_password: "secure-password"
    use_apt_cache: false

    # Timezone
    TZ: America/Chicago
```

Understanding Host Groups

Ansible organizes hosts into groups that correspond to roles:

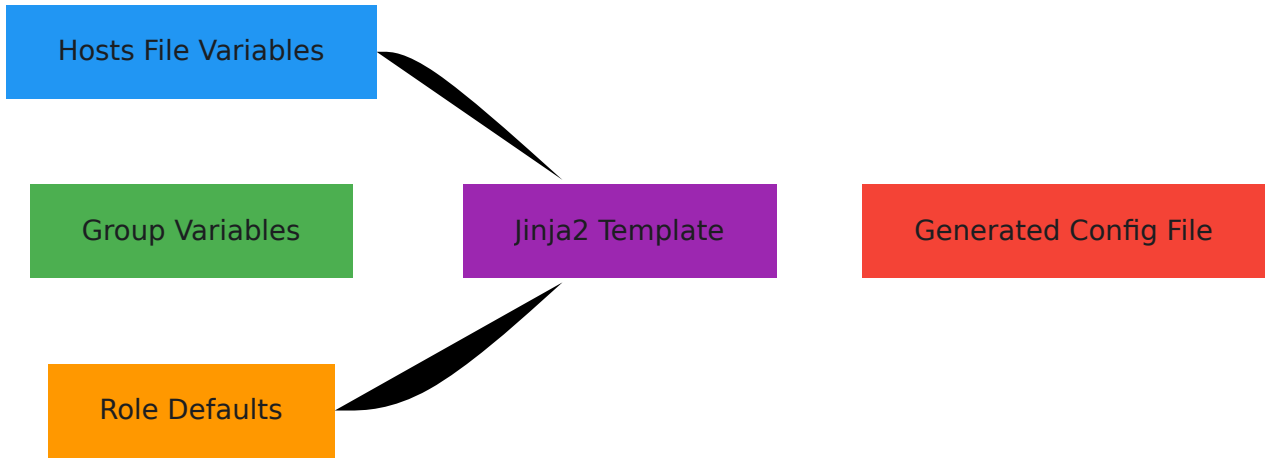


When you run a playbook targeting `mme`, it applies to all hosts in the `mme:hosts:` section.

Configuration with Jinja2 Templates

Ansible uses **Jinja2 templating** to generate configuration files from the variables defined in your hosts file and group_vars.

How Jinja2 Works



Example Template Usage

Hosts file defines:

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

Jinja2 template (in role):

```
# 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
```

Generated configuration file:

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

Common Jinja2 Patterns

Accessing nested variables:

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

Conditional logic:

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

Loops:

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

Formatting:

```
# Zero-pad to 3 digits
mnc{{ '%03d' | format(plmn_id.mnc|int) }}
```

Overriding Variables with group_vars

While the hosts file defines infrastructure and host-specific settings, `group_vars` can override defaults for groups of hosts.

See: Group Variables Configuration

Complete Example Hosts File

Here's a more complete example (with sensitive data obscured):

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

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

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

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

# IMS Core
pcscf:
```

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

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

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

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

# Support Services
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"

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

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

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

    # Charging Configuration
    charging:
```

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

    # Firewall Rules
    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'

    # Hypervisor Configuration (Proxmox example)
    proxmoxServers:
      customer-prxmx01:
        proxmoxServerAddress: 10.10.0.100
        proxmoxServerPort: 8006
        proxmoxApiTokenName: Customer
        proxmoxApiTokenSecret: "token-secret"
        proxmoxTemplateName: ubuntu-24.04-cloud-init-template
        proxmoxNodeName: pve01
```

See Proxmox VM/LXC Deployment for complete Proxmox setup and configuration details.

Product Documentation References

For detailed configuration of each component, refer to the official product documentation:

OmniCore Components:

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

- **OmniHSS** - Home Subscriber Server
- **OmniSGW** - Serving Gateway (Control plane)
- **OmniPGW** - Packet Gateway (Control plane)
- **OmniUPF** - User Plane Function
- **OmniDRA** - Diameter Routing Agent
- **OmniTWAG** - Trusted WLAN Access Gateway

OmniCall Components:

- **OmniCall Documentation**: https://docs.omnitouch.com.au/docs/repos/OmniCall
- **OmniTAS** - IMS Application Server (VoLTE/VoNR)
- **OmniCall CSCF** - Call Session Control Functions
- **OmniMessage** - SMS Center
- **OmniMessage SMPP** - SMPP Protocol Support
- **OmniSS7** - SS7 Signaling Stack
- **VisualVoicemail** - Voicemail

OmniCharge/OmniCRM:

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

Related Documentation

- Introduction to Ansible Deployment - Overall deployment process
- Configuration Reference - **Complete guide to all configuration variables**
- Group Variables Configuration - Overriding default configurations
- IP Planning Standard - **Network architecture and IP allocation guidelines**
- Netplan Configuration - **Secondary IPs and advanced network configuration**
- APT Cache System - Package distribution
- License Server - License management

- Deployment Architecture Overview - Complete system view

Next Steps

1. Create your hosts file based on this template
2. Define your PLMN and network identity
3. Configure APT repository access
4. Set up license server
5. Customize with group_vars as needed
6. Deploy with Ansible playbooks

OmniCore IP Planning Standard

Overview

This document outlines the standard IP planning approach for OmniCore deployments. The architecture requires **four internal subnets** to properly segment network functions for security, performance, and operational clarity.

IP Allocation Requirements

Standard Allocation: Four /24 Subnets

Each OmniCore deployment requires four distinct subnets for internal networking:

1. **Packet Core Network** - First /24
2. **Signaling Network** - Second /24
3. **IMS Internal Network** - Third /24
4. **UE Public Network** - Fourth /24

Important: These are Recommendations, Not Requirements

The subnet allocation described in this document is a **recommended best practice** for organizing OmniCore deployments. However, the architecture is **completely flexible**:

- **All hosts in one subnet**: You can place all components in a single subnet if that suits your deployment needs
- **Each host type in its own subnet**: You can create separate subnets for each component type (one for MMEs, one for HSS, etc.)

- **Custom groupings**: You can organize hosts into any subnet structure that makes sense for your specific requirements
- **Mix internal and public IPs**: Some hosts can use internal (RFC 1918) addresses while others use public IPs, all within the same deployment

The recommended four-subnet approach provides optimal **security isolation, traffic management, and operational clarity**, which is why we suggest it for production deployments. However, you should adapt the IP plan to fit your specific network topology, available address space, and operational requirements.

Network Segment Breakdown

1. Packet Core Network (First /24)

Purpose: User plane and core control plane elements

Components:

- OmniMME (Mobility Management Entity)
- OmniSGW (Serving Gateway)
- OmniPGW-C (PDN Gateway Control Plane)
- OmniUPF/PGW-U (User Plane Function / PDN Gateway User Plane)

Example: `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. Signaling Network (Second /24)

Purpose: Diameter signaling, policy, charging, and management functions

Components:

- OmniHSS (Home Subscriber Server)
- OmniCharge OCS (Online Charging System)
- OminiHSS PCRF (Policy and Charging Rules Function)
- OmniDRA DRA (Diameter Routing Agent)
- DNS Servers
- TAP3/CDR Servers
- Monitoring/OAM
- SIP capture
- License Server
- RAN Monitor
- Omnitouch Warning Link CBC (Cell Broadcast Center) - if deployed
- APT Cache Servers - if deployed

Example: `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 Internal Network (Third /24)

Purpose: IMS core signaling and services (internal SIP signaling)

Components:

- OmniCSCF S-CSCF (Serving Call Session Control Function)

- OmniCSCF I-CSCF (Interrogating Call Session Control Function)
- OmniTAS (Telephony Application Server / Application Server)
- OmniMessage (SMS Controller, SMPP, IMS)
- OmniSS7 STP (SS7 Signaling Transfer Point)
- OmniSS7 HLR (Home Location Register) - for 2G/3G
- OmniSS7 IP-SM-GW (MAP SMSc)
- OmniSS7 CAMEL Gateway

Example: `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 Public Network (Fourth /24)

Purpose: User-facing services such as IMS and DNS

Components:

- OmniCSCF P-CSCF (Proxy Call Session Control Function)
- XCAP Servers
- Visual Voicemail Servers
- Customer DNS

Example: `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"
```

Implementation Methods

OmniCore supports two primary methods for implementing this network segmentation:

Method 1: Physical/Virtual Network Interfaces (Recommended for Production)

Use separate NICs or virtual bridges for each network segment. This provides the strongest isolation and is the recommended approach for production deployments.

Example:

```yaml
# Packet Core - vmbr1
mme:
  hosts:
    omni-lab07-mme01:
      ansible_host: 10.179.1.15
      gateway: 10.179.1.1
      host_vm_network: "vmbr1"

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

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

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

Method 2: VLAN-Based Segmentation

Use a single physical interface with VLAN tagging to separate networks. This is suitable for smaller deployments or when physical NICs are limited.

Example:

```yaml
# All components use vmbr12 with different VLANs
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"
```

Network Configuration:

- Configure VLANs on the physical switch
- Tag traffic appropriately at the hypervisor level
- Route between VLANs at the gateway/firewall

Example VLAN Mapping:

```
VLAN 10: 10.x.1.0/24 (Packet Core)
VLAN 20: 10.x.2.0/24 (Signaling)
VLAN 30: 10.x.3.0/24 (IMS Internal)
VLAN 40: 10.x.4.0/24 (UE Public)
```

Working with Public IP Addresses

Overview

Many OmniCore deployments require some components to have public IP addresses for external connectivity, such as:

- **DRA** - For roaming diameter signaling with external carriers
- **Roaming SGW/PGW** - For GTP traffic from roaming partners
- **ePDG** - For WiFi calling (IPsec tunnels from UEs)
- **SMSC Gateway** - For SMPP connections to external SMS aggregators
- **P-CSCF** (in some deployments) - For direct UE SIP registration

How to Assign Public IPs

Public IPs are handled **exactly the same way as internal IPs** in your host inventory files. Simply specify the public IP address in the `ansible_host` field along with the appropriate gateway and netmask.

Example: Roaming SGW/PGW with Public IPs

```yaml
sgw:
  hosts:
    # Internal SGWs on private network
    opt-site-sgw01:
      ansible_host: 10.4.1.25
      gateway: 10.4.1.1
      host_vm_network: "v400-omni-packet-core"

    # Roaming SGWs with public IPs
    opt-site-roaming-sgw01:
      ansible_host: 203.0.113.10
      gateway: 203.0.113.9
      netmask: 255.255.255.248       # /29 subnet
      host_vm_network: "498-public-servers"
      in_pool: False
      cdrs_enabled: True

smf:  # PGWs
  hosts:
    # Roaming PGW with public IP
    opt-site-roaming-pgw01:
      ansible_host: 203.0.113.20
      gateway: 203.0.113.17
      netmask: 255.255.255.240       # /28 subnet
      host_vm_network: "497-public-services-LTE"
      in_pool: False
      ip_pools:
        - '100.64.24.0/22'
```

Example: DRA with Public IP

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

Example: ePDG with Public IP

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

Mixing Internal and Public IPs

It's common to have a mix of internal and public IPs within the same component group. For example:

- Internal SGWs for local sites using GTP
- Public SGWs specifically for roaming traffic from external carriers
- The same PGW-C can manage both internal and external SGWs

OmniCore's architecture handles this seamlessly - just configure each host with its appropriate IP addressing.

License Server

Overview

The License Server manages feature activation for all Omnitouch components. Each component validates its license on startup and periodically during operation.

Setup

1. Define in Hosts File

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

2. Provide License File

Place `license.json` (provided by Omnitouch) in `hosts/Customer/group_vars/`

3. Deploy

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

You can check the status of all license by browsing to https://license_server .

Network Requirements

Firewall Configuration

Client site firewalls must be configured to allow HTTPS (port 443) traffic to the following Omnitouch license validation servers:

Hostname	IP Address	Purpose
time.omnitouch.com.au	160.22.43.18	License validation server 1
time.omnitouch.com.au	160.22.43.66	License validation server 2
time.omnitouch.com.au	160.22.43.114	License validation server 3

Required outbound rules:

- Protocol: HTTPS (TCP/443)
- Destination: 160.22.43.18, 160.22.43.66, 160.22.43.114
- Direction: Outbound

DNS Requirements

The license server requires functional DNS resolution to communicate with the Omnitouch license validation infrastructure.

Required DNS configuration:

- The license server must have access to public DNS servers
- Configure DNS to use one of the following:
  - 1.1.1.1 (Cloudflare - supports secure DNS)
  - 8.8.8.8 (Google Public DNS)
- Do not use internal/corporate DNS servers for the license server

Note: The Omnitouch license servers use secure DNS (DoH/DoT). Using public DNS servers ensures proper DNSSEC validation and prevents issues with DNS interception by security appliances.

Related Documentation

- Configuration Reference
- Hosts File Configuration

Netplan Configuration

Overview

OmniCore can automatically configure network interfaces on deployed VMs using netplan. This is useful for:

- Setting up the primary management interface (eth0)
- Adding secondary interfaces for public IPs, peering connections, or dedicated traffic
- Configuring static routes for specific destinations

Enabling Netplan Configuration

To enable automatic netplan configuration for a host, add the `netplan_config` variable pointing to a Jinja2 template in your `group_vars` folder:

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

The template will be sourced from `hosts/<customer>/group_vars/netplan.yaml.j2`.

Template Reference

Here is the complete `netplan.yaml.j2` template with comments explaining each section:

```
network:
  version: 2
  ethernets:
    # Primary interface - uses ansible_host and gateway from
inventory
    eth0:
      addresses:
      - "{{ ansible_host }}/{{ mask_cidr | default(24) }}"
      nameservers:
        addresses:
{% if 'dns' in group_names %}
        # If this host IS a DNS server, use external DNS to avoid
circular dependency
        - 8.8.8.8
{% else %}
        # Otherwise, use DNS servers from the 'dns' group in
inventory
{% 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 %}
    # Secondary interfaces - loop through secondary_ips dict from
inventory
    # Interface naming: 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 %}
      # Static routes for this interface - each route uses this
interface's gateway
      routes:
{% for route in nic_config.routes %}
      - to: "{{ route }}"
        via: "{{ nic_config.gateway }}"
```

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

Key points:

- `ansible_host` and `gateway` come from the host's inventory entry
- DNS servers are dynamically pulled from hosts in the `dns` group
- Secondary interfaces are named `ens19`, `ens20`, etc. to match Proxmox NIC naming
- Each secondary IP can have its own gateway and static routes

Primary Interface Configuration

The primary interface (eth0) is configured automatically using:

- `ansible_host` - The IP address
- `gateway` - The default gateway
- `mask_cidr` - Network mask (defaults to 24)

DNS servers are automatically set to:

- Hosts in the `dns` group (uses their `ansible_host` IPs)
- Falls back to `8.8.8.8` if the host is itself a DNS server

Secondary Interfaces

For hosts requiring additional network interfaces (public IPs, peering, etc.), use the `secondary_ips` configuration.

Schema

```
secondary_ips:
  <logical_name>:
    ip_address: <ip_address>
    gateway: <gateway_ip>
    host_vm_network: <proxmox_bridge>
    vlanid: <vlan_id>
    routes:                    # Optional - static routes via this
interface
      - '<destination_cidr>'
      - '<destination_cidr>'
```

Interface Naming

Secondary interfaces are automatically named using Ubuntu's predictable naming scheme:

- First secondary interface: `ens19`
- Second secondary interface: `ens20`
- Third secondary interface: `ens21`
- And so on...

This matches the interface names assigned by Proxmox when adding additional NICs to a VM.

Example Configuration

```yaml
dra:
  hosts:
    <hostname>:
      ansible_host: 10.0.1.100
      gateway: 10.0.1.1
      host_vm_network: "ovsbr1"
      vlanid: "100"
      netplan_config: netplan.yaml.j2
      secondary_ips:
        public_ip:
          ip_address: 192.0.2.50
          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'
```

Generated Netplan Output

The above configuration generates:

```yaml
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 Integration

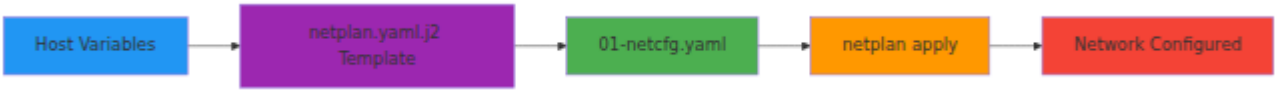
When using the `proxmox.yml` playbook, secondary NICs are automatically created on the VM:

1. **New VMs**: Secondary NICs are added during initial provisioning
2. **Existing VMs**: Secondary NICs are added and the VM is rebooted to apply changes

The Proxmox configuration uses:

- `host_vm_network` - The bridge to attach the NIC to
- `vlanid` - VLAN tag for the interface

How It Works



1. Variables from hosts file are passed to the Jinja2 template
2. Template renders to `/etc/netplan/01-netcfg.yaml`
3. Any existing netplan configs are removed to prevent conflicts
4. `netplan apply` activates the configuration
5. IP addresses are verified with `ip addr show`

Common Use Cases

Diameter Edge Agent (DEA) with Public IP

```
<hostname>:
  ansible_host: 10.0.1.100           # Internal management IP
  gateway: 10.0.1.1
  netplan_config: netplan.yaml.j2
  secondary_ips:
    diameter_roaming:
      ip_address: 192.0.2.50         # Public IP for roaming
partners
      gateway: 192.0.2.1
      host_vm_network: "vmbr0"
      vlanid: "200"
      routes:
        - '198.51.100.0/24'          # Roaming partner network
```

PGW with S5/S8 Interface

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

Multi-homed Server with Separate Management and Data Networks

```
<hostname>:
  ansible_host: 10.0.1.100           # Management network
  gateway: 10.0.1.1
  netplan_config: netplan.yaml.j2
  secondary_ips:
    data_network:
      ip_address: 10.0.2.100         # Data network
      gateway: 10.0.2.1
      host_vm_network: "ovsbr2"
      vlanid: "200"
    backup_network:
      ip_address: 10.0.3.100         # Backup network
      gateway: 10.0.3.1
      host_vm_network: "ovsbr3"
      vlanid: "300"
```

Referencing Secondary IPs in Templates

You can reference secondary IP addresses in other Jinja2 templates and configuration files.

On the Same Host

When configuring a service on the same host that has secondary IPs, you can reference directly or use `inventory_hostname`:

```
# Reference directly (simplest)
{{ secondary_ips.diameter_public_ip.ip_address }}

# Or explicitly via inventory_hostname (same result)
{{ hostvars[inventory_hostname]['secondary_ips']
['diameter_public_ip']['ip_address'] }}

# Access other properties
{{ secondary_ips.diameter_public_ip.gateway }}
{{ secondary_ips.diameter_public_ip.vlanid }}
```

From Another Host

When you need to reference a *different* host's secondary IP (e.g., configuring a peer connection), use `hostvars` with the target hostname:

```
# Reference first host in dra group
{{ hostvars[groups['dra'][0]]['secondary_ips']
['diameter_public_ip']['ip_address'] }}

# Loop through all DRA hosts and get their public IPs
{% for host in groups['dra'] %}
{% if hostvars[host]['secondary_ips'] is defined %}
  - {{ hostvars[host]['secondary_ips']['diameter_public_ip']
['ip_address'] }}
{% endif %}
{% endfor %}
```

Example: DRA Peer Configuration

Configure a diameter peer to bind to its own public IP:

```
# In dra_config.yaml.j2 - use inventory_hostname for the current
host
peers:
  - name: external_peer
    # Bind to this host's public diameter IP
    local_ip: {{ hostvars[inventory_hostname]['secondary_ips']
['diameter_public_ip']['ip_address'] }}
    remote_ip: 198.51.100.50
    port: 3868
```

Checking if Secondary IPs Exist

Always check if the variable exists before using it:

```
{% 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 %}
```

Troubleshooting

Verify Interface Names

SSH to the VM and check interface names:

```
ip link show
```

Expected output for a VM with two secondary interfaces:

```
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> ...
```

Check Netplan Configuration

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

Apply Netplan Manually

```
netplan apply
```

Debug Netplan

```
netplan --debug apply
```

Verify Routes

```
ip route show
```

Related Documentation

- Hosts File Configuration - Host inventory setup
- Proxmox VM/LXC Deployment - VM provisioning
- Configuration Reference - All configuration variables

Proxmox VM/LXC Deployment

The majority of our customers run the OmniCore stack on Proxmox, this guide explains in detail how to use the `proxmox` plays to setup their environment using Proxmox.

We still continue to support VMware, HyperV and cloud (Currently Vultr / AWS / GCP) for deployments.

See Also:

- Hosts File Configuration - Define VMs to deploy
- IP Planning Standard - IP address assignment guidelines
- Netplan Configuration - Secondary IPs and multi-NIC setup
- Deployment Architecture - Complete deployment workflow

LXC vs VM

LXC Containers:

- Lightweight, shares host kernel
- Fast startup, low overhead
- Limited isolation
- Cannot run custom kernels or kernel modules
- **Not suitable for production deployments**
- **Cannot run UPF** (requires kernel modules/TUN devices)

VMs (KVM):

- Full virtualization with dedicated kernel
- Complete isolation
- Can run kernel modules and custom networking
- Higher resource overhead

- **Recommended for production**
- **Required for UPF deployments**

Use Cases:

- **VMs**: Production sites, UPF, all network functions
- **LXC**: Lab/test environments, lightweight services (apt-cache, monitoring)

Proxmox Setup

1. Create API Token

```
# In Proxmox UI: Datacenter → Permissions → API Tokens
# Create token: root@pam!<TokenName>
# Copy the token secret (shown once)
```

2. Create Cloud-Init VM Template (for VMs only)

Run this script on the Proxmox host. It downloads Ubuntu's cloud image and creates a template with cloud-init user credentials.

```
#!/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 "=== Downloading Ubuntu cloud image ==="
cd /var/lib/vz/template/iso
wget -N "$IMAGE_URL"

echo "=== Cleaning up old template ==="
qm destroy $TEMPLATE_ID --purge 2>/dev/null || true

echo "=== Enabling snippets storage ==="
pvesm set local --content images,vztmpl,iso,backup,snippets

echo "=== Creating cloud-init user-data ==="
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 "=== Creating template VM ==="
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 "=== Template $TEMPLATE_ID created successfully ==="
```

Notes:

- Template provides a fallback login: `omnitouch` / `password` (for console access if cloud-init fails)
- When cloning via Ansible, credentials are overridden from `local_users` in your hosts file:
  - Username: First user's key from `local_users`
  - Password: First user's `password` field (defaults to 'password' if not set)
  - SSH key: First user's `public_key` field
- `--vga std` ensures the Proxmox web console works
- `-N` flag on wget only downloads if newer than local copy

Alternative: Manual Template from ISO

If cloud images aren't available or you need a custom install:

Step 1: Create VM via Web UI

- Create New VM → VM ID 9000, Name: ubuntu-2404-template
- OS: Upload Ubuntu Server ISO or use existing ISO
- System: Default (SCSI Controller: VirtIO SCSI)
- Disks: 32GB, Bus: SCSI
- CPU: 2 cores
- Memory: 2048 MB
- Network: VirtIO, bridge vmbr0
- Start VM and install Ubuntu Server

Step 2: Inside VM - Clean and prepare

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

# Clean machine-specific data
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

# Clear bash history and shutdown
history -c
sudo poweroff
```

Step 3: Add Cloud-Init and Convert to Template

- Select VM → Hardware → Add → CloudInit Drive (select storage e.g., local-lvm)
- Cloud-Init → User: `omnitouch`, Password: `password`
- Hardware → Options → QEMU Guest Agent → Enable
- Right-click VM → Convert to Template

3. Download LXC Template (for LXC only)

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

Hosts File Configuration

For VM Deployment (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  # optional
      pve-node-02:
        # ... second node config

    # User credentials - first user is used for VM cloud-init
    local_users:
      admin_user:
        name: Admin User
        public_key: "ssh-rsa AAAA..."
        password: "optional-password"  # defaults to 'password' if
not set

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

For LXC Deployment (proxmox_lxc.yml)

```yaml
all:
  vars:
    proxmoxServerAddress: 192.168.1.100
    proxmoxServerPort: 8006
    proxmoxNodeName: ['pve-node-01', 'pve-node-02']  # single or
list
    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

    # User credentials - first user is used for initial VM/LXC
access
    local_users:
      admin_user:
        name: Admin User
        public_key: "ssh-rsa AAAA..."
        password: "optional-password"  # defaults to 'password' if
not set

apt_cache_servers:
  hosts:
    site-cache:
      ansible_host: 192.168.1.20
      gateway: 192.168.1.1
      vlanid: "100"  # optional
      proxmoxLxcDiskSizeGb: 120  # per-host override
```

Usage

Deploy VMs

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

Deploy LXC Containers

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

Delete VMs/LXCs

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

Behavior

proxmox.yml

- Checks if VM with same name already exists in Proxmox
- Distributes VMs across nodes using round-robin
- Clones from template
- Configures static IP, tags, and cloud-init
- **Sets cloud-init user credentials from first `local_users` entry**
- Supports VLAN tagging

proxmox_lxc.yml

- Checks container doesn't exist by name or IP
- Distributes LXCs across nodes using round-robin

- Creates container with static IP
- **Automatically creates first `local_users` account with sudo access and SSH key**
- Configures netplan for networking
- Auto-starts containers
- Excludes UPF hosts

proxmox_delete.yml

- Stops and deletes VM/LXC matching inventory hostname
- Searches across all configured nodes
- Force stops after 20 seconds

VM/LXC Distribution & Tagging

Round-Robin Distribution

VMs and LXCs are automatically distributed across Proxmox nodes using round-robin (modulo) logic:

Example with 3 hypervisors and 5 MMEs:

```
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)
```

How it works:

1. Playbook identifies the host's role group (e.g., `mme`, `sgw`, `hss`)
2. Calculates host index within that group (0-based)
3. Uses modulo operation: `host_index % number_of_nodes`
4. Selects hypervisor based on result

Configuration:

```
# For VMs (proxmox.yml) - define multiple servers
proxmoxServers:
  pve-node-01: { ... }
  pve-node-02: { ... }
  pve-node-03: { ... }

# For LXCs (proxmox_lxc.yml) - list multiple nodes
proxmoxNodeName: ['pve-node-01', 'pve-node-02', 'pve-node-03']
```

Automatic Tagging

VMs and LXCs are automatically tagged with:

- **Role/Group names**: All Ansible groups the host belongs to
- **Site name**: The `site_name` variable

Example:

```
site_name: "melbourne-prod"

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

Result: VM/LXC tagged with: `mme`, `melbourne-prod`

Tags are visible in Proxmox UI and useful for filtering/organization.

Per-Host Overrides

Override defaults on specific hosts:

```yaml
hosts:
  high-spec-host:
    ansible_host: 192.168.1.50
    gateway: 192.168.1.1
    proxmoxLxcCores: 8        # override cores
    proxmoxLxcMemoryMb: 16384  # override memory
    proxmoxLxcDiskSizeGb: 100  # override disk
```

Utility Playbooks

Utility playbooks provide operational tools for managing deployed OmniCore infrastructure. These playbooks are located in the `util_playbooks/` directory and can be run independently to perform common maintenance and troubleshooting tasks.

Quick Reference

Playbook	Purpose
`health_check.yml`	Generate comprehensive health report for all services
`restore_hss.yml`	Restore HSS database and/or configuration from backup
`ip_plan_generator.yml`	Generate network documentation with Mermaid diagrams
`get_ports.yml`	Audit open ports and listening services across all hosts
`getLocalCapture.yml`	Retrieve packet capture files from hosts
`delete_local_user.yml`	Remove a local user account from all hosts
`updateMtu.yml`	Set MTU to 9000 (jumbo frames) on network interfaces
`systemctl status.yml`	Check service status on EPC components

Health Check

File: `util_playbooks/health_check.yml`

Generates a comprehensive HTML health report covering all deployed OmniCore and OmniCall services.

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

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

Information Collected

Component	Data Collected
All services	Service status, version, uptime
OmniHSS	Database status, Diameter peer connections
OmniDRA	Diameter peer connections and status
OmniTAS	Active calls, sessions, registrations, CPU usage
OCS	KeyDB replication status

HSS Restore

File: `util_playbooks/restore_hss.yml`

Restores OmniHSS from backup files. Supports restoring database only, configuration only, or both.

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

Backup File Formats

Type	Filename Pattern	Contents
Database	`hss_dump_<hostname>_<timestamp>.sql`	MySQL dump of `omnihss` database
Config	`hss_<hostname>_<timestamp>.tar.gz`	Archive of `/etc/omnihss` directory

IP Plan Generator

File: `util_playbooks/ip_plan_generator.yml`

Generates network documentation from inventory, including:

- Host IP assignments (primary and secondary NICs)
- Network segment overview
- Interface connectivity diagrams (Diameter, GTP, PFCP, SIP, SS7)

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

Output Files

File	Format	Description
`/tmp/ip_plan_<customer>_<site>.md`	Markdown	Text-based documentation
`/tmp/ip_plan_<customer>_<site>.html`	HTML	Interactive diagram with filterable layers

Port Audit

File: `util_playbooks/get_ports.yml`

Audits all listening ports across the deployment and generates documentation.

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

Output Files

File	Description
`/tmp/all_ports.csv`	CSV with hostname, IP, protocol, port, service
`./open_ports.rst`	reStructuredText table for Sphinx documentation

Data Collected

Field	Description
Hostname	Inventory hostname
IP	Host's `ansible_host` IP address
IP Version	IPv4 or IPv6
Transport	TCP or UDP
Port	Listening port number
Service	Process name

Local Capture Retrieval

File: `util_playbooks/getLocalCapture.yml`

Retrieves the two most recent packet capture files from each host's `/etc/localcapture` directory.

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

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

User Management

File: `util_playbooks/delete_local_user.yml`

Removes a local user account from all hosts in the inventory.

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

Prompt: Enter the username to delete when prompted.

MTU Configuration

File: `util_playbooks/updateMtu.yml`

Sets the MTU to 9000 (jumbo frames) on the `ens160` interface across all hosts.

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

Note: This playbook is hardcoded for `ens160` interface. Modify the playbook if your environment uses different interface names.

Running Utility Playbooks

Basic Syntax

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

Common Options

Option	Description
`-i <inventory>`	Specify inventory file
`--limit <hosts>`	Limit to specific hosts or groups
`-v` / `-vv` / `-vvv`	Increase verbosity
`--check`	Dry run (show what would change)
`--diff`	Show file differences

Examples

```
# Run health check on production
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/health_check.yml

# Restore HSS on a specific host
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/restore_hss.yml --limit hss01

# Generate IP plan with verbose output
ansible-playbook -i hosts/acme/host_files/production.yml
util_playbooks/ip_plan_generator.yml -v
```