Ucentral-Client and VYOS Setup as Gateway for APNOS

B17O739-79

Router Architects

TABLE OF CONTENTS

Date	Revision	Description	Modified by	Reviewed by
28/10/2025	.9	OLG POC POC Setup OLG Architecture Ucentral Image	Navneet Barwal	
13/11/2025	0.9.1	Updates for Topology	Navneet Barwal	
20/11/2025	1.0	Modification in VyOS Configuration Generation Logic MDU POC OLG Schema Modification for MDU	Navneet Barwal	
4/12/2025	1.1	Modification in OLG Configuration Architecture	Navneet Barwal	

POC OverView

It is a PoC to show the interoperability of OpenWiFi AP with OLG (Ucentral Client + VyOS) which should be configured by the Cloud Controller.

VyOS and Ucentral Client run in separate Docker Containers, Ucentral container handles configure commands from the cloud controller and convert these to VyOS style configuration and calls VyOS HTTP Server API's to push those configurations to VyOS container . The VyOS Container then provides connectivity to APNOS devices by configuring a DHCP server on its LAN. The host machine on which this is tested runs UBUNTU 24.04.1. The image generation for VyOS and Ucentral Client will be covered at last. The setup comes with a prerequisite to have both container images. The OpenWifi Version of Ucentral Client for OLG, and Cloud Controller is 3.1.

Objective

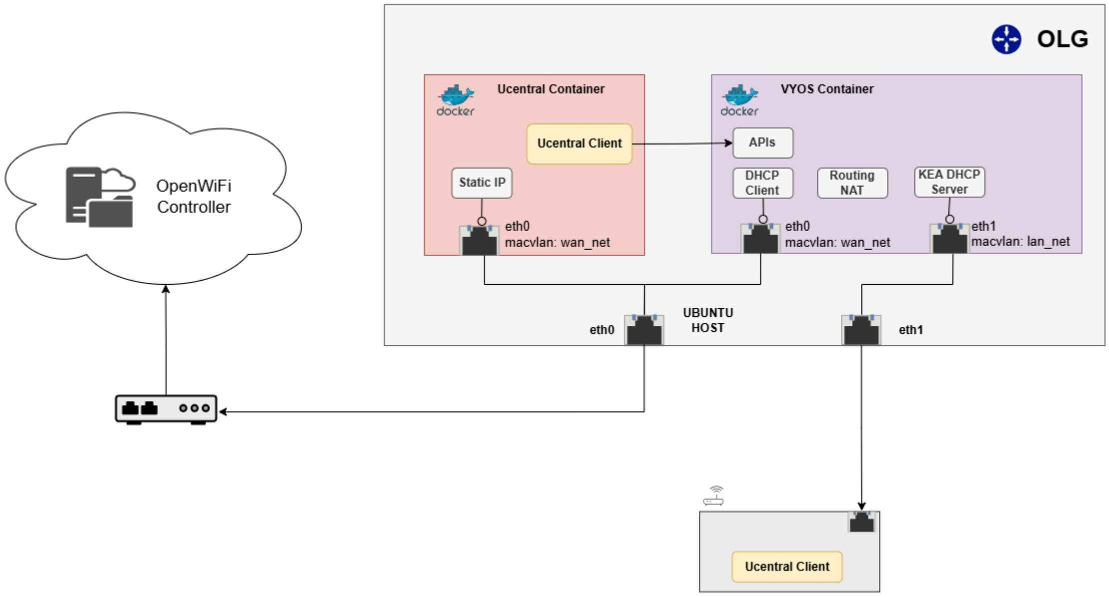
- Demonstrate how the OpenWiFi AP-NOS (via uCentral Client) can be managed from the cloud (uCentral Gateway) with a gateway device (VyOS) acting as the "edge" router.
- Validate that the gateway (VyOS) is properly configured (WAN/LAN separation) and that the AP-NOS devices can connect through to the cloud.
- Demonstrate zero-touch or near zero-touch provisioning of the AP via uCentral Client when pointed to the gateway and cloud.
- Confirm traffic flows: from AP → LAN net → VyOS LAN interface → VyOS WAN interface → Internet .

Prerequisites

- A Linux/Ubuntu host with **2 physical ethernet interfaces**:
    - `eth0` to WAN, configured via DHCP from upstream or manually.
    - `eth1` to LAN side, manual static IP (e.g. `192.168.50.2`).
- Docker installed; ability to use macvlan networks (requires appropriate kernel & privileges).
- VyOS container (image: `vyos-2025.09.10-0018-rolling-generic:olgV1`) — acts as the gateway/router.
- uCentral container (image: `ucentral-client:olgV1`) — acts as the configuration endpoint for the VyOS.
- The contents for host vyos_config volume which contains a postconfig script and config.boot to load default VyOS configurations.

Demo Network Topology

The diagram below illustrates the **OpenWiFi Local Gateway (OLG) Proof-of-Concept topology** for integrating **uCentral-Client** with **VyOS** as a gateway. The setup demonstrates how an APNOS connects through a local VyOS router toward the uCentral cloud, using Docker-based virtual networks on a single host system.



Components Overview

- **Host Machine :** The physical or virtual Linux system running Docker. It has two physical interfaces — `eth0` for the WAN side and `eth1` for the LAN side. These interfaces act as parents for the macvlan networks used by the containers.
- **uCentral Container :** It runs the uCentral Client process responsible for establishing a WebSocket connection to the uCentral cloud server. It uses a static IP in the `wan_net` subnet and connects to the cloud directly. It also translates json type uCentral configurations into VyOS text style configurations and also calls VyOS HTTPS Server API's to configure these configurations received from the cloud.
- **VyOS Container** : Acts as the **gateway router**. It provides Layer-3 routing and NAT between wan_net and lan_net. It has:
    - **eth0** in wan_net (DHCP client)
    - **eth1** in lan_net A **KEA DHCP Server** that assigns IP addresses to clients on the LAN subnet.
- **uCentral Client Device :** Represents the access point or OpenWiFi device running the uCentral client. It connects to the VyOS LAN (lan_net) and obtains an IP address

via DHCP from VyOS. From there, it reaches the cloud through NAT on the VyOS container.
- **Cloud Controller :** The remote endpoint (e.g., openwifi1.routerarchitects.com:15002) that manages device configuration, telemetry, and provisioning.

Network Structure

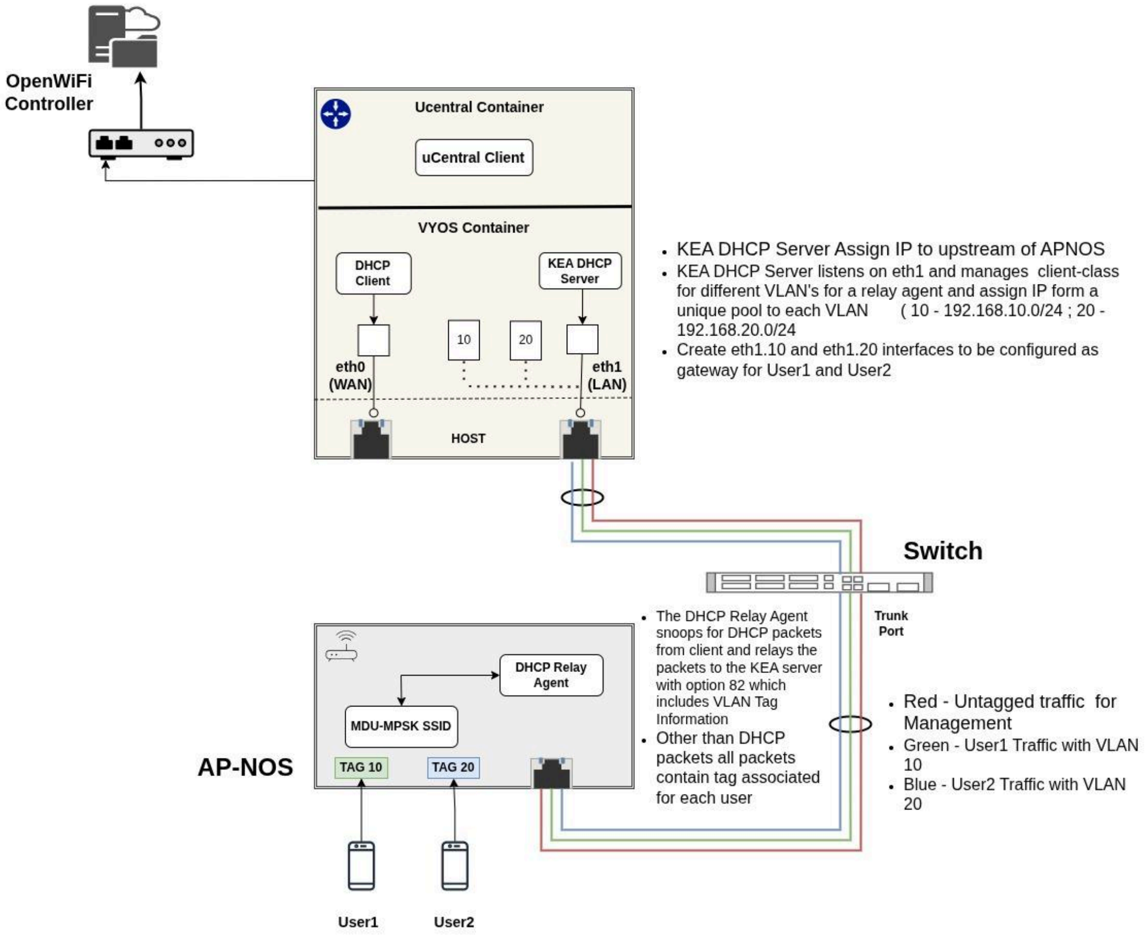
The system uses two isolated Docker macvlan networks to emulate real physical subnets:

- **wan_net :** Connects VyOS and uCentral containers to the WAN side (Internet/cloud).
- **lan_net :** Connects VyOS (LAN side) and APNOS.

Each container has its own macvlan child interface on the host's physical NIC, effectively isolating their traffic like separate physical devices.

MDU POC OverView

This is the next POC after POC Overview. In this POC topology we cover the scenario of a typical MDU setup where multiple tenants can connect to a single MDU SSID and their traffic remains isolated. This can be achieved with the help of MPSK and DHCP Relay and can be scaled with EMPSK using Radius. Its eth1 interface is configured as a trunk port carrying multiple VLANs (e.g. 10, 20, 100) toward the AP. Inside VyOS, virtual subinterfaces eth1.10 and eth1.20 serve as gateways for each user over a VLAN. On the AP (AP-NOS), the DHCP Relay Agent intercepts client DHCP messages from the MPSK SSIDs and forwards them to the Kea DHCP server in the VyOS container, embedding Option 82 (circuit ID) that indicates the VLAN tag. The Kea server uses client-classes to assign IP addresses from distinct pools (e.g. 192.168.10.0/24 for VLAN10, 192.168.20.0/24 for VLAN20).



- KEA DHCP Server Assign IP to upstream of APNOS
- KEA DHCP Server listens on eth1 and manages client-class for different VLAN's for a relay agent and assign IP form a unique pool to each VLAN (10 - 192.168.10.0/24 ; 20 - 192.168.20.0/24
- Create eth1.10 and eth1.20 interfaces to be configured as gateway for User1 and User2

- The DHCP Relay Agent snoops for DHCP packets from client and relays the packets to the KEA server with option 82 which includes VLAN Tag Information
- Other than DHCP packets all packets contain tag associated for each user

- Red - Untagged traffic for Management
- Green - User1 Traffic with VLAN 10
- Blue - User2 Traffic with VLAN 20

Objective

- Demonstrate how the OLG can be configured in a MDU Setup
- Validate that the gateway (VyOS) is properly configured and the clients connected to APNOS can get to the internet..
- Confirm traffic flows.

MDU POC Execution Guide

1. Modify olg_setup.sh to include new images with Version2 for
    - VyOS container (image: `vyos-2025.09.10-0018-rolling-generic:olgV2`) — acts as the gateway/router.
    - uCentral container (image: `ucentral-client:olgV2`) — acts as the configuration endpoint for the VyOS.
2. Remove the mounting for config volume now it is made part of VyOS container Image.
3. **Configure OLG from Cloud Controller**

```
{
  "interfaces": [
    {
      "ethernet": [
        {
          "select-ports": [
            "WAN*"
          ]
        }
      ],
      "ipv4": {
        "addressing": "dynamic"
      },
      "name": "WAN",
      "role": "upstream",
      "services": [
        "ssh",
        "dhcp-relay"
      ]
    },
    {
      "ethernet": [
        {
          "select-ports": [
            "LAN*"
          ]
        }
      ],
      "ipv4": {
        "addressing": "static",
        "dhcp": {
          "lease-count": 100,
          "lease-first": 10,
          "lease-time": "6h"
        },
        "subnet": "192.168.80.1/24"
      },
      "name": "LAN",
      "role": "downstream",
      "services": [
        "ssh"
      ],
      "vif": [
        {
```

```
                    "dhcp": {
                        "lease-count": 100,
                        "lease-first": 10,
                        "lease-time": "6h"
                    },
                    "subnet": "192.168.10.1/24",
                    "vlan": 10
                },
                {
                    "dhcp": {
                        "lease-count": 100,
                        "lease-first": 10,
                        "lease-time": "6h"
                    },
                    "subnet": "192.168.20.1/24",
                    "vlan": 20
                }
            ]
        }
    ],
    "services": {
        "ssh": {
            "port": 22
        }
    },
    "uuid": 1763625588
}
```

4. Configure APNOS from Cloud Controller

```
{
    "interfaces": [
        {
            "ethernet": [
                {
                    "select-ports": [
                        "WAN*"
                    ]
                }
            ],
            "ipv4": {
                "addressing": "dynamic"
            },
      "name": "WAN",
      "role": "upstream",
      "services": [
        "lldp",
        "ssh",
        "dhcp-relay"
      ]
    },
    {
      "ethernet": [
        {
          "select-ports": [
            "LAN*"
          ]
        }
      ],
      "ipv4": {
        "addressing": "static",
        "dhcp": {
          "lease-count": 128,
          "lease-first": 1,
          "lease-time": "6h"
        },
        "gateway": "192.168.1.1",
        "send-hostname": true,
        "subnet": "192.168.1.1/24"
      },
      "name": "LAN",
      "role": "downstream",
      "services": [
        "ssh",
        "lldp"
      ],
      "ssids": [
        {
          "bss-mode": "ap",
          "dtim-period": 2,
          "encryption": {
            "ieee80211w": "optional",
            "key": "NewUser@Key",
            "proto": "psk-mixed"
          },
          "fils-discovery-interval": 20,
            "hidden-ssid": false,
            "isolate-clients": false,
            "maximum-clients": 64,
            "multi-psk": [
               {
                  "key": "User1@Key",
                  "vlan-id": 10
               },
               {
                  "key": "User2@Key",
                  "vlan-id": 20
               }
            ],
            "name": "MDU-MPSK SSID",
            "roaming": {
               "generate-psk": true,
               "message-exchange": "ds"
            },
            "tip-information-element": true,
            "wifi-bands": [
               "5G",
               "2G"
            ]
         }
      ]
   }
],
"metrics": {
   "health": {
      "dhcp-local": true,
      "dhcp-remote": false,
      "dns-local": true,
      "dns-remote": true,
      "interval": 120
   },
   "statistics": {
      "interval": 120,
      "types": [
         "ssids",
         "clients",
         "lldp"
      ]
   }
```

```
},
"radios": [
  {
    "allow-dfs": true,
    "band": "2G",
    "bandwidth": 5,
    "beacon-interval": 100,
    "channel": "auto",
    "channel-mode": "HE",
    "channel-width": 40,
    "country": "US",
    "maximum-clients": 64,
    "tx-power": 0
  },
  {
    "allow-dfs": true,
    "band": "5G",
    "bandwidth": 5,
    "beacon-interval": 100,
    "channel": "auto",
    "channel-mode": "HE",
    "channel-width": 80,
    "country": "US",
    "maximum-clients": 64,
    "tx-power": 0
  }
],
"services": {
  "dhcp-relay": {
    "select-ports": [
      "WAN*"
    ],
    "vlans": [
      {
        "circuit-id-format": "vlan-id",
        "relay-server": "192.168.80.1",
        "remote-id-format": "ap-mac",
        "vlan": 10
      },
      {
        "circuit-id-format": "vlan-id",
        "relay-server": "192.168.80.1",
        "remote-id-format": "ap-mac",
          "vlan": 20
        }
      ]
    },
    "lldp": {
      "describe": "auto",
      "location": "auto"
    },
    "ssh": {
      "password-authentication": true,
      "port": 22
    }
  },
  "uuid": 1762508356
}
```

uCentral Schema Enhancement

In order to configure the VyOS for MDU specific configuration we have to enhance the schema of OLG which is currently derived from the OpenWifi Schema. Here are the details of the schema enhanced inside.

root -> interfaces -> interfaces items ->

```
{
 "vif": {
  "description": "List of VLAN subinterfaces (VIFs) attached to this logical interface.
Each entry describes a single VLAN ID with its own IPv4 subnet and optional DHCP
parameters.",
  "type": "array",
  "items": {
   "type": "object",
   "description": "Definition of a single VLAN subinterface (VIF) with its own subnet
and DHCP configuration.",
   "properties": {
    "vlan": {
        "description": "The VLAN ID used for this subinterface. Must be in the standard VLAN range.",
        "type": "integer",
        "minimum": 1,
        "maximum": 4094,
        "examples": [10, 20]
      },
      "subnet": {
        "description": "IPv4 subnet assigned to this VLAN interface, written in CIDR notation. The host address part will be used as the interface IP.",
        "type": "string",
        "examples": ["192.168.10.1/24", "192.168.20.1/24"]
      },
      "dhcp": {
        "description": "DHCP pool configuration for this VLAN interface.",
        "type": "object",
        "properties": {
          "lease-count": {
            "description": "Number of DHCP leases to offer in this subnet, starting from the offset given by lease-first.",
            "type": "integer",
            "minimum": 1,
            "examples": [100]
          },
          "lease-first": {
            "description": "Offset from the network base address for the first DHCP address. For example, 10 means start at network IP + 10.",
            "type": "integer",
       "minimum": 1,
       "examples": [10]
      },
      "lease-time": {
       "description": "How long the lease is valid before a RENEW must be issued.",
       "type": "string",
       "format": "uc-timeout",
       "default": "6h"
      }
     },
     "additionalProperties": false
    }
   },
   "required": ["vlan", "subnet"],
   "additionalProperties": false
  }
 }
}
```

Prerequisites

- A Linux/Ubuntu host with **2 physical ethernet interfaces**:
  - `eth0` to WAN, configured via DHCP from upstream or manually.
  - `eth1` to LAN side, manual static IP (e.g. `192.168.50.2`).
- Docker installed; ability to use macvlan networks (requires appropriate kernel & privileges).
- VyOS container (image: `vyos-2025.09.10-0018-rolling-generic:olgV2`) — acts as the gateway/router.

- uCentral container (image: `ucentral-client:olgV2`) — acts as the configuration endpoint for the VyOS.

OLG Architecture

The **Open Local Gateway (OLG)** functions as an **Edge Gateway** bridging the **uCentral Client** (northbound interface) with the **VyOS Router** (southbound interface).
It acts as an intermediary layer between the **Cloud Controller** (OpenWiFi backend) and **APNOS/OLS devices** deployed at the edge.

The OLG runs on a host system (for example, Ubuntu) and consists of two key containers:

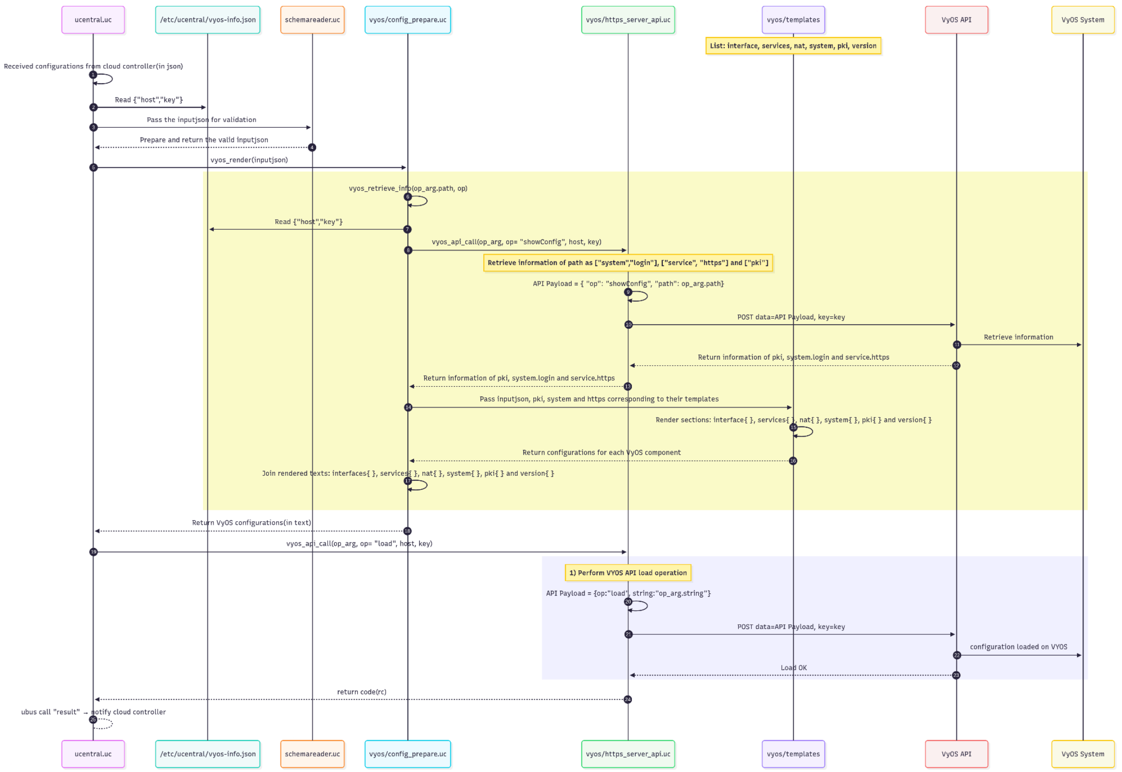
- **uCentral Client Container** – Handles communication with the OpenWiFi Cloud Controller, receives configuration updates, and reports status through the uCentral protocol.

- **VyOS Container** – Acts as a virtual gateway device responsible for routing, NAT, and DHCP functionality. It applies configurations dynamically as received from the cloud via uCentral.

This section focuses on the **interaction between uCentral Client and VyOS** containers within the OLG environment.

Communication Workflow Between uCentral and VyOS

The diagram below illustrates the step-by-step interaction flow that occurs when the **uCentral Client** receives configuration data from the **Cloud Controller**, processes it, and applies it to **VyOS** through its REST API. This is an upgrade to the previous logic where (load+merge) were used to configure VyoS for the uCentral configure command which disrupts the data path. The upgrade logic now implements (retrieve + load) and this helps in avoiding data path disruption.


ucentral.uc
/etc/ucentral/vyos-info.json
schemareader.uc
vyos/config_prepare.uc
vyos/https_server_api.uc
vyos/templates
VyOS API
VyOS System
List: interface, services, nat, system, pki, version
Received configurations from cloud controller(in json)
Read {"host","key"}
Pass the inputjson for validation
Prepare and return the valid inputjson
vyos_render(inputjson)
vyos_retrieve_info(op_arg.path, op)
Read {"host","key"}
vyos_api_call(op_arg, op= "showConfig", host, key)
Retrieve information of path as ["system","login"], ["service", "https"] and ["pki"]
API Payload = { "op": "showConfig", "path": op_arg.path}
POST data=API Payload, key=key
Retrieve information
Return information of pki, system.login and service.https
Return information of pki, system.login and service.https
Pass inputjson, pki, system and https corresponding to their templates
Render sections: interface{ }, services{ }, nat{ }, system{ }, pki{ } and version{ }
Return configurations for each VyOS component
Join rendered texts: interfaces{ }, services{ }, nat{ }, system{ }, pki{ } and version{ }
Return VyOS configurations(in text)
vyos_api_call(op_arg, op= "load", host, key)
1) Perform VYOS API load operation
API Payload = {op:"load", string:"op_arg.string"}
POST data=API Payload, key=key
configuration loaded on VYOS
Load OK
return code(rc)
ubus call "result" → notify cloud controller
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Step-by-Step Process

1. Configuration Reception (uCentral.uc)

- The uCentral Client receives configuration data (in JSON format) according to ucentral-schema from the Cloud Controller.

2. Reading Local VyOS Configuration Info

- The uCentral client reads `/etc/ucentral/vyos-info.json`. From this file it extracts
  - `host` – VyOS API endpoint (IP / hostname).
  - `key` – authentication/token for the VyOS HTTPS API.

These values (`host`, `key`) will be reused later for all API calls towards VyOS.

3. Validate the Incoming Configuration (schemareader.uc)

- uCentral sends the received "**inputjson**" to schemareader.uc for validation.
- schemareader.uc checks the JSON against the expected schema and normalises it if needed.
- It then returns a **validated** JSON back to uCentral.

4. **Start VyOS Render Pipeline (vyos/config_prepare.uc)**
   uCentral calls `vyos_render(inputjson)` implemented in `vyos/config_prepare.uc`. From this point, `vyos/config_prepare.uc` orchestrates fetching bits of current VyOS config (system.login, service.https, PKI), and rendering a full VyOS configuration using templates.
   - Retrieve Existing Config from VyOS (vyos/config_prepare.uc ↔ vyos/https_server_api.uc ↔ VyOS HTTPS Server API)
   - `vyos_retrieve_info()` prepares an operation argument `op_arg` describing what it wants to read (paths like `["system","login"]`, `["service","https"]`, `["pki"]`), and calls `vyos_api_call(op_arg, op = "showConfig", host, key).`
   - `vyos_api_call()` is implemented in `vyos/https_server_api.uc`. It builds the HTTP API payload and calls the VYOS HTTPS Server API's asking it to **retrieve configuration** for the requested paths.
   - VyOS returns the requested configuration fragments (specifically `pki`, `system.login` and `service.https`) back through the VyOS API to `vyos/https_server_api.uc`.
   - `vyos/https_server_api.uc` gives these configuration fragments back to `vyos/config_prepare.uc`.
     At this point `vyos/config_prepare.uc` has:
     - the validated `inputjson` from the cloud,
     - and the current VyOS `pki`, `system.login` and `service.https` blocks.

5. **Render VyOS Sections Using Templates (vyos/templates)**

   `vyos/config_prepare.uc` now calls into `vyos/templates`, passing:

   - `inputjson`
   - the retrieved `pki` configuration
   - the retrieved `system` (specifically `system.login`)
   - the retrieved `service.https` configuration

   `vyos/templates` contains multiple templates and each template renders its corresponding VyOS section

   - `interface <-> interfaces { … }`
   - `services <-> service { … }`
   - `nat <-> nat { … }`
   - `system <-> system { … }`
   - `pki <-> pki { … }`

`vyos/templates` returns the rendered text for each individual section back to `vyos/config_prepare.uc`.

NOTE: The version.uc is temporary until we figure out migration logic and get more info around thgis.

6. **Assemble Final VyOS Configuration Text (vyos/config_prepare.uc)**

   `vyos/config_prepare.uc` joins the individual rendered snippets into one combined VyOS configuration string. This final VyOS configuration text is then returned from `vyos_render()` to the uCentral logic running in `ucentral.uc`.

7. **Call VyOS API "load" Operation (vyos/https_server_api.uc)**

   With the full VyOS configuration text in hand, uCentral prepares a second API call:
   `vyos_api_call(op_arg, op = "load", host, key)`
   where op_arg.string carries the assembled VyOS configuration text.

   Inside `vyos/https_server_api.uc`, this is translated into the payload:

   `{ "op": "load", "string": "op_arg.string" }`
   `vyos/https_server_api.uc` sends this payload to the VyOS API. The VyOS API pushes this configuration to the VyOS system, which **loads** the configuration. VyOS returns a status (e.g. **"Load OK"** or an error) back via the VyOS API to `vyos/https_server_api.uc`.

   `vyos/https_server_api.uc` converts that into a return code `rc` and sends it back to uCentral.

8. **Result Handling and Notification**

   - uCentral receives the return code `rc` from the load operation.
   - It then issues a `ubus` call named `"result"` to report the outcome back to the Cloud Controller.
     - On success: informs that the VyOS configuration has been applied.
     - On failure: sends appropriate error details / code.
   - This closes the feedback loop between the cloud, OLG, and VyOS gateway.

POC Setup

Step 1: Setup Host, VyOS and Ucentral-Client Containers

This is a script to setup Host and Containers for OLG (olg_setup.sh)

```
#!/usr/bin/env bash


# === CONFIG VARIABLES ===
# Network & IP settings
WAN_NET_SUBNET="192.168.76.0/24"
WAN_NET_GATEWAY="192.168.76.1"
LAN_NET_SUBNET="192.168.50.0/24"
LAN_NET_GATEWAY="192.168.50.254"

UCENTRAL_IP="192.168.76.31"

# Volume / paths
VYOS_CONFIG_VOLUME="./vyos/vyos_config/"
MODULES_VOLUME="/lib/modules"

# Image names
VYOS_IMAGE="routerarchitect123/vyos-2025.09.10-0018-rolling-generic:olgV1"
UCENTRAL_IMAGE="routerarchitect123/ucentral-client:olgV1"

# === END CONFIG VARIABLES ===

# --- Clean up previous run ---
docker stop vyos-olg     2>/dev/null || true
docker rm   vyos-olg     2>/dev/null || true
docker stop ucentral-olg 2>/dev/null || true
docker rm   ucentral-olg 2>/dev/null
docker network rm wan_net 2>/dev/null || true
docker network rm lan_net 2>/dev/null || true


# --- Create networks ---
docker network create -d macvlan \
  --subnet=${WAN_NET_SUBNET} \
  --gateway=${WAN_NET_GATEWAY} \
  --subnet=fd00:aaaa:bbbb::/64 \
  --ipv6 \
  --gateway=fd00:aaaa:bbbb::1 \
  -o parent=eth0 \
  wan_net


docker network create -d macvlan \
  --subnet=${LAN_NET_SUBNET} \
  --gateway=${LAN_NET_GATEWAY} \
  --subnet=fd00:cccc:dddd::/64 \
  --gateway=fd00:cccc:dddd::1 \
  --ipv6 \
  -o parent=eth1 \
  --aux-address="vyos=192.168.50.1" \
  lan_net

# --- Run VyOS container ---
docker run -d --name vyos-olg --privileged \
  --network wan_net \
  -v ${MODULES_VOLUME}:/lib/modules \
  -v ${VYOS_CONFIG_VOLUME}:/opt/vyatta/etc/config \
  ${VYOS_IMAGE} /sbin/init

docker network connect lan_net vyos-olg

# --- Run UCentral container ---
docker run -dit --name ucentral-olg --privileged \
  --network wan_net --ip ${UCENTRAL_IP} \
  ${UCENTRAL_IMAGE}

echo "Setup completed successfully."
```

What this script does

- Removes any previous containers/networks to start clean.
- Creates two macvlan docker networks: `wan_net` (for WAN side) and `lan_net` (for LAN side).
- Runs a VyOS container attached first to `wan_net`, then connects to `lan_net` (so the container has two interfaces: one in WAN, one in LAN).
- Runs a uCentral container on the WAN network with a static IP.
- Leaves you with a VyOS container having both WAN & LAN connectivity; uCentral container on WAN side.
- Note: The variables are provided to be configured as per your requirement.

Step 2: Wait for the VyOS Container to become Functional

- Get Access to the VyOS container , it may take over a minute.

  ```
  docker exec -it vyos-olg su - vyos
  ```

- Inside the VyOS container: wait until eth0 (WAN) gets an IP from DHCP Server and have the default route set on eth0 . Note the IP received by eth0 , we need to provide this IP to Ucentral Configuration for VyOS.

Step 3: Run the Ucentral Container

- Get Access to the Ucentral Container.

  ```
  docker exec -it ucentral-olg /bin/ash
  ```

- Modify /etc/ucentral/vyos-info.json file with IP Address of the VyOS container noted earlier.
- Start Ubus , execute the following commands

  ```
  /sbin/ubusd  &
  ```

- This is a preconfigured image with certificates and a serial number for demo. Hence the serial number and cloud controller URL remains static . Now start the Ucentral process in the foreground.

  ```
  /usr/sbin/ucentral -S 74d4ddb965dc -s openwifi1.routerarchitects.com -P 15002 -d
  ```

Step 4: Verify Default Configurations

Inside the VyOS Container check eth1 interface is configured with IP provided for LAN from Ucentral and DHCP Server and this DHCP server provides connectivity to the APNOS Client connected to the eth1 interface.

VyOS Command to check DHCP lease

```
show dhcp server leases state all
```

You can ssh onto this device and also check its internet connectivity and can see in the list of added devices in the cloud.

Step 5: Modify Configurations from Cloud

Now the cloud controller modifies the configuration of Gateway Ucentral(74d4ddb965dc) for changing the network of LAN and it should reflect successfully on the device.

You can also monitor logs on VyOS through command

```
monitor log
```

Container Image Generation for VYOS

- Download VYOS Rolling Image from https://vyos.net/get/nightly-builds/

Steps to Build VyOS Docker image

```
mkdir vyos && cd vyos

mkdir rootfs

sudo mount -o loop vyos-2025.09.10-0018-rolling-generic-amd64.iso rootfs

sudo apt-get install -y squashfs-tools

mkdir unsquashfs

sudo unsquashfs -f -d unsquashfs/ rootfs/live/filesystem.squashfs

sudo tar -C unsquashfs -c . | docker import - vyos-2025.09.10-0018-rolling-generic

sudo umount rootfs

cd ..

sudo rm -rf vyos
```

Container Image Generation for Ucentral Client

Build Image RootFs and Prepare paths and a clean build context

```
mkdir WORKSPACE
cd WORKSPACE
mkdir OPENWIFI_WLANAP
cd OPENWIFI_WLANAP
git clone https://github.com/routerarchitects/ra-openwifi-wlan-ap.git
git checkout  release/v3.1.0
./build.sh x64_vm
```

After complete build check

```
ls openwrt/build_dir/target-x86_64_musl/root-x86/
```

Then Set Paths to copy contents for ucentral image

```
ROOT=~/WORKSPACE/OPENWIFI_WLANAP/ra-openwifi-wlan-ap/openwrt/build_
dir/target-x86_64_musl/root-x86
DEST=~/ucentral-init
rm -rf "$DEST"
mkdir -p
"$DEST"/{bin,sbin,usr/bin,usr/sbin,usr/share,lib,usr/lib,etc/ucentral,lib/config,lib/
functions}
```

Copy the Files Required to prepare image for Ucentral Client for OLG

```
cp -a "$ROOT"/usr/sbin/ucentral     "$DEST"/usr/sbin/
cp -a "$ROOT"/etc/group    "$DEST"/etc/
cp -a "$ROOT"/etc/passwd  "$DEST"/etc/
cp -a "$ROOT"/sbin/ubusd   "$DEST"/sbin/
cp -a "$ROOT"/bin/ubus   "$DEST"/bin/
cp -a "$ROOT"/usr/bin/curl  "$DEST"/usr/bin/
```

Add BusyBox and applet symlinks (so sh, ps, ls, … work)

```
cp -a "$ROOT"/bin/busybox "$DEST"/bin/
( cd "$DEST/bin" && ln -sf busybox sh && ln -sf busybox ash )
for a in ls ps ifconfig ping ip cat grep cut mkdir rm cp mv ln touch \
     mount umount basename readlink vi date uname echo sleep dmesg logger
flock; do
  ln -sf busybox "$DEST/bin/$a"
done
# if BusyBox lacked an applet, copy real ones if present:
[ -f "$ROOT/usr/bin/logger" ] && cp -a "$ROOT/usr/bin/logger" "$DEST/usr/bin/"
[ -f "$ROOT/usr/bin/flock"  ] && cp -a "$ROOT/usr/bin/flock"  "$DEST/usr/bin/"
```

Copy Ucentral’s ucode scripts

```
cp -a "$ROOT"/usr/share/ucentral   "$DEST"/usr/share/       2>/dev/null || true
cp -a "$ROOT"/usr/bin/ucode       "$DEST"/usr/bin/        2>/dev/null || true
cp -a "$ROOT"/usr/lib/ucode       "$DEST"/usr/lib/        2>/dev/null || true
```

Copy All Libraries

```
cp -a "$ROOT"/lib/*   "$DEST"/lib/
cp -a "$ROOT"/usr/lib "$DEST"/usr/
```

Make Executables Runnable

```
chmod +x "$DEST"/bin/busybox \
      "$DEST"/usr/sbin/ucentral \
      "$DEST"/usr/bin/ucode 2>/dev/null || true
```

Copy necessary files required to make ucentral functional with cloud controller and VyOS

```
cp cas.pem "$DEST"/etc/ucentral/
cp cert.pem "$DEST"/etc/ucentral/
cp key.pem "$DEST"/etc/ucentral/
cp capabilities.json "$DEST"/etc/ucentral/
cp vyos-info.json "$DEST"/etc/ucentral/
cp ucentral.cfg.0000000001 "$DEST"/etc/ucentral/
cp vyos_config_gen.uc "$DEST"/usr/share/ucentral/
cp  vyos_api_caller.uc "$DEST"/usr/share/ucentral/
cp  ucentral.uc "$DEST"/usr/share/ucentral/
```

Build Container Image for Ucentral

```
cat > "$DEST/Dockerfile" <<'EOF'
FROM scratch
COPY bin/  /bin/
COPY sbin/ /sbin/
COPY usr/  /usr/
COPY lib/  /lib/
COPY etc/  /etc/
RUN mkdir -p /var/run/
RUN mkdir -p /var/lock/
RUN mkdir -p /tmp
```

```
ENV PATH=/bin:/sbin:/usr/bin:/usr/sbin
ENV LD_LIBRARY_PATH=/lib:/usr/lib
WORKDIR /
CMD ["/bin/busybox","sleep","infinity"]
EOF

cd "$DEST"
docker build -t ucentral:latest .
```