Hewlett Packard Enterprise

Central / Configuration via API

14-Nov-25

Central Configuration via API

An Application Programming Interface (API) defines how one system, such as a client application, communicates with another, such as a server.

HPE Aruba Networking Central is built with an API-first architecture, offering comprehensive API coverage for all configuration and operational tasks.

Powered by the HPE GreenLake platform and based on a microservices architecture, Central provides a unified management solution for network devices across campus, branch, remote, and data center environments.

▼ Table of contents

Feedback

API-First Architecture

By exposing configuration and operational controls programmatically, APIs enable network teams to align infrastructure management with modern DevOps practices. They replace manual, error-prone processes with automated workflows that scale efficiently across large environments and maintain configuration integrity through continuous validation and compliance checks.

The following are key aspects of Central's API implementation:

- APIs are not an afterthought; they are first-class, versioned, well-documented, and thoroughly tested components of the platform.
- All platform functionality—including configuration, telemetry, assignments, and policy—is fully exposed through Representational State Transfer (REST) APIs.
- Developers, automation platforms, orchestration tools, and third-party systems can program and manage networks directly through the API.

This API-driven model transforms the network into a software-defined, future-ready platform.

REST APIs

Central's configuration APIs are based on the Representational State Transfer (REST) architecture. They use standard HTTP methods (POST, GET, PUT, DELETE) to perform operations on network resources. The APIs provide the most direct and flexible way to interact with Central, allowing you to build custom scripts and applications.

HTTP Verb	Function	Example in Central
`GET`	Retrieve objects	Get all WLAN configuration profiles
`POST`	Create objects	Create a new VLAN
`PUT/PATCH`	Update objects	Change WLAN configuration profile passphrase
`DELETE`	Remove objects	Delete ACL

API Credentials

Interacting with the API requires authorization to protect organizational data and prevent unauthorized changes. Access tokens included in the HTTP authorization header verify that an API request is allowed.

Access tokens are generated using HPE GreenLake API client credentials. The Developer Hub provides instructions for creating API client credentials and generating access tokens. For additional security, access tokens are ephemeral with a two-hour default lifetime.

Base URL

Each Central cluster has a unique base URL for making API requests. The Developer Hub maintains a list of Base URLs.

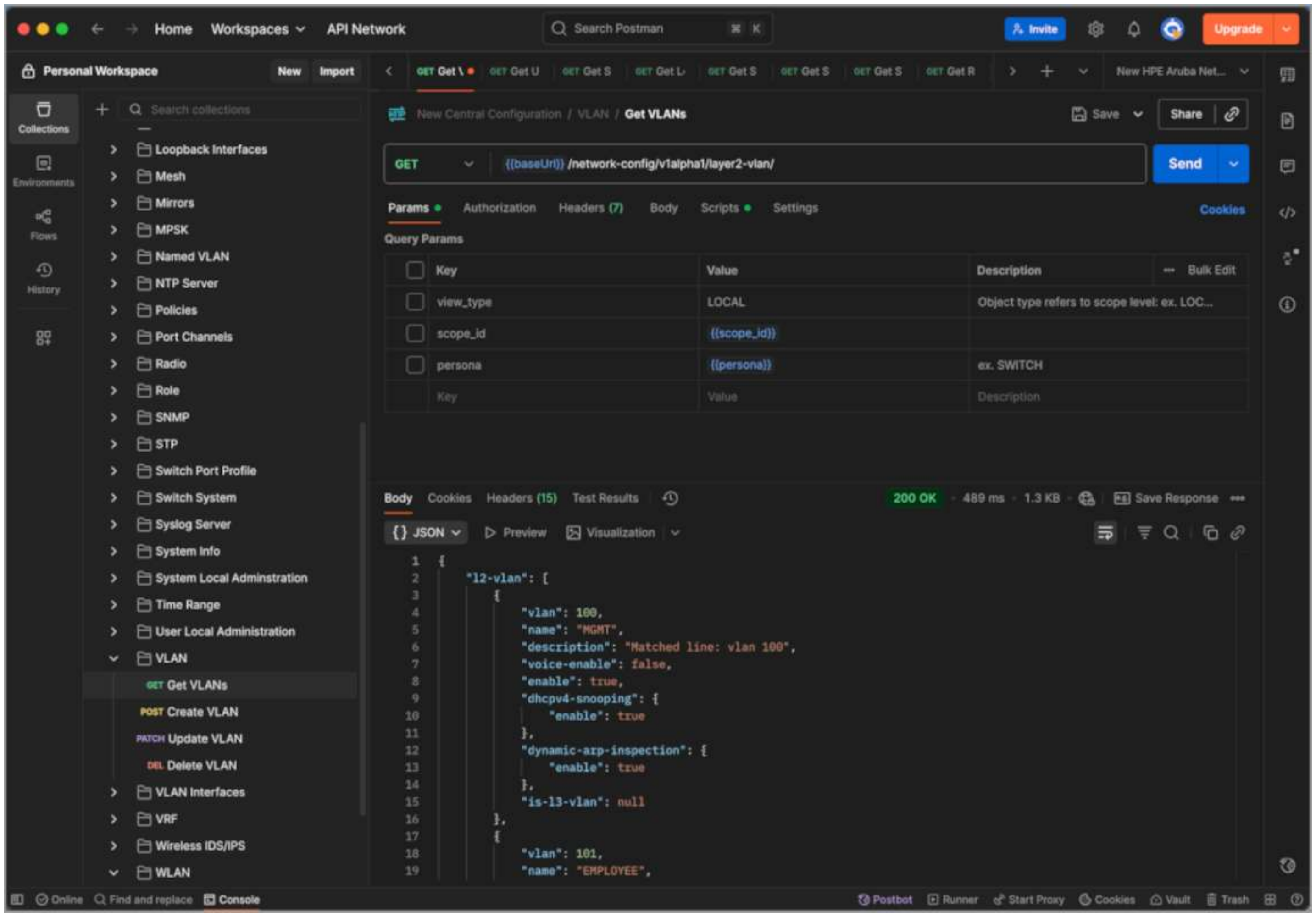
Resources for Automation Development

The HPE Aruba Networking Developer Hub offers comprehensive resources to help customers get started with automation and development. Tools are provided for Postman and the Python scripting language to interact with Central's APIs.

Note: The HPE Aruba Networking Developer Hub also includes automation tools for Classic Central and the latest version of Central. This guide focuses on the latest Central release, which is referred to as *New Central* in Developer Hub documentation and repositories.

Postman Collections

One of the most widely used tools in API development and testing, **Postman** simplifies how developers and testers interact with APIs using a full-featured GUI interface.



Loopback Interfaces
Mesh
Mirrors
MPSK
Named VLAN
NTP Server
Policies
Port Channels
Radio
Role
SNMP
STP
Switch Port Profile
Switch System
Syslog Server
System Info
System Local Adminstration
Time Range
User Local Administration
VLAN
GET Get VLANs
POST Create VLAN
PATCH Update VLAN
DEL Delete VLAN
VLAN Interfaces
VRF
Wireless IDS/IPS
WLAN

New Central Configuration / VLAN / **Get VLANs**

GET {{baseUrl}} /network-config/v1alpha1/layer2-vlan/

Params · Authorization · Headers (7) · Body · Scripts · Settings

Query Params

Key	Value	Description
view_type	LOCAL	Object type refers to scope level: ex. LOC...
scope_id	{{scope_id}}	
persona	{{persona}}	ex. SWITCH

Body · Cookies · Headers (15) · Test Results — 200 OK · 489 ms · 1.3 KB

```
{
    "l2-vlan": [
        {
            "vlan": 100,
            "name": "MGMT",
            "description": "Matched line: vlan 100",
            "voice-enable": false,
            "enable": true,
            "dhcpv4-snooping": {
                "enable": true
            },
            "dynamic-arp-inspection": {
                "enable": true
            },
            "is-l3-vlan": null
        },
        {
            "vlan": 101,
            "name": "EMPLOYEE",
```

The Developer Hub provides pre-built Postman collections to facilitate Postman-based automation for Central. These collections contain pre-configured requests for common tasks that can be used to explore the APIs, test workflows, and prototype automation scripts quickly.

Benefits of Using Postman

- **API testing without writing code**: HTTP requests can be sent with just a few clicks, instead of writing scripts or crafting complex Curl commands.
- **Quick Debugging**: Postman shows request headers, parameters, body, and the server's response (including status codes and error messages), which makes debugging faster.
- **Automation and Collections**: Developers can save multiple API requests in collections, organize them by project, and automate test runs using environments with pre/post-scripts.
- **Team Collaboration**: Developers, testers, and product teams can share collections, environment, and documentation. This improves communication about how an API works.
- **Mock Servers**: Developers can create mock endpoints to simulate API responses before the back end is fully implemented. This helps front-end developers work in parallel.
- **Context Switching**: Postman allows switching between development, staging, and production environments easily by storing base URLs, API Keys, and tokens in environment variables.
- **Documentation Generation**: Postman can generate and publish API documentation automatically from collections, providing consistency between development and documentation.

Postman Development Use Cases

- **API Development**: Developers test API endpoints as they build them, checking inputs, outputs, and error handling.
- **API Testing and Validation**: QA teams verify APIs against functional requirements, edge cases, and error scenarios.
- **Before Front-End Integration**: Back-end APIs can be tested and validated in Postman before front-end developers integrate them into apps/websites.
- **Continuous Integration/Continuous Development (CI/CD)**: Automated Postman tests can be integrated into CI/CD pipelines for regression testing by running Postman collections using its CLI tool.
- **Learning/Exploring Third-Party APIs**: Developers exploring public APIs can quickly test requests and understand responses (such Aruba Central APIs, Google Maps APIs, etc).

In summary, Postman is used throughout the entire API lifecycle, from development, debugging, and documentation to testing and automation.

PyCentral SDK

For Python developers, the PyCentral SDK is a Software Development Kit that provides a high-level, easy-to-use interface to Central's APIs. It reduces the complexity of direct API calls, allowing users to write more readable and maintainable Python scripts. The SDK is ideal for building automation workflows, managing configurations, and integrating Central into existing IT systems.

Benefits of Using PyCentral SDK

- **Simplified API Calls**: Provide Python methods instead of raw REST calls, reducing coding overhead.
- **Built-in Authentication**: Handles token generation and management, removing manual token handling.
- **Consistency**: Ensures that API requests align with Aruba Central's supported schemas and best practices.
- **Rapid Prototyping**: Speeds script and workflow development by abstracting repetitive tasks.
- **Integration Ready**: Easily embeds into automation frameworks (Ansible, CI/CD, GitOps pipelines).
- **Error Handling and Reliability**: Uses standardized methods to reduce the risk of malformed requests.
- **Developer Friendly Operations**: Lower the learning curve for engineers new to Aruba Central APIs, enabling quicker adoption.

Central-Python-Workflows

The central-python-workflows repository provides a curated set of Python-based workflows, code samples, and supporting Postman collections designed to accelerate automation with the Central and HPE GreenLake Platform (GLP) APIs. Built on the PyCentral SDK, it delivers a structured framework for interacting with Central's REST APIs and extensibility features. With these workflows, administrators can programmatically create, update, and assign configurations using Python, reducing repetitive manual tasks. This approach improves operational efficiency while aligning network operations with modern DevOps practices, CI/CD pipelines, and Infrastructure-as-Code methodologies.

Benefits of Using Central-Python-Workflows

- **Faster Automation**: Ready-to-use samples shorten the time to begin automating Central tasks.
- **Consistency and Reliability**: Enforces API best practices, lowering the risk of misconfiguration.
- **Extensibility**: Built on PyCentral SDK for easy customization and integration with existing toolchains.
- **Efficiency Gains**: Automates repetitive operations such as WLAN/VLAN configuration profile creation, scope assignments, or monitoring, freeing administrators for higher-value work.
- **CI/CD Integration**: Supports Infrastructure-as-Code pipelines, version control, and automated testing.
- **Cross-Platform Enablement**: Complements Postman collections, giving teams multiple options to test and deploy APIs.
- **Scalability**: Scales across sites, groups, and tenants, making it suitable for both enterprise deployments and MSP environments.

Best Practice for API Config Automation

Best practice for using these resources includes:

- Prototype workflows in Postman.
- Move to Python scripts for automation with PyCentral SDK.
- Use YAML configs for flexibility.
- Securely manage tokens and secrets.
- Validate every change with a `GET`.
- Adopt CI/CD integration for production rollouts.

Configuration with PyCentral SDK

The PyCentral SDK provides a Python interface to the Central APIs hosted on the HPE GreenLake platform. It abstracts authentication, API calls, and resource operations into structured Python classes that map directly to Central API categories (configuration, monitoring, inventory, etc.).

Directory Structure

```
pycentral/
├── __init__.py
├── base.py
├── classic/                   #
│      ├── base.py             # token management and base API calls
│      ├── configurations.py   # group related configs etc
│      ├── topology.py         # many other basic utils and management python files
│      ├── ....
│
├── glp/
│      ├── devices.py         # devices, subscriptions and services related files
│       ├── ....
│
├── profiles/           # WLANs, VLANs, roles, groups profiles mgmt.
│   ├── __init__.py
│   ├── policy.py
│   ├── profiles.py
│   └── role.py
│   ├── system_info.py
│   ├── vlan.py
│   └── wlan.py
│
├── scopes/              # scopes, sites, site_collection mgmt
│   ├── device.py
│   └── scopes.py
│   └── site.py
│   └── site_collections.py
│
├── utils/                  # glp/profiles/scope utils
│
└── exceptions/             # error management
│
└── troubleshooting/        # troubleshooting related
```

Authentication and Session Handling

The **NewCentralBase** class initializes a secure session with Central. It uses OAuth 2.0 tokens obtained from the GreenLake identity service.

```
from pycentral import NewCentralBase


central_info = {
  "new_central": {
      "base_url": "central-api-base-url",
      "client_id": "new-central-client-id",
      "client_secret": "ew-central-client-secret"


  }
}


central_conn = NewCentralBase(
   token_info=central_info,
    log_level="CRITICAL",
    enable_scope=True,
)
```

After instantiation, the same session object is passed to all API category classes for stateful operations for client identity and authentication.

Module and Class Hierarchy

Each API category is represented by a class where each method corresponds to one API endpoint. Endpoints are URLs that define the location of the resource on the server. For example, `https://api.example.com/users` might be an endpoint for accessing user data. The following table shows modules written for implementing WLAN configuration profiles, VLAN configuration profiles, device monitoring, and troubleshooting, with example methods

Module	Class	Example Methods	Purpose
`pycentral.profiles.wlan`	Wlan	`create_wlan()`, `update_wlan()`, `delete_wlan()`	WLAN and SSID management
`pycentral.profiles.vlan`	`Vlan`	`create_vlan()`, `get_vlan()`	VLAN configuration
`pycentral.monitoring`	`DeviceMonitoring`	`get_device_status()`, `get_ap_metrics()`	Monitoring APIs
`pycentral.troubleshooting`	`Troubleshooting`	`ping_device()`, `capture_logs()`	New diagnostic APIs (v2.x)

API Call Models

Step 1 Structured (Typed) Calls using PyCentral SDK

Use this option when working with well-defined, stable REST APIs. It uses the PyCentral SDK’s built-in classes and methods to interact with Central, rather than direct API calls.

```
from pycentral.profiles import Wlan

resp = Wlan.create_wlan(session, group_name="Campus_APs",
                        wlan_name="Guest_SSID",
                        ssid_type="WPA2_PERSONAL",
                        passphrase="guest123")
```

Step 2 Direct Base Calls using REST APIs

Use this approach for early-access or beta API endpoints that are not yet supported by the PyCentral SDK. In such cases, direct REST API calls enable testing and automation of new features before they are officially integrated into the SDK.

```
from pycentral import NewCentralBase


resp = session.command(apiMethod="POST",
                       apiPath="/network-config/v1alpha1/wlan-ssids",
                       apiData={"ssid": "New_WLAN", "enable": True})
```

PyCentral SDK Deployment Template

Configuring with the PyCentral SDK follows a logical sequence similar to the Central UI workflow, but in fully automated form. The SDK abstracts the complexity of REST API calls, providing higher-level functions and objects that let administrators focus on defining the desired configuration outcome rather than managing individual API requests.

The following outline summarizes the key steps for creating and deploying configurations using the PyCentral SDK. The steps use deployment of a WLAN configuration profile to highlight the details with snippets of Python script.

Step 1 Instantiate the Central API Client: The first step is to establish a connection to the Central account. The PyCentral SDK handles authentication and token management. Provide the API Gateway URL and access token in the script configuration. Refer to the Developer Hub for token creation instructions.

```
import sys
import yaml
from pycentral import NewCentralBase
from pycentral.profiles import Wlan

# Connect to Central instance with credentials from account_credentials.yaml file
# you can set the log_level to either DEBUG or INFO for more detailed output
central_conn = NewCentralBase(
  token_info="account_credentials.yaml",
  log_level="CRITICAL",
  enable_scope=True,
  )
if not central_conn:
     sys.exit(1)
```

Step 2 Define Configuration Data: The script takes a YAML file as input, which defines the WLAN configuration profile parameters. This structured file specifies essential attributes such as the SSID name, encryption type (for example, WPA2-Personal), and passphrase, along with optional settings supported by the Central API. The `targets` section in the YAML file identifies the site or scope where the WLAN profile is deployed. Using a YAML-based input enables clear, human-readable configuration, supports version control, and ensures repeatable automated deployments across environments. The snippet below is from the *wlan_config.yaml* file used in this example script.

```
# the content of wlan_config.yaml file

wlan:
  ssid: "SecureWiFi"
  enable: true
  forward-mode: "FORWARD_MODE_L2"
  essid:
    name: "SecureWiFi"
  opmode: "WPA2_PERSONAL"
  personal-security:
    passphrase-format: "STRING"
    wpa-passphrase: "MySecretPresharedKey1234567890"
  default-role: "secureTunnel-authenticated"
  vlan-selector: "VLAN_RANGES"
  vlan-id-range:
    - "101"

# The WLAN profile will be deployed on the following target, where the APs are associated with.
targets:
  sites:
    - "NYCCP-CNX"
```

Step 3 Create Configuration Profile: The first part in the code snippet below reads the data from the input *wlan_config.yaml* file. The second part of the snippet uses a function *Wlan.create_wlan()* from the PyCentral SDK to create the WLAN configuration profile.

```
# Read the input wlan_config.yaml file for wlan profile and site details
with open("wlan_config.yaml", "r") as f:
  cfg = yaml.safe_load(f) or {}
  wlan = cfg.get("wlan") or {}
  targets = cfg.get("targets", {}) or {}
  target = targets.get("sites")[0]
  if not wlan:
       sys.exit(1)

print(f"\nThis demo will attempt to create WLAN config Profile for {wlan['ssid']} and associate the profile with Site -

# create WLAN configuration Profile.
print(f"Step 1: Create Wlan Configuration Profile for {wlan['ssid']}")
```

```
resp = Wlan.create_wlan(central_conn, wlan,{})
if not resp:
  print(f"\n\tWlan config profile Create operation Failed: {wlan['ssid']}")
  sys.exit(1)
else:
  print(f"\n\tWlan config profile Created : {wlan['ssid']}")
```

Step 4 Assign Profile to a Scope and Device Function: After the profile is created, it is assigned to site scope and device function. The function *assign_profile_to_scope()* from PyCentral SDK associates the WLAN profile to a site scope

```
# Associate Wlan config profile with Site scope, but first - get ID for the site scope
sites = central_conn.scopes.get_all_sites()
scope_id=""
for site in sites:
 if site.name == target:
    scope_id = site.get_id()

# Now Associate Wlan config profile with Site scope. The profile persona parameter in the
# assign_profile_to_scope() function call is the device persona, which in this case is CAMPUS ACCESS POINTs
profile_res = f"{Wlan.get_resource()}/{wlan['ssid']}"
print(f"\nStep 2: Associate Wlan Configuration Profile with site - {target}")
central_conn.scopes.assign_profile_to_scope(
  profile_name=profile_res,
  profile_persona="CAMPUS_AP",
  scope="site",
  scope_id=scope_id
  )

# we are done
print("\tAssigned profile to site successfully\n\nStep 3: Demo Run complete.. ")
sys.exit(0)
```

Step 5 Error Handling and Troubleshooting

The REST API calls always return an error code as part of the response message on completion. This return value is checked for success [code equals *200] or failure [code does not equal *200*] of the API call. Common error codes include:

Error Code	Meaning	Common Fix
200	Success	—
400	Bad Request	Fix JSON/YAML syntax
401/403	Unauthorized	Refresh token
409	Conflict	Object already exists

Using PyCentral SDK streamlines the entire automation process, from initial connection to final configuration push, significantly reducing the amount of code required and minimizing the risk of errors.

The Aruba Developer Hub **New Central** → **Configuration API** section details how to use Configuration API.

Package Installation and Environment Setup

The next step is to setup the runtime environment by installing the PyCentral SDK and all referenced Python dependencies. This structured setup ensures consistent behavior across environments, enables reliable authentication to Central APIs, and provides a repeatable foundation for deploying and validating WLAN configuration profiles through automation.

> **Note:** The configuration workflows described in this document reference functionality included in a beta release of the pycentral SDK (v2.0a8+). This version is currently not publicly available. External users will be able to access these capabilities only after the SDK is officially released.

Python packages used by this script:

- `pycentral` - Central's API client library (beta version v2.0a8 or later)
- `PyYAML` - YAML parsing for configuration files

Setup

Open a terminal session in the working directory use to store the script and configuration files. Ensure that Python 3.9 or newer is installed on the system

Step 1 Use the following commands to create a virtual environment to execute the script:

```
python -m venv venv
source venv/bin/activate
```

Step 2 Install the required packages. Copy the content below to a *requirements.txt* file and run the pip install command as shown

```
# copy the following lines and create a requirements.txt file

pycentral==2.0a9
PyYAML>=6.0
```

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

Step 3 Copy the following new_central info into an *account_credentials.yaml* file and update the file with the base URL of the Central cluster, client ID, and client secret issued. Refer to the Developer Hub for token creation instructions. Next, copy the WLAN info into *wlan_config.yaml* file, along with target site information.

```
# create a account_credentials.yaml file with this info

new_central:
   base_url: "base URL for your Central cluster"
   client_id: "enter your client_id"
   client_secret: "enter your client_secrete"


# Create a wlan_config.yaml file with this content

# WLAN profile
# This assumes that the AP is on-boarded and is online. The sites/device-groups already configured and APs assigned.

wlan:
 ssid: "SecureTunnel"
 enable: true
 forward-mode: "FORWARD_MODE_L2"
 essid:
   name: "SecureTunnel"
 opmode: "WPA2_PERSONAL"
 personal-security:
   passphrase-format: "STRING"
   wpa-passphrase: "MySecretPresharedKey1234567890"
 default-role: "secureTunnel-authenticated"
 vlan-id-range:
   - "101"
 vlan-selector: "VLAN_RANGES"

# The WLAN profile will be deployed on the following target list
# one or more sites and one or more device-groups - if any is specified
```

```
targets:
 sites:
   - "NYCCP-CNX"
 groups:
```

Step 4 Copy the following script into a file named *wlan_deploy.py*.

```
#!/usr/bin/env python3
"""
# DISCLAIMER:
# This script is provided for demonstration and educational purposes only.
# It is not supported or maintained and should be used at your own discretion.
#

# This script depends on an internal beta of pycentral (v2.0a8 or later).
# This version is not currently available to external users and will only be
# accessible after the official public release.


Purpose
-------
Create a WLAN configuration profile in Central from wlan_config.yaml input and hen
assign that profile to existing Site.

Usage
-----
 python3 wlan_deploy.py

runtime
-------
 1. step 1: connects to Central instance
 2. Step 2: creates WLAN configuration profile based on input yaml content
 3. step 3: Associates the WLAN configuration profile to the target Site
"""

import sys
import yaml
from pycentral import NewCentralBase
from pycentral.profiles import Wlan

# Connect to Central instance with credentials from account_credentials.yaml file
# you can set the log_level to either DEBUG or INFO for more detailed output
central_conn = NewCentralBase(
   token_info="account_credentials.yaml",
   log_level="CRITICAL",
   enable_scope=True,
   )
if not central_conn:
      sys.exit(1)

# Read the input wlan_config.yaml file for wlan profile and site details

with open("wlan_config.yaml", "r") as f:
   cfg = yaml.safe_load(f) or {}
   wlan = cfg.get("wlan") or {}
   targets = cfg.get("targets", {}) or {}
   target = targets.get("sites")[0]
   if not wlan:
        sys.exit(1)

print(f"\nThis demo will attempt to create WLAN config Profile for {wlan['ssid']} and associate the profile with Site -

# create WLAN configuration Profile
print(f"Step 1: Create Wlan Configuration Profile for {wlan['ssid']}")
resp = Wlan.create_wlan(central_conn, wlan,{})
if not resp:
   print(f"\n\tWlan config profile Create operation Failed: {wlan['ssid']}")
   sys.exit(1)
else:
   print(f"\n\tWlan config profile Created : {wlan['ssid']}")

# Associate Wlan config profile with Site scope, but first - get ID for the site scope
sites = central_conn.scopes.get_all_sites()
scope_id=""
for site in sites:
  if site.name == target:
     scope_id = site.get_id()

# Now Associate Wlan config profile with Site scope. The profile_persona parameter is the device function and in this c
profile_res = f"{Wlan.get_resource()}/{wlan['ssid']}"
print(f"\nStep 2: Associate Wlan Configuration Profile with site - {target}")
central_conn.scopes.assign_profile_to_scope(
   profile_name=profile_res,
        profile_persona="CAMPUS_AP",
        scope="site",
        scope_id=scope_id
        )

    # we are done
    print("\tAssigned profile to site successfully\n\nStep 3: Demo Run complete.. ")
    sys.exit(0)
```

Step 5 Run the script.

Before running the script, ensure that all Access Points (APs) are onboarded onto their respective Site(s) within Central and that they are online. This guarantees that configuration assignments are applied successfully. When executed, the script authenticates the user with Central using the credentials and token information provided, creates the defined WLAN configuration profile, and assigns that profile to the appropriate scope (Site) within the tenant hierarchy. This scoped deployment ensures that only the intended Site receives the WLAN configuration, maintaining consistent, tenant-aware policy application across the environment.

Make sure the following files are in the working directory before running the script

```
demo/
├── account_credentials.yaml   # baseurl, client_id and client_secret info
├── wlan_config.yaml           # wlan profile and target info
├── requirements.txt           # packages names and version desired info
├── venv/                      # virtual environment directory created after setup
├── wlan_deploy.py             # The demo script
```

Run the script.

```
python wlan_deploy.py
```