Oktanet Team

AUTOMATION

Automate Oktavia with its API, Ansible and Python

API Ansible Python November 10, 2026 · 12-minute read

Oktavia already orchestrates the network from its interface and from Argos. Many operations teams also live in Ansible, AAP or their own Python scripts. This article shows how to use Oktavia from there, from zero, without losing what makes it safe: intent as the source of truth, the preview before saving and the Digital Twin with a person who approves.

The rule: automate around the Twin, not through it

Oktavia is the only path to the network. A playbook or script **reads, proposes and verifies**: it queries inventory and compliance, asks for a change preview, saves the intent and gets the change proposed in the Digital Twin. It **never approves, applies or confirms** on a device: a person does, with each platform's prior validation and confirmed commit. It is not a limitation of the kit; it is the design.

Step 1 · The keys

There are two kinds of key, each for its own job:

KEY	WHERE IT IS CREATED	WHAT IT OPENS
Engine scoped key (`okt_…`)	An administrator creates it on the engine, bound to one tenant, read or write.	Inventory, compliance, intent, metrics and a change preview. It does not save intent or touch the Twin.
Interface automation key (`oktui_…`)	Integrations → Automation keys, administrators only.	The customer service flows: preview, save and remove. It never approves or applies.

Both are shown once, stored hashed, revoked without touching the others, and every use is logged. None of them goes in files: use environment variables or ansible-vault.

```
export OKTAVIA_URL=https://oktavia-engine.example:8000
export OKTAVIA_ORG=my-tenant
export OKTAVIA_API_KEY=okt_…          # engine scoped key
```

```
export OKTAVIA_UI_URL=https://oktavia.example
export OKTAVIA_UI_KEY=oktui_…         # automation key
```

Step 2 · Dynamic inventory

The Ansible kit ships an inventory plugin that reads the tenant's devices and groups them by role, site, site type and platform. There is no `hosts` file to maintain: the inventory is Oktavia's.

```
ansible-inventory --graph
@all:
  |--@role_pe:
  |  |--pe-cdmx-01
  |--@platform_xr:
  |  |--pe-cdmx-01
  |  |--p-core-01
```

Step 3 · Read compliance and state

The `01_lectura.yml` playbook builds a report with devices, compliance per test, failing tests, customer services and BGP and IS-IS neighbors. The same in Python:

```
import os, requests

url, org = os.environ["OKTAVIA_URL"], os.environ["OKTAVIA_ORG"]
h = {"X-API-Key": os.environ["OKTAVIA_API_KEY"]}
q = {"organization_slug": org}

inventory = requests.get(f"{url}/api/inventory", headers=h, params=q, timeout=30).json()
results = requests.get(f"{url}/api/compliance/results", headers=h, params=q, timeout=60).json()
bgp = requests.get(f"{url}/api/metrics/bgp-peers", headers=h, params=q, timeout=30).json()
```

The API returns secrets as `<oculto>` to any scoped key: a report cannot leak a community or a key.

Step 4 · Change a parameter with a preview

A change is a file with the intent path and the value. The `02_cambiar_parametro.yml` playbook asks Oktavia for the preview —the CLI lines or NETCONF XML that change on each device— and stops without saving.

```
# cambios/ntp_ce.yml
oktavia_cambio:
  descripcion: "CE NTP"
  ruta: "by_role.ce.services.ntp.servers"
  valor: ["192.0.2.10", "192.0.2.11"]
  equipos: [ce-customer-01, ce-customer-02]
```

```
ansible-playbook playbooks/02_cambiar_parametro.yml -e @cambios/ntp_ce.yml
```

The path is the same one edited in Services: `<scope>[.<key>].services.<service>…`. Lists are replaced whole, as in the interface. To see the impact before saving from Python, send the candidate intent to `POST /api/intended/preview-vars` with the device list.

Step 5 · A customer service by playbook

The **Argos → Custom orchestration** flows also exist as playbooks: `10_pseudowire.yml`, `11_l3vpn_sitio.yml`, `12_color_odn.yml` and `13_multicast.yml`. Each takes the service in a file and an action:

```
# servicios/pseudowire.yml
oktavia_servicio:
  customer: CUSTOMER_A
  name: CDMX-MTY-3
  vlan: 102
  subnet: 192.0.2.16/29
  a: {pe: pe-cdmx-01, pe_interface: GigabitEthernet0/0/0/2, ce: ce-customer-01, ce_interface:
GigabitEthernet0/1}
  b: {pe: pe-mty-01,  pe_interface: GigabitEthernet0/0/0/2, ce: ce-customer-02, ce_interface:
GigabitEthernet0/1}
```

```
ansible-playbook -i localhost, playbooks/10_pseudowire.yml -e @servicios/pseudowire.yml -e
accion=preview
ansible-playbook -i localhost, playbooks/10_pseudowire.yml -e @servicios/pseudowire.yml -e
accion=apply
```

- **preview** validates and shows what changes; it saves nothing.
- **apply** saves the intent and returns the change proposed in the Twin, with its link.
- **retire** removes the service the same way.

They are idempotent: if the service is already there, they say so and do nothing. In Python it is the same call:

```
r = requests.post(f"{os.environ['OKTAVIA_UI_URL']}/api/argos/l2vpn",
                  headers={"Authorization": f"Bearer {os.environ['OKTAVIA_UI_KEY']}"},
                  json={"action": "preview", **service}, timeout=180)
```

Then a person opens the change in the Digital Twin, reviews the diff, approves it, runs the test and applies it. On IOS XR it is a confirmed commit: if verification fails, the device rolls back on its own. In Argos every flow has **Do it with Ansible** with the command equivalent to what was filled in the form.

Step 6 · Verify templates and intent

The `03_verificar_plantillas.yml` playbook renders Oktavia's Jinja templates locally with the tenant's intent and compares the result with Oktavia's desired configuration, service by service. It helps to understand how configuration is built and to review an extension before publishing it. It follows the engine's

rules: scopes in order, objects that merge, lists that are replaced, and a sandbox where the template cannot call Ansible.

Good practices

- **One key per integration**, read-only unless it really needs to write. Rotate it and revoke it when a project ends.
- **Always preview.** Run `preview` in the pipeline and keep `apply` as a step approved by a person.
- **Secrets in the vault.** The intent carries `<<vault:…>>`, never the value.
- **Documentation addresses** in the examples you share: `192.0.2.0/24`, `198.51.100.0/24`, `203.0.113.0/24`.
- **Close the loop with the audit.** A change ends when compliance confirms it, not when the playbook finishes.

The full kit —dynamic inventory, playbooks and examples— ships with Oktavia. Ansible, API and NETCONF fundamentals are in the free course at the Academy.

About Oktanet

Oktanet is a Mexican network automation company focused on controlled, traceable, and scalable operations for enterprise environments across Latin America.

LinkedIn · **YouTube**

© 2026 Oktanet · Oktavia · Public article, with no confidential customer information.