Stop clicking,

Start scripting.

Automating infrastructure with REST API and RESTCONF

Amin Hamidi Younessi - Netrotik Oy

It's Not About the Vendor.

MikroTik

- Proprietary REST API
- RouterOS V7.1beta4
- HTTPS + JASON
- Popular in EU ISPs and IXPs

Cisco IOS-XE

- RESTCONF (RFC 8040)
- YANG data models
- HTTPS + JASON
- Enterprise and carrier grade

The Truth

- Same HTTP protocol
- Same Python library
- Same automation goal
- Only the path differs

How many of you...

Made an SSH to 10+ routers this week?

Copy-paste config. Repeat 30 times.
One typo breaks a peer.

Manually provisioned a BGP peer?

Winbox / SSH session by session.
No audit trail. No rollback.

Updated NTP on your entire boxes?

Half a day of your time.
For a one-line change.

By the end of this session, a script does all of that in less than 10 seconds

3. > CODE COMPARISON

Same Python. Different Endpoint.

MikroTik — Proprietary REST

```
# BGP session state
get("routing/bgp/session")

# Provision peer — PUT
put("routing/bgp/connection", {
  "name": "IXP-Member-R2",
  "instance": "instance1",
  "remote.address": "192.168.200.2",
  "remote.as": "64512",
  "local.role": "ebgp"
})

# NTP push — POST
post("system/ntp/client/set", {...})
```

Cisco IOS-XE — RESTCONF RFC 8040

```
# BGP neighbor state
get("Cisco-IOS-XE-bgp-oper:"
    "bgp-state-data/neighbors")

# Provision BGP — PUT
put("Cisco-IOS-XE-native:native/router/bgp", {
  "Cisco-IOS-XE-bgp:bgp": [{
    "id": 65000,
    "neighbor": [
     {"id":"192.168.200.2",
      "remote-as": 64512}
    ]
  }]
})

# NTP push — PATCH
patch("Cisco-IOS-XE-native:native/ntp", {...})
```

Live Demo — Cisco IOS-XE via RESTCONF

Same topology. Same Python. YANG models instead of proprietary paths.

01 **Query State**
GET interfaces + live BGP session status

02 **Provision BGP**
PUT BGP AS65000 + 2 IXP peers — zero CLI

03 **MASS NTP Push**
PATCH NTP to all 4 routers simultaneously

Lab Stack

→ EVE-NG on GCP

→ 4× CSR1000v

→ IOS-XE 17.03.05

→ RESTCONF enabled

→ Ubuntu Linux node

→ Python 3 + requests

→ VSCode

DEMO 01 QUERY ROUTER STATE

Pull Live Data via RESTCONF

1. GET hostname & IOS-XE version
2. GET all interfaces with IP addresses
3. GET live BGP neighbor state
4. Show established sessions + prefix count

```
cisco_demo_01.py
 HOST    = "https://203.0.113.1"
 HEADERS = {"Accept":
    "application/yang-data+json"}

 # BGP operational state
 oper = get(

 "Cisco-IOS-XE-bgp-oper:"
  "bgp-state-data/neighbors"
 )

 for n in oper["neighbors"]["neighbor"]:
  state  = n["connection"]["state"]
  icon   = "✓" if state == "established" else "○"
  print(f"{icon} {n["neighbor-id"]} [{state}]")
```

DEMO 02 BGP PROVISIONING

Provision IXP Peers via RESTCONF

1. BEFORE: confirm R1 has no BGP config
2. PUT BGP AS65000 + router-id + 2 neighbors
3. Payload uses Cisco-IOS-XE-bgp:bgp YANG key
4. AFTER: verify config + live session state

cisco_demo_02.py

```
# Exact YANG key from live router GET
payload = {
  "Cisco-IOS-XE-bgp:bgp": [{
    "id": 65000,
    "bgp": {"router-id":
      {"ip-id":
"203.0.113.1"}},
    "neighbor": [
      {"id": "192.168.200.2",
       "remote-as": 64512},
      {"id": "192.168.200.3",
       "remote-as": 64513}
    ]
  }]
}

requests.put(HOST + "/restconf/data/...bgp",
        json=payload, auth=AUTH, verify=
False)
```

DEMO 03 MASS NTP PUSH

One Script. Four Routers. Zero CLI.

1. Define fleet — 4× CSR1000v routers
2. PATCH NTP YANG config to each router
3. Before / after verification per host
4. Summary: 4 / 4 updated

cisco_demo_03.py

```
ROUTERS = [
 {"name":"R1-RouteServer", "host":"203.0.113.1"},
 {"name":"R2-IXPMember1",  "host":"203.0.113.2"},
 {"name":"R3-IXPMember2",  "host":"203.0.113.3"},
 {"name":"R4-EdgeRouter",  "host":"203.0.113.4"},
]

for router in ROUTERS:
  r = requests.patch(
    f"https://{router['host']}"
    "/restconf/data/"
    "Cisco-IOS-XE-native:native/ntp",
    json=NTP_PAYLOAD,
    auth=AUTH, verify=False
  )
  icon = "✓" if r.ok else "✗"
  print(f"{icon} {router['name']}")
```

What to Take Away

- **The API is just a path**

MikroTik uses /rest/.
Cisco uses /restconf/data/.
The Python around it is identical. Learn the pattern once, apply it everywhere.

- **YANG is the open standard**

MikroTik uses /rest/.

RESTCONF + YANG is RFC 8040. Cisco, Juniper, Nokia — any vendor supporting it speaks the same language.

- **Start small, scale fast**

One script for one task. BGP provisioning, NTP push, interface audit. Same pattern whether 4 routers or 400.

- **Knowledge over tooling**

Ansible, Netmiko, RESTCONF — they are tools. HTTP, JSON, and YANG make you effective on any platform, any vendor.