🌐 One model, a whole network

*The fabric layer of **netclab-xp**, powered by AVD*

> ***An extensible data model that defines Arista's Unified Cloud Network architecture as "code"***
> Powered by avd.arista.com and crossplane.io

📐 Where this sits

One network `Fabric`

▼

One device `Router`

▼

One setting `RoutedInterface` / `BgpNeighbor` / `IpRouting`

🧩 Not a separate product

The `Fabric` API is **authored in function-avd** and ships as `configuration-avd`, which netclab-xp names in `dependsOn`:

```yaml
kind: Configuration
metadata:
  name: netclab-xp
spec:
  dependsOn:
    # ... provider-http, function-eapi and three contrib functions
    - apiVersion: pkg.crossplane.io/v1
      kind: Configuration
      package: xpkg.upbound.io/netclab/configuration-avd
      version: ">=v0.1.5"
```

Installing netclab-xp installs the fabric layer with it.

⚙ What the function does

- **Takes one AVD `eos_designs` document** → tenants, VRFs, VLANs, uplinks
- **Runs pyavd** → a complete, validated configuration for *every* device in the design
- **Pushes over eAPI** → to the devices you say are actually running
- **Validation lands on `status`** → a bad model is a message, not a broken switch

One document in. Per-device configuration out.

📌 The design is pinned, not copied

The scenario does not hold a copy of the model. It **reaches for it, at a tag**:

```
resources:
  - github.com/netclab/function-avd/examples/lab?ref=v0.1.6
```

A topology derived from an AVD model is only valid for the AVD version that derived it — and that version lives upstream.

> ***So "what happens when AVD moves?" has an answer: nothing, until someone moves the pin.***

✍️ The entire scenario

```
- op: add
  path: /spec/design/tenants/0/vrfs/0/svis/-
  value:
    id: 13
    name: VRF10_VLAN13
    enabled: true
    ip_address_virtual: 10.10.13.1/24
```

Four lines. One SVI, in a tenant VRF.

What reached `dc1-leaf1a`

```
vlan 13
   name VRF10_VLAN13
interface Vlan13
   description VRF10_VLAN13
   vrf VRF10
   ip address virtual 10.10.13.1/24
interface Port-Channel8
   switchport trunk allowed vlan 11-13,21-22,3401-3402
interface Vxlan1
   vxlan vlan 13 vni 10013
router bgp 65101
   vlan 13
      rd 10.255.0.3:10013
      route-target both 10013:10013
```

🧠 What you did not write

- `vni 10013` — the VXLAN VNI
- `rd 10.255.0.3:10013` — the route distinguisher, built from the leaf's own loopback
- `route-target both 10013:10013` — the EVPN route-target
- `switchport trunk allowed vlan 11-13,…` — the trunk list, *widened* rather than replaced

None of it appears in the patch. All of it is derived from the design.

🎯 And where it did *not* land

`dc1-spine1` **received the push too** — and has **no** `vlan 13` **at all**.

```
kubectl -n avd exec dc1-spine1 -- Cli -p 15 -c "show vlan 13"
```

Nothing was excluded by hand. A tenant SVI belongs on leaves, so the model put it on leaves.

> ***The interesting output is the device that was configured with nothing.***

📊 One document, eight switches

```
deviceCount: 8
validation:
  ok: true
Ready: True
```

- **Every device in the design gets a rendered configuration** — eight of them here, from one document
- **pyavd validates the model first**, so a bad design is a message on `status` rather than a broken network

The design is what you maintain. The per-device configuration is derived.

♻️ Teardown leaves the switch configured

```
kubectl delete -k scenarios/fabric
```

The `Fabric`, its `Device`s, the rendered ConfigMaps and the requests all go. **`vlan 13` stays.**

This is the opposite of every other netclab-xp scenario, and it is deliberate: the requests carry `Observe, Create, Update` and **no `Delete`**.

> ***A layer that pushes a switch's entire configuration must not have a teardown that wipes the switch.***

To reset a device, restart its pod.

🎯 Try it

https://netclab.dev

Repos — the package, and the function that renders the fabric:

https://github.com/netclab/netclab-xp

https://github.com/netclab/function-avd

Registry:

https://marketplace.upbound.io/configurations/netclab/netclab-xp