SANOG 44 · CONTAINERLAB

Containerlab

A modern approach to networking labs

Lab-as-code with FRR, Cisco, Nokia SR Linux, Juniper & MikroTik — including a full bdNOG 21 BGP topology.

Suman Kumar Saha

suman@hashtag.com.bd · 30-minute session

AGENDA

What we'll cover

1. What & why

Lab-as-code, the workflow

2. Install

One-liner on Linux / WSL2 / Proxmox

3. Node types & images

FRR · Cisco · SR Linux · Juniper · MikroTik

4. Build & deploy

Topology file, lifecycle, startup configs

5. BIRD route server

IXP route server demo

6. VM NOSes

vrnetlab: MikroTik CHR, vMX

7. bdNOG 21 topology

The BGP workshop, as code

8. Capture & Edgeshark

BGP packets and session states

9. Telemetry

Prometheus + Grafana, step by step

10. Proxmox & resources

LXC setup and real usage

MOTIVATION

Why I went looking for something better

> *I was preparing a multivendor labs on GNS3 and EVE-NG. Every router was a full virtual machine, so I upgraded my desktop to 32 GB of RAM — and I still could not run the whole topology at once.*

Every VM boots a whole OS

A modest ISP-style lab means a dozen full operating systems fighting for the same RAM and CPU.

So labs get cut down

You end up practising on 3–4 routers instead of a production-shaped network — and the gap shows.

Containers change the economics

Containerised NOSes share the host kernel. The same topology that choked my desktop runs quietly in a small Proxmox container.

WHAT IS IT

A ‘Containerlab’ as code” way to deploy networking labs

200K+	100+	30+	24
installations	contributors	supported NOSes	releases / 12 mo

Light weight **Declarative** **Shareable** **Container-based** **Fast** **Open**

One YAML file describes your nodes and links, **and clab builds the whole topology — FRR, Cisco, Nokia SR Linux, Juniper, MikroTik, BIRD, telemetry, all in one file.**

`containerlab.dev`

Collaboration · Versioning · Sharing

Collaboration

The lab is a text file. Review it, comment on it, hand it to a teammate.

Versioning

Commit topologies to Git. Diff changes, roll back, branch a variant.

Sharing

Clone a repo and get an identical lab in seconds.

A lab is just this:

```
name: my-lab
topology:
  nodes:
    frr1: { kind: linux, image: quay.io/frrouting/frr:9.1.0 }
    pop1: { kind: cisco_iol, image: vrnetlab/cisco_iol:17.12.01 }
  links:
    - endpoints: [frr1:eth1, pop1:e0/1]
```

THE WORKFLOW

Install → Create → Deploy → Interact → Save

The loop is always the same. Install once, then create or clone a topology, deploy it, log into nodes, and save your work back to Git.

INSTALL

The all-in-one installer

Works on Linux, WSL2 OR inside LXC — installs Docker CE, containerlab and the GitHub CLI in one shot.

```
# The fastest way to get started
curl -sL https://containerlab.dev/setup \
  | sudo bash -s "all"

# Then log out / log in for group changes
```

Installs

- docker-ce
- containerlab
- GitHub CLI

Verify the install:

```
$ docker run --rm hello-world          # → Hello from Docker!

$ containerlab version
  version: 0.6x.x

# Other options: apt/yum packages · GitHub release binary · clab container image
```

NODE TYPES

Three kinds of nodes

Containerized NOS

vendor-built, native

- FRR (open source)
- Cisco XRd · IOL
- Nokia SR Linux
- Juniper cRPD
- Arista cEOS
- Fast, small, shareable

VM-in-container

via vrnetlab

- MikroTik RouterOS (CHR)
- Juniper vMX / vEVO
- Cisco XRv9k / c8000v
- Nokia SR OS
- Bring-your-own qcow2

Regular containers

any Docker image

- BIRD (route server)
- Linux hosts / clients
- Prometheus, Grafana
- gNMIC collectors
- Routinator (RPKI)
- Traffic generators

Getting the images we use today

FRR	`quay.io/frrouting/frr:9.1.0` Public pull. Runs as kind: linux with daemons + frr.conf bind-mounted.
Nokia SR Linux	`ghcr.io/nokia/srlinux` Public pull, no registration. Native kind: nokia_srlinux.
Juniper cRPD	`crpd:24.x (docker load)` Download from Juniper support, load the tarball. kind: juniper_crpd.
MikroTik RouterOS	`vrnetlab/mikrotik_routeros:7.x` Free CHR qcow2 from mikrotik.com → build once with vrnetlab. kind: mikrotik_ros.

Cisco gets its own slide — several images, several routes to obtain them →

Cisco images available today

cisco_iol

IOS On Linux — classic IOS-XE CLI, smallest footprint. From the CML image bundle.

cisco_xrd

XRd Control Plane — native IOS XR container (7.x+). Cisco software download.

cisco_xrv9k

IOS XRv 9000 — full XR VM via vrnetlab (heavier, full data plane).

cisco_c8000v

Catalyst 8000v / CSR1000v — IOS-XE VM via vrnetlab.

cisco_n9kv

Nexus 9000v — NX-OS VM via vrnetlab, for DC labs.

Use a production Cisco router in the lab

Containerlab can attach a lab link to a physical NIC or Linux bridge on the host — so a real router cabled to that port peers directly with your lab nodes.

```
links:
  # lab node ↔ host NIC (real router
  # is cabled to enp2s0 / a VLAN)
  - endpoints: ["pop1:e0/4",
                "host:enp2s0"]
```

- Great for testing real gear against a virtual fabric

A minimal FRR + Cisco topology

```
name: basic
topology:
  nodes:
    frr1:
      kind: linux
      image: quay.io/frrouting/frr:9.1.0
      binds:
        - daemons:/etc/frr/daemons
        - frr1.]conf:/etc/frr/frr.conf
    iol1:
      kind: cisco_iol
      image: vrnetlab/cisco_iol:17.12.01
  links:
    - endpoints: [frr1:eth1, iol1:e0/1]
```

Node

A container. The node name becomes the container name.

Kind

How clab boots it — linux for FRR, cisco_iol for IOL.

Binds

Mount FRR's daemons + frr.conf straight into the node.

Links

A veth pair joining frr1:eth1 to iol1:e0/1.

Deploy, then inspect what's running

```
$ sudo containerlab deploy -t basic.clab.yml
+---+-----------------+--------------+-------------------------+-----------+---------+----------------+
| # | Name            | Container ID | Image                   | Kind      | State   | IPv4 Address   |
| 1 | clab-basic-frr1 | ab12f9fe0c51 | quay.io/frrouting/frr:9.1.0| linux     | running | 172.20.20.2/24 |
```

Connect to nodes

```
# Names auto-added to /etc/hosts
$ ssh clab-basic-iol1
$ docker exec -it clab-basic-frr1 vtysh
frr1# show ip route
```

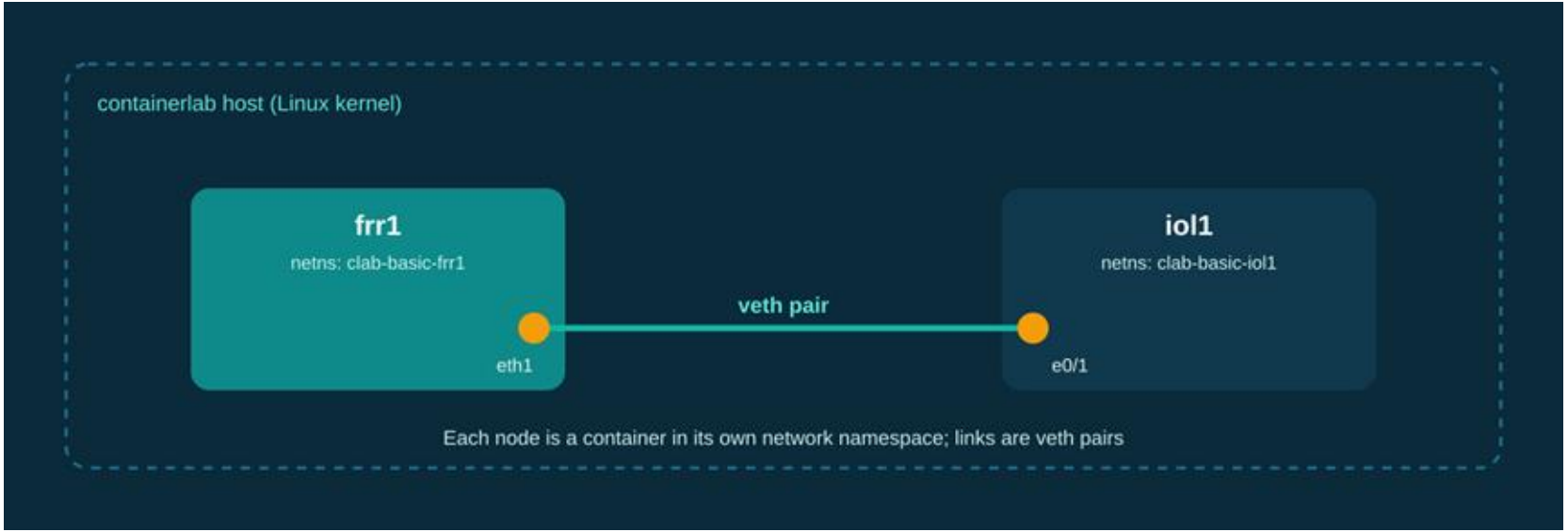
Nodes come pre-baked

- Management reachable over the clab mgmt network
- SSH keys set up per kind
- vtysh drops you straight into the FRR CLI

List labs

```
$ sudo clab inspect -t basic.clab.yml       # this lab       $ sudo clab inspect --all   # every lab
```

How containerlab wires nodes



Every node runs in its own network namespace.

The lab directory & cleaning up

```
clab-basic/
├── ansible-inventory.yml
├── authorized_keys
├── topology-data.json
├── frr1/  (configs, logs)
└── iol1/  (nvram, logs)
```

`clab destroy -t lab.yml`

Remove containers; keep the lab directory

`clab destroy -t lab.yml --cleanup`

Remove containers and the lab directory

`clab destroy --all`

Remove containers for every running lab

`clab save -t lab.yml`

Write running configs back to startup files

STARTUP CONFIGS

Declarative device configuration

Nodes boot configured

No manual paste — config applies at deploy.

Keep it in Git

External files make the lab fully declarative.

Each kind's native syntax

FRR frr.conf, IOS CLI, RouterOS script.

Reference it in the topology file

```
topology:
  nodes:
    core1:                                  pop1:
      kind: linux                             kind: cisco_iol
      binds:                                  startup-config: startup/pop1.cfg
        - startup/core1/frr.conf:/etc/frr/frr.conf
```

Same lab, two dialects: FRR & Cisco IOL

FRR — startup/core1/frr.conf

```
hostname core1
!
interface eth1
 ip address 10.10.0.0/31
!
interface lo
 ip address 10.0.0.1/32
!
router bgp 10
 bgp router-id 10.0.0.1
 neighbor 10.10.0.1 remote-as 10
 address-family ipv4 unicast
  network 10.0.0.1/32
 exit-address-family
```

Cisco IOL — startup/pop1.cfg

```
hostname pop1
!
interface Ethernet0/1
 ip address 10.10.0.1 255.255.255.254
 no shutdown
!
interface Loopback0
 ip address 10.0.0.21 255.255.255.255
!
router bgp 10
 bgp router-id 10.0.0.21
 neighbor 10.10.0.0 remote-as 10
 network 10.0.0.21 mask 255.255.255.255
```

BIRD: the IXP route server, in your lab

What BIRD is

- Open-source routing daemon (BGP, OSPF, RIP, RPKI)
- **The de-facto route server at real IXPs worldwide**
- Tiny footprint — a few MB of RAM in a plain container
- Powerful filter language for per-peer policy
- In containerlab: just a linux node with bird.conf bind-mounted

Deploy a BIRD route server (AS33)

1 · Topology node

```
rs:
  kind: linux
  image: pierky/bird:2.15
  binds:
    - bird/rs-bird.conf:/etc/bird/bird.conf:ro
```

2 · bird.conf — the rs client line is the magic

```
router id 10.33.0.33;
protocol bgp iig1 {
  local 10.33.0.33 as 33;
  neighbor 10.33.0.11 as 11;
  rs client;    # don't prepend AS33
  ipv4 { import all; export all; };
}
# ...one protocol block per member
```

3 · Verify with birdc

```
$ docker exec -it clab-bdnog-rs \
    birdc show protocols
Name    Proto   State    Info
iig1    BGP     up       Established
iig2    BGP     up       Established

$ docker exec -it clab-bdnog-rs \
    birdc show route all
```

4 · Prove it's a route server

On IIG2, look at a prefix learned from IIG1 via the RS:
`AS_PATH = 11 40` — no 33 in the path. The RS reflected the route without inserting itself.

VM-BASED NOSES

“Where are my good old VM NOSes?”

Traditional VM network OSes run in containerlab too — **packaged as a VM inside a container via the vrnetlab project.**

Supported via vrnetlab

- **MikroTik RouterOS (CHR)**
- **Juniper vMX / vSwitch / vEVO**
- Cisco XRv9k / c8000v / NX-OS
- Nokia SR OS
- Fortinet, Palo Alto, Huawei, OcNOS …

Why it matters

Mix container-native and VM NOSes in one topology.

That's how we rebuild the bdNOG 21 workshop: FRR core, Cisco IOL POPs, MikroTik externals and a BIRD route server — all in one YAML file.

Building the MikroTik image with vrnetlab


```
$ git clone https://github.com/hellt/vrnetlab.git && cd vrnetlab/routeros
$ cp ~/images/chr-7.16.img.zip .        # free CHR image from mikrotik.com
$ make                                   # → vrnetlab/mikrotik_routeros:7.16
```

CASE STUDY

The bdNOG 21 Advanced BGP workshop — as code

The original	A 30-node RouterOS topology: ISP1 (AS10) with cores, aggregation, POPs, DMZ, IXP routers and CDN caches — surrounded by upstreams, IXPs, CDNs and customers in their own ASes.
Our rebuild	Same design, multi-vendor: FRR for the AS10 core/aggregation, Cisco IOL for the POPs, MikroTik CHR for every external AS (as in the workshop), and BIRD for the AS33 route server.
Why it's a great test	Transit, IXP peering, route-server policy, CDN on-net caches, customer BGP — every pattern an ISP engineer needs to practise, on one laptop-sized lab.

AS10 · AS11 · AS20 · AS22 · AS30 · AS33 · AS40 · AS50 · AS55 · AS65 · AS70 · AS80 · AS85 · AS95 · AS65501

Showing the topology view in containerlab

One command serves an interactive graph rendered from the YAML — the lab does not even need to be deployed.

Run it (sample output)

```
$ sudo containerlab graph -t bdnog.clab.yml
INFO Parsing & checking topology file:
       bdnog.clab.yml
INFO Serving topology graph on
       http://0.0.0.0:50080
```

Browse `http://<lxc-ip>:50080` — the next slide is exactly what you get.

Export flavours

```
$ ./show-topology.sh drawio    # editable .drawio for slides
$ ./show-topology.sh mermaid   # markup for READMEs
```

Web view · :50080

containerlab graph — quick and interactive; works before deploy

VS Code TopoViewer

Containerlab extension: live view; click a node to open its CLI. group/graph-level labels in the YAML drive the layout

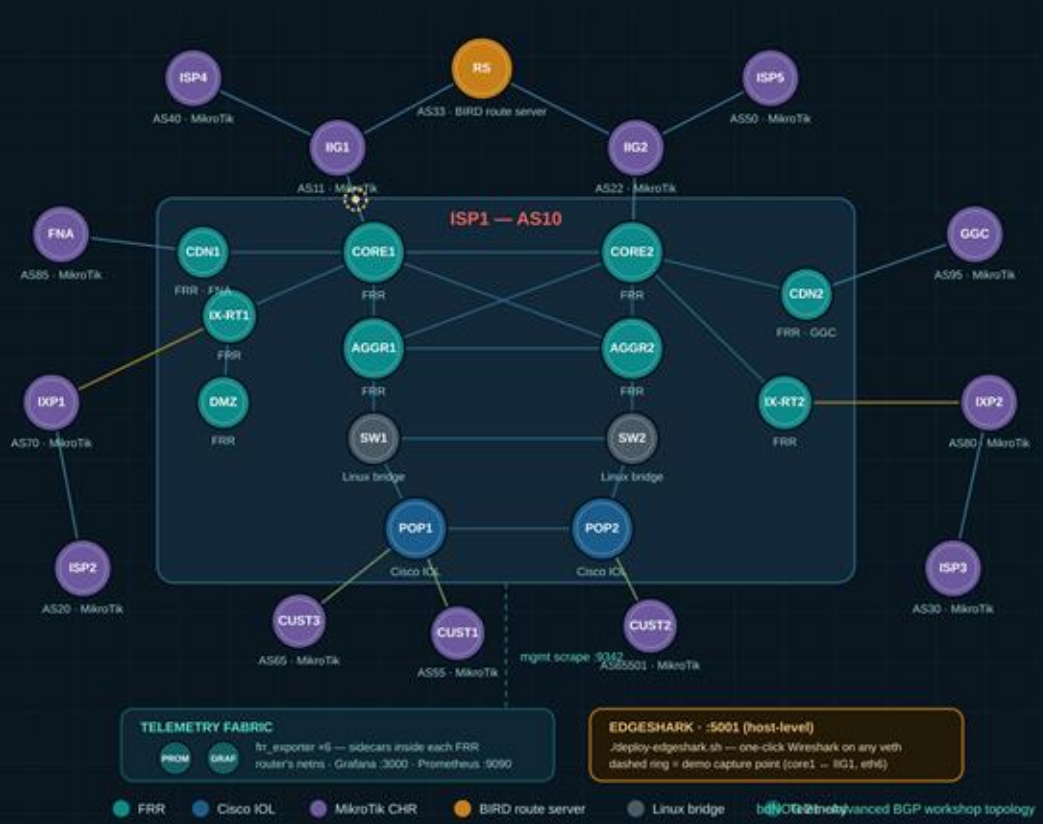
draw.io export

--drawio → restyle in diagrams.net, export PNG/SVG

Mermaid export

--mermaid → paste into GitHub docs

Sample output — http://<host>:50080



bdnog.clab.yml — kinds & nodes

```
name: bdnog
topology:
  kinds:
    linux:                          # FRR
      image: quay.io/frrouting/frr:9.1.0
    cisco_iol:
      image: vrnetlab/cisco_iol:17.12.01
    mikrotik_ros:
      image: vrnetlab/mikrotik_routeros:7.16

  nodes:
    # ISP1 — AS10 (FRR)
    core1: { kind: linux, binds: [...] }
    core2: { kind: linux, binds: [...] }
    aggr1: { kind: linux, binds: [...] }
    aggr2: { kind: linux, binds: [...] }
    ix-rt1: { kind: linux, binds: [...] }
    cdn1:  { kind: linux, binds: [...] }
```

```
    # POPs — Cisco IOL
    pop1: { kind: cisco_iol,
            startup-config: startup/pop1.cfg }
    pop2: { kind: cisco_iol,
            startup-config: startup/pop2.cfg }

    # Route server — AS33 (BIRD)
    rs:
      kind: linux
      image: pierky/bird:2.15
      binds: [bird/rs-bird.conf:...]

    # Externals — MikroTik CHR
    iig1: { kind: mikrotik_ros }   # AS11
    iig2: { kind: mikrotik_ros }   # AS22
    isp2: { kind: mikrotik_ros }   # AS20
    # ...isp3-5, ixp1/2, fna-sp,
    #    ggc-sp, cust1-3 (14 total)

    # Telemetry fabric
    prometheus: { ports: [9090:9090] }
    grafana:    { ports: [3000:3000] }
    core1-exp:                  # x6
      network-mode: container:core1
      # frr_exporter in the router's
      # netns -> metrics on :9342
```

Links, then one command to build it all

```
links:
  # AS10 core mesh
  - endpoints: ["core1:eth1", "core2:eth1"]
  - endpoints: ["core1:eth2", "aggr1:eth1"]
  # POPs behind the switches
  - endpoints: ["switch1:eth3", "pop1:e0/1"]
  # transit & route server
  - endpoints: ["core1:eth6", "iig1:eth1"]
  - endpoints: ["iig1:eth3",  "rs:eth1"]
  # IXP fabric · customers
  - endpoints: ["ix-rt1:eth3", "ixp1-sw:eth1"]
  - endpoints: ["pop1:e0/3",   "cust1:eth1"]
  # ... 34 links total
```

Deploy & watch

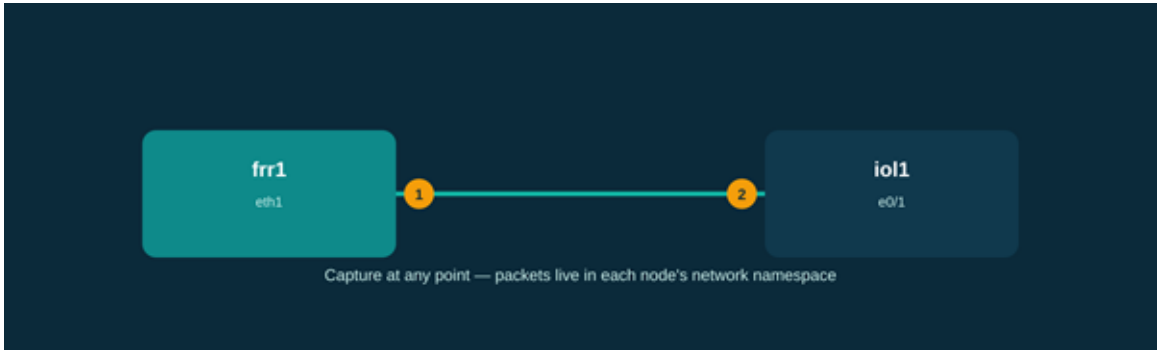
```
$ cd bdnog
$ sudo containerlab deploy \
    -t bdnog.clab.yml

# 37 nodes (incl. 6 exporters)
# ~2-3 min first boot
# (CHR VMs are the slow part)
```

Quick checks

- clab inspect -t bdnog.clab.yml
- vtysh on core1: show bgp summary
- birdc on rs: show protocols
- Grafana :3000 · Prometheus :9090
- ./deploy-edgeshark.sh -> :5001

Three ways to sniff traffic



1 · Local
tcpdump in the node's netns, on the host

2 · Remote
SSH pipe straight into desktop Wireshark

3 · Web UI
Edgeshark — click an interface, capture

Because links are native Linux veth pairs, **you can capture on any interface with tools you already know — no agents inside the nodes.**

Local & remote capture

Local — output on the host

```
$ sudo ip netns exec clab-bdnog-core1 tcpdump -nni eth3 tcp port 179
```

Remote — live into desktop Wireshark

```
ssh user@clab-host "sudo ip netns exec clab-bdnog-core1 tcpdump -U -nni eth3 -w -" \
  | wireshark -k -i -
```

On Windows (WSL2), point at the Windows Wireshark binary

```
ssh user@clab-host "sudo ip netns exec clab-bdnog-core1 tcpdump -U -nni eth3 -w -" \
  | /mnt/c/Program\ Files/Wireshark/wireshark.exe -k -i -
```

Edgeshark — how it actually works

1 **Ghostwire (web UI, :5001)**

Discovers every container, network namespace and interface on the host and lists them in the browser — refresh and the whole bdNOG lab appears.

2 **Packetflix (capture service)**

Streams packets from any chosen interface over a packetflix:// URL — no capture agent is installed in the nodes.

3 **cshargextcap (Wireshark plugin)**

A one-time install on your laptop. It registers the packetflix:// handler, so clicking the shark-fin opens Wireshark already capturing live.

```
curl -sL https://github.com/siemens/edgeshark/raw/main/deployments/wget/docker-compose.yaml | docker compose -f - up -d
```

Demo: capture BGP between core1 and IIG1

1	**Open the UI**	http://<lab-host>:5001 — hit the refresh icon; every clab-bdnog-* container appears.
2	**Pick the interface**	Find clab-bdnog-core1 → eth3 (the transit link to IIG1, AS11).
3	**Click the shark-fin**	Wireshark opens on your laptop, already streaming that veth live.
4	**Filter to BGP**	Display filter: bgp (or tcp.port == 179 to include the TCP handshake).
5	**Shake the session**	On core1: vtysh -c 'clear bgp 10.10.1.1' — and watch the FSM walk back to Established.

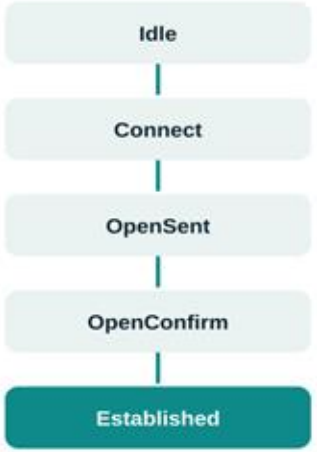
What you see: BGP packets & session states

Wireshark — clab-bdnog-core1 : eth3 (display filter: bgp)

No.	Time	Source	Destination	Proto	Info
1	0.000	10.10.1.1	10.10.1.2	TCP	179 → 41522 [SYN, ACK]
2	0.014	10.10.1.2	10.10.1.1	BGP	OPEN Message — AS10, Hold 180
3	0.019	10.10.1.1	10.10.1.2	BGP	OPEN Message — AS11, Hold 180
4	0.021	10.10.1.2	10.10.1.1	BGP	KEEPALIVE Message
5	0.024	10.10.1.1	10.10.1.2	BGP	KEEPALIVE Message
6	0.310	10.10.1.1	10.10.1.2	BGP	UPDATE — 24 NLRI, AS_PATH 11 40
7	0.342	10.10.1.2	10.10.1.1	BGP	UPDATE — 12 NLRI, AS_PATH 10

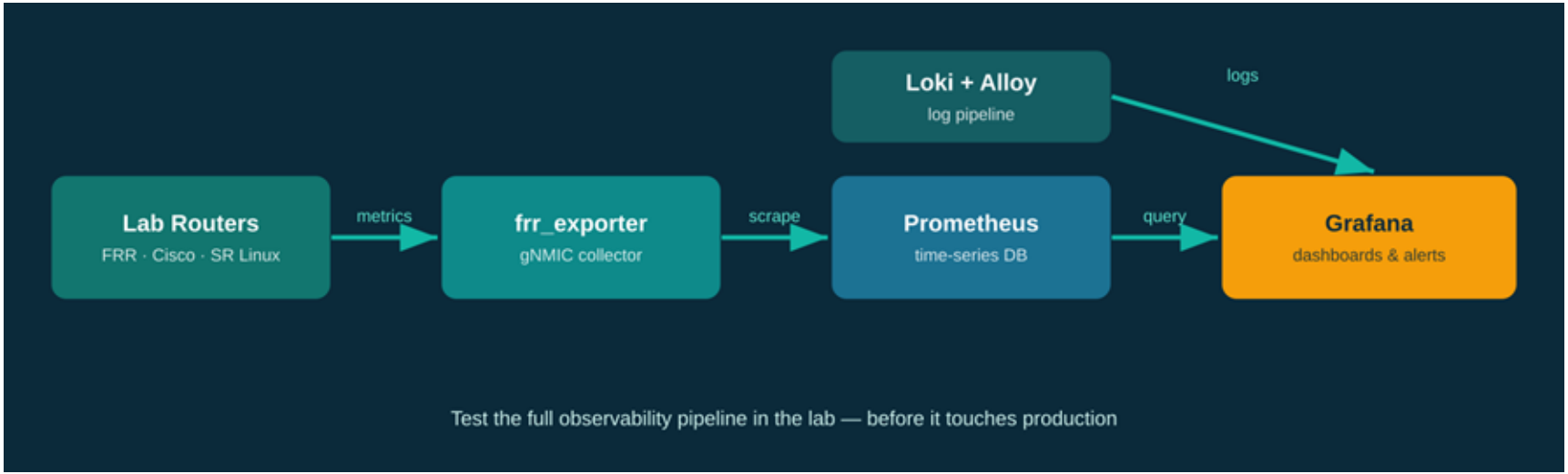
```
▶ Border Gateway Protocol — OPEN Message
Version: 4
My AS: 10 Hold Time: 180
BGP Identifier: 10.0.0.1
Capabilities: MP-BGP (IPv4/IPv6), Route Refresh,
4-octet AS, Graceful Restart
```

BGP session FSM




```
core1# show bgp summary
10.10.1.1 AS11 ... Estab 00:04:12
```

Stream telemetry from the bdNOG fabric



The whole observability pipeline runs inside the same topology: **frr_exporter on each FRR node → Prometheus → Grafana.**

Five steps to a live dashboard

1

Pin management IPs

mgmt: ipv4-subnet + per-node mgmt-ipv4 — so scrape targets survive redeploys.

```
mgmt: { ipv4-subnet: 172.90.90.0/24 }
core1: { mgmt-ipv4: 172.90.90.11 }
```

2

Attach frr_exporter

One sidecar per FRR node, sharing its network namespace (port 9342).

```
core1-exp:
  network-mode: container:clab-bdnog-core1
  image: quay.io/frrouting/frr_exporter
```

3

Point Prometheus at them

Static targets by pinned IP; scrape every 15 s.

```
- targets: ['172.90.90.11:9342',
            '172.90.90.12:9342']
```

4

Provision Grafana

Datasource + dashboard JSON bind-mounted — zero clicks after deploy.

```
binds:
  - telemetry/grafana/provisioning:...
```

5

Deploy & generate traffic

clab deploy, start pings/iperf between customers, open :3000.

```
$ sudo containerlab deploy -t bdnog.clab.yml
$ ./traffic.sh start all
```

Step-by-step: this whole lab in a Proxmox LXC

1 — Create a privileged LXC

```
pct create 9000 ... --unprivileged 0 --features
nesting=1,keyctl=1  (6 vCPU / 12 GiB / 40 GB)
```

2 — Host-side tweaks

```
In /etc/pve/lxc/9000.conf: apparmor unconfined, cgroup2
devices allow, proc/sys rw mounts
```

3 — Start & prep

```
pct start 9000 → pct enter 9000 → apt update && apt install -
y curl git
```

4 — Install the stack

```
curl -sL https://containerlab.dev/setup | sudo bash -s "all"
→ verify docker + clab
```

5 — Build VM images once

```
vrnetlab: make MikroTik CHR 7.16 and Cisco IOL images; docker
images to confirm
```

6 — Clone & deploy

```
git clone <your repo> && cd bdnog && sudo containerlab deploy
-t bdnog.clab.yml
```

7 — Add the tooling

```
Edgeshark compose up (:5001)  · Grafana already provisioned
(:3000)
```

8 — Work, save, destroy

```
clab save to capture configs  · clab destroy -c when done —
the YAML rebuilds it anytime
```

A full written guide with every command accompanies this talk.

PUTTING IT TOGETHER

One YAML file → a working ISP, observed end to end

Describe

37 nodes incl. telemetry, 4 vendors + BIRD — one file in Git

Deploy

containerlab deploy; minutes, not evenings

Inspect

Edgeshark → live BGP packets and FSM states

Observe

frr_exporter → Prometheus → Grafana

Afford it

Runs in one modest Proxmox LXC

Share it

A colleague clones the repo and has the same ISP

Realistic, multi-vendor, observable — and light enough for anyone in this room to run tonight.

GET IN TOUCH

Where to go next

Thank you — questions?

Suman Kumar Saha · suman@hashtag.com.bd · SANOG 44 · Topology adapted from the bdNOG 21 Advanced BGP workshop; structure inspired by the NANOG 93 Containerlab workshop