ContainerLab + SONiC VS

Build a full data-center fabric on your Laptop / VS

Author: Srinivas M Koppal , Senior Technical Lead, PalC Networks Pvt Ltd

Problem Statement

Validating Data-Center network designs traditionally requires access to costly physical hardware, making iterative testing and CI/CD integration impractical for most engineering teams. Engineers lack a lightweight, reproducible environment to experiment with production-grade NOS behavior — such as BGP policy, ECMP, and interface configuration — before changes reach the live fabric. ContainerLab combined with SONiC Virtual Switch addresses this gap by enabling full spine-leaf topologies to be deployed on a standard laptop in minutes, at zero hardware cost.

ContainerLab is an open-source network lab orchestration framework that uses Docker containers to emulate real network operating systems. Combined with SONiC Virtual Switch, network engineers can spin up full spine-leaf data-center topologies - complete with live BGP sessions, FRR routing, and production-grade configuration workflows - in under three minutes, on a standard Laptop or Virtual Machine, with zero cost.

This white paper covers the architecture of both tools, walks through a complete two-spine four-leaf deployment, documents every operational command, and explains the interface and BGP configuration model in detail. It is intended for network engineers, automation practitioners, and anyone evaluating open-source NOS platforms for lab, CI/CD, or training environments.

< 3 min	6 nodes	100%	Real NOS
To deploy a full fabric	2 Spines + 4 Leaves	open-source, zero cost	FRR + SONiC config_db

Lab Topology — Two-Spine, Four-Leaf

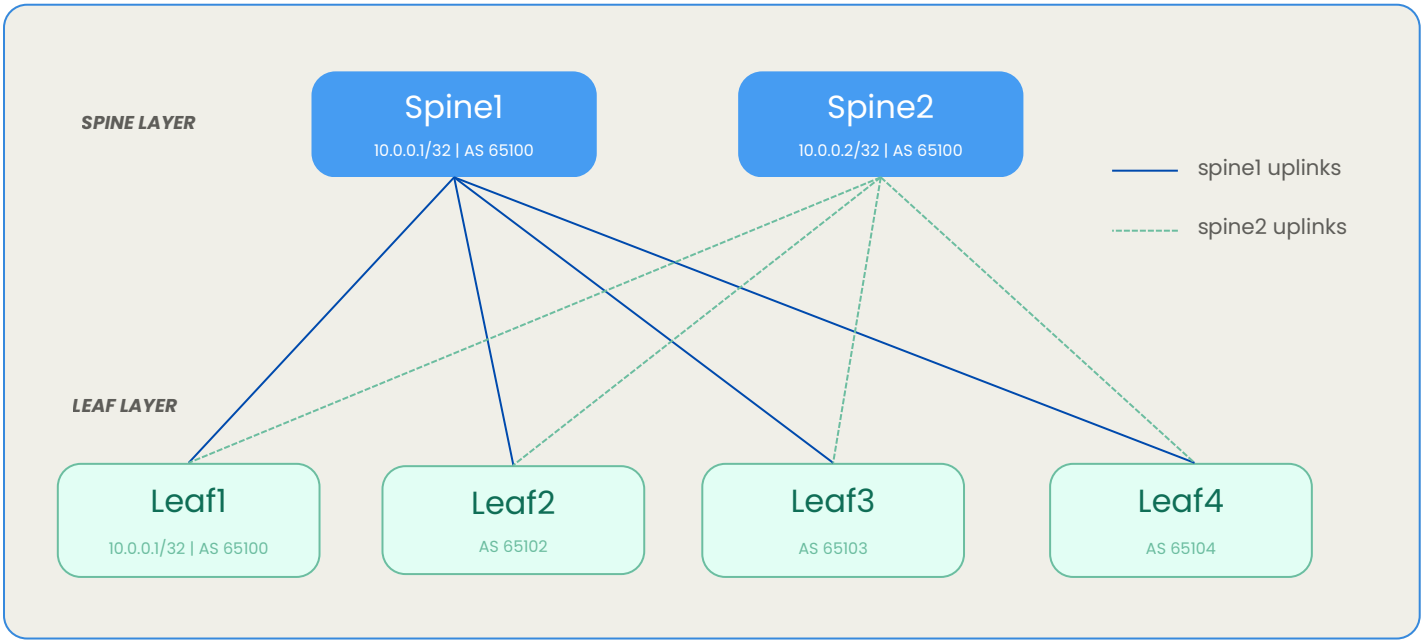

Figure 1: SONiC VS spine-leaf fabric deployed with ContainerLab. Solid lines = spine1 uplinks; dashed = spine2 uplinks.

1. ContainerLab — Architecture & Core Concepts

ContainerLab is a Go-based CLI tool that manages the full lifecycle of containerised network topologies. It reads a declarative YAML topology file and translates it into a set of Docker containers connected by virtual Ethernet (veth) pairs. Every node gets a management IP on an isolated Docker bridge network (clab-mgmt), and ContainerLab injects SSH keys automatically so you can reach any node with a single ssh command immediately after deployment.

ContainerLab supports a wide range of network operating systems as container images: Nokia SR Linux, Arista cEOS, Juniper cRPD, FRRouting, and SONiC Virtual Switch — among others. For SONiC VS specifically, ContainerLab maps Linux ethN interfaces inside the container to SONiC's EthernetN naming scheme, handling the offset arithmetic automatically.

1.1 Topology File Structure

A topology file is a YAML document with three main sections: **name** (lab identifier), **mgmt** (management network config), and **topology** (nodes and links). Each node entry specifies the container image, kind (e.g. **sonic-vs**), and optional startup config or environment variables. Links are defined as pairs of node:interface endpoints.

```yaml
# spine-leaf.clab.yml — ContainerLab topology definition
name: sonic-spine-leaf

mgmt:
  network: clab-mgmt
  ipv4-subnet: 172.20.20.0/24

topology:
  kinds:
    sonic-vs:
      image: docker.io/sonic-vs:latest

  nodes:
    spine1:
      kind: sonic-vs
      mgmt-ipv4: 172.20.20.7
    spine2:
      kind: sonic-vs
      mgmt-ipv4: 172.20.20.3
    leaf1:
      kind: sonic-vs
      mgmt-ipv4: 172.20.20.4
    # ... leaf2, leaf3, leaf4 follow same pattern

  links:
    - endpoints: ["spine1:eth1", "leaf1:eth1"]
    - endpoints: ["spine1:eth2", "leaf2:eth1"]
    - endpoints: ["spine1:eth3", "leaf3:eth1"]
    - endpoints: ["spine1:eth4", "leaf4:eth1"]
    - endpoints: ["spine2:eth1", "leaf1:eth2"]
    - endpoints: ["spine2:eth2", "leaf2:eth2"]
    - endpoints: ["spine2:eth3", "leaf3:eth2"]
    - endpoints: ["spine2:eth4", "leaf4:eth2"]
```

1.2 Lab Lifecycle Commands

```
# spine-leaf.clab.yml - ContainerLab topology definition
name: sonic-spine-leaf

mgmt:
  network: clab-mgmt
  ipv4-subnet: 172.20.20.0/24

topology:
  kinds:
    sonic-vs:
      image: docker.io/sonic-vs:latest

  nodes:
    spine1:
      kind: sonic-vs
      mgmt-ipv4: 172.20.20.7
    spine2:
      kind: sonic-vs
      mgmt-ipv4: 172.20.20.3
    leaf1:
      kind: sonic-vs
      mgmt-ipv4: 172.20.20.4
    # leaf2, leaf3, leaf4 follow same pattern

  links:
    - endpoints: ["spine1:eth1", "leaf1:eth1"]
    - endpoints: ["spine1:eth2", "leaf2:eth1"]
    - endpoints: ["spine1:eth3", "leaf3:eth1"]
    - endpoints: ["spine1:eth4", "leaf4:eth1"]
    - endpoints: ["spine2:eth1", "leaf1:eth2"]
    - endpoints: ["spine2:eth2", "leaf2:eth2"]
    - endpoints: ["spine2:eth3", "leaf3:eth2"]
    - endpoints: ["spine2:eth4", "leaf4:eth2"]
```

2. SONiC Virtual Switch — Architecture & Internals

Software for Open Networking in the Cloud (SONiC) is a Linux-based, open-source network operating system originally developed by Microsoft for Azure's data-center fabric. It has since become a Linux Foundation project and is deployed at hyperscale by Google, Meta, Alibaba, and numerous cloud providers.

SONiC's key architectural innovation is the Switch Abstraction Interface (SAI) — a vendor-neutral API that decouples the control plane from the switching ASIC. Above SAI, SONiC uses a central Redis database called CONFIG_DB as the single source of truth for all switch configuration. Every subsystem (BGP daemon, interface manager, ACL manager) reads from and writes to CONFIG_DB, making the entire state observable and reproducible.

2.1 SONiC VS Architecture

Component	Description	Relevance in VS
CONFIG_DB	Redis-based central config store	All interface and BGP config lives here
FRRouting(FRR)	BGP, OSPF, IS-IS routing suite	Handles all routing protocol processing
bgpd	BGP daemon within FRR	Must be enabled manually in VS
vtysh	FRR unified CLI shell	Primary interface for BGP verification
supervisord	Process supervisor	Manages all SONiC daemons
syncd	SAI sync daemon	In VS, uses sairedis stub (no real ASIC)
portsyncd	port state sync	Maps eth interfaces to EthernetIN
teamd	LAG/LACP daemon	Port-channel support
config CLI	SONiC config tool	Wraps CONFIG_DB writes for operators

2.2 Interface Naming & Mapping

ContainerLab creates Linux **ethN** interfaces inside the SONiC VS container. SONiC maps these to its internal **EthernetN** naming convention using a fixed offset of 4 per port. This mapping is critical when writing configs — always use the SONiC name in config commands, but understand it corresponds to the ContainerLab link endpoint.

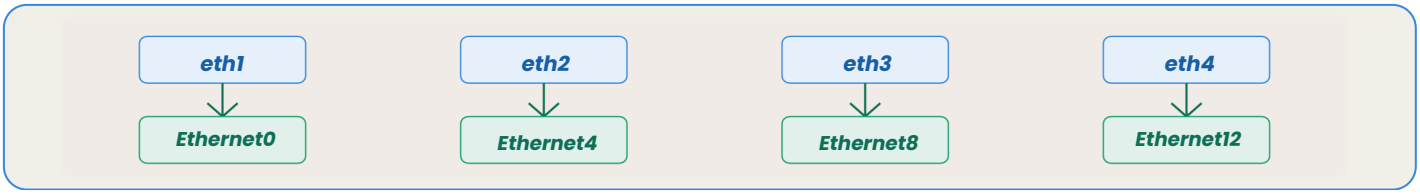

Figure 2: ContainerLab ethSONiC Ethernet interface mapping (offset of 4 per port).

2.3 Service Management

```
# Check all running SONiC services
supervisorctl status

# bgpd is NOT started by default in SONiC VS — enable it:
sed -i 's/bgpd=no/bgpd=yes/' /etc/frr/daemons
supervisorctl start bgpd
supervisorctl restart bgpd

# Verify bgpd is running
supervisorctl status bgpd
ps aux | grep bgpd
```

3. Interface & BGP Configuration

3.1 Bringing Up Interfaces

In SONiC VS,all data-plane interfaces start ina down state. You must explicitly bring each interface up and assign an IP address using the **config** CLI tool. These commands write directly to CONFIG_DB and persist across restarts if you save the config.

```
# Interface bring-up (run inside the SONiC VS node)
config interface startup Ethernet0
config interface startup Ethernet4

# Assign point-to-point IP to Ethernet0 (spine1 -> leaf1)
config interface ip add Ethernet0 192.168.1.0/31
config interface ip add Ethernet4 192.168.1.2/31
config interface ip add Ethernet8 192.168.1.4/31
config interface ip add Ethernet12 192.168.1.6/31

# Add loopback IP (used as BGP router-id and for reachability)
config interface ip add Loopback0 10.0.0.1/32

# Verify interface state
show interface status
show ip interfaces
```

3.2 BGP Configuration via vtysh

FRRouting'svtysh provides a Cisco-IOS-style CLI for configuring and verifying BGP. The spine nodes run eBGP with AS 65100 and peer with each leaf's unique private AS (65101–65104). This is a standard unnumbered eBGP spine-leaf design commonly used in modern data-center fabrics following RFC 7938.

```
# Enter vtysh and configure BGP on spine1
vtysh
spine1# configure terminal
spine1(config)# router bgp 65100
spine1(config-router)# bgp router-id 10.0.0.1
spine1(config-router)# no bgp ebgp-requires-policy
# Peer with leaf1 on Ethernet0 link
spine1(config-router)# neighbor 192.168.1.1 remote-as 65101
spine1(config-router)# neighbor 192.168.1.1 description leaf1
spine1(config-router)# neighbor 192.168.1.3 remote-as 65102
spine1(config-router)# neighbor 192.168.1.3 description leaf2
spine1(config-router)# neighbor 192.168.1.5 remote-as 65103
spine1(config-router)# neighbor 192.168.1.7 remote-as 65104

# Advertise loopback into BGP
spine1(config-router)# address-family ipv4 unicast
spine1(config-router-af)# network 10.0.0.1/32
spine1(config-router-af)# exit-address-family
spine1(config-router)# exit
spine1(config)# end
spine1# write memory
# Summary of all BGP peers and state
spine1# show bgp summary

# Detailed per-neighbor info (timers, prefix counts)
spine1# show bgp neighbors
spine1# show bgp neighbors 192.168.1.1
```

4. Node Access & Docker Operations

4.1 Accessing Nodes

```
# Enter node shell directly as root
docker exec -u root -it clab-sonic-spine-leaf-spine1 bash
docker exec -u root -it clab-sonic-spine-leaf-leaf1 bash

# SSH using ContainerLab hostname (keys auto-injected)
ssh admin@clab-sonic-spine-leaf-spine1
ssh admin@clab-sonic-spine-leaf-leaf1

# SSH using management IP directly
ssh admin@172.20.20.7  # spine1
ssh admin@172.20.20.3  # spine2
ssh admin@172.20.20.4  # leaf1

# Run a single command without entering shell
docker exec -u root clab-sonic-spine-leaf-spine1 vtysh -c "show bgp summary"
docker exec -u root clab-sonic-spine-leaf-spine1 supervisorctl status
```

4.2 Container Lifecycle & Monitoring

```
# Resource usage of all clab containers
docker stats $(docker ps --format "{{.Names}}" | grep clab)

# Container logs
docker logs clab-sonic-spine-leaf-spine1
docker logs -f clab-sonic-spine-leaf-spine1    # follow live

# Restart / Stop / Start individual nodes
docker restart clab-sonic-spine-leaf-spine1
docker stop clab-sonic-spine-leaf-spine1
docker start clab-sonic-spine-leaf-spine1

# Docker view of all clab containers
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Image}}" | grep clab
```

4.3 File Transfer To/From Nodes & Link inspection

```
# Copy a config file INTO a node
docker cp myconfig.json clab-sonic-spine-leaf-spine1:/etc/sonic/

# Copy the running config OUT of a node
docker cp clab-sonic-spine-leaf-spine1:/etc/sonic/config_db.json ./spine1-config.json

# Save all node configs at once (ContainerLab native)
sudo containerlab save -t spine-leaf.clab.yml
# Configs saved to: ~/sriniv/palc-sonic/clab-sonic-spine-leaf/

# List all clab-related docker networks
docker network ls | grep clab

# Inspect the management bridge network
docker network inspect clab

# Check veth links between nodes (from host)
ip link show | grep clab

# Capture traffic on a veth link
tcpdump -i clab-spine1-eth1 -n
```

5. Lab Node Reference

5.1 Management & BGP Reference

Node	Mgmt Ip	Loopback0	BGP AS	Role
Spine1	172.20.20.7	10.0.0.1/32	65100	Spine
Spine2	172.20.20.3	10.0.0.2/32	65100	Spine
Leaf1	172.20.20.4	10.0.0.11/32	65101	Leaf
Leaf2	172.20.20.5	10.0.0.12/32	65102	Leaf
Leaf1	172.20.20.2	10.0.0.13/32	65103	Leaf
Leaf4	172.20.20.6	10.0.0.14/32	65104	Leaf

5.2 Interface Mapping Reference

ContainerLab Link	SONiC Interface	Notes
eth1	ethernet0	First data-plane port
eth2	ethernet4	Offset +4 per port
eth3	ethernet8	Offset +8
eth4	ethernet12	Offset +12

Component	Description	Relevance in VS
CONFIG_DB	Redis based central config store	All interface and BGP config lives here
FRRouting(FRR)	BGP, OSPF, IS-IS routing suite	Handles all routing protocol processing
bgpd	BGP daemon within FRR	Must be enabled manually in VS
vtysh	FRR unified CLI shell	Primary interface for BGP verification
Supervisord	Process supervisor	Manages all SONiC daemons
Syncd	SAI sync daemon	In VS, uses sairedis stub (no real ASIC)
portsyncd	Port state sync	Maps eth interfaces to EthernetN
teamd	LAG/LACP daemon	Port-channel support
Config CLI	SONiC config tool	Wraps CONFIG_DB writes for operators

6. Quick Command Reference

6.1 ContainerLab Lifecycle

Task	Command
Deploy lab	sudo containerlab deploy -t
Redeploy fresh	sudo containerlab deploy -t --reconfigure
Destroy lab	sudo containerlab destroy -t
Destroy + cleanup	sudo containerlab destroy -t --cleanup
Destroy all labs	sudo containerlab destroy --all
List all labs	sudo containerlab inspect --all
Table format output	sudo containerlab inspect --all --format table
JSON output	sudo containerlab inspect --all --format json
Save configs	sudo containerlab save -t
View Topology	sudo containerlab graph -t
Offline Topology HTML	sudo containerlab graph -t --offline
Version Info	containerlab version

6.2 Docker & Node Operations

Task	Command
Enter node shell	docker exec -u root -it bash
SSH into node	ssh admin@
Node logs	docker logs
Follow logs live	docker logs -f
Restart node	docker restart
Stop node	docker stop
Start node	docker start
Copy file to node	docker cp :
Copy file from node	docker cp :
Check clab network	docker network ls | grep cLab
Check veth links	ip link show | grep clab
Container resources	docker stats $(docker ps --format "{{.Names}}" | grep clab)

6.3 SONiC VS Inside-Node Commands

Task	Command
Check all services	supervisorctl status
Enable bgpd	sed -i 's/bgpd=no/bgpd=yes/' /etc/frr/daemons
Start bgpd	supervisorctl start bgpd
Bring up interface	config interface startup Ethernet0
Add IP to interface	config interface ip add Ethernet0 192.168.1.0/31
Add loopback IP	config interface ip add Loopback0 10.0.0.1/32
Show interface status	show interfaces status
Show IP interfaces	show ip interfaces
Enter vtysh	vtysh
BGP summary	show bgp summary
BGP neighbors detail	show bgp neighbors
BGP routing table	show ip route bgp
Show all BGP routes	show bgp ipv4 unicast
Save FRR config	write memory (inside vtysh)

7. Who Should Use This Stack & Why

ContainerLab + SONiC VS is relevant across a wide range of engineering roles and use cases. The following breakdown explains who benefits most and what specific problems the stack solves for each group.

7.1 Network Engineer

> Validate BGP policy changes, ACL rule sets, and interface configs in an exact replica of your production topology before a single change window. --reconfigure gives you a clean-slate environment in seconds.

- Test complex BGP route-map and community policies without risk to production peers.
- Reproduce specific failure scenarios (link down, node failure) and verify convergence behaviour.
- Validate firmware/image upgrades on VS before rolling to hardware.
- Build topology-as-code: your YAML file and saved configs live in Git alongside your other IaC.

7.2 Network Automation & DevOps Teams

> Run Ansible playbooks, Nornir scripts, or NAPALM operations against a real SONiC NOS in a disposable, CI/CD-friendly environment. Every test is reproducible and self-contained.

- Integrate ContainerLab into GitHub Actions or GitLab CI to test automation before merge.
- Use the JSON inspection output for dynamic inventory generation in Ansible/Nornir.
- Test Python scripts that interact with config_db or the SONiC management REST API.
- Validate configuration templates (Jinja2, etc.) render correctly and push cleanly.

7.2 Network Automation & DevOps Teams

> Learn modern data-center networking — eBGP spine-leaf, EVPN/VXLAN, SONiC architecture — without access to physical hardware or expensive vendor licences

- Hands-on learning of FRR/vtysh commands in a safe, disposable environment.
- Explore SONiC's config_db architecture to understand how cloud-scale NOS design works.
- Practice CCIE/CCDE-level data-center design patterns on a Laptop or Virtual Machine.