

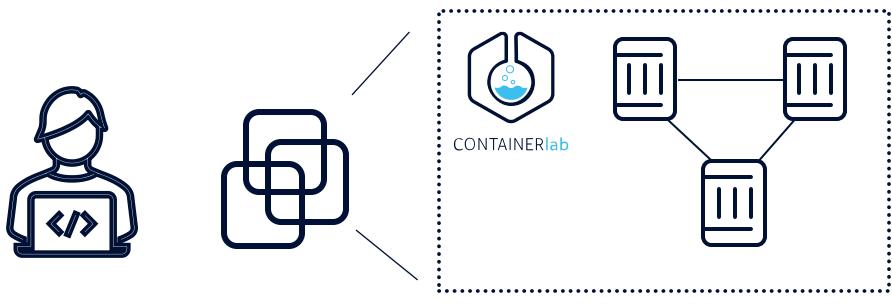
Containerlab Workshop

Reda Laichi

Saju Salahudeen

The Nokia logo, featuring the word "NOKIA" in its signature white sans-serif font, positioned on a large white diagonal arrow pointing from the bottom-left towards the top-right.

Agenda



1. Install containerlab
2. Container-based NOS lab
3. Startup configs
4. Working with Container registry
5. VM-based labs
6. Traffic capture
7. Larger and feature-rich labs

Containerlab

“Lab as code” way to deploy networking labs

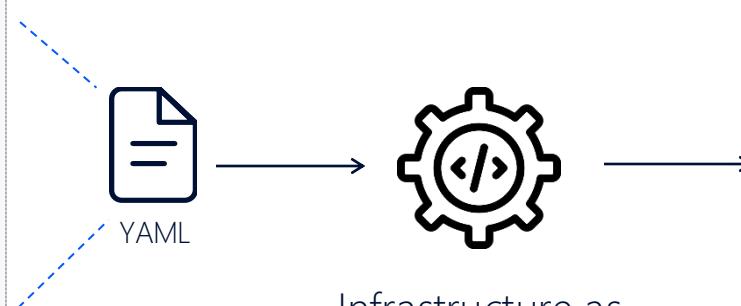


200K	100+
Installations	Contributors
30	24
Supported NOSes	Releases in 12mo

How do application teams deploy infrastructure?

Declarative approach

```
api: v1  
  
services:  
  backend:  
    - [REDACTED]  
    - [REDACTED]  
  
  frontend:  
    - [REDACTED]
```

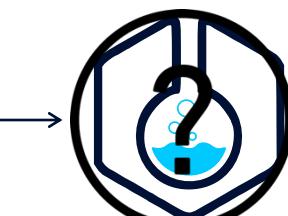


Infrastructure as
Code
tool

Lab as Code

Declarative labs as code with containers

```
name: my-lab  
  
topology:  
  nodes:  
    - :  
    - :  
  
  links:  
    - :  
      - :
```



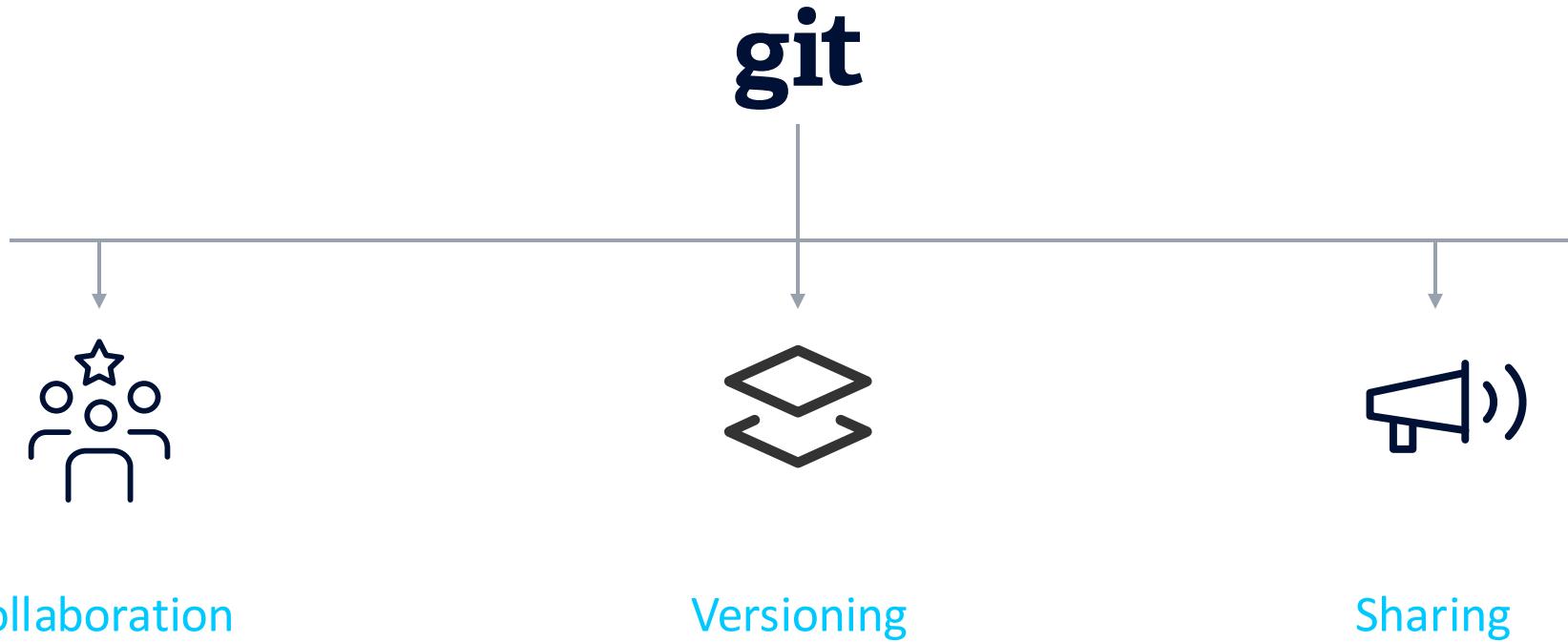
CONTAINERlab

Lab as Code
tool



containerlab.dev

Why “Lab as code”?



They use Containerlab



ARISTA

JUNIPER
NETWORKS

Hewlett Packard
Enterprise

Google

EQUINIX

redhat

Apple

The New York Times

cilium

netbox

NANOG™

PACKETFABRIC

Proton

>>> network .toCode()

KEYSIGHT

OVHcloud

linx

telenet

DE CIX

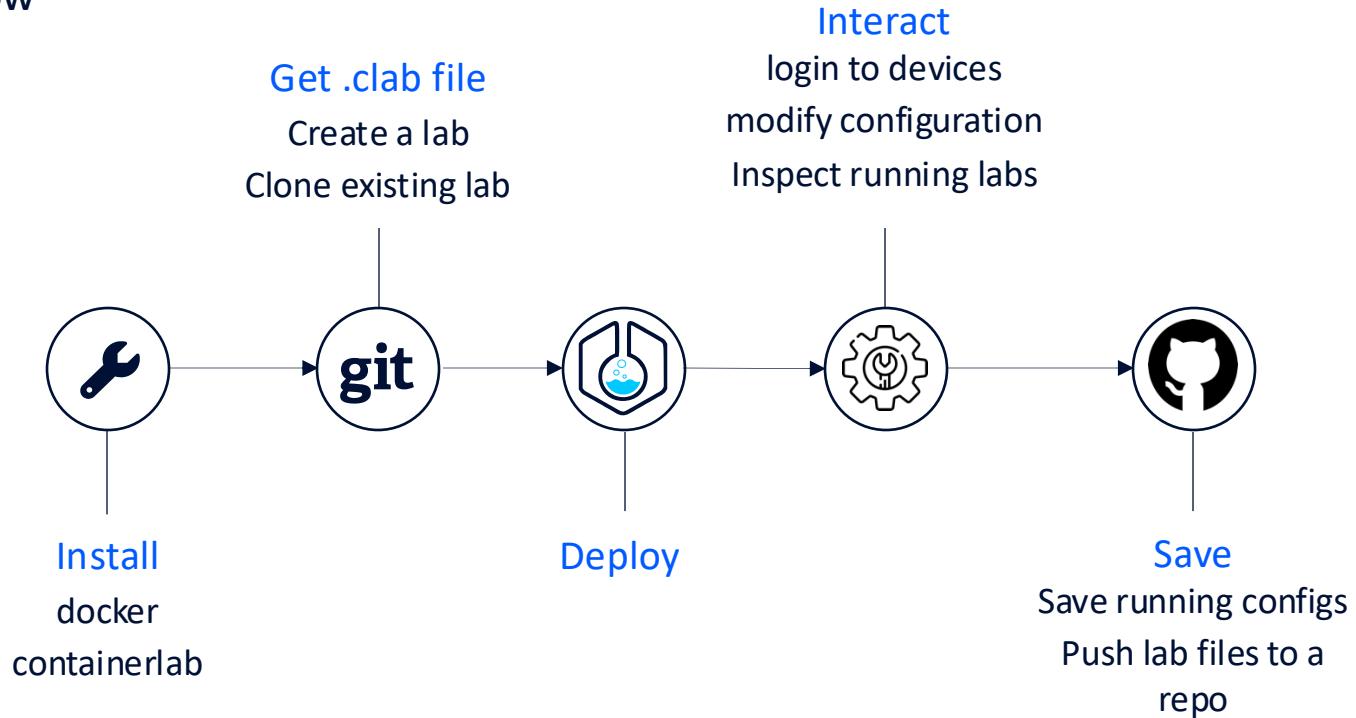
orange™

cloud

NOKIA

Containerlab

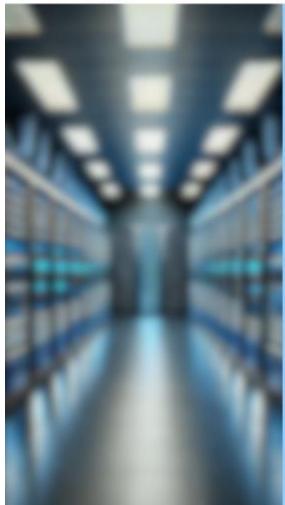
The Workflow



Your personal hands-on experience

And how to get the most out of it

INTERNET2 CONTAINERLAB WORKSHOP



SSID: Internet2
Password: eduroam4all
Hostname: vm .srexperts.net
Username: user
Password: i2ClabW\$



<https://github.com/srlinuxamericas/i2-clab>



u: admin
p: NokiaSrl1!



u: admin
p: admin



u: clab
p: clab@123



NOKIA

INTERNET2 CONTAINERLAB WORKSHOP



Got Questions?

Visit us at Booth A



CONTAINERLAB
DISCORD



SR LINUX
DISCORD

Instructors

Reda Laichi
reda.laichi@nokia.com

Saju Salahudeen
saju.salahudeen@nokia.com

Lab Code of Conduct

- Infra is precious; please use your assigned VM only.
- Please use the VM as instructed— no outside-scope activities.
- By looking at this slide, you accept these simple rules.



Workshop repository

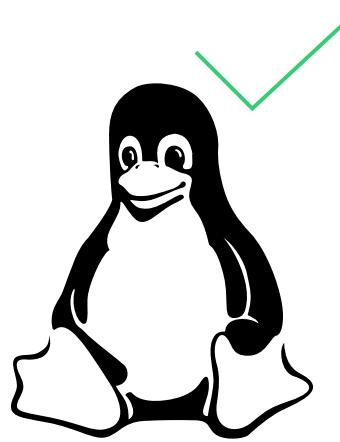
Repo

github.com/srlinuxamericas/i2-clab

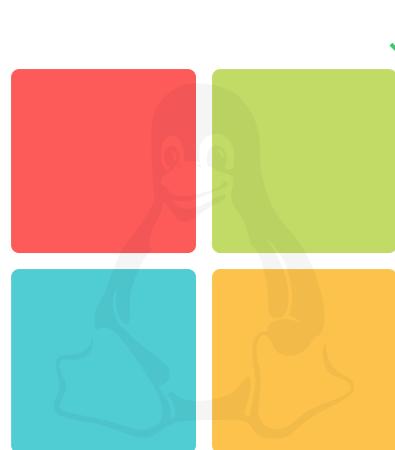
Installing containerlab



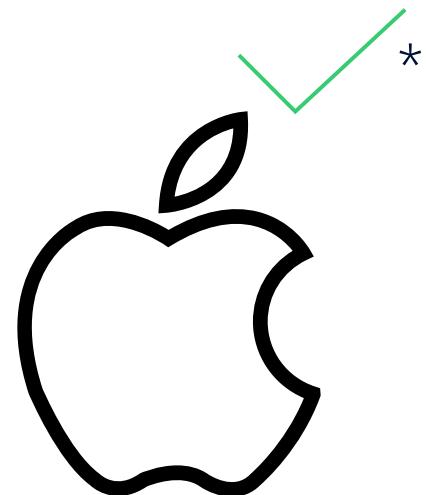
<https://containerlab.dev/install/>



LINUX



WSL2



MacOS

All In One installer

The fastest way to get started

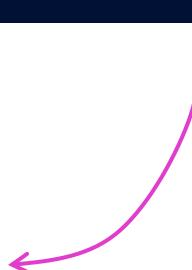


05-install

```
[~] → curl -L http://containerlab.dev/setup | \
    sudo bash -s "all"
```

Installs:

- docker-ce
- containerlab
- GitHub CLI



Log out and Log in
for group changes to take effect

Installing containerlab



05-install

Verify install

● ● ●

```
[~]
└──> docker run --rm hello-world
# Expected output: Hello from Docker!
```

```
[*]--[<ID>]--[~]
└──> containerlab version
```



```
version: 0.59.0
```

Installing containerlab

Other installation options



05-install

- Containerlab-only installation script
- APT/YUM/DNF package managers
- Binary via GitHub releases
- Container image with Containerlab



<https://containerlab.dev/install/>

Containerlab node types

Containerized Network OSes

- Sourced by the vendor
- Fast to spin up
- Small footprint
- Shareability and versioning

Current trend is to **move away**
from **VM** packaging towards
containers
for new NOses

NOKIA
SR Linux

JUNIPER
NETWORKS

cRPD

ARISTA

cEOS

 **CISCO**

XRd


NVIDIA

cVX


KEYSIGHT
TECHNOLOGIES

IXIA-c


FRR

and others...

Containerlab node types

Regular container images

- All available container images
- Emulating clients
- Hundreds of network-focused software
 - Telemetry, logging stacks
 - Peering software
 - Flow collectors
 - etc



Get / Set / Subscribe / Collect



NOKIA



10-basics

Basic lab

Quickstart lab

A simple starting point



Topology definition

```
name: basic

topology:
  nodes:

    srl:
      kind: nokia_srlinux
      image: ghcr.io/nokia/srlinux

    xrd:
      kind: cisco_xrd
      image: xrd:7.8.1
    links:
      - endpoints: [srl:e1-1, xrd:Gi0-0-0-0]
```

Logical view



Quickstart lab

Cloning the workshop repo



10-basics



```
cd ~ && git clone https://github.com/srlinuxamericas/i2-clab.git \
&& cd i2-clab/10-basics
```

- Expected output:

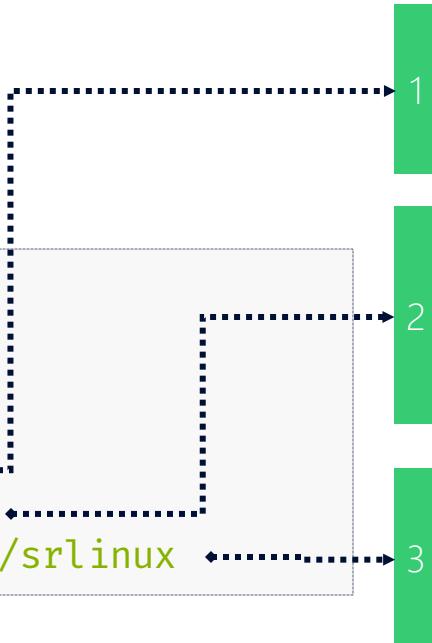


```
[~/i2-clab/10-basics]
└─> cat basic.clab.yml
```

Containerlab Topology File

Basic node definition

```
name: basic  
  
topology:  
  nodes:  
    srl:  
      kind: nokia_srlinux  
      image: ghcr.io/nokia/srlinux
```



1 Node definition container.
Container name will be the node name.
[Read more](#)

2 Kinds define the flavour of the node,
it says if the node is a specific containerized
Network OS or something else.
[Read more](#)

3 Image specifies container image to use for this
node.
[Read more](#)



[topology definition file](#)

Containerlab Topology File

Links definition

Topology definition

```
name: basic

topology:
  nodes:
    srl: <img alt="Node srl icon" data-bbox="113 250 150 280"/>
    xrd: <img alt="Node xrd icon" data-bbox="113 380 150 410"/>
  links:
    - endpoints: [srl:e1-1, xrd:Gi0-0-0-0]
```

Logical view



Containerlab Topology File

Bringing nodes and links together

Topology definition

```
name: basic

topology:
  nodes:

    srl:
      kind: nokia_srlinux
      image: ghcr.io/nokia/srlinux

    xrd:
      kind: cisco_xrd
      image: xrd:7.8.1
    links:
      - endpoints: [srl:e1-1, xrd:Gi0-0-0-0]
```

Logical view



Quickstart

Deploying the lab



```
[~/i2-clab/10-basics]  
└─> sudo containerlab deploy -t basic.clab.yml
```



Containerlab pulls container images specified in the topology file



Error response from daemon: pull access denied for **xrd**,
repository does not exist or may require 'docker login': denied:
requested access to the resource is denied

Local container image store

Topology definition	
<pre>srl: kind: nokia_srlinux image: ghcr.io/nokia/srlinux</pre>	<pre>xrd: kind: cisco_xrd image: xrd:7.8.1</pre>

```
Local image store [ ]  
● ● ●  
[~/i2-clab/10-basics]  
└─> docker images  
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE  
hello-world        latest    d2c94e258dcb  11 months ago  13.3kB
```

Container image notation

```
image: ghcr.io/nokia/srlinux
```

1

Fully qualified name:
`ghcr.io` – registry name
`nokia` – org name
`srlinux` – repo name
`latest` – implicit repo tag

```
image: ceos:4.33.0F
```

2

Fully qualified name:
`docker.io` – implied registry name
`library` – implied org name
`ceos` – repo name
`4.33.0F` – repo tag

```
image: prom/prometheus:v2.47
```

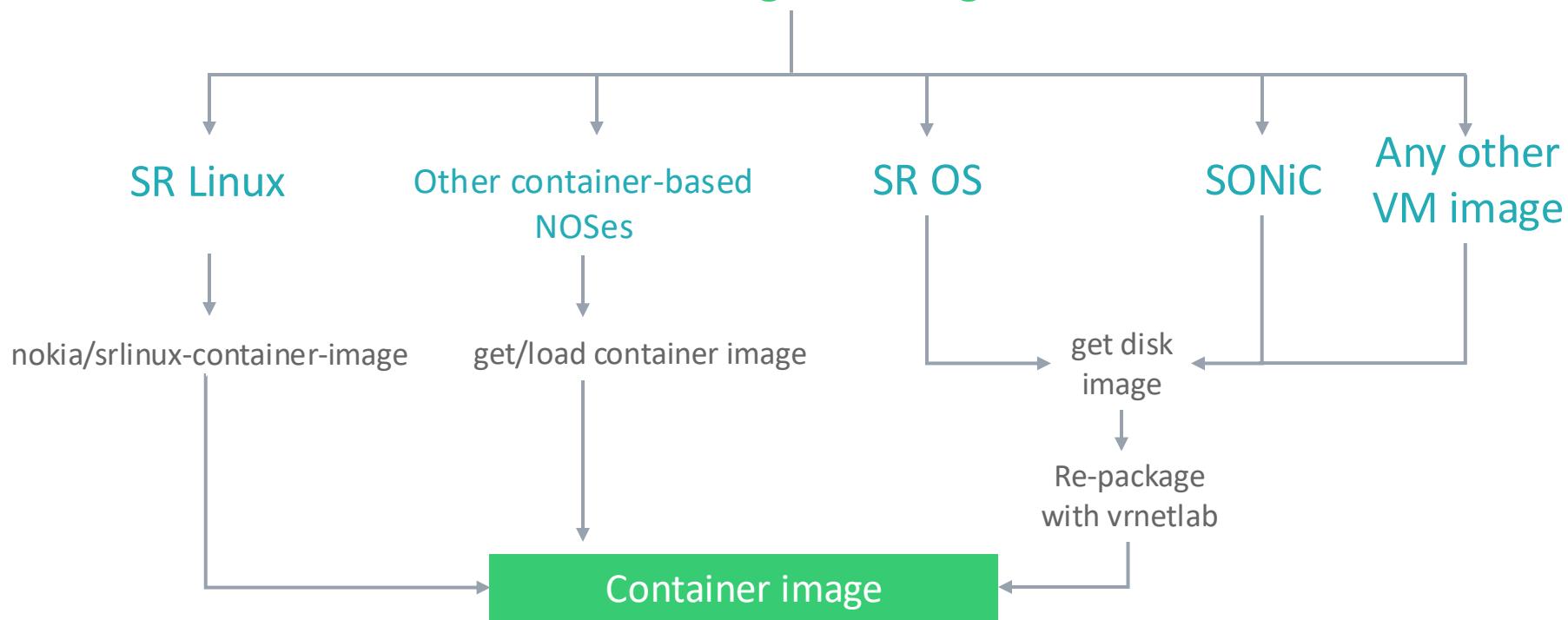
2

Fully qualified name:
`docker.io` – implied registry name
`prom` – implied org name
`prometheus` – repo name
`v2.47` – repo tag

Container images

Where do I get one?

How do I get an image?



Pulling the public image



10-basics

SR Linux container image



[~/i2-clab/10-basics]

└─> **docker pull ghcr.io/nokia/srlinux**

Using default tag: latest

latest: Pulling from nokia/srlinux

1411e7dd712e: Downloading [=====] 312.7MB/874.8MB

Add :<version> at the
end to pull a specific
version



[~/i2-clab/10-basics]

└─> **docker images**

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ghcr.io/nokia/srlinux	latest	36427a3c297e	3 weeks ago	3.03GB
hello-world	latest	d2c94e258dcb	11 months ago	13.3kB

Importing Cisco XRd image



10-basics



containerlab.dev/manual/kinds/xrd/



```
[~/i2-clab/10-basics]
└─> docker load -i ~/images/xrd-7.8.1.tar.xz
--silence for some time--

Loaded image: registry.srlinux.dev/pub/xrd/xrd-control-plane:7.8.1

[~/i2-clab/10-basics]
└─> docker tag registry.srlinux.dev/pub/xrd/xrd-control-plane:7.8.1
xrd:7.8.1
```

Checking local image store



10-basics

Topology definition

```
srl:  
  kind: nokia_srlinux  
  image: ghcr.io/nokia/srlinux
```

```
xrd:  
  kind: cisco_xrd  
  image: xrd:7.8.1
```

Local image store



```
[~/i2-clab/10-basics]  
└──> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
xrd	7.8.1	abc71bccbc4f	5 seconds ago
ghcr.io/nokia/srlinux	latest	36427a3c297e	3 weeks ago

Quickstart lab

Retry deploying the lab



10-basics



[~/i2-clab/10-basics]

└─> **sudo containerlab deploy -t basic.clab.yml**

INFO[0031] Adding ssh config for containerlab nodes

#	Name	Container ID	Image	Kind	State	IPv4 Address	IPv6 Address
1	clab-basic-srl	ab12f9fe0c51	ghcr.io/nokia/srlinux	nokia_srlinux	running	172.20.20.2/24	3fff:172:20:20::2/64
2	clab-basic-xrd	a37ed24a2e4e	xrd:7.8.1	cisco_xrd	running	172.20.20.3/24	3fff:172:20:20::3/64

Quickstart lab



Connecting to the nodes

#	Name	Kind	IPv4 Address	IPv6 Address
1	clab-basic-xrd	cisco_xrd	172.20.20.3/24	2001:172:20:20::3/64
2	clab-basic-srl	nokia_srlinux	172.20.20.2/24	2001:172:20:20::2/64



```
● ● ●  
ssh clab-basic-srl
```



```
● ● ●  
ssh admin@172.20.20.3
```

Default node configuration

- Management interfaces enabled
gNMI, snmp, Netconf, REST API
- TLS certificates created*
- LLDP enabled*
- SSH Keys*
- Documented in the kind documentation

*for some nodes

Node configuration

SR Linux uses a `/etc/opt/srlinux/config.json` file to persist its configuration. By default, containerlab starts nodes of `srl` kind with a basic "default" config, and with the `startup-config` parameter, it is possible to provide a custom config file that will be used as a startup one.

Default node configuration

When a node is defined without the `startup-config` statement present, containerlab will make **additional configurations** on top of the factory config:

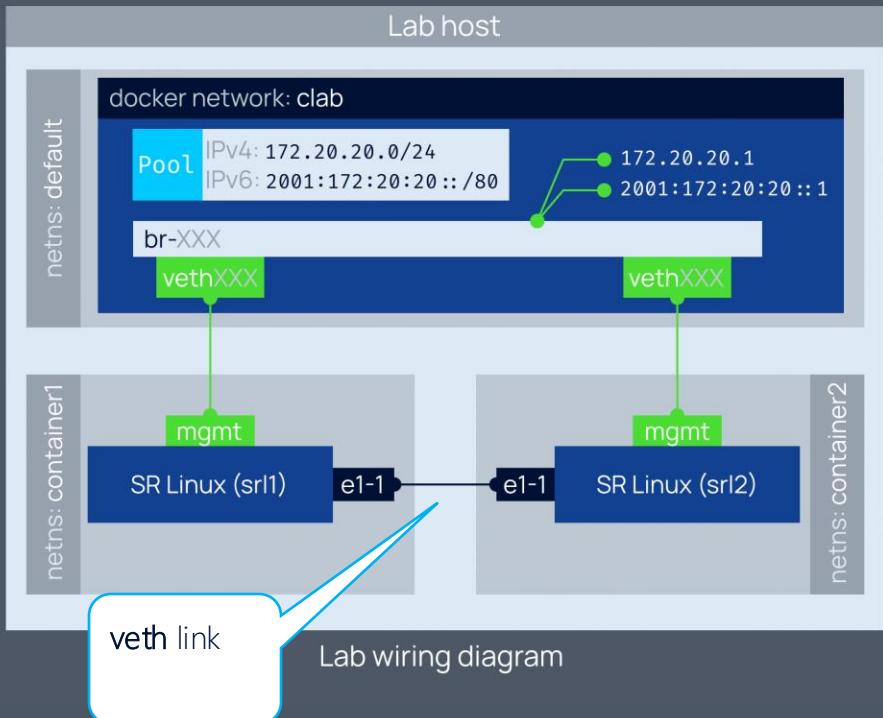
```
# example of a topo file that does not define a custom startup-config
# as a result, the default configuration will be used by this node
```

```
name: srl_lab
topology:
  nodes:
    srl1:
      kind: nokia_srlinux
      type: ixrd3
```



containerlab.dev/manual/kinds/srl/#node-configuration

Containerlab networking



Listing running labs

Command	Effect
<code>sudo containerlab inspect [--topo <path to clab file>]</code>	List nodes of a lab found in the \${PWD} or by the --topo path.
<code>sudo clab ins [-t <path>]</code>	
<code>sudo containerlab inspect -all</code>	List all deployed labs and their nodes
<code>sudo clab ins -a</code>	



```
sudo clab ins -a
```

Lab directory

a.k.a Persistent Storage



[configuration artifacts](#)

```
> tree -L 3 clab-basic
clab-basic
├── ansible-inventory.yml
├── authorized_keys
└── xrd
    └── xr-storage
        ├── config
        └── log
└── srl
    └── config
        └── config.json
└── topology-data.json
```

Lab directory name: **clab-basic**

fixed prefix
lab name

Lab directory:

- Takes precedence over the startup-config
- Better not be checked into git
- Use startup-config instead of relying on the config in the lab dir
- Use startup-config instead of relying on the lab-directory

Destroying the lab

Command	Effect
<code>sudo containerlab destroy [--topo <path to clab file>]</code>	Removes containers for a given topology. Leaves a lab directory intact
<code>sudo containerlab destroy [--topo <path to clab file>] --cleanup</code>	Removes containers and a lab directory for a given topology.
<code>sudo containerlab destroy --all</code>	Remove containers for all running labs. Keep lab directories.
<code>sudo containerlab destroy --all --cleanup</code>	Remove containers and lab directories for all running labs.



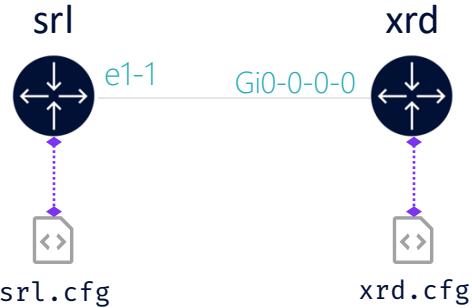
```
[~/i2-clab/10-basics]  
└─> sudo clab des -c
```

Startup configuration

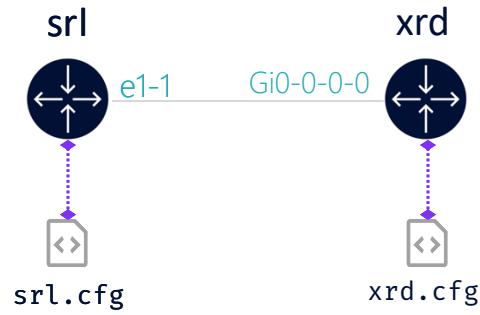
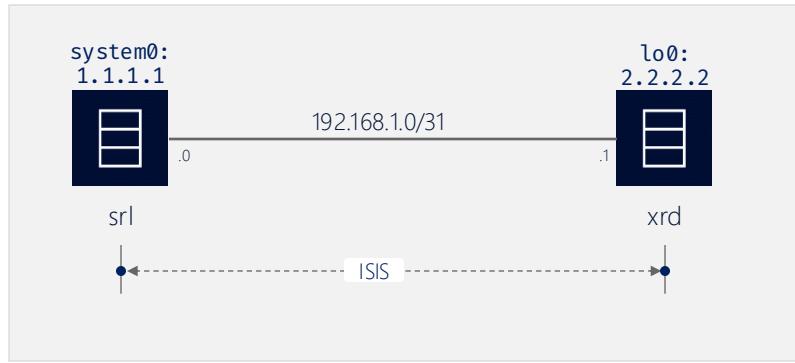
Startup configuration

Startup configuration

- The best way to provide the initial config
- Should be checked into git to keep the lab fully declarative
- Typically contained in an external file
- Provided in the CLI syntax and/or the serialization format of the configuration
- Check the Kind documentation for details



Startup configuration



Providing startup configuration

topology definition

```
name: startup  
  
topology:  
  nodes:  
  
    srl:  
      kind: nokia_srlinux  
      image: ghcr.io/nokia/srlinux  
      startup-config: srl.cfg  
  
    xrd:  
      kind: cisco_xrd  
      image: xrd:7.8.1  
      startup-config: xrd.cfg
```

startup.clab.yml

logical view



What is inside the startup config?

xrd.cfg

srl.cfg

```
interface ethernet-1/1 {  
    subinterface 0 {  
        admin-state enable  
        ipv4 {  
            admin-state enable  
            address 192.168.1.0/31 {  
            }  
        }  
    }  
}  
  
interface lo0 {  
    subinterface 0 {  
        admin-state enable  
        ipv4 {  
            address 1.1.1.1/32 {  
            }  
        }  
    }  
}
```

```
interface Loopback0  
    ipv4 address 2.2.2.2 255.255.255.255  
!  
interface GigabitEthernet0/0/0/0  
    ipv4 address 192.168.1.1 255.255.255.254  
!  
router isis 1  
    net 49.0000.0020.0200.2002.00  
    address-family ipv4 unicast  
!  
interface Loopback0  
    passive  
    address-family ipv4 unicast  
!  
!  
interface GigabitEthernet0/0/0/0  
    point-to-point  
    address-family ipv4 unicast  
!  
!
```

Deploy and see the effect of the startup config



15-startup



```
[~/i2-clab/15-startup]  
└─> sudo clab dep -c
```

1



```
[~/i2-clab/15-startup]  
└─> ssh clab-startup-srl
```

2



```
--{ running }--[ ]--  
A:srl# show network-instance default protocols isis adjacency
```

3

Deploy and see the effect of the startup config



Ping xrd interface address from SR Linux

5

```
● ● ●  
--{ running }--[ ]--  
A:srl# ping 192.168.1.1
```

6

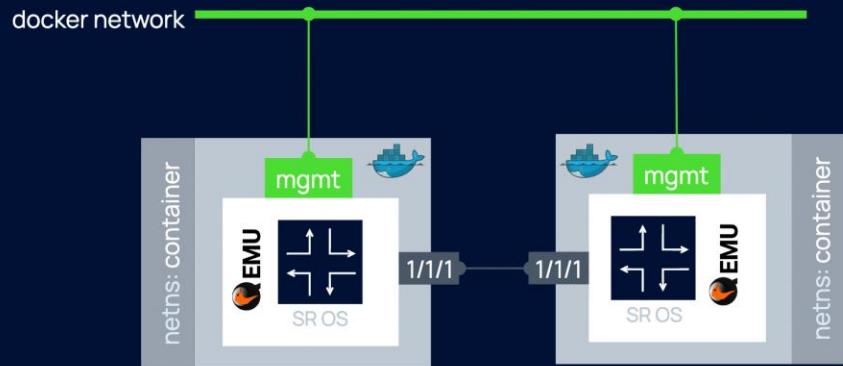
```
● ● ●  
[~/i2-clab/15-startup]  
└─> docker exec -t clab-vm-srl sr_cli 'ping 192.168.1.1 network-instance default -c 3'
```

Where are my
Good old VM-based
Network OSes?

Containerlab node types

Virtual machines in a container package

- Traditional Network OS packaged as a VM
- Integrated with containerlab via vrnetlab open-source project
- Onboard existing VM-based NOSes

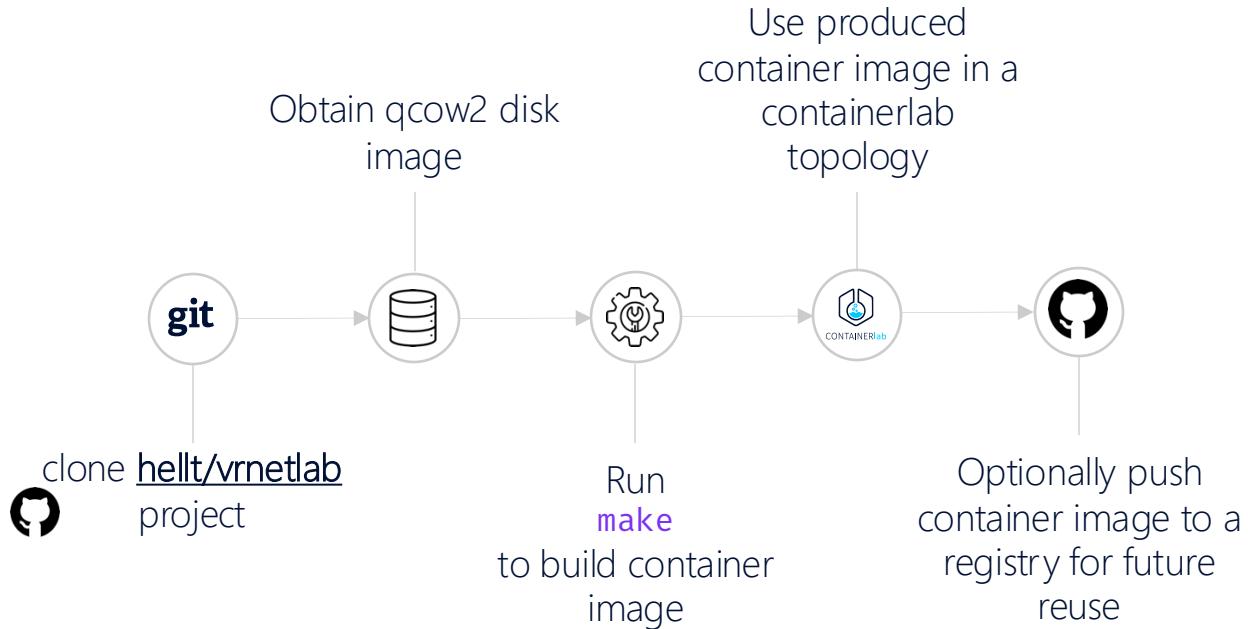


Nokia SR OS
Juniper vMX, vSwitch, EVO
Arista vEOS
Cisco XRv9k, c8000v, NX-OS
Fortinet Fortigate
IPInfusion OcNOS
Palo Alto PAN-OS
Huawei VRP,
And more!

* using hellt/vrnetlab project

Bringing VM-based nodes to Containerlab

The workflow



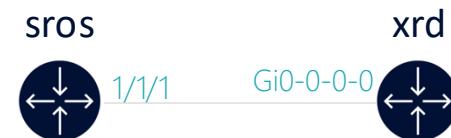
VM-based nodes topology

topology definition

```
name: vm  
  
topology:  
  nodes:  
  
    sros:  
      kind: nokia_sros  
      image: vrnetlab/vr-sros:24.3.R1  
      license: ~/images/sros-v24.lic  
  
    xrd:  
      kind: cisco_xrd  
      image: xrd:7.8.1  
  
links:  
  - endpoints: [sros:1/1/1, xrd:Gi0-0-0-0]
```

vm.clab.yml

logical view



Cloning hellt/vrnetlab



```
[*]-[<ID>]-[~/clab-workshop/15-startup]  
└─> cd ~  
git clone https://github.com/hellt/vrnetlab.git  
cd ~/vrnetlab
```

Building SR OS container image 1/2



```
[*]-[<ID>]-[ ~/vrnetlab]  
└─> cp ~/images/sros-vm-24.7.R1.qcow2 ~/vrnetlab/sros/
```

Building SR OS container image 2/2



20-vm

```
● ● ●
[*]-[<ID>]-[~/vrnetlab]
└─> cd ~/vrnetlab/sros
     make

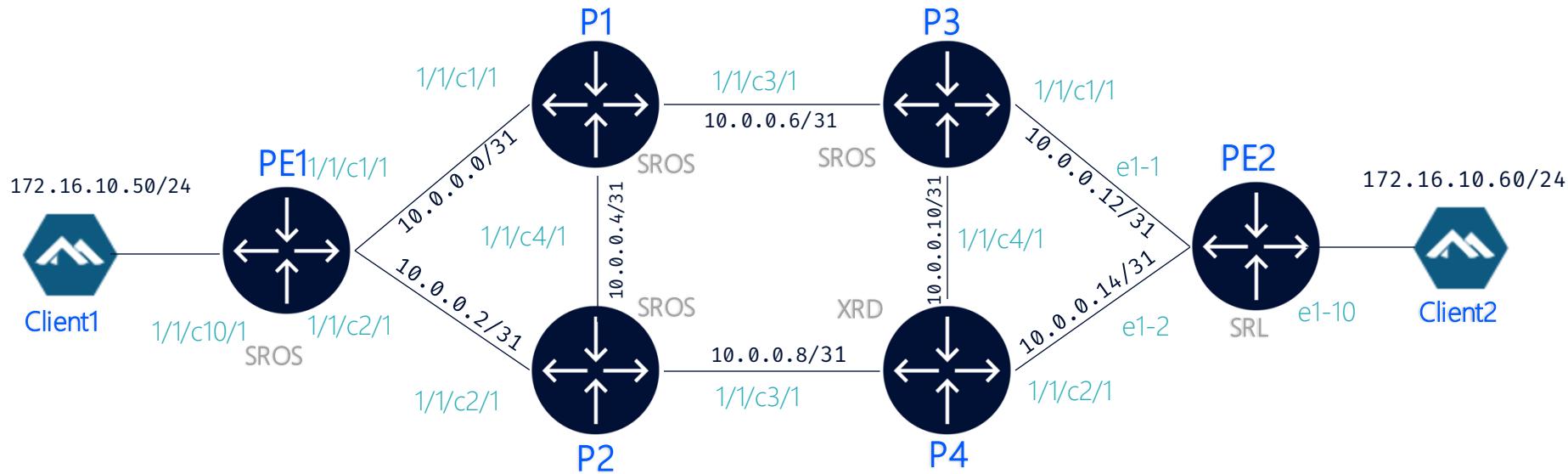
=> => naming to docker.io/vrnetlab/nokia_sros:24.7.R1
```

```
● ● ●
[*]-[<ID>]-[~/vrnetlab]
└─> docker images | grep sros

vrnetlab/nokia_sros    24.7.R1    0a9146ccdd14    19 minutes ago    1.43GB
```

Topology

20-vm



Topology YAML file

```
name: i2-vm-evpn

prefix: ""
topology:
  defaults:
    kind: nokia_sros
  kinds:
    nokia_sros:
      image: vr-sros:24.7.R1
      type: sr-1
      license: /home/user/images/sros-24.lic
    linux:
      image: ghcr.io/srl-labs/network-multitool
  nodes:
    pe1-sr1:
      startup-config: startup/pe1-sr1-startup-config.txt
    pe2-srL:
      kind: nokia_srlinux
      image: ghcr.io/nokia/srlinux:24.10.1
      type: ixrx1b
      license: /home/user/images/srl-24.10.lic
      startup-config: startup/pe2-srL-startup-config.txt
    p1-sr1:
      startup-config: startup/p1-sr1-startup-config.txt
    p2-sr1:
      startup-config: startup/p2-sr1-startup-config.txt
    p3-sr1:
      startup-config: startup/p3-sr1-startup-config.txt
```

```
p4-xrd:
  kind: cisco_xrd
  image: xrd:7.8.1
  startup-config: startup/p4-xrd-startup-config.txt
client1:
  kind: linux
  exec:
    - ip address add 172.16.10.50/24 dev eth1
client2:
  kind: linux
  exec:
    - ip address add 172.16.10.60/24 dev eth1
links:
  - endpoints: ["pe1-sr1:eth1", "p1-sr1:eth1"]
  - endpoints: ["pe1-sr1:eth2", "p2-sr1:eth2"]
  - endpoints: ["p1-sr1:eth4", "p2-sr1:eth4"]
  - endpoints: ["p1-sr1:eth3", "p3-sr1:eth3"]
  - endpoints: ["p2-sr1:eth3", "p4-xrd:Gi0-0-0-3"]
  - endpoints: ["p3-sr1:eth4", "p4-xrd:Gi0-0-0-4"]
  - endpoints: ["pe2-srL:e1-1", "p3-sr1:eth1"]
  - endpoints: ["pe2-srL:e1-2", "p4-xrd:Gi0-0-0-2"]
  - endpoints: ["client1:eth1", "pe1-sr1:eth10"]
  - endpoints: ["client2:eth1", "pe2-srL:e1-10"]
```

Deploying the lab



```
● ● ●  
[*]-[<ID>]-[ ~ ]  
└─> cd ~/clab-workshop/20-vm  
     sudo clab dep -c
```

Monitoring the boot process



20-vm

```
● ● ●  
[*]-[<ID>]-[~]  
└─> docker logs -f pe1-sr1
```

Boot log

- The first thing to check when something “does not work”
- Gives insight into what is provisioned by Containerlab in terms of the base configuration

Connecting to the nodes



#	Name	Kind	IPv4 Address	IPv6 Address
1	pe1-sr1	nokia_sros	172.20.20.14/24	2001:172:20:20::14/64
2	pe2-srL	nokia_srlinux	172.20.20.12/24	2001:172:20:20::12/64

A dark blue rectangular box containing three small colored dots (red, orange, and green) in the top-left corner.

ssh pe2-srL

A dark blue rectangular box containing three small colored dots (red, orange, and green) in the top-left corner.

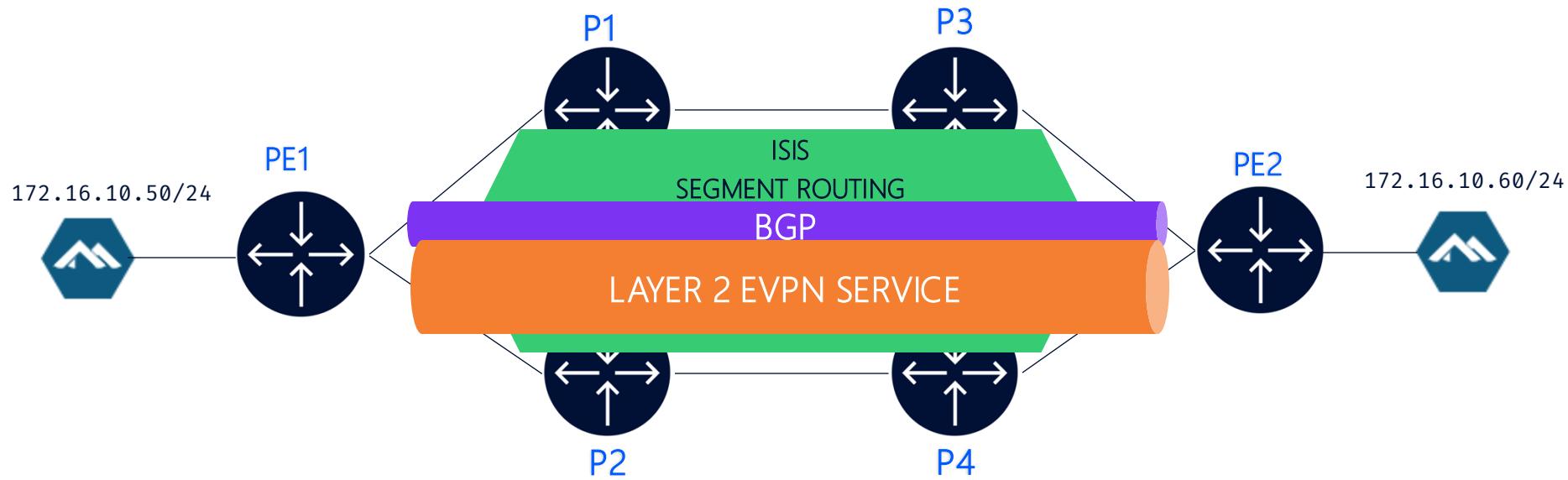
ssh pe1-sr1

Configuration

Topology Configuration

20-vm

All configuration is deployed as part of the startup config file.



Interface Verification

SR OS

```
# show router interface
```

SR Linux

```
# show interface
```

IS-IS Verification

SR OS

```
# show router isis status  
  
# show router isis adjacency
```

SR Linux

```
# show network-instance default protocols isis summary  
  
# show network-instance default protocols isis adjacency
```

Segment Routing Verification

SR OS

```
# show router tunnel-table
```

SR Linux

```
# show network-instance default tunnel-table
```

BGP Verification

SR OS

```
# show router bgp summary  
  
# show router bgp routes evpn incl-mcast
```

SR Linux

```
# show network-instance default protocols bgp summary  
  
# show network-instance default protocols bgp neighbor  
  
# show network-instance default protocols bgp routes evpn route-type summary
```

Service Verification

SR OS

```
# show service id "store1" base  
  
# show service id "store1" evpn-mpls detail
```

SR Linux

```
# show network-instance store1 summary  
  
# show network-instance store1 protocols bgp-evpn bgp-instance 1
```

Service Verification – Ping

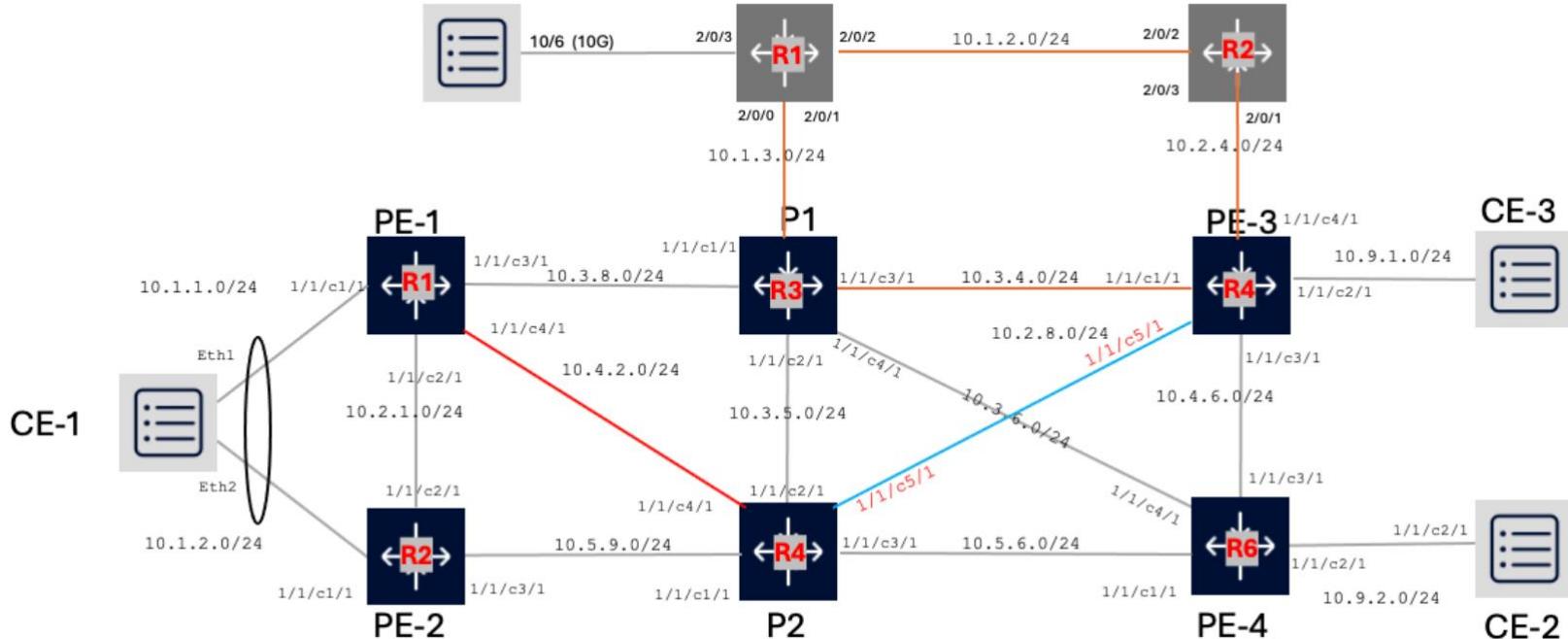
Login to Client1 and ping Client 2

```
# docker exec -it client1 sh  
  
# ping 172.16.10.60
```

Check BGP routes to verify the EVPN Type 2 MAC route

```
# show router bgp routes evpn mac (SROS)  
  
# show network-instance default protocols bgp routes evpn route-type 2 summary (SR Linux)
```

Additional learning - Network for REN

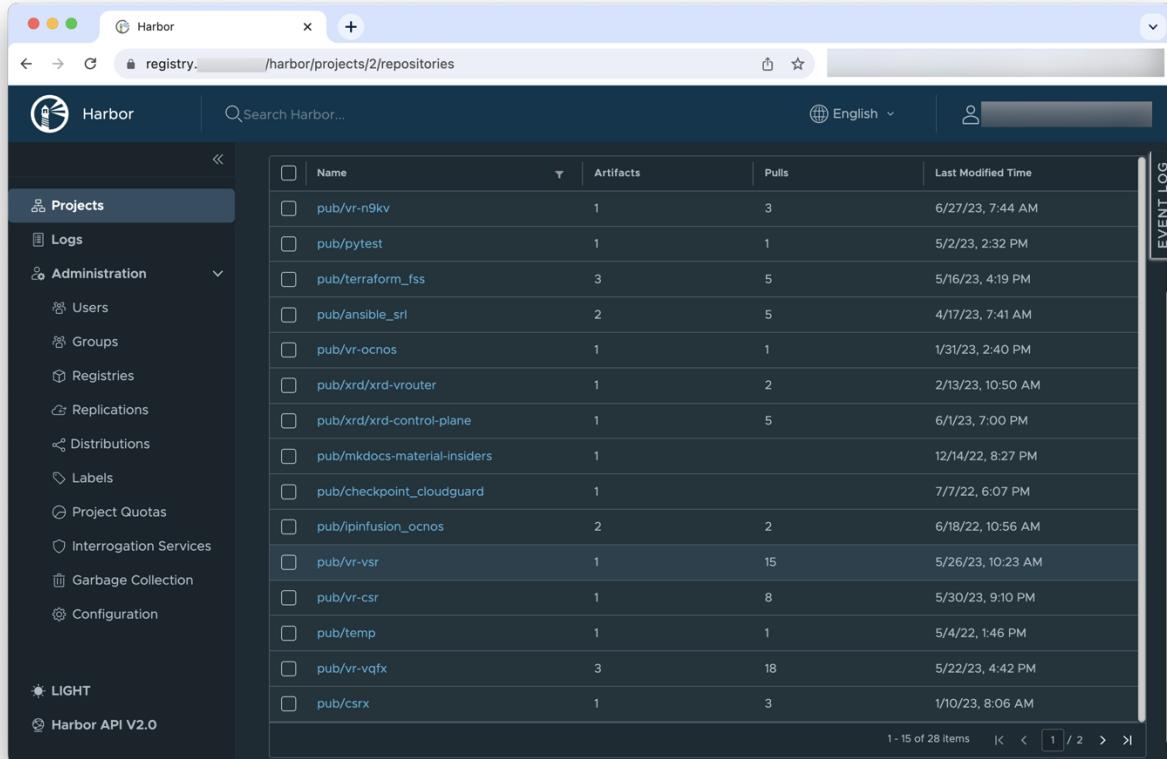


Container registry

Container registry

Taking back control over your images

- Centralized image repository
- Version control (tags, SHA)
- Seamless containerlab integration
- Granular access control



The screenshot shows the Harbor Container Registry web interface. The left sidebar contains navigation links: Projects, Logs, Administration (with sub-links for Users, Groups, Registries, Replications, Distributions, Labels, Project Quotas, Interrogation Services, Garbage Collection, and Configuration), LIGHT mode, and Harbor API V2.0. The main content area displays a table of repositories. The table has columns for Name, Artifacts, Pulls, and Last Modified Time. There are 28 items listed, with the first 15 shown in the main view and the last 13 in a scrollable bottom section. The table includes a header row with sorting icons for Name, Artifacts, Pulls, and Last Modified Time.

Name	Artifacts	Pulls	Last Modified Time
pub/vr-n9kv	1	3	6/27/23, 7:44 AM
pub/pytest	1	1	5/2/23, 2:32 PM
pub/terraform_fss	3	5	5/16/23, 4:19 PM
pub/ansible_srl	2	5	4/17/23, 7:41 AM
pub/vr-ocnos	1	1	1/31/23, 2:40 PM
pub/xrd/xrd-vrouter	1	2	2/13/23, 10:50 AM
pub/xrd/xrd-control-plane	1	5	6/1/23, 7:00 PM
pub/mkdocs-material-insiders	1		12/14/22, 8:27 PM
pub/checkpoint_cloudguard	1		7/7/22, 6:07 PM
pub/ipinfusion_ocnos	2	2	6/18/22, 10:56 AM
pub/vr-vsr	1	15	5/26/23, 10:23 AM
pub/vr-csr	1	8	5/30/23, 9:10 PM
pub/temp	1	1	5/4/22, 1:46 PM
pub/vr-vqfx	3	18	5/22/23, 4:42 PM
pub/csrx	1	3	1/10/23, 8:06 AM

Harbor container registry

Web access

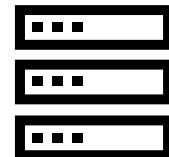


<https://registry-i2.srexperts.net>

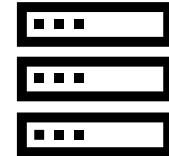
Login: admin

i2ClabW\$

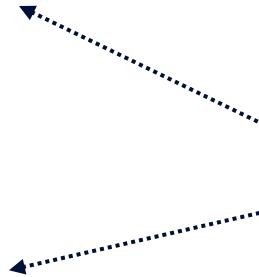
The screenshot shows the Harbor container registry web interface. The top navigation bar includes links for 'Search Harbor...', 'English', 'Default', and a user dropdown for 'admin'. The left sidebar has sections for 'Projects', 'Logs', 'Administration' (with 'Users' selected), 'Robot Accounts', 'Registries', 'Replications', 'Distributions', 'Labels', 'Project Quotas', 'Interrogation Services', 'Clean Up', 'Job Service Dashboard', and 'Configuration'. The main content area displays a summary of projects and repositories, showing 0 private and 1 public project, and 0 private and 2 public repositories. It also shows a quota usage of 1.64 GB. Below this is a table for managing projects, with one entry for 'library' which is public, has a Project Admin role, is a Project type, has 2 repositories, and was created on 4/11/24, 5:55 PM. The table includes columns for 'Project Name', 'Access Level', 'Role', 'Type', 'Repositories Count', and 'Creation Time'. A 'Page size' dropdown is set to 15, and it shows 1-1 of 1 items.



User 10



User XX



registry-i2.srexperts.net

Pushing images to a registry 1/2

1 Login

For reference only !

```
● ● ●  
[ ~ ]> docker login registry-i2.srexperts.net  
user: i2ClabW$
```

2 List images

```
● ● ●  
[*]-[<ID>]-[ ~ ]> docker images
```

REPOSITORY	TAG
vrnetlab/nokia_sros	24.7.R1
ghcr.io/nokia/srlinux	latest
hello-world	

Pushing images to a registry 2/2

REPOSITORY
vrnetlab/nokia_sros

TAG
24.7.R1

For reference only !

3 Tag the image



```
>docker tag vrnetlab/nokia_sros:24.7.R1 registry-i2.srexperts.net/library/nokia_sros:24.7.R1
```

4 Upload images

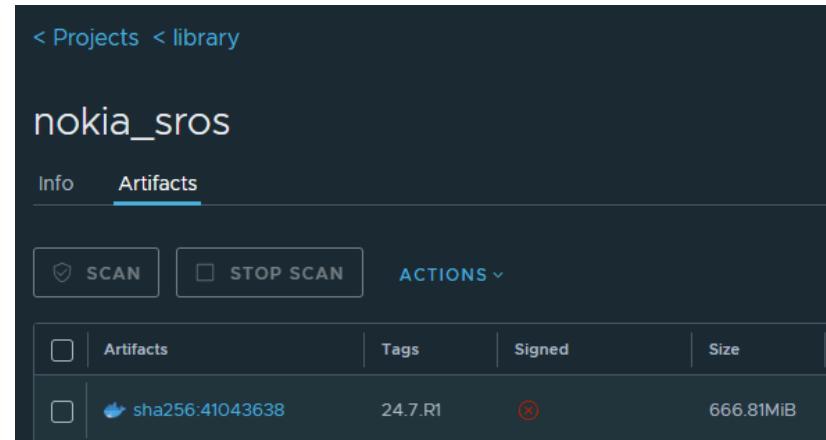


```
>docker push registry-i2.srexperts.net/library/nokia_sros:24.7.R1
```

Expected error:
Permission denied

Using images in a registry (1/2)

1 Use SROS image in the registry



2 Delete SROS image in local docker repo

```
user@vm1:~$ docker images
REPOSITORY
vrnetlab/nokia_sros
registry-i2.srexperts.net/library/nokia_sros
ghcr.io/nokia/srlinux          latest

# Destroy existing labs
user@vm1:~$ sudo clab des -a
```

```
# Delete sros docker image
user@vm1:~$ docker image rm -f b91919a90761
```

TAG	IMAGE ID
24.7.R1	b91919a90761
24.7.R1	b91919a90761
	eb2a823cd8ce

Using images in a registry (2/2)

3 Update 20-vm.clab.yml to pull SROS image from the registry

```
name: vm

prefix: ""
topology:
  defaults:
    kind: nokia_sros
  kinds:
    nokia_sros:
      image: vnetlab/nokia_sros:24.7.R1
      image: registry-i2.srexperts.net/library/nokia_sros:24.7.R1
```

Deploy the lab

Docker will pull the image from the registry during lab deployment

Verify ping between Client 1 and Client 2

Packet capture

Wireshark

Several ways to sniff traffic

1 Local capture

- Via container's shell

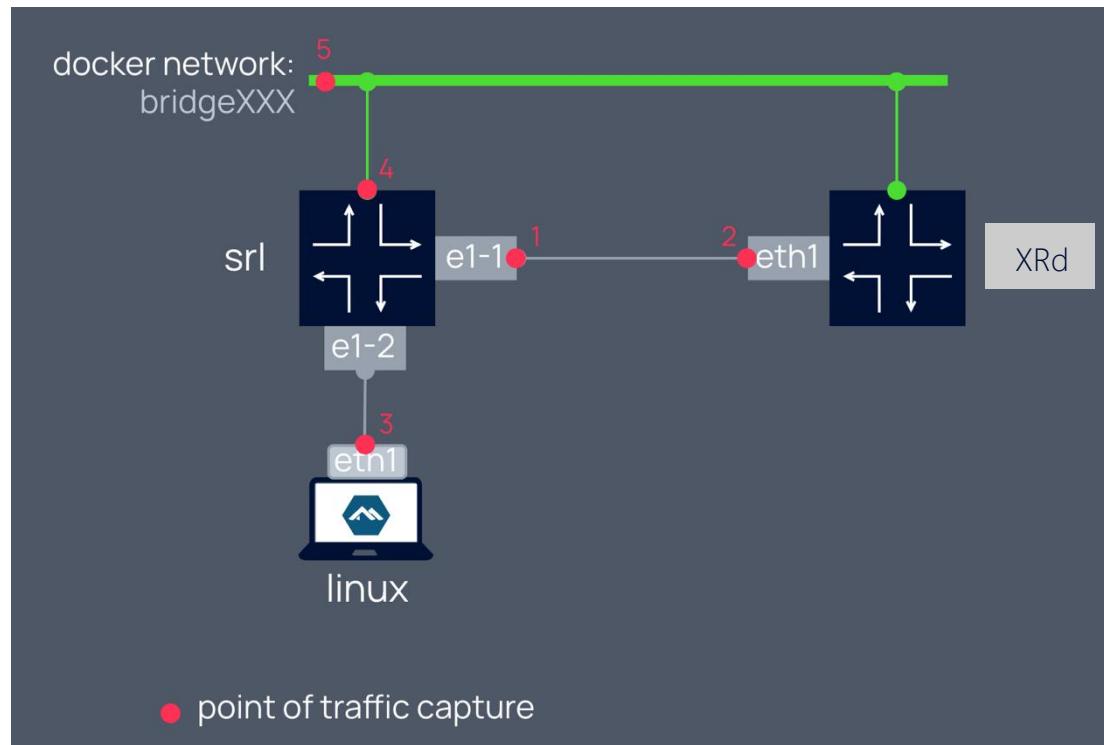
2 Remote capture

- Via ssh and pipes

3 Web UI capture

- Via Edgeshark

This section is using the startup lab (with startup configs). Re-deploy this lab if already destroyed.



containerlab.dev/manual/wireshark

Local capture

From host CLI

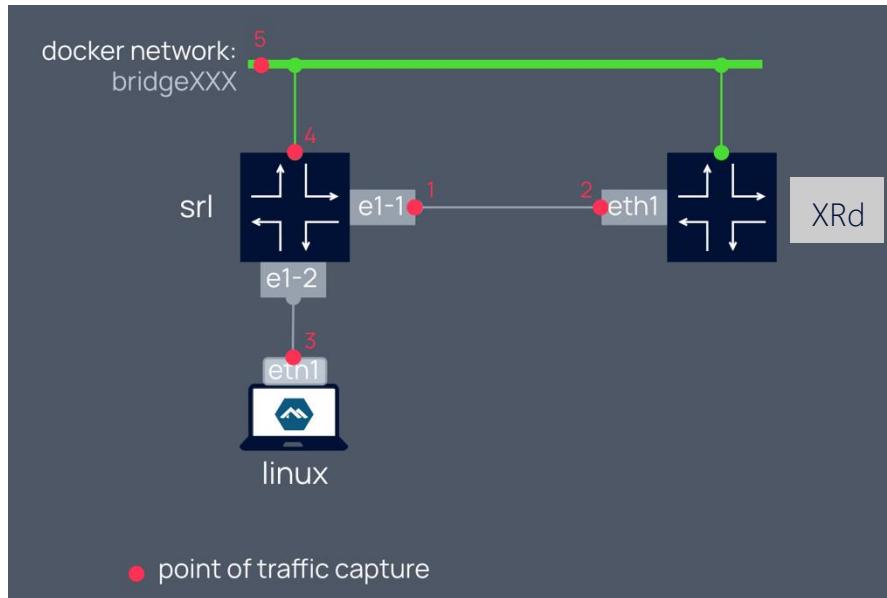
Command to capture at point #1



```
sudo ip netns exec clab-startup-srl tcpdump -nni  
e1-1
```

Notes:

- Output is displayed on screen after capture is stopped



Remote capture

A quick way to use Wireshark for packet capture

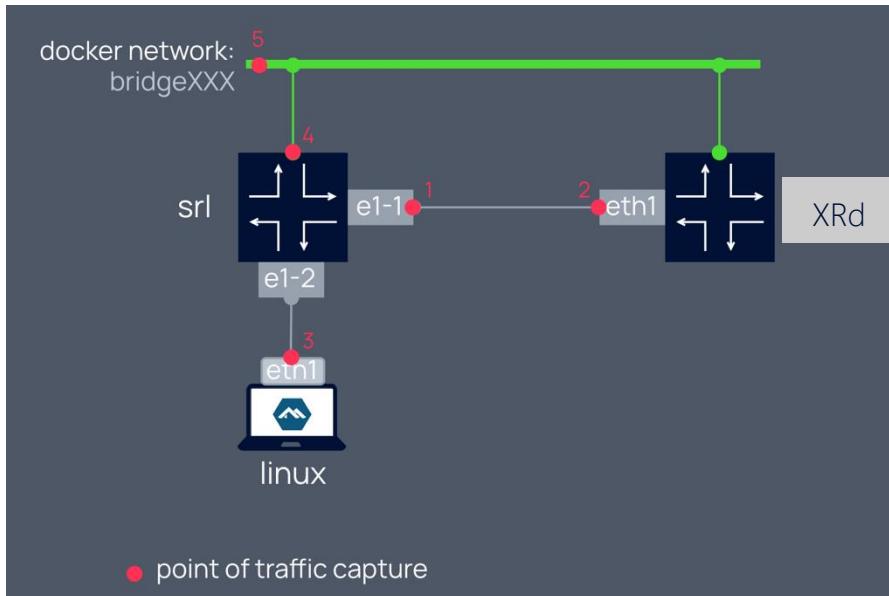
Command to capture at point #1



```
ssh user@$clab_host "sudo ip netns exec srl  
tcpdump -U -nni e1-1 -w -" | wireshark -k -i -
```

Notes:

- Use WSL on windows
 - Wrap this long command in a pcap.sh that takes in the note/interface arguments
 - No extra packages/modifications required



Remote capture



40-packet-capture

A quick way to use Wireshark for packet capture

Start a ping from SR Linux to SONiC.

Capturing traffic from **SR Linux** port **ethernet-1/1** running on host and displaying in **Wireshark**

```
ssh user@vm<X>.srexperts.net \
“sudo ip netns exec clab-startup-srl tcpdump -U -nni e1-1 -w -” | \
/mnt/c/Program\ Files/Wireshark/wireshark.exe -k -i -
```



```
ssh user@vm<X>.srexperts.net \
“sudo ip netns exec clab-startup-srl tcpdump -U -nni e1-1 -w -” | \
/Applications/Wireshark.app/Contents/MacOS/Wireshark -k -i -
```

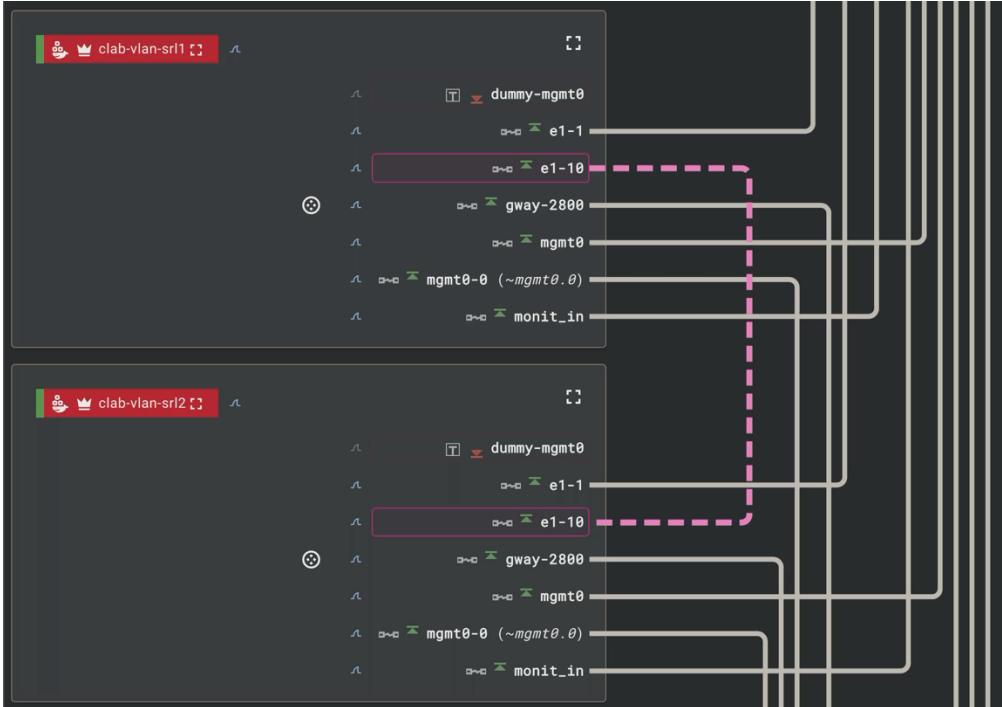


Replace <X> with your ID.

Edgeshark

Web UI for Wireshark

- Web-based, no installation required
- Container-based, no need for dependencies
- Uses native Wireshark
- Open-source
- Free



<https://containerlab.dev/manual/wireshark/#edgeshark-integration>

Deploying Edgeshark



40-packet-capture

1 Deploy Edgeshark service

```
● ● ●  
[~] └─> curl -sL \  
https://github.com/siemens/edgeshark/raw/main/deployments/wget/docker-  
compose.yaml | docker compose -f - up -d
```

Possible to install Wireshark plug-in

See the workshop's readme!

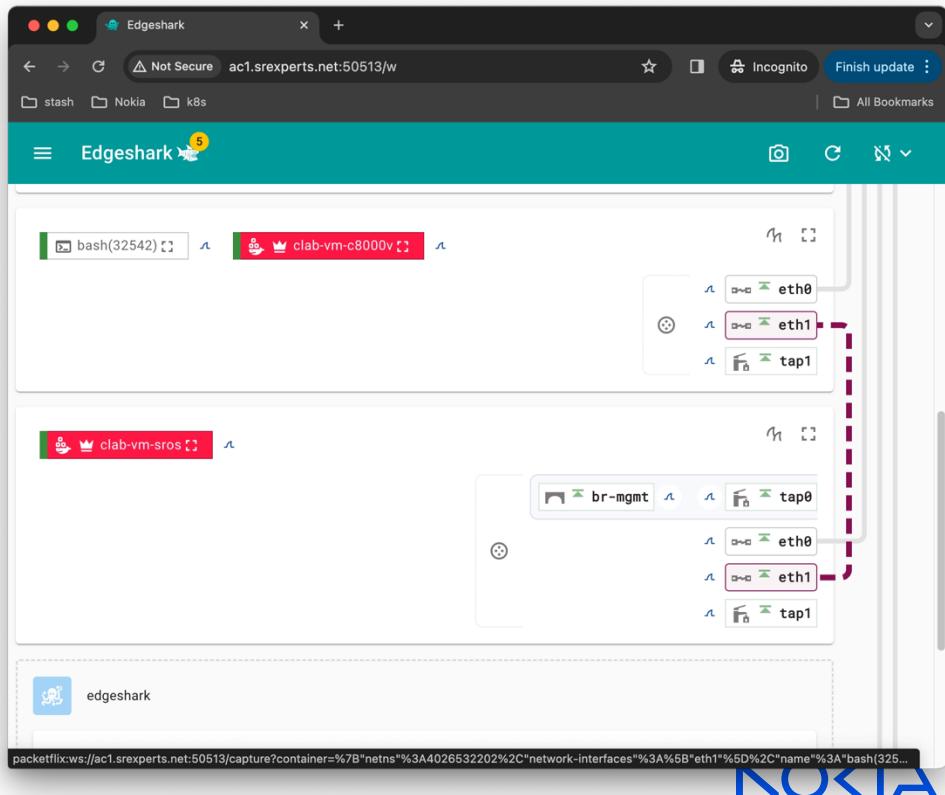
Using Edgeshark



<http://vm<ID>.srexperts.net:5001>

ID: assigned number

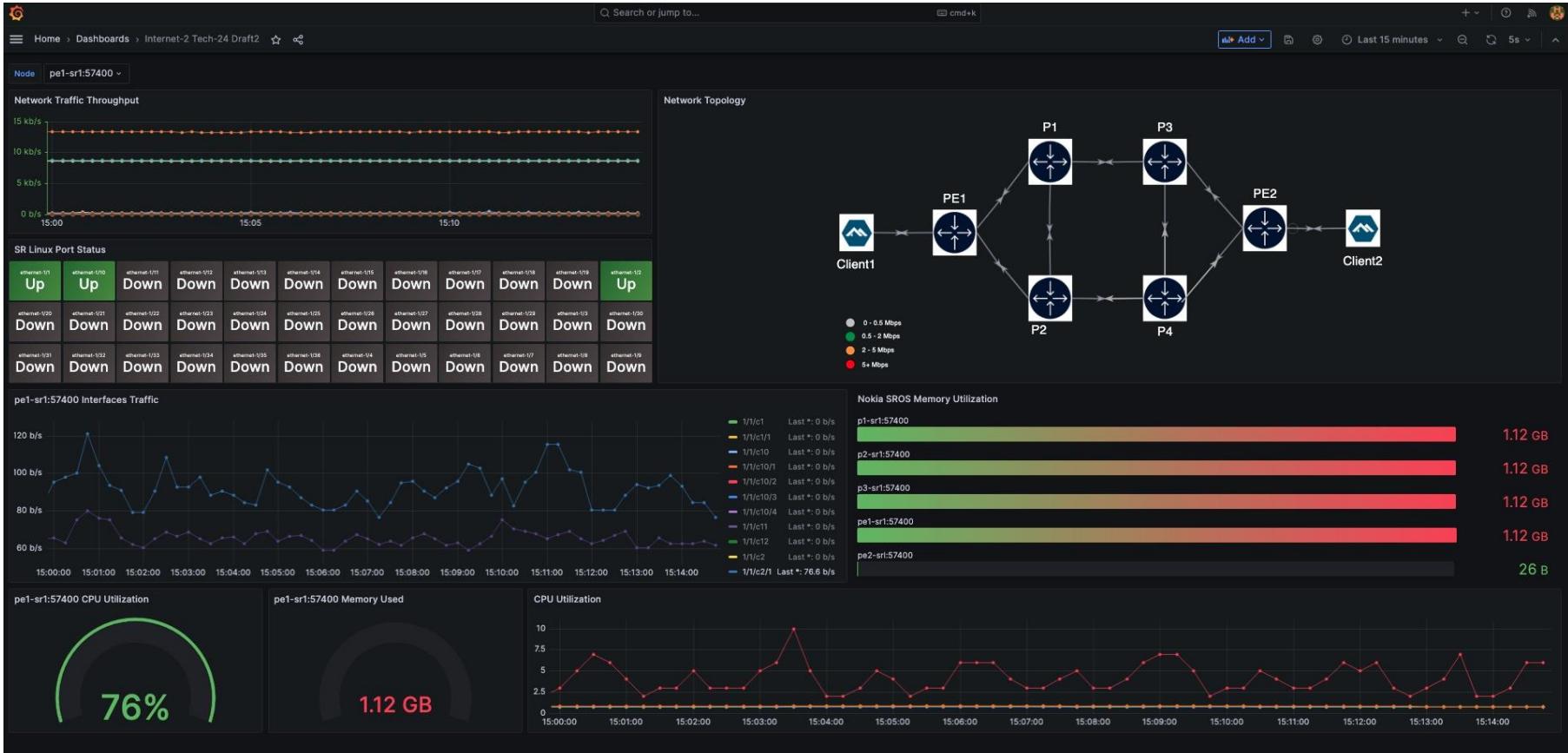
When the page opens, hit  to refresh the list of containers



Network Streaming Telemetry Lab

Deploying Telemetry network

45-streaming-telemetry



Deploying Network Telemetry lab

1 Deploy the telemetry lab

```
 [ cd ~/i2-clab/45-streaming-telemetry ]  
 [ * ]-[ <ID> ]-[ ~ ]  
 [ ]-> sudo containerlab deploy -t st_i2_45.clab.yaml
```

Deploying Network Telemetry lab

2 Verify the deployment



[*]-[<ID>]-[~]

└─> sudo containerlab inspect -all

#	Name	Container ID	Image	Kind	State	IPv4 Address	IPv6 Address
1	client1	3adad1da3513	ghcr.io/srl-labs/network-multitool	linux	running	172.20.20.9/24	3fff:172:20:20::9/
2	client2	68fbb8e9620d	ghcr.io/srl-labs/network-multitool	linux	running	172.20.20.11/24	3fff:172:20:20::b/
3	gnmic	041d82a39554	ghcr.io/openconfig/gnmic:0.30.0	linux	running	172.20.20.4/24	3fff:172:20:20::4/
4	grafana	32959b842da7	grafana/grafana:9.5.2	linux	running	172.20.20.7/24	3fff:172:20:20::7/
5	p1-sr1	448afb1c5ae8	vr-sros:24.7.R1	nokia_sros	running	172.20.20.12/24	3fff:172:20:20::c/
6	p2-sr1	13d26231dd65	vr-sros:24.7.R1	nokia_sros	running	172.20.20.8/24	3fff:172:20:20::8/
7	p3-sr1	d4f14da193a0	vr-sros:24.7.R1	nokia_sros	running	172.20.20.3/24	3fff:172:20:20::3/
8	p4-xrd	4d3739485e54	registry-i2.srexperts.net/library/xrd-control-plane:7.8.1	cisco_xrd	running	172.20.20.2/24	3fff:172:20:20::2/
9	p1-sr1	0d071af95e6a	vr-sros:24.7.R1	nokia_sros	running	172.20.20.10/24	3fff:172:20:20::a/
10	pe2-srL	9fcfed33a67c7	ghcr.io/nokia/srlinux:24.10.1	nokia_srlinux	running	172.20.20.5/24	3fff:172:20:20::5/
11	prometheus	e6edc8eb73f	prom/prometheus:v2.37.8	linux	running	172.20.20.6/24	3fff:172:20:20::6/

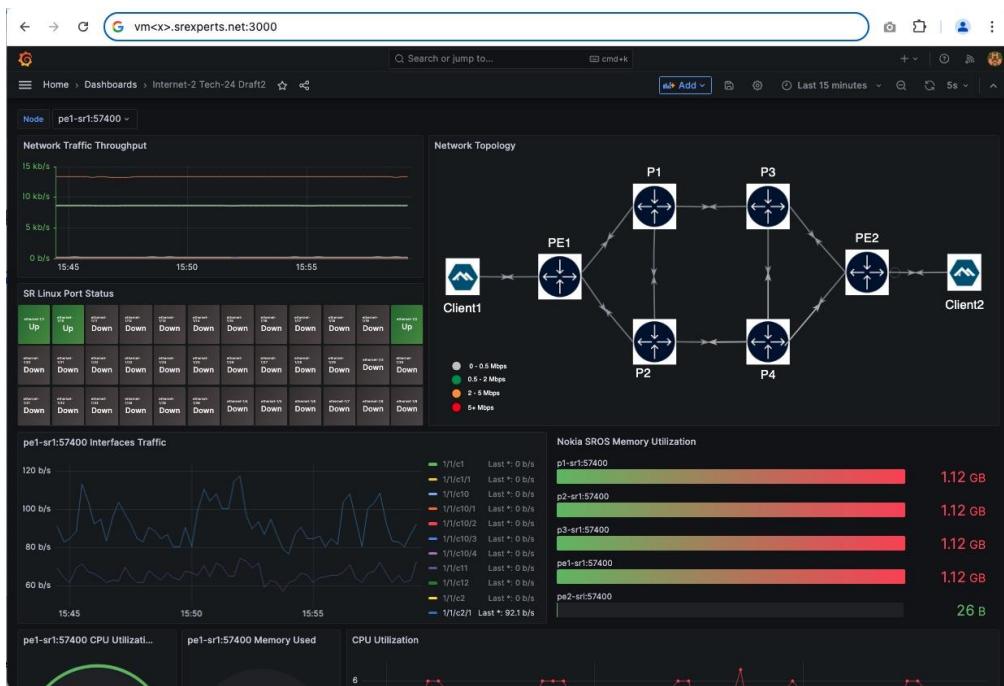
Deploying Network Telemetry lab

Web access



<http://vm<x>.srexper...> 3000

Login: admin / admin



Deploying Network Telemetry lab

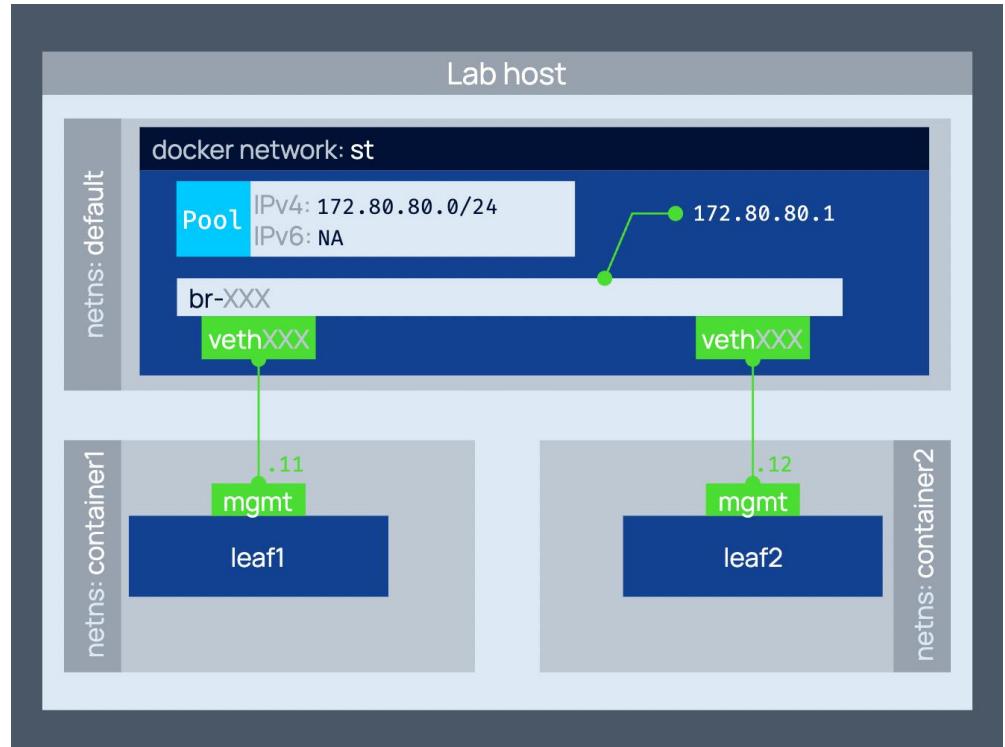
4 Start traffic

```
● ● ●  
[*]-[<ID>]-[~]  
└─> sudo ./traffic.sh start all
```

Management network

Statically assigned IP addresses

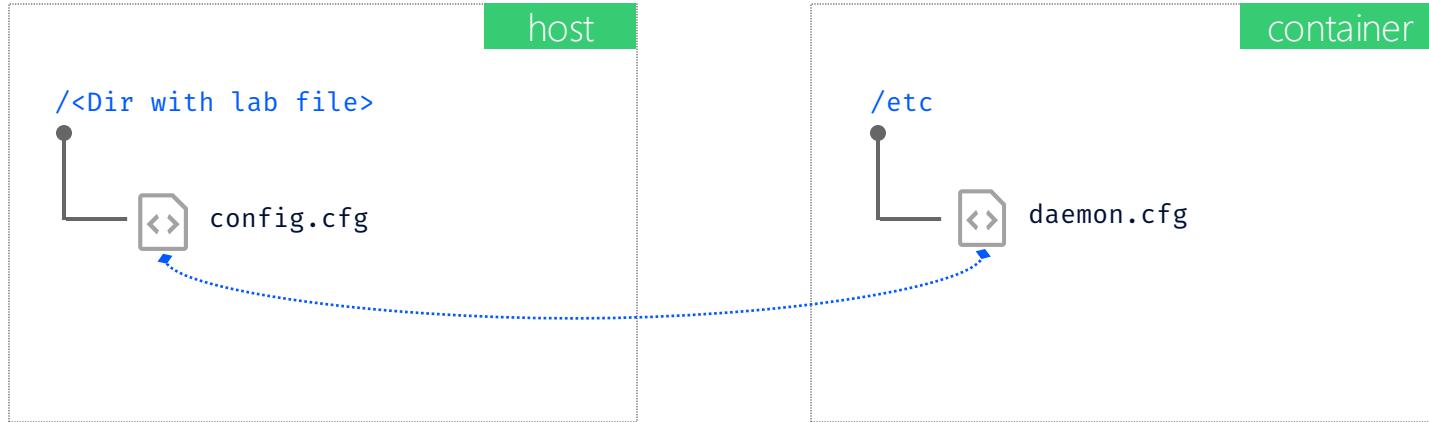
```
mgmt:  
  network: st  
  ipv4-subnet: 172.80.80.0/24  
  
topology:  
  nodes:  
    leaf1:  
      mgmt-ipv4: 172.80.80.11  
  
    leaf2:  
      mgmt-ipv4: 172.80.80.12
```



File binding

```
client2:  
kind: linux  
binds:  
- configs/client2:/config
```

- Bind mount files from host to a container
 - Providing configuration files
 - Providing executable scripts
 - Access to container's files



Entrypoint and command

- Entrypoint is the “command” that starts in a container
- Command (aka CMD) is a list of arguments passed to the entrypoint

```
gnmic:  
  kind: linux  
  image: ghcr.io/openconfig/gnmic:0.30.0  
  binds:  
    - gnmic-config.yml:/gnmic-config.yml:ro  
  cmd: --config /gnmic-config.yml --log subscribe
```

Environment variables

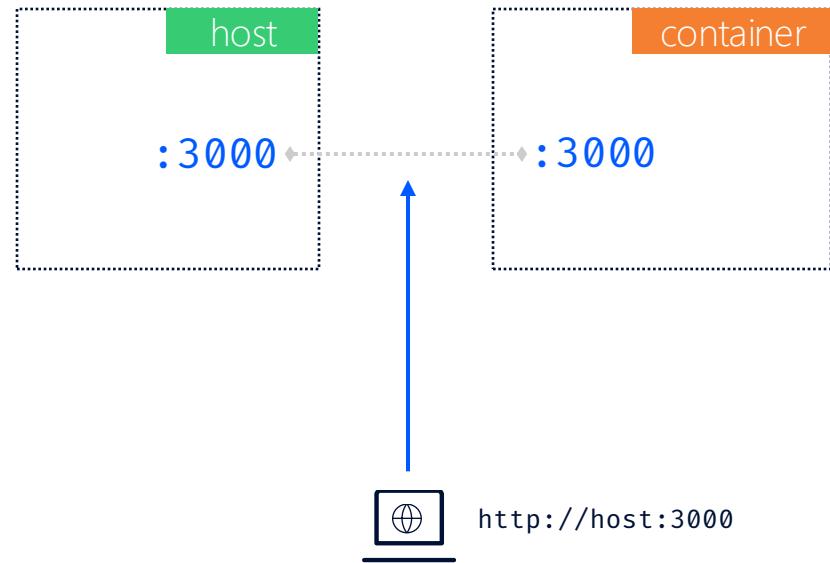
- Configure processes via env vars

```
grafana:  
  kind: linux  
  image: grafana/grafana:9.5.2  
  env:  
    GF_AUTH_ANONYMOUS_ENABLED: "true"  
    GF_AUTH_ANONYMOUS: "true"
```

Exposing ports

- Make services inside a container available to containerlab host

```
● ● ●  
grafana:  
  kind: linux  
  image: grafana/grafana:9.5.2  
  ports:  
    - 3000:3000
```



Codespaces

Free compute*

- Microsoft-backed compute cloud
- Generous free tier with a monthly quota reset
- 120cpu/hours a month **FREE**
- Containerlab integrated with Codespaces
- No \$\$\$ until explicitly committed

Nokia SR Linux Streaming Telemetry Lab

SR Linux has first-class Streaming Telemetry support thanks to [100% YANG coverage](#) of state and config data. The holistic coverage enables SR Linux users to stream **any** data off of the NOS with on-change, sample, or target-defined support. A discrepancy in visibility across APIs is not about SR Linux.

This lab represents a small Clos fabric with [Nokia SR Linux](#) switches running as containers. The lab topology consists of a Clos topology, plus a Streaming Telemetry stack comprised of [gnmic](#), [prometheus](#) and [grafana](#) applications.



Run In Codespaces

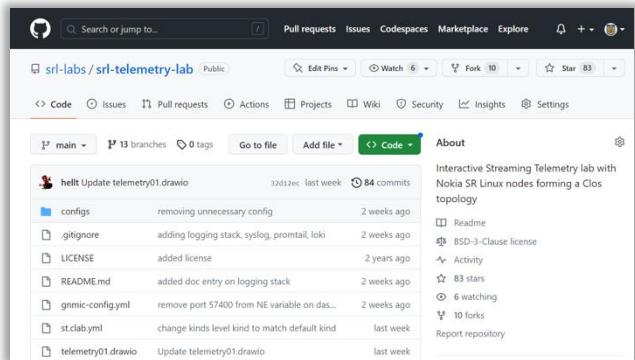
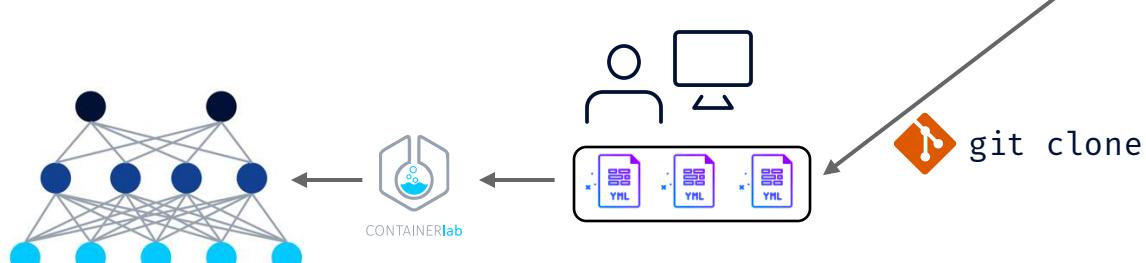
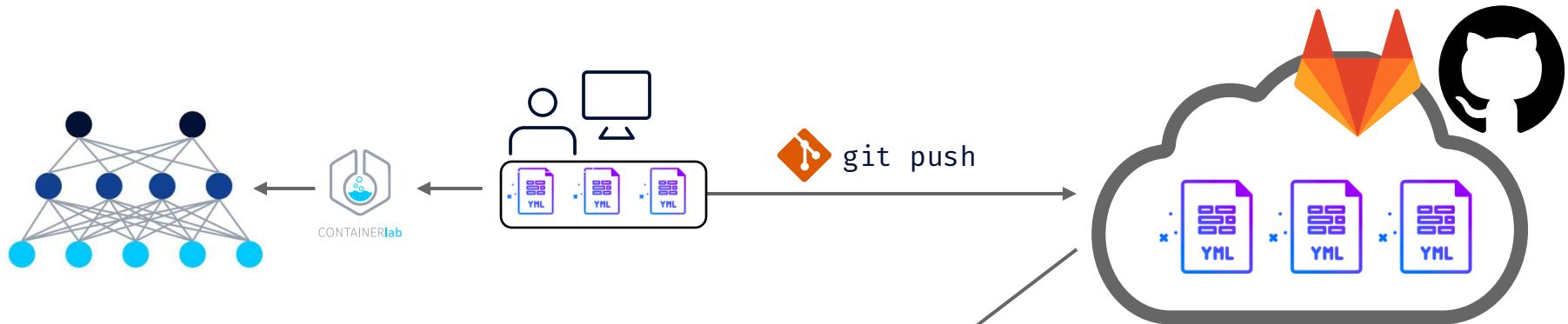
[Run this lab in GitHub Codespaces for free.](#)

[Learn more](#) about Containerlab for Codespaces.

Sharing and finding labs

Share your labs

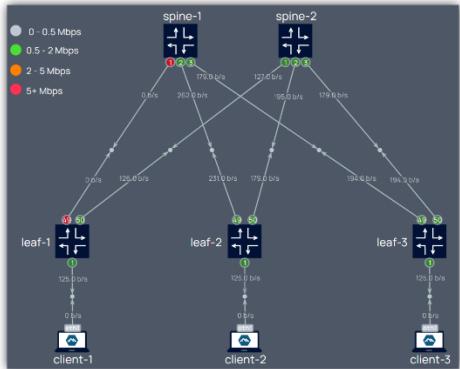
Lab As Code



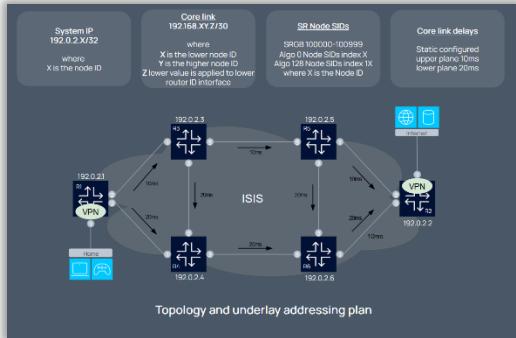
Demo labs



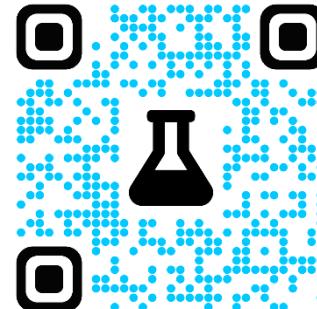
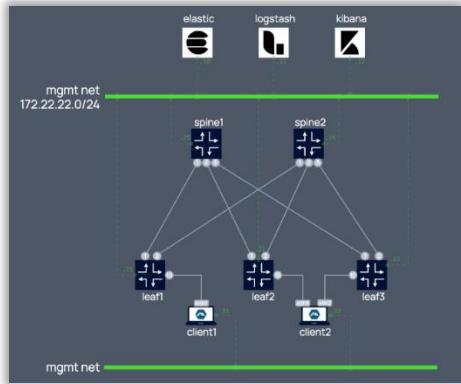
SR Linux Telemetry



Segment Routing



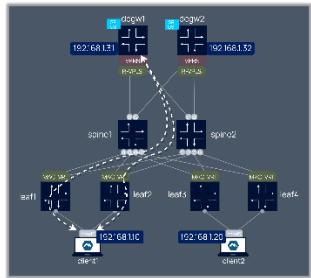
ELK Logging



Other labs for the community



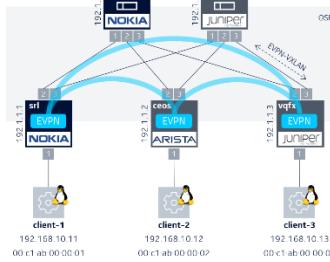
Nokia EVPN Interop



[srl-labs/nokia-evpn-lab](https://github.com/srl-labs/nokia-evpn-lab)



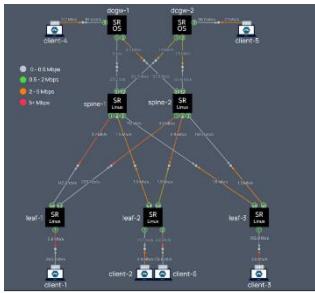
Multivendor EVPN



[srl-labs/multivendor-evpn-lab](https://github.com/srl-labs/multivendor-evpn-lab)



SR Linux & SROS Telemetry



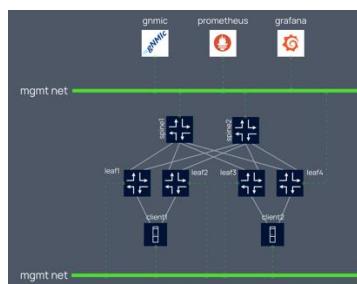
[srl-labs/srl-sros-telemetry-lab](https://github.com/srl-labs/srl-sros-telemetry-lab)



CONTAINERlab



SR Linux Oper-Group



[srl-labs/opergroup-lab](https://github.com/srl-labs/opergroup-lab)

NOKIA

Distributed collection of runnable labs

Add [clab-topo](#) topic to your repo

- Make your labs easily discoverable in a distributed way
- Each repo represents a runnable topology
- Publish your lab and others will see it!

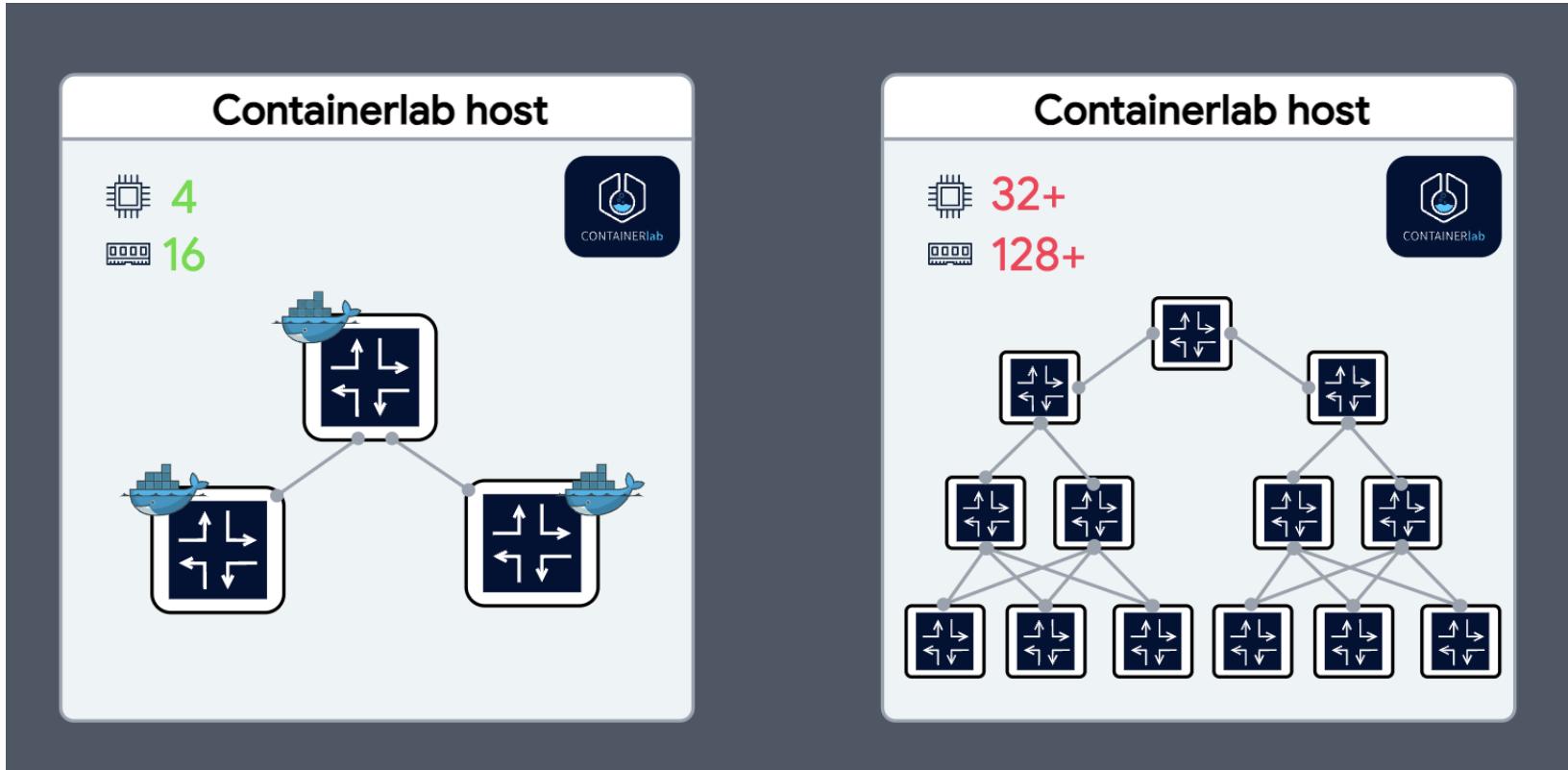
The screenshot shows the GitHub Topics interface. At the top, there's a navigation bar with 'Explore', 'Topics' (which is underlined), 'Trending', 'Collections', 'Events', and 'GitHub Sponsors'. Below this, a search bar and various icons are visible. The main area displays the '# clab-topo' topic page, which lists 20 public repositories matching the topic. Two repositories are shown in detail:

- srl-labs / srl-telemetry-lab**: A repository for an Interactive Streaming Telemetry lab with Nokia SR Linux nodes forming a Clos topology. It has 110 stars. The 'clab-topo' topic badge is present. The repository was updated on Nov 19, 2023, and is written in Shell.
- srl-labs / nokia-segment-routing-lab**: A repository for a lab to demonstrate Flex-Algo use case. It has 1 star. The 'clab-topo' topic badge is present. The repository was updated on May 24, 2023, and is written in JavaScript.

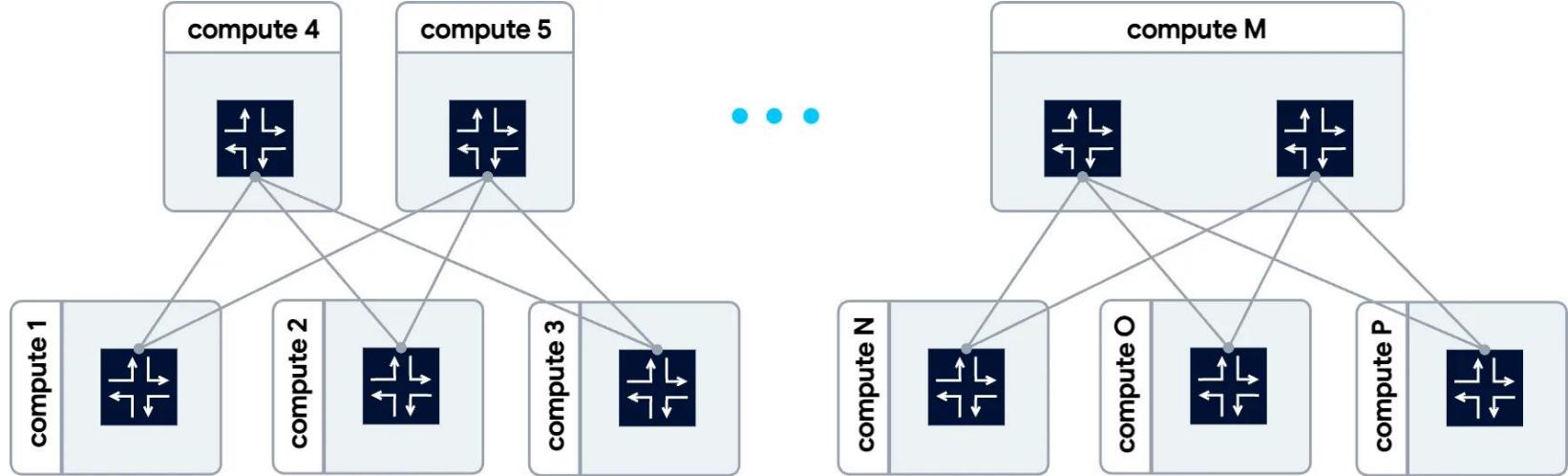
On the right side of the topic page, there are sections for 'Improve this page' (with a link to 'Curate this topic') and 'Add this topic to your repo' (with a link to 'Learn more'). A modal window is open over the second repository, titled 'About', which provides a summary of the lab's purpose and its tags: 'gnmi', 'streaming-telemetry', and 'clab-topo'.

But can it scale?

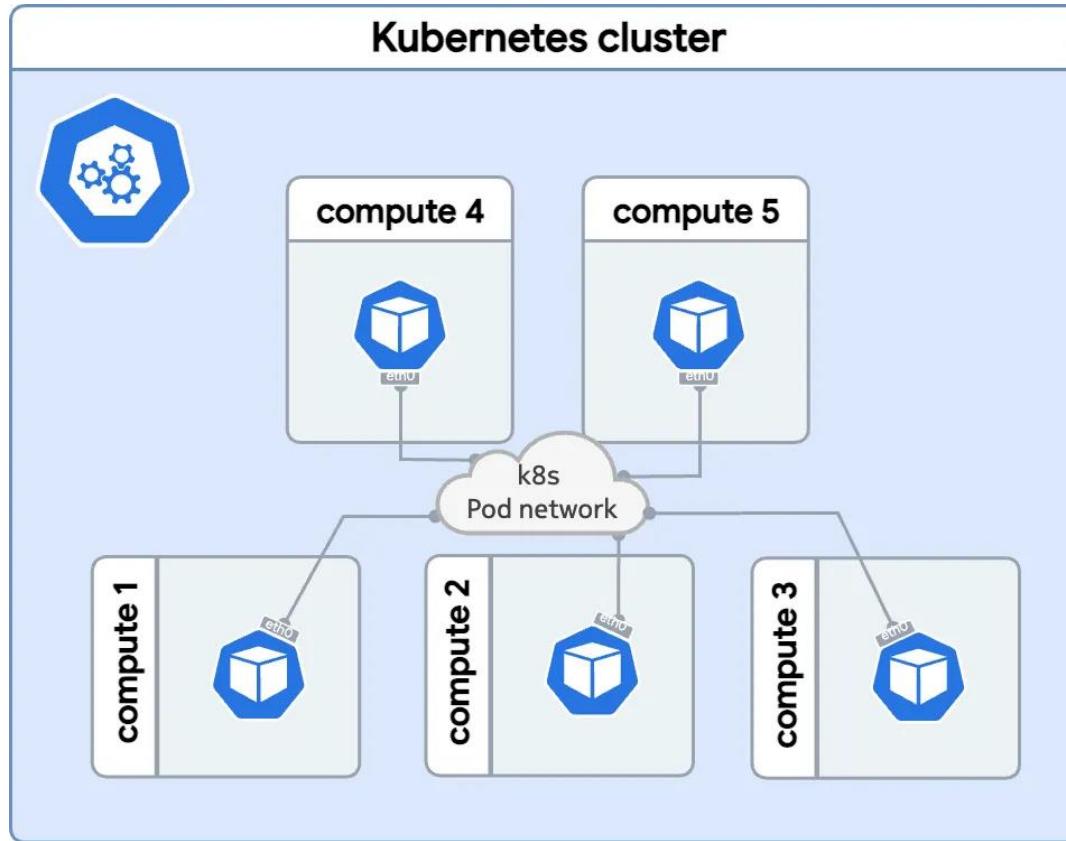
Vertical scaling is costly



Horizontal scaling is cheaper

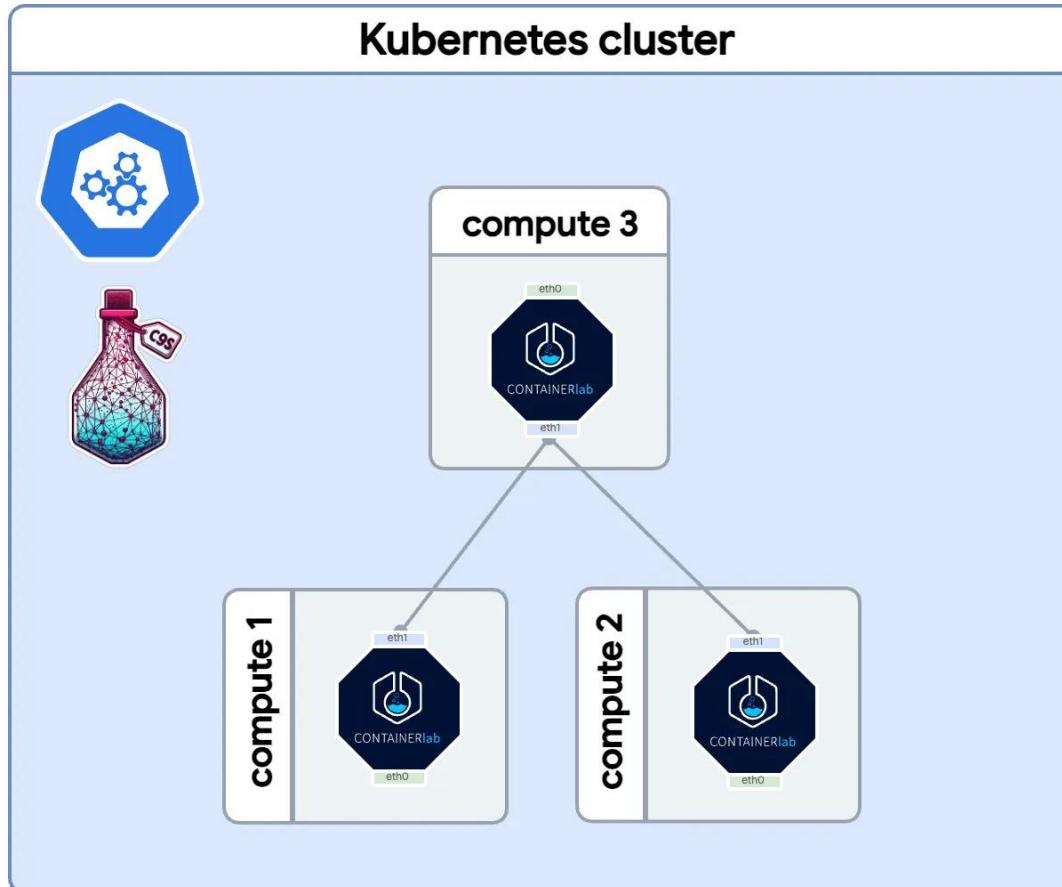


If only we had a system to schedule workload on top of workers...



Enter Clabernetes

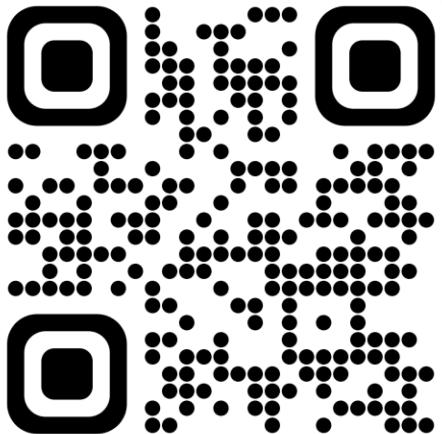
containerlab.dev/manual/clabernetes/



- Same topology structure
- Same supported NOSes
- With horizontal scale

Conclusion & Get in touch!

Containerlab docs



A screenshot of a web browser displaying the Containerlab documentation at containerlab.dev. The browser interface includes a back/forward button, a refresh button, and a search bar. The address bar shows the URL. The top right corner features an "Incognito" mode button and a "Relaunch to update" button. Below the address bar, there are links for "stash", "Nokia", and "k8s". A sidebar on the left contains a logo of a flask icon and the text "containerlab". The sidebar menu includes links for "Home", "Installation", "Quick start", "Kinds", "User manual", "Command reference", "Lab examples", "Release notes", and "Community". The main content area features a large logo of a flask containing liquid, with the text "CONTAINERlab" below it. At the bottom of the main content, there are social media links for "follow @go_containerlab" and "discord 165 online". The footer of the page contains descriptive text about the need for containerized lab environments and how Containerlab addresses this need.

containerlab

containerlab.dev

stash Nokia k8s

Search

srl-labs/containerlab
v0.49.0 ⭐1.1k 📈 216

Table of contents

Features

Use cases

Join us

Home

Installation

Quick start

Kinds

User manual >

Command reference >

Lab examples >

Release notes >

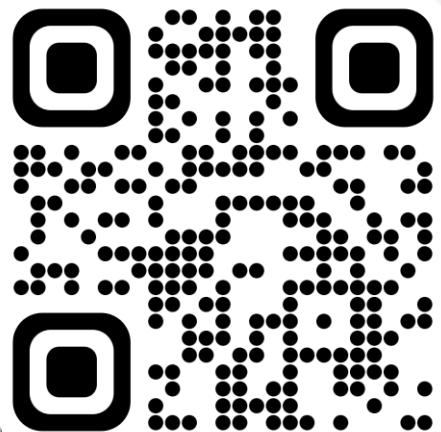
Community

CONTAINERlab

follow @go_containerlab discord 165 online

Containerized Network Operating Systems grows the demand to easily run multi lab topologies.
tration tools like docker-compose are not a good fit for that purpose, as they create connections between the containers which define a topology.

Containerlab Community



Screenshot of the Containerlab Community Slack interface:

General Channel:

Assigned to the management port where are no routes learned by the mgmt instance in your table. You can check the mgmt interface address using `show interface mgmt0`

Messages:

- @Saju Hi, the routes you see for instance mgmt is the local IPv6 address assigned
Ernesto Sánchez Yesterday at 10:31 PM
- Thanks Saju, but I don't understand why it doesn't learn through the eth interface. This is my cfg for one of the SRL: set / interface ethernet-1/1 set / interface ethernet-1/1 subinterface 0 set / interface ethernet-1/1 subinterface 0 ipv6 admin-state enable set / interface ethernet-1/1 subinterface 0 ipv6 address 2001:db8:bbbb:1::1/64 set / interface ethernet-1/1 subinterface 0 ipv6 router-advertisement router-role admin-state enable prefix 2001:db8:bbbb:1::/64 set / network-instance default set / network-instance default interface ethernet-1/1.
- @Ernesto Sánchez Thanks Saju, but I don't understand why it doesn't learn through
Roman Yesterday at 11:07 PM
- first I would check with tcpdump if you get RAs arriving to srl1 e1-1 interface
`docker exec -it <srl-container-name> tcpdump -nni e1-1`
- @Roman first I would check with tcpdump if you get RAs arriving to srl1 e1-1 interface
Ernesto Sánchez Yesterday at 11:08 PM
- Excellent, thanks @Roman I will try

Members:

- CLAB TEAM — 3
 - henderiw
 - karimra
 - Roman
- ONLINE — 162
 - ZX-
 - 1_damage
 - 3zn00b
 - 404
 - 81uemana
 - _johnski_
 - Aaron
 - AbMedhat
 - alejo_guevara
 - alexhassan

NOKIA

Where to go next?

- Discord server - <https://discord.gg/2A8ZxM7hD9>
- Clabernetes docs - <https://containerlab.dev/manual/clabernetes/>



CONTAINER**lab**

<https://containerlab.dev>



and Clabernetes!

NOKIA

Copyright and confidentiality

The contents of this document are proprietary and confidential property of Nokia. This document is provided subject to confidentiality obligations of the applicable agreement(s).

This document is intended for use by Nokia's customers and collaborators only for the purpose for which this document is submitted by Nokia. No part of this document may be reproduced or made available to the public or to any third party in any form or means without the prior written permission of Nokia. This document is to be used by properly trained professional personnel. Any use of the contents in this document is limited strictly to the use(s) specifically created in the applicable agreement(s) under which the document is submitted. The user of this document may voluntarily provide suggestions, comments or other feedback to Nokia in respect of the contents of this document ("Feedback"). Such Feedback may be used in Nokia products and

related specifications or other documentation. Accordingly, if the user of this document gives Nokia Feedback on the contents of this document, Nokia may freely use, disclose, reproduce, license, distribute and otherwise commercialize the feedback in any Nokia product, technology, service, specification or other documentation.

Nokia operates a policy of ongoing development. Nokia reserves the right to make changes and improvements to any of the products and/or services described in this document or withdraw this document at any time without prior notice.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability or contents

of this document. NOKIA SHALL NOT BE RESPONSIBLE IN ANY EVENT FOR ERRORS IN THIS DOCUMENT or for any loss of data or income or any special, incidental, consequential, indirect or direct damages howsoever caused, that might arise from the use of this document or any contents of this document.

This document and the product(s) it describes are protected by copyright according to the applicable laws.

Nokia is a registered trademark of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.