Network Automation in light of Model Driven Management

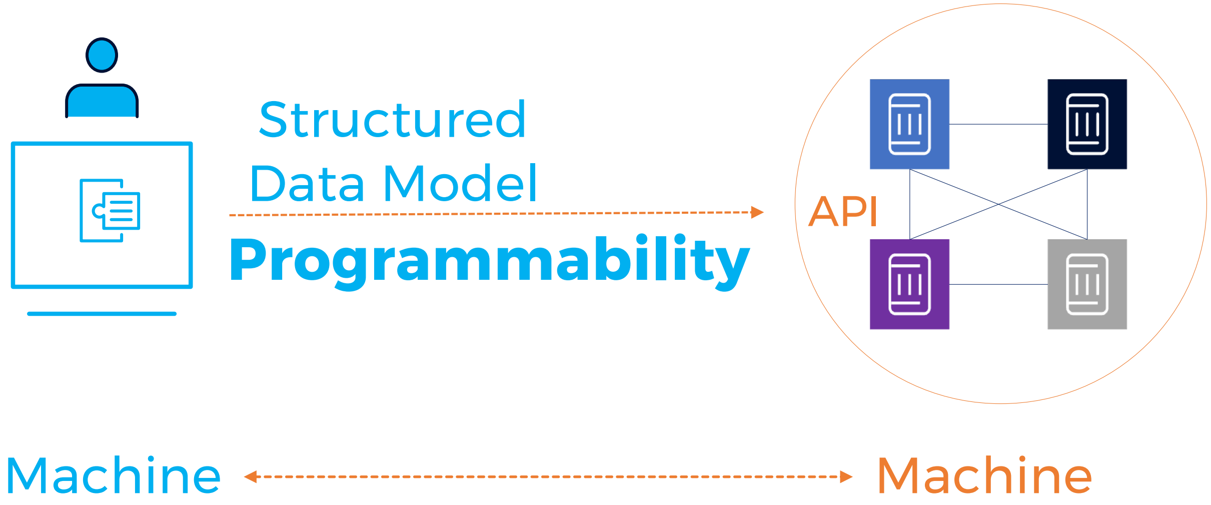
June-2024

NANOG™

Evolution of Network Management: Old but (not) gold anymore

- Designed primarily as a human interface
- Vendor Specific
- Lack of Standardization
- Limited Configuration Capabilities
- Scalability Issues

Robust management and automation capabilities …is a necessity



Structured Data Model

Programmability

API

Machine

Machine

- Structured Data

Standardized and organized format of representing information

- API

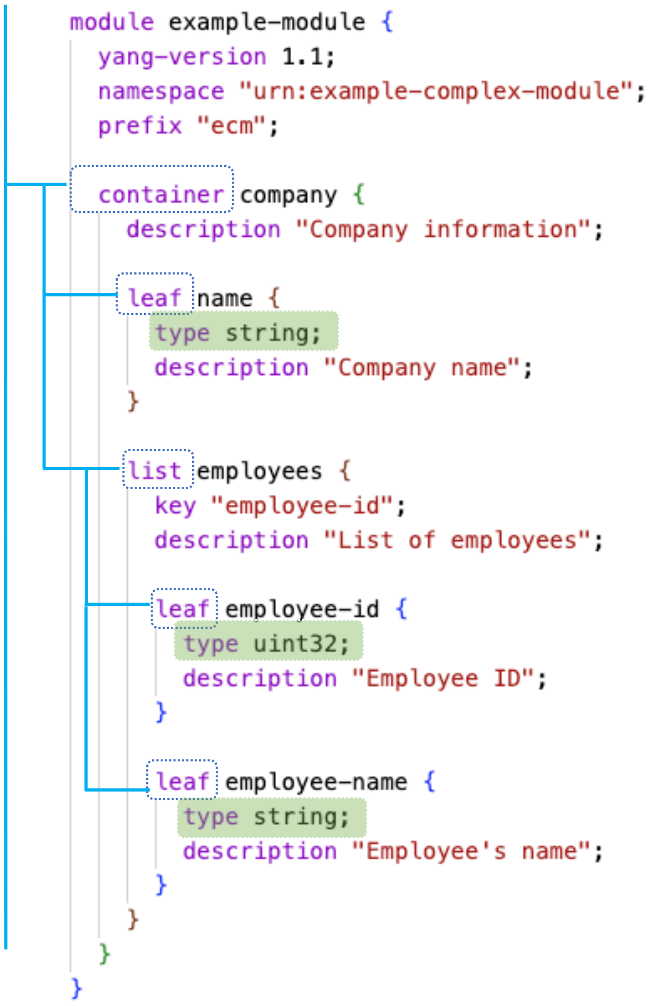
Standardized way for different systems and devices to communicate and exchange information

Key for automation: Describe Your World of Data

YANG : framework for modelling data

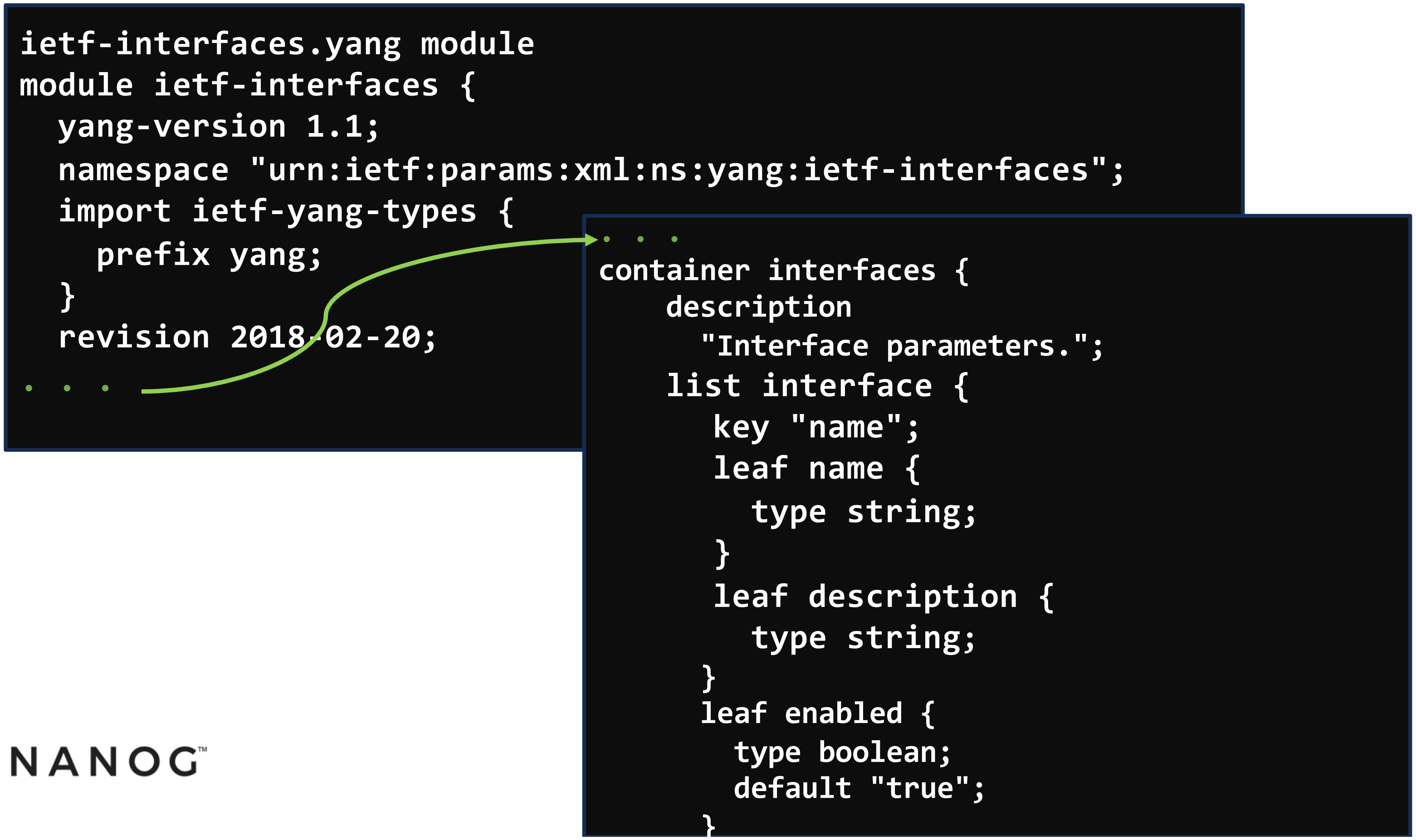
Defines how the data should be

- Structured — hierarchical structure, similar to a tree
- Represented
  - **Containers**
    group related data nodes together
  - **Lists**
    define an ordered collection of elements
  - **Leaf**
    represent the basic pieces of data
- Formatted — various data types, including basic types like integers, strings



```
module example-module {
  yang-version 1.1;
  namespace "urn:example-complex-module";
  prefix "ecm";

  container company {
    description "Company information";

    leaf name {
      type string;
      description "Company name";
    }

    list employees {
      key "employee-id";
      description "List of employees";

      leaf employee-id {
        type uint32;
        description "Employee ID";
      }

      leaf employee-name {
        type string;
        description "Employee's name";
      }
    }
  }
}
```

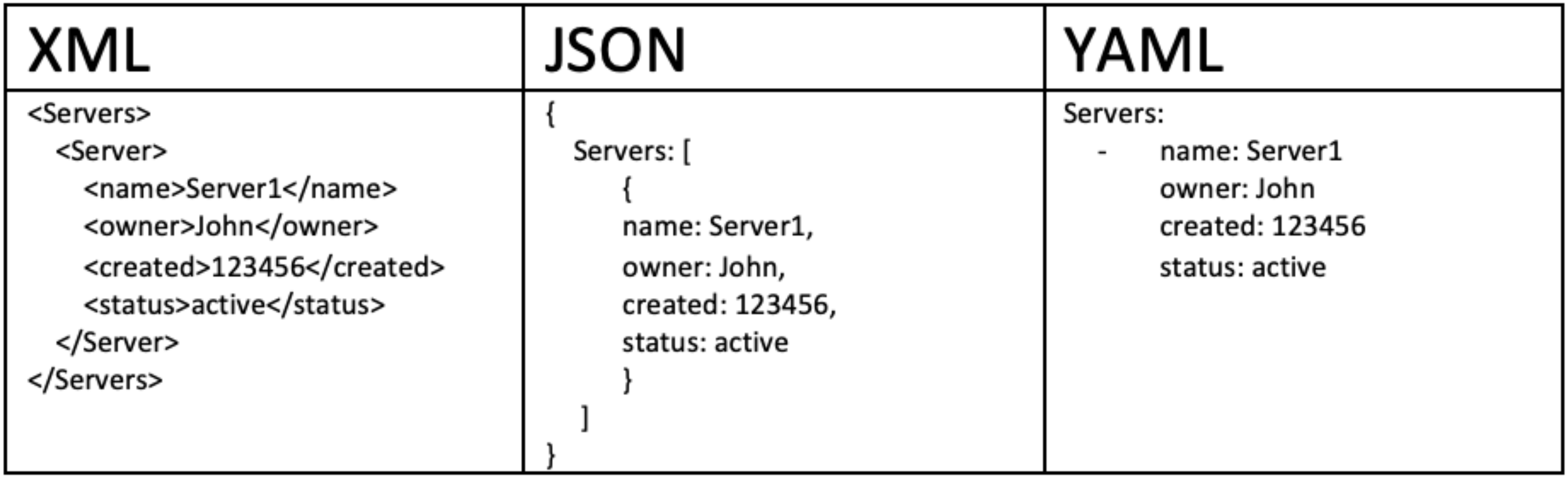
Interface YANG Example



```
ietf-interfaces.yang module
module ietf-interfaces {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
  import ietf-yang-types {
    prefix yang;
  }
  revision 2018-02-20;
. . .
```

```
. . .
container interfaces {
    description
     "Interface parameters.";
    list interface {
      key "name";
      leaf name {
        type string;
      }
      leaf description {
        type string;
     }
     leaf enabled {
       type boolean;
       default "true";
     }
```

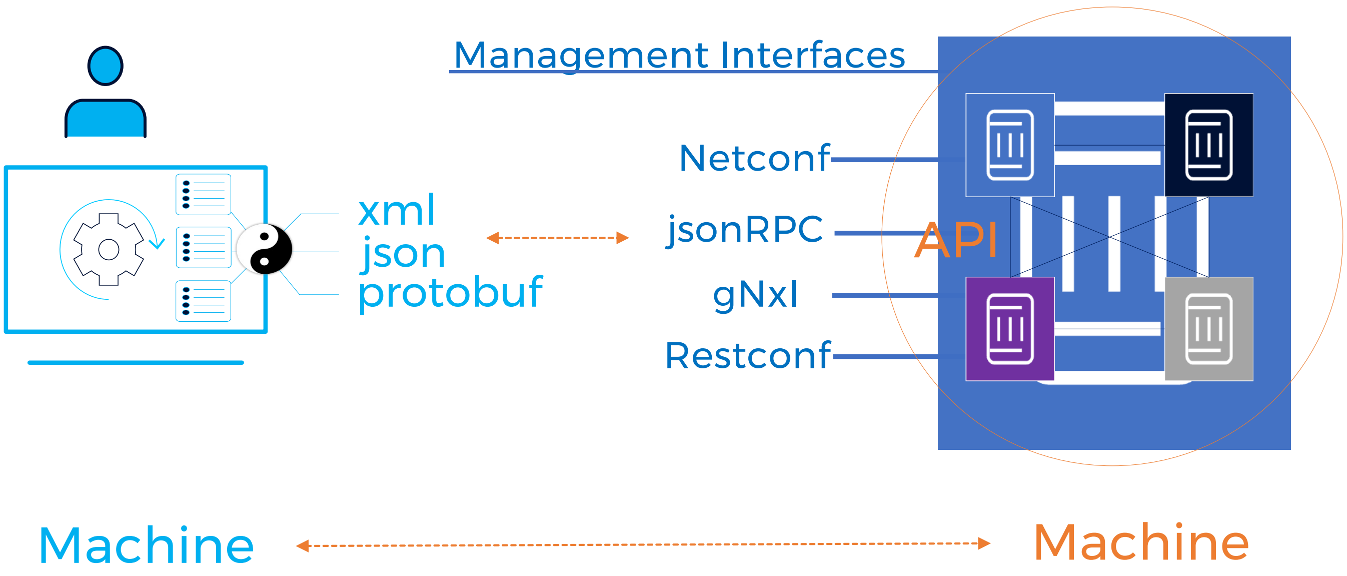
Consistency matters



XML	JSON	YAML

<Servers>
   <Server>
      <name>Server1</name>
      <owner>John</owner>
      <created>123456</created>
      <status>active</status>
   </Server>
</Servers>
	{
	   Servers: [
	        {
	        name: Server1,
	        owner: John,
	        created: 123456,
	        status: active
	        }
	    ]
	}
		Servers:
		   - name: Server1
		           owner: John
		           created: 123456
		           status: active

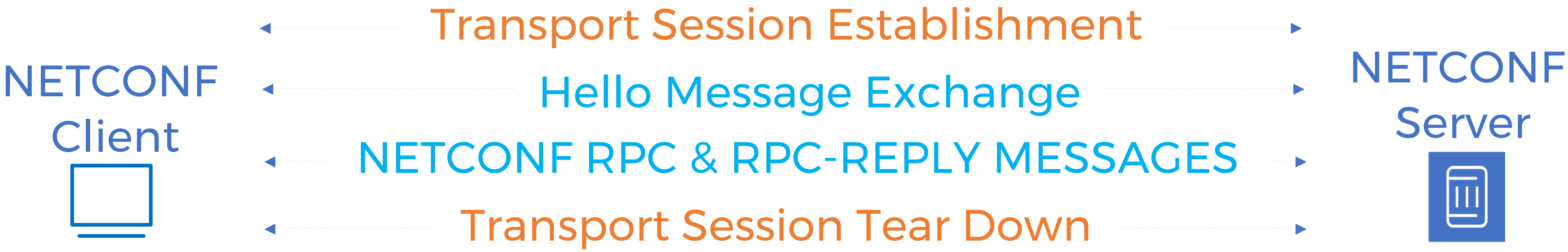
YANG: Unleashing the Potential of Network APIs



- ☯ YANG models configuration and state data of network devices
- Network management protocol that uses YANG models to encode data in different formats

Programmatic Interfaces: Netconf

```
ssh user@hostname -p 830 -s netconf
```



NETCONF Client

Transport Session Establishment

Hello Message Exchange

NETCONF RPC & RPC-REPLY MESSAGES

Transport Session Tear Down

NETCONF Server

Content	Yang Models
Operations	<get>,<get-config>,<edit-config>,<copy-config>
Messages	RPCs, Notifications, Hello (Instructions, requests formatted in XML)
Transport	SSH, TLS

Programmatic Interfaces: Restconf

http(s)://<hostname>:<port>/restconf/data/<yang-module>:<data-node>

RESTCONF Client

Similar to RESTful API model

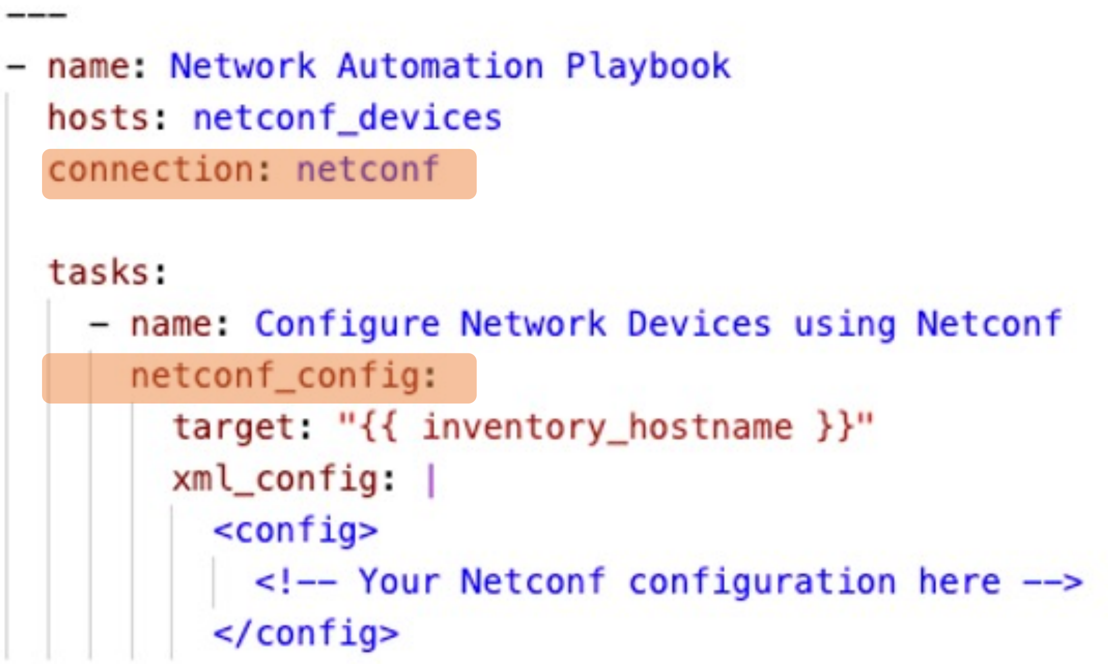
Secure and connection-oriented session

HTTP Request, Http Response MESSAGES

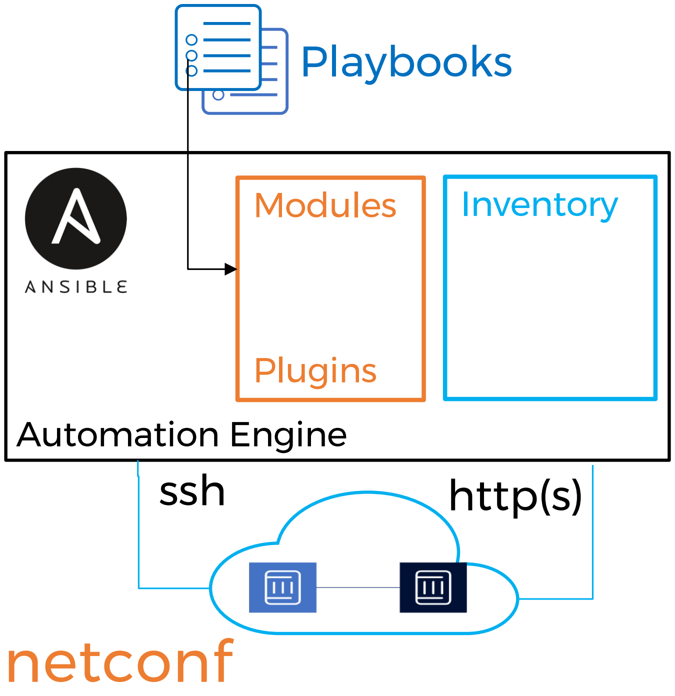
RESTCONF Server

Content	Yang Models
Operations	Get, Post, Put, Patch, Delete
Messages	HTTP Request, Http Response (Instructions, requests formatted in XML or JSON)
Transport	HTTP, HTTPs

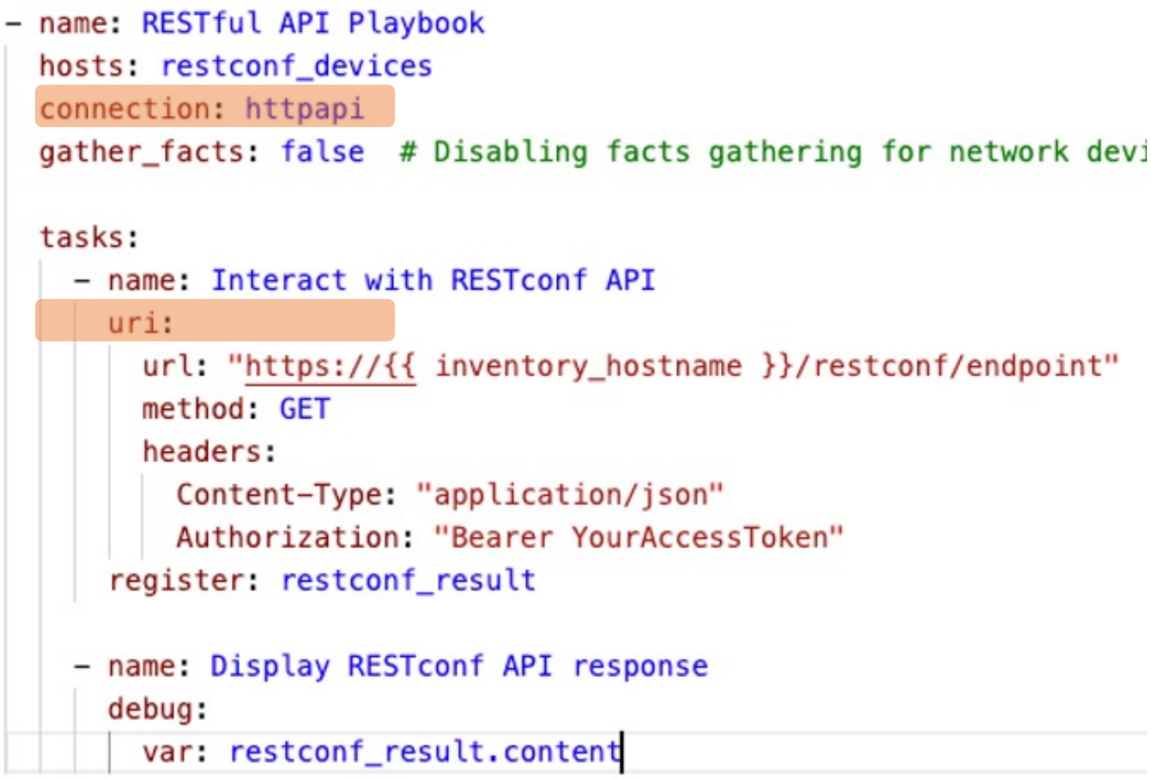
Ansible Module: ansible.netcommon.netconf_config



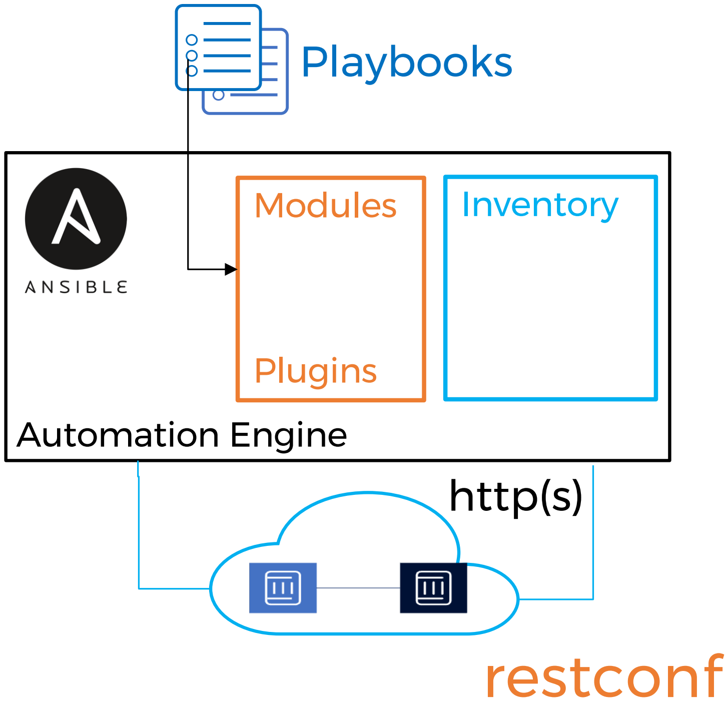
```
---
- name: Network Automation Playbook
  hosts: netconf_devices
  connection: netconf

  tasks:
    - name: Configure Network Devices using Netconf
      netconf_config:
        target: "{{ inventory_hostname }}"
        xml_config: |
          <config>
            <!-- Your Netconf configuration here -->
          </config>
```



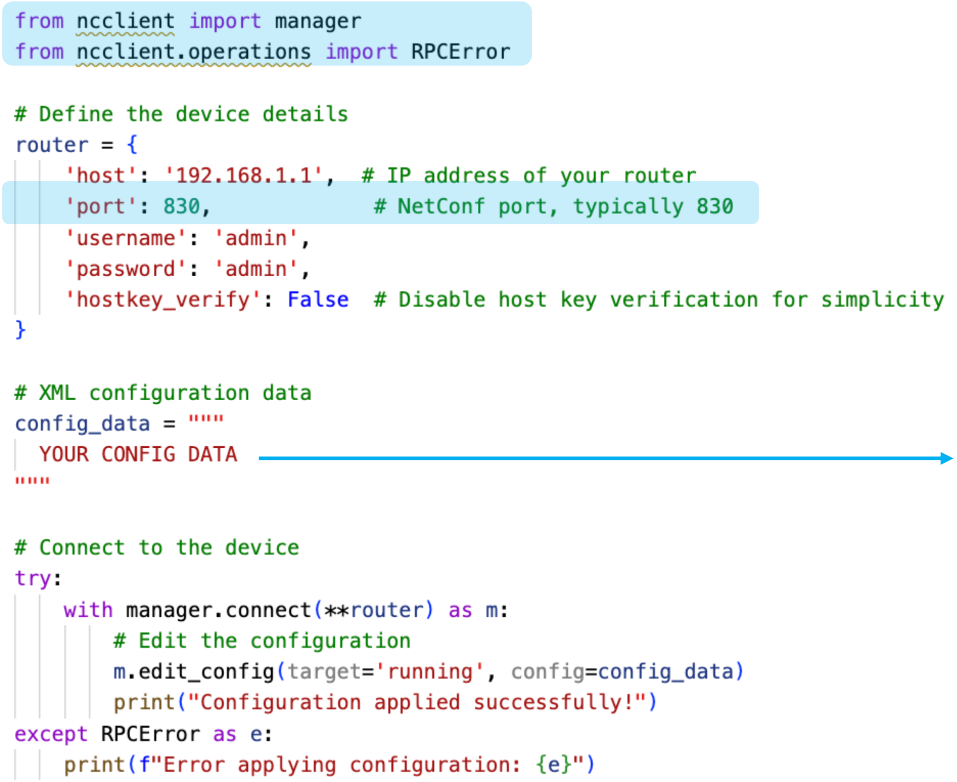
Ansible Module: ansible.netcommon.restconf_config



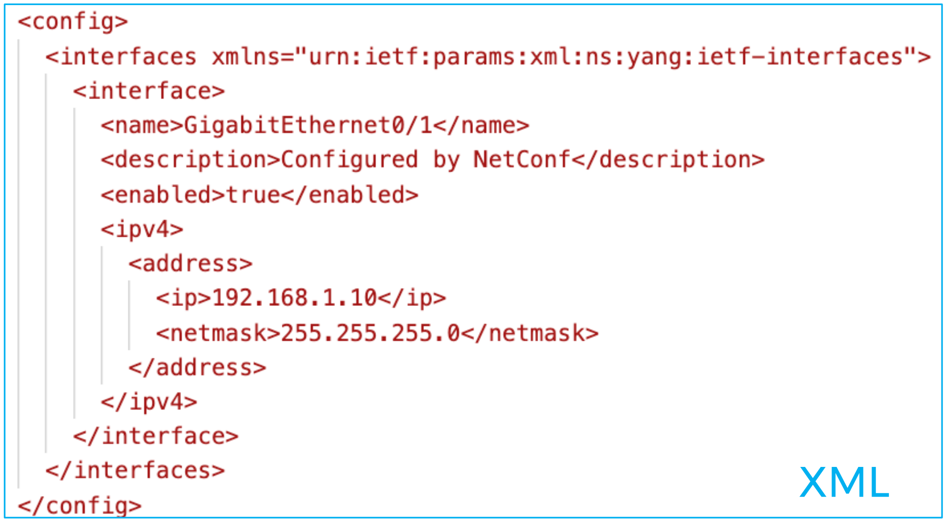
```
- name: RESTful API Playbook
  hosts: restconf_devices
  connection: httpapi
  gather_facts: false  # Disabling facts gathering for network devi

  tasks:
    - name: Interact with RESTconf API
      uri:
        url: "https://{{ inventory_hostname }}/restconf/endpoint"
        method: GET
        headers:
          Content-Type: "application/json"
          Authorization: "Bearer YourAccessToken"
      register: restconf_result

    - name: Display RESTconf API response
      debug:
        var: restconf_result.content
```



Opensource eco system: python



```python
from ncclient import manager
from ncclient.operations import RPCError

# Define the device details
router = {
    'host': '192.168.1.1',  # IP address of your router
    'port': 830,             # NetConf port, typically 830
    'username': 'admin',
    'password': 'admin',
    'hostkey_verify': False  # Disable host key verification for simplicity
}

# XML configuration data
config_data = """
  YOUR CONFIG DATA
"""

# Connect to the device
try:
    with manager.connect(**router) as m:
        # Edit the configuration
        m.edit_config(target='running', config=config_data)
        print("Configuration applied successfully!")
except RPCError as e:
    print(f"Error applying configuration: {e}")
```



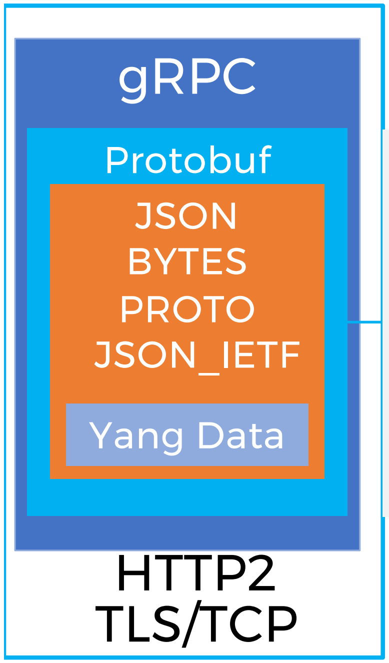
```xml
<config>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>GigabitEthernet0/1</name>
      <description>Configured by NetConf</description>
      <enabled>true</enabled>
      <ipv4>
        <address>
          <ip>192.168.1.10</ip>
          <netmask>255.255.255.0</netmask>
        </address>
      </ipv4>
    </interface>
  </interfaces>
</config>
```

XML

gRPC Services: gNMI

Network Management

Streaming Telemetry



gNMI (gRPC Network Management Interface)

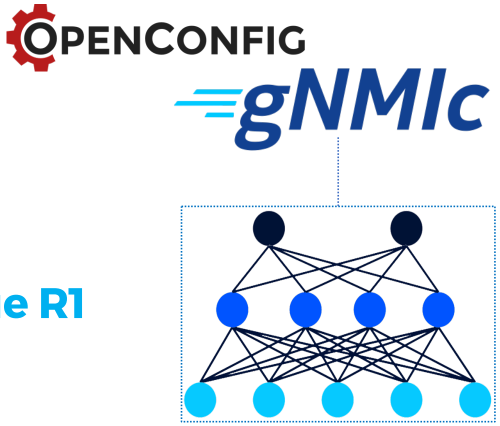
```
service gNMI {
  rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
  rpc Get(GetRequest) returns (GetResponse);
  rpc Set(SetRequest) returns (SetResponse);
  rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```

NANOG

Nokia donated gNMIc to Openconfig

cli interface

```
gnmic capabilities/set/get/subscribe/listen...
```



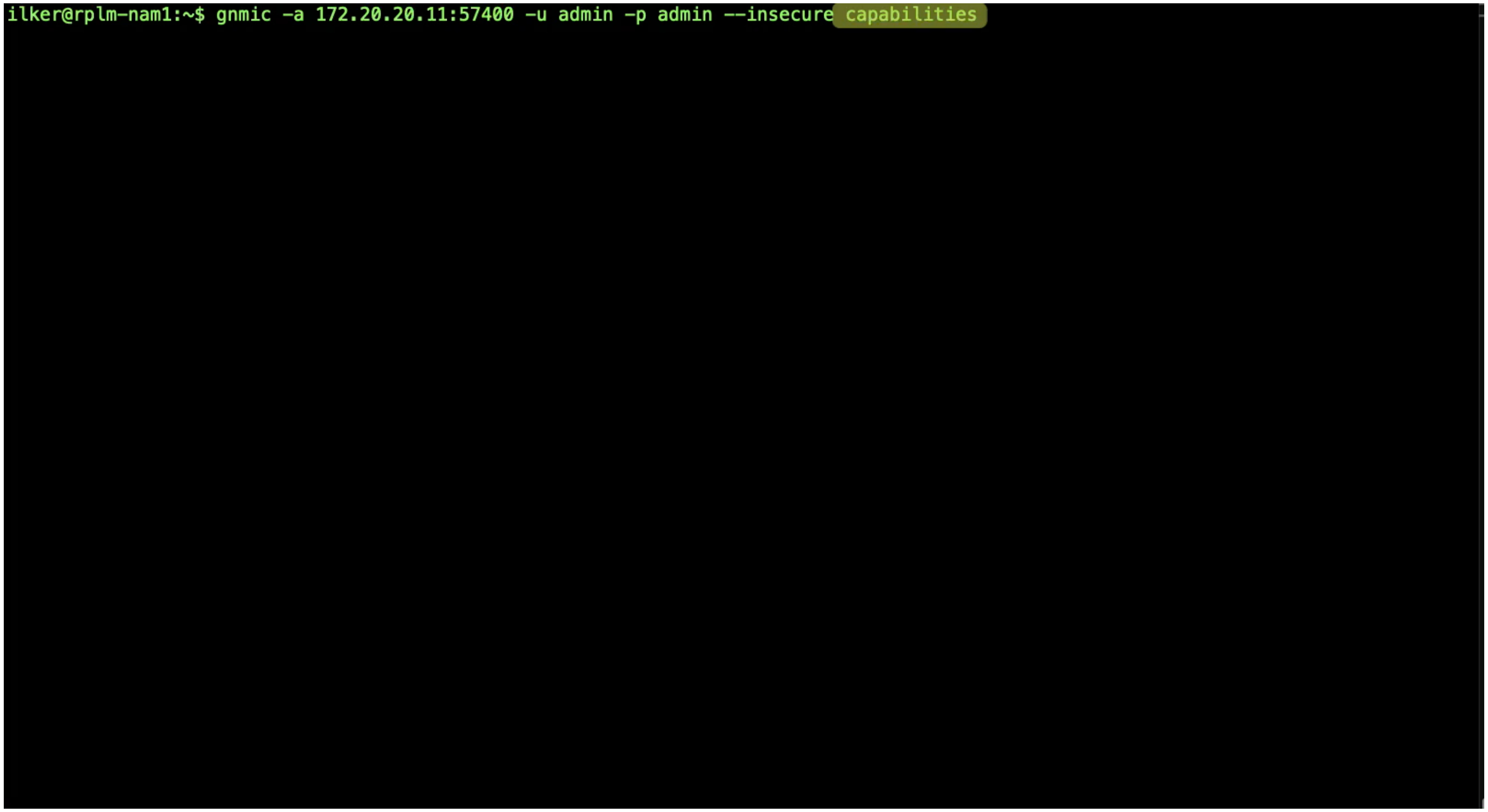
Configuration

```
gnmic set --update-path /configure/system/name --update-value R1
```

Telemetry data collection

```
gnmic get --path "/state/ports[port-id=*]"
gnmic sub --path "/state/ports[port-id=*]"
```

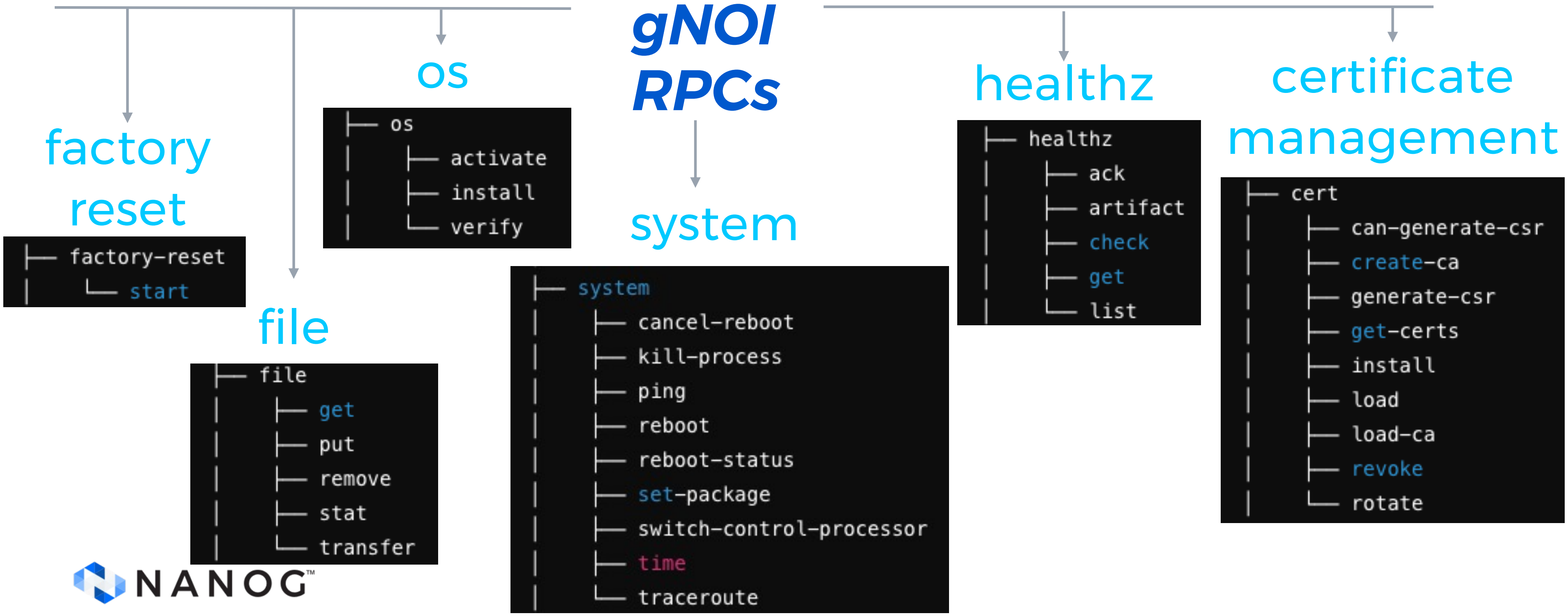
gNMIc on the go



```
ilker@rplm-nam1:~$ gnmic -a 172.20.20.11:57400 -u admin -p admin --insecure capabilities
```

gRPC Services: gNOI

gNOI (gRPC Network Operations Interface)



gNOI RPCs

factory reset

```
├── factory-reset
│    └── start
```

file

```
├── file
│    ├── get
│    ├── put
│    ├── remove
│    ├── stat
│    └── transfer
```

os

```
├── os
│    ├── activate
│    ├── install
│    └── verify
```

system

```
├── system
│    ├── cancel-reboot
│    ├── kill-process
│    ├── ping
│    ├── reboot
│    ├── reboot-status
│    ├── set-package
│    ├── switch-control-processor
│    ├── time
│    └── traceroute
```

healthz

```
├── healthz
│    ├── ack
│    ├── artifact
│    ├── check
│    ├── get
│    └── list
```

certificate management

```
├── cert
│    ├── can-generate-csr
│    ├── create-ca
│    ├── generate-csr
│    ├── get-certs
│    ├── install
│    ├── load
│    ├── load-ca
│    ├── revoke
│    └── rotate
```

gNOI use case: simplifying sw upgrade

1 PRE-UPGRADE HEALTH CHECK

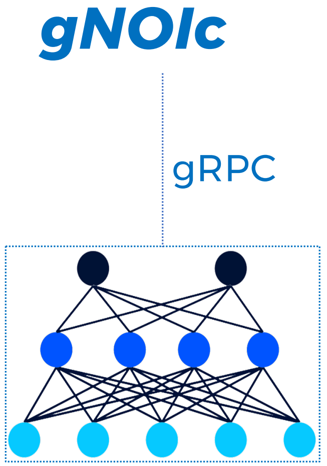
2 CONFIG BACKUP

3 SOFTWARE TRANSFER

4 SOFTWARE ACTIVATE

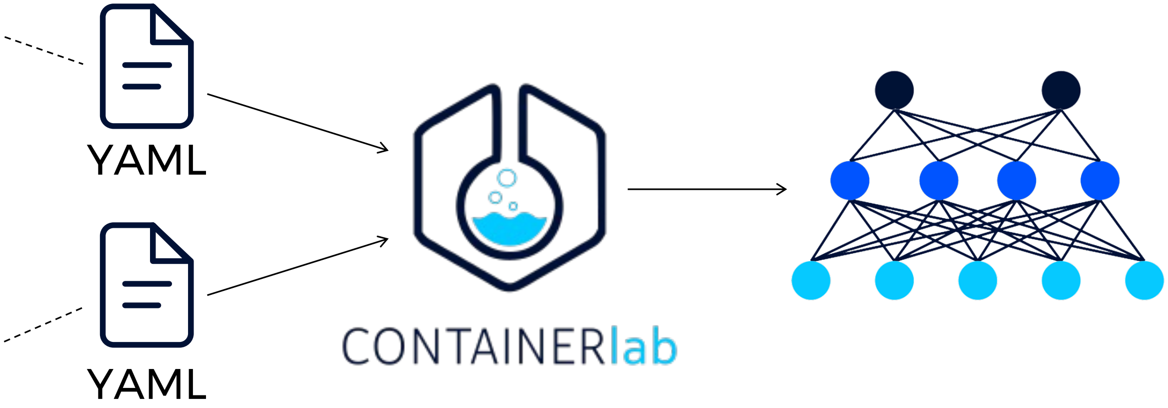
5 SOFTWARE VERIFY

- **healthz**
  - CHECK
  - GET
- **file**
  - GET
- **os**
  - INSTALL
  - ACTIVATE
  - VERIFY



Containerlab: Bringing declarativeness to network labs

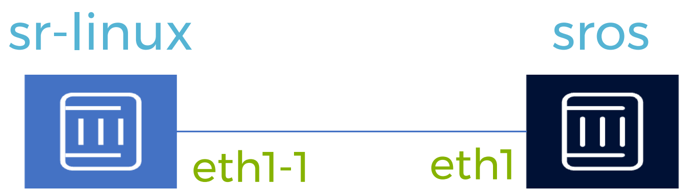
```
name: mylab

topology:
  nodes:
      :
      :

  links:
   -   :
```



Containerlab: Topology Definition

```yaml
name: mylab

topology:
  nodes:

    srl:
      kind: nokia_srlinux
      image: ghcr.io/nokia/srlinux:23.3.1

    sros:
      kind: vr-nokia_sros
      image: sros:23.3.R1
      license: license.txt

  links:
   - endpoints: ["srl:e1-1", "sros:eth1"]
```

Logical view



Containerlab: Multivendor images

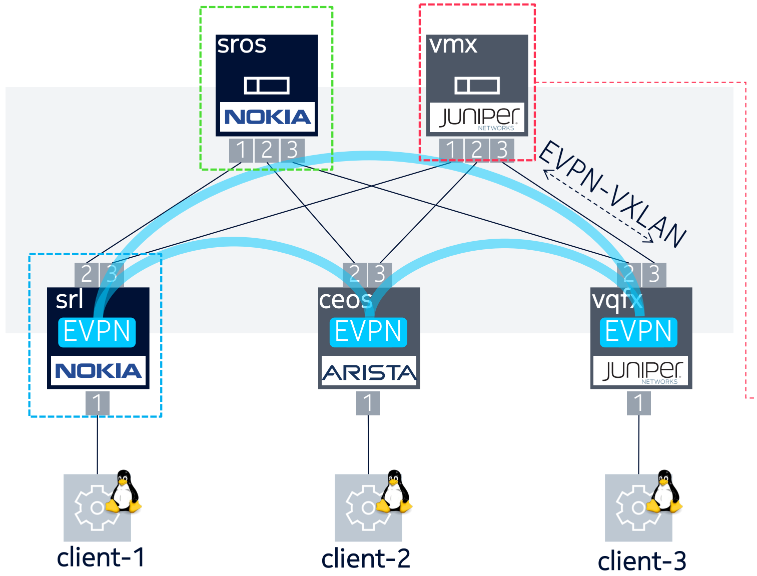
NOKIA sr-linux ▥ vr-sros	JUNIPER NETWORKS vr-vmx vr-vqfx crpd ▥	ARISTA ceos ▥ vr-veos	CISCO vr-xrv9k vr-csr c8000 vr-n9kv
NVIDIA cvx ▥	paloalto NETWORKS vr-pan	DELL vr-ftosv	ixia ▥ Keysight_ixia-c
HPE aruba networking vr-aoscx	MikroTik vr-ros	sonic-vs ▥ frr ▥	ip infusion ipinfusion_ocnos

NANOG

vrnetlab/vrnetlab: Run virtual routers with docker

▥ = Container image

Containerlab: Topology File

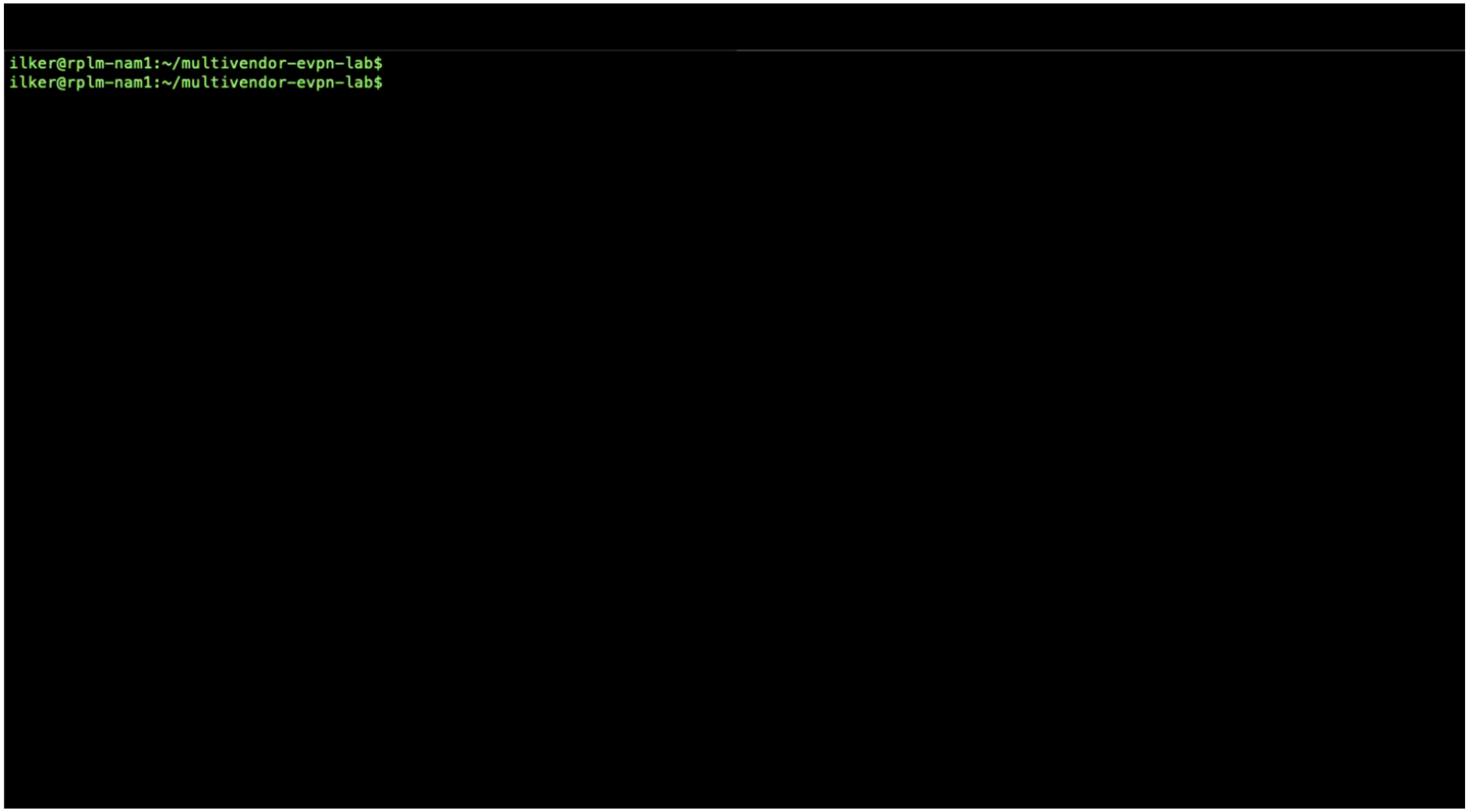


multivendor-evpn.clab.yml

```
name: multivendor

topology:
  kinds:
    srl:
      image: ghcr.io/nokia/srlinux:21.11.3
    vr-sros:
      image: registry.srlinux.dev/pub/vr-sros:22.5.R1
      license: ./license/sros.lic
    ceos:
      image: registry.srlinux.dev/pub/ceos:4.26.2.1F
    vr-vmx:
      image: registry.srlinux.dev/pub/vr-vmx:21.1R1.11
    vr-vqfx:
      image: registry.srlinux.dev/pub/vr-vqfx:19.4R1.10
    linux:
      image: ghcr.io/hellt/network-multitool

  nodes:
🟩  sros:
        kind: vr-sros
        startup-config: ./config/sros.cfg
🟥  vmx:
        kind: vr-vmx
        startup-config: ./config/vmx.cfg
🟦  srl:
        kind: srl
        type: ixrd2
        startup-config: ./config/srl.cfg
<snipp>
```

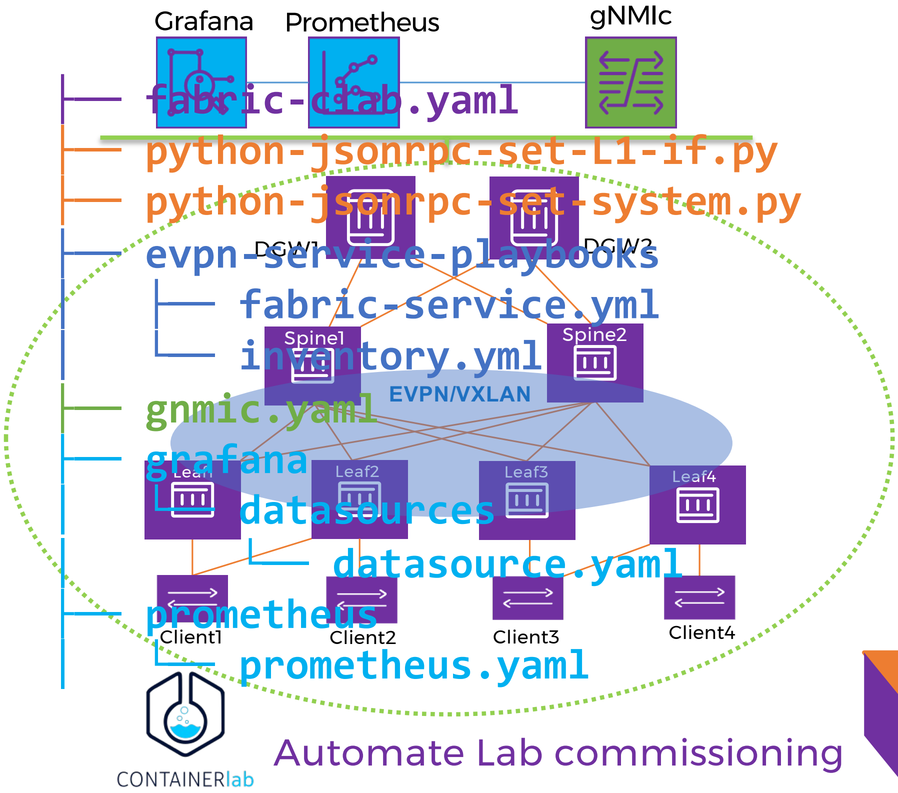
```
links:
      ##CLOS fabric
    - endpoints: ["sros:eth1", "srl:e1-2"]
    - endpoints: ["sros:eth2", "ceos:eth2"]
    - endpoints: ["sros:eth3", "vqfx:eth2"]
    - endpoints: ["vmx:eth1", "srl:e1-3"]
    - endpoints: ["vmx:eth2", "ceos:eth3"]
    - endpoints: ["vmx:eth3", "vqfx:eth3"]

      ##CE
    - endpoints: ["client-1:eth1", "srl:e1-1"]
    - endpoints: ["client-2:eth1", "ceos:eth1"]
    - endpoints: ["client-3:eth1", "vqfx:eth1"]
```

Containerlab: How?



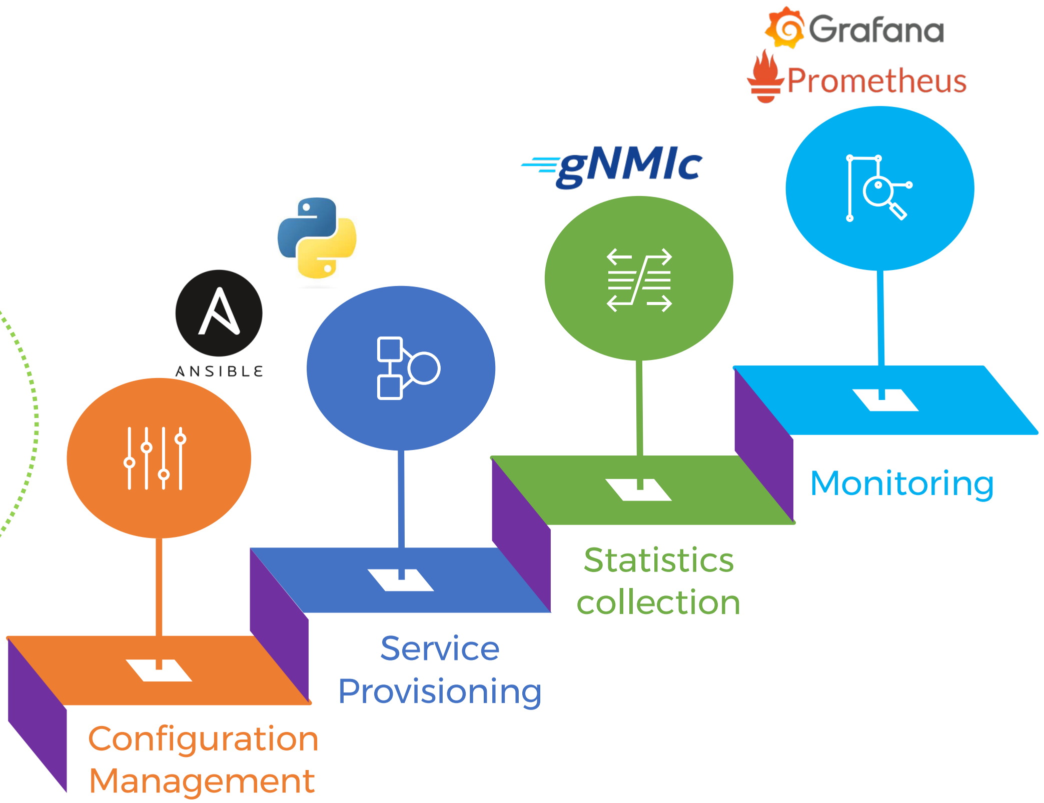
```
ilker@rplm-nam1:~/multivendor-evpn-lab$
ilker@rplm-nam1:~/multivendor-evpn-lab$
```

Witness e2e Automation

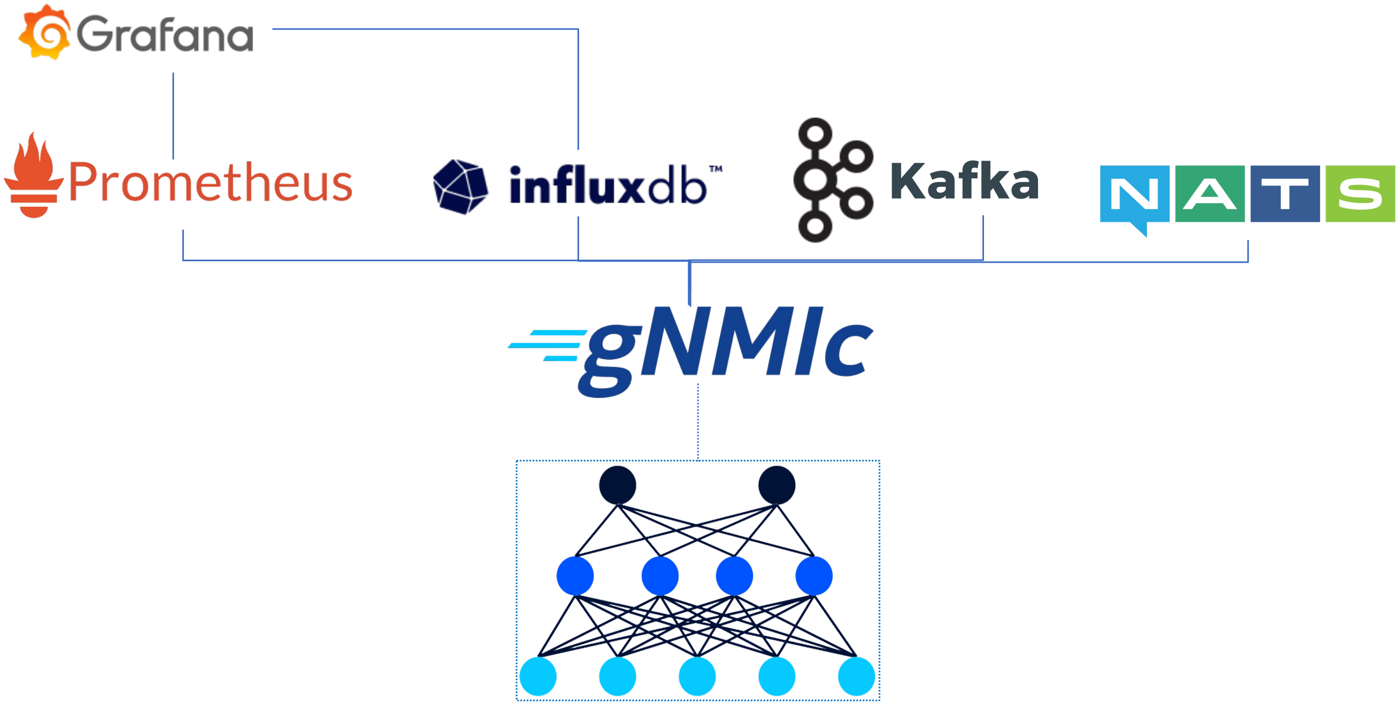


```
├── fabric-clab.yaml
├── python-jsonrpc-set-L1-if.py
├── python-jsonrpc-set-system.py
├── evpn-service-playbooks
│   ├── fabric-service.yml
│   └── inventory.yml
├── gnmic.yaml
├── grafana
│   └── datasources
│       └── datasource.yaml
├── prometheus
│   └── prometheus.yaml
```

Automate Lab commissioning

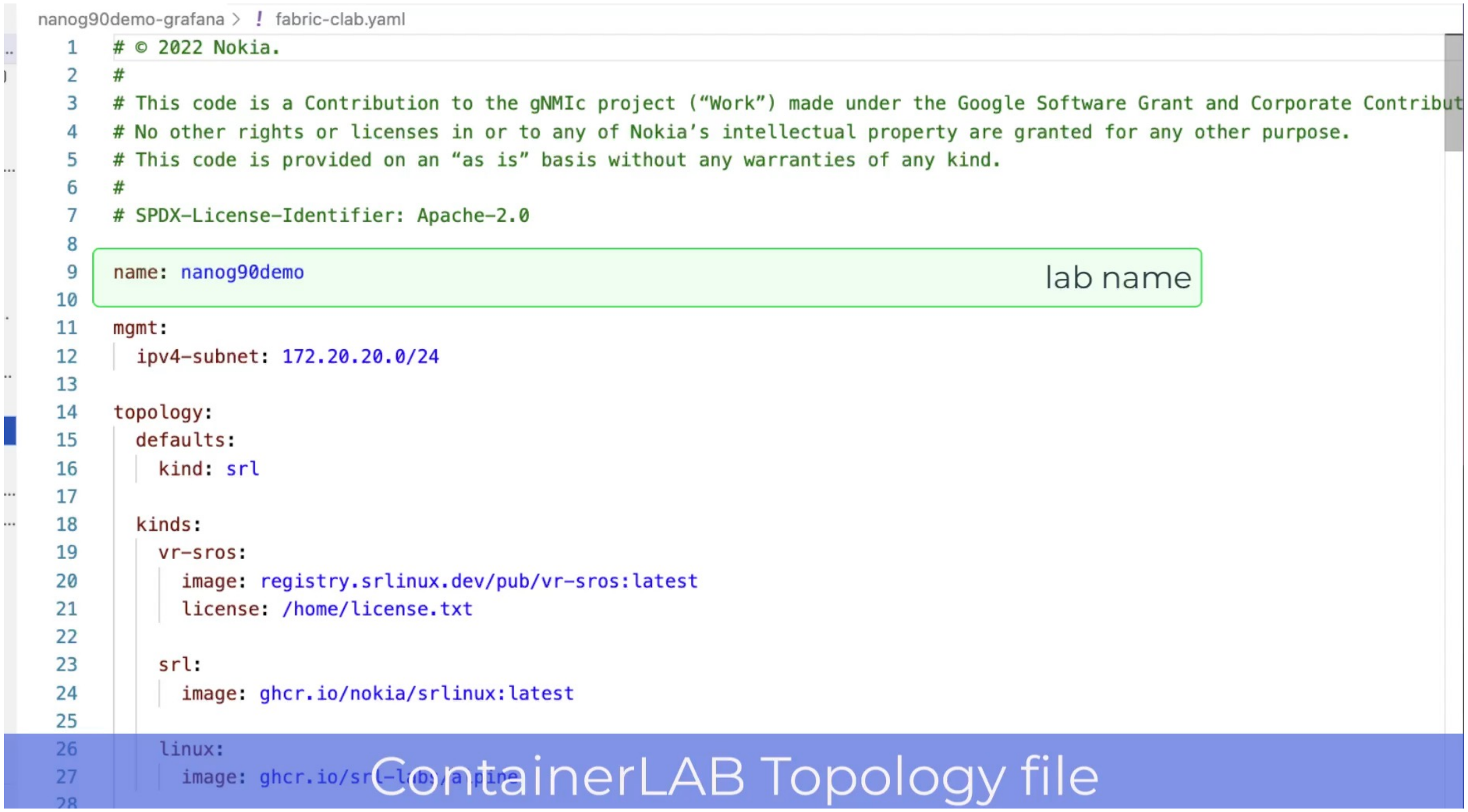


gNMIc deployment examples



NANOG™

fabric-clab.yaml



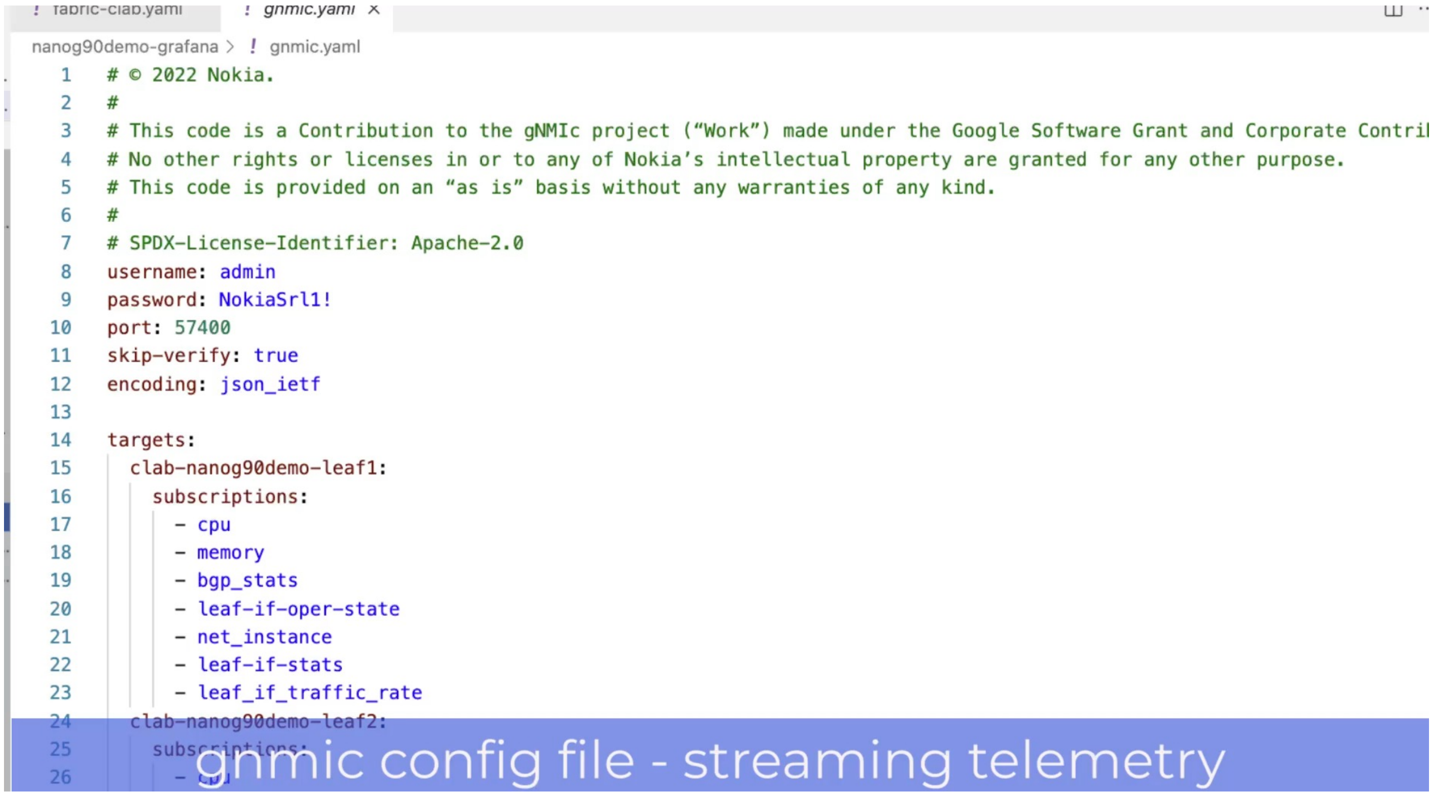
nanog90demo-grafana > ! fabric-clab.yaml

```yaml
# © 2022 Nokia.
#
# This code is a Contribution to the gNMIc project (“Work”) made under the Google Software Grant and Corporate Contribut
# No other rights or licenses in or to any of Nokia’s intellectual property are granted for any other purpose.
# This code is provided on an “as is” basis without any warranties of any kind.
#
# SPDX-License-Identifier: Apache-2.0

name: nanog90demo

mgmt:
  ipv4-subnet: 172.20.20.0/24

topology:
  defaults:
    kind: srl

  kinds:
    vr-sros:
      image: registry.srlinux.dev/pub/vr-sros:latest
      license: /home/license.txt

    srl:
      image: ghcr.io/nokia/srlinux:latest

    linux:
      image: ghcr.io/srl-labs/alpine
```

lab name

ContainerLAB Topology file

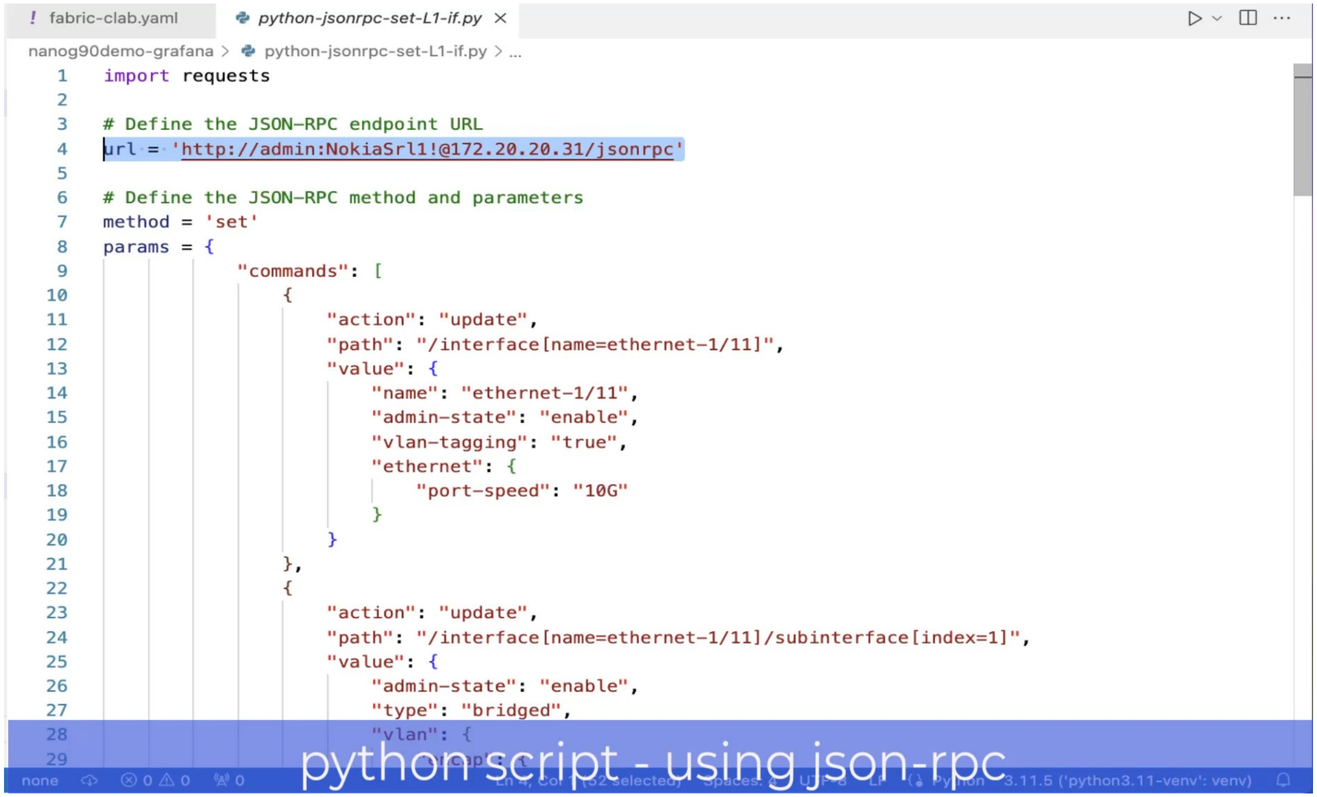
gnmic.yaml



```yaml
# © 2022 Nokia.
#
# This code is a Contribution to the gNMIc project (“Work”) made under the Google Software Grant and Corporate Contri
# No other rights or licenses in or to any of Nokia’s intellectual property are granted for any other purpose.
# This code is provided on an “as is” basis without any warranties of any kind.
#
# SPDX-License-Identifier: Apache-2.0
username: admin
password: NokiaSrl1!
port: 57400
skip-verify: true
encoding: json_ietf

targets:
  clab-nanog90demo-leaf1:
    subscriptions:
      - cpu
      - memory
      - bgp_stats
      - leaf-if-oper-state
      - net_instance
      - leaf-if-stats
      - leaf_if_traffic_rate
  clab-nanog90demo-leaf2:
    subscriptions:
      - cpu
```

gnmic config file - streaming telemetry

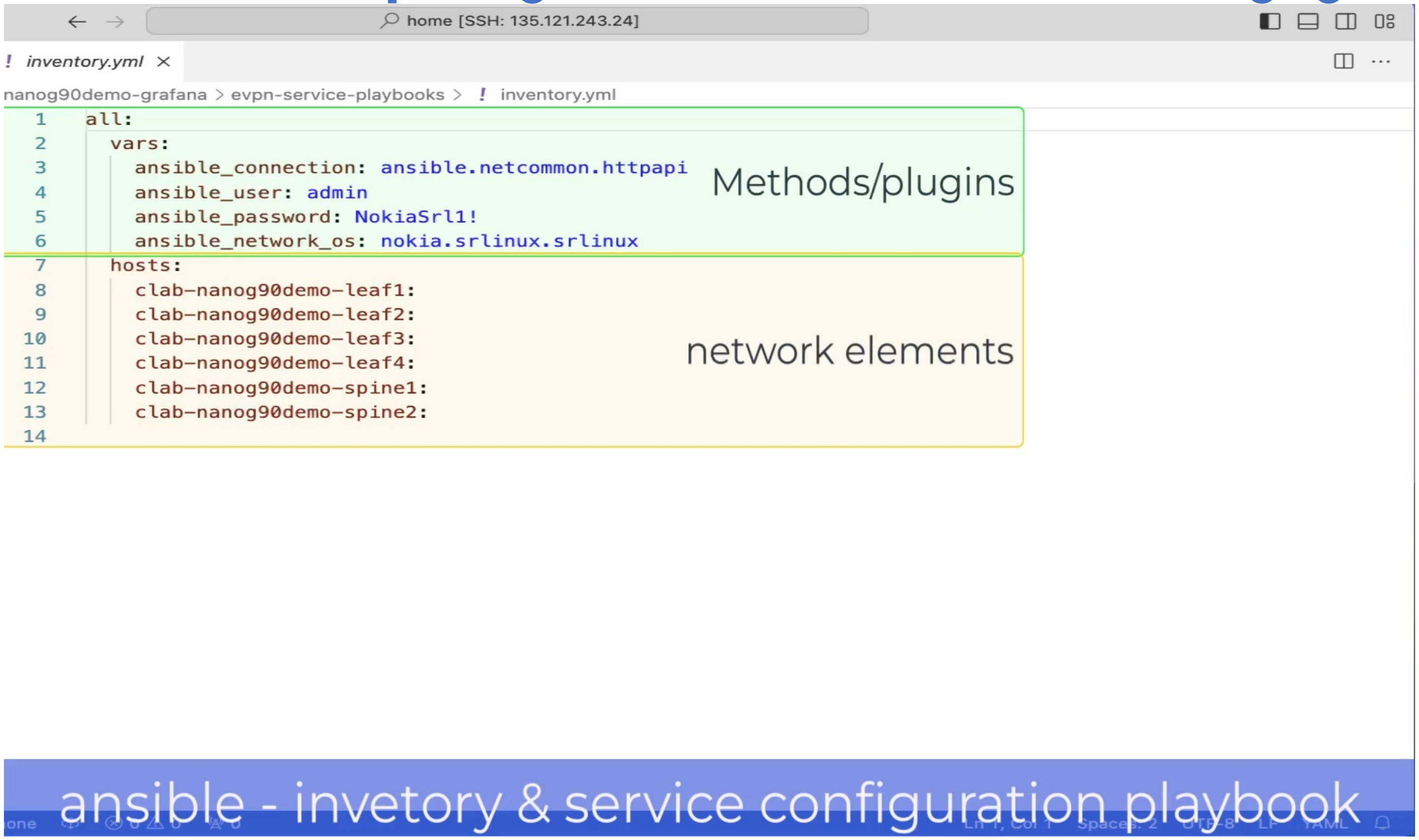
python-jsonrpc-set-L1-if.py


fabric-clab.yaml
python-jsonrpc-set-L1-if.py
nanog90demo-grafana > python-jsonrpc-set-L1-if.py > ...
none ⊗ 0 ⚠ 0 0
Ln 4, Col 1 (52 selected) Spaces: 4 UTF-8 LF Python 3.11.5 ('python3.11-venv': venv)

```python
import requests

# Define the JSON-RPC endpoint URL
url = 'http://admin:NokiaSrl1!@172.20.20.31/jsonrpc'

# Define the JSON-RPC method and parameters
method = 'set'
params = {
            "commands": [
                {
                    "action": "update",
                    "path": "/interface[name=ethernet-1/11]",
                    "value": {
                        "name": "ethernet-1/11",
                        "admin-state": "enable",
                        "vlan-tagging": "true",
                        "ethernet": {
                            "port-speed": "10G"
                        }
                    }
                },
                {
                    "action": "update",
                    "path": "/interface[name=ethernet-1/11]/subinterface[index=1]",
                    "value": {
                        "admin-state": "enable",
                        "type": "bridged",
                        "vlan": {
                            "encap": {
```

python script - using json-rpc

evpn-service-playbooks/inventory.yml


home [SSH: 135.121.243.24]
! inventory.yml
nanog90demo-grafana > evpn-service-playbooks > ! inventory.yml

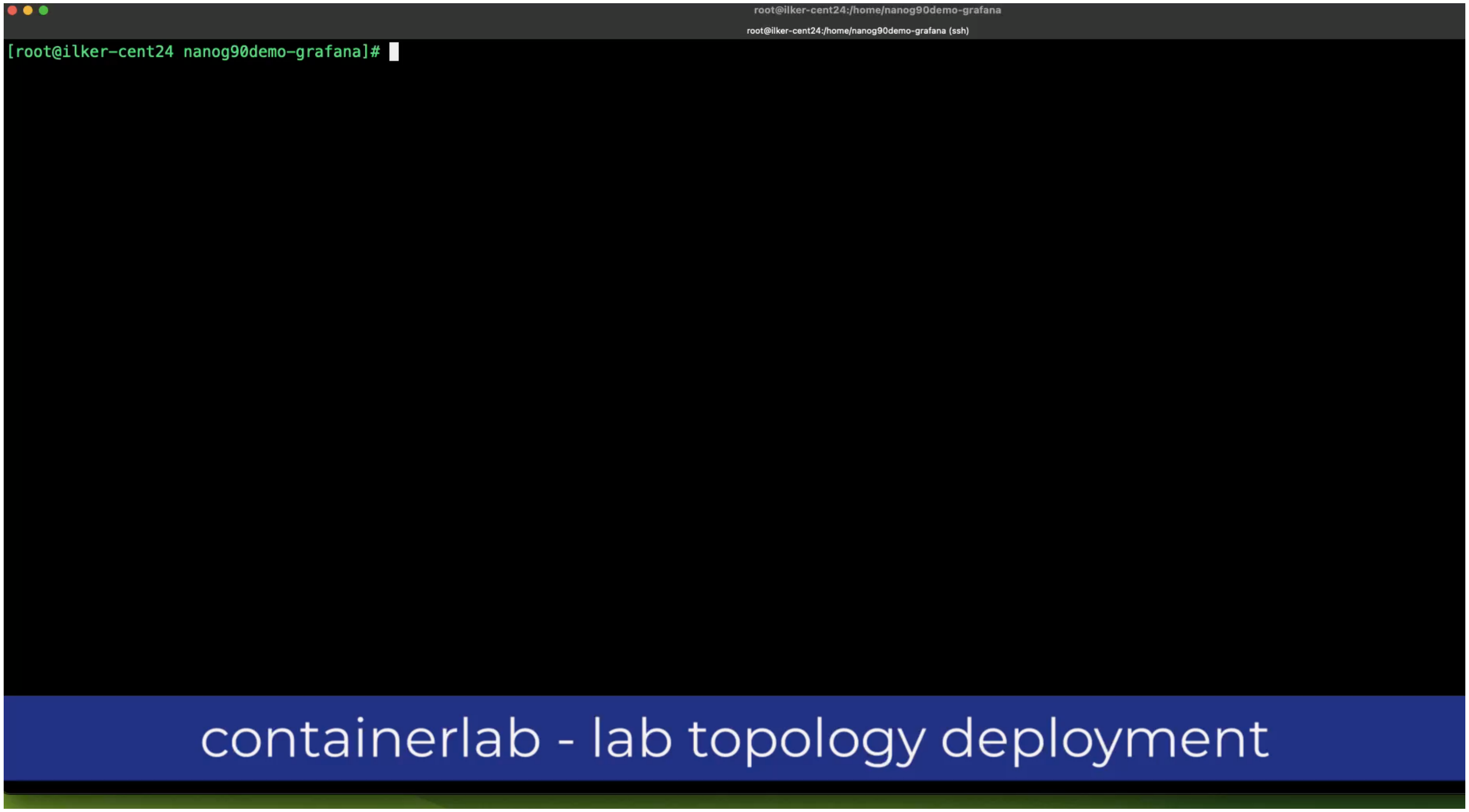
```yaml
all:
  vars:
    ansible_connection: ansible.netcommon.httpapi
    ansible_user: admin
    ansible_password: NokiaSrl1!
    ansible_network_os: nokia.srlinux.srlinux
  hosts:
    clab-nanog90demo-leaf1:
    clab-nanog90demo-leaf2:
    clab-nanog90demo-leaf3:
    clab-nanog90demo-leaf4:
    clab-nanog90demo-spine1:
    clab-nanog90demo-spine2:

```

Methods/plugins

network elements

ansible - invetory & service configuration playbook

Deployment

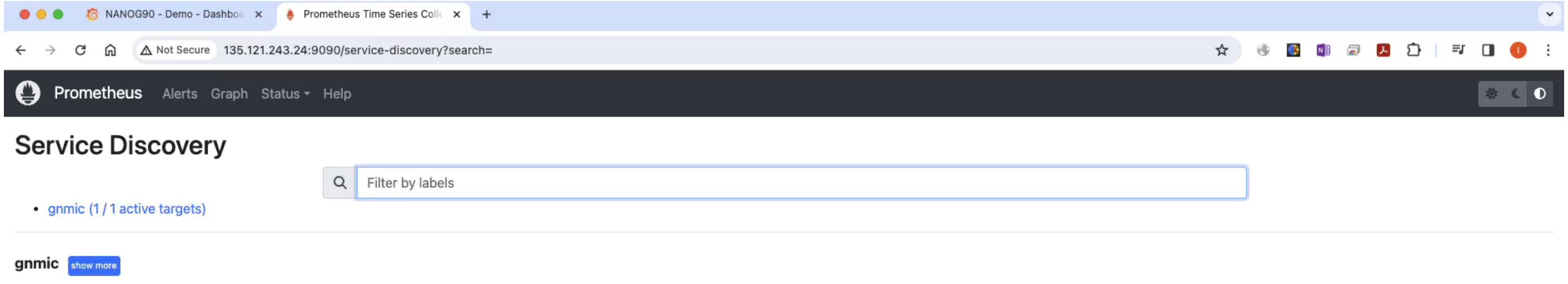


root@ilker-cent24:/home/nanog90demo-grafana
root@ilker-cent24:/home/nanog90demo-grafana (ssh)

```
[root@ilker-cent24 nanog90demo-grafana]# 
```

containerlab - lab topology deployment

grafana / prometheus



Service Discovery

- gnmic (1 / 1 active targets)

gnmic

Thank you

JUNE-2024

NANOG™

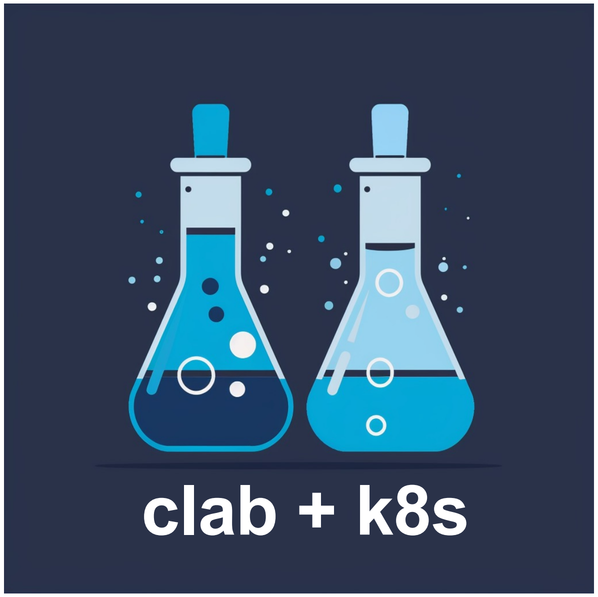
Network Automation in light of Model Driven Management – Continued

June-2024

NANOG™

Clabernetes demo

clabernetes

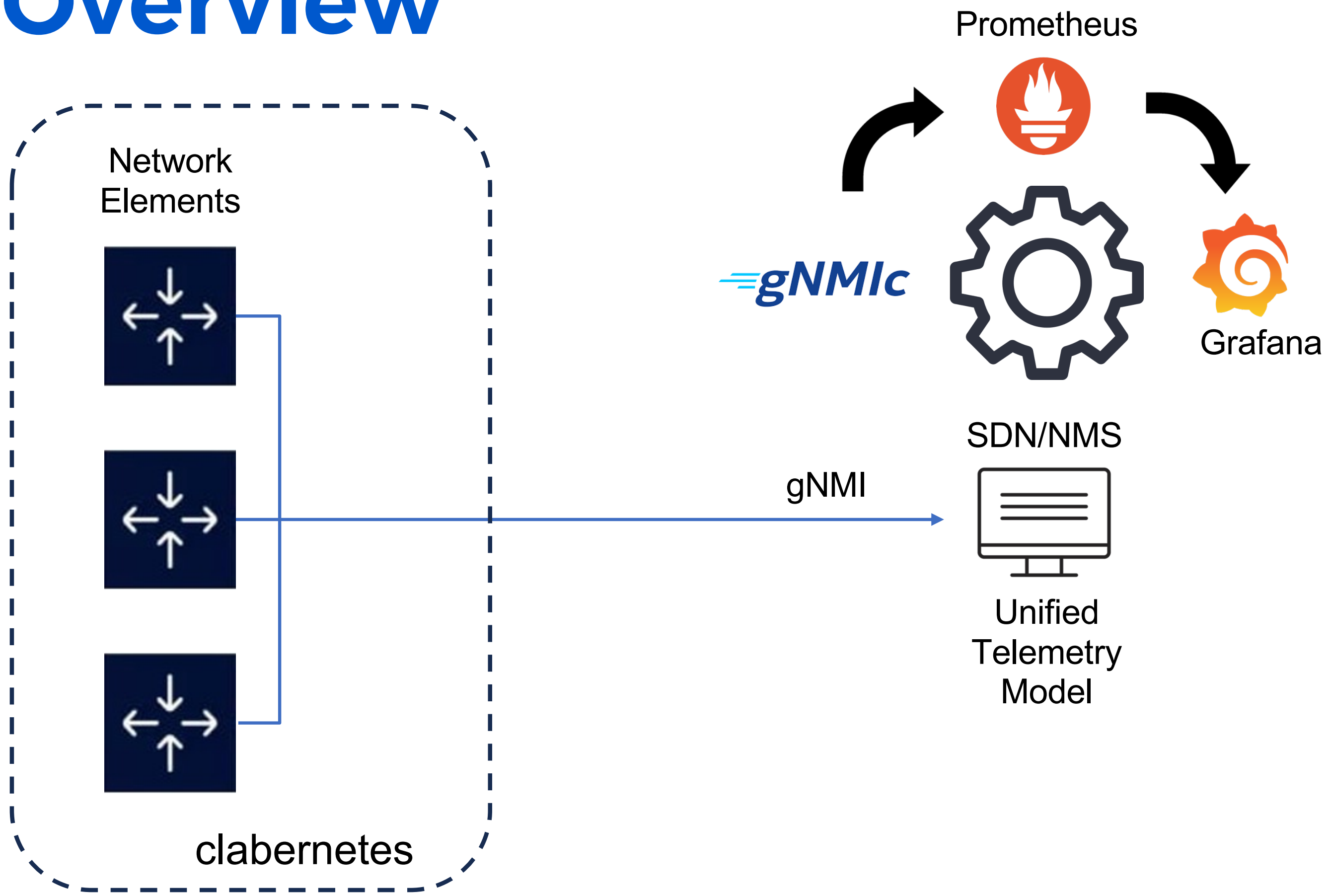


WHY?

clabernetes

- Scale Containerlab beyond a single node while keeping the user experience
- Create large topologies powered by a k8s cluster

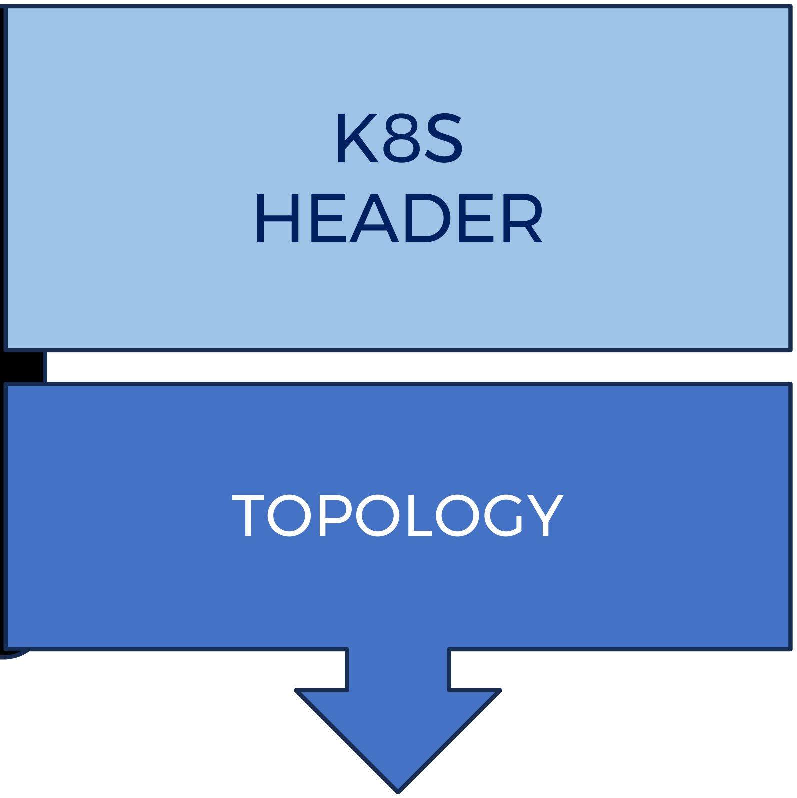
Demo Overview



NANOG™

Topo files >> Manifest

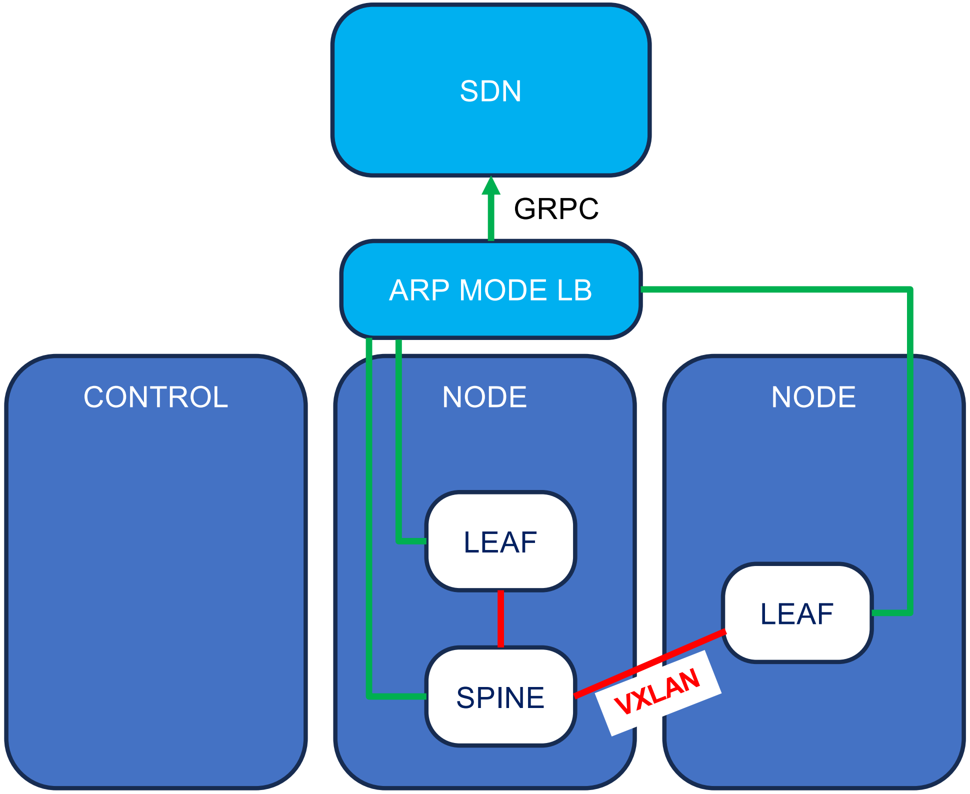
```yaml
kind: Topology
metadata:
  name: clos01
  namespace: c9s-clos01
spec:
  naming: non-prefixed
  connectivity: vxlan
  definition:
    containerlab: |-
        name: clos01
        topology:
          kinds:
            nokia_srlinux:
              image: ghcr.io/nokia/srlinux:23.10.1
```

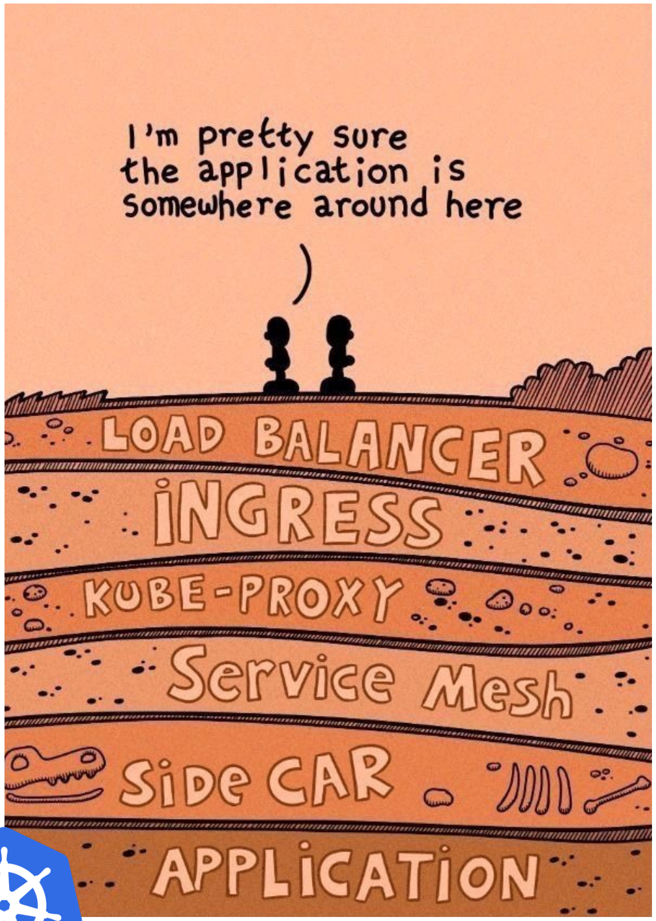


K8S HEADER

TOPOLOGY

Clabernetes Setup





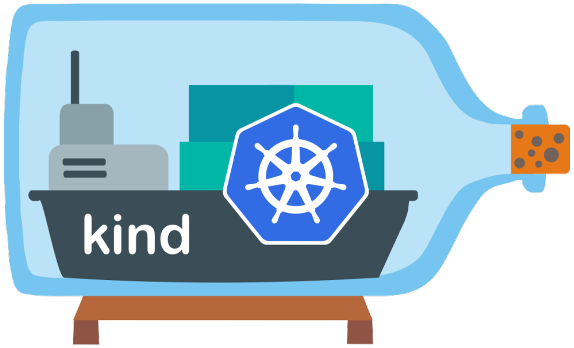
Clabernetes Setup Summary

1. Use an existing Kubernetes Cluster or install a local one using Kind
   - You would need a loadbalancer (ex. kube-vip)
2. Install Helm
3. Install Clabernetes Helm Package
4. Convert existing Clab topo files to K8s Manifests
5. Install manifest

Next slides we'll see how to install C9s into one server (baremetal)

Kind Kubernetes

- Local Kubernetes clusters using Docker container “nodes”.
- Designed for testing Kubernetes.
- Requirements:
  - go (1.17+)
  - Uses docker for node instances



github.com/kubernetes-sigs/kind/

Install K8s Kind

- You can install kind with:

```
go install sigs.k8s.io/kind@v0.18.0
```

- This will put kind in $(go env GOPATH)/bin.
  - You may need to add that directory to your $PATH as shown here if you encounter the error kind: command not found after installation.
- Kind uses docker for node instances
- Once you have docker running you can create a cluster with:

```
kind create cluster
```

Kind config

- For this demo we'll use 2 workers and one controller
- Use "`kind load`" to upload local images (i.e. router) for apps after cluster is created
  - Unless you want to setup a private registry for local images
  - **You can use the public image for clabernetes**

```
# cluster.cfg file
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
containerdConfigPatches:
- |-

[plugins."io.containerd.grpc.v1.cri".containerd]
```

```
kind create cluster \
--name nanog91 --config cluster.cfg
```

Kube-vip Loadbalancer

- To get access to the nodes deployed by clabernetes from outside
- Any load balancer will do, but we will use kube-vip here
- kube-vip provides a virtual IP and load balancer for both the control plane (for building a highly-available cluster) and Kubernetes Services of type LoadBalancer without relying on any external hardware or software

github.com/kube-vip/kube-vip

Kube-vip Install

```
#kube-vip-install.sh
kubectl apply -f https://kube-vip.io/manifests/rbac.yaml
kubectl apply -f https://raw.githubusercontent.com/kube-vip/kube-vip-cloud-
provider/main/manifest/kube-vip-cloud-controller.yaml
kubectl create configmap --namespace kube-system kubevip \
  --from-literal range-global=172.20.5.100-172.20.5.200
KVVERSION=$(curl -sL https://api.github.com/repos/kube-vip/kube-vip/releases | \
  jq -r ".[0].name")
alias kube-vip="docker run --network host \
  --rm ghcr.io/kube-vip/kube-vip:$KVVERSION"
kube-vip manifest daemonset --services --inCluster --arp --interface eth0 | \
kubectl apply -f -
```

Clabernetes install

- Use helm to install clabernetes package

```
sudo helm upgrade --install --create-namespace --namespace c9s  \
   clabernetes oci://ghcr.io/srl-labs/clabernetes/clabernetes
```

- Check if the pods are up and running

```
]$ sudo kubectl get pods -A  | grep clab
c9s                  clabernetes-manager-76fb8b4474-6l2sv            1/1     Running   0          53s
c9s                  clabernetes-manager-76fb8b4474-pwvlr            1/1     Running   0          53s
c9s                  clabernetes-manager-76fb8b4474-w9j2q            1/1     Running   0          53s
```

clabverter

- It takes a containerlab topology file and converts it to several manifests native to Kubernetes and clabernetes

```
#clabverter.sh
alias clabverter='sudo docker run --user $(id -    -v $(pwd):/clabernetes/work --rm \
    ghcr.io/srl-labs/clabernetes/clabverter'
clabverter --topologyFile $1 --naming non-prefixed
```

```
sudo ./clabverter.sh clos01.yml  # clos01-ns.yaml  clos01.yaml
```

K8s Manifest

```
#clos01-ns.yaml
---
apiVersion: v1
kind: Namespace
metadata:
    name: c9s-clos01
```

```
#clos01.yaml
---
apiVersion: clabernetes.containerlab.dev/v1alpha1
kind: Topology
metadata:
  name: clos01
  namespace: c9s-clos01
spec:
  naming: non-prefixed
  connectivity: vxlan
  definition:
    containerlab: |-
        # topology documentation: http://containerlab.dev/lab-examples/min-clos/
        name: clos01
        topology:
# more after this
```

Installing Manifests

- Use kubectl to apply your new manifests

```
sudo kubectl apply -f clos01-ns.yaml
sudo kubectl apply -f clos01.yaml
```

- Check if they are working

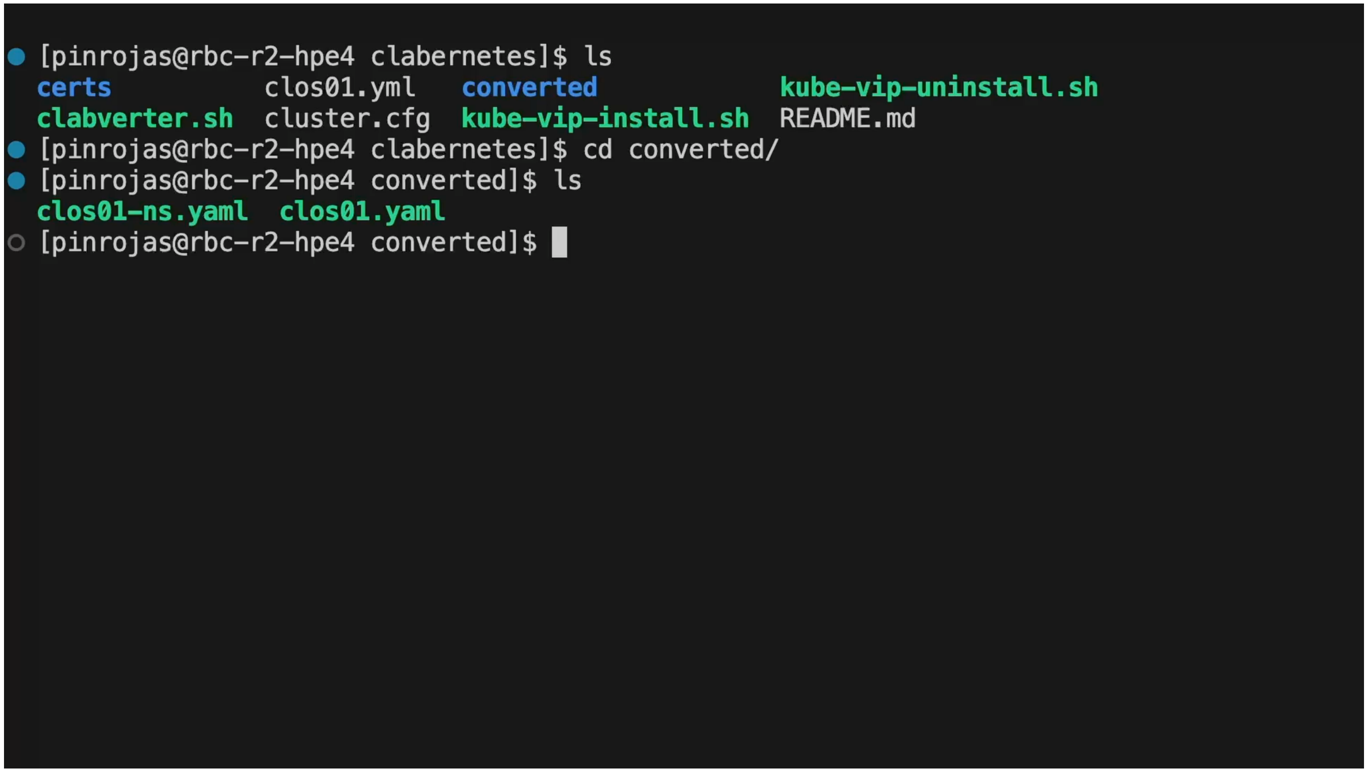
```
sudo kubectl get pods -A  | grep clos
c9s-clos01          client1-79d5bbf46b-gz9ch                        1/1     Running   0              7m34
c9s-clos01          client2-569f6c88b-lsjzn                         1/1     Running   0              7m34
c9s-clos01          leaf1-d9b44d76f-2wztq                           1/1     Running   0              7m34
c9s-clos01          leaf2-5556c7b49-26275                           1/1     Running   0              7m34
c9s-clos01          spine-7464fddbb5-8xtfx                          1/1     Running   0              7m34
```

Connect to net devices

- You can connect from an external source

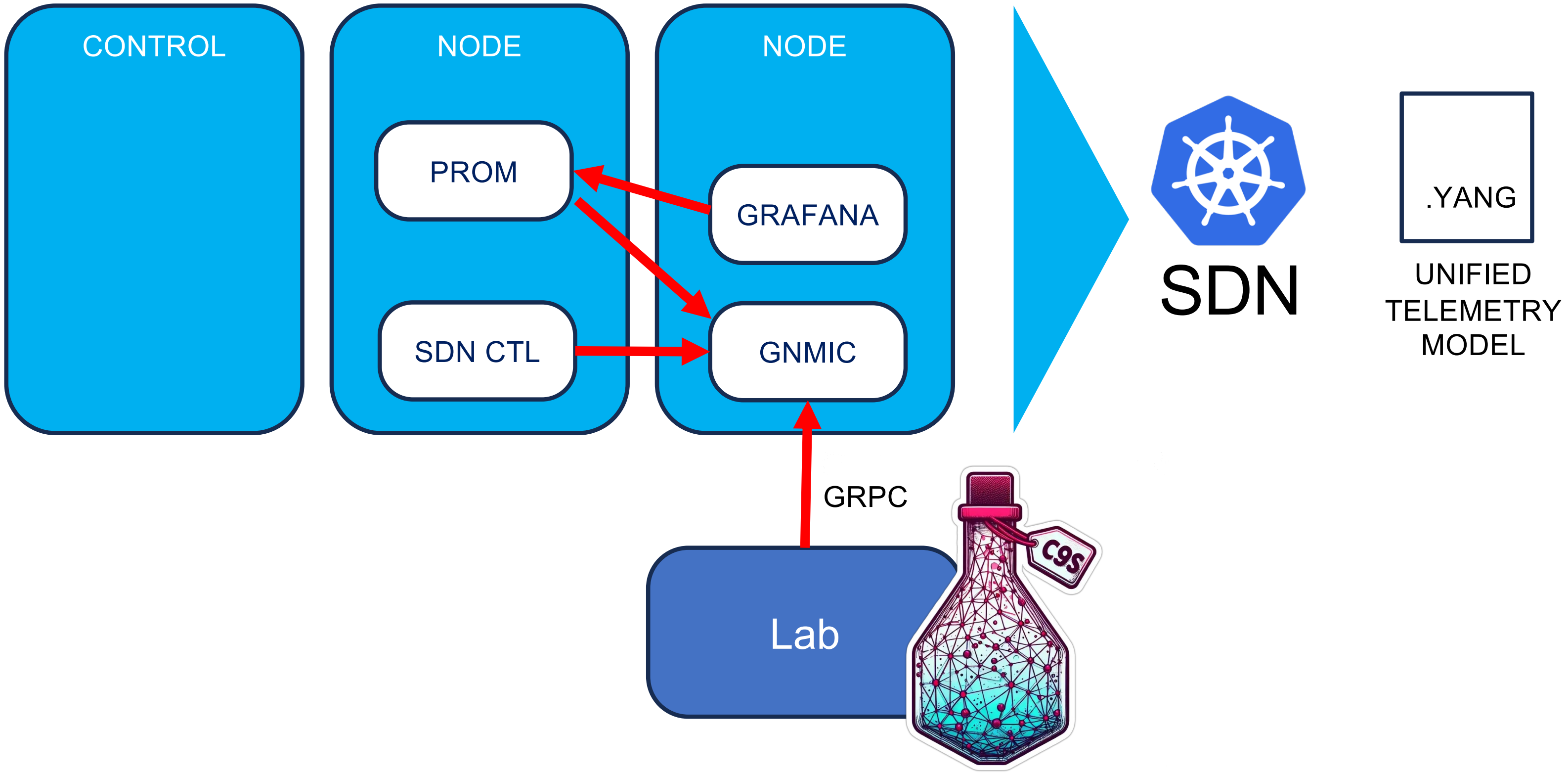
```
sudo kubectl get svc -n c9s-clos01
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)
AGE
client1      LoadBalancer   10.96.100.8    172.20.5.103
161:31307/UDP,21:30702/TCP,22:31514/TCP,23:30563/TCP,80:32216/TCP,443:32680/TCP,830:32186/TCP,5000:
31054/TCP,5900:31798/TCP,6030:32544/TCP,9339:31881/TCP,9340:30928/TCP,9559:30762/TCP,57400:30013/TC
P   10m
client2      LoadBalancer   10.96.239.15   172.20.5.104
161:30920/UDP,21:30308/TCP,22:31693/TCP,23:31082/TCP,80:30671/TCP,443:30931/TCP,830:32454/TCP,5000:
30316/TCP,5900:30071/TCP,6030:32355/TCP,9339:30703/TCP,9340:31799/TCP,9559:30344/TCP,57400:31481/TC
P   10m
```

Clabernetes Setup

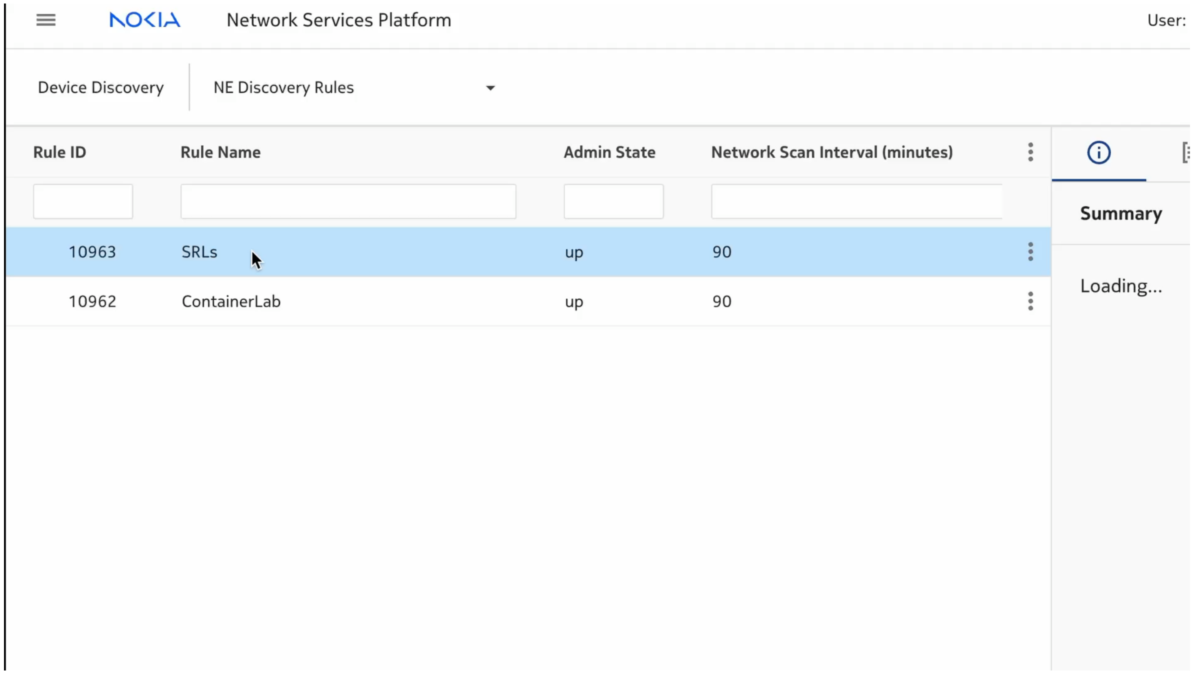


```
● [pinrojas@rbc-r2-hpe4 clabernetes]$ ls
  certs          clos01.yml   converted            kube-vip-uninstall.sh
  clabverter.sh  cluster.cfg  kube-vip-install.sh  README.md
● [pinrojas@rbc-r2-hpe4 clabernetes]$ cd converted/
● [pinrojas@rbc-r2-hpe4 converted]$ ls
  clos01-ns.yaml  clos01.yaml
○ [pinrojas@rbc-r2-hpe4 converted]$ 
```

SDN Setup


CONTROL
NODE
NODE
PROM
SDN CTL
GRAFANA
GNMIC
GRPC
Lab
C9S
SDN
.YANG
UNIFIED
TELEMETRY
MODEL

SDN Setup



Network Services Platform

User:

Device Discovery | NE Discovery Rules

Rule ID	Rule Name	Admin State	Network Scan Interval (minutes)
10963	SRLs	up	90
10962	ContainerLab	up	90

Summary

Loading...

Additional Info

- clabernetes: https://containerlab.dev/manual/clabernetes/
- containerlab https://containerlab.dev/
- Kubernetes 101 for Network Professionals
  - https://youtu.be/n2kgApcXij0

Thank you

JUNE-2024

NANOG™