narrowIN

Building a Digital Twin

with MikroTik CHR Containerlab, and Ansible

Tomáš Horyl
Senior Network Engineer
narrowin.ch

Who?

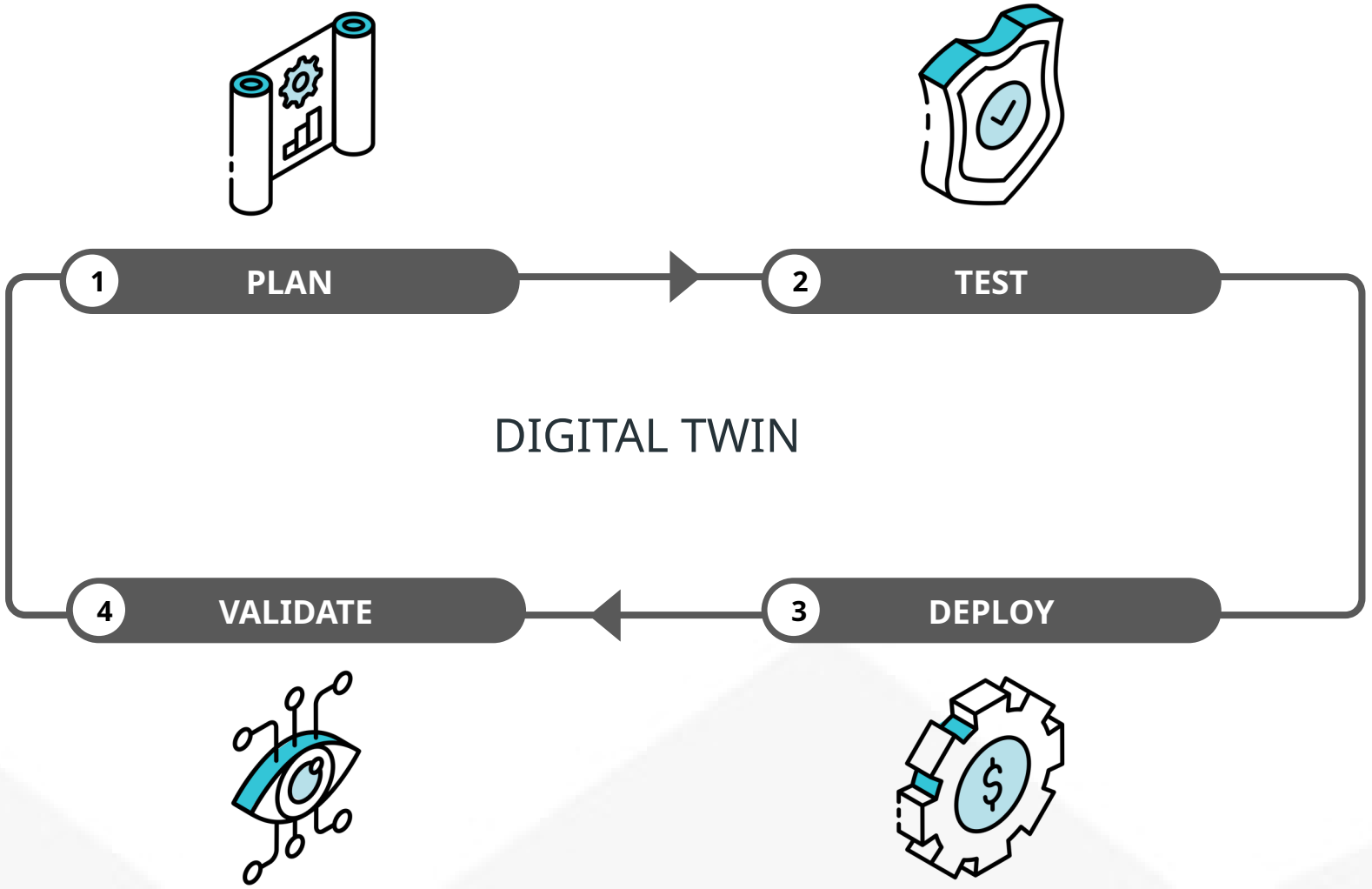
Tomáš Horyl

- Network design and development
- Linux system administration
- Computer and network infrastructure maintenance

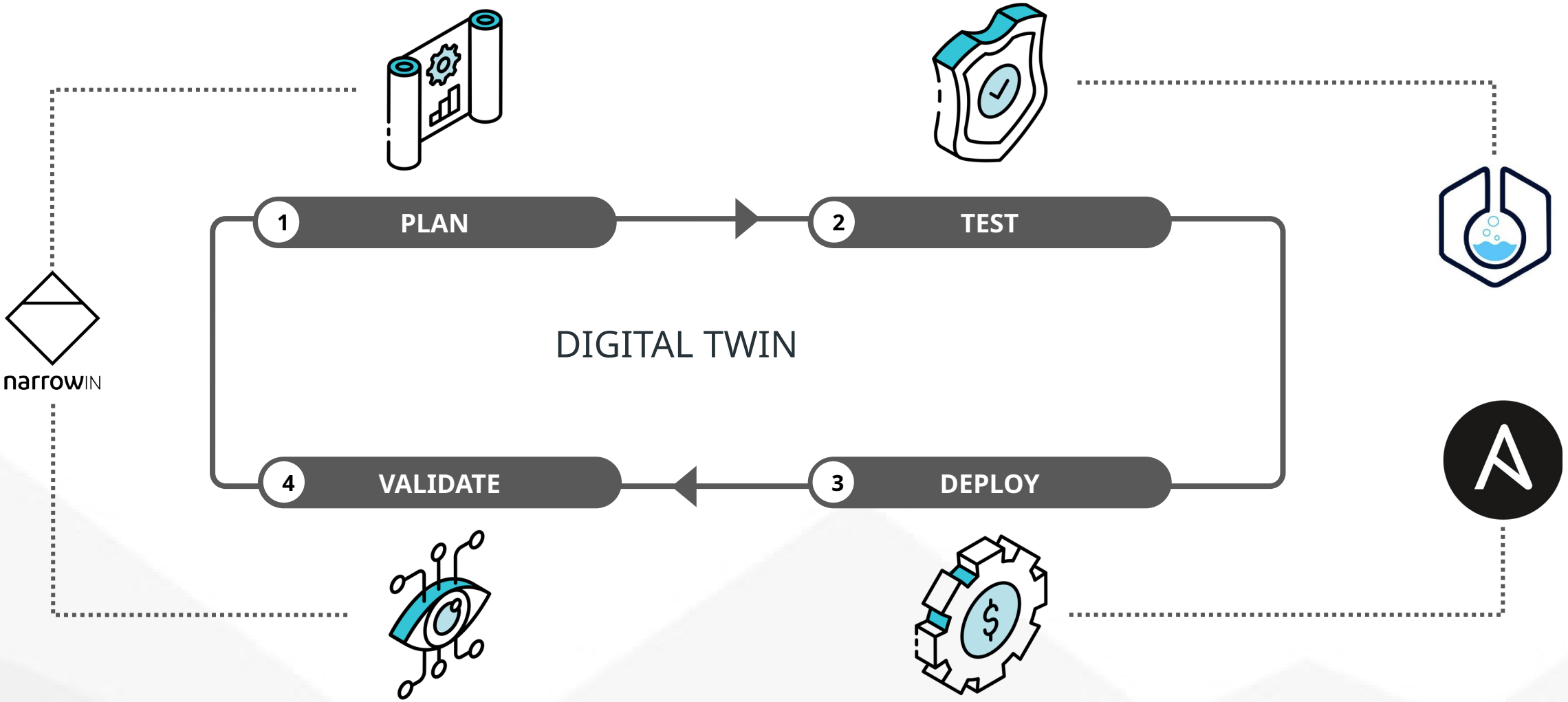
narrowin.ch

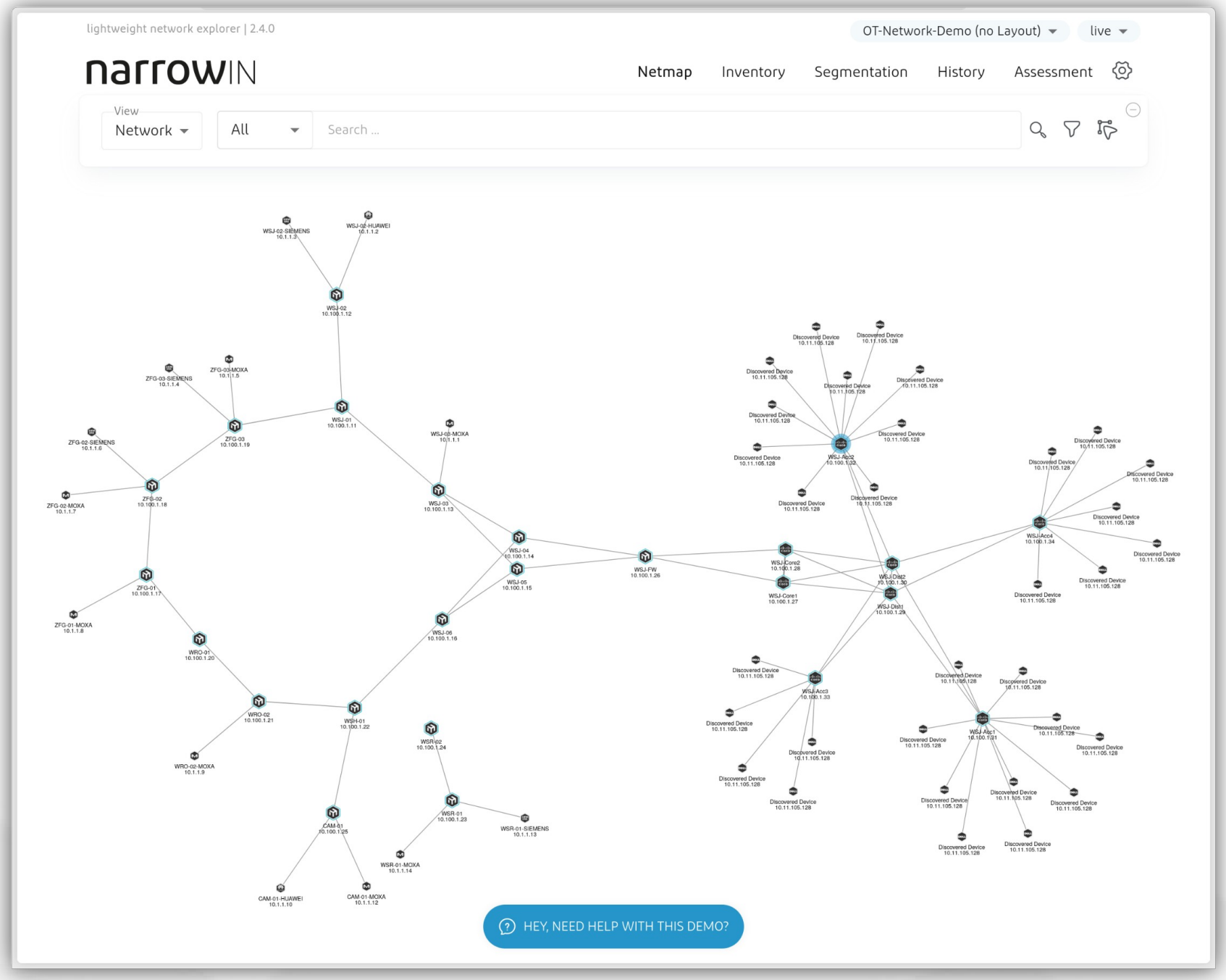
- Swiss university spin-off
- Networking and security
- Lightweight Network Explorer

Why? Where we want to go...



Why? Where we want to go...

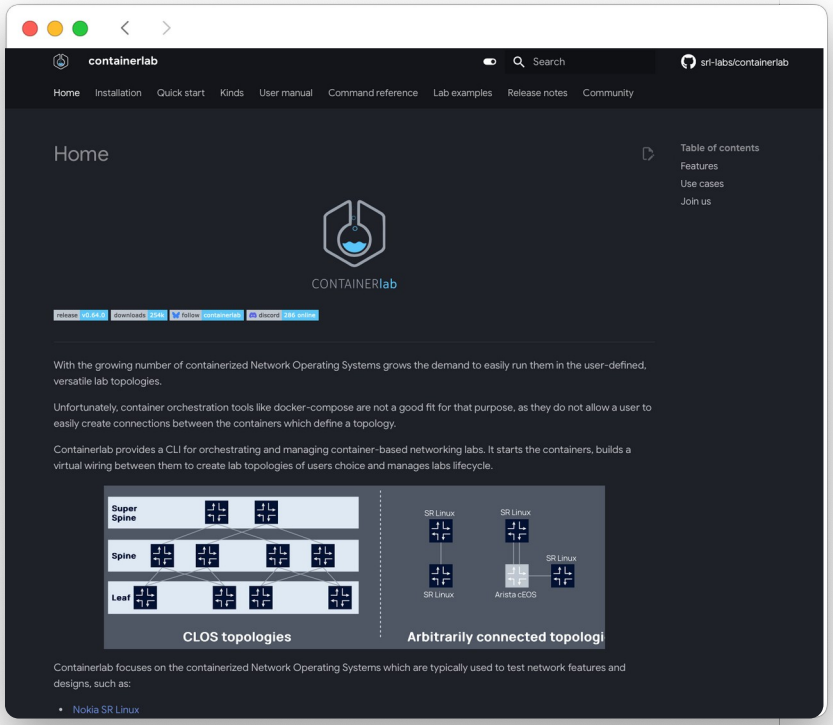



lightweight network explorer | 2.4.0
OT-Network-Demo (no Layout)
live
narrowIN
Netmap
Inventory
Segmentation
History
Assessment
View
Network
All
Search ...
WSJ-02-SIEMENS
10.1.1.3
WSJ-02-HUAWEI
10.1.1.2
WSJ-02
10.100.1.12
ZFG-03-SIEMENS
10.1.1.4
ZFG-03-MOXA
10.1.1.5
WSJ-01
10.100.1.11
ZFG-03
10.100.1.19
ZFG-02-SIEMENS
10.1.1.6
WSJ-03-MOXA
10.1.1.1
ZFG-02-MOXA
10.1.1.7
ZFG-02
10.100.1.18
WSJ-03
10.100.1.13
WSJ-04
10.100.1.14
WSJ-05
10.100.1.15
WSJ-FW
10.100.1.26
ZFG-01
10.100.1.17
ZFG-01-MOXA
10.1.1.8
WSJ-06
10.100.1.16
WRO-01
10.100.1.20
WRO-02
10.100.1.21
WSH-01
10.100.1.22
WRO-02-MOXA
10.1.1.9
WSR-02
10.100.1.24
WSR-01
10.100.1.23
WSR-01-SIEMENS
10.1.1.13
CAM-01
10.100.1.25
WSR-01-MOXA
10.1.1.14
CAM-01-HUAWEI
10.1.1.10
CAM-01-MOXA
10.1.1.12
WSJ-Acc2
10.100.1.32
WSJ-Acc4
10.100.1.34
WSJ-Core2
10.100.1.28
WSJ-Core1
10.100.1.27
WSJ-Dist2
10.100.1.30
WSJ-Dist1
10.100.1.29
WSJ-Acc3
10.100.1.33
WSJ-Acc1
10.100.1.31
Discovered Device
10.11.105.128
HEY, NEED HELP WITH THIS DEMO?

Introducing Containerlab

https://containerlab.dev

- Containerized network operating systems (NOS)
- Can also launch traditional virtual machine-based routers
- Can interconnect arbitrary Linux containers
- Runs network operating systems in containers (Docker/Podman)
- Linux network namespaces



- ✓ Ideal solution for test environments
- ✓ Runs network OSes in omnipresent containers
- ✓ Covers lots of major vendors
- ✓ Easy topology definition (text based - scriptable).

«Containerlab provides a CLI for orchestrating and managing container-based networking labs. It starts the containers, builds a virtual wiring between them to create lab topologies of users' choice and manages labs lifecycle.»

Containerlab: How does the topology file look like?

```yaml
name: mylab
topology:
  nodes:
    mkt1:
      kind: mikrotik_ros
      image: vrnetlab/mikrotik_routeros:7.16.2
    mkt2:
      kind: mikrotik_ros
      image: vrnetlab/mikrotik_routeros:7.16.2
  links:
    - endpoints: ["mkt1:ether2", "mkt2:ether2"]
```

`containerlab deploy` deploy the topology (start the lab).

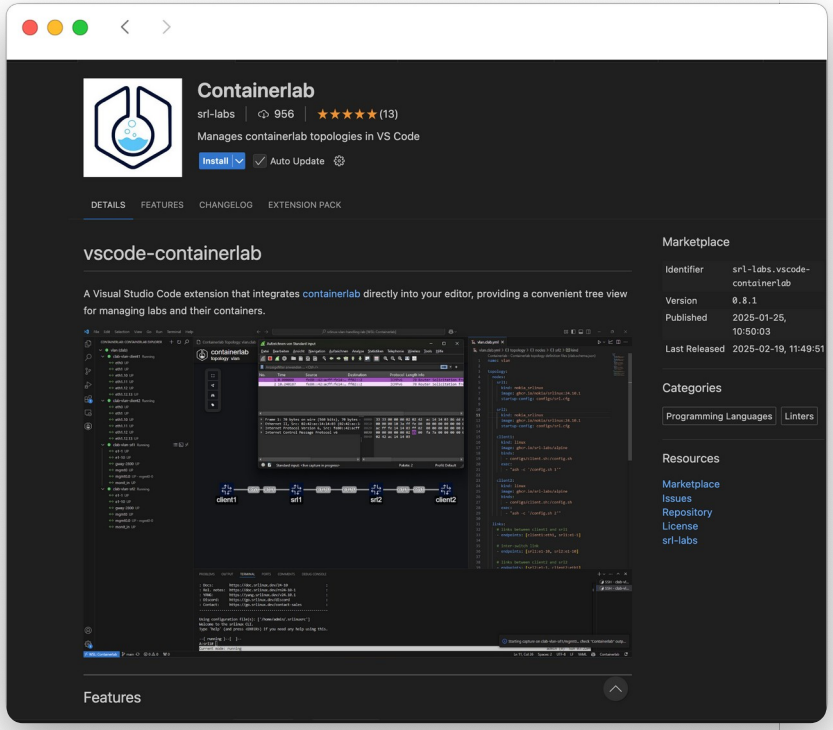
`containerlab destroy` shut down the lab.

`ssh clab-mylab-mkt1` connect to the node.

Containerlab creates static entries in the `/etc/hosts` file and sets up `/etc/ssh_config.d/` to allow you to use SSH.

… and there‘s a helpful VSCode extension

Simplified workflow for almost everything from the command line.
Useful even for network engineers – like me – who are more accustomed to working in a CLI-driven environment ;-)



Features:

- Lab explorer: Real-time monitoring of lab status, including nodes and links.
- Lab Editor: topology modifications within VS Code environment.
- TopoViewer: visual representation of the lab setup.
- Packet Capture: Wireshark integration, capture traffic on a selected link.
- Direct CLI Access: connect to node consoles.
- Link Impairment Tuning: simulation of network delays, packet loss, etc.

but CHR can‘t be containerized, right…?

https://github.com/hellt/vrnetlab

- Many routing network operating systems cannot be containerized and can only run as virtual machines.
- With vrnetlab integration, Containerlab is capable of launching topologies with VM-based routers within the same topology definition file, alongside containerized NOS.

Important: Containerlab uses original vrnetlab project fork hellt/vrnetlab. Container built with upstream vrnetlab project will not be compatible with Containerlab.

Introducing Ansible: An agentless automation tool

Manual Deployment

..of configurations across multiple devices is time-consuming, error-prone, inconsistent and inefficient.

Automated Deployment

Automate tasks efficiently & free up time for critical operations

- Predefined configuration templates
- Deploying changes with a single command
- Control node can be any Linux-based system with Python installed, including Windows WSL.

ANSIBLE

Detailed installation instructions: https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

How Ansible Handle Network OSes

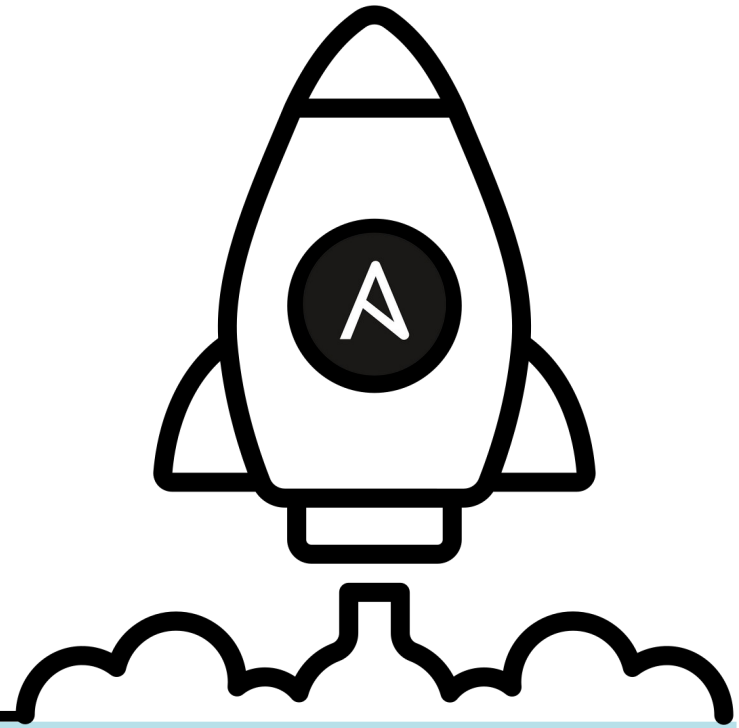
Network OSes with Python interpreter available:

- Ansible uses SSH or API to connect.
- Copies Python modules to the remote device and executes them locally on the device.
- Full Ansible module support.

MikroTik RouterOS without Python support:

- Ansible cannot copy Python scripts to execute them on the host.
- Instead, it relies on API or CLI commands.
- Command (community.routeros.command module):
  - Human-readable, works over SSH.
  - Not structured, parsing output is complex.
- API (community.routeros.api module):
  - Structured and machine-readable output (no need for complex output parsing).
  - Faster processing for batch operations.
  - Requires enabling API service (/ip service enable api).
  - Not all CLI features are available.

Getting started with Ansible



Getting started can feel overwhelming and complex at first…
but once you get started it’s straightforward – I promise!

get kick-started:

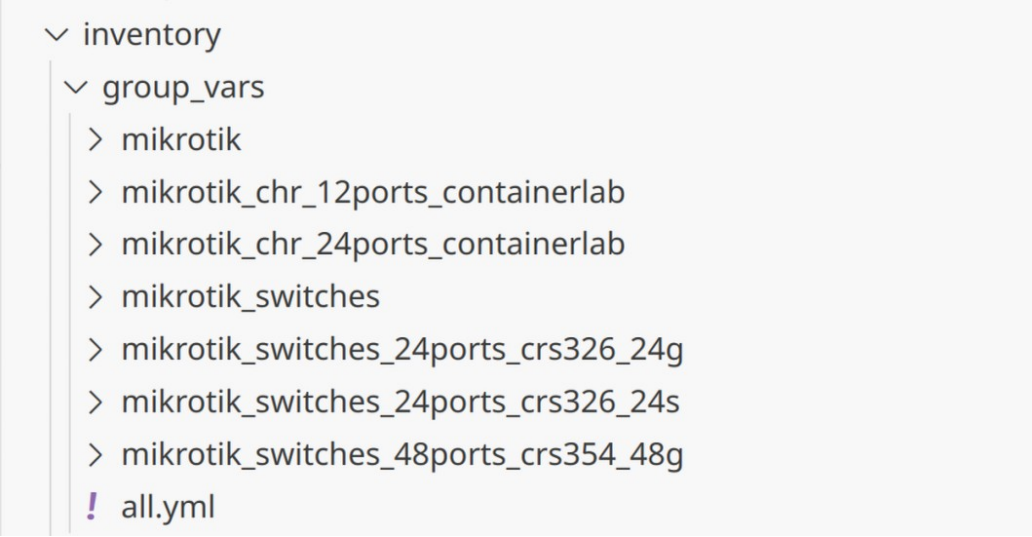
https://github.com/narrowin/ansible-mikrotik

- Clone the repository.
- Install requirements: `pip install -r requirements.txt`.
- Get Ansible Galaxy collections: `ansible-galaxy collection install -r requirements.yml -p collections`
- Or you can try the repo in self-contained DevContainers (DevPod) or even in GitHub CodeSpaces.

Ansible Inventory

- Inform Ansible where the devices are and how to connect to them.
- Build inventory file with management IP addresses of network nodes, stored in `inventory/mikrotik`. The node/network device in Ansible is referred to as a host.
- Modify credentials and SSH keys in `inventory/mikrotik`.
- Use inventory groups variables (`inventory/group_vars`) and host-specific variables (`inventory/host_vars`) to define device settings.

Groups:



1st example: Ansible Backup Playbook

- Backing up device configurations is the first crucial step in automation—anyone who has had to restore a device knows its importance. And it’s also a good starting step.
- The backup process leverages the `community.routeros.command` module, which is ideal for running commands and retrieving output, though it lacks idempotency for configuration management.
- The backup playbook requires only:
- The Ansible host, provided by Ansible as `inventory_hostname`
- The backup directory `local_backups_top_folder`, defined in `inventory/group_vars/all.yml`

```
...
- name: "Starting ROS Configuration Backup to memory"
  community.routeros.command:
    commands: "/export show-sensitive terse file={{ inventory_hostname }}.cfg.backup"

- name: "Copy ROS config backup to ansible control host"
  ansible.netcommon.net_get:
    src: "{{ inventory_hostname }}.cfg.backup.rsc"
    dest: "{{ local_backups_top_folder }}/mikrotik/{{ inventory_hostname }}/{{ inventory_hostname }}.cfg.backup.rsc"

- name: "Delete backup file from memory"
  community.routeros.command:
    commands: "/file/remove {{ inventory_hostname }}.cfg.backup"
...
```

2nd example: Step-By-Step Hosts Onboarding - Hostname

- Transition to the `community.routeros.api` module for configuration tasks, as it ensures idempotency.
- Why is idempotency important?
  - Idempotency is a key principle in Ansible automation that ensures running the same playbook multiple times produces the same result, regardless of how many times it is executed.
  - Idempotency ensures that the outcome is always repeatable and predictable (e.g. nobody wants to duplicate already existing firewall rules by appending to them).
- Start with a simple task - hostname checking.

```
...
    - name: Hostname
      community.routeros.api_modify:
        path: system identity
        data: "{{ routeros_system_identity }}"
      when:
        - routeros_system_identity is defined
      tags: hostname
...
```

`routeros_system_identity` – `inventory/group_vars/mikrotik/system_identity.yml`

```
ansible-playbook playbooks/mikrotik-configure.yml --tags hostname --check --diff
```

3rd example Ansible - Onboarding - Interfaces (1/2)

- `/interface ethernet`
- The difference from the hostname - variable number of ports.

```
...
    - name: Hostname
      community.routeros.api_modify:
        path: system identity
        data: "{{ routeros_system_identity }}"
      when:
        - routeros_system_identity is defined
      tags: hostname
...
```

- `routeros_interface_ethernet` - in `inventory/group_vars` on various levels and `inventory/host_vars` for device specific configuration.
- Common values set for all grouped devices on higher level (`group_vars`).
- Device specific values overwritten in `host_vars`.

3rd example Ansible - Onboarding - Interfaces (2/2)

- Before deploying the configuration, perform a dry-run to preview the changes without applying them.
- `ansible-playbook playbooks/mikrotik-configure.yml --tags ethernet_ports --check --diff`
- Review the diff output, adjust values as needed.
- Repeat the process until the expected configuration is achieved.
- Once satisfied, deploy the final configuration without `--check --diff`

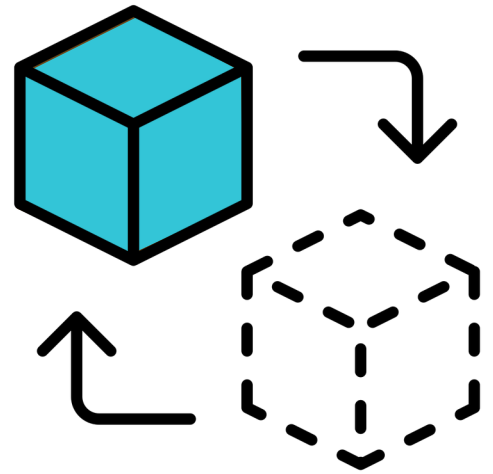
Live Demo / Screencast

of this process

Live Demo / Screencast

Final Network-Wide Deployment

Ansible - How To Transform Real Network Into Digital Twin and vice-versa

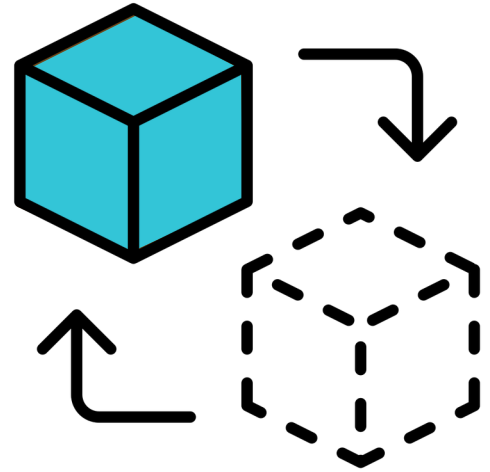


- Start with the inventory (how to reach the devices).
- It is always good practice to start with backup (leverage Ansible to get your devices' backups).
- Take small steps when onboarding existing devices:
  Limit your ground by doing dry-runs (`--check`) on limited number of hosts (`--limit`)
  and with specific features (`--tags`).

Example:

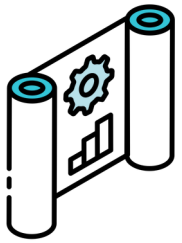
- Let us start with `hostname` feature only on `sw-acc-01`.
- `ansible-playbook playbooks/mikrotik-configure.yml --tags hostname --check --diff --limit sw-acc-01`
- Review the diff, change the variables accordingly and deploy the change.
- `ansible-playbook playbooks/mikrotik-configure.yml --tags hostname --limit sw-acc-01`
- Once this works, do the dry-run for all switches or another manageable group of hosts.
- `ansible-playbook playbooks/mikrotik-configure.yml --tags hostname --check --diff --limit mikrotik_switches`
- Review the diffs, ...
- Repeat for all desired configuration features until you achieve your goal.

Going Beyond - Incorporating Ansible In Your Workflows

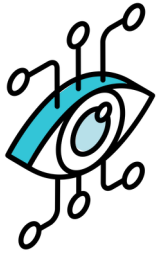


- New device provisioning.
  - Zero-Touch Provisioning - automate device bring-up with minimal steps.
  - Baseline configuration - management IP, SSH keys - ensuring the device is accessible by Ansible.
- Full device configuration.
  - All common features (applicable to all network devices - automatically inherited by being part of group `mikrotik`).
  - Platform specific features inherited by being assigned to the correct platform group `mikrotik_switches/mikrotik_siwtches_24ports_crs326_24g/`...
  - Host specific features assigned in `host_vars`.
- Regular configuration backups (text and binary formats), preferably integrated with some version control system (Git).
- Ad-hoc firmware upgrades (TBD).

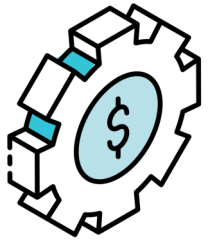
Conclusion



Test-before-deploy approach in critical networks



Centralized Network Source of Truth



Full-cycle automation: design-test-deploy-observe