Building Containerlab with cEOS

Workshop

How to build a lab environment with Containerlab and cEOS-lab

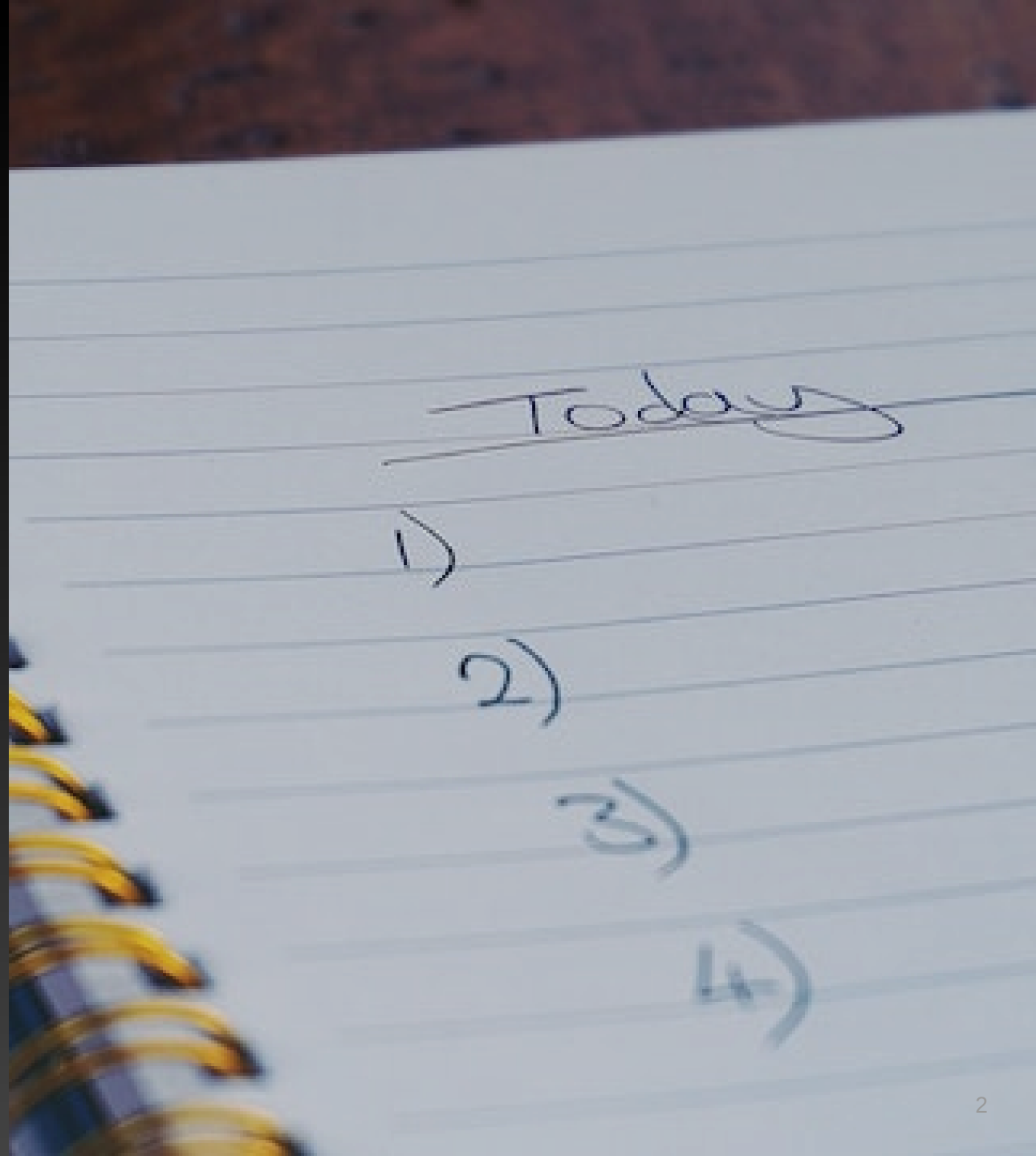
Petr Ankudinov, 2024

CONTAINERlab

Agenda

- Setup Docker on the host
- Install Containerlab and import cEOS-lab image
- Deploy the lab
- Inspect and destroy the lab
- Deploy the lab with a custom startup config
- Make a packet capture
- cLab in a Container
- Possible caveats

> This workshop is a step-by-step guide explaining how to build a lab environment with Containerlab and Arista cEOS-lab. It is focusing on essential and cEOS-lab specific features. Please check Containerlab documentation for more details.



How To Run The Workshop

To run the labs in this workshop, you can use one of the following options:

- Build you own Ubuntu VM from scratch. (Recommended!)
  - This option will allow you to experience the entire cLab enviroment build process from the very start, without any pre-installed dependancies
  - Requirements
    - Ubuntu LTS 22.04 or later
    - 8 GB RAM and 4 vCPUs
- Start the Github Codespace from this repository. (Fastest!)
- Open the lab dev container locally on your laptop with Docker Desktop or sever with Docker CE:
  - you can download required files here and open them in VSCode
  - when running the lab locally, you must set all required environment variables on your machine

CPU Architechture

- Only x86 CPU architecture is supported!
- It is technically possible to run Container lab on ARM, but there are no network device images available for ARM as of May 2024.

Setup Docker on the Host

> Check if Docker is already installed. In this case you can skip the steps below.

1. Install Docker on the host. The detailed instructions are available here. You can used one-liner script for that.
2. Add your user to the `docker` group.
3. Logout and login again to apply the changes.
4. Check the Docker version and run `hello-world` container to test functionality.
5. You must be able to run `docker` commands without `sudo` if it was installed correctly.

```
# install Docker
sudo curl -fsSL https://get.docker.com | sh
# add user to the docker group
sudo usermod -aG docker ${USER}
# test docker
docker --version
docker run hello-world
```

> **NOTE**: If you are running this workshop in Codespace or provided dev container, Docker is pre-installed. As the workshop magic happens inside a container in this case, we rely on Docker-in-Docker concept to provide required functionality.

Setup Git (Optional)

- Git must be pre-installed on a Linux system. Otherwise you are in a wrong place. Escape! 👾 🚀
- Setup your name and email address:

```
git config --global user.name "<first-and-2nd-name>"
git config --global user.email "<your-email>"
```

- Check the current configuration:

```
git config --list
```

> **NOTE**: On Codespace Git is pre-installed and pre-configured.

Download cEOS-lab Image

1. Login to Arista Software Download portal. You need to have an account to download the image.
2. Select `EOS > Active Releases > 4.30 > EOS-4.30.6M > cEOS-lab`.
3. Download `cEOS-lab-4.30.6M.tar.xz` image.
4. Upload the image to your lab VM. For example, you can use SFTP to transfer the image:

```
sftp ${REMOTE_USER}@${UBUNTU_VM_IP}:/home/${REMOTE_USER}/${IMAGE_DIR} <<< $'put cEOS-lab-4.30.6M.tar*'
# for example:
# sftp user@10.10.10.11:/home/user/images <<< $'put cEOS-lab-4.30.6M.tar*'
```

> If Github Codespace or provided Dev Container is used and the Arista token is set, the image will be pulled from arista.com automatically.

Import cEOS-lab Image

1. Go to the directory with the uploaded image and import the image:

```
docker import cEOS-lab-4.30.6M.tar.xz arista/ceos:4.30.6M
```

> NOTE: you can also import the image with the tag latest to allow quick "upgrade" of those lab where specific version is not required: `docker tag arista/ceos:4.30.6M arista/ceos:latest`

2. Confirm that the image was imported successfully:

```
$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
arista/ceos      4.30.6M   21b540a4a343   45 minutes ago   1.95GB
arista/ceos      latest    21b540a4a343   45 minutes ago   1.95GB
hello-world   latest    b038788ddb22   3 months ago     9.14kB
```

Install Containerlab

- It's just a one-liner:

```
bash -c "$(curl -sL https://get.containerlab.dev)"
```

- Refer to the Containerlab quick start documentation for the details.
- Containerlab is pre-installed if you are using Codespaces.

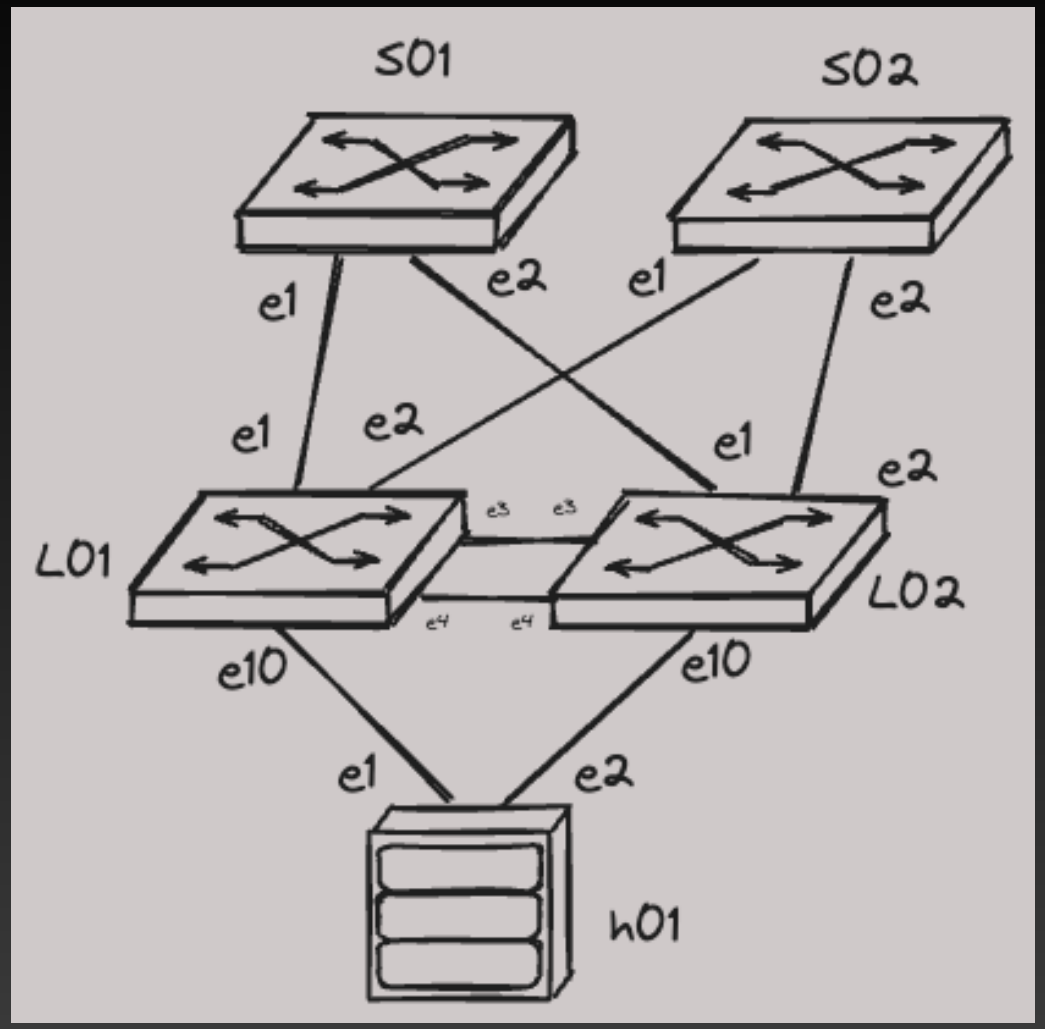
Deploy The Lab

- Inspect `topology.clab.yml` and deploy the lab:

```
sudo containerlab deploy
```

- This command will deploy Containerlab with the default EOS configuration provided by Containerlab.
- (Optional): you can add `--debug` flag to get additional information while Containerlab is starting.

> NOTE: there is no need to specify topology file explicitely, as there only one `.clab.yml` file in the current directory. When multiple topologies are present, the topology to be started must be specified explicitely.



Inspect the Lab

Once the lab is ready, you'll see a table with the list of deployed containers, their host names and management IPs:

```
+---+------+--------------+---------------------+------+---------+--------------+--------------+
| # | Name | Container ID |        Image        | Kind |  State  | IPv4 Address | IPv6 Address |
+---+------+--------------+---------------------+------+---------+--------------+--------------+
| 1 | h01  | 5367c60bcb1c | arista/ceos:4.30.6M | ceos | running | 10.0.3.1/16  | N/A          |
| 2 | l01  | 783f209af70e | arista/ceos:4.30.6M | ceos | running | 10.0.2.1/16  | N/A          |
| 3 | l02  | 47f9904801ce | arista/ceos:4.30.6M | ceos | running | 10.0.2.2/16  | N/A          |
| 4 | s01  | 82812ceefb42 | arista/ceos:4.30.6M | ceos | running | 10.0.1.1/16  | N/A          |
| 5 | s02  | 2839bc4a1ca7 | arista/ceos:4.30.6M | ceos | running | 10.0.1.2/16  | N/A          |
+---+------+--------------+---------------------+------+---------+--------------+--------------+
```

> You can call the table again any time with `sudo clab inspect -t topology.clab.yml`. Or simply `sudo clab inspect`.

Containerlab creates corresponding entries in the `/etc/hosts` file as well:

```
$ cat /etc/hosts | grep clab- -A 5
###### CLAB-build-clab-with-ceos-START ######
10.0.2.1        l01
10.0.2.2        l02
10.0.1.1        s01
10.0.1.2        s02
10.0.3.1        h01
###### CLAB-build-clab-with-ceos-END ######
```

You can also list containers using docker command:

```
$ docker container ls
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS     NAMES
5367c60bcb1c   arista/ceos:4.30.6M   "bash -c '/mnt/flash…"   18 minutes ago   Up 18 minutes             h01
82812ceefb42   arista/ceos:4.30.6M   "bash -c '/mnt/flash…"   18 minutes ago   Up 18 minutes             s01
783f209af70e   arista/ceos:4.30.6M   "bash -c '/mnt/flash…"   18 minutes ago   Up 18 minutes             l01
2839bc4a1ca7   arista/ceos:4.30.6M   "bash -c '/mnt/flash…"   18 minutes ago   Up 18 minutes             s02
47f9904801ce   arista/ceos:4.30.6M   "bash -c '/mnt/flash…"   18 minutes ago   Up 18 minutes             l02
```

Access cEOS-lab CLI

There are few options to access cEOS-lab CLI:

- SSH to the container. For ex.:

```
# the default login is `admin` and password is `admin`
ssh admin@l01
```

- Connect to the "console" using Docker command. For ex.: `docker exec -it l01 CLi`

> NOTE: `docker exec -it l01 bash` allows to connect directly to the switch shell.

Execute few command to confirm that cEOS-lab is functioning:

- `show version`
- `show lldp neighbors`
- `show running-config`

Destroy the Lab

- Destroy the lab with `sudo containerlab destroy`
- This will stop all containers, but will keep the files created by clab for the next run. For example, startup-configs.
- Check the flash content for leaf1 and inspect it's startup config:

```
$ ls clab-build-clab-with-ceos/l01/flash
AsuFastPktTransmit.log  SsuRestoreLegacy.log  debug             kickstart-config  startup-config
Fossil                  aboot                 fastpkttx.backup  persist           system_mac_address
SsuRestore.log          boot-config           if-wait.sh        schedule          tpm-data
```

- To remove these files and have a clean environment on the next run, use `--cleanup` flag:

```
sudo containerlab destroy --cleanup
```

Deploy the Lab with Custom Startup Config

- Deploy the lab with the custom configuration:

```
sudo containerlab deploy --debug --topo clab/topology.clab.yml --reconfigure
```

> NOTE: `--reconfigure` is required if `--cleanup` flag was not specified in the previous step. Otherwise custom configs can be ignored and startup configs in `clab-build-clab-with-ceos/` will be used instead.

- Custom startup configs are located in the `clab/init-configs` directory and assigned to every node using `startup-config:` key in the `topology.clab.yml`. This allows creating pre-configured labs. In this workshop switches are preconfigured with a full EVPN MLAG setup. Host is pre-configured as well and should be able to ping the default gateway and diagnostic loopbacks of leaf switches:

```
$ ssh admin@h01
Password:
h01>en
h01#ping 10.100.100.1
h01#bash for i in {3..4}; do ping -c 4 100.64.101.${i}; done
```

Additional Checks

Execute following commands on leaf1 to confirm that it is functioning as expected:

- `show ip bgp summary`
- `show bgp evpn summary`
- `show mlag`
- `show port-channel dense`

```
l01#sh ip bgp summary
BGP summary information for VRF default
Router identifier 100.65.255.3, local AS number 65101
Neighbor Status Codes: m - Under maintenance
  Description              Neighbor   V AS           MsgRcvd   MsgSent  InQ OutQ  Up/Down State   PfxRcd PfxAcc
  s01_Ethernet1/1          100.65.0.0 4 65100            12        14    0    0 00:04:26 Estab   1      1
  s02_Ethernet1/1          100.65.0.2 4 65100            11        13    0    0 00:04:26 Estab   1      1
  l02                      100.65.2.1 4 65101            12        12    0    0 00:04:29 Estab   4      4
```

cEOS-lab Interface Mapping

The lab with custom configs also has a custom interface mapping defined in `interface_mapping.json`.
This can be useful to match real interface names, for example to have `Management1` interface on cEOS-lab instead of the default `Management0` or to get `EthernetX/X` style naming.

To get `/` as part of an interface name you can simply use `_` (underscore) in cLab topology file. There is no need to define interface map for that. However management interface can only be renamed via interface mapping.

```
{
    "ManagementIntf": {
        "eth0": "Management1"
    },
    "EthernetIntf": {
        "eth1_1": "Ethernet1/1",
        "eth2_1": "Ethernet2/1",
        "eth3_1": "Ethernet3/1",
        "eth4_1": "Ethernet4/1",
        "eth10_1": "Ethernet10/1"
    }
}
```

Make Packet Capture

- Every container has it's own Linux namespace. To list all interfaces for leaf1, execute following command:

```
sudo ip netns exec l01 ip link
```

- Run following command and wait a few minutes to capture a BGP packets:

```
sudo ip netns exec l01 tcpdump -nni eth1_1 port 179 -vvv
```

- You can clear BGP sessions on `l01` if it takes too long to capture keepalives: `clear ip bgp *`
- For additional details about packet capture check cLab documentation.

```
$ sudo ip netns exec l01 tcpdump -nni eth1_1 port 179 -vvv
tcpdump: listening on eth1_1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
12:44:46.487613 IP (tos 0xc0, ttl 1, id 5838, offset 0, flags [DF], proto TCP (6), length 71)
    100.65.0.0.41659 > 100.65.0.1.179: Flags [P.], cksum 0x5113 (correct), seq 4189457049:4189457068, ack 2029471463, win 215, options [nop,nop,TS val 2090790905 ecr 910314922], length 19: BGP
        Keepalive Message (4), length: 19
12:44:46.487696 IP (tos 0xc0, ttl 1, id 37600, offset 0, flags [DF], proto TCP (6), length 52)
    100.65.0.1.179 > 100.65.0.0.41659: Flags [.], cksum 0x0b93 (correct), seq 1, ack 19, win 215, options [nop,nop,TS val 910333765 ecr 2090790905], length 0
12:44:56.117576 IP (tos 0xc0, ttl 3, id 16321, offset 0, flags [DF], proto TCP (6), length 71)
    100.65.255.3.179 > 100.64.255.1.36257: Flags [P.], cksum 0xba92 (correct), seq 3638527337:3638527356, ack 2720785880, win 211, options [nop,nop,TS val 4112950959 ecr 1286050690], length 19: BGP
        Keepalive Message (4), length: 19
12:44:56.117754 IP (tos 0xc0, ttl 3, id 31784, offset 0, flags [DF], proto TCP (6), length 52)
    100.64.255.1.36257 > 100.65.255.3.179: Flags [.], cksum 0x5add (correct), seq 1, ack 19, win 212, options [nop,nop,TS val 1286076241 ecr 4112950959], length 0
12:45:14.505482 IP (tos 0xc0, ttl 1, id 37601, offset 0, flags [DF], proto TCP (6), length 71)
    100.65.0.1.179 > 100.65.0.0.41659: Flags [P.], cksum 0x99f2 (correct), seq 1:20, ack 19, win 215, options [nop,nop,TS val 910361783 ecr 2090790905], length 19: BGP
```

Containerlab in a Container

- Destroy the lab with cleanup flag: `sudo containerlab destroy --topo clab/topology.clab.yml --cleanup`
- It is possible to run the containerlab on the host without installing it by simply running it in a container. This is helpful on MacBooks (the only way to run cLab) and advanced use cases (like this workshop).
- Start Containerlab by using this command:

```
docker run --rm -it --privileged \
  --network host \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /etc/hosts:/etc/hosts \
  --pid="host" \
  -w $(pwd) \
  -v $(pwd):$(pwd) \
  ghcr.io/srl-labs/clab bash
```

- This will start the container in the interactive mode. Once in the container prompt, execute following command to start the lab:

```
containerlab deploy --debug --topo clab/topology.clab.yml --reconfigure
```

- Destroy the lab with `containerlab destroy --topo clab/topology.clab.yml --cleanup` when ready and exit the container by typing `exit`.

Containerlab in a Container (Non-Interactive)

- Running cLab container in non-interactive mode is helpful to create shortcuts, etc.
- You can test it now or skip this step.
- Check the documentation for additional details.

```
# deploy the lab
docker run --rm --privileged \
  --network host \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /etc/hosts:/etc/hosts \
  --pid="host" \
  -w $(pwd) \
  -v $(pwd):$(pwd) \
  ghcr.io/srl-labs/clab containerlab deploy --debug --topo clab/topology.clab.yml --reconfigure


# destroy the lab
docker run --rm --privileged \
  --network host \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /etc/hosts:/etc/hosts \
  --pid="host" \
  -w $(pwd) \
  -v $(pwd):$(pwd) \
  ghcr.io/srl-labs/clab containerlab destroy --topo clab/topology.clab.yml --cleanup
```

Crafting Your Own Container

- It is easy to craft your own container with Containerlab installed.
- You can check some Dockerfiles in this repository for inspiration or check cLab documentation
- Possible reasons to create your own container:
  - Produce a consistent environment that is easy to share.
  - Pre-install additional tools. (Ansible, docker-in-docker, etc.)
  - Add aliases, etc.
- This workshop is a good example of an enviroment using a custom pre-build container with Docker-in-Docker and cLab

Ansible with Containerlab

- When containerlab starts it automatically creates Ansible inventory that can be used to automate certain tasks in the lab.
- Start the lab and inspect the inventory file: `cat clab-build-clab-with-ceos/ansible-inventory.yml`
- Check if ansible is already installed: `ansible --version`
- Install Ansible if it's not present:

```
pip3 install "ansible-core>=2.14.0,<2.16"
ansible-galaxy collection install ansible.netcommon
ansible-galaxy collection install arista.eos
# install community.general to support callback plugins in ansible.cfg, etc.
ansible-galaxy collection install community.general
```

- Inspect `ansible.cfg` and make sure that it is matching your environment.
- Run the playbook: `ansible-playbook playbooks/check_the_lab.yml`
- The playbook will execute number of show commands on all switches in the lab and print output on the screen.

Possible Caveats

> WARNING: If you are planning to deploy a high scale lab, test it on a non-production host that you can access and recover any time. Incorrectly deployed Containerlab at scale can bring your host down due to high CPU utilization on start.

- It's always good to add `--max-workers` and `--timeout` flags to your containerlab deploy command.
- Use recent cEOS-lab version. 4.30 or higher is strongly recommended!
- cLab is creating a lot of files as root. That can cause permission issues. For example, make sure that all cLab files are gitignored:

```
# ignore clab files
clab-*
*.bak
```

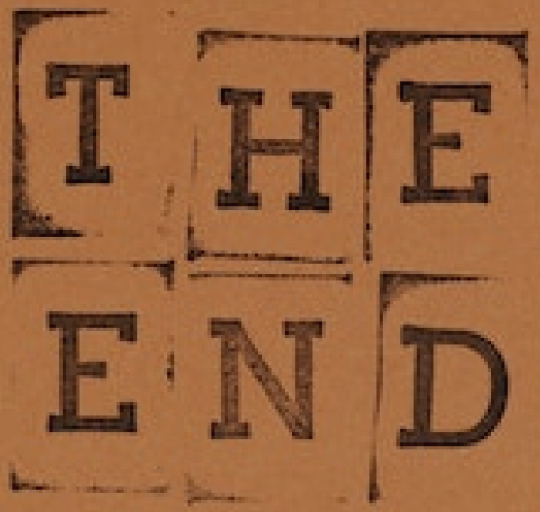
Additional Scale Caveats

- In the past Ubuntu used to have low `fs.inotify.max_user_instances` limit. On top, older cEOS-lab versions were decreasing this system limit to 1256. This was causing issues with high scale labs.
- On a modern system and any cEOS-lab later than 4.28 this system is high enough. 62800 is the default. Increasing this limit on a modern host with high memory is not causing any issues. Feel free to play with this parameter if required:

```
# set system limit
sudo sysctl -w fs.inotify.max_user_instances=62800
# create 99-zceos.conf
sudo sh -c 'echo "fs.inotify.max_user_instances = 62800" > /etc/sysctl.d/99-zceos.conf'
# check the limit
sudo sysctl -a  | grep -i inotify
```

```
topology:
kinds:
  ceos:
    # mount custom 99-zceos.conf to cEOS-lab containers
    binds:
      - /etc/sysctl.d/99-zceos.conf:/etc/sysctl.d/99-zceos.conf:ro
```

- Generally you don't have to touch that. But be aware and check in case of issues.



Q&A

- Containerlab
- This repository