

Workshop

Building Containerlab with cEOS-lab

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

Petr Ankudinov, 2023



CONTAINERlab

Credits and References

Credits to [Roman Dodin](#) and [other cLab contributors](#) for making the world a better place!

This repository is based on many awesome open source repositories and some free/commercial Github features:

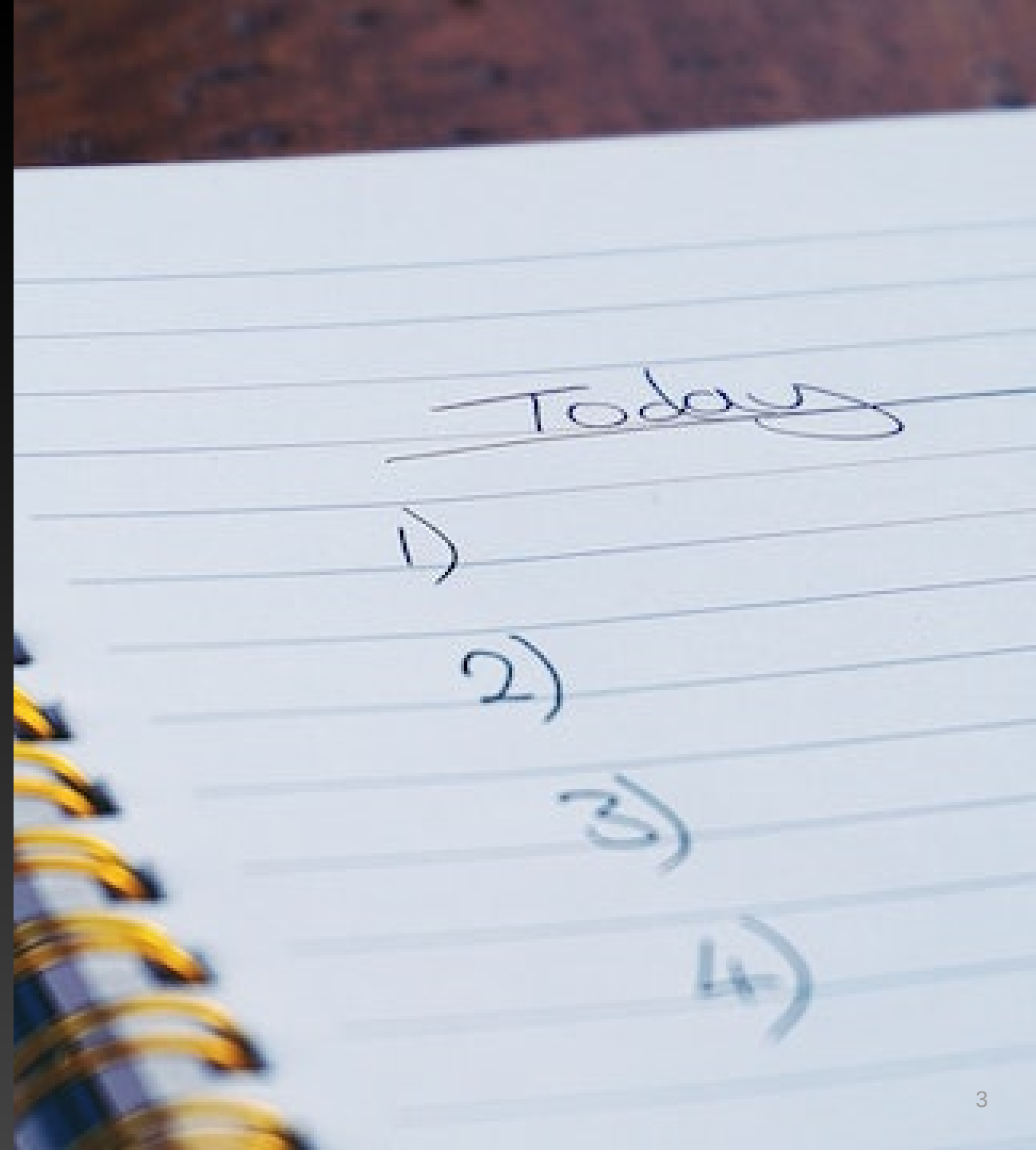
- [Containerlab](#)
- [VS Code](#)
- [DevContainers](#)
- [Marp](#)
- [Excalidraw VS Code Plugin](#)
- [Github Actions](#)
- [Github Pages](#)
- [Github Codespaces](#)
- [Carbon](#)
- And many more...

All photos are taken from [Pexels](#) and [Unsplash](#). Excellent free stock photos resources. It's not possible to reference every author individually, but their work is highly appreciated.

Agenda

- Setup Docker on the host
- Install Containerlab and import cEOS-lab image
- Clone this repository and 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 details.



Prerequisites

- This workshop requires:
 - Ubuntu LTS 22.04 or later
 - 8 GB RAM and 4 vCPUs
- Only x86 architecture is supported. It is technically possible to [run Container lab on ARM](#), but there are no network images available for ARM as of Aug 2023.
- You can use [Github Codespaces](#) or [VSCode devcontainer](#) for this workshop. The detailed procedure is described in the appendix.
- The appendix also provides instructions for creating a KVM VM with Ubuntu Cloud Image.
- There is also Vagrant file available in this repository. Use it at your own risk.

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 use 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.

```
# 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
```

Setup Git (Optional)

- Git must be pre-installed. 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
```

Clone this Repository

```
$ cd ${HOME}
$ git clone https://github.com/arista-netdevops-community/building-containerlab-with-ceos.git
Cloning into 'building-containerlab-with-ceos'...
remote: Enumerating objects: 198, done.
remote: Counting objects: 100% (198/198), done.
remote: Compressing objects: 100% (120/120), done.
remote: Total 198 (delta 109), reused 152 (delta 66), pack-reused 0
Receiving objects: 100% (198/198), 1.31 MiB | 6.59 MiB/s, done.
Resolving deltas: 100% (109/109), done.
$ ls | grep ceos
building-containerlab-with-ceos
$ cd building-containerlab-with-ceos
```

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-4.30.2F > cEOS-lab`.
3. Download `cEOS-lab-4.30.2F.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-4.30.2F.tar*'
# for example:
# sftp user@10.10.10.11:/home/user/images <<< $'put cEOS-lab-4.30.2F.tar*'
```

NOTE: if you are using Vagrant, add the image to `.gitignored` directory. It will be automatically copied to the VM.

If Github Codespace is used and token is set, the image will be pulled from arista.com automatically.

EOS

Active Releases

4.30

EOS-4.30.2F

vEOS-lab

Docs

cEOS-lab

`cEOS-lab-4.30.2F.tar.xz`

`cEOS-lab-4.30.2F.tar.xz.json`

`cEOS-lab-4.30.2F.tar.xz.md5sum`

`cEOS-lab-4.30.2F.tar.xz.sha512sum`

`cEOS64-lab-4.30.2F.tar.xz`

`cEOS64-lab-4.30.2F.tar.xz.json`

`cEOS64-lab-4.30.2F.tar.xz.md5sum`

`cEOS64-lab-4.30.2F.tar.xz.sha512sum`

Import cEOS-lab Image

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

```
docker import cEOS-lab-4.30.2F.tar.xz ceos-lab:4.30.2F
```

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 ceos-lab:4.30.2F ceos-lab:latest`

2. Confirm that the image was imported successfully:

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ceos-lab	4.30.2F	21b540a4a343	45 minutes ago	1.95GB
ceos-lab	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.

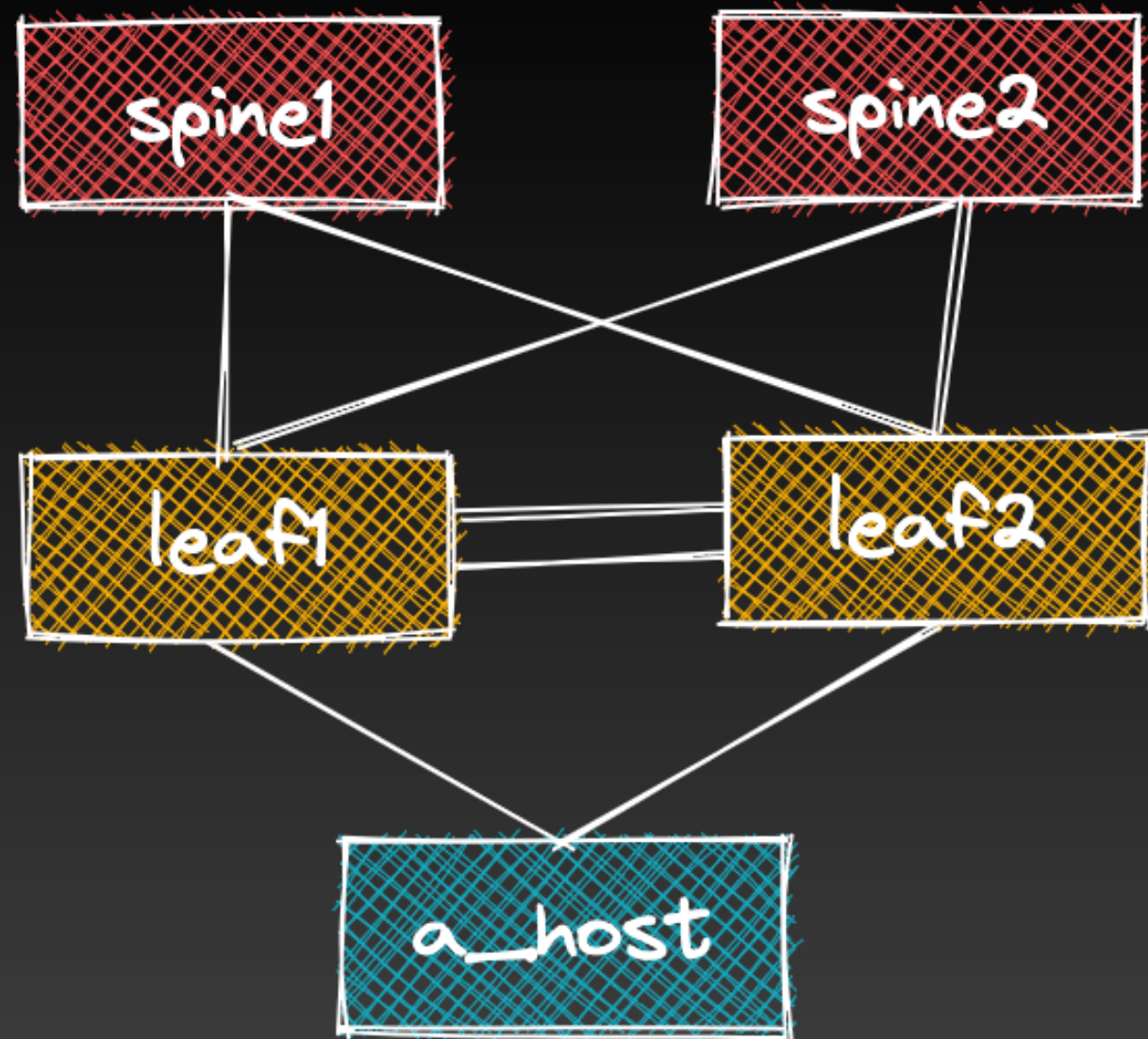
Deploy The Lab

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

```
sudo containerlab deploy --debug --topo default_cfg.clab.yml
```

- This command will deploy Containerlab with the default EOS configuration provided by Containerlab. The `--debug` flag is optional, but provides additional information while Containerlab is starting.

NOTE: If there is a single `.clab.yml` file in the current directory, it is possible to use `sudo containerlab deploy` command without specifying the topology file. As we have multiple files in the directory, we must specify the topology explicitly.



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	clab-ceos-lab-a_host	421665f3e67f	ceos-lab:latest	ceos	running	192.168.123.100/24	N/A
2	clab-ceos-lab-leaf1	a7f7c80aa90f	ceos-lab:latest	ceos	running	192.168.123.21/24	N/A
3	clab-ceos-lab-leaf2	142ab91f0ceb	ceos-lab:latest	ceos	running	192.168.123.22/24	N/A
4	clab-ceos-lab-spine1	22464060dcf8	ceos-lab:latest	ceos	running	192.168.123.11/24	N/A
5	clab-ceos-lab-spine2	3f53d1de7add	ceos-lab:latest	ceos	running	192.168.123.12/24	N/A

You can call the table again any time with `sudo clab inspect -t default_cfg.clab.yml`.

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

```
$ cat /etc/hosts | grep -i clab-
##### CLAB-ceos-lab-START #####
192.168.123.22 clab-ceos-lab-leaf2
192.168.123.11 clab-ceos-lab-spine1
192.168.123.12 clab-ceos-lab-spine2
192.168.123.100 clab-ceos-lab-a_host
192.168.123.21 clab-ceos-lab-leaf1
##### CLAB-ceos-lab-END #####
```

You can also list containers using docker command:

```
$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
22464060dcf8   ceos-lab:latest "bash -c '/mnt/flash..." 5 minutes ago  Up 5 minutes                clab-ceos-lab-spine1
3f53d1de7add   ceos-lab:latest "bash -c '/mnt/flash..." 5 minutes ago  Up 5 minutes                clab-ceos-lab-spine2
a7f7c80aa90f   ceos-lab:latest "bash -c '/mnt/flash..." 5 minutes ago  Up 5 minutes                clab-ceos-lab-leaf1
421665f3e67f   ceos-lab:latest "bash -c '/mnt/flash..." 5 minutes ago  Up 5 minutes                clab-ceos-lab-a_host
142ab91f0ceb   ceos-lab:latest "bash -c '/mnt/flash..." 5 minutes ago  Up 5 minutes                clab-ceos-lab-leaf2
```

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@clab-ceos-lab-leaf1
```

- Connect to the "console" using Docker command. For ex.: `docker exec -it clab-ceos-lab-leaf1 cli`

NOTE: `docker exec -it clab-ceos-lab-leaf1 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 -t default_cfg.clab.yml`
- 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 its startup config:

```
$ ls clab-ceos-lab/leaf1/flash/
AsuFastPktTransmit.log  SsuRestore.log      aboot  fastpkttx.backup  kickstart-config  schedule  system_mac_address
Fossil                 SsuRestoreLegacy.log debug  if-wait.sh        persist          startup-config
```

```
$ cat clab-ceos-lab/leaf1/flash/startup-config
```

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

```
sudo containerlab destroy -t default_cfg.clab.yml --cleanup
```

Deploy the Lab with Custom Startup Config

- Deploy the lab with the custom configuration:

```
sudo containerlab deploy -t custom_cfg.clab.yml --reconfigure
```

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

- Custom startup configs are located in the `init-configs` directory and assigned to every node using `startup-config:` key in the `custom_cfg.clab.yml`. This allows creating pre-configured labs. In this case pre-configured MLAG between leaf switches and basic BGP underlay configuration. Host should be able to ping loopbacks of all leaf and spine switches. Connect to the host and run following commands to confirm that:

```
$ ssh admin@clab-ceos-lab-a_host
Password:
a_host>en
a_host#bash for i in {1..4}; do ping -c 4 10.${i}.${i}.${i}; done
```

Additional Checks

Run following commands on leaf1:

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

NOTE: `ambassadors_custom_cfg.clab.yml` has custom interface mapping defined in `interface_mapping.json` and assigned to cEOS-lab containers as bind mount. This helps to change default Management0 interface to Management1 as on physical switches.

Make Packet Capture

Every container has its own namespace. To list all interfaces for leaf1, execute following command on the lab VM:

```
sudo ip netns exec clab-ambassadors_clab-leaf1 ip link
```

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

```
sudo ip netns exec clab-ambassadors_clab-leaf1 tcpdump -nni eth1_1 port 179 -vvv
```

For additional details about packet capture check [cLab documentation](#).

```
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 bash
```

This will start the container with cLab interactively. Once inside the container prompt, execute the following command to start the lab:

```
containerlab deploy -t ambassadors_custom_cfg.clab.yml --reconfigure
```

Check the lab and destroy it: `containerlab destroy -t ambassadors_custom_cfg.clab.yml --cleanup`
Exit the container.

The default `ghcr.io/srl-labs/clab` container is making all changes as root. That can cause permissions issues if you are working with your repository from the container prompt. It is better to use `ghcr.io/srl-labs/clab` as non-interactive or craft your own container to map the user ID correctly.

To use the container in non-interactive way execute following command:

```
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 bash
```

The custom container has following features:

- ZSH and a nice prompt with a whale. =)
- Number of Linux tools pre-installed.
- Docker (in Docker) and Containerlab installed
- Aliases to start and stop the lab and connect to the lab switches
- Entrypoint
- UID and GID inside the container matching UID and GID outside the container
- Ansible included

Let's build our own container now:

```
# build a temp container with UID 1000
docker build --rm --pull --no-cache -f Dockerfile -t ambassadors_temp_image .
# build final container with matching UID
docker build -f updateUID.Dockerfile -t ambassadors_clab:latest --build-arg BASE_IMAGE=ambassadors_temp_image --build-arg REMOTE_USER=clab --build-arg NEW_UID=$(id -u) --build-arg NEW_GID=$(id -g) --build-arg IMAGE_USER=clab .
```

Start the container:

```
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) \
  ambassadors_clab:latest
```

Ansible with Containerlab

When containerlab starts it automatically creates Ansible inventory that can be used to automate certain tasks in the lab.

Start `ambassadors_clab:latest` container we have created earlier and deploy the lab.

Inspect the Ansible inventory: `cat clab-ambassadors_clab/ansible-inventory.yml`

Ansible is already installed inside the container and `ansible.cfg` is provided in the repository as well as the playbook `check_the_lab.yml`.

Run the playbook by executing command `ansible-playbook playbooks/check_the_lab.yml`

This playbook will execute number of show commands on all switches in the lab and present output on the screen.

Generally, Ubuntu systems have quite low `fs.inotify.max_user_instances` limit by default. Even if it was increased, older cEOS-lab containers can decrease system limit to 1256. That is not sufficient for a high scale lab. The lab may fail to start and even bring your host down due to high CPU.

In reality increasing inotify limit on a modern host with high RAM will not create any disadvantages. If you are planning to deploy older cEOS-lab container, you can increase it manually.

1st, define your inotify limit. You can safely assume that it will not be more than $1256 \times \text{number of containers}$. But the required limit is expected to be significantly below that. Newer cEOS-lab images set the limit to 62800, that is a good number for most cLab deployments.

Set your 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`

Mount the custom 99-zceos.conf to your cEOS-lab containers in the topology file:

```
topology:
  kinds:
    ceos:
      binds:
        - /etc/sysctl.d/99-zceos.conf:/etc/sysctl.d/99-zceos.conf:ro
```

Add `--max-workers` and `--timeout` flags to your containerlab deploy command.

References

- [avd-quickstart-containerlab](#)
- [avd-all-in-one-container](#)
- [avd-cEOS-Lab](#)
- [kvm-lab-for-network-engineers](#)
- [Containerlab documentation](#)