

Better Labs on GitHub and Private Infra

```
{ "event": "Autocon5", "session": "WS:A4" }
```

```
#!/usr/bin/env python3

__author__ = [
    "Petr Ankudinov (pa@), Senior Solutions Engineer",
    "Mitch Vaughan (mitch@), Principal Engineer",
    "Carl Buchmann (carl.buchmann@), Manager, Solutions Engineering"
]
```

```
$ date +"%b %Y"
Jun 2026
```



About the Slide Deck and WiFi

- Slides are created in [Marp](#) by Yuki Hattori
- Marp makes it easy to build, share, and publish rich Markdown slides from CI
- Check [Marp awesome list](#) and build your next slide deck in Marp
- Most images are from [Pexels](#) or generated by AI
- Wi-Fi & Connectivity
 - Network Name: [AutoCon5]
 - Password: [MunichRocks!]
- Workshop repo: <https://github.com/ankudinov/aclabs-workshop-june-2026>
 - Or type less with <https://tinyurl.com/6er72uw2>

Scan QR code to access the workshop repository.



Agenda

▶ Start: 09:00

☕ Break 10:45-11:15

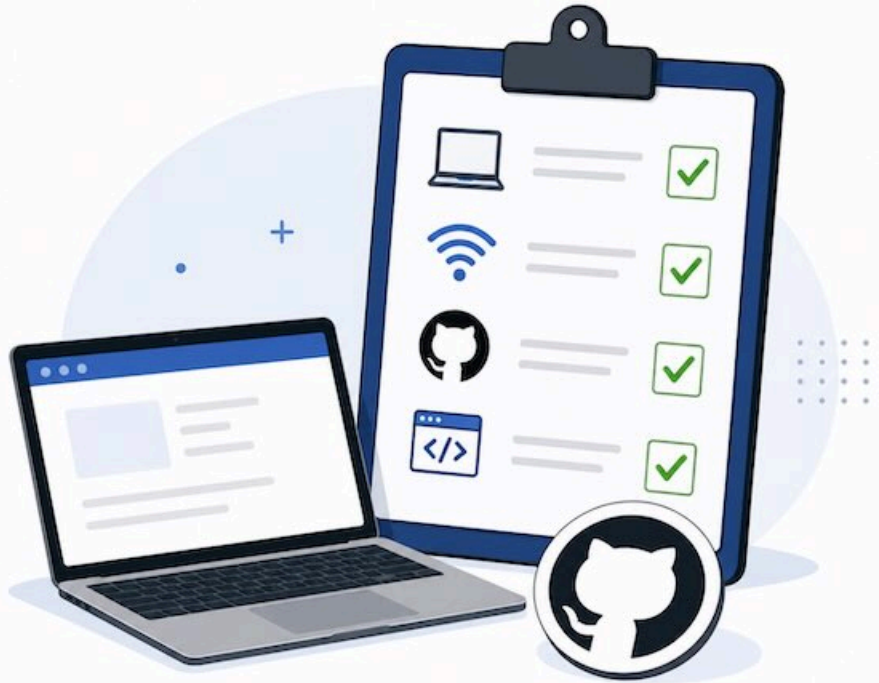
Morning 09:00 - 10:45

- Intro, logistics, and **labs.arista** environment overview and user assignment
- Evolution of the Network Labs and why better labs matter
- Dev Containers, Codespaces and deeplink hands-on
- Pre-built containers, GitHub Actions and GHCR
- A survival guide to SELinux, the kernel and permissions
- The story of two platforms: ARMin your lab

● Stop: 13:00

Afternoon 11:15 - 13:00

- GitHub Actions: reusable workflows and matrix builds
- Add Containerlab, code-server and inventory
- Using ENTRYPOINT for image import and more
- Build a simple deeplink API to trigger lab deployment
- Code-server - the perfect UI
 - Use tasks and other VSCode features to customize the lab



What You Need Today

Participants need:

- A healthy laptop with a working browser
- Reliable Internet access for the full session
- A GitHub account you can sign into quickly
- Optional GitHub Codespaces access

A good level of system administration / Linux skills expected

- Useful reference materials
 - [GitSCM book](#)
 - [Git cheat-sheet](#)
 - [container.training](#)
 - [VSCode tutorial](#)

GitHub access is the critical dependency for the hands-on flow.

Meet **labs.arista**

- labs.arista.com provides UI, API, Cloud runtime and event management for unified containerized lab backend
- Arista employees and customers can access most of the labs in a single click
 - Example: [AVD Playground](#)
- However for this workshop we'll use event management tool to avoid auth restrictions
 - **IMPORTANT:** Provide a valid email when registering!
- [Click this link](#) to get your lab instance
 - The lab VM will be running for approx. 8 hours and is not persistent
 - Save any progress you need for later outside of the VM
- [The extension link](#). Please wait until previous event is full!

Pa-Test3

SCHEDULED



Event Duration : May 7th 2026, 11:15:00 am to May 8th 2026, 12:00:00 pm



Owner Name : Petr Ankudinov

RCF QuickStart

AVAILABLE

3 **3**
Total Available



Lab Scheduled

Start Time : May 7th 2026,
11:15:00 am

End Time : May 7th 2026,
12:01:00 pm

REQUEST

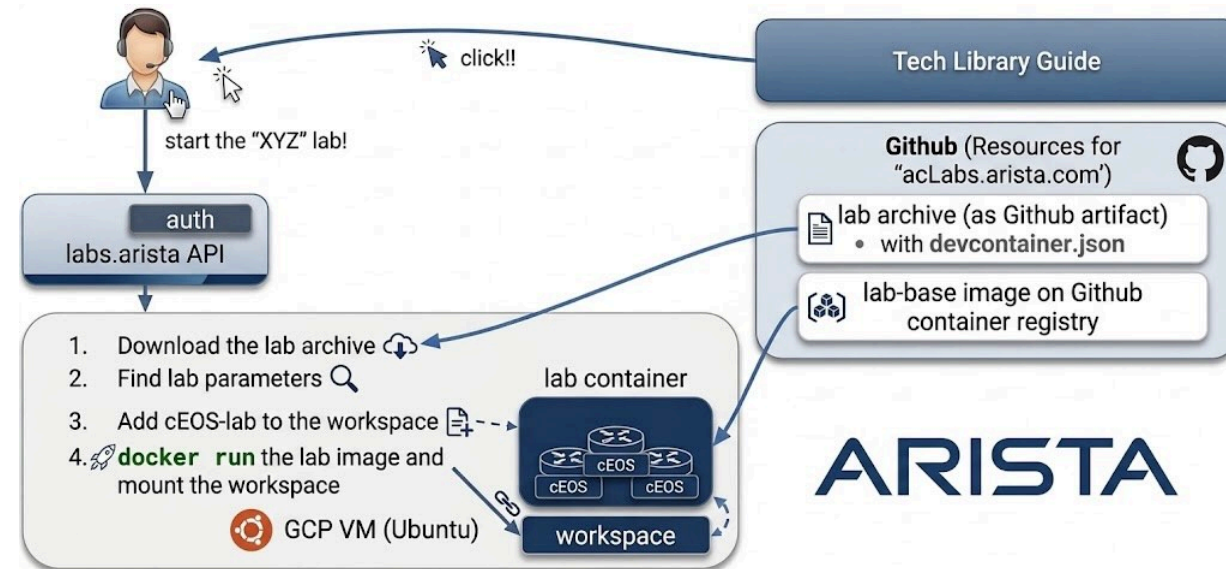
RELEASE

How It Works

- The key idea
 - Single click to start any lab
 - The link can be integrated to any Web resource, doc or [this slide](#)
 - Consistent and ready to use lab environment every time
 - Familiar VSCode UI and features for everything
 - Build on top of existing Open Source initiatives!
- Components
 - A document in your web browser to click the link and open the lab UI
 - [labs.arista](#) for auth, API and GCP backend
 - This is the only proprietary part of the solution!
 - [acLabs](#) - lab packages and lab-base container images

TARGET 🎯: Build a very similar setup during the workshop!

WHY: The way we used to build labs is obsolete. Let's build better labs!



THE EVOLUTION OF NETWORK LABS

FROM MANUAL TO AUTOMATED. FROM COMPLEX TO CODE.



Why It's Important

- Industry requires a better way to build labs
- Important criteria
 - Time-to-start
 - Git friendly and declarative
 - Portable - any platform, any machine, any cloud
 - Easy to share, ready to support any content
 - Standard based and low maintenance
 - No need to re-invent another custom square-shaped wheel when there are great tools out there already
- We are not that different from devs
- But... Standard Dev Container / Codespaces workflow brings challenges
 - Not optimized for network lab use case
 - Not under lab maintainer full control
 - High entrance barrier for many network engineers
- Some use cases
 - Modern lab collection: Git-powered, runs anywhere, AI-friendly, etc.
 - [TechLibrary](#)
 - [acLabs](#)
 - [Quick reference or demo](#)
 - [Interactive digital twin for PR review](#)

FANTASTIC LABS

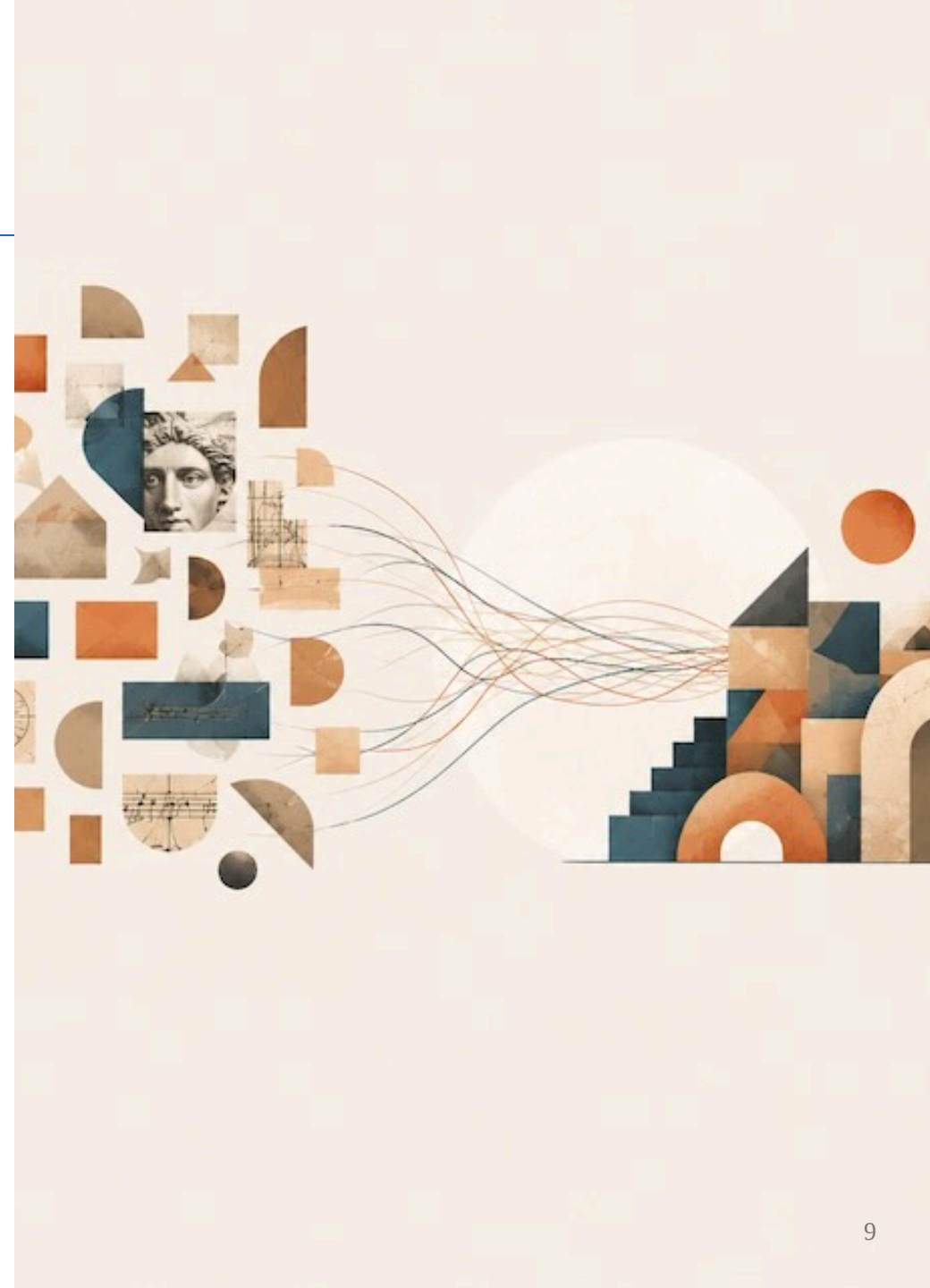
AND WHERE TO FIND THEM



How It Started

New ideas are simply old thoughts that finally found each other

- 2020 / 2021
 - It all started with Arista AVD and solving good old "Works on my machine" challenge
 - dev container specification emerges
 - baby steps: all-in-one pre-build container
- End 2022 / 2023
 - Github Codespaces GA
 - First attempt to socialize the new approach to labs at [Ansible Community Day](#)
- 2024
 - AVD container images with every release
 - Works well with Containerlab!
 - [One-Click SE Demos](#)
 - [TechLibrary](#)
 - [NAN074](#) with Eric Chou and Roman Dodin
- Jan 2026
 - Project cArL birthday (Containerized Arista Labs)
 - labs.arista / acLabs / TechLibrary / AVD integration
- Now
 - Time to make this the new lab standard



Notable Projects

"If I have seen further, it is by standing on the shoulders of giants"

Building a better lab would be much harder without:

- Devcontainer specification
- Code Server / VSCode
- GitHub and GitHub Actions, Pages and Packages
- Containerlab

Honorable mention:

- labs.iximiuz.com
 - Not directly related and not aimed at network engineers
 - Still a great and inspiring lab implementation



What Is Dev Container

- It's a container 😊
 - When it runs, it's not much different to "normal" container
 - However it's empowered by Dev Container specification
- [Development Container Specification @ containers.dev](#)
 - Originally a proprietary standard for VSCode remote development
 - Open-sourced by Microsoft in 2022
 - Defines metadata to build, open and customize the dev environment
 - Works best with a [supporting tool](#)
 - The most well known is VS Code with Dev Containers extension
 - Keeps the environment definition close to the repo

```
// .devcontainer/devcontainer.json
{
  // Start from a known base image
  "image": "mcr.microsoft.com/devcontainers/base:ubuntu",

  // Add standard tools during create
  "features": {
    "ghcr.io/devcontainers/features/python:1": { "version": "3.11" },
    "ghcr.io/devcontainers/features/docker-in-docker:2": {}
  },

  // Make the repo ready on first open
  "postCreateCommand": "pip install -r requirements.txt",

  // Useful for docs preview or a small lab UI
  "forwardPorts": [8000]
}
```

Hands-on: Create a Dev Container and Start/Stop a Codespace

- Init a new empty repo on Github
 - Call it `ac5-workshop`
- Clone the repo to the Sandbox Lab environment
 - Do not forget to authenticate!
- Create following files:
 - `.devcontainer/devcontainer.json`
 - `requirements.txt`
 - `init.sh`
- Make init script executable:
 - `chmod +x init.sh`
- Commit and push
- Start the Codespace
- Check
- Stop and delete the Codespace

```
// .devcontainer/devcontainer.json
{
  "image": "ghcr.io/aristanetworks/avd/universal:python3.12-avd-v6.1.0",
  "remoteUser": "avd",
  "onCreateCommand": "${containerWorkspaceFolder}/init.sh"
}
```

```
# requirements.txt
rich==13.9.4
```

```
#!/usr/bin/env bash

# init.sh

set +e
pip install -r requirements.txt
wget https://raw.githubusercontent.com/textualize/rich/refs/heads/master/examples/tree.py
echo "alias rich='python3 tree.py $(pwd)'" >> /home/avd/.zshrc
cp -r /home/avd/.ansible/collections/ansible_collections/arista/avd/examples/single-dc-l3ls/* .
```

Fun fact: AVD Container Images might be the first ever pre-build dev environment for network engineers

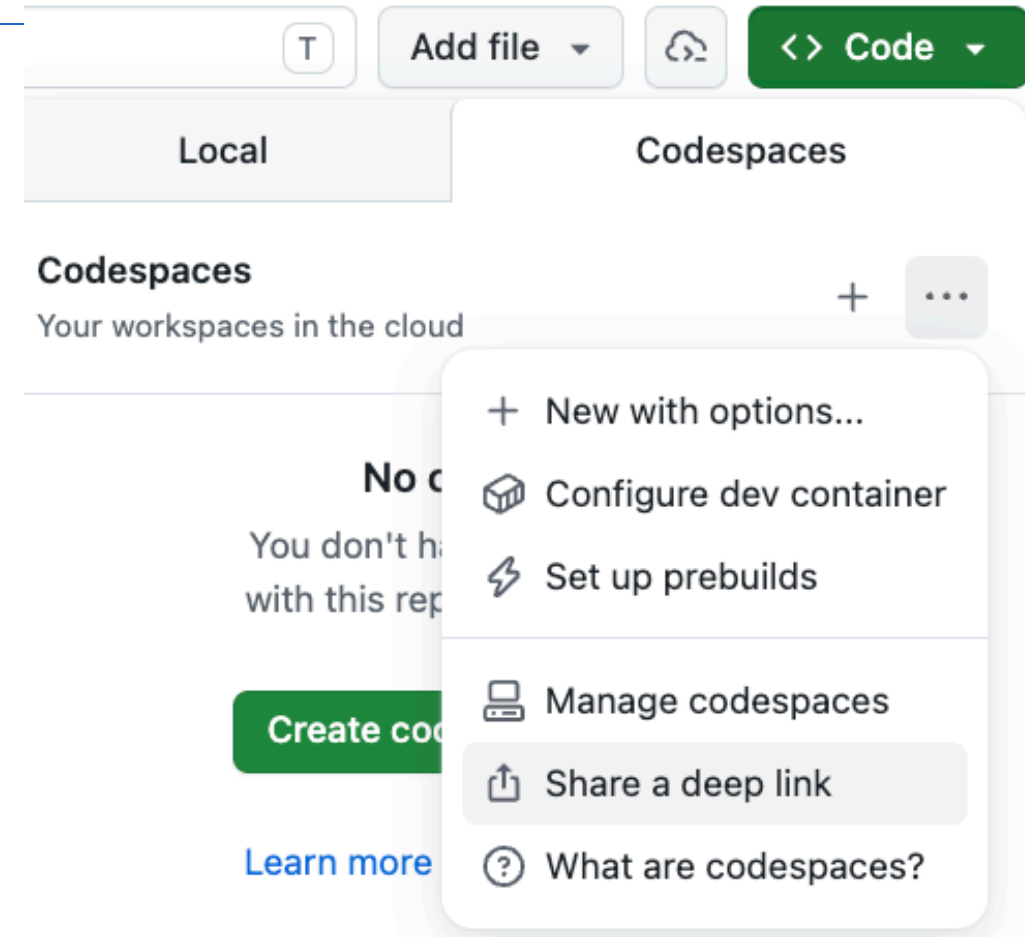
Deeplink

- Probably the best part about Codespaces is deeplink support
- **Deeplink** is a specific URL that points to a particular piece of content or a specific page within a website
- Example:

```
https://codespaces.new/ankudinov/ac5-workshop
      |           |           |
      +---> API      +---> GH Org  +---> Repo Name
```

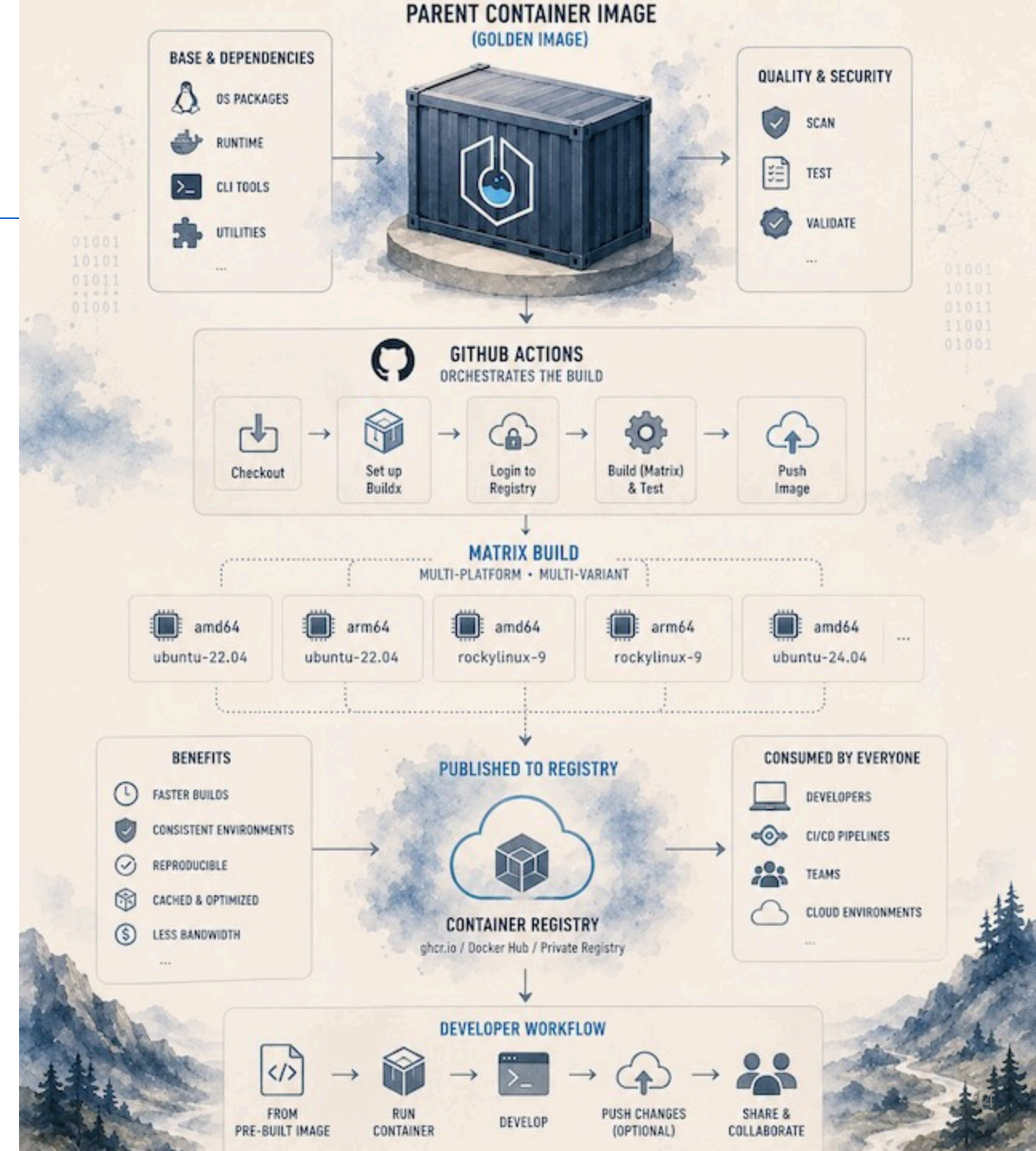
- This allows to us to start a lab in a single click from anywhere:
 - [a doc](#)
 - [a slide](#)
 - [a pull request](#)
 - etc.

Hands-on: add the README.md with the link to Codespace.



Pre-built Containers

- Now we have a very nice setup:
 - A functional environment that can be used as Codespace or own machine
 - A link to start the Codespace from literally anywhere
- It is still not perfect. The key drawback:
 - The container build is happening at runtime
- Drawbacks:
 - More time required to setup environments (heavy builds can be quite slow)
 - Things can change:
 - Broken APIs (Galaxy, PyPi, etc.)
 - Removed/deprecated libraries
 - Updated libraries, especially if version was not fixed in requirements
 - and more ...
- Solution:
 - Build container in advance!



Pre-built Checklist

- There are different ways to build containers
- We are going to use:
 -  GitHub Actions
 -  GitHub Packages
 -  Matrices
 -  Reusable workflows
 -  `devcontainers/ci` action



A Simple Workflow to Build a Container

- containers/my_container/.devcontainer/devcontainer.json :

```
{
  "image": "ghcr.io/aristanetworks/avd/universal:python3.12-avd-v6.1.0",
  "remoteUser": "avd",
  "onCreateCommand": "${containerWorkspaceFolder}/init.sh"
}
```

- containers/my_container/init.sh (do not forget `chmod +x`):

```
#!/usr/bin/env bash

set +e
pip install -r requirements.txt
wget https://raw.githubusercontent.com/textualize/rich/refs/heads/master/examples/tree.py
echo "alias rich='python3 tree.py $(pwd)'" >> /home/avd/.zshrc
cp -r /home/avd/.ansible/collections/ansible_collections/arista/avd/examples/single-dc-l3ls/* .
```

- containers/my_container/requirements.txt

```
rich==13.9.4
```

- 📁 same files as before, but the workflow requires some comments ➡

You can now test `ghcr.io/<your-github-account>/ac5-workshop/my_container:latest`

```
---
# .github/workflows/build_image.yml
name: Reusable build container workflow

env:
  # BUILDX_NO_DEFAULT_ATTESTATIONS must be set to build only arm64 and amd64 images.
  # The devcontainers/ci@v0.3 build will fail if this env variable is not set.
  BUILDX_NO_DEFAULT_ATTESTATIONS: 1

on:
  push:
    branches: ['**']

permissions:
  packages: write

jobs:
  build_image:
    runs-on: ubuntu-22.04
    steps:
      - name: Checkout code ✅
        uses: actions/checkout@v6

      - name: Convert Github repository name to lowercase 📄
        id: gh_repo
        run: echo "name_lowercase=${GITHUB_REPOSITORY,,}" >> $GITHUB_OUTPUT

      - name: Setup QEMU for multi-arch builds ⚠️
        uses: docker/setup-qemu-action@v4
        with:
          platforms: ${inputs.platform}

      - name: Setup Docker buildx for multi-arch builds ⚠️
        uses: docker/setup-buildx-action@v4

      - name: Login to the container registry 🗝️
        uses: docker/login-action@v4
        with:
          registry: ghcr.io
          username: ${inputs.github.actor}
          password: ${inputs.secrets.GITHUB_TOKEN}

      - name: Pre-build dev container image 🔧
        uses: devcontainers/ci@v0.3
        env:
          FROM_IMAGE: ${inputs.from_image}
          FROM_VARIANT: ${inputs.from_variant}
          USERNAME: ${inputs.username}
          UID: ${inputs.user_id}
          GID: ${inputs.group_id}
        with:
          subFolder: containers/my_container
          imageName: ghcr.io/${steps.gh_repo.outputs.name_lowercase}/my_container
          imageTag: latest
          platform: linux/arm64/v8,linux/amd64
          push: always
```


Container Build Workflow - 1

- `env`
 - disables default attestations for ARM images. If missing: the workflow can fail
- `on.push`
 - run the workflow on every branch, filter to specific branch in prod
- `permissions.packages: write`
 - allow pushes to `ghcr.io`. Check your GitHub Actions configuration as well
- `actions/checkout`
 - clone repo files onto the runner first
- `gh_repo`
 - lowercase the repo name for GHCR image tags
 - if skipped: mixed-case names can break image naming.
- `setup-qemu`
 - QEMU is required on an `amd64` runner to build `arm64` (cross-platform builds)
 - you can simply use ARM-runner instead, which is likely a better way
- `setup-buildx`
 - enable the multi-arch builder
- `docker/login-action`
 - sign in to `ghcr.io` with `${{ secrets.GITHUB_TOKEN }}`

```
---
# .github/workflows/build_image.yml
name: Reusable build container workflow

env:
  # BUILDX_NO_DEFAULT_ATTESTATIONS must be set to build only arm64 and amd64 images.
  # The devcontainers/ci@v0.3 build will fail if this env variable is not set.
  BUILDX_NO_DEFAULT_ATTESTATIONS: 1

on:
  push:
    branches: ['**']

permissions:
  packages: write

jobs:
  build_image:
    runs-on: ubuntu-22.04
    steps:
      - name: Checkout code ✓
        uses: actions/checkout@v6

      - name: Convert Github repository name to lowercase ➕
        id: gh_repo
        run: echo "name_lowercase=${GITHUB_REPOSITORY,,}" >> $GITHUB_OUTPUT

      - name: Setup QEMU for multi-arch builds 🚧
        uses: docker/setup-qemu-action@v4
        with:
          platforms: ${{ inputs.platform }}

      - name: Setup Docker buildX for multi-arch builds 🚧
        uses: docker/setup-buildx-action@v4

      - name: Login to the container registry 🔑
        uses: docker/login-action@v4
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
```

A Story of Two Platforms

- Year of 2026...
- MacBook users everywhere, rise of AI and obvious ARM success
- Yet some software and container images are x86 only!
- Fight old habits! Always build for ARM and x86!



Container Build Workflow – 2

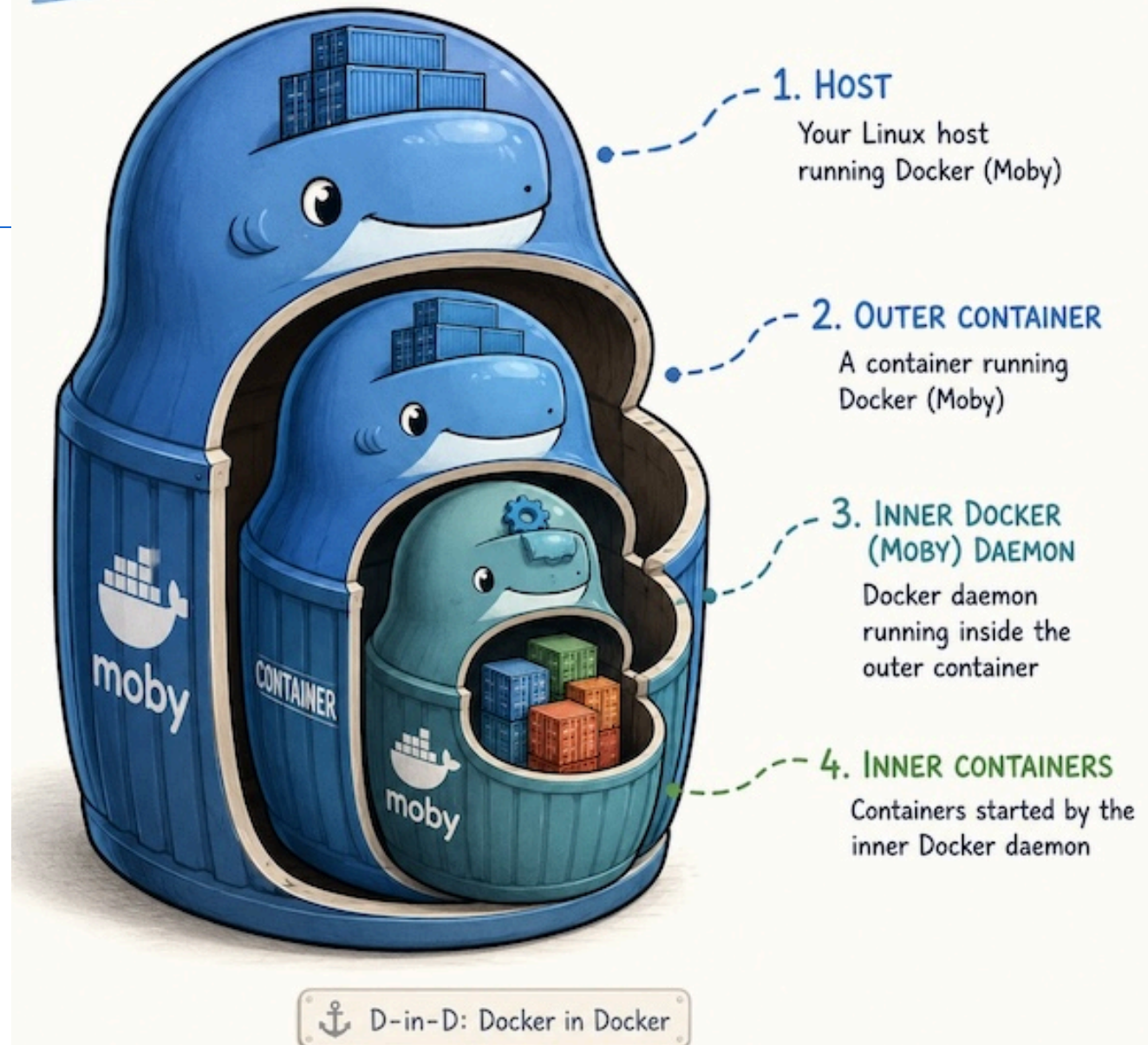
- The [devcontainer/ci](#) action allows pre-building a Dev Container image inside a CI pipeline
- It's optional, but makes build easier as we can refer to a Dev Container definition in our repository
- This is the same Dev Container we would use locally or in a Codespace
- There are many other ways to build

```
- name: Pre-build dev container image 🚧  
  # Reuse the same Dev Container definition we open locally.  
  uses: devcontainers/ci@v0.3  
  env:  
    # Feed build-time values into the Dockerfile / devcontainer template.  
    FROM_IMAGE: ${ inputs.from_image }  
    FROM_VARIANT: ${ inputs.from_variant }  
    USERNAME: ${ inputs.username }  
    UID: ${ inputs.user_id }  
    GID: ${ inputs.group_id }  
  with:  
    # Build from one container folder in this repository.  
    subFolder: containers/my_container  
    # Publish to a predictable path in GitHub Container Registry.  
    imageName: ghcr.io/${ steps.gh_repo.outputs.name_lowercase }/my_container  
    imageTag: latest  
    # One manifest, two CPU families.  
    platform: linux/arm64/v8,linux/amd64  
    # The runner is ephemeral, so publish before it disappears.  
    push: always
```


On D-in-D and Moby

- Go to your sandbox terminal and run the container image we just published
- Wait! How we run docker container inside another container?
- DooD (Docker-out-of-Docker)
 - Mount the host Docker socket `/var/run/docker.sock` to the parent container
 - Run child containers directly on the host
- DinD (Docker-in-Docker)
 - Uses some "hackity-hack" 🧑🔧🐳 (Moby) to have a full Docker installation inside a parent container
 - Run child containers inside the parent container
- Both have advantages and disadvantages
 - D-in-D is already [preinstalled in AVD image](#)

D-in-D with Moby



Interacting with The Container via CLI

- Go to your sandbox terminal and use

- `docker run --rm -it -v $(pwd):/home/avd/workspace -w /home/avd/workspace ghcr.io/<username>/ac5-workshop/my_container:latest zsh`

- Note following errors:

```
$ rich
zsh: command not found: rich
$ touch test.tmp
touch: cannot touch 'test.tmp': Permission denied
```

1. Failing `rich` command

- We must specify `init.sh`. Better way - add ENTRYPOINT
- ENTRYPOINT - is the command/script that is executed when a container starts

2. `Permission denied`

- UIDs are incredibly important when dealing with `non-root users` on `most` operating systems and must match
- `Root is not good!`
- Tools like VSCode can re-build images to fix UID, but we'll simply build another container image

A survival guide to SELinux, the kernel and permissions

- Never deploy your labs on "random" machines!
- Containers do NOT provide full isolation and host machine is part of your environment
- Understand UIDs, permissions and the impact of the root user account
- Avoid SELinux and other great security features, when not strictly required
 - Focus on lab isolation and low lifetime instead
 - When you must – build a good level of understanding first! Deploying random stuff on SELinux hurts!
- Pick a friendly Linux distribution – Debian / Ubuntu
- Watch your kernel!
 - Also during upgrades!
- Kernel challenge examples:
 - 6.10 Linuxkit with CONFIG_TCP_MD5SIG disabled. cEOS-lab BGP password authentication breaks
 - legacy ip_tables are not loaded by default. This was breaking [VSCode D-in-D helper script](#)

```
# list loaded kernel modules
lsmod
# check kernel version
uname -a
# check kernel configuration
cat /lib/modules/$(uname -r)/config
```

Entering The Matrix

- Matrices is a GitHub actions feature that allows permutations of various parameters
- We'll use a parent workflow with matrix to trigger the child workflow making the build
 - Reusable workflows feature
- Create `.github/workflows/build_parent_matrix.yml`

```
---
name: build container images

on:
  push:
    paths:
      - .github/workflows/build_child.yml
      - .github/workflows/build_parent_matrix.yml
      - containers/lab/**

permissions:
  packages: write

jobs:
  build-lab-containers:
    uses: ../.github/workflows/build_child.yml
    strategy:
      matrix:
        from_image: ["ghcr.io/aristanetworks/avd/universal"]
        from_variant: ["python3.12-avd-v6.1.0"]
        user_id: ["1000", "1009"]
        include:
          - user_id: "1000"
            group_id: "1000"
          - user_id: "1009"
            group_id: "1009"
    with:
      from_image: ${ matrix.from_image }
      from_variant: ${ matrix.from_variant }
      user_id: ${ matrix.user_id }
      group_id: ${ matrix.group_id }
```

Add Lab Container Definition

- containers/lab/.devcontainer/Dockerfile

```
ARG FROM_IMAGE
ARG FROM_VARIANT

FROM ${FROM_IMAGE}:${FROM_VARIANT}

ARG USERNAME
ARG UID
ARG GID

USER root

# This magic sed edits are very important
# They allow to build a container with different UID/GID of a base
ENV OLD_GID="1000"
RUN sed -i -e "s/\(${USERNAME}\:[^:]*\)[^:]*:[^:]*/\1${UID}:${GID}/" /etc/passwd; \
    sed -i -e "s/\([^:]*:[^:]*\)[^:]*:[^:]*/\1${OLD_GID}:/\1${GID}:/" /etc/group; \
    chown -R $UID:$GID /home/$USERNAME

# copy code-server entrypoint
COPY ./entrypoint.sh /bin/entrypoint
RUN chmod +x /bin/entrypoint
ENTRYPOINT [ "/bin/entrypoint" ]

USER ${USERNAME}
```

- Do NOT forget `chmod +x` for the entrypoint.sh ➡

- Add containers/lab/.devcontainer/devcontainer.json

```
{
  "build": {
    "dockerfile": "Dockerfile",
    "args": {
      "FROM_IMAGE": "${localEnv:FROM_IMAGE}",
      "FROM_VARIANT": "${localEnv:FROM_VARIANT}",
      "USERNAME": "${localEnv:USERNAME}",
      "UID": "${localEnv:UID}",
      "GID": "${localEnv:GID}"
    }
  }
}
```

- Add basic ENTRYPOINT containers/lab/.devcontainer/entrypoint.sh

```
#!/usr/bin/env bash

set +e

# on Codespaces this will not work correctly
if ! ${CODESPACES:-false}; then
  # Execute command from docker cli if any.
  if [ ${@+True} ]; then
    exec "$@"
  # Otherwise just enter sh or zsh.
  else
    if [ -f "/bin/zsh" ]; then
      exec zsh
    else
      exec sh
    fi
  fi
fi
```

Building a Child Workflow

- Rename `build_image.yml` to `build_child.yml`
- Change `on:` section:

```
on:
  workflow_call:
    inputs:
      from_image:
        required: false
        type: string
        default: ghcr.io/aristanetworks/avd/universal
      from_variant:
        required: false
        type: string
        default: latest
      username:
        required: false
        type: string
        default: avd
      user_id:
        required: false
        type: string
        default: 1000
      group_id:
        required: false
        type: string
        default: 1000
```

- Edit Pre-build dev container image in `jobs:` `ci` section:

```
- name: Pre-build dev container image
  uses: devcontainers/ci@v0.3
  env:
    FROM_IMAGE: ${ inputs.from_image }
    FROM_VARIANT: ${ inputs.from_variant }
    USERNAME: ${ inputs.username }
    UID: ${ inputs.user_id }
    GID: ${ inputs.group_id }
  with:
    subFolder: containers/lab
    imageName: ghcr.io/${ steps.gh_repo.outputs.name_lowercase }/lab
    imageTag: uid-${ inputs.user_id }
    platform: linux/arm64/v8,linux/amd64
    push: always
```

- Test with:
 - `docker run --rm -it -v $(pwd):/home/avd/workspace -w /home/avd/workspace ghcr.io/<username>/ac5-workshop/lab:uid-1009`
 - `touch test.tmp`

Tip: Make sure to exit the container!

Add Containerlab and Code Server

- Add following to the Dockerfile `containers/lab/.devcontainer/Dockerfile`

```
# install the latest containerlab
RUN bash -c "$(curl -sL https://get.containerlab.dev)"
# install coder
RUN curl -fsSL https://code-server.dev/install.sh | sh -s -- --version="4.115.0"
```

- Update the ENTRYPOINT `containers/lab/.devcontainer/entrypoint.sh` . Add after `set +e`

```
if [ -z "${CODE_SERVER_BIND_ADDR}" ]; then
    CODE_SERVER_BIND_ADDR="0.0.0.0:5000"
fi
code-server --bind-addr ${CODE_SERVER_BIND_ADDR} --auth password --disable-telemetry --disable-update-check --disable-workspace-trust "${CONTAINERWSF}" &
```

- Add revision build_child.yml workflow:

```
imageTag: uid-${{ inputs.user_id }}-rev0.1
```

- Run the new lab container and connect to Code Server via port 5000:

```
# set any secret string as PASSWORD for authentication (can be random in prod)
docker run --rm -it --privileged --name lab --detach -w /lab -v /sandbox/lab_dir:/lab -v /var/lib/docker -e PASSWORD=labpass124 -p 5000:5000 ghcr.io/<username>/ac5-workshop/lab:uid-1009-rev0.1
```

The Inception: Container Style

- Let's document all the nesting in our setup:
 - GCP VM -> acLabs lab-base container -> ac5-workshop/lab container
 - Code Server, Containerlab, cEOS-lab lab containers and more ...
- It's amazing, that setup like this can actually work
- However, we can't access the "inner" Code Server as the "outer" instance will intercept all requests
- A true engineer mind can not simply accept this fact

```
docker run \  
  --name cloudflare-tunnel \  
  --network host \  
  cloudflare/cloudflared:latest \  
  tunnel --url http://127.0.0.1:5000
```

- Alternative:

```
ssh -p 443 -o ServerAliveInterval=30 -o ServerAliveCountMax=3 -R0:127.0.0.1:5000 free.pinggy.io
```

💀 WARNING 💀: Never use in prod! This slide was just another dream.



The Lab Inventory

- This is NOT a Containerlab training. Check [the docs](#) when in doubt.
- We'll simply copy AVD example inventories supplied with every AVD container image
 - L3LS

```
mkdir -p l3ls-lab
cp -r /home/avd/.ansible/collections/ansible_collections/arista/avd/examples/single-dc-l3ls/* l3ls-lab/
```

- L2LS

```
mkdir -p l2ls-lab
cp -r /home/avd/.ansible/collections/ansible_collections/arista/avd/examples/l2ls-fabric/* l2ls-lab
```

- We can now switch between different labs by specifying `-v /sandbox/<the-lab-name>:/lab` in the `docker run` command
- We could build a "fancier" setup and upload lab inventories to GitHub Pages as Artifacts
 - Reason - download option for portability
 - Let's keep setup simple for this workshop!

Working With Containerized Network OS Images

!! WARNING !!

!! avoid committing images to your repository, as this violates EULA !!

!! `echo "*.tar.gz" >> .gitignore` !!

- All vendors are different
 - Similar approach can likely be applied to other images, but we take no responsibility
- Fortunately cEOS-lab is one of the best network os images available
 - It's enough to register, `accept EULA` and download the image
 - In the workshop we'll use:
 - On the parent container: `docker save arista/ceos:latest | gzip > ceos_lab.tar.gz; cp ceos_lab.tar.gz l3ls-lab`
 - On the child container:
 - Start the docker daemon inside the container `/usr/local/share/docker-init.sh`
 - `docker load < ceos_lab.tar.gz` (`docker import` if you work with downloaded image)
- `-v /var/lib/docker` is very important for performance!

Image Import With ENTRYPOINT

- Update the ENTRYPOINT `containers/lab/.devcontainer/entrypoint.sh` and build rev0.2 image.

- Add this after the `code-server` block

```
# run magic moby script for D-in-D
/usr/local/share/docker-init.sh

# check if ceos-lab image already present
if [ -z "$(${CONTAINER_ENGINE} image ls | grep 'arista/ceos')" ]; then
    docker load < ceos_lab.tar.gz
    echo "WARNING: cEOS-lab image was successfully loaded."
fi

# start the lab
make start
```

- Create make shortcut to start and stop the lab. Use TABs, not spaces!

```
CURRENT_DIR := $(shell pwd)

.PHONY: help
help: ## Display help message
    @grep -E '^[0-9a-zA-Z_-]+\.[0-9a-zA-Z_-]+:.*?## .*$$' $(MAKEFILE_LIST) | sort | awk 'BEGIN {FS = ":.*?## ";} {printf "\033[36m%-30s\033[0m %s\n", $$1, $$2}'

.PHONY: l3ls
l3ls: ## Deploy l3ls lab
    cp ceos_lab.tar.gz l3ls-lab
    docker run --rm -it --privileged --name l3ls -w /lab -v $(CURRENT_DIR)/l3ls-lab:/lab -v /var/lib/docker -e PASSWORD=labpass124 -p 5000:5000 ghcr.io/<username>/ac5-workshop/lab:uid-1009-rev0.2

.PHONY: l2ls
l2ls: ## Deploy l2ls lab
    cp ceos_lab.tar.gz l2ls-lab
    docker run --rm -it --privileged --name l2ls -w /lab -v $(CURRENT_DIR)/l2ls-lab:/lab -v /var/lib/docker -e PASSWORD=labpass125 -p 5000:5000 ghcr.io/<username>/ac5-workshop/lab:uid-1009-rev0.2
```


Add Deeplink API

- Let's use a small AI generated script to emulate deeplink-like API
- Yes! We use AI in this workshop! 🎉🤖
- `https://127.0.0.1/<lab-name>` will start specific lab
- Install Python requirements
 - `pip install fastapi uvicorn`
- Add `--detach` to your Make shortcuts for every lab
- Tunnel ports 5000 and 5001
- `python3 lab.py` (keep the tab open to check logs)
- (Optional): Add `https://<tunnel-url>/<lab-name>` to your README
- Go and click that link!

```
import subprocess
from fastapi import FastAPI, HTTPException
import uvicorn

app = FastAPI()

@app.get("/{lab_name}")
def run_lab(lab_name: str):
    if any(char in lab_name for char in [";", "&", "|", ".", "/"]):
        raise HTTPException(status_code=400, detail="Invalid lab name")

    try:
        result = subprocess.run(
            ["make", lab_name], capture_output=True, text=True, check=True)
        return {"status": "success", "output": result.stdout}
    except subprocess.CalledProcessError as e:
        raise HTTPException(status_code=500, detail=f"Make failed:\n{e.stderr}")

if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=5001)
```

Use VSCode Tasks to Customize Lab Look-and-Feel

- VSCode / Code Server is the best base you can imagine for your lab
- We can further tweak and customize our labs with very low effort
- Let's add following to our setup:
 - Containerlab extension
 - A random Python script that will greet us on lab start

```
// .vscode/tasks.json
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "init_lab",
      "command": "python3 greet.py",
      "type": "shell",
      "presentation": {
        "reveal": "always"
      },
      "runOptions": {
        "runOn": "folderOpen"
      }
    }
  ]
}
```

What to Do Next

- Get a VM, experiment, learn and ask questions
- Build similar setup @work
 - Modify any building blocks! Nothing is set in stone as long as it works!
 - Beware of licensing restrictions, but cEOS-lab can be used as long as it does NOT leave customer premises
- Build a small collection of pre-defined labs for your use cases, replicate your network (full or fragments)
 - Key requirement: spinning labs for everyday use should be effortless!
 - Git init everything and commit if important! No on-box save!
 - Use your labs for testing, demos, integrate into your slides for change approval
 - Or post mortem 😊
- Promote this to your colleagues and management
- Integrate to you CI pipeline if you have one
 - Especially great for PR review!
- Get promoted!



Q&A

- [Ansible AVD](#)
- [labs.arista](#)
- [acLabs](#)
- [This repository](#)

- One more slide!
+ No more slides.

```
git commit -m "The END!"
```