02

CTI1301 - VCO (VIRTUALIZATION, CLOUD, AND OPERATIONS)

Containerized WordPress Deployment on Rocky Linux with Docker and Docker Compose

Installing Docker Engine on a Proxmox-hosted Rocky Linux VM and standing up a two-container WordPress + MySQL stack with port-mapped access

COURSE
CTI1301 - VCO (Virtualization, Cloud, and Operations)

DATE
Section C202411; lab executed Sun Nov 17, 2024 (19:06 EST per terminal date output)

ENVIRONMENT
Proxmox VE 8.1.4 virtualization cluster (Full Sail IT lab, node IT127)

PAGES (REPORT)	STUDENT	ORGANIZATION
19	Cody Richard	Full Sail University

TECHNICAL BRIEF · EXECUTIVE SUMMARY

Containerized WordPress Deployment on Rocky Linux with Docker and Docker Compose

For this final Docker lab I installed Docker Engine from source on a Rocky Linux VM running on the Full Sail IT lab Proxmox VE 8.1.4 cluster (VM 289 "CodyRichard-Rocky," 4 GB RAM, 4 cores, accessed via noVNC at itlab.fsemergingtech.com), then used Docker Compose to deploy a complete WordPress application stack. I added the official Docker CE repository, installed and enabled the docker service, wrote a docker-compose.yml defining WordPress and MySQL 8.0 containers, and ran the stack with docker compose up -d. I verified both containers (wordpress-wordpress-1 and wordpress-db-1) were running via docker ps, confirmed the port mapping 0.0.0.0:8080->80/tcp, and inspected the host and Docker bridge networking (ens18 at 10.1.102.52/22, docker0 at 172.17.0.1/16). The deployment was validated end to end by reaching the WordPress installation and admin portal from a separate Windows 10 Pro lab VM through the host:port URL 10.1.102.52:8080, then cleanly powering down all VMs.

▸ Objectives

- Install Docker Engine (Docker CE) on a Rocky Linux host using the official Docker repository
- Enable and verify the docker systemd service so containers persist across reboots
- Author and execute a docker-compose.yml to orchestrate a multi-container WordPress + MySQL stack
- Use Docker port mapping to expose the containerized web app to the lab network
- Validate the running deployment via docker ps, host networking, and a browser-based install/admin walkthrough
- Demonstrate full container lifecycle including a clean shutdown of all lab VMs

▸ Environment

```
Proxmox VE 8.1.4 virtualization cluster (Full Sail IT lab, node IT127)
```

```
Rocky Linux guest VM 289 'CodyRichard-Rocky' - 4 GB RAM, 4 sockets/4 cores (SandyBridge), VMXNET3/virtio NIC on vmbr0
```

```
Console access via noVNC over itlab.fsemergingtech.com (QEMU/KVM)
```

```
Host networking: ens18 10.1.102.52/22; Docker bridges docker0 172.17.0.1/16 and br-4dd4f7bb5c55 172.18.0.1/16
```

```
Separate Windows 10 Pro VM (VM 795, node IT122) used as the client to reach the site via Microsoft Edge
```

```
WordPress container published on host port 8080 mapped to container port 80
```

WALKTHROUGH & EVIDENCE

This lab walks through standing up a containerized WordPress stack on a Rocky Linux VM running on the Full Sail Proxmox VE 8.1.4 cluster. I install Docker Engine from the official Docker CE repository, author a docker-compose.yml to orchestrate WordPress alongside MySQL 8.0, then publish the app on host port 8080 and validate it end to end from a separate Windows 10 client. Every step is captured from the noVNC console and the client browser, from repo setup through a clean shutdown of all VMs.

ENVIRONMENT

Rocky Linux VM on Proxmox

The lab host is VM 289 "CodyRichard-Rocky" provisioned on the Full Sail Proxmox VE 8.1.4 cluster with 4 GB RAM and 4 cores, accessed over a noVNC console.

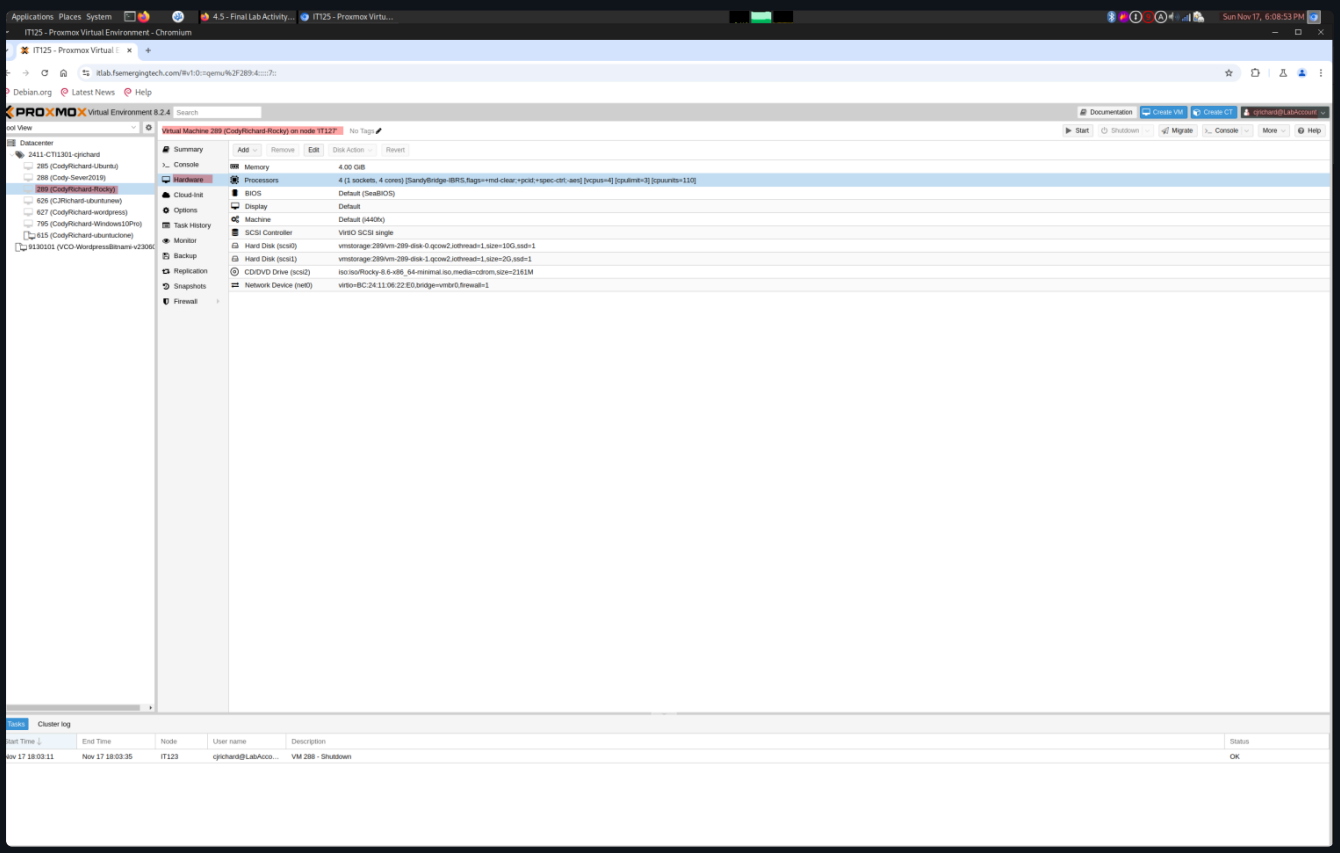

FIG 01 — Proxmox VE hardware view for VM 289 CodyRichard-Rocky, showing the Rocky Linux guest configured with 4 GB RAM and 4 sockets/cores on the Full Sail IT lab cluster.

SETUP

Repository Tooling

Before adding the Docker repository, I install yum-utils to provide yum-config-manager.

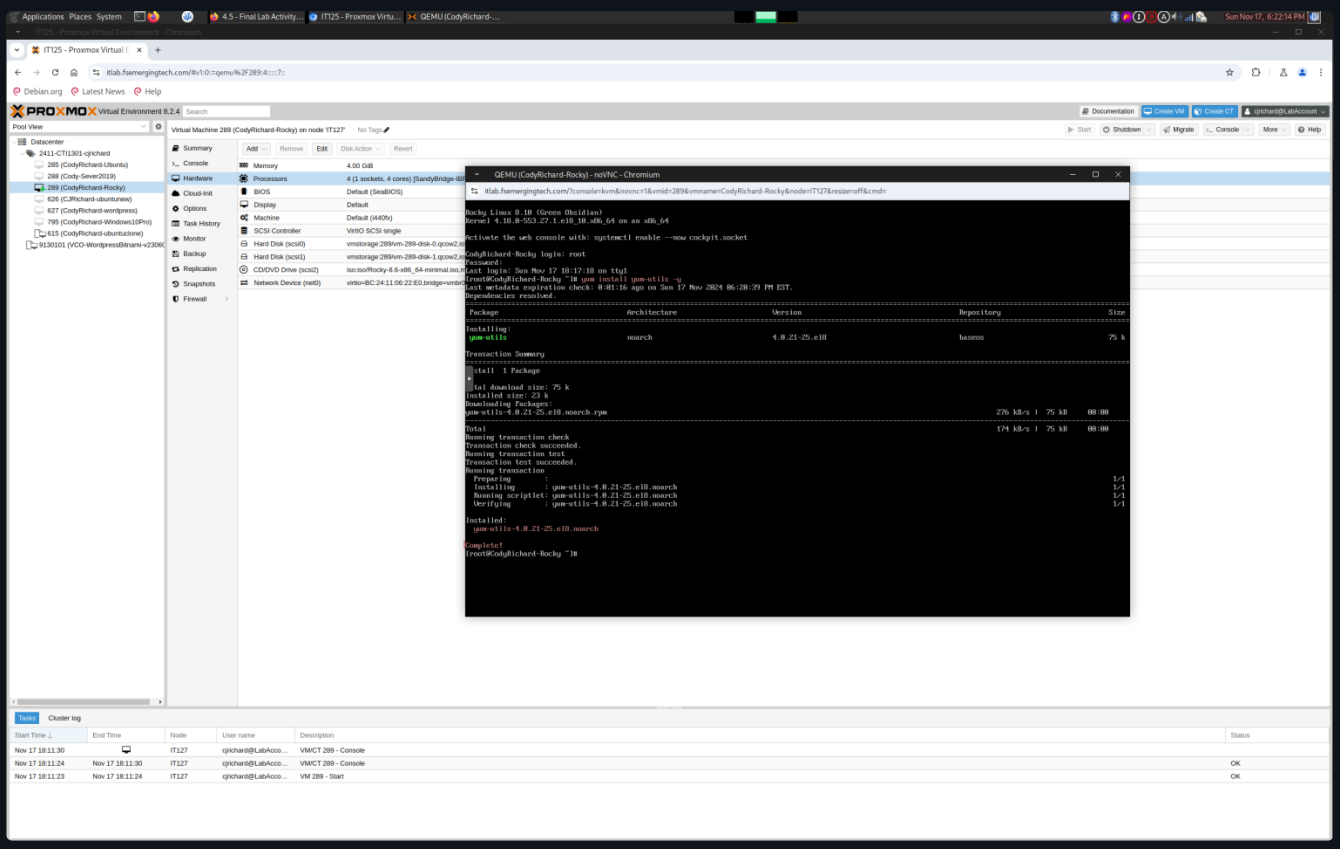

FIG 02 — Installing the yum-utils package, which supplies the yum-config-manager tooling used to register the Docker CE repository.

SETUP

Adding the Docker CE Repository

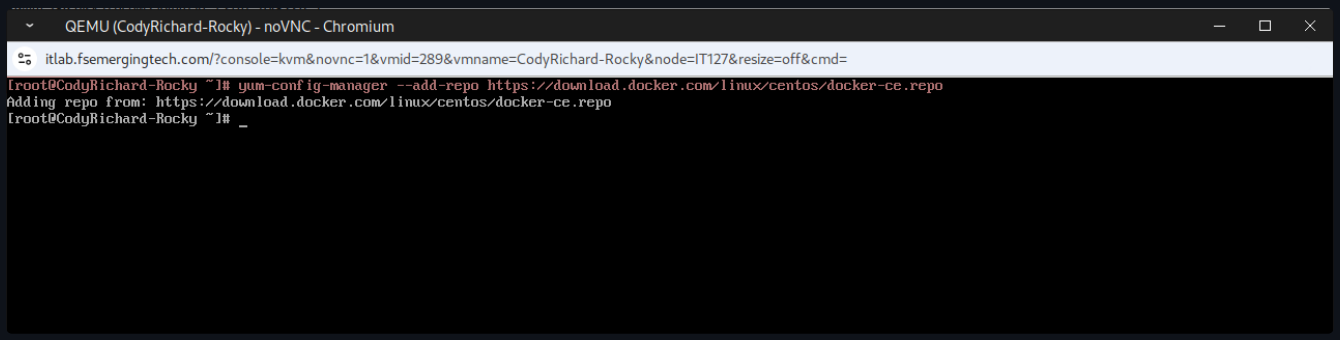

FIG 03 — Registering the official Docker repository: yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo.

INSTALL

Installing Docker Engine

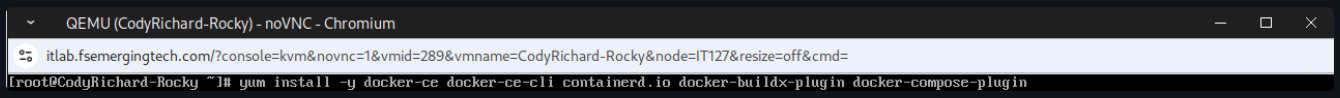

FIG 04 — Issuing the Docker Engine install command via the package manager to pull docker-ce and its dependencies from the newly added repository.

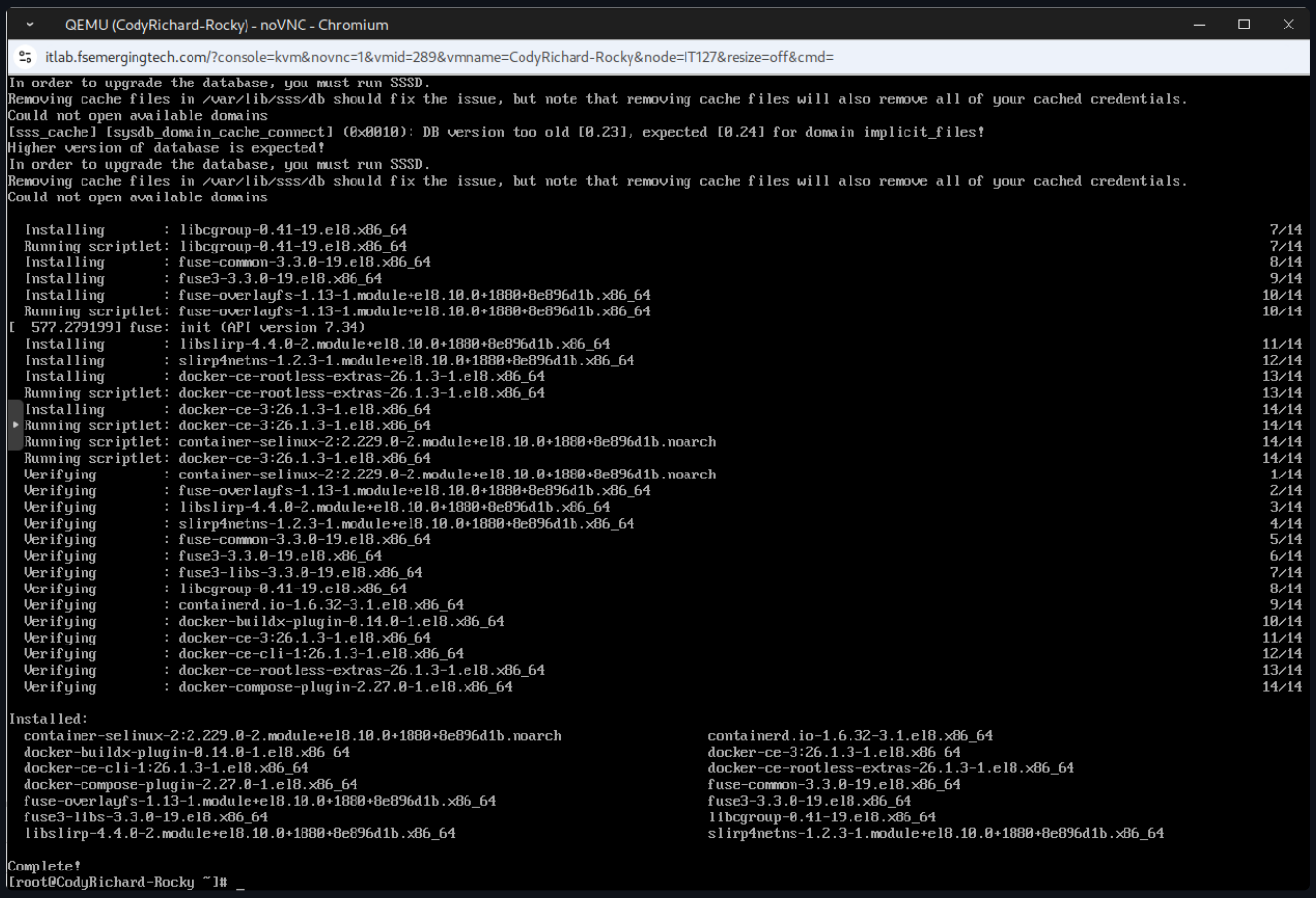

FIG 05 — Tail of the Docker install output confirming the package installation completed successfully on the Rocky Linux host.

SERVICE

Enabling the Docker Service

With the engine installed, I enable the systemd unit so containers persist across reboots, then confirm the daemon is active.

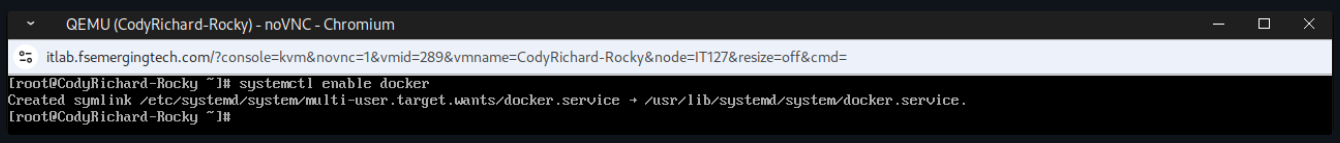

FIG 06 — Enabling the service with systemctl enable docker, which creates the multi-user.target symlink to /usr/lib/systemd/system/docker.service.

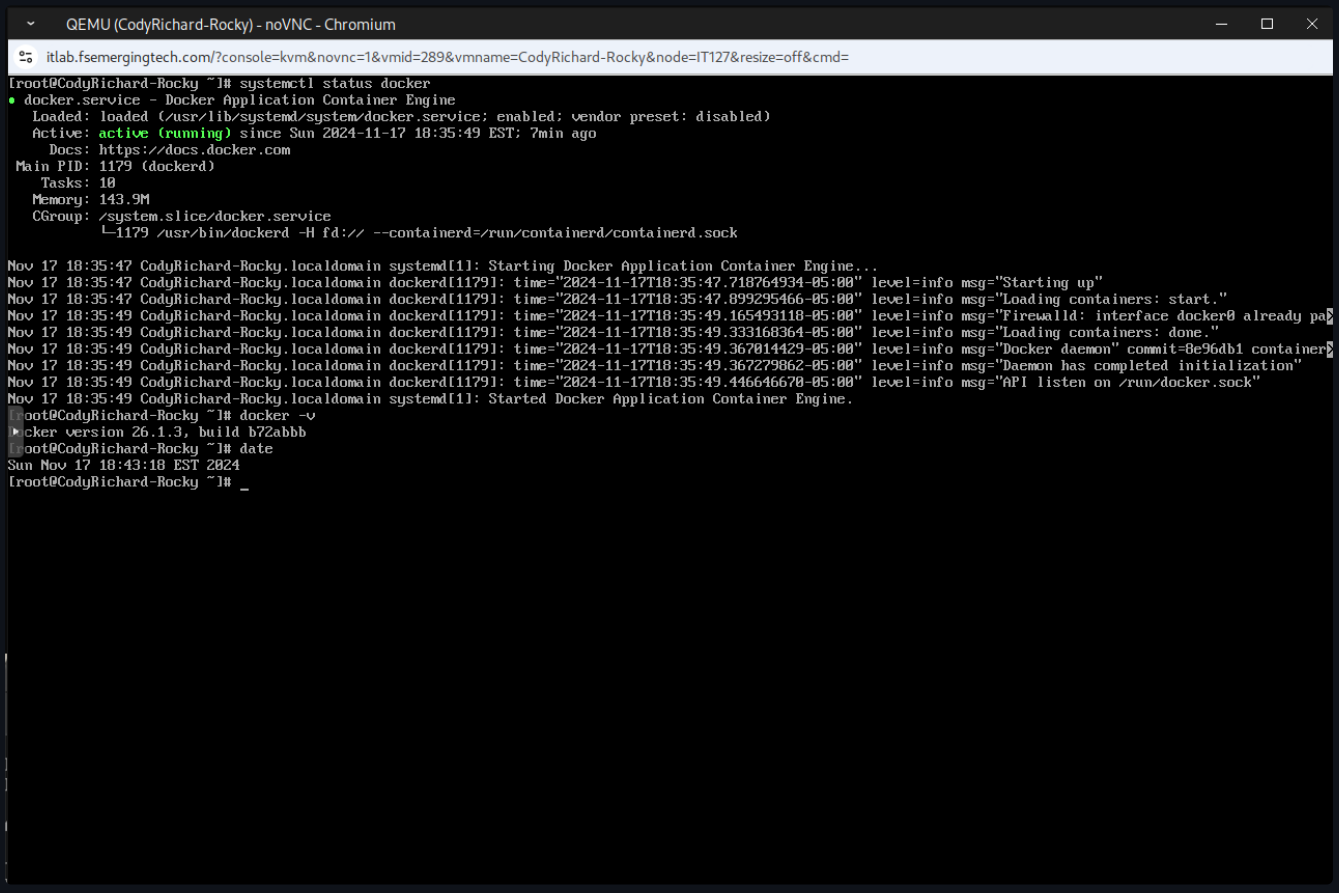

FIG 07 — Docker confirmed active, with the version output and date command verifying the running daemon on the host.

DEPLOYMENT

Preparing the Compose Stack

I install wget, then author the docker-compose.yml that defines the WordPress and MySQL 8.0 services.

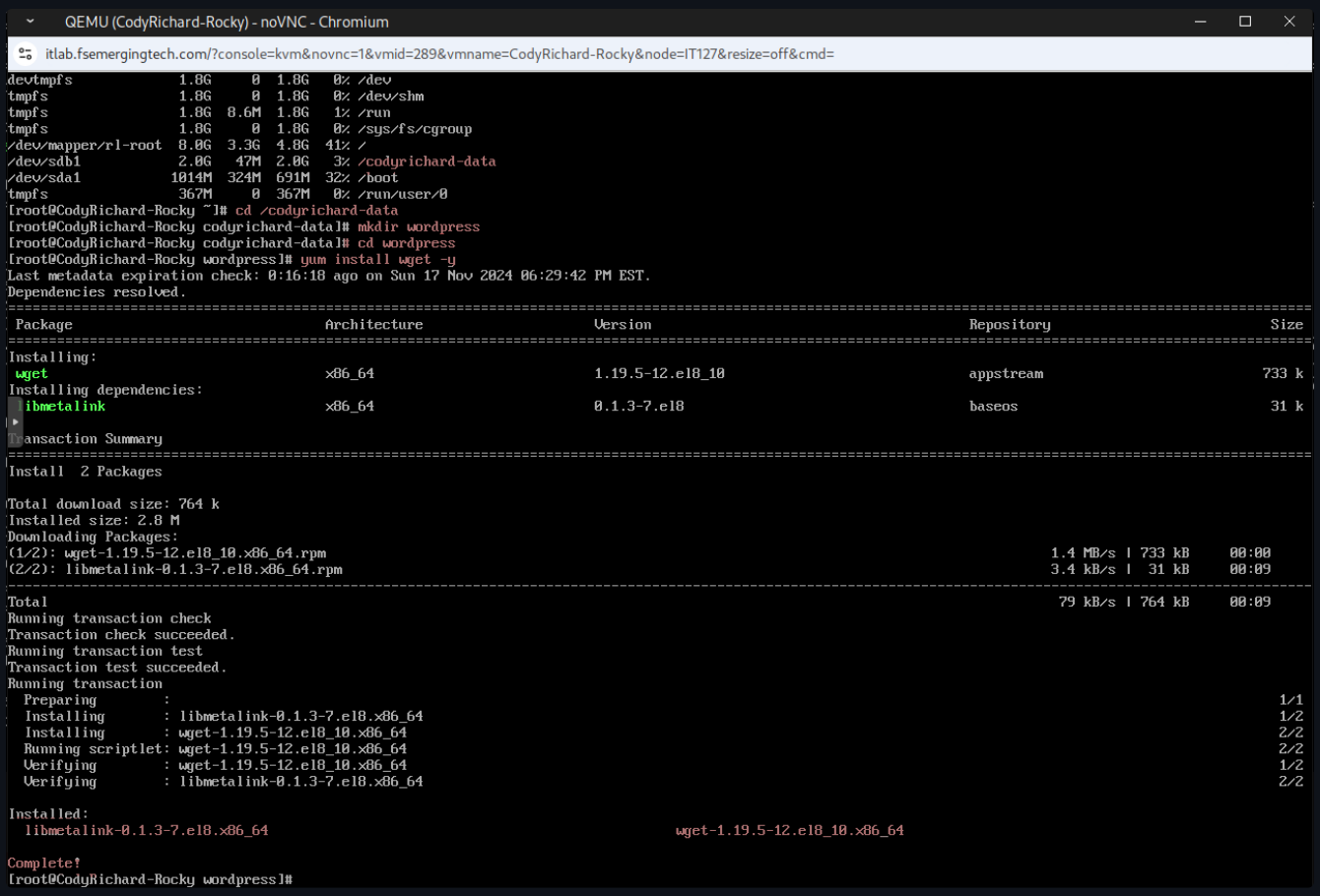

FIG 08 — Installing wget on the Rocky Linux host in preparation for staging the Compose project files.

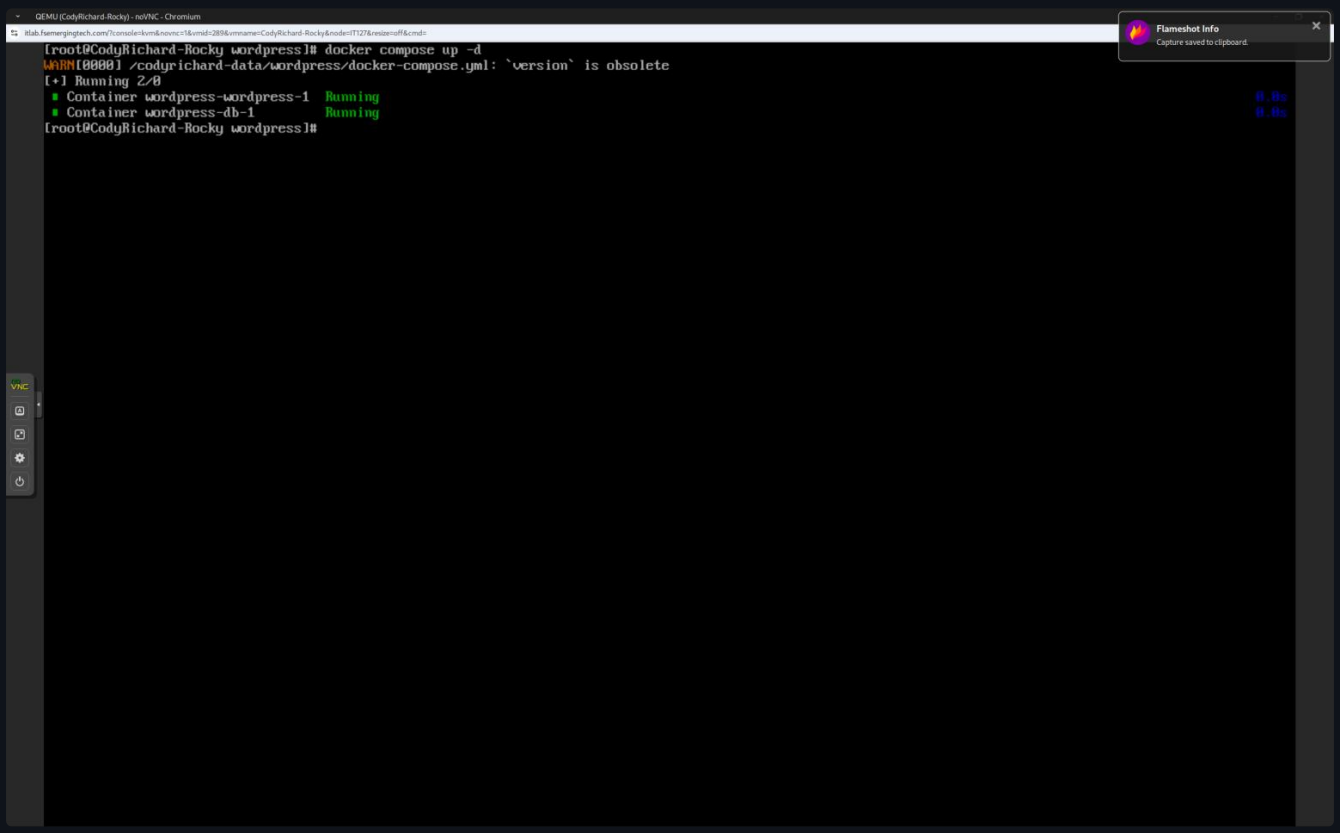

FIG 09 — The docker-compose.yml under /codyrichard-data/wordpress/ defining the WordPress and MySQL 8.0 services, deployed with docker compose up -d to bring up wordpress-wordpress-1 and wordpress-db-1.

VERIFICATION

Host State and Disk

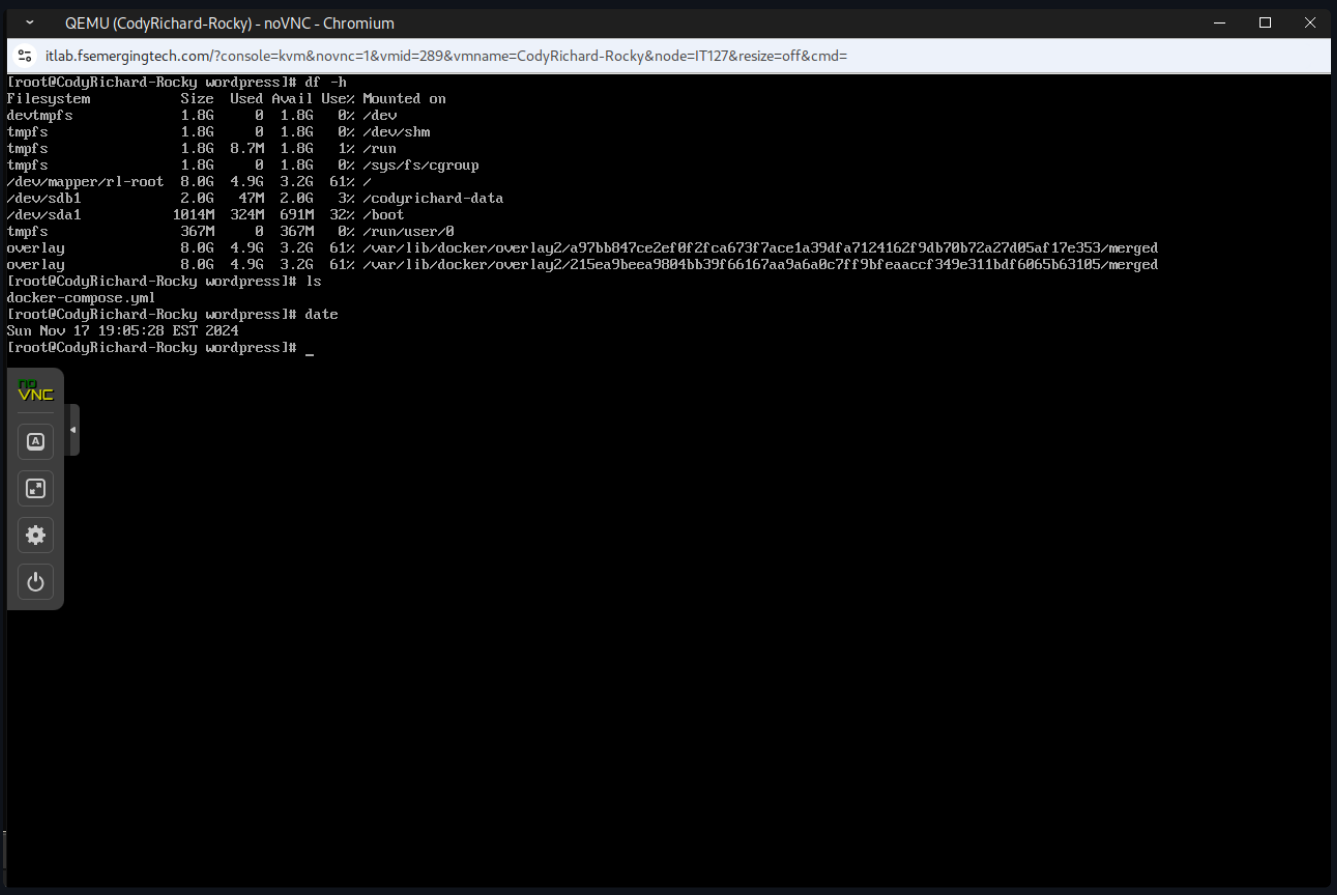

FIG 10 — Output of df -h, ls, and date, showing disk utilization and the working directory state on the host after deployment.

VERIFICATION

Containers and Networking

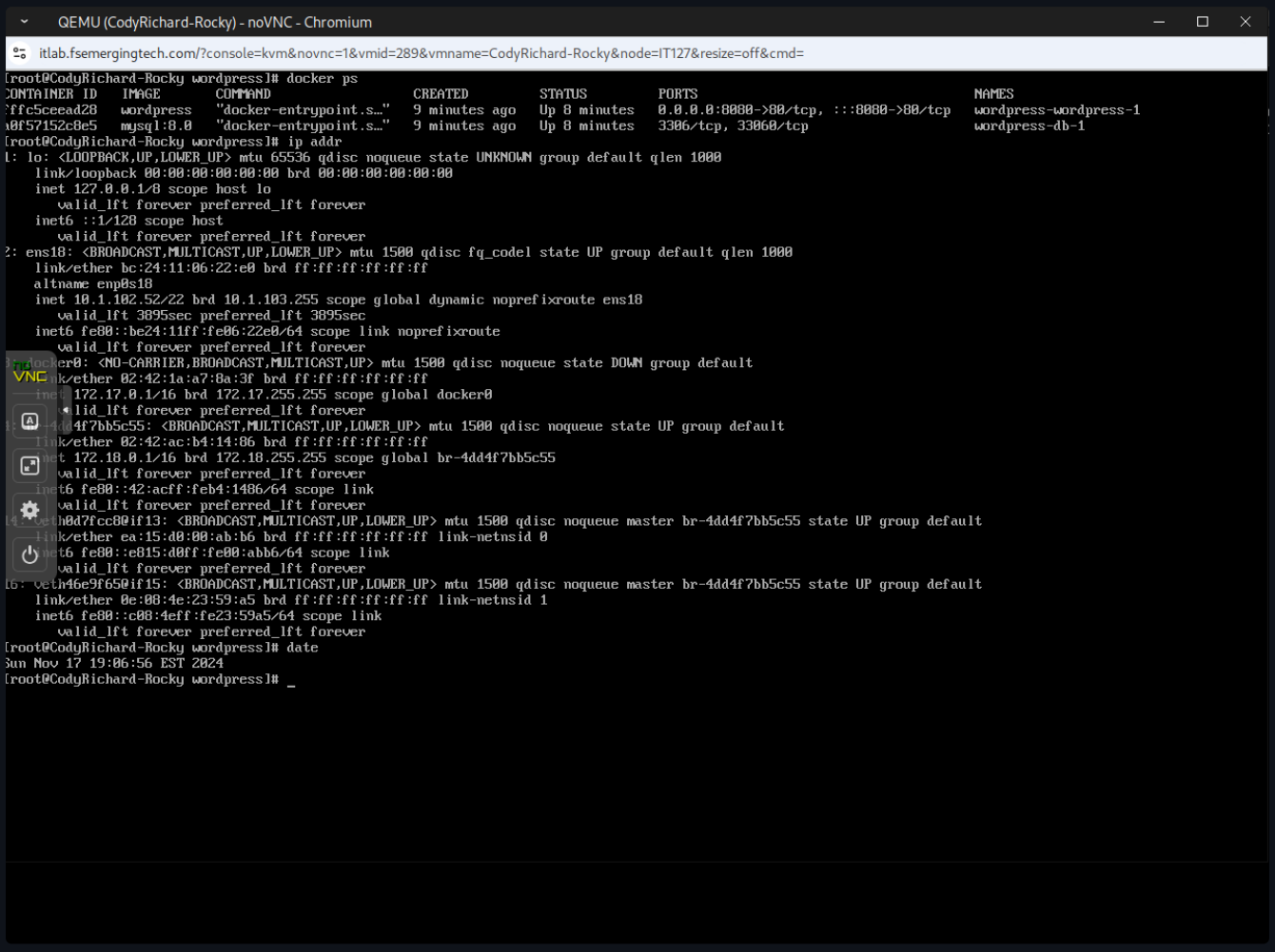

FIG 11 — docker ps with ip addr and date output, confirming both containers Running, the port mapping 0.0.0.0:8080->80/tcp, MySQL on 3306/33060, and host ens18 at 10.1.102.52/22 alongside the docker0 bridge at 172.17.0.1/16.

VALIDATION

Reaching the WordPress Installer

From a separate Windows 10 Pro client, I hit the host:port URL to confirm the published container is reachable across the lab network.

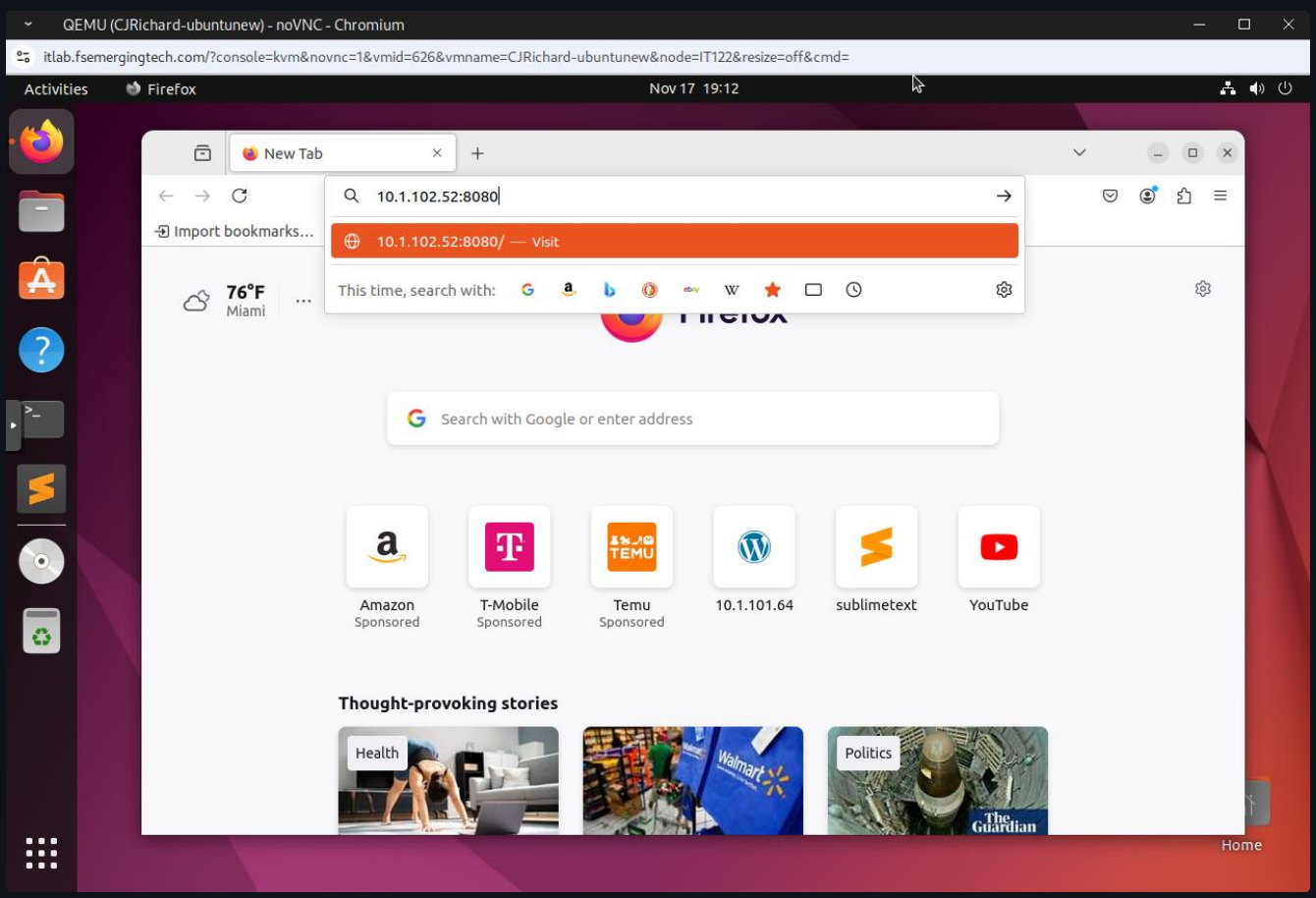

FIG 12 — Using the port mapping to load the WordPress installation page in a browser at http://10.1.102.52:8080, served by the containerized stack.

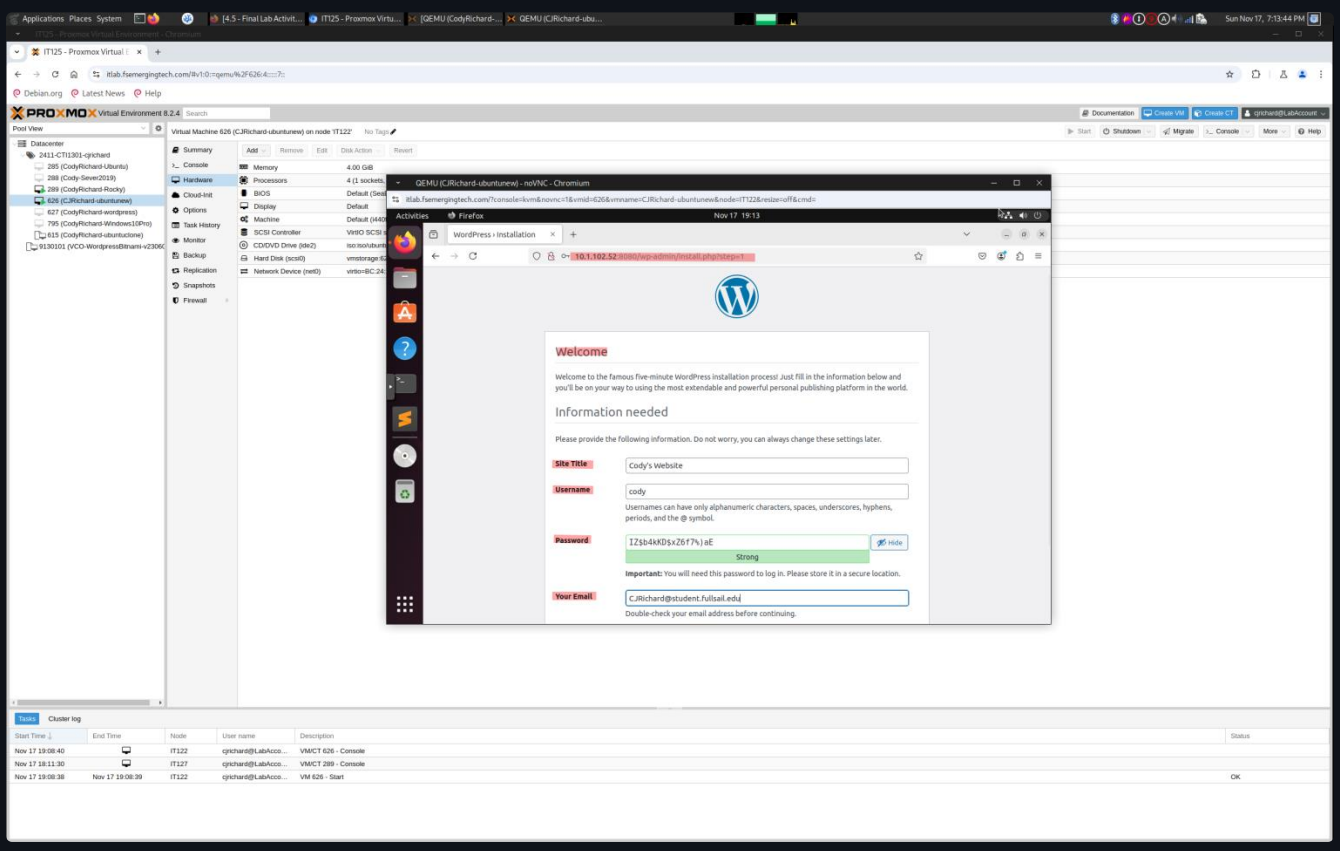

FIG 13 — The WordPress installation wizard collecting site title and admin credentials over the published port mapping.

VALIDATION

Installation Complete

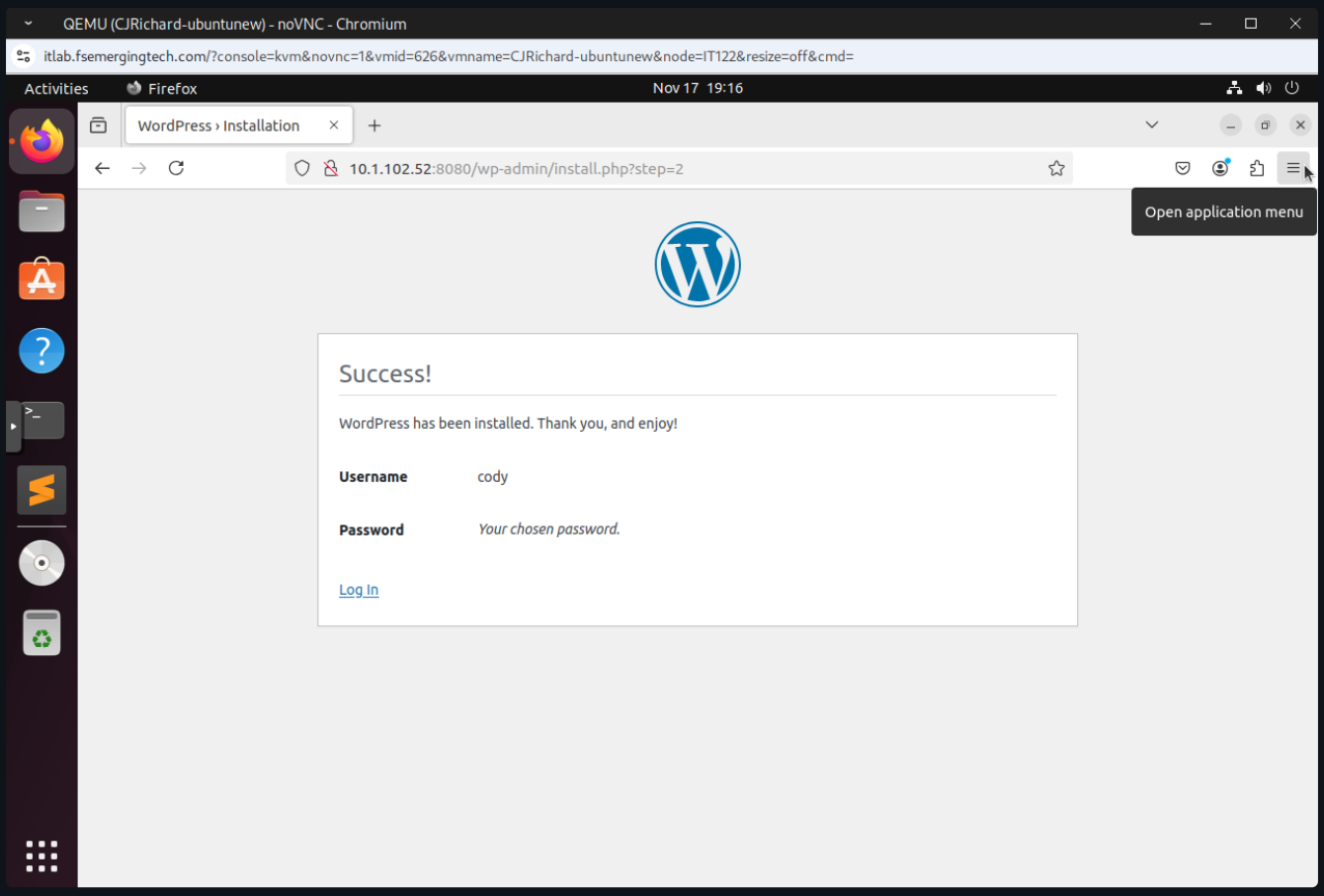

FIG 14 — WordPress reporting a successful installation, confirming the application and its MySQL 8.0 backend are wired correctly.

VALIDATION

Admin Portal

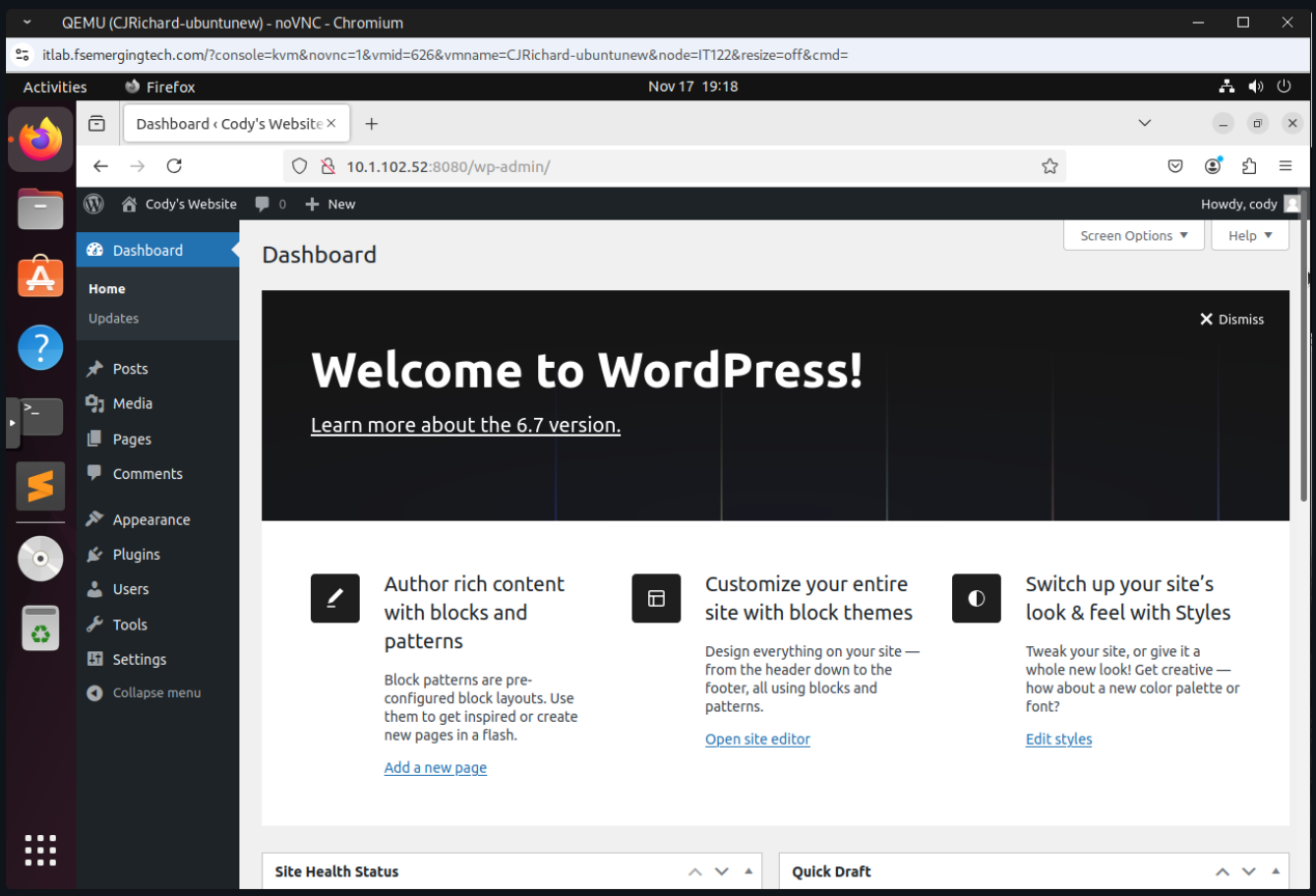

FIG 15 — The WordPress admin dashboard reached from the client, validating an authenticated session against the containerized site.

VALIDATION

Front-End Access

Final client-side checks confirm the public site loads through the same host:port mapping.

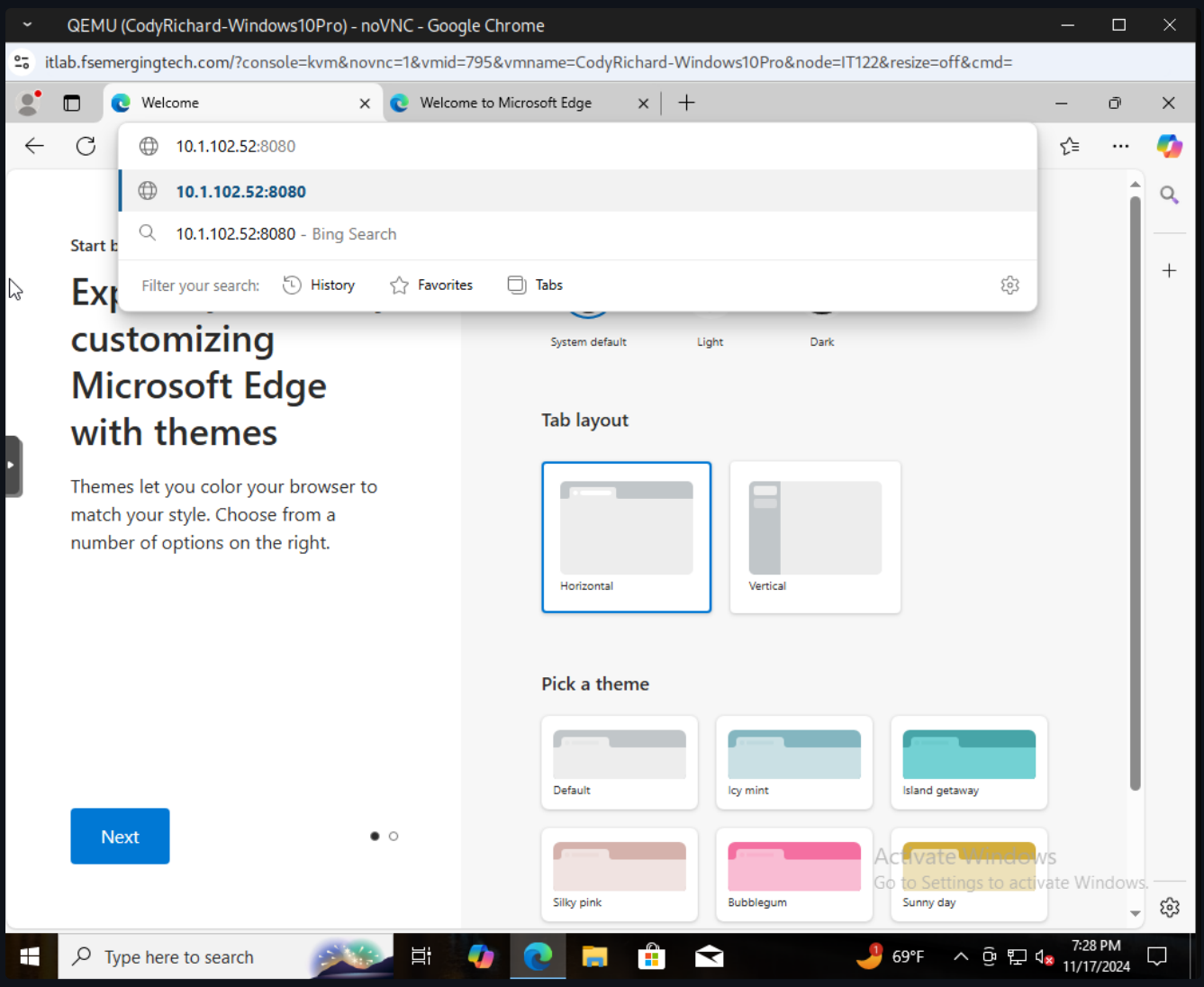

FIG 16 — The browser address bar showing the host:port URL (10.1.102.52:8080) used to reach the WordPress container from the lab network.

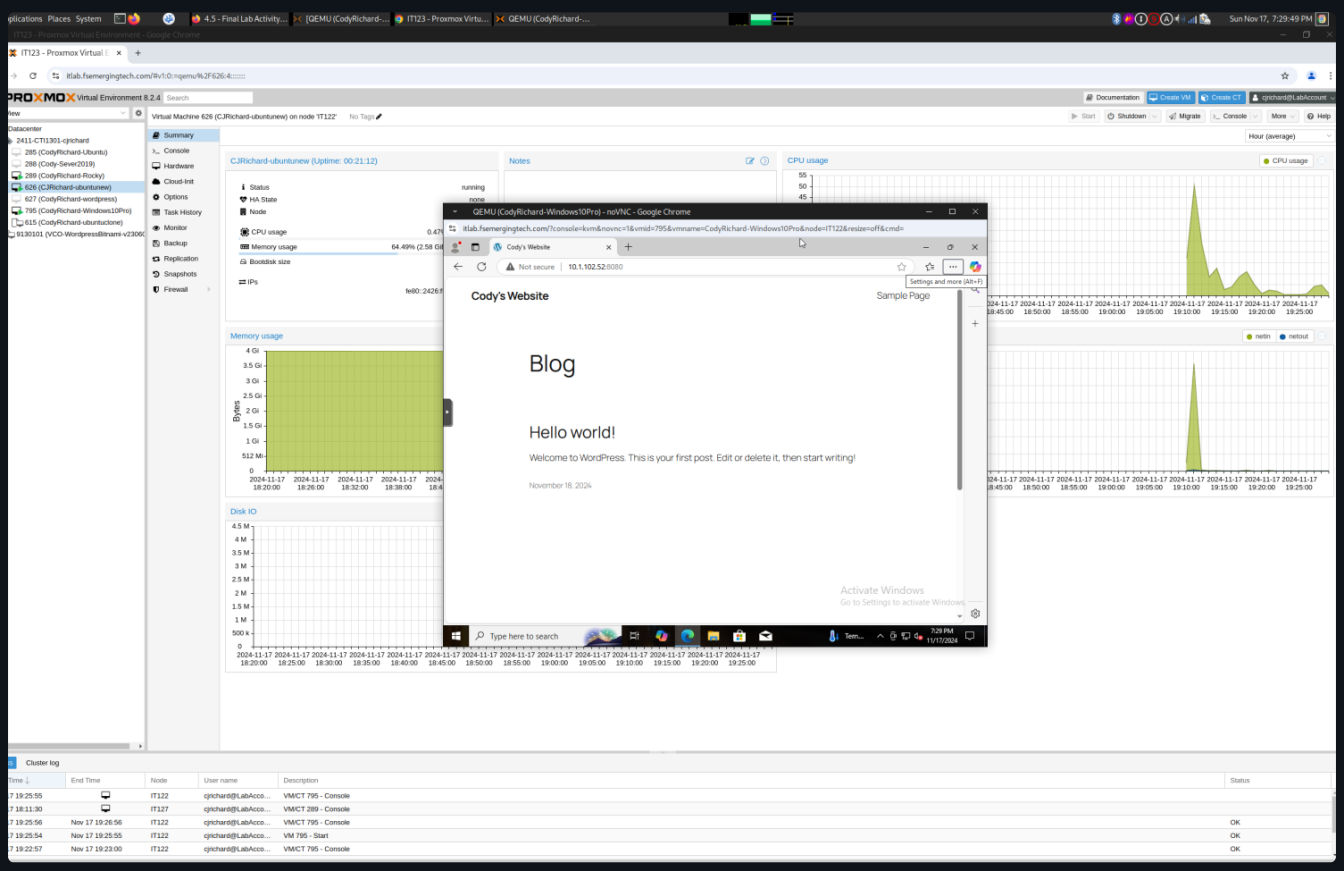

FIG 17 — The rendered WordPress front-end site loading successfully from the client, confirming end-to-end delivery from the published container.

SHUTDOWN

Clean Lab Teardown

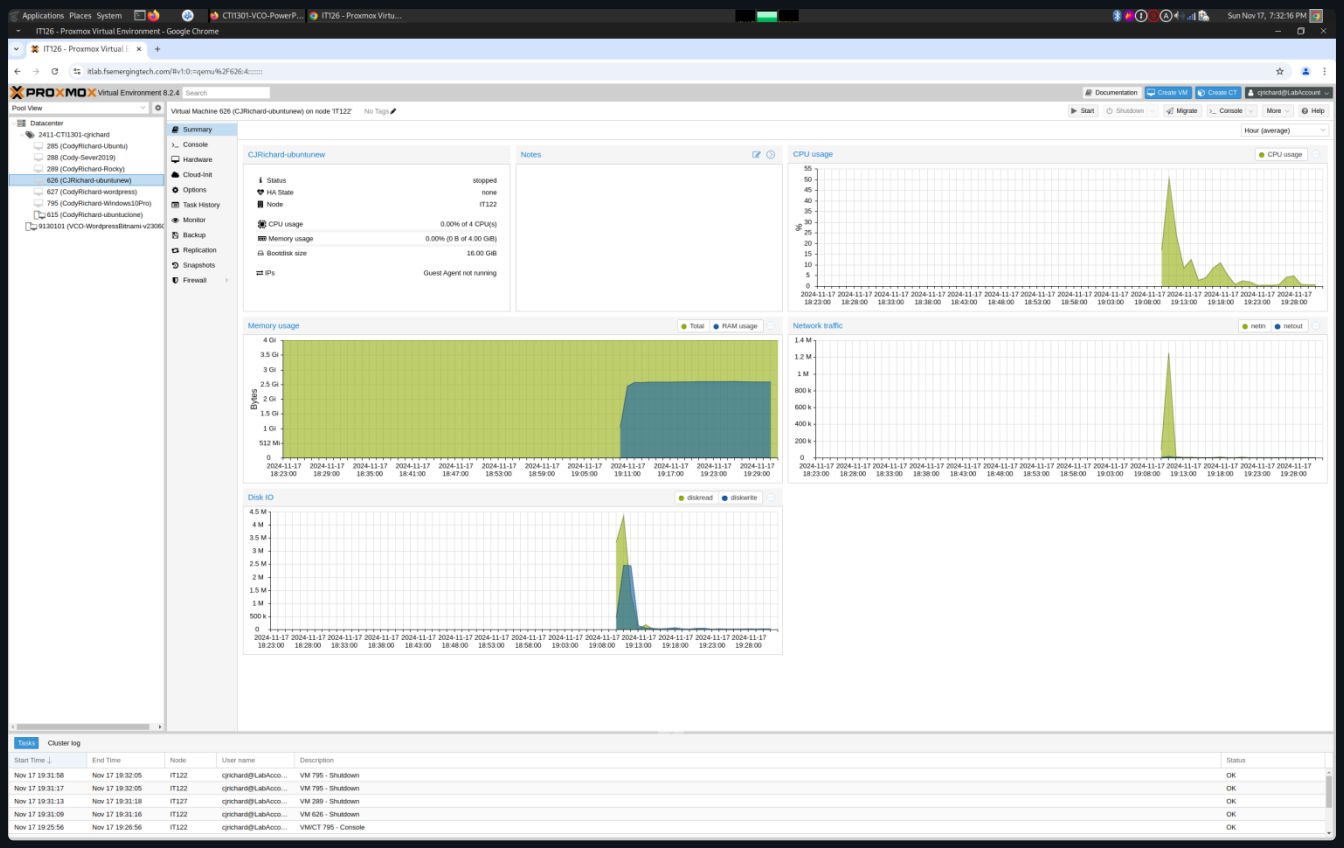

FIG 18 — Proxmox showing all lab VMs powered down, closing out the deployment with a clean shutdown.