OSOMUDEYA ZUDONU

BUILD YOUR OWN DEVOPS LAB

Build Your Own DevOps Lab

The Zero-Cost Infrastructure Guide

Virtual Box • Vagrant • Proxmox • Docker • Kubernetes • Terraform • Ansible

Complete setups, automation scripts, and real portfolio projects, no cloud bills required

A practical guide for engineers who want production-ready skills without burning money on AWS

All Code Files Available On Github:
https://github.com/DevOps-Projects-by-Zee/home-lab-zee

About This Book

You don't need a cloud account to learn DevOps. You don't need AWS credits, Azure subscriptions, or a corporate expense budget.

Your laptop sitting right there has more computing power than entire datacenters had 20 years ago. This book shows you how to use it.

What you'll build:

- Complete virtualization platforms (Proxmox or VirtualBox + Vagrant)
- Container environments (Docker + Kubernetes)
- Infrastructure automation (Terraform + Ansible)
- Real portfolio projects (LAMP stack, microservices, CI/CD, monitoring)

What you'll learn:

- Manual setups (so you understand what's happening)
- Automation scripts (so you work like a pro)
- Troubleshooting (because everything breaks)
- Real-world patterns (not toy examples)

What it costs:

- Your time: 2-4 weeks
- Cloud bills: $0
- Value: Job-ready DevOps skills

No cloud account required. No credit card needed. No surprise bills.

Just your laptop, these instructions, and the willingness to break things (and fix them).

Let's build your lab.

Connect with the author:
Email: talk2osomudeya@gmail.com
Website: shipwithzee.com
LinkedIn: https://www.linkedin.com/in/osomudeya-zudonu-17290b124/
Github: https://github.com/DevOps-Projects-by-Zee/home-lab-zee

Build Your Own DevOps Lab

Platform Compatibility: This book covers tools that work across Mac, Windows, and Linux. ***Proxmox VE (Chapter 2) requires dedicated hardware and is not compatible with macOS.*** Mac users should start with Chapter 3 (VirtualBox + Vagrant) or Chapter 4 (Docker).

First Edition, October 2025

Published independently

What You'll Become

By the end of this, you won't feel like someone "learning DevOps" anymore.

You Will Become

- Someone who can **set up environments from scratch**
- Someone who can **debug when things break**
- Someone who can **build and deploy real systems**
- Someone with **proof of work and not just certificates**

This is the difference between watching tutorials and actually doing the job.

Also:

By the end of this book, you will:

- Run your own multi-node DevOps lab locally
- Deploy real applications using Docker and Kubernetes
- Automate infrastructure with Terraform and Ansible
- Have 4 portfolio projects you can show in interviews

This is not theory, but real experience without cloud bills.

TABLE OF CONTENTS

CRITICAL: Read This First

About Proxmox

Proxmox WILL ERASE YOUR ENTIRE COMPUTER. No undo. No recovery.

Only install Proxmox on:

- Spare PC you don't use
- Old laptop collecting dust
- Dedicated hardware, you're okay wiping forever

Never install on:

- Your daily laptop
- Any Mac (won't work)
- Your only computer

No spare hardware? Skip Proxmox. Use VirtualBox + Vagrant instead (Chapter 1, Section 3): same skills, zero risk.

Mac Users: Your Path

- VirtualBox + Vagrant (Chapter 1, Section 3)
- Docker + Kubernetes (Chapter 2)
- Terraform + Ansible (Chapter 3)
- All 4 projects (Chapter 4)
- Skip Proxmox (needs separate PC hardware)

System Requirements

Minimum: 8GB RAM, 50GB disk space, Virtualization enabled.

Recommended: 16GB RAM, 100GB disk space

Introduction

Why Your Laptop Is Enough

Cloud bills add up fast. That EC2 instance you forgot? $47 for doing nothing.

The truth is, **90% of DevOps skills can be learned offline.** Your laptop has more power than datacenters had 20 years ago.

Kubernetes works the same on AWS as it does on your laptop. Docker doesn't check your credit card. Terraform code is identical everywhere.

What You'll Build

- Virtualization platform (VirtualBox/Vagrant or Proxmox)
- Containers (Docker + Kubernetes)
- Infrastructure as Code (Terraform + Ansible)
- 4 portfolio projects employers want to see

Cost: $0

Each chapter builds on the last. You don't need to finish everything in one sitting, every section is self-contained, so you can stop after any project, destroy the VMs, and pick it back up later from exactly where you left off. If something breaks and you can't find the fix in the main instructions, jump to the Troubleshooting Guide at the back. Most errors that stop beginners cold are already listed there.

Chapter 1

Your Virtualization Platform

Section 1: Choosing Your Path

Mac Users: Use VirtualBox + Vagrant (Section 3)
Windows/Linux with spare PC: Use Proxmox (Section 2) + VirtualBox
Windows/Linux without spare PC: Use VirtualBox + Vagrant (Section 3)

All paths teach identical skills.

Your laptop can run multiple full operating systems at the same time. That sounds expensive, but it isn't.
Virtualization software turns your one machine into several. This is what cloud providers do at massive scale. You're doing the same thing locally, for free. When you spin up a VM here, you're doing exactly what AWS does when you launch an EC2 instance, minus the invoice.

You're about to turn your machine into a machine that runs machines.

Section 2: Proxmox (Spare Hardware Only)

If this is your first time installing Proxmox, watch this step-by-step setup:
Don't Install Proxmox Until you see this! (11 steps)

This will help you avoid common issues like:

- Boot problems
- Network interface errors
- Hardware compatibility

Requirements:

- Separate PC/laptop (NOT your daily machine)
- Intel/AMD x86_64 processor
- Another device to access the web interface
- Accepted that this machine will become Proxmox only forever

Important:
Proxmox does NOT support WiFi during installation.
You need a **wired Ethernet connection**.

If you don't have this:
👉 Skip Proxmox and use VirtualBox (same skills, no issues)

Quick Install

Step 1: Create a Bootable USB

Download ISO: https://www.proxmox.com/en/downloads

Windows: Use Rufus (https://rufus.ie),
Mac: Use Etcher (https://etcher.balena.io), **Linux:**

```
sudo dd if=proxmox-ve_9.0-1.iso of=/dev/sdX bs=4M status=progress
```

Step 2: Boot and Install

1. Boot from USB (press F12/F2 during startup)
2. Select "Install Proxmox VE (Graphical)"
3. Accept EULA
4. Select disk (THIS ERASES EVERYTHING)

5. Choose filesystem (ext4 recommended)
6. Set timezone
7. Set root password (WRITE IT DOWN)
8. Configure network:
    - IP: 192.168.1.100/24 (example)
    - Gateway: 192.168.1.1 (your router)
    - DNS: 8.8.8.8
9. Install (takes 5-15 minutes)

Step 3: Access Web UI

From another computer, visit: https://192.168.1.100:8006

Click through the security warning. Login: root / (your password)

Step 4: Post-Install

In web UI, click Shell and run:

```
# Remove subscription nag
sed -i.backup "s/data.status !== 'Active'/false/g"
/usr/share/javascript/proxmox-widget-toolkit/proxmoxlib.js &&
systemctl restart pveproxy

# Fix repositories
echo "# deb https://enterprise.proxmox.com/debian/pve bookworm
pve-enterprise" > /etc/apt/sources.list.d/pve-enterprise.list
echo "deb http://download.proxmox.com/debian/pve bookworm
pve-no-subscription" >
/etc/apt/sources.list.d/pve-no-subscription.list

# Update system
apt update && apt dist-upgrade -y

# Install tools
apt install -y htop vim curl wget
```

Done! Proxmox is ready.

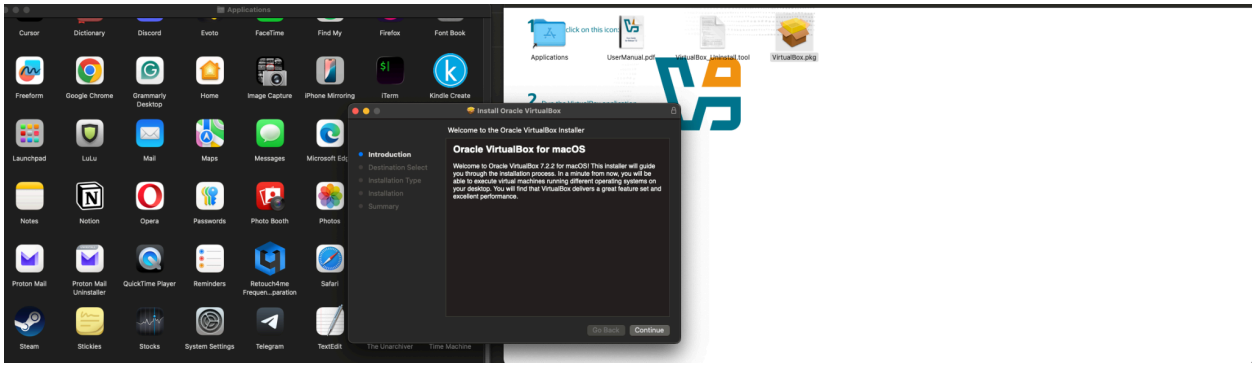
Section 3: VirtualBox + Vagrant (Universal)

Works on Mac, Windows, and Linux.

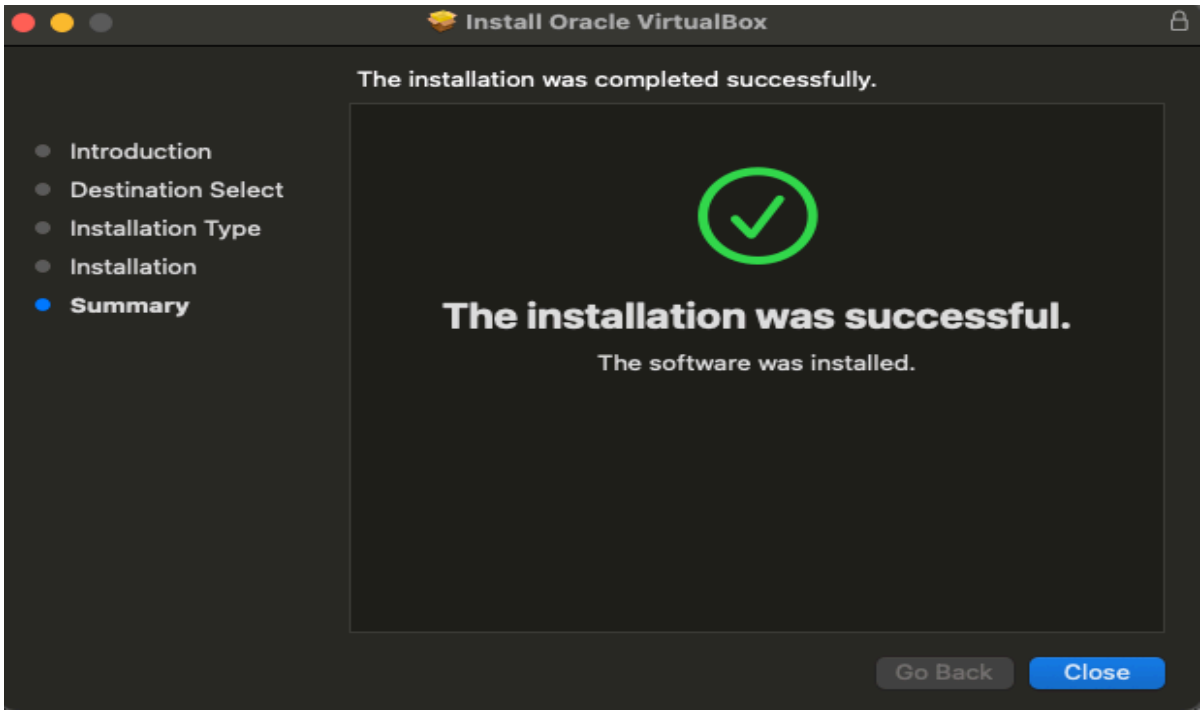
Install VirtualBox

All platforms: Download from https://www.virtualbox.org/wiki/Downloads

Windows: Run the installer, accept the network warning.
Mac: Install, grant security permissions in System Preferences.

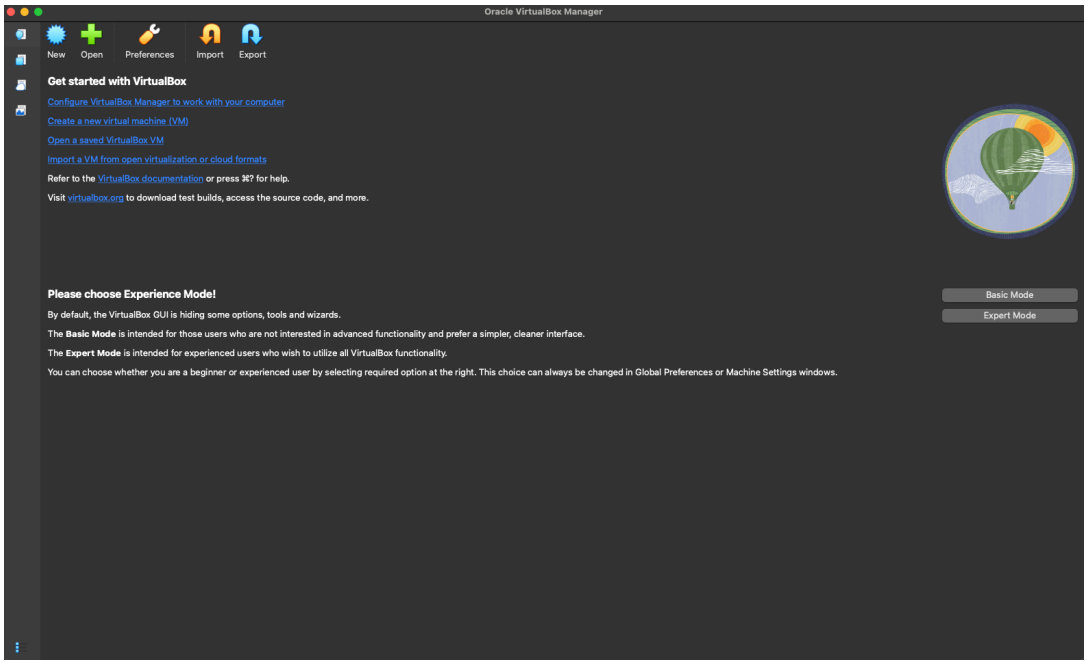


Installation in Progress



Installation Complete

https://github.com/DevOps-Projects-by-Zee/home-lab-zee



VirtualBox Now In My Apps

https://github.com/DevOps-Projects-by-Zee/home-lab-zee

For Linux (Ubuntu):

```
sudo apt install virtualbox -y
```

Verify: vboxmanage --version

Output:

```
mac@Zee-MacBook-Pro ~ % vboxmanage --version
7.2.2r170484
```

Install Vagrant

Windows (PowerShell as Admin):

```
choco install vagrant #Chocolatey is just a Windows package manager (like apt on Linux)
```

Mac:

```
brew install vagrant
```

Output:

```
==> Downloading https://releases.hashicorp.com/vagrant/2.4.1/...
==> Installing vagrant
🍺  /usr/local/Cellar/vagrant/2.4.1: 1,234 files, 256MB
```

Linux (Ubuntu):

```
wget
https://releases.hashicorp.com/vagrant/2.4.1/vagrant_2.4.1-1_amd64.deb
sudo dpkg -i vagrant_2.4.1-1_amd64.deb
```

Restart the terminal. **Verify:** vagrant --version

```
Output:
Vagrant 2.4.9
```

Set up Your First VM

```
mkdir ~/lab && cd ~/lab
vagrant init ubuntu/jammy64
```

Output:

```
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment!
```

```
#Run
vagrant up
```

Output (first time):

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'ubuntu/jammy64 ' could not be found. Attempting to find and
install...
==> default: Loading metadata for box 'ubuntu/jammy64'
==> default: Adding box 'ubuntu/jammy64' (v20231215.0.0) for provider: virtualbox
==> default: Downloading: https://vagrantcloud.com/ubuntu/boxes/focal64/...
    default: Progress: 23% (Rate: 8.5MB/s, Estimated time remaining: 0:01:15)
==> default: Successfully added box 'ubuntu/jammy64' (v20231215.0.0)
==> default: Importing base box 'ubuntu/jammy64'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: lab_default_1702834567890_12345
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
==> default: Mounting shared folders...
    default: /vagrant => /Users/yourname/lab
```

What happens: Downloads Ubuntu image (500MB, first time only), creates VM, starts it.

Terminal showing vagrant up output with progress

```
vagrant ssh
```

Output:

```
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-216-generic x86_64)
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
 System information as of Fri Oct 17 12:25:23 UTC 2025
System load:             0.25
  Usage of /:              3.7% of 38.70GB
  Memory usage:            21%
  Swap usage:              0%
  Processes:               121
  Users logged in:         0
  IPv4 address for enp0s3: 10.0.2.15
  IPv6 address for enp0s3: fd17:625c:f037:2:1a:a2ff:fe1e:9db7
Expanded Security Maintenance for Infrastructure is not enabled.
0 updates can be applied immediately.

Enable ESM Infra to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
The list of available updates is more than a week old.
To check for new updates, run: sudo apt update
New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

vagrant@ubuntu-focal:~$
```

You're inside Ubuntu! Try: `lsb_release -a`, `free -`

```
#To exit or leave vagrant, run

exit

vagrant halt
```

Output:

```
logout
Connection to 127.0.0.1 closed.
```

Then:

```
==> default: Attempting graceful shutdown of VM...
```

Custom VM with Web Server

Now let's customize that Vagrantfile to actually do something useful.

IMPORTANT: You already have a Vagrantfile from the vagrant init command; you will see it when you run the **ls** command in the lab folder. Now you'll **replace its contents** with this configuration.

Open the Vagrantfile in your text editor:

```
# Mac/Linux
nano Vagrantfile

# Or use any editor you like:

# code Vagrantfile  (VS Code)

# vim Vagrantfile   (Vim)

# open -a TextEdit Vagrantfile  (Mac TextEdit)


# use CTRL+Shift+6 to mark the beginning of your block

# move the cursor with arrow keys to the end of your block, the text will be
highlighted.

#use CTRL+K to cut/delete a block.
```

Delete everything in the file and replace it with:

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/jammy64"
  config.vm.hostname = "webserver"
  config.vm.network "forwarded_port", guest: 80, host: 8080
```

```
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 1
  end

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y nginx
    echo "<h1>Hello from Vagrant!</h1>" > /var/www/html/index.html
  SHELL
end
```

Save and close the file (in nano: Ctrl+X, then Y, then Enter).

What this configuration does:

- config.vm.box = Use Ubuntu 20.04
- config.vm.hostname = Name the VM "webserver"
- config.vm.network "forwarded_port" = Connect VM port 80 to your port 8080
- vb.memory = "1024" = Give VM 1GB RAM
- vb.cpus = 1 = Give VM 1 CPU core
- config.vm.provision = Automatically install nginx and create a web page

What this does:

- Creates Ubuntu VM
- Installs the nginx web server automatically
- Forward VM port 80 to your port 8080

```
# destroy previous Vagrant resources and spin up our custom file

vagrant destroy -f && vagrant up
```

Output:

```
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/jammy64'...
==> default: Matching MAC address for NAT networking...
==> default: Setting hostname...
==> default: Configuring and enabling network interfaces...
==> default: Forwarding ports...
    default: 80 (guest) => 8080 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot...
==> default: Machine booted and ready!
==> default: Running provisioner: shell...
    default: Running: inline script
    default: Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
    default: Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease
    default: Reading package lists...
    default: Reading package lists...
    default: Building dependency tree...
    default: The following NEW packages will be installed:
    default:   nginx nginx-common nginx-core
    default: Setting up nginx (1.18.0-0ubuntu1.4) ...
```

Visit http://localhost:8080 - you'll see your page!

```
# If you run into port conflicts:

# run lsof -i :8080 or netstat -an | grep 8080

Modify the Vagrant file to use a different port in line 4 # host:
8080
```

Useful Commands:

```
vagrant up         # Start

vagrant halt       # Stop

vagrant destroy    # Delete

vagrant reload     # Restart

vagrant ssh        # Connect

Vagrant status     #status
```

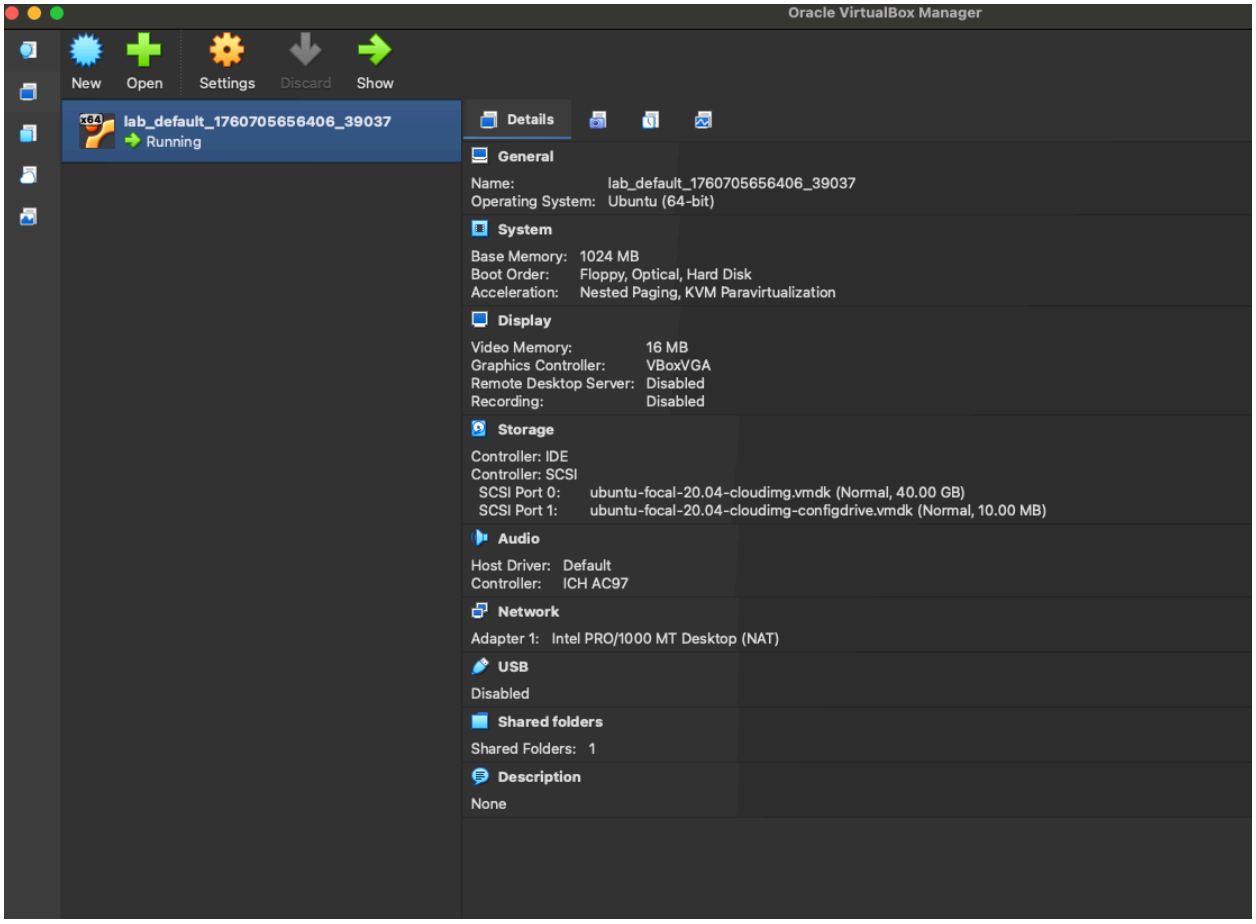
Output for vagrant status:

```
Current machine states:

default                   running (VirtualBox)

The VM is running. To stop this VM, you can run `vagrant halt`
```

Running Vm


Oracle VirtualBox Manager
New
Open
Settings
Discard
Show
lab_default_1760705656406_39037
Running
Details
General
Name: lab_default_1760705656406_39037
Operating System: Ubuntu (64-bit)
System
Base Memory: 1024 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: Nested Paging, KVM Paravirtualization
Display
Video Memory: 16 MB
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled
Storage
Controller: IDE
Controller: SCSI
SCSI Port 0: ubuntu-focal-20.04-cloudimg.vmdk (Normal, 40.00 GB)
SCSI Port 1: ubuntu-focal-20.04-cloudimg-configdrive.vmdk (Normal, 10.00 MB)
Audio
Host Driver: Default
Controller: ICH AC97
Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)
USB
Disabled
Shared folders
Shared Folders: 1
Description
None

Windows 11 + VirtualBox: Important Notes for Docker & Hostname Rendering

Windows handles file systems and virtualization differently from Linux and macOS. When running Docker inside a VirtualBox VM (or using Docker Desktop with WSL2), environment variables inside static HTML files are **not** automatically expanded unless you explicitly generate those files inside the container.

This affects the `${HOSTNAME}` output used later in the load balancer project.

Why `${HOSTNAME}` sometimes shows literally in the browser

Static HTML files do *not* process environment variables. Unless the HTML file is generated inside the container (for example using `envsubst` or a startup script), the browser will display:

```
Server: ${HOSTNAME}
```

even though the backend is correct.

Why this happens more often on Windows

Windows host volume mounts and VirtualBox shared folders sometimes deliver cached or unprocessed files to containers. This prevents dynamic variables from rendering correctly.

Whenever you need `${HOSTNAME}` inside a web page:

- **Generate the HTML file inside the container at startup**, so the hostname is baked into the file directly
- OR
- Use `envsubst` to convert a template into a real `index.html` before starting nginx

The simplest, most reliable method (especially for Windows users) is included as an alternative code block in **Chapter 4 → Load Balancer Setup**.

Chapter 2

Containers

Before containers, shipping software meant shipping instructions: install this version of Python, this library, this config file, and hope the server matched your laptop. It usually didn't.
Docker wraps your app and everything it needs into one package. Same package runs on your machine, your teammate's machine, a server in Frankfurt. That's the whole idea.
Kubernetes is what you use when you have more containers than you can manage by hand, it decides where they run, restarts them when they crash, and spreads traffic across them. You don't need cloud credits to learn either of these. They work the same way here.

Section 1: Docker Basics

Install Docker

Windows/Mac: Download Docker Desktop from
https://www.docker.com/products/docker-desktop

Linux (Ubuntu):

```
sudo apt update
sudo apt install -y docker.io docker compose
sudo systemctl start docker
sudo usermod -aG docker $USER
# Log out and back in for group changes to take effect
```

Log out and back in. Verify: docker --version

Output:

```
Docker version 28.5.7, build afdd53b
```

First Container

```
docker run -d -p 8080:80 --name web nginx
```

Output (first time):

```
Unable to find image 'nginx: latest' locally
latest: Pulling from library/nginx
a2abf6c4d29d: Pull complete
a9edb18cadd1: Pull complete
589b7251471a: Pull complete
186b1aaa4aa6: Pull complete
b4df32aa5a72: Pull complete
a0bcbecc962e: Pull complete
Digest: sha256:0d17b565c37bcbd895e9d92315a05c1c3c9a29f762b011a10c54a66cd53c9b31
Status: Downloaded newer image for nginx:latest
9a8f7c5e3d2b1a6c4f8e9d7c5b3a2e1f0d9c8b7a6e5d4c3b2a1f0e9d8c7b6a5
```

What this does:

- run = Create and start a container
- -d = Run in background
- -p 8080:80 = Map your port 8080 to the container's port 80
- --name web = Name it "web"
- nginx = Image to use

Visit http://localhost:8080 - nginx welcome page!

Essential Commands:

```
docker ps                    # List running
docker ps -a                 # List all
docker logs web              # View logs
docker exec -it web bash     # Shell into container
docker stop web              # Stop
docker rm web                # Remove
docker images                # List images
```

Output for docker ps:

```
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
PORTS                  NAMES
9a8f7c5e3d2b   nginx     "/docker-entrypoint...."   30 seconds ago   Up 29 seconds
0.0.0.0:8080->80/tcp   web
```

Output for Docker images:

```
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    605c77e624dd   2 weeks ago   141 MB
```

Build Custom Image

Create Dockerfile:

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Create index.html:

```
<h1>My Custom Container!</h1>
```

Build and run:

```
docker build -t myapp:v1 .
```

Output:

```
[+] Building 2.3s (8/8) FINISHED
 => [internal] load build definition from Dockerfile                    0.0s
 => => transferring dockerfile: 156B                                    0.0s
 => [internal] load .dockerignore                                       0.0s
 => => transferring context: 2B                                         0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine         1.8s
 => [1/2] FROM docker.io/library/nginx:alpine@sha256:...                0.0s
 => [internal] load build context                                       0.0s
 => => transferring context: 89B                                        0.0s
 => CACHED [1/2] FROM docker.io/library/nginx:alpine                    0.0s
 => [2/2] COPY index.html /usr/share/nginx/html/                        0.1s
 => exporting to image                                                  0.1s
 => => exporting layers                                                 0.1s
 => => writing image sha256:a3b2c1d0e9f8...                             0.0s
 => => naming to docker.io/library/myapp:v1                             0.0s
```

```
docker run -d -p 8080:80 myapp:v1
```

Output:

```
c9b8a7f6e5d4c3b2a1f0e9d8c7b6a5e4d3c2b1a0f9e8d7c6b5a4e3d2c1b0a9f8
```

You should see: