

Homework — Computing & Virtualization

Estimated time: ~60 min · No programming · Two guided tours: one virtual machine, one container

Two tool-tour exercises. The goal is to go from reading about VMs and containers to actually running one of each — even briefly — so the mental model from the theory page gets grounded in something you have touched. Neither task requires you to write any code or understand any command in depth. You are just observing: how much resource it asks for, what the UI looks like, what “running” means in practice.

Before you start

Keep a short notes file open (or grab a piece of paper). For each task, jot down answers to the lettered self-check questions. You are not turning these in — the notes are for you to confirm you actually *did* the things, not just read about them.

Both tasks involve free downloads. Neither installs anything into your system beyond the application itself. Both can be uninstalled cleanly when you are done.

The two tasks

Task 1 — Spin up a virtual machine with VirtualBox

This is the Type 2 hypervisor experience: a full virtual machine running inside a window on your current laptop, exactly as described in the theory. You will create a minimal Linux VM, start it, poke around for a moment, and note what resources it asked for.

macOS alternative (Multipass)

If you are on a Mac with Apple Silicon (M1/M2/M3/M4) and VirtualBox does not work for you, use **Multipass** instead — it is Canonical's free lightweight VM manager that runs natively on Apple Silicon. Download from multipass.run/install. The self-check questions below apply to either tool.

1. Download and install **VirtualBox** from virtualbox.org/wiki/Downloads. Choose the installer for your operating system. The install is straightforward — click through the defaults.
2. You also need a disk image (ISO file) for the guest OS. Download the **Ubuntu Server 24.04 LTS ISO** from ubuntu.com/download/server. It is about 2 GB. While it downloads, continue reading.
3. Open VirtualBox. Click **New** to create a virtual machine. Give it a name (e.g. "test-vm"), set the type to **Linux**, version **Ubuntu (64-bit)**. On the next screen, allocate **2048 MB (2 GB) of RAM** and **1 CPU**. Create a new virtual hard disk at the default size (10 GB is fine; it does not immediately use 10 GB on your real disk — it grows lazily).
4. Before starting the VM, attach the ISO: in VirtualBox, select your new VM, click **Settings → Storage → Empty** (under the optical drive icon) → click the disc icon on the right → "Choose a disk file" → select the Ubuntu ISO you downloaded.
5. Click **Start**. The VM boots into the Ubuntu installer. You do not need to complete the installation — watching the boot sequence and seeing the installer menu is enough for this exercise. Notice:
 - The boot takes several seconds. Compare that to how fast a container starts (Task 2).
 - The VM runs in its own window. Your host OS is completely unaffected by what you do inside it.
 - In VirtualBox's main window, look at the **Details** pane for your VM — it shows the allocated RAM, CPU count, and virtual disk size.
6. Close the VM window and choose "Power Off the Machine" when asked.

Self-check — answer all three:

- **(a)** How much RAM and how many CPUs did you allocate to the VM? Does the VM installer see them as if they were real hardware?

- **(b)** Roughly how long did it take from clicking Start to seeing the Ubuntu installer menu? (Time it.)
- **(c)** In the VirtualBox Details pane, can you see where the virtual hard disk file is stored on your real hard drive? Note the path — this is the “VM is just a file” concept from the theory.

✓ You have run a Type 2 hypervisor, created a VM, and seen hardware virtualization from the inside. The numbers — boot time, RAM, disk — are now something you observed, not just read.

Task 2 — Run a container with Docker Desktop

This task shows the container experience: something that starts in milliseconds, shares the host kernel, and disappears without a trace when you stop it. You will run two containers — the obligatory “hello-world” and an interactive Ubuntu shell — and then look at the Docker Desktop dashboard.

1. Download and install **Docker Desktop** from docker.com/products/docker-desktop. It is free for personal use. The installer sets up Docker’s engine and a graphical dashboard. Restart if prompted.
2. Open a terminal (Terminal on macOS/Linux; PowerShell or Command Prompt on Windows). Type the following command and press Enter:

```
docker run hello-world
```

Docker downloads a tiny image (a few kilobytes) and runs it. It prints a “Hello from Docker!” message explaining what just happened, then exits. The whole thing takes a few seconds — mostly download time on first run, fractions of a second on repeat runs.

3. Now run:

```
docker run -it ubuntu bash
```

This pulls the official Ubuntu container image and drops you into a bash shell *inside the container*. Your prompt will change to something like

```
root@a3b7c9d2e1f0:/#
```

. Try a few commands to look around:

- `ls /` — lists the root filesystem (it looks like a Linux install, but it's a container)
- `cat /etc/os-release` — shows the Ubuntu version inside the container
- `ps aux` — shows running processes. Notice there are very few — just bash and ps itself.

Type `exit` to leave the container shell.

4. Open **Docker Desktop**. Click the **Containers** tab. You will see the containers you just ran listed there, with their status (Exited). Click the **Images** tab — you can see the `ubuntu` and `hello-world` images that were downloaded, along with their sizes.

Self-check — answer all three:

- **(a)** How large is the `ubuntu` container image shown in Docker Desktop? Compare that to the 2 GB Ubuntu Server ISO from Task 1. What is the ratio? Why is the container image so much smaller?
- **(b)** When you ran `ps aux` inside the Ubuntu container, how many processes were listed? Now open your host system's task manager or Activity Monitor and note how many processes your real OS is running. What does that gap tell you about container isolation vs VM isolation?
- **(c)** How long did the second run of `docker run hello-world` take (or how fast did the ubuntu container start, after the image was already downloaded)? Compare that to the VM boot time you measured in Task 1.

✓ You have run two containers, looked inside one, and seen the Docker Desktop dashboard. The theory concepts — shared kernel, lightweight images, millisecond start — are now numbers you measured yourself.

What “done” looks like

You are done when, in your notes, you have:

- VM resource allocation (RAM, CPU), boot time, and the virtual disk file path from Task 1.

- Container image size, process count comparison, and container start time from Task 2.

The point of these two exercises is not to teach you to manage VMs or write Dockerfiles. It is to make the words “virtual machine” and “container” refer to things you have held in your hands, not just read in a diagram. The next time an engineer says “we containerized it” or “the VM was snapshotted before the deploy,” you will have a picture — and a number for how fast each one starts.

Uninstall note

Both VirtualBox and Docker Desktop are self-contained applications. To remove them: on macOS, drag them to the Trash; on Windows, uninstall via Settings → Apps. The virtual machine files (the .vdi or .vmdk disk image) are stored in your home folder under `VirtualBox VMs/` and can be deleted separately if you want to free the disk space.

Where this goes next

Lesson 10 moves to **DevOps & Deployment**: how code travels from a developer’s laptop through a CI/CD pipeline and lands on the compute infrastructure you just explored. The containers you ran in Task 2 are exactly what those pipelines package and ship.