

Homework — DevOps & Deployment

Estimated time: ~45 min · No code to run · A browser, a GitHub account, and a notebook

Three tool-tour exercises. The goal is to see the moving parts of a modern deployment pipeline with your own eyes — a real CI workflow running on a real project, a real container image in a public registry, and a real Infrastructure-as-Code file. By the end you'll have watched an automated build, browsed a Docker image's tags, and read a Terraform file in the wild — without installing anything.

Before you start

Keep a one-page notebook open (paper or a text file). For each task, jot the short answers under (a)/(b)/(c). You are not turning this in — the point is to build the muscle of *poking at* a real deployment pipeline, not just reading about one. None of these tasks require writing code or running commands.

The three tasks

Task 1 — Watch a real CI pipeline run

Open github.com. Pick any well-known open-source project — good candidates are [facebook/react](https://github.com/facebook/react), [microsoft/vscode](https://github.com/microsoft/vscode), or [vercel/next.js](https://github.com/vercel/next.js). Click the **Actions** tab at the top of the repo. (If you'd rather see CI in the context of a code change, open the **Pull requests** tab, click any open PR, and scroll to the bottom — the same workflow runs appear inline under each PR.)

Find a recent workflow run — ideally a green one that finished in the last day or two. Click into it. You'll see a list of **jobs** (often anywhere from 5 to 50+ depending on the project). Click into one job. Watch the *steps* — each one is a

small command (checkout repo, install deps, compile, run tests, upload artifact) with its own log and its own duration.

Self-check — answer all three:

- **(a)** How many jobs ran in this workflow?
- **(b)** How long did the whole workflow take from *queued* to *completed*?
- **(c)** Take a screenshot of one expanded job showing its step-by-step log.

✓ You do NOT need to understand the test output — the goal is to see *the shape of an automated pipeline running on every change*.

Task 2 — Tour a Docker image (no install required)

Open hub.docker.com — Docker's public image registry. Search for [nginx](#) and open the official **nginx** image page (the one with the "Docker Official Image" badge). Read the **Overview** tab — just skim the intro and the "How to use this image" section, don't try to follow it.

Now click the **Tags** tab. Notice that the same image has many different versions side by side: [1.27-alpine](#), [1.27-bookworm](#), [1.27-perl](#), [latest](#), and so on. Each one is a different combination of nginx version + base operating system + extras. Click into one tag and look at its *compressed size* — the [alpine](#) variants are often under 30 MB while the full Debian-based ones can be several times larger.

Self-check — answer all three:

- **(a)** Roughly how many times has this image been pulled in total? (Docker Hub shows the lifetime pull count on the image header — the number is staggering, often in the billions.)
- **(b)** Roughly how many distinct tags / versions are listed?
- **(c)** Why does one logical "image" have versions like [1.27-alpine](#) vs [1.27-bookworm](#)? (Hint: different base operating systems — different size / security trade-offs.)

✓ You do NOT need Docker installed — this is a museum tour of an image registry.

Task 3 — Read an Infrastructure-as-Code file in the wild

Open `github.com` and search for any real-world Terraform file. Two reliable shortcuts:

- In the GitHub search bar, type `language:hcl filename:main.tf` and pick any result that opens a substantial `main.tf`.
- Or browse `github.com/terraform-aws-modules` — open any module repo (e.g. `terraform-aws-vpc`) and click its `main.tf`.

Read 20–40 lines. Don't worry about syntax — just look for the shape: a `provider` block at the top (`aws` / `google` / `azurerm`) followed by a series of `resource "<type>" "<name>" { ... }` blocks. Each resource block is one piece of cloud infrastructure declared in text instead of clicked into existence.

Self-check — answer all three:

- **(a)** Which cloud is this file targeting? (Look for `provider "aws"`, `provider "google"`, or `provider "azurerm"`.)
- **(b)** Name two resources being declared (e.g. `aws_instance`, `aws_s3_bucket`, `aws_vpc`).
- **(c)** Why is declaring infrastructure this way preferable to clicking the same setup into the cloud console? (One sentence — hint: review, version control, reproducibility, from theory §7.)

✓ Take a screenshot of one declared `resource` block for your notes.

What "done" looks like

You're done when, in your notebook, you have:

- two numbers and one screenshot from Task 1 (CI pipeline)

- a pull-count number, a tag count, and a one-sentence answer from Task 2 (Docker Hub)
- a cloud name, two resource types, a one-sentence rationale, and one screenshot from Task 3 (Terraform)

The aim was never to make you a DevOps engineer in 45 minutes. It was to make sure that the next time you hear *"CI is red"* or *"we'll containerize it"* or *"that's all in Terraform"*, you have a concrete picture in your head — because you've now watched a pipeline run, browsed an image registry, and read a real IaC file.

Where this goes next

Lesson 11 closes Module 3 by zooming in on the other half of "runs and stays safe" — the security side. We'll look at the shape of modern attacks (phishing, supply chain, ransomware) and the layered defenses (identity, patching, backups, monitoring) that protect everything you've toured here.