Veebirakenduse projekt (ITI0302)

Lab 2 — Server, Docker, deploy by hand

Siim Rebane

Goal of today

- By the end of this lab:
  - You are **logged in to your own server** over SSH and can move around it
  - **nginx and Docker** are installed; `docker run hello-world` works
  - Your **Lab 1 application** runs on the server, in a container, **behind nginx**
  - `http://<ip>/api/<your resource>` answers **from your laptop**
- The lecture deck is your reference — every command today is in it
- Stuck at any step → raise your hand, that's what the lab is for

Before you start

- You submitted your **SSH public key** in Moodle; your server was provisioned from it. If you didn't, do it now and tell me — provisioning takes a minute
- You have your server's **IP address** (from me / the Moodle page)
- Your **Lab 1 project** builds on your laptop: `GET /api/<resource>` returns a list
- Open the **lecture 2 deck** beside you — you will copy commands from it

Log in

On your laptop:

```
ssh ubuntu@<ip>
```

- First time: it shows a fingerprint once — answer `yes`
- Key not in the default place: `ssh -i ~/.ssh/<file> ubuntu@<ip>`
- `Permissions … are too open` → `chmod 600 ~/.ssh/<your-private-key>`
- You are in when the prompt reads `ubuntu@...:~$`. `exit` leaves; the server keeps running

Get your bearings

Run these and read what comes back — this is the whole shell you need today:

```
pwd                 # where am I → /home/ubuntu
ls -l               # what is here → almost nothing yet, that's fine
cd /etc/nginx       # move somewhere
pwd                 # confirm you moved
cd ~                # back home
```

Tab completes names, the up arrow repeats commands, Ctrl+C stops anything stuck. If a command says `Permission denied`, it needs `sudo` in front.

Install nginx and Docker

```
sudo apt update
sudo apt install -y nginx docker.io docker-compose-v2
sudo usermod -aG docker ubuntu
```

Log out and back in after that last line — it lets you use Docker without `sudo`, and only new sessions get the change. Then:

```
docker run hello-world
```

A greeting means Docker works and your user may drive it. `permission denied` here = you didn't log out and in.

Build the jar — on your laptop

The image is built from a jar, so build it first, in your project, and name it `app.jar`:

Maven

```
./mvnw clean package -DskipTests
cp target/*.jar app.jar
```

Gradle

```
./gradlew clean bootJar
cp build/libs/*.jar app.jar
```

- `-DskipTests` today — this lab is about packaging, not tests
- Gradle: the runnable jar is the larger one, without `-plain` in the name
- Add `app.jar` to `.gitignore` — it's a build product

Write the Dockerfile

In your project root, a file named `Dockerfile` (no extension):

```
FROM eclipse-temurin:25-jre-alpine
RUN addgroup -S app && adduser -S app -G app
WORKDIR /app
COPY app.jar app.jar
RUN chown app:app app.jar
USER app
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Start from a small image that has Java's runtime, copy the jar in, run it as a user with no rights. The lecture deck explains each line.

Write compose.yaml

Next to the Dockerfile, a file named `compose.yaml`:

```yaml
services:
  app:
    build: .
    image: iti0302-app
    container_name: app
    ports:
      - "8080:8080"
    restart: unless-stopped
```

This is the long `docker run` line kept as a file, so you type `docker compose up` instead of remembering six options. `restart: unless-stopped` brings the app back after a crash or reboot.

Copy the three files to the server

On your **laptop**, from the project root:

```
ssh ubuntu@<ip> mkdir -p app
scp app.jar Dockerfile compose.yaml ubuntu@<ip>:~/app/
```

`scp` is copy over SSH; it runs on the laptop, direction laptop → server. The `:~/app/` is the destination in your home on the server. `COPY failed: app.jar not found` later means one of the three didn't land next to the others.

Run it on the server

Back on the **server**:

```
cd ~/app
docker compose up -d --build      # build the image, start the container, in the background
docker compose logs -f            # watch until "Started … in N seconds", then Ctrl+C
curl http://localhost:8080/api/<your resource>
```

An empty list `[]` is the right answer — the app runs, nothing stored yet. The same URL from your laptop does **not** answer, and it is not supposed to: 8080 is not open to the internet.

Put nginx in front

Edit the default site — `sudo nano /etc/nginx/sites-enabled/default` — and inside the `server { … }` block add:

```
location /api {
    proxy_pass http://localhost:8080;
}
```

Save (Ctrl+O, Enter), exit (Ctrl+X), then:

```
sudo nginx -t                    # syntax check — never reload without it
sudo systemctl reload nginx
```

From your **laptop**:

```
curl http://<ip>/api/<your resource>
```

nginx owns port 80 and forwards `/api` to the app on 8080. Now the world reaches it.

If you have time: a name and HTTPS

```
# after you create an A record pointing <name> at your <ip>:
sudo apt install -y certbot python3-certbot-nginx
sudo certbot --nginx -d <name>
curl https://<name>/api/<your resource>
```

A free name: GitHub Student Developer Pack (a real domain), or freedns.afraid.org / DuckDNS (a subdomain). certbot proves the name is yours, installs the certificate, and rewrites nginx to use it. Port 443 is already open on your server.

Common snags

Symptom	Fix
`permission denied` on `docker …`	log out and back in after `usermod`
`http://<ip>:8080` refused from the laptop	correct — go through nginx on port 80
`COPY failed: app.jar not found`	the jar wasn't copied next to the Dockerfile, or has another name
`address already in use` on 8080	an old container still runs: `docker ps`, then `docker stop <name>`
nginx returns `502`	the app isn't up on 8080: `docker compose ps`, `docker compose logs`
`nginx -t` fails	a missing `;` or `}` in the file you edited

You are done when

- `ssh ubuntu@<ip>` logs you in and you can move around
- `docker run hello-world` works without `sudo`
- Your app runs: `curl http://localhost:8080/api/<resource>` on the server returns a list
- `http://<ip>/api/<resource>` returns the same list **from your laptop**, through nginx
- Ahead of everyone? Add the name and HTTPS, then read the CI/CD basics

Next week

- The five by-hand steps from today — build, copy, build image, swap container, check — become a **pipeline** that runs them on every push
- That is where `GET /api/version` finally works: the pipeline stamps the version in
- Keep your server and your `~/app` directory as they are; the pipeline deploys to exactly this setup