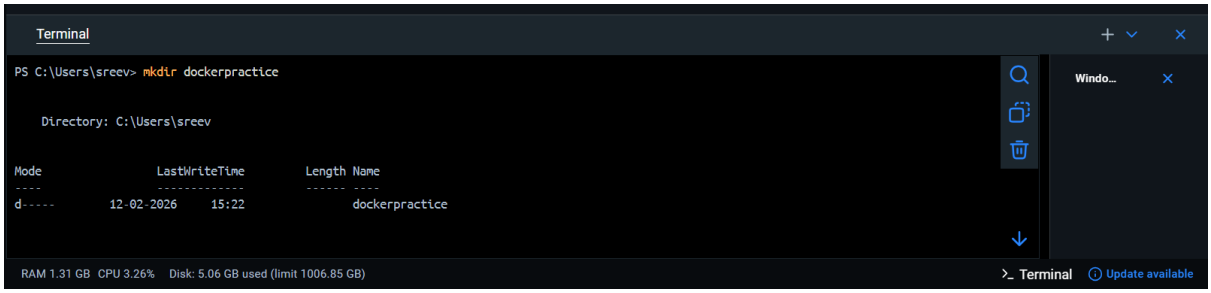
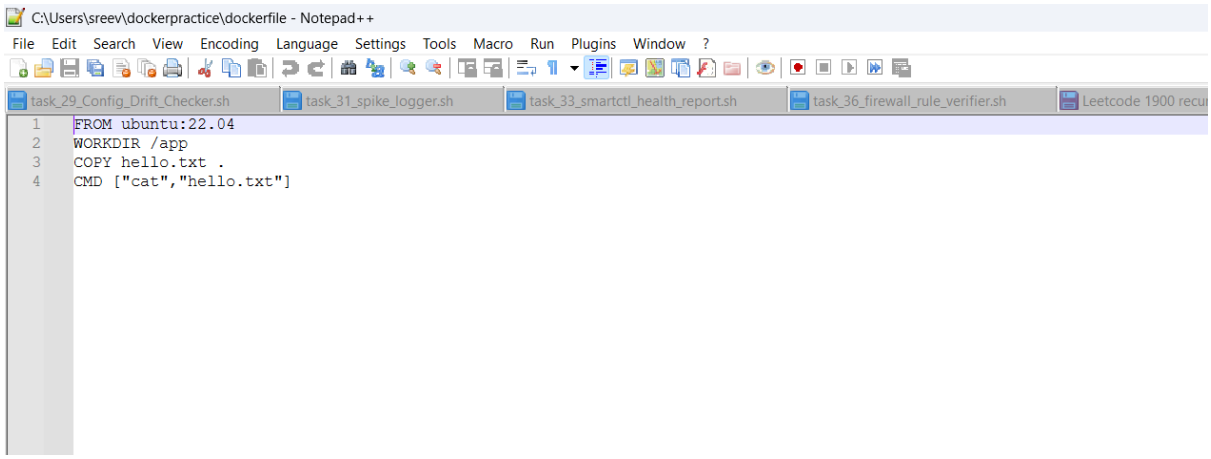
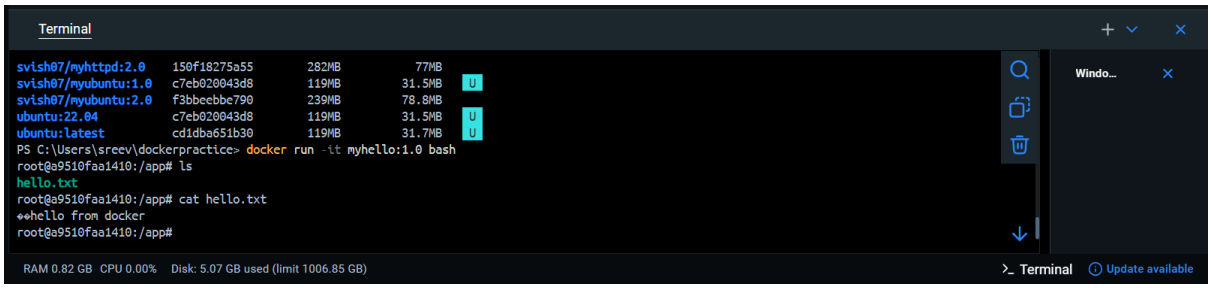
Continuation on my progress on docker:

Dockerfile practicals:







What is a Dockerfile?

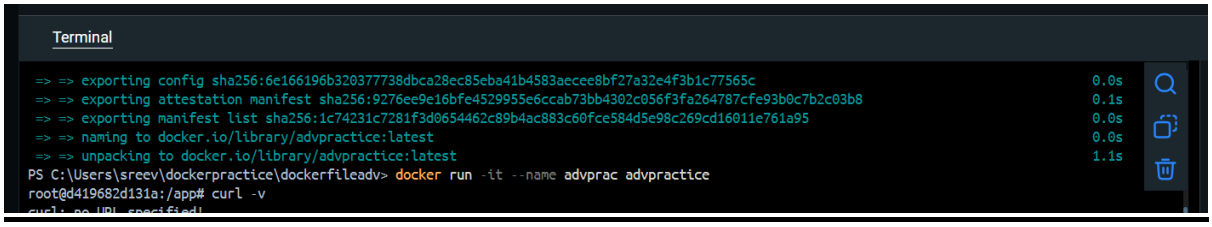
A Dockerfile is a blueprint that defines how to build a Docker image by specifying:

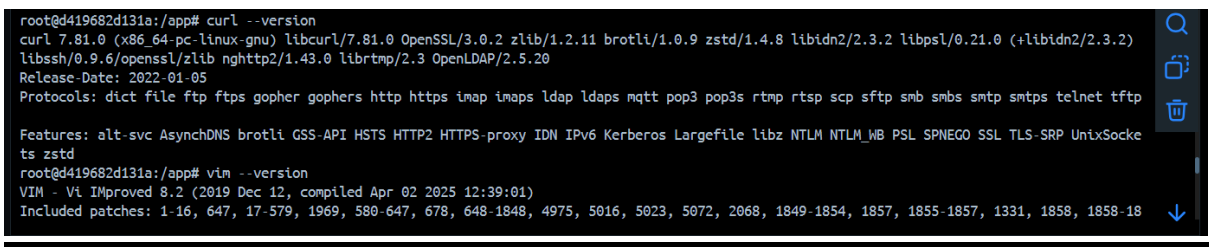
- The base image (FROM)
- The working directory (WORKDIR)
- Files to copy (COPY)
- Commands to run
- The default startup command (CMD)

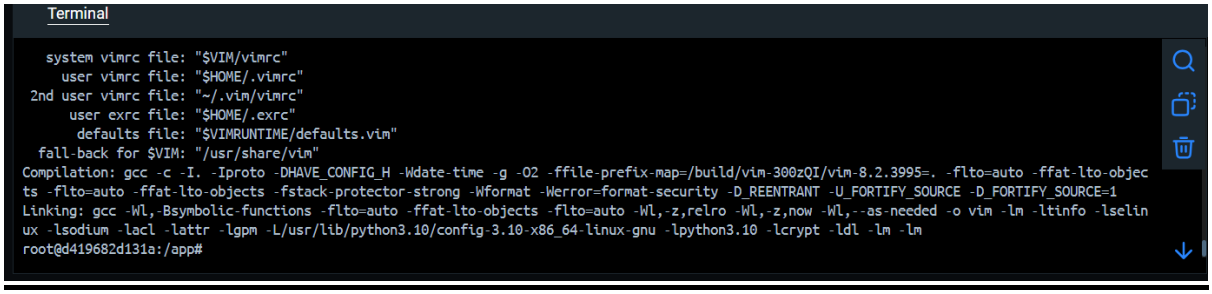
Docker reads it step-by-step and creates layered images.

In this practical, a Dockerfile was created to define how a custom image should be built using Ubuntu as the base image, setting /app as the working directory, copying a file from the build context into the image, and defining a default startup command. Using docker build -t myhello:1.0 ., Docker read the Dockerfile, processed each instruction as a layer, and created a tagged image. Running the image with docker run created a container that executed the defined CMD and displayed the output.

Advanced practical on Dockerfile:



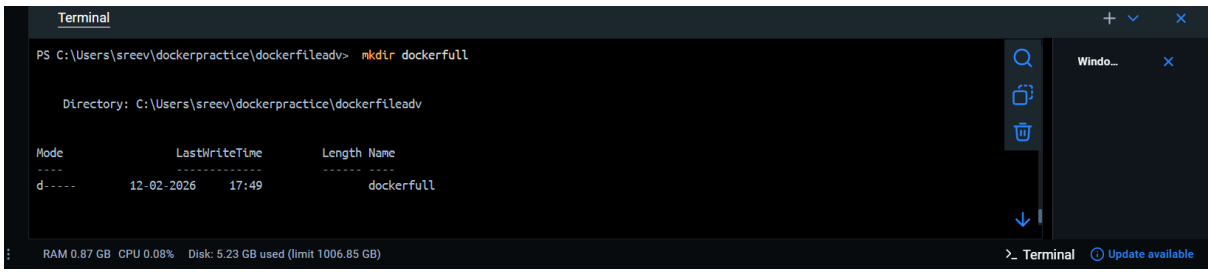




Summary:

In this practical, I went to create a custom Docker image using Ubuntu as the base, defined a working directory with WORKDIR /app, and installed system tools (curl and vim) using a carefully structured RUN instruction that chained apt update, installation, and cleanup commands with && to ensure controlled execution and reduced image size. By combining installation and cache removal (apt clean and rm -rf /var/lib/apt/lists/*) in a single layer, we followed an optimized and elegant image-building approach. The image was built using docker build -t mytools:1.0 ., tagged meaningfully, and then run interactively to verify that the installed tools were correctly available inside the container. This practical demonstrated layered image construction, efficient package management, and runtime validation in a clean, reproducible Docker workflow.

Docker Image Build Pipeline and Runtime Override Practical:



```
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> rm .dockerignore.txt
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> ren dockerfile.txt dockerfile
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> ls


    Directory: C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
```

Terminal

```
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> type dockerfile
FROM ubuntu:22.04
ENV APP_ENV=production
WORKDIR /app
RUN apt update && apt install -y curl && apt clean && rm -rf /var/lib/apt/lists/*
COPY index.html .
EXPOSE 8080
VOLUME ["/app/data"]
ENTRYPOINT ["echo"]
CMD ["Container started in production mode"]
```

RAM 0.86 GB CPU 0.17% Disk: 5.24 GB used (limit 1006.85 GB) >_ Terminal Update available

Terminal

```
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> docker build -t fullapp:1.0 .
[+] Building 43.8s (9/9) FINISHED                                  docker:desktop-linux
 => [internal] load build definition from dockerfile                               0.1s
 => => transferring dockerfile: 300B                                               0.0s
 => [internal] load metadata for docker.io/library/ubuntu:22.04                    0.3s
 => [internal] load .dockerignore                                                  0.1s
 => => transferring context: 69B                                                   0.0s
 => [1/4] FROM docker.io/library/ubuntu:22.04@sha256:c7eb020043d8fc2ae0793fb35a37b  0.1s
 => => resolve docker.io/library/ubuntu:22.04@sha256:c7eb020043d8fc2ae0793fb35a37b  0.1s
 => [internal] load build context                                                  0.1s
 => => transferring context: 101B                                                  0.0s
```

RAM 0.86 GB CPU 0.08% Disk: 5.24 GB used (limit 1006.85 GB) >_ Terminal Update available

Terminal

```
 => [4/4] COPY index.html .                                                        0.1s
 => exporting to image                                                             1.1s
 => => exporting layers                                                            0.5s
 => => exporting manifest sha256:2f8aa854c07c59c335e1dd307b70abc5057f8f1fe7ddda255  0.0s
 => => exporting config sha256:8b0e2d2ddcee72af979dd01d4e7fb62e9802d1704b8d96df1db  0.0s
 => => exporting attestation manifest sha256:c1b1fa4cab1accbe2d4f9d040ff464a062315  0.1s
 => => exporting manifest list sha256:c5d981fffebb5bad1b1ff1a7170659af2e1a6e1d1dc  0.0s
 => => naming to docker.io/library/fullapp:1.0                                     0.0s
 => => unpacking to docker.io/library/fullapp:1.0                                  0.3s
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> docker run -it --name myfullapp fullapp:1.0
Container started in production mode
```

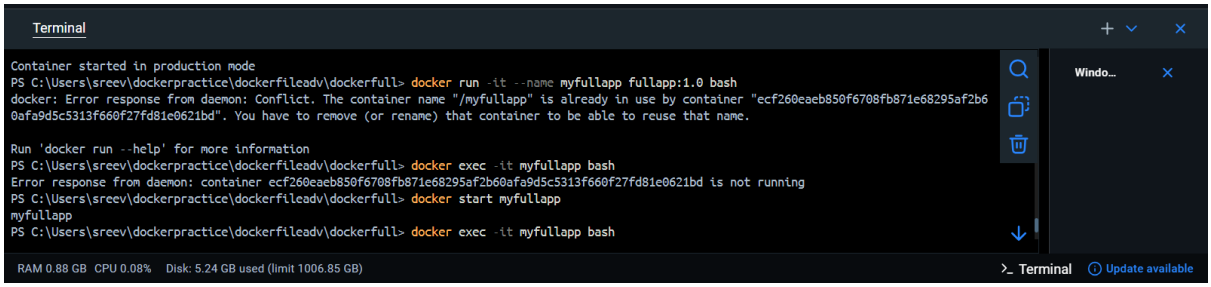
RAM 0.86 GB CPU 0.00% Disk: 5.24 GB used (limit 1006.85 GB) >_ Terminal Update available

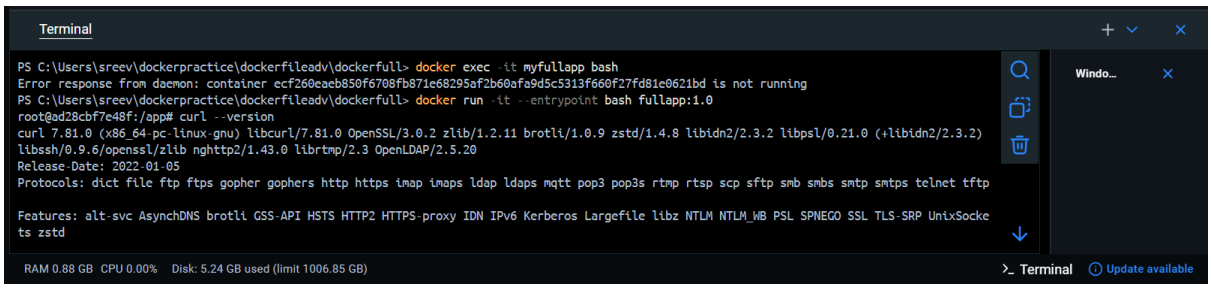
```
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> docker run -it --name myfullapp fullapp:1.0
Container started in production mode
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> docker run -it --name myfullapp fullapp:1.0 bash
docker: Error response from daemon: Conflict. The container name "/myfullapp" is already in use by container "ecf260eaeb850f6708fb871e68295af2b6
0afa9d5c5313f660f27fd81e0621bd". You have to remove (or rename) that container to be able to reuse that name.

Run 'docker run --help' for more information
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull>
```

RAM 0.86 GB CPU 0.42% Disk: 5.24 GB used (limit 1006.85 GB) >_ Terminal Update available

```
Run 'docker run --help' for more information
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> docker exec -it myfullapp bash
Error response from daemon: container ecf260eaeb850f6708fb871e68295af2b60afa9d5c5313f660f27fd81e0621bd is not running
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> docker start myfullapp
myfullapp
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull> docker exec -it myfullapp bash
Error response from daemon: container ecf260eaeb850f6708fb871e68295af2b60afa9d5c5313f660f27fd81e0621bd is not running
PS C:\Users\sreev\dockerpractice\dockerfileadv\dockerfull>
```

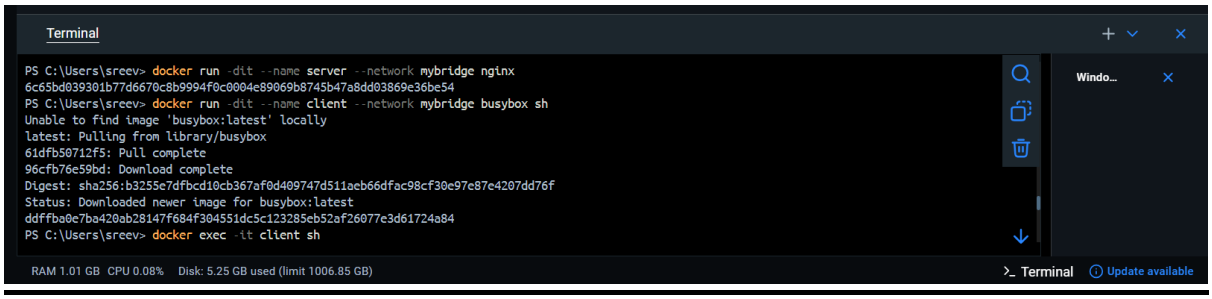
RAM 0.86 GB CPU 0.00% Disk: 5.24 GB used (limit 1006.85 GB) >_ Terminal Update available

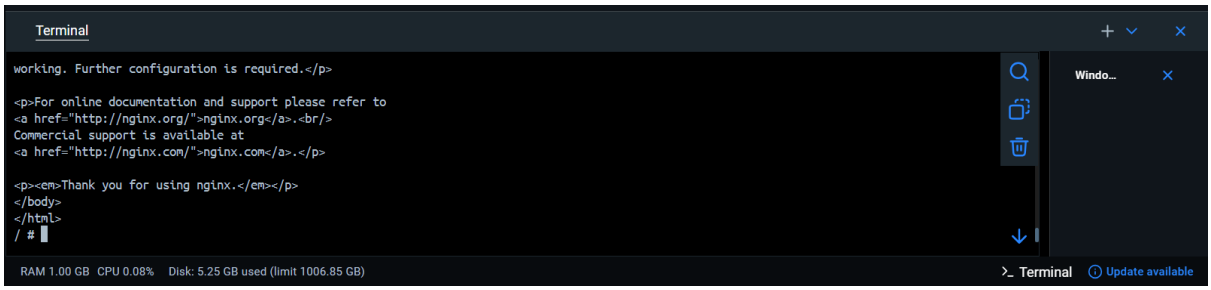




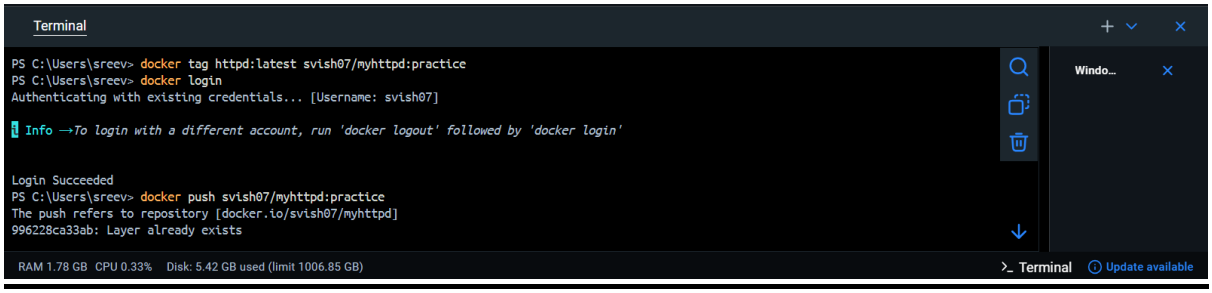
In this practical, I moved beyond simply building an image and instead explored the full choreography of Docker's build-time and run-time mechanics: starting with a controlled build context using .dockerignore to consciously shape what the daemon receives, then designing a layered Dockerfile that defined a base system (FROM), structured execution space (WORKDIR), controlled installation and cleanup through chained RUN commands, and intentional metadata through ENV, EXPOSE, and VOLUME. We examined how ENTRYPOINT establishes a container's core executable while CMD supplies its default arguments, discovering through experimentation that a container lives only as long as its primary process runs. When the container exited instantly, we learned that behaviour is not failure but design and that runtime overrides like --entrypoint and docker exec allow inspection and flexibility without rebuilding images. Altogether, the exercise revealed Docker as a disciplined separation of immutable image architecture and mutable runtime behaviour, where context, layering, execution flow, and lifecycle awareness converge into elegant container engineering.

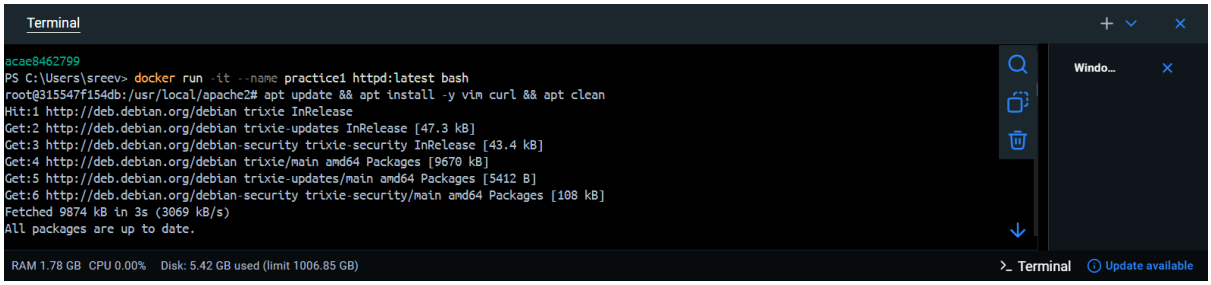
Advanced networking simulation practicals:

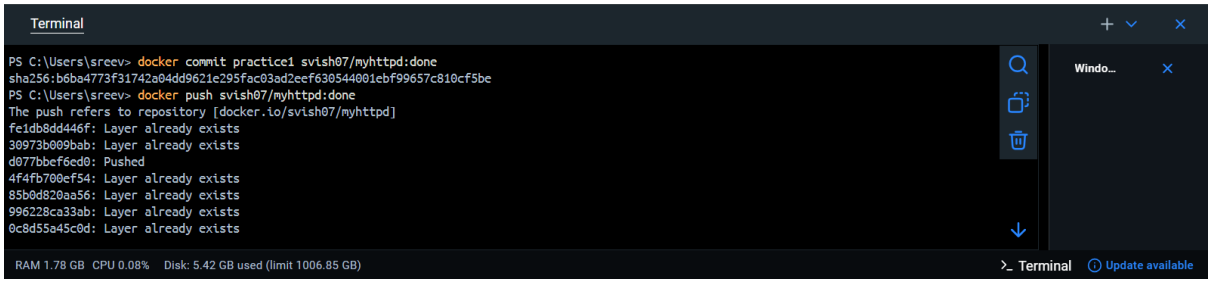




Practical: Customizing and Publishing an HTTPD Docker Image







In this practical, we transitioned from merely consuming an official httpd image to owning and evolving it, pulling the base image, re-tagging it under a personal namespace, authenticating with Docker Hub, and publishing it as versioned artifacts. By overriding the default runtime to access the container shell, we introduced additional tooling (vim, curl), then captured those changes through docker commit, effectively converting a transient container state into a new immutable image layer. Finally, by pushing the updated image to Docker Hub and exposing it via port mapping, we demonstrated the full lifecycle: base image acquisition, customization, versioning, distribution, and runtime validation, showcasing Docker not just as a container runtime, but as a disciplined system for image ownership, reproducibility, and controlled evolution.