SECURE SYSTEMS ENGINEERING

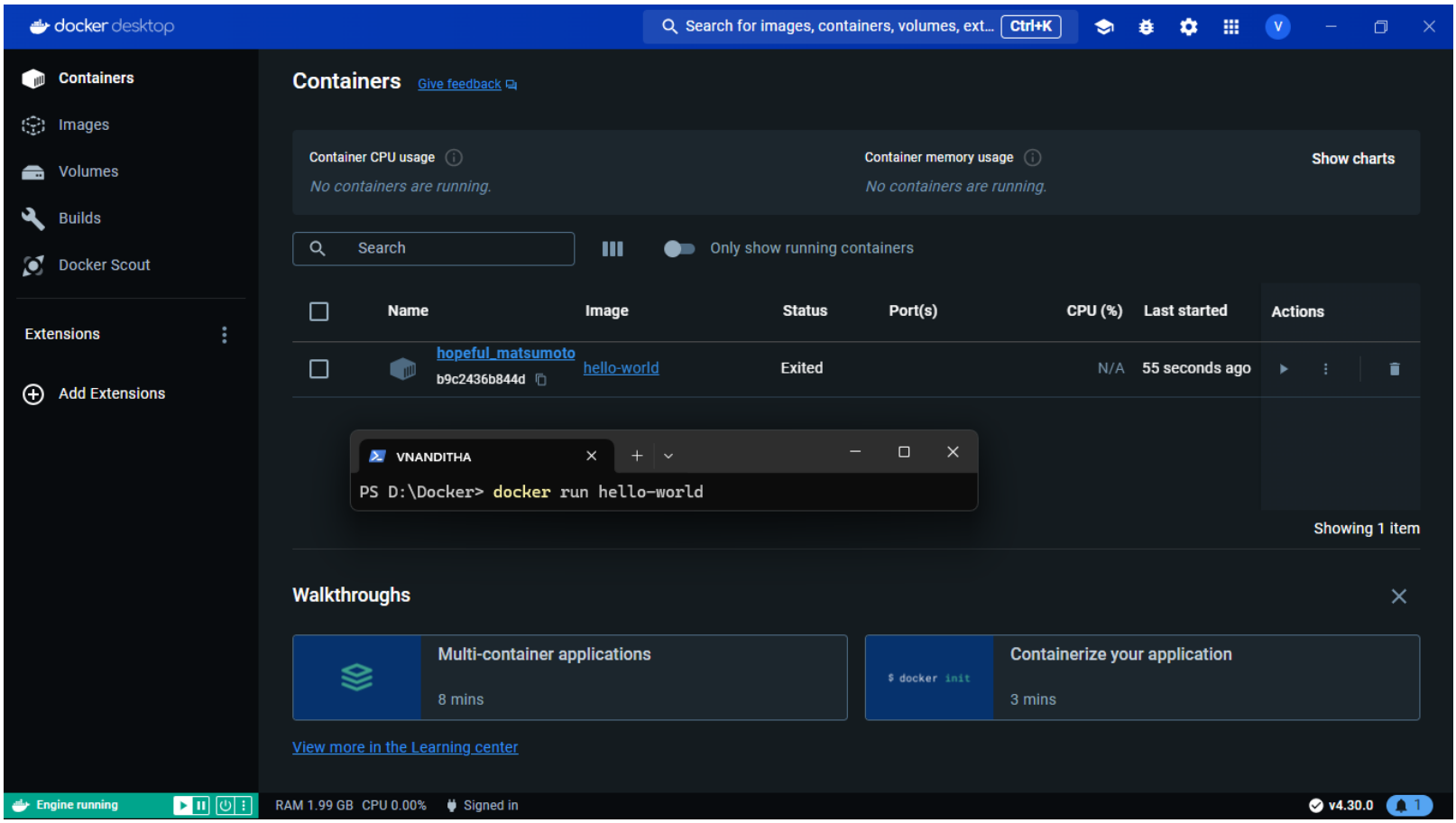
Assignment – 3

(DOCKER & KUBERNETES)

V NANDITHA
CB.SC.P2CYS23018
(2nd Year / 3rd Sem- MTech, CYBERSECURITY)

1) Create and use a Docker container interactively

- Open Docker Desktop, Terminal where the Folder is Stored.
- RUN : `docker run hello-world`

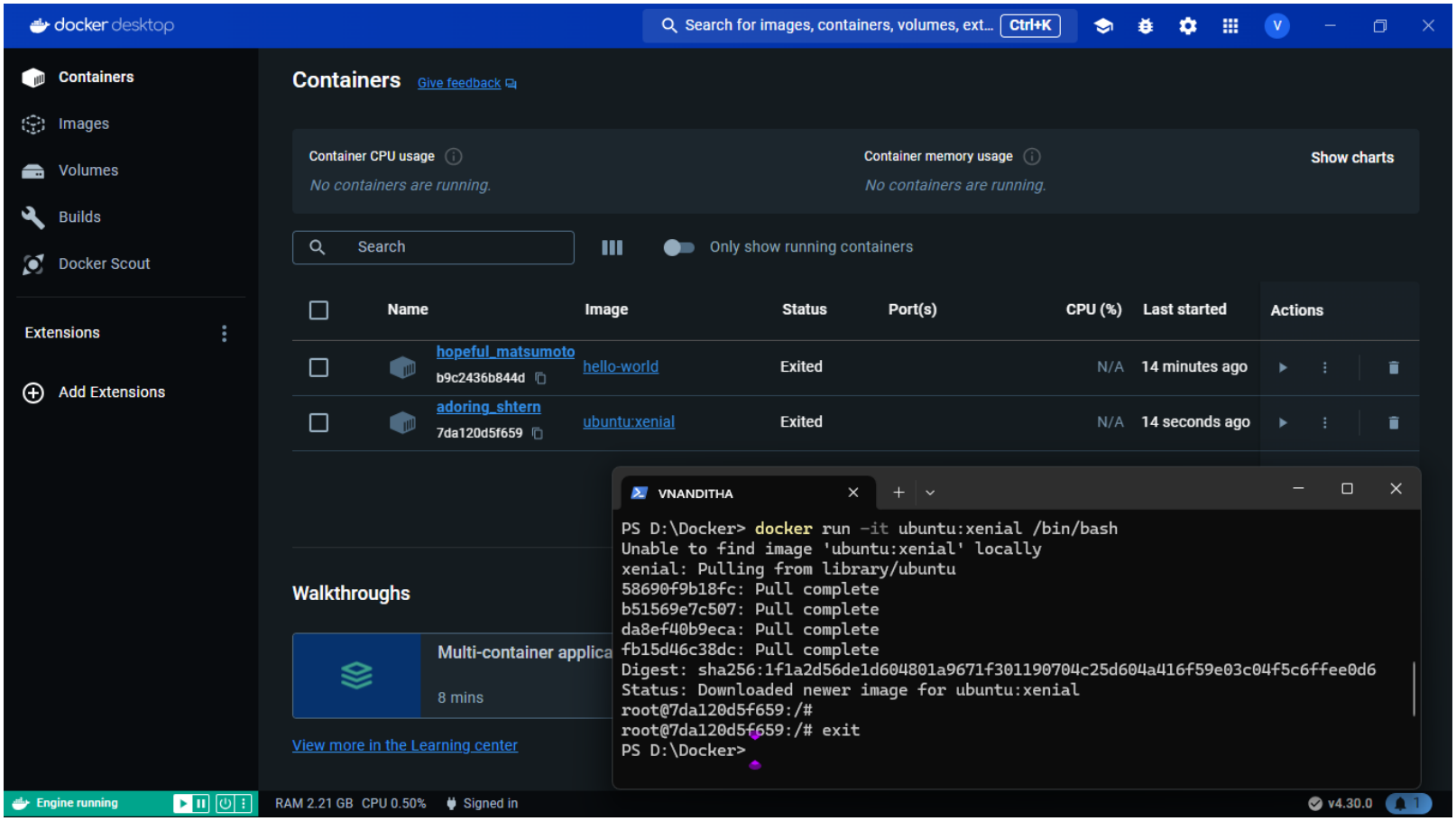



```
VNANDITHA
PS D:\Docker> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:266b191e926f65542fa8daaec01a192c4d292bff79426f47300a046e1bc576fd
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

PS D:\Docker>
```

Output When Typed: `docker run hello-world`

SUCCESSFULLY CREATED A DOCKER CONTAINER

- RUN : docker run -it ubuntu:xenial /bin/bash

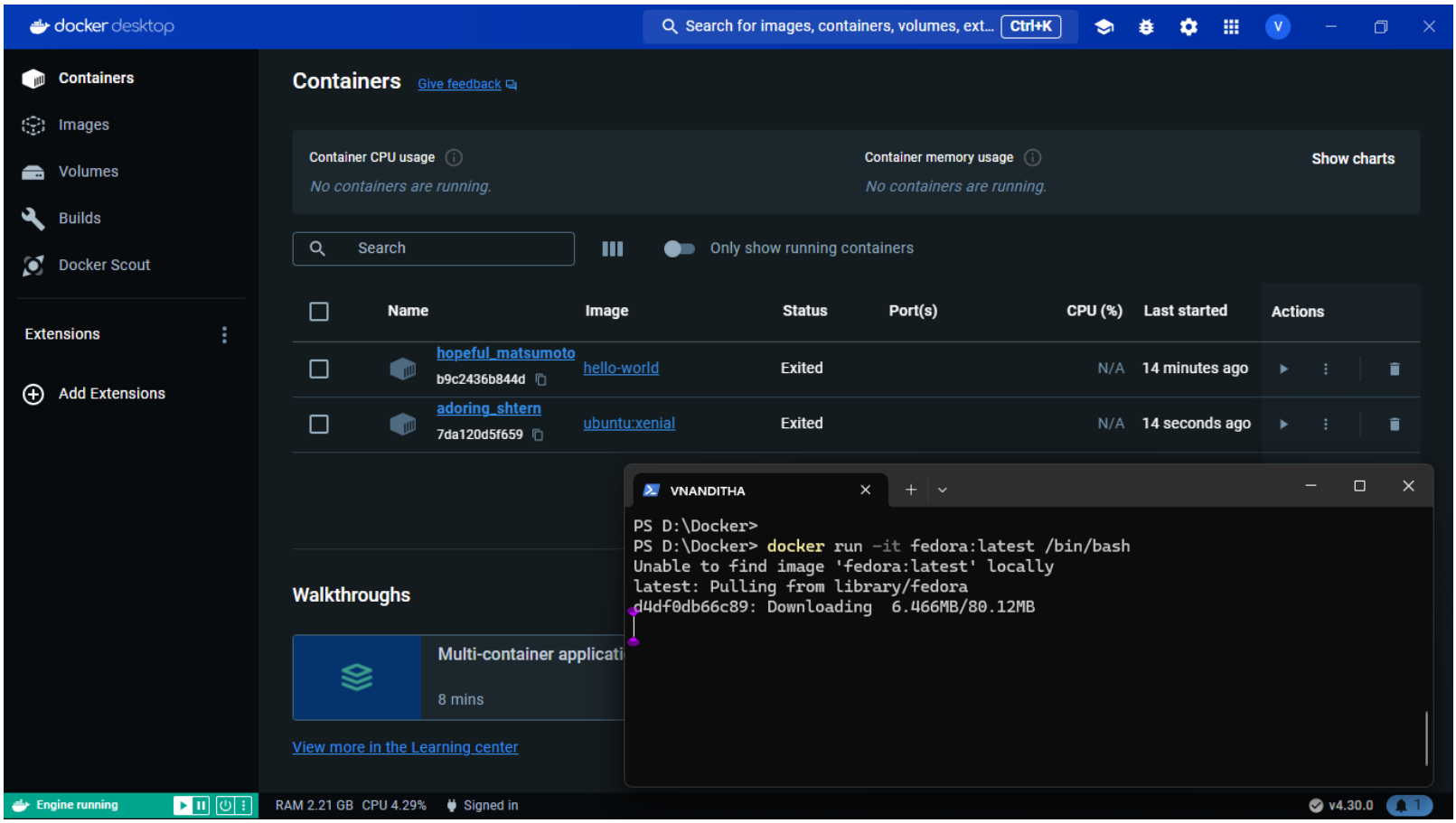


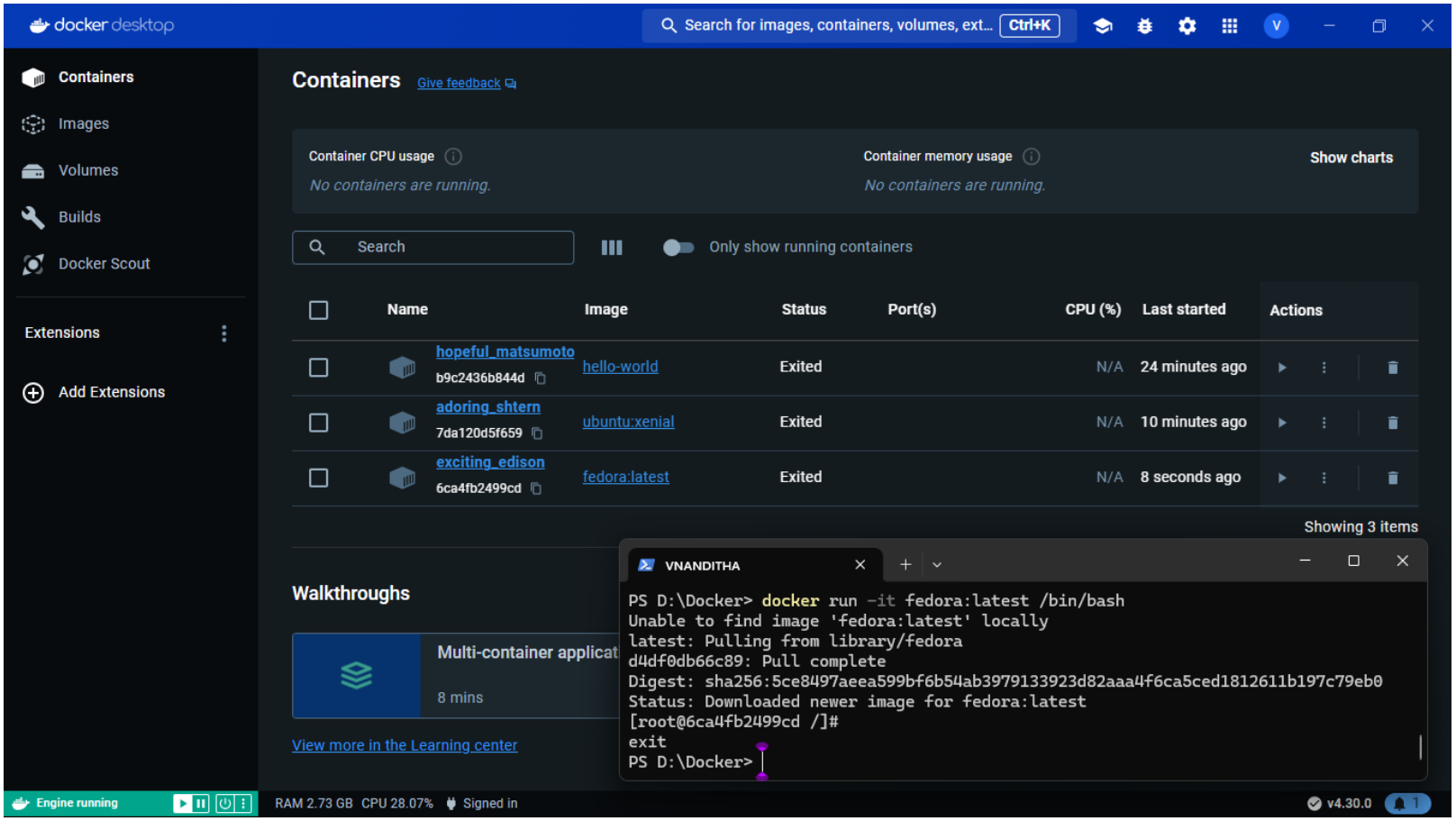
The -i flag tells docker to keep STDIN open to your container, and the -t flag allocates a pseudo-TTY for you. Basically, you need both for you to have a way to enter text and have this display properly. At the end of the command, /bin/bash is just the command you want to run once the container starts up.

Notice how even though your VM is running Debian, you were able to run the Xenial version of Ubuntu in a container.

This all works because Linux distributions all share the Linux kernel. For that same reason, you won't be able to run MacOS or Windows in a container.

We can Observe that the Containers are being Interactive.

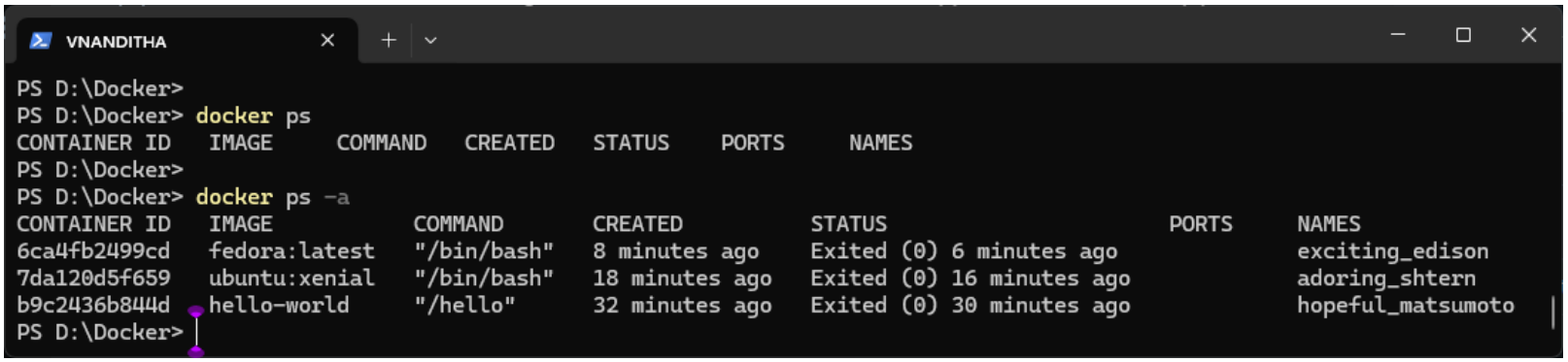




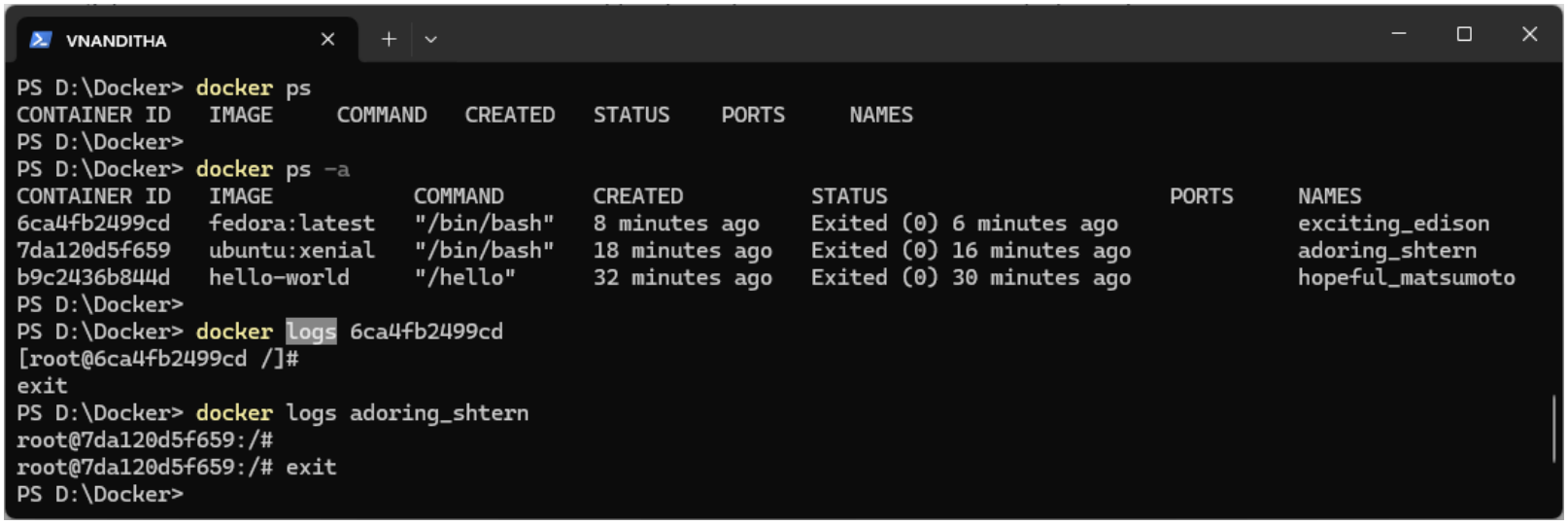
SUCCESSFULLY RAN AN INTERACTIVE DOCKER CONTAINER

2) Create a Docker file, which allows you to declaratively define your containers

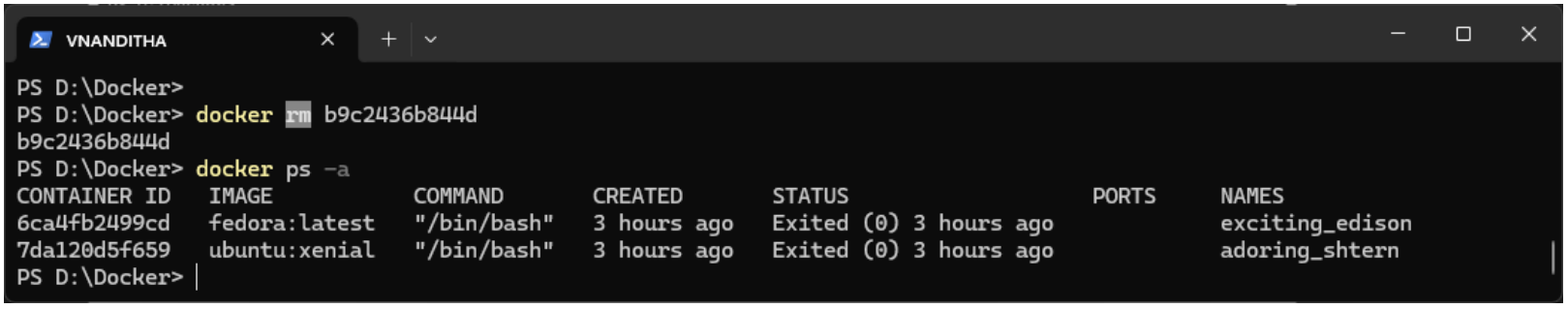
- To see the running containers on a system : `docker ps` `docker ps -a`



- To see the logs : `docker logs <name or id>`

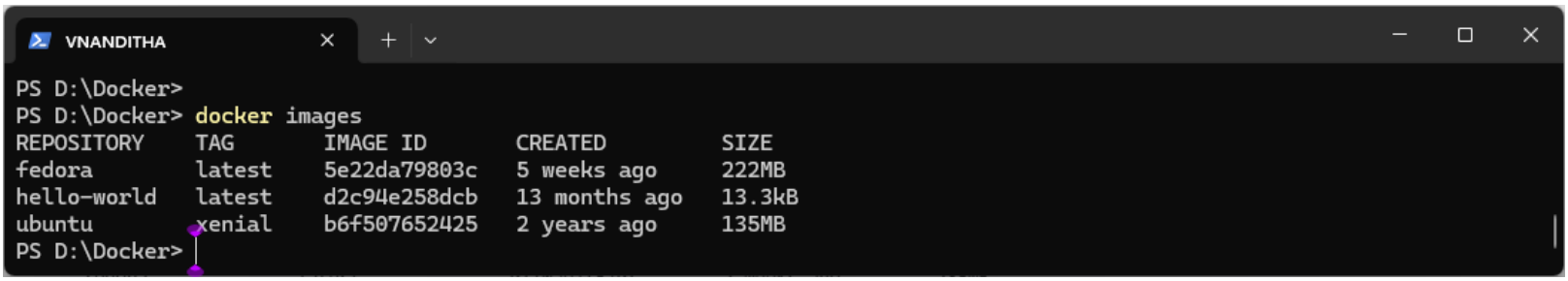


- To remove the Container : `docker rm <name or id>`



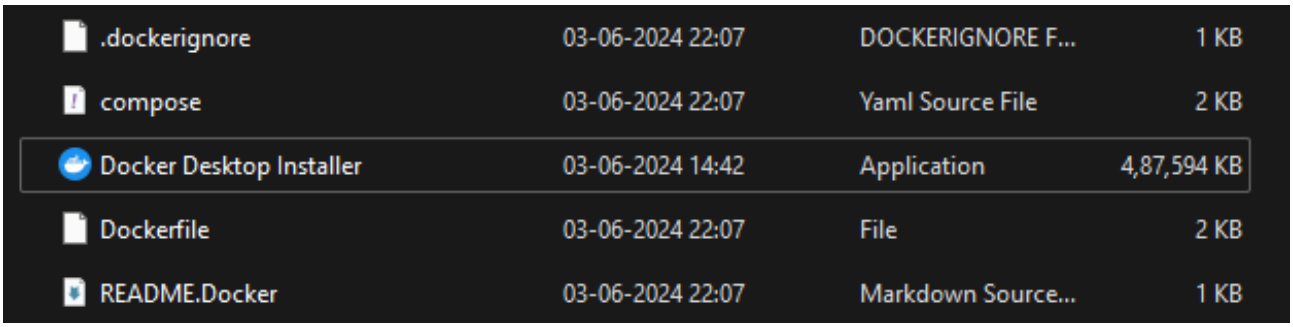
- It even gets removed from the Docker Desktop Application

- To get Docker Images : `docker images`



- To remove Docker Image : `docker rmi <image_id>`

- Initializing Docker

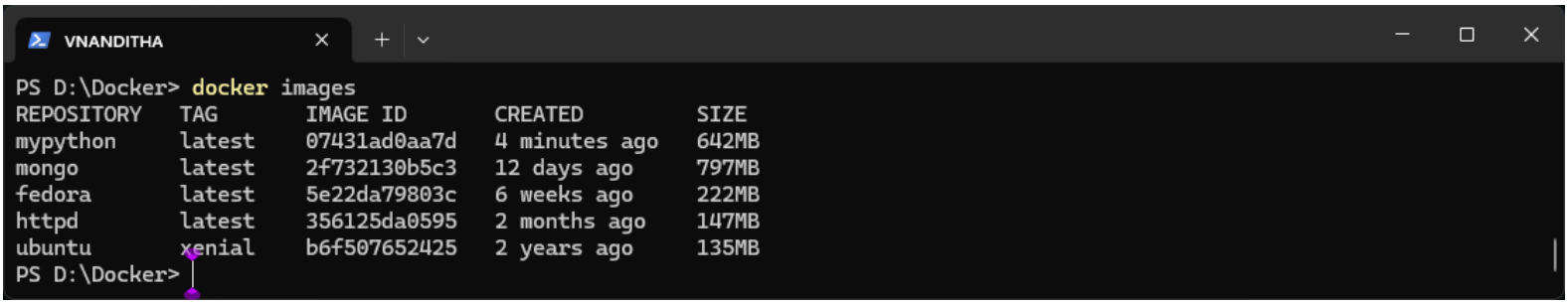


- These files get created in the Dockers Directory

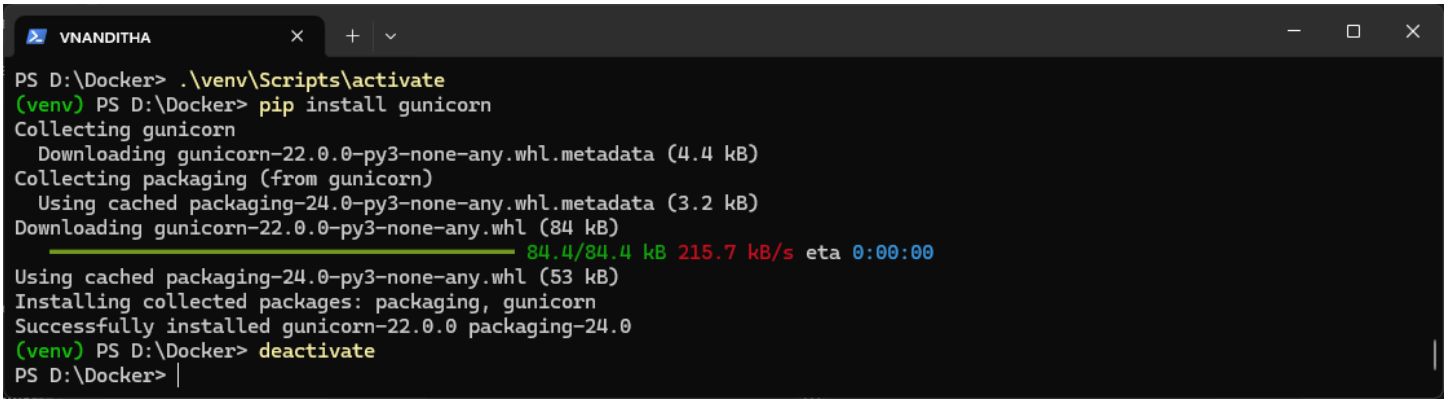
```
VNANDITHA
PS D:\Docker> docker init

Welcome to the Docker Init CLI!

This utility will walk you through creating the following files with sensible defaults for your project:
  - .dockerignore
  - Dockerfile
  - compose.yaml
  - README.Docker.md

Let's get started!

? What application platform does your project use? Python
? What version of Python do you want to use? (3.11.9)  3.11.9

? What version of Python do you want to use?  3.11.9
? What port do you want your app to listen on? (8000) 8000

? What port do you want your app to listen on? 8000
? What is the command you use to run your app (e.g., gunicorn 'myapp.example:app' --bind=0.0.0.0:8000)?

X Sorry, your reply was invalid: Value is required
? What is the command you use to run your app (e.g., gunicorn 'myapp.example:app' --bind=0.0.0.0:8000)? gunicorn 'myapp.example:app' --bind=0.0.0.0:
8000

✓ Created → .dockerignore
✓ Created → Dockerfile
✓ Created → compose.yaml
✓ Created → README.Docker.md

→ Your Docker files are ready!
  Review your Docker files and tailor them to your application.
  Consult README.Docker.md for information about using the generated files.

! Warning → No requirements.txt file found. Create one with the dependencies for your application, including an entry for the gunicorn package, befo
re running it.

What's next?
  Start your application by running → docker compose up --build
  Your application will be available at http://localhost:8000
PS D:\Docker>
```

- Building the file : `docker build -t mypython:latest .`

```
VNANDITHA

PS D:\Docker> python -m venv venv
PS D:\Docker> .\venv\Scripts\activate
(venv) PS D:\Docker> pip freeze > requirements.txt
(venv) PS D:\Docker>
(venv) PS D:\Docker>
(venv) PS D:\Docker>
(venv) PS D:\Docker>
(venv) PS D:\Docker> ^C
(venv) PS D:\Docker>
(venv) PS D:\Docker> deactivate
PS D:\Docker> docker build -t mypython:latest .
[+] Building 30.6s (14/14) FINISHED                                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                                            0.2s
 => => transferring dockerfile: 1.69kB                                                                                          0.1s
 => resolve image config for docker-image://docker.io/docker/dockerfile:1                                                       2.4s
 => [auth] docker/dockerfile:pull token for registry-1.docker.io                                                                0.0s
 => CACHED docker-image://docker.io/docker/dockerfile:1@sha256:a57df69d0ea827fb7266491f2813635de6f17269be881f696fbfdf2d83dda33e  0.0s
 => [internal] load metadata for docker.io/library/python:3.11.9-slim                                                           2.1s
 => [auth] library/python:pull token for registry-1.docker.io                                                                   0.0s
 => [internal] load .dockerignore                                                                                               0.0s
 => => transferring context: 671B                                                                                               0.0s
 => [base 1/5] FROM docker.io/library/python:3.11.9-slim@sha256:fc39d2e68b554c3f0a5cb8a776280c0b3d73b4c04b83dbade835e2a171ca27ef  0.0s
 => [internal] load build context                                                                                               8.8s
 => => transferring context: 12.03MB                                                                                            8.8s
 => CACHED [base 2/5] WORKDIR /app                                                                                              0.0s
 => CACHED [base 3/5] RUN adduser     --disabled-password     --gecos ""     --home "/nonexistent"     --shell "/sbin/nologin"     --no-creat  0.0s
 => [base 4/5] RUN --mount=type=cache,target=/root/.cache/pip     --mount=type=bind,source=requirements.txt,target=requirements.txt     pytho  6.5s
 => [base 5/5] COPY . .                                                                                                         5.0s
 => exporting to image                                                                                                          5.0s
 => => exporting layers                                                                                                         4.9s
 => => writing image sha256:07431ad0aa7d7d376591633668254629 7c4f889ff4335c83d89cb51423a0488e                                   0.0s
 => => naming to docker.io/library/mypython:latest                                                                              0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS D:\Docker>
```

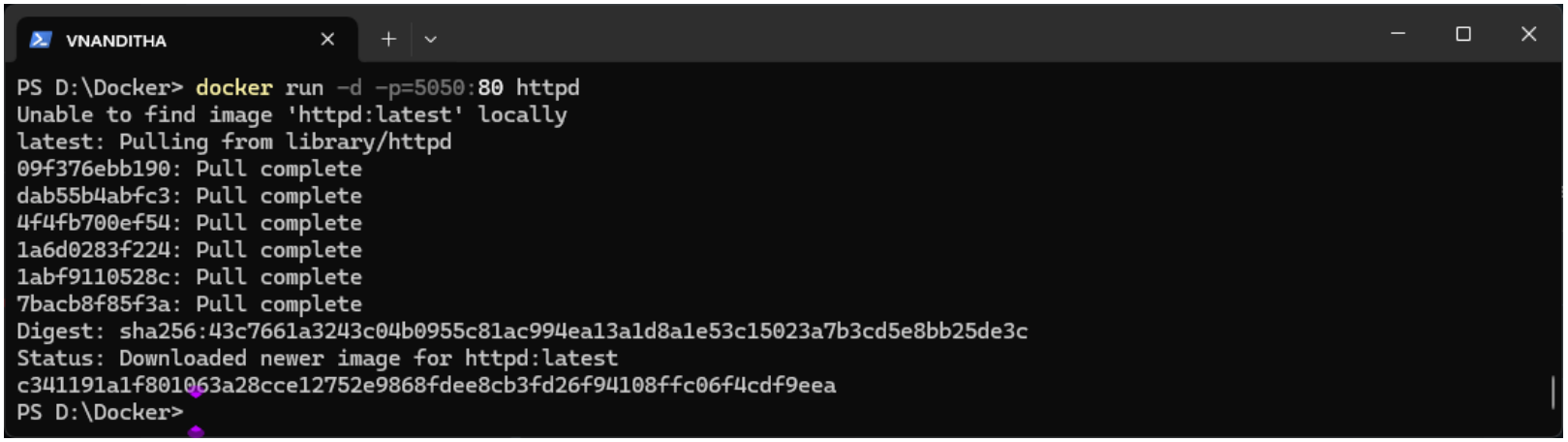


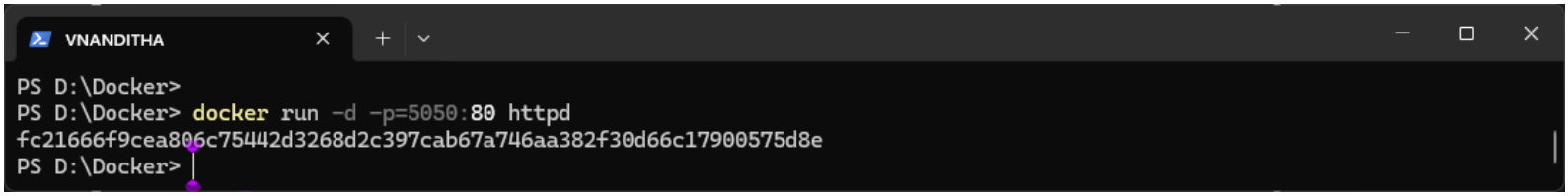
Interactively run the Docker Image and execute Python commands.

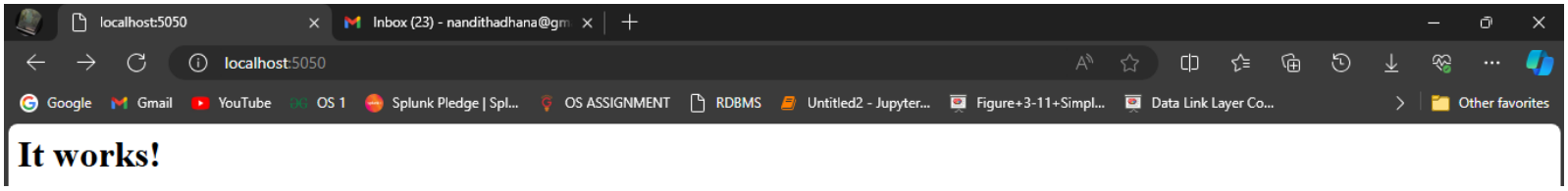


3) Run detached containers and understand port forwarding

- Docker creates a separate virtual network for containers, so you will need to do forward your host port to your container's port (this is called Port Forwarding, or port mapping). The container is listening on port 80, so let's try to forward our host machine's port 5050 to the container's port 80 when we run the container : `docker run -d -p=5050:80 httpd`

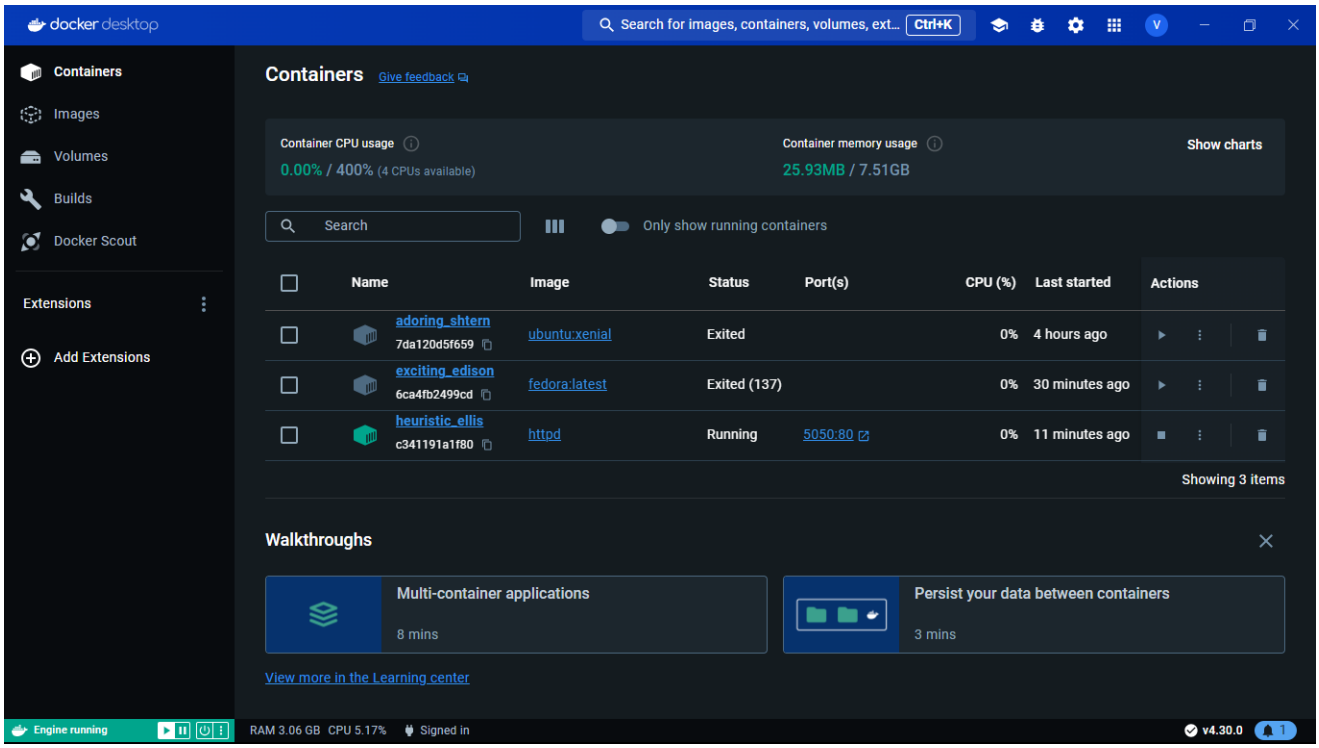






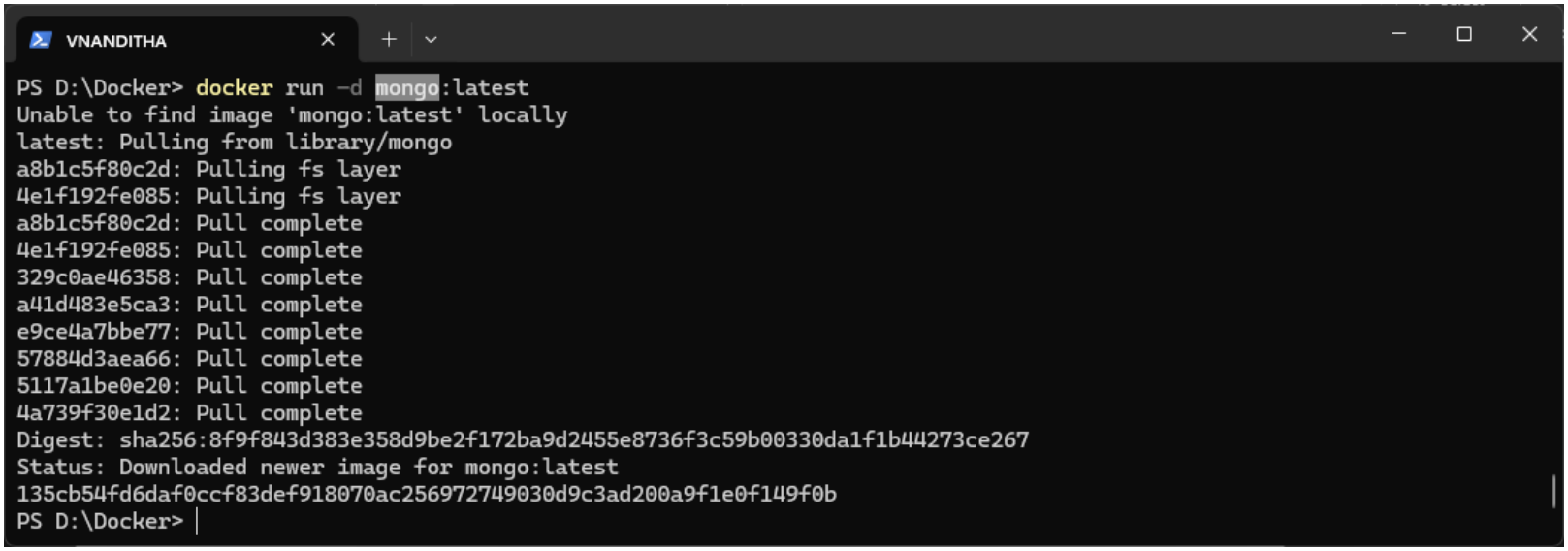
- To Stop a Running Process of Container : `docker stop <id>`

- For Restarting : `docker restart <container_id_or_name>`

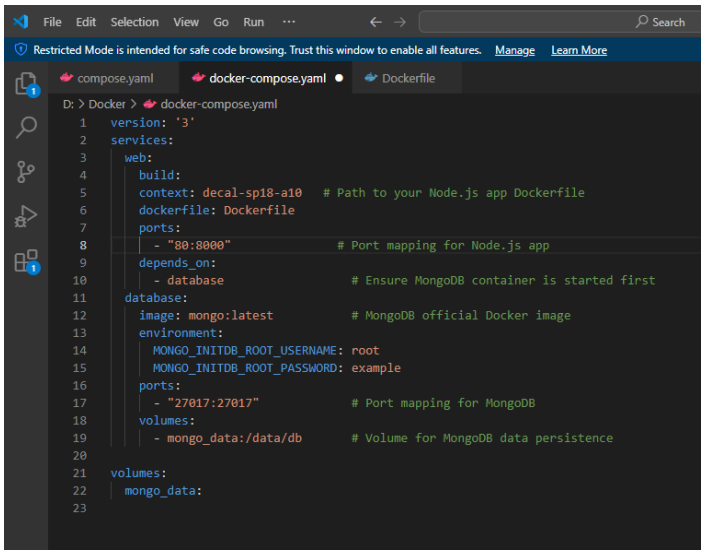


4) Use docker-compose to run a multi-container web application

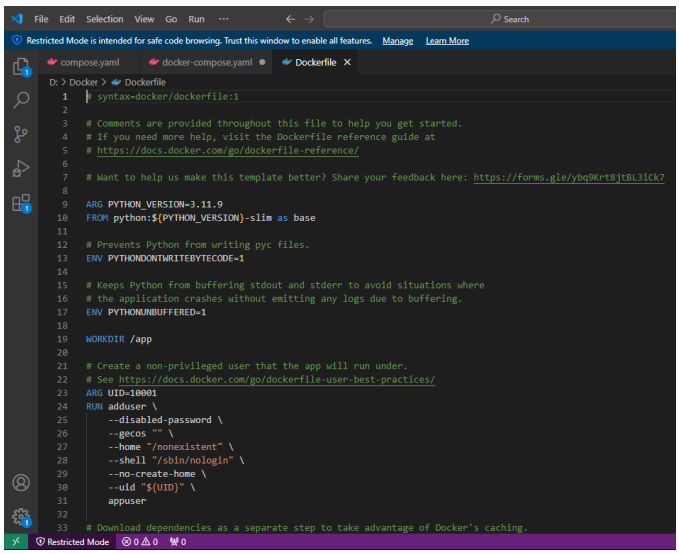
- Run MongoDB within a container in detached mode :
  `docker run -d mongo:latest`



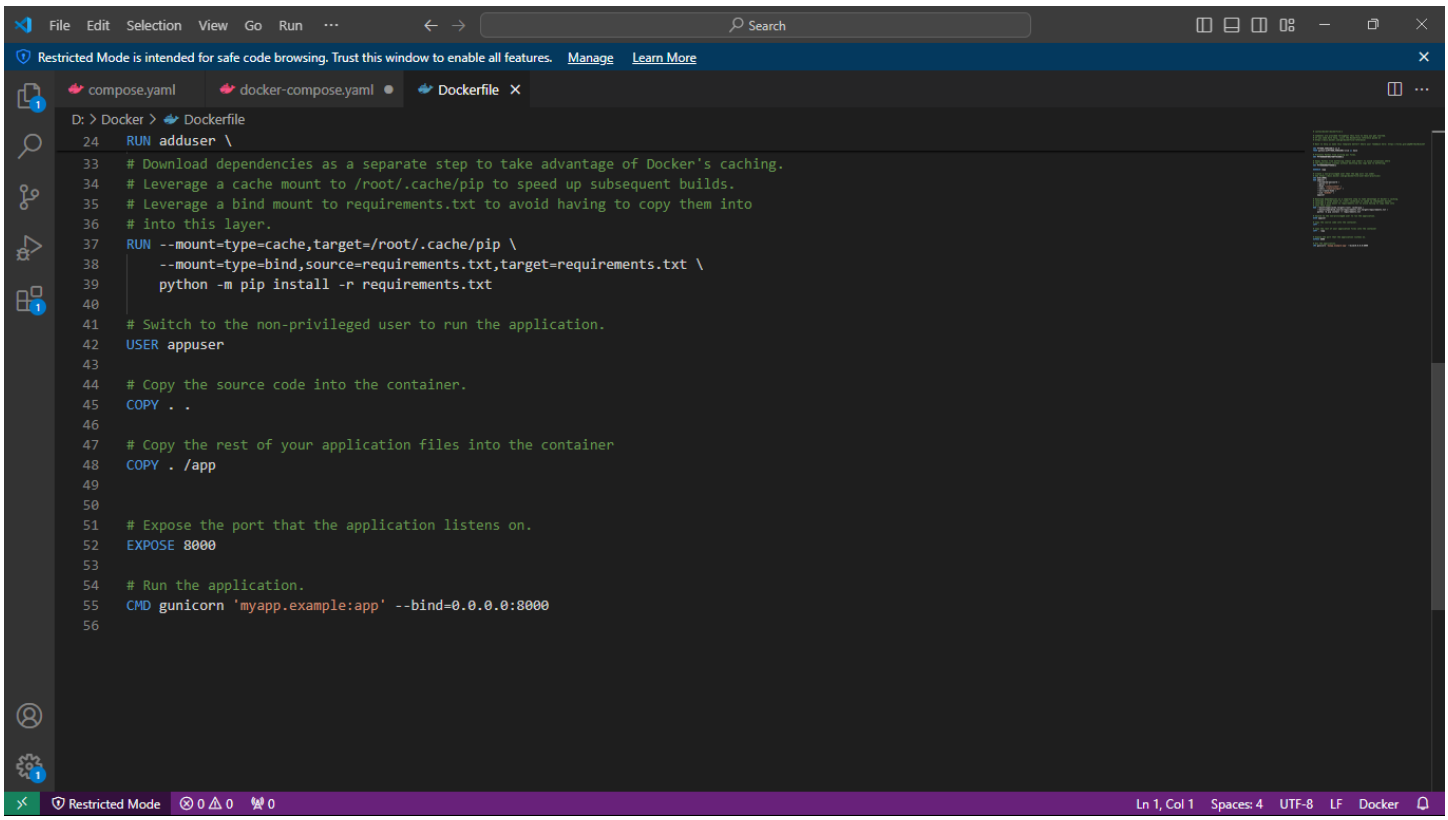
The NodeJS project is downloaded from the provided location by typing the below commands : git clone https://github.com/0xcf/decal-sp18-a10.git



```yaml
version: '3'
services:
  web:
    build:
      context: decal-sp18-a10   # Path to your Node.js app Dockerfile
      dockerfile: Dockerfile
      ports:
        - "80:8000"             # Port mapping for Node.js app
      depends_on:
        - database              # Ensure MongoDB container is started first
  database:
    image: mongo:latest         # MongoDB official Docker image
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
    ports:
      - "27017:27017"           # Port mapping for MongoDB
    volumes:
      - mongo_data:/data/db     # Volume for MongoDB data persistence

volumes:
  mongo_data:
```

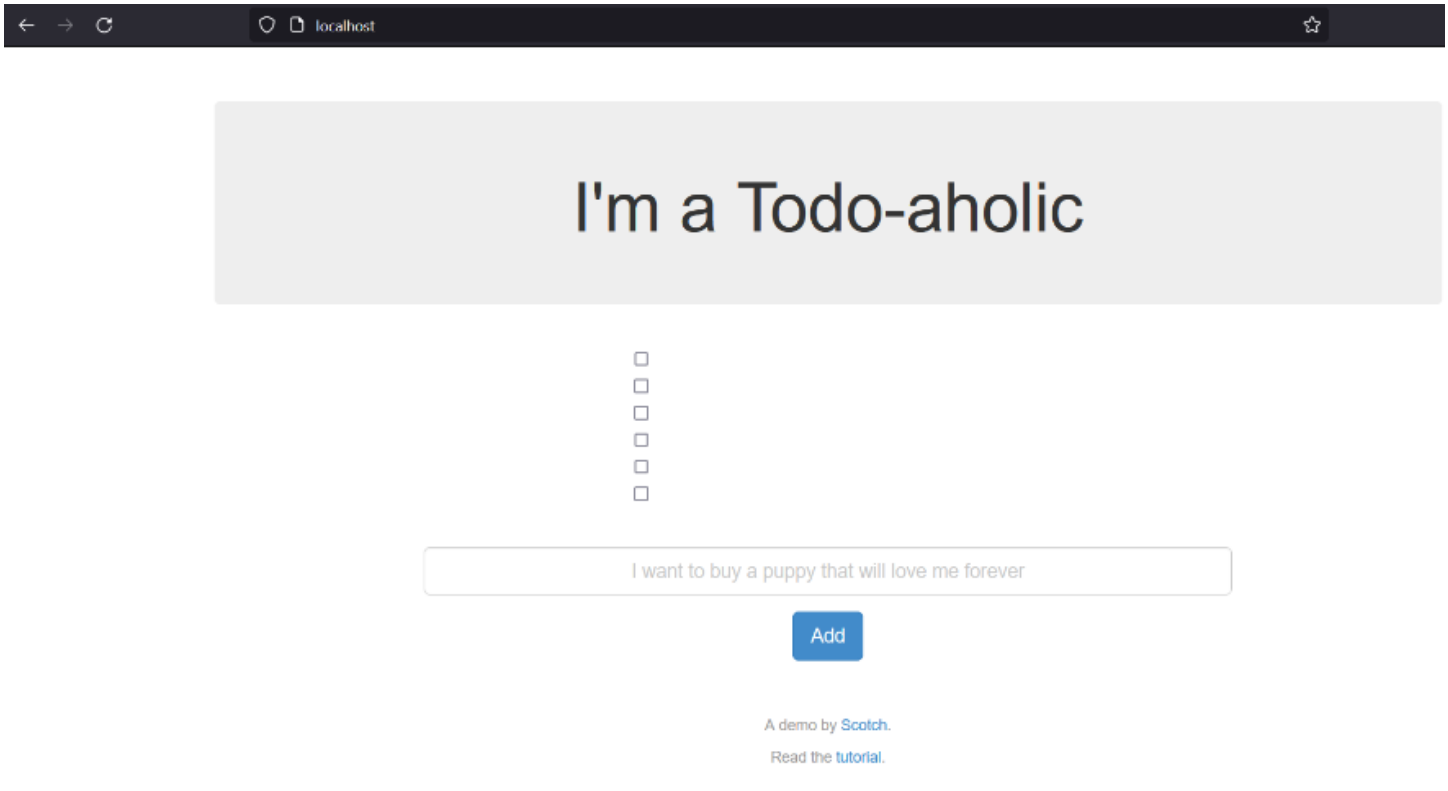


```
# syntax=docker/dockerfile:1

# Comments are provided throughout this file to help you get started.
# If you need more help, visit the Dockerfile reference guide at
# https://docs.docker.com/go/dockerfile-reference/

# Want to help us make this template better? Share your feedback here: https://forms.gle/ybq9Krt8jtBL3iCk7

ARG PYTHON_VERSION=3.11.9
FROM python:${PYTHON_VERSION}-slim as base

# Prevents Python from writing pyc files.
ENV PYTHONDONTWRITEBYTECODE=1

# Keeps Python from buffering stdout and stderr to avoid situations where
# the application crashes without emitting any logs due to buffering.
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# Create a non-privileged user that the app will run under.
# See https://docs.docker.com/go/dockerfile-user-best-practices/
ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    appuser

# Download dependencies as a separate step to take advantage of Docker's caching.
```



```
RUN adduser \
# Download dependencies as a separate step to take advantage of Docker's caching.
# Leverage a cache mount to /root/.cache/pip to speed up subsequent builds.
# Leverage a bind mount to requirements.txt to avoid having to copy them into
# into this layer.
RUN --mount=type=cache,target=/root/.cache/pip \
    --mount=type=bind,source=requirements.txt,target=requirements.txt \
    python -m pip install -r requirements.txt

# Switch to the non-privileged user to run the application.
USER appuser

# Copy the source code into the container.
COPY . .

# Copy the rest of your application files into the container
COPY . /app


# Expose the port that the application listens on.
EXPOSE 8000

# Run the application.
CMD gunicorn 'myapp.example:app' --bind=0.0.0.0:8000
```

```
VNANDITHA
PS D:\Docker> docker-compose up
time="2024-06-05T14:44:18+05:30" level=warning msg="Found multiple config files with supported names: D:\\Docker\\compose.yaml, D:\\Docker\\docker-compose.yaml"
time="2024-06-05T14:44:18+05:30" level=warning msg="Using D:\\Docker\\compose.yaml"
time="2024-06-05T14:44:18+05:30" level=warning msg="Found multiple config files with supported names: D:\\Docker\\compose.yaml, D:\\Docker\\docker-compose.yaml"
time="2024-06-05T14:44:18+05:30" level=warning msg="Using D:\\Docker\\compose.yaml"
[+] Building 32.9s (15/15) FINISHED                                                                                    docker:default
 => [server internal] load build definition from Dockerfile                                                                   0.1s
 => => transferring dockerfile: 1.76kB                                                                                        0.1s
 => [server] resolve image config for docker-image://docker.io/docker/dockerfile:1                                            4.6s
 => [server auth] docker/dockerfile:pull token for registry-1.docker.io                                                       0.0s
 => CACHED [server] docker-image://docker.io/docker/dockerfile:1@sha256:a57df69d0ea827fb7266491f2813635de6f17269be881f696fbfdf2d83dda33e      0.0s
 => [server internal] load metadata for docker.io/library/python:3.11.9-slim                                                  3.2s
 => [server auth] library/python:pull token for registry-1.docker.io                                                          0.0s
 => [server internal] load .dockerignore                                                                                      0.5s
 => => transferring context: 671B                                                                                             0.2s
 => [server base 1/6] FROM docker.io/library/python:3.11.9-slim@sha256:fc39d2e68b554c3f0a5cb8a776280c0b3d73b4c04b83dbade835e2a171ca27ef      0.0s
 => [server internal] load build context                                                                                      1.2s
 => => transferring context: 84.16kB                                                                                          1.0s
 => CACHED [server base 2/6] WORKDIR /app                                                                                     0.0s
 => CACHED [server base 3/6] RUN adduser     --disabled-password     --gecos ""     --home "/nonexistent"     --shell "/sbin/nologin"     --n  0.0s
 => CACHED [server base 4/6] RUN --mount=type=cache,target=/root/.cache/pip     --mount=type=bind,source=requirements.txt,target=requirements  0.0s
 => [server base 5/6] COPY . .                                                                                                4.7s
 => [server base 6/6] COPY . /app                                                                                             4.7s
 => [server] exporting to image                                                                                              12.5s
 => => exporting layers                                                                                                      12.3s
 => => writing image sha256:0fae46e907212cc010fb3a2edca9b0c09dcbdb0078ab733ddb8e901afacd23cf                                  0.0s
 => => naming to docker.io/library/docker-server                                                                              0.0s
[+] Running 2/2
 ✔Network docker_default     Created                                                                                          0.2s
 ✔Container docker-server-1  Created                                                                                          0.3s
Attaching to server-1
server-1  | [2024-06-05 09:14:58 +0000] [7] [INFO] Starting gunicorn 22.0.0
server-1  | [2024-06-05 09:14:58 +0000] [7] [INFO] Listening at: http://0.0.0.0:8000 (7)
server-1  | [2024-06-05 09:14:58 +0000] [7] [INFO] Using worker: sync
server-1  | [2024-06-05 09:14:58 +0000] [8] [INFO] Booting worker with pid: 8
```



5) Task 1: Run some simple Docker containers

My ROOT-ID : 192.168.0.18

Time: Approximately 30 minutes

Tasks:

- Task 0: Prerequisites
- Task 1: Run some simple Docker containers
- Task 2: Package and run a custom app using Docker
- Task 3: Modify a Running Website

Task 0: Prerequisites

You will need all of the following to complete this lab:

- A clone of the lab's GitHub repo.
- A DockerID.

Clone the Lab's GitHub Repo

Use the following command to clone the lab's repo from GitHub (you can click the command or manually type it). This will make a copy of the lab's repo in a new sub-directory called `linux_tweet_app`.

```
git clone https://github.com/dockersamples/linux_tweet_app
```

Make sure you have a DockerID

If you do not have a DockerID (a free login used to access Docker Hub), please visit Docker Hub and register for one. You will need this for later steps.

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
#################################################################
#                          WARNING!!!!                          #
# This is a sandbox environment. Using personal credentials    #
# is HIGHLY! discouraged. Any consequences of doing so are     #
# completely the user's responsibilites.                       #
#                                                              #
# The PWD team.                                                #
#################################################################
[node1] (local) root@192.168.0.18 ~
$ git clone https://github.com/dockersamples/linux_tweet_app
Cloning into 'linux_tweet_app'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 14 (delta 4), reused 4 (delta 4), pack-reused 6
Receiving objects: 100% (14/14), 10.79 KiB | 5.39 MiB/s, done.
Resolving deltas: 100% (5/5), done.
[node1] (local) root@192.168.0.18 ~
$
```

Task 1: Run some simple Docker containers

There are different ways to use containers. These include:

1. **To run a single task:** This could be a shell script or a custom app.
2. **Interactively:** This connects you to the container similar to the way you SSH into a remote server.
3. **In the background:** For long-running services like websites and databases.

In this section you'll try each of those options and see how Docker manages the workload.

Run a single task in an Alpine Linux container

In this step we're going to start a new container and tell it to run the `hostname` command. The container will start, execute the `hostname` command, then exit.

1. Run the following command in your Linux console.

   ```
   docker container run alpine hostname
   ```

   The output below shows that the `alpine:latest` image could not be found locally. When this happens, Docker automatically *pull*s it from Docker Hub.

   After the image is pulled, the container's hostname is displayed (`888e89a3b36b` in the example below).

   ```
   Unable to find image 'alpine:latest' locally
   latest: Pulling from library/alpine
   88286f41530e: Pull complete
   Digest: sha256:f006ecbb824d87947d0b51ab8488634bf69fe4094959d935c0c103
   f4820a417d
   Status: Downloaded newer image for alpine:latest
   888e89a3b36b
   ```

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$  docker container run alpine hostname
Unable to find image 'alpine:latest' locally
 docker container run alpine hostname
latest: Pulling from library/alpine
d25f557d7f31: Pull complete
Digest: sha256:77726ef6b57ddf65bb551896826ec38bc3e53f75cdde31354fbffb4f25238ebd
Status: Downloaded newer image for alpine:latest
92ebf3e67ec2
[node1] (local) root@192.168.0.18 ~
$  docker container run alpine hostname
9bf316d9fd45
[node1] (local) root@192.168.0.18 ~
$
```

2. Docker keeps a container running as long as the process it started inside the container is still running. In this case the `hostname` process exits as soon as the output is written. This means the container stops. However, Docker doesn't delete resources by default, so the container still exists in the `Exited` state.

   List all containers.

   ```
   docker container ls --all
   ```

   Notice that your Alpine Linux container is in the `Exited` state.

   ```
    CONTAINER ID        IMAGE               COMMAND             CREATE
   D              STATUS              PORTS               NAMES
    888e89a3b36b        alpine              "hostname"          50 sec
   onds ago       Exited (0) 49 seconds ago                        aweso
   me_elion
   ```

   > **Note:** The container ID *is* the hostname that the container displayed. In the example above it's `888e89a3b36b`.

Containers which do one task and then exit can be very useful. You could build a Docker image that executes a script to configure something. Anyone can execute that task just by running the container - they don't need the actual scripts or configuration information.

Run an interactive Ubuntu container

You can run a container based on a different version of Linux than is running on your Docker host.

In the next example, we are going to run an Ubuntu Linux container on top of an Alpine Linux Docker host (Play With Docker uses Alpine Linux for its nodes).

```
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$
[node1] (local) root@192.168.0.18 ~
$  docker container run alpine hostname
Unable to find image 'alpine:latest' locally
 docker container run alpine hostname
latest: Pulling from library/alpine
d25f557d7f31: Pull complete
Digest: sha256:77726ef6b57ddf65bb551896826ec38bc3e53f75cdde31354fbffb4f25238ebd
Status: Downloaded newer image for alpine:latest
92ebf3e67ec2
[node1] (local) root@192.168.0.18 ~
$  docker container run alpine hostname
9bf316d9fd45
[node1] (local) root@192.168.0.18 ~
$  docker container ls --all
CONTAINER ID   IMAGE     COMMAND      CREATED              STATUS
            PORTS     NAMES
9bf316d9fd45   alpine    "hostname"   About a minute ago   Exited (0) About a m
inute ago             eloquent_gagarin
92ebf3e67ec2   alpine    "hostname"   About a minute ago   Exited (0) About a m
inute ago             reverent_cray
[node1] (local) root@192.168.0.18 ~
$
```

1. Run a Docker container and access its shell.

   ```
   docker container run --interactive --tty --rm ubuntu bash
   ```

   In this example, we're giving Docker three parameters:

   - `--interactive` says you want an interactive session.
   - `--tty` allocates a pseudo-tty.
   - `--rm` tells Docker to go ahead and remove the container when it's done executing.

   The first two parameters allow you to interact with the Docker container.

   We're also telling the container to run `bash` as its main process (PID 1).

   When the container starts you'll drop into the bash shell with the default prompt `root@<container id>:/#`. Docker has attached to the shell in the container, relaying input and output between your local session and the shell session in the container.

2. Run the following commands in the container.

   `ls /` will list the contents of the root directory in the container, `ps aux` will show running processes in the container, `cat /etc/issue` will show which Linux distro the container is running, in this case Ubuntu 20.04.3 LTS.

   ```
   ls /
   ```

   ```
   ps aux
   ```

   ```
   cat /etc/issue
   ```

3. Type `exit` to leave the shell session. This will terminate the `bash` process, causing the container to exit.

```
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
92ebf3e67ec2
[node1] (local) root@192.168.0.18 ~
$  docker container run alpine hostname
9bf316d9fd45
[node1] (local) root@192.168.0.18 ~
$  docker container ls --all
CONTAINER ID   IMAGE     COMMAND      CREATED              STATUS
            PORTS     NAMES
9bf316d9fd45   alpine    "hostname"   About a minute ago   Exited (0) About a m
inute ago             eloquent_gagarin
92ebf3e67ec2   alpine    "hostname"   About a minute ago   Exited (0) About a m
inute ago             reverent_cray
[node1] (local) root@192.168.0.18 ~
$ docker container run --interactive --tty --rm ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
00d679a470c4: Pull complete
Digest: sha256:e3f92abc0967a6c19d0dfa2d55838833e947b9d74edbcb0113e48535ad4be12a
Status: Downloaded newer image for ubuntu:latest
root@c9339599d435:/# ls/
bash: ls/: No such file or directory
root@c9339599d435:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.4  0.0   4580  4128 pts/0    Ss   09:46   0:00 bash
root           9  0.0  0.0   7880  4116 pts/0    R+   09:47   0:00 ps aux
root@c9339599d435:/# cat /etc/issue
Ubuntu 24.04 LTS \n \l

root@c9339599d435:/#
```

3. Type `exit` to leave the shell session. This will terminate the `bash` process, causing the container to exit.

```
exit
```

> **Note:** As we used the `--rm` flag when we started the container, Docker removed the container when it stopped. This means if you run another `docker container ls --all` you won't see the Ubuntu container.

4. For fun, let's check the version of our host VM.

```
cat /etc/issue
```

You should see:

```
Welcome to Alpine Linux 3.8
Kernel \r on an \m (\l)
```

Notice that our host VM is running Alpine Linux, yet we were able to run an Ubuntu container. As previously mentioned, the distribution of Linux inside the container does not need to match the distribution of Linux running on the Docker host.

However, Linux containers require the Docker host to be running a Linux kernel. For example, Linux containers cannot run directly on Windows Docker hosts. The same is true of Windows containers - they need to run on a Docker host with a Windows kernel.

Interactive containers are useful when you are putting together your own image. You can run a container and verify all the steps you need to deploy your app, and capture them in a Dockerfile.

> You can commit a container to make an image from it, but you

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
9bf316d9fd45   alpine    "hostname"   About a minute ago   Exited (0) About a m
inute ago             eloquent_gagarin
92ebf3e67ec2   alpine    "hostname"   About a minute ago   Exited (0) About a m
inute ago             reverent_cray
[node1] (local) root@192.168.0.18 ~
$ docker container run --interactive --tty --rm ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
00d679a470c4: Pull complete
Digest: sha256:e3f92abc0967a6c19d0dfa2d55838833e947b9d74edbcb0113e48535ad4be12a
Status: Downloaded newer image for ubuntu:latest
root@c9339599d435:/# ls/
bash: ls/: No such file or directory
root@c9339599d435:/# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.4  0.0   4580  4128 pts/0    Ss   09:46   0:00 bash
root          9  0.0  0.0   7880  4116 pts/0    R+   09:47   0:00 ps aux
root@c9339599d435:/# cat /etc/issue
Ubuntu 24.04 LTS \n \l

root@c9339599d435:/# exit
exit
[node1] (local) root@192.168.0.18 ~
$ cat /etc/issue
Welcome to Alpine Linux 3.18
Kernel \r on an \m (\l)

[node1] (local) root@192.168.0.18 ~
$
```

Run a background MySQL container

Background containers are how you'll run most applications. Here's a simple example using MySQL.

1. Run a new MySQL container with the following command.

```
docker container run \
--detach \
--name mydb \
-e MYSQL_ROOT_PASSWORD=my-secret-pw \
mysql:latest
```

   - `--detach` will run the container in the background.
   - `--name` will name it **mydb**.
   - `-e` will use an environment variable to specify the root password (NOTE: This should never be done in production).

As the MySQL image was not available locally, Docker automatically pulled it from Docker Hub.

```
 Unable to find image 'mysql:latest' locallylatest: Pulling from li
brary/mysql
 aa18ad1a0d33: Pull complete
 fdb8d83dece3: Pull complete
 75b6ce7b50d3: Pull complete
 ed1d0a3a64e4: Pull complete
 8eb36a82c85b: Pull complete
 41be6f1a1c40: Pull complete
 0e1b414eac71: Pull complete
 914c28654a91: Pull complete
 587693eb988c: Pull complete
 b183c3585729: Pull complete
 315e21657aa4: Pull complete
 Digest: sha256:0dc3dacb751ef46a6647234abdec2d47400f0dfbe77ab490b02
```

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
[node1] (local) root@192.168.0.18 ~
$ docker container run \
> --detach \
> --name mydb \
> -e MYSQL_ROOT_PASSWORD=my-secret-pw \
>  docker container run \
>  --detach \
>  --name mydb \
>  -e MYSQL_ROOT_PASSWORD=my-secret-pw \
>  mysql:latest
Unable to find image 'docker:latest' locally
latest: Pulling from library/docker
d25f557d7f31: Already exists
709fdef6a978: Pull complete
4f4fb700ef54: Pull complete
aee0e04186d5: Pull complete
5040cb473dd8: Pull complete
033d787e83ee: Pull complete
ca0e4ad1d5fb: Pull complete
addd3fa802ed: Pull complete
fa82f68d1319: Pull complete
945935a84494: Pull complete
eb849552362f: Pull complete
0b8dd1c36edf: Pull complete
0c2644864cc4: Pull complete
934d8d2105be: Pull complete
51f324d3ecf3: Pull complete
a4382da9867c: Pull complete
```

```
a4382da9867c: Pull complete
Digest: sha256:76ba10a4aed708c7b2db09d45740d711edf707f7368f6808bd32a53eaed33404
Status: Downloaded newer image for docker:latest
6a65da86cc54e8d371845f02a05111ea72d2d61f67daf45e2fd7e190a7943d6f
[node1] (local) root@192.168.0.18 ~
$
```

6) Task 2 : Package & Run a custom app using Docker

Task 2: Package and run a custom app using Docker

In this step you'll learn how to package your own apps as Docker images using a Dockerfile.

The Dockerfile syntax is straightforward. In this task, we're going to create a simple NGINX website from a Dockerfile.

Build a simple website image

Let's have a look at the Dockerfile we'll be using, which builds a simple website that allows you to send a tweet.

1. Make sure you're in the `linux_tweet_app` directory.

```
cd ~/linux_tweet_app
```

2. Display the contents of the Dockerfile.

```
cat Dockerfile
```

```
FROM nginx:latest

COPY index.html /usr/share/nginx/html
COPY linux.png /usr/share/nginx/html

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

Let's see what each of these lines in the Dockerfile do.

- FROM specifies the base image to use as the starting point for this new image you're

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
$  cd ~/linux_tweet_app
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  cat Dockerfile
FROM nginx:latest

COPY index.html /usr/share/nginx/html
COPY linux.png /usr/share/nginx/html

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
```

3. In order to make the following commands more copy/paste friendly, export an environment variable containing your DockerID (if you don't have a DockerID you can get one for free via Docker Hub).

   You will have to manually type this command as it requires your unique DockerID.

   `export DOCKERID=<your docker id>`

4. Echo the value of the variable back to the terminal to ensure it was stored correctly.

```
echo $DOCKERID
```

5. Use the `docker image build` command to create a new Docker image using the instructions in the Dockerfile.

   - `--tag` allows us to give the image a custom name. In this case it's comprised of our DockerID, the application name, and a version. Having the Docker ID attached to the name will allow us to store it on Docker Hub in a later step
   - `.` tells Docker to use the current directory as the build context

   Be sure to include period (`.`) at the end of the command.

```
docker image build --tag $DOCKERID/linux_tweet_app:1.0 .
```

The output below shows the Docker daemon executing each line in the Dockerfile

```
Sending build context to Docker daemon  32.77kB
Step 1/5 : FROM nginx:latest
latest: Pulling from library/nginx
afeb2bfd31c0: Pull complete
7ff5d10493db: Pull complete
d2562f1ae1d0: Pull complete
Digest: sha256:af32e714a9cc3157157374e68c818b05ebe9e0737aac06b55a09da374209a8f9
Status: Downloaded newer image for nginx:latest
---> da5939581ac8
Step 2/5 : COPY index.html /usr/share/nginx/html
```

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  echo $DOCKERID

[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker image build --tag $DOCKERID/linux_tweet_app:1.0 .
[+] Building 0.0s (0/0)                                         docker:default
ERROR: invalid tag "/linux_tweet_app:1.0": invalid reference format
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  echo $DOCKERID

[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker image build --tag $DOCKERID/linux_tweet_app:1.0 .
[+] Building 0.0s (0/0)                                         docker:default
ERROR: invalid tag "/linux_tweet_app:1.0": invalid reference format
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker container run \
>  --detach \
>  --publish 80:80 \
>  --name linux_tweet_app \
>  $DOCKERID/linux_tweet_app:1.0
docker: invalid reference format.
See 'docker run --help'.
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker container rm --force linux_tweet_app
Error response from daemon: No such container: linux_tweet_app
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
```

6. Use the `docker container run` command to start a new container from the image you created.

   As this container will be running an NGINX web server, we'll use the `--publish` flag to publish port 80 inside the container onto port 80 on the host. This will allow traffic coming in to the Docker host on port 80 to be directed to port 80 in the container. The format of the `--publish` flag is `host_port : container_port` .

   ```
   docker container run \
   --detach \
   --publish 80:80 \
   --name linux_tweet_app \
   $DOCKERID/linux_tweet_app:1.0
   ```

   Any external traffic coming into the server on port 80 will now be directed into the container on port 80.

   In a later step you will see how to map traffic from two different ports - this is necessary when two containers use the same port to communicate since you can only expose the port once on the host.

7. Click here to load the website which should be running.

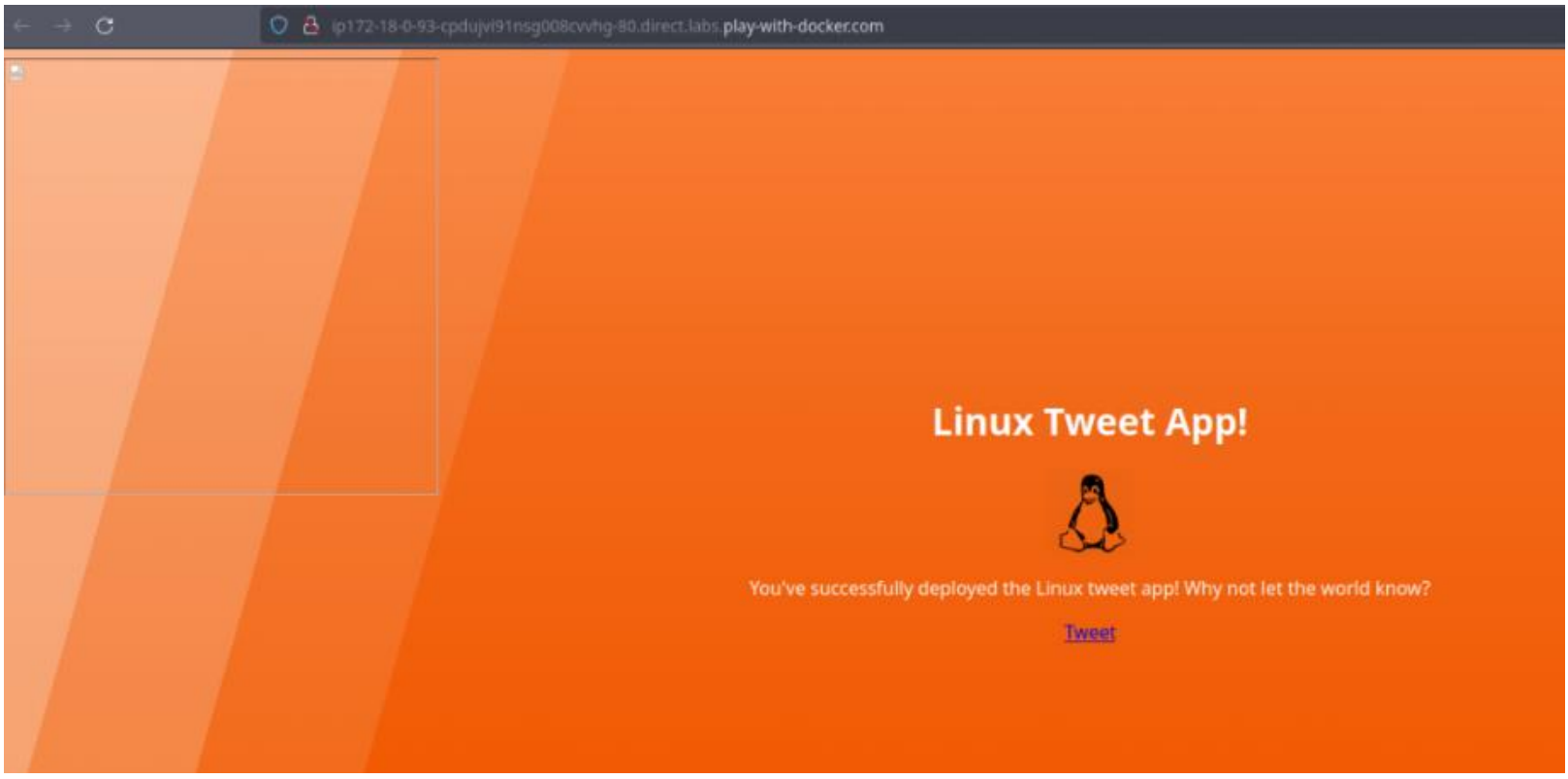
8. Once you've accessed your website, shut it down and remove it.

   ```
   docker container rm --force linux_tweet_app
   ```

   > **Note:** We used the `--force` parameter to remove the running container without shutting it down. This will ungracefully shutdown the container and permanently remove it from the Docker host.
   >
   > In a production environment you may want to use `docker container stop` to gracefully stop the container and leave it on the host. You can then use `docker container rm` to permanently remove it.

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
$
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  echo $DOCKERID

[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker image build --tag $DOCKERID/linux_tweet_app:1.0 .
[+] Building 0.0s (0/0)                                         docker:default
ERROR: invalid tag "/linux_tweet_app:1.0": invalid reference format
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  echo $DOCKERID

[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker image build --tag $DOCKERID/linux_tweet_app:1.0 .
[+] Building 0.0s (0/0)                                         docker:default
ERROR: invalid tag "/linux_tweet_app:1.0": invalid reference format
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker container run \
>  --detach \
>  --publish 80:80 \
>  --name linux_tweet_app \
>  $DOCKERID/linux_tweet_app:1.0
docker: invalid reference format.
See 'docker run --help'.
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$  docker container rm --force linux_tweet_app
Error response from daemon: No such container: linux_tweet_app
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
```



7) Task 3: Modify the Running Website

Modify the running website

Bind mounts mean that any changes made to the local file system are immediately reflected in the running container.

1. Copy a new `index.html` into the container.

   The Git repo that you pulled earlier contains several different versions of an index.html file. You can manually run an `ls` command from within the `~/linux_tweet_app` directory to see a list of them. In this step we'll replace `index.html` with `index-new.html`.

   ```
   cp index-new.html index.html
   ```

2. Go to the running website and **refresh the page**. Notice that the site has changed.

> If you are comfortable with `vi` you can use it to load the local `index.html` file and make additional changes. Those too would be reflected when you reload the webpage. If you are really adventurous, why not try using `exec` to access the running container and modify the files stored there.

Even though we've modified the `index.html` local filesystem and seen it reflected in the running container, we've not actually changed the Docker image that the container was started from.

To show this, stop the current container and re-run the `1.0` image without a bind mount.

1. Stop and remove the currently running container.

   ```
   docker rm --force linux_tweet_app
   ```

2. Rerun the current version without a bind mount.

   ```
   docker container run \
   ```

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
Username: vnanditha
Password:
Error response from daemon: Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or password
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: cyberops17
Password:
Error response from daemon: Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or password
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$ docker image push $DOCKERID/linux_tweet_app:1.0
invalid reference format
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
cp index-new.html index.html
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$ docker rm --force linux_tweet_app
Error response from daemon: No such container: linux_tweet_app
[node1] (local) root@192.168.0.18 ~/linux_tweet_app
$
```

Test the new version

1. Run a new container from the new version of the image.

   ```
   docker container run \
   --detach \
   --publish 80:80 \
   --name linux_tweet_app \
   $DOCKERID/linux_tweet_app:2.0
   ```

2. Check the new version of the website (**You may need to refresh your browser to get the new version to load**).

   The web page will have an orange background.

   We can run both versions side by side. The only thing we need to be aware of is that we cannot have two containers using port 80 on the same host.

   As we're already using port 80 for the container running from the `2.0` version of the image, we will start a new container and publish it on port 8080. Additionally, we need to give our container a unique name (`old_linux_tweet_app`)

3. Run another new container, this time from the old version of the image.

   Notice that this command maps the new container to port 8080 on the host. This is because two containers cannot map to the same port on a single Docker host.

   ```
   docker container run \
   --detach \
   --publish 8080:80 \
   --name old_linux_tweet_app \
   $DOCKERID/linux_tweet_app:1.0
   ```

4. View the old version of the website.

```
 => [2/3] COPY index.html /usr/share/nginx/html                                0.1s
 => [3/3] COPY linux.png /usr/share/nginx/html                                 0.1s
 => exporting to image                                                         0.0s
 => => exporting layers                                                        0.0s
 => => writing image sha256:38816e0c9671297a899553a01ba2f2d057d00f8862fc400edce153578942608  0.0s
 => => naming to docker.io/shrimadhav/linux_tweet_app:2.0                      0.0s
[node1] (local) root@192.168.0.8 ~/linux_tweet_app
$ docker image ls
REPOSITORY                 TAG       IMAGE ID       CREATED          SIZE
shrimadhav/linux_tweet_app   2.0       38816e0c9671   6 seconds ago    188MB
shrimadhav/linux_tweet_app   1.0       ac471bf29ba8   10 minutes ago   188MB
[node1] (local) root@192.168.0.8 ~/linux_tweet_app
$ docker container run \
> --detach \
> --publish 80:80 \
> --name linux_tweet_app \
> $DOCKERID/linux_tweet_app:2.0
d935545b6cb54a3bc06f33630e0aaa287f83efa45767852586986dc666883fd9
[node1] (local) root@192.168.0.8 ~/linux_tweet_app
$ docker container run \
> --detach \
> --publish 8080:80 \
> --name old_linux_tweet_app \
> $DOCKERID/linux_tweet_app:1.0
96b48a1ccf7d673f9bc4e8ecaa3e4c5137a613369bda29fd37a799993064b2c9
[node1] (local) root@192.168.0.8 ~/linux_tweet_app
$
```

Docker Hub is the default public registry for Docker images.

2. Before you can push your images, you will need to log into Docker Hub.

```
docker login
```

You will need to supply your Docker ID credentials when prompted.

```
Username: <your docker id>
Password: <your docker id password>
Login Succeeded
```

3. Push version `1.0` of your web app using `docker image push`.

```
docker image push $DOCKERID/linux_tweet_app:1.0
```

```
ub. If you don't have a Docker ID, head over to https://hub.docker.com/ to create
 one.
You can log in with your password or a Personal Access Token (PAT). Using a limit
ed-scope PAT grants better security and is required for organizations using SSO.
Learn more at https://docs.docker.com/go/access-tokens/

Username: cyberops17
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[node1] (local) root@192.168.0.8 ~/linux_tweet_app
$
```

Sort by Newest | Filter Tags | Delete

TAG

1.0

Last pushed a few seconds ago by cyberops17

`docker pull cyberops17/linux_tweet_app:1.0` Copy

Digest	OS/ARCH	Last pull	Compressed Size
612063e0747f	linux/amd64	a few seconds ago	67.71 MB

TAG

2.0

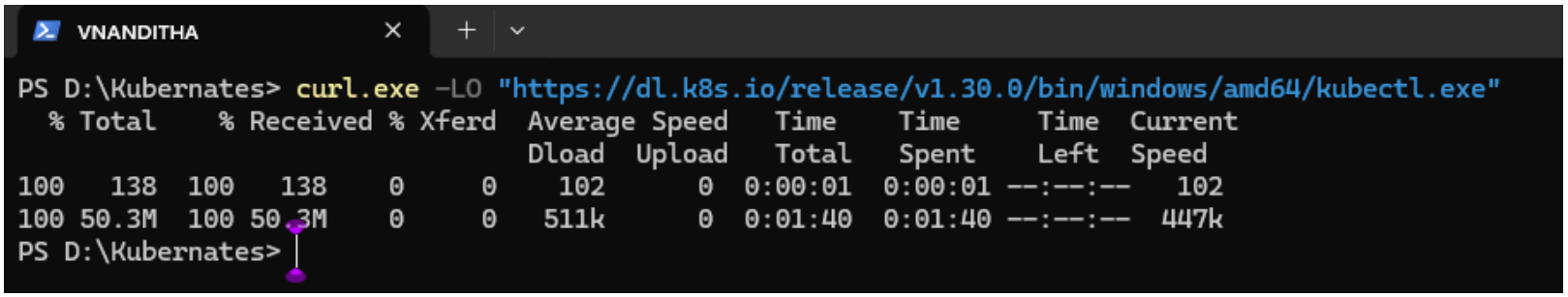
Last pushed a few seconds ago by cyberops17

`docker pull cyberops17/linux_tweet_app:2.0` Copy

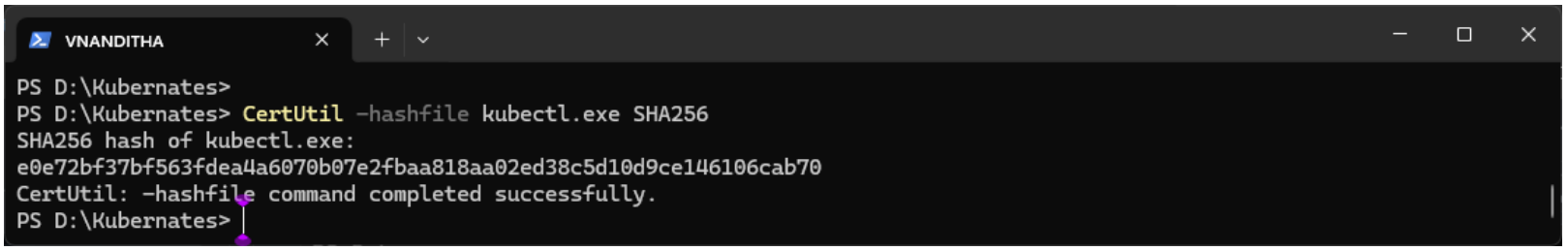
Digest	OS/ARCH	Last pull	Compressed Size
612063e0747f	linux/amd64	a few seconds ago	67.71 MB

KUBERNETES

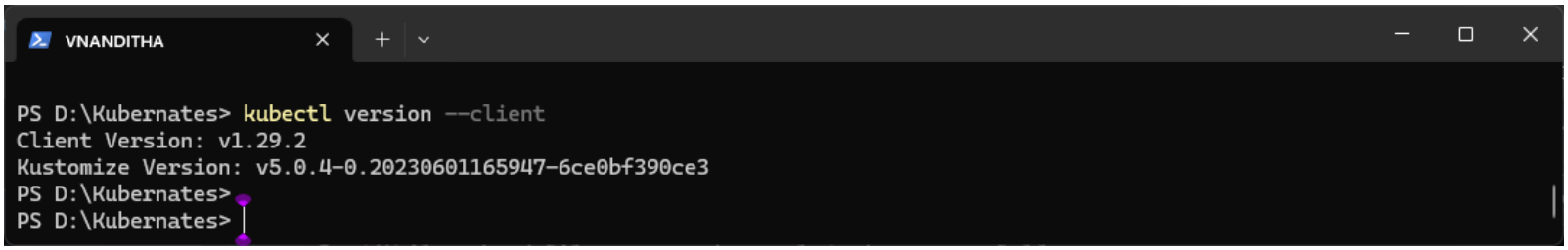
INSTALLATION : curl.exe -LO "https://dl.k8s.io/release/v1.30.0/bin/windows/amd64/kubectl.exe"



CertUtil -hashfile kubectl.exe SHA256

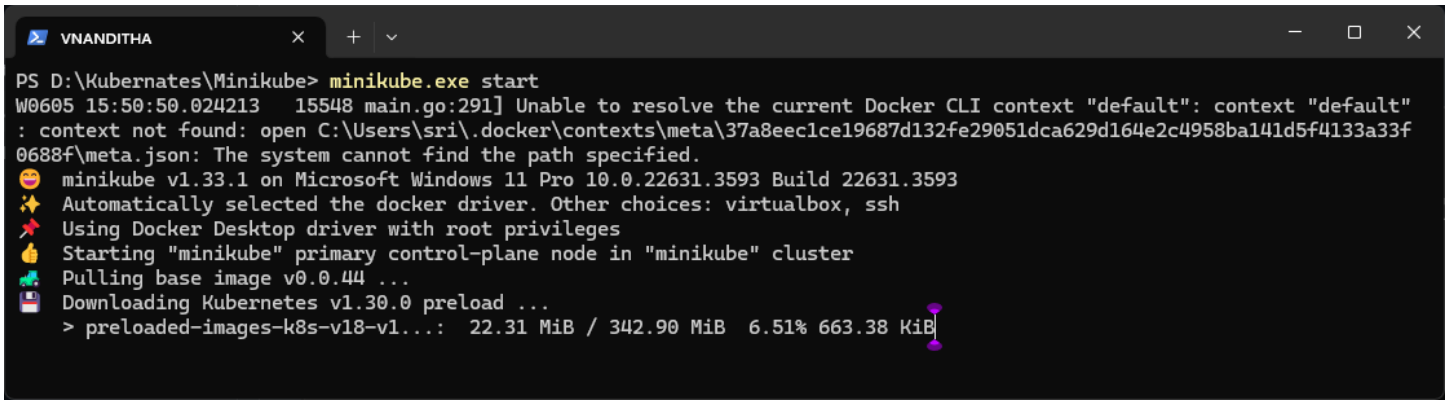


kubectl version –client

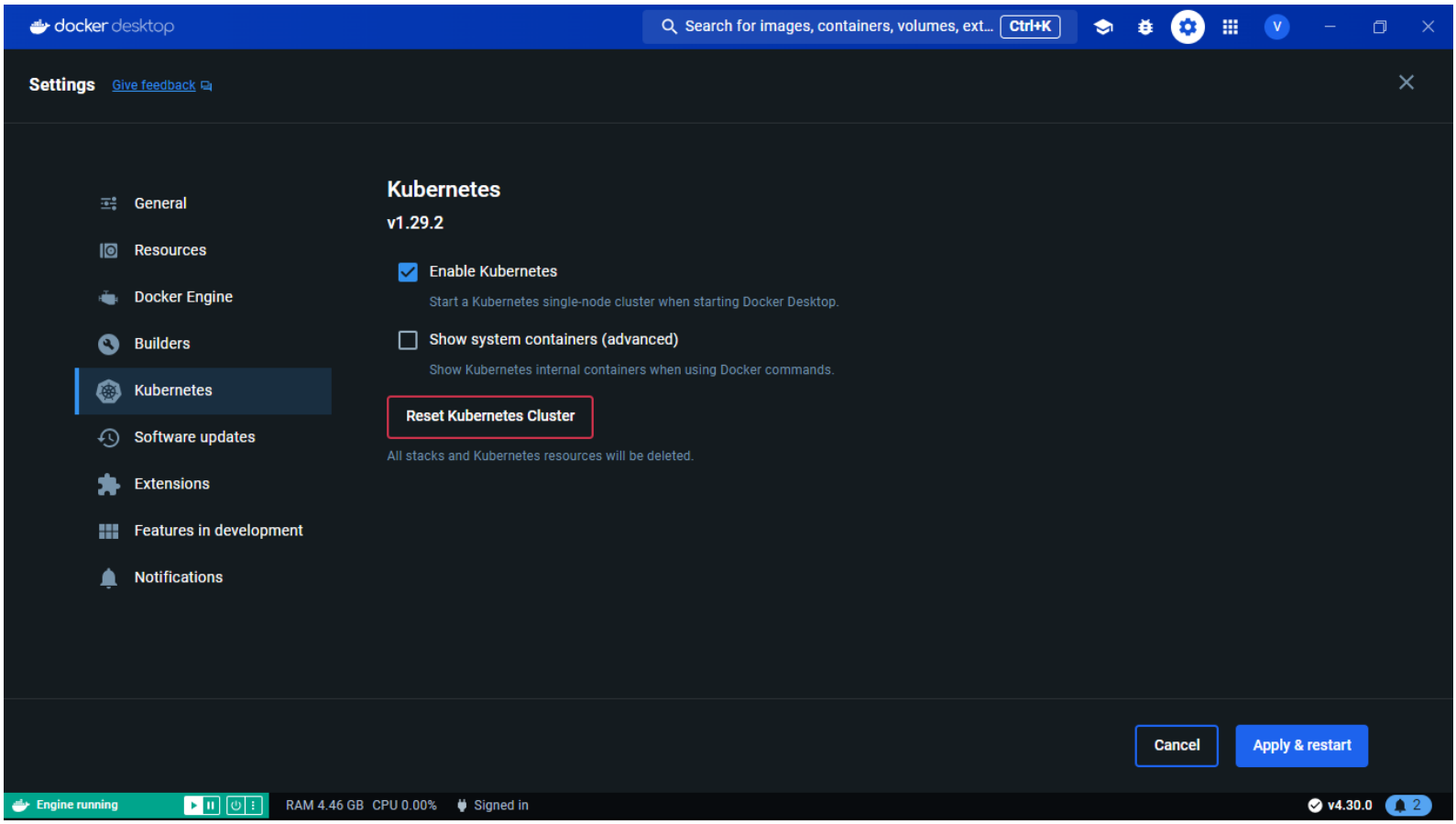


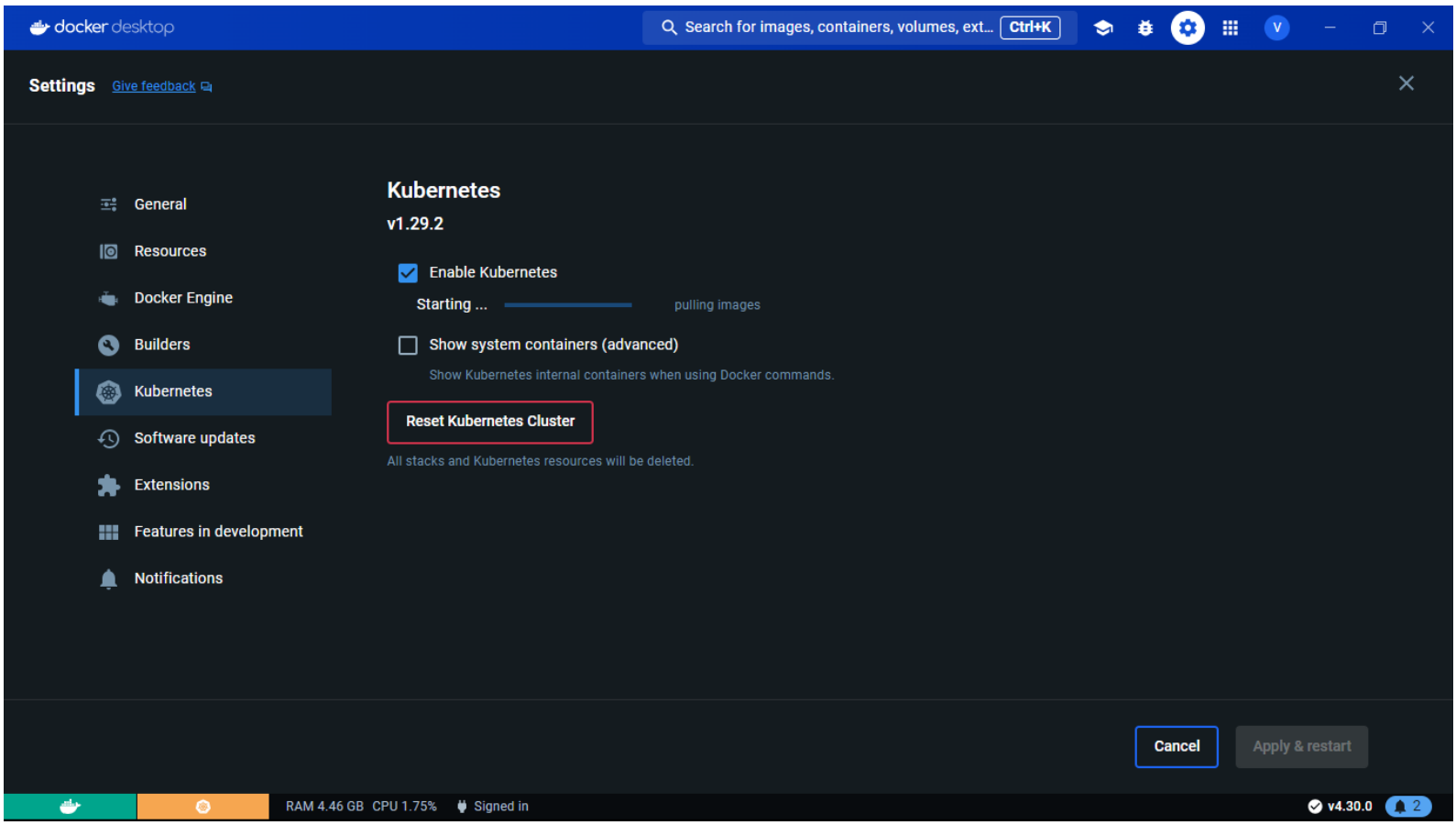
Open the downloaded directory and start minikube.exe

minikube.exe start



Docker Container > Settings > Kubernetes > Enable Kubernetes > Apply Restart

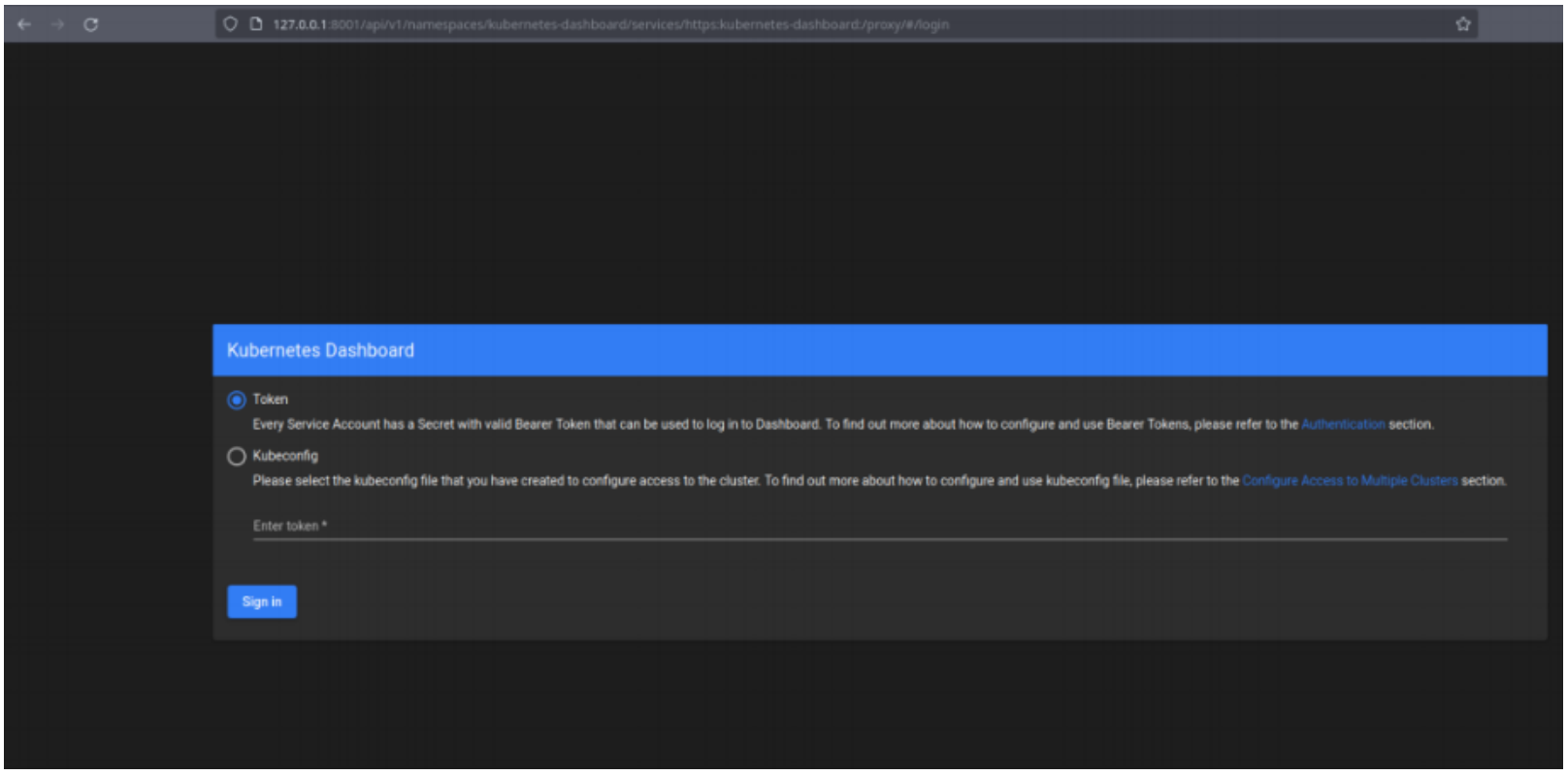


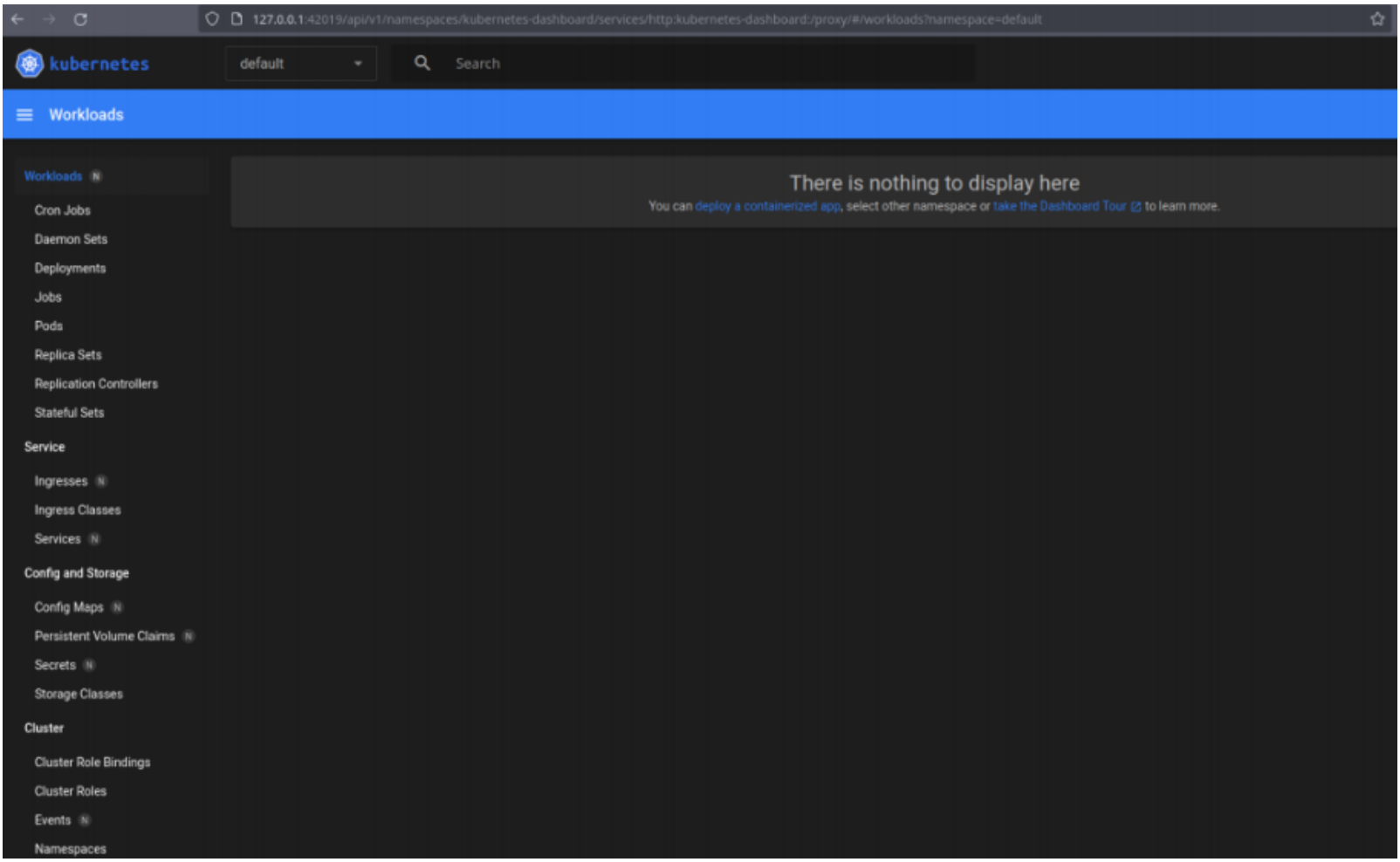


```
PS D:\> kubectl version
Client Version: v1.29.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.30.0
```

```
PS D:\> kubectl get pods --namespace=kube-system
NAME                               READY   STATUS    RESTARTS         AGE
coredns-7db6d8ff4d-6vjvp           1/1     Running   0                94m
etcd-minikube                      1/1     Running   0                94m
kube-apiserver-minikube            1/1     Running   1 (5m29s ago)    94m
kube-controller-manager-minikube   1/1     Running   0                94m
kube-proxy-bzkhg                   1/1     Running   0                94m
kube-scheduler-minikube            1/1     Running   0                94m
storage-provisioner                1/1     Running   2 (5m18s ago)    94m
```

```
PS D:\> kubectl apply -f https://github.com/kubernetes/dashboard/raw/v2.7.0/aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```





V NANDITHA
[CB.SC.P2CYS23018]