DOCKER

TEAM: 121AD0012, 121AD0014, 121AD0019, 121AD0020

What is Docker and why do we need it?

Docker is a platform that helps developers build, run and share applications using *containers.*

We need docker because:

Deploying the same code over systems having different packages or modules or different architecture will pose a problem of giving new errors each time. This is a hassle for the developers to edit and debug the code every time it is shifted from one system to another. To overcome this problem we are using software like Docker. It helps keep the module and packages of the same version for the code to be deployed anywhere and anytime with ease.

Docker Images and Containers

Image is the template of the project. ***Container*** is the running instance of the image. We can say that the details of the application are fed into the image, which is read-only. It will not change if the changes to application are made after building the image. So once the image is created, we are ready to run it via containers. We cannot run the images directly.

We can have a flowchart as follows for better understanding:

Installing Docker

Install the docker desktop installer. Then directly run it without any changes. Once it is run, then all files related are installed. Now after completion, we can run it directly.

Running Alpine

We use the command: *docker pull alpine*. Alpine is a minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size.

```
PS C:\Users\gssou> docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
4abcf2066143: Pull complete
Digest: sha256:c5b1261d6d3e43071626931fc004f70149baeba2c8ec672bd4f27761f8e1ad6b
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview alpine
PS C:\Users\gssou> docker run -it alpine
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # cd etc
/etc # ls
alpine-release   group            logrotate.d      network          profile          services         sysctl.d
apk              hostname         modprobe.d       nsswitch.conf    profile.d        shadow           udhcpc
busybox-paths.d  hosts            modules          opt              protocols        shells           udhcpd.conf
conf.d           init.d           modules-load.d   os-release       resolv.conf      ssl
crontabs         inittab          motd             passwd           secfixes.d       ssl1.1
fstab            issue            mtab             periodic         securetty        sysctl.conf
/etc # cd apk
/etc/apk # ls
arch               keys               protected_paths.d  repositories       world
/etc/apk # nano world
/bin/sh: nano: not found
/etc/apk # world
/bin/sh: world: not found
/etc/apk # exit
PS C:\Users\gssou>
```

Running a sample Python App on the containers

1. **Create a python file of app**

   I have created a python file named “app.py” which contains the sample code for a basic calculator. I have saved the file in directory path = “D:\Semester VI Stuff\srid”.

2. **Create a Dockerfile file**

   In the same directory, I have created a file named “Dockerfile” through the text editor - Notepad. But it saves the files with extension “txt”. We need to keep a note of it that the Dockerfile should not have any kind of extension to it else it will be difficult for docker to understand that it is the instruction file.

   I faced this problem while executing the build command directly. I got the following error:

```
[+] Building 0.1s (2/2) FINISHED
docker:default
 => [internal] load build definition from Dockerfile  0.0s
 => => transferring dockerfile: 2B                    0.0s
 => [internal] load .dockerignore                     0.0s
 => => transferring context: 2B                            0.0s
ERROR: failed to solve: failed to read dockerfile: open
/var/lib/docker/tmp/buildkit-mount3481637707/Dockerfile: no
such file or directory
```

We can overcome this error by specifying and telling the docker that Dockerfile is actually *Dockerfile.txt.* The Dockerfile contains:

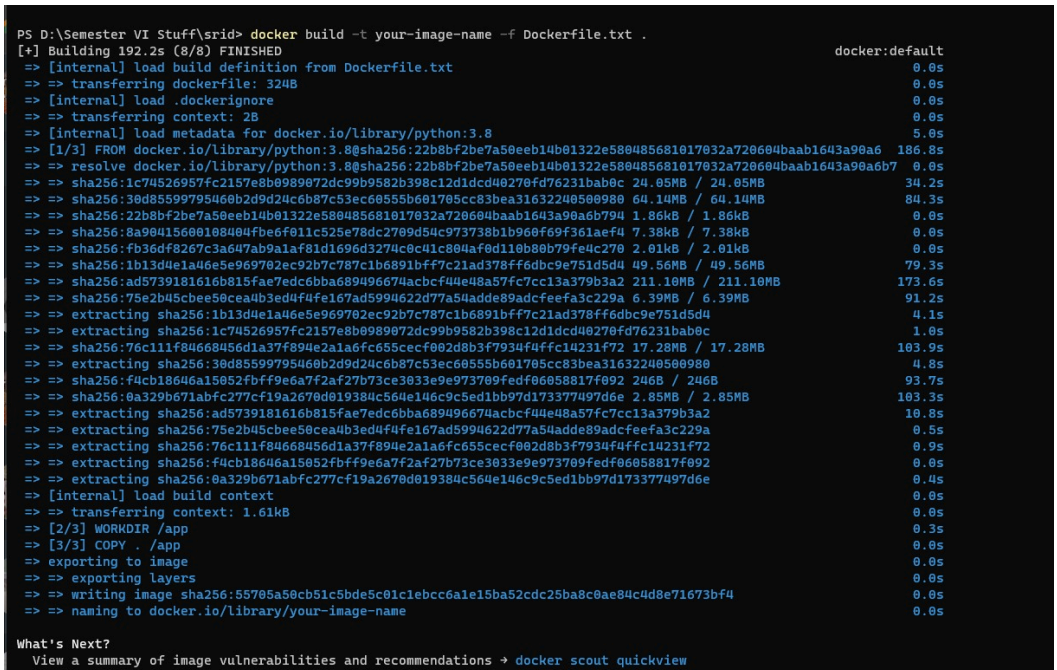
```
# Use the official Python image as the base image
FROM python:3.8
# Set the working directory in the container
WORKDIR /app
# Copy the current directory contents into the container at
/app
COPY . /app
# Run app.py when the container launches
CMD ["python", "app.py"]
```

These commands are like a script file that will execute one by one when the image is run on containers.

3. **Build the dockerfile into image**

Command: *docker build -t your-image-name -f Dockerfile.txt .*

-t is used to tag the image with a name and ***-f*** is used to specify the name of the Dockerfile. Once specifying these options, we can see the results as follows:



4. **Run the image through container**

   Command: *docker run -it your-image-name*

   -it is for interactive mode. It makes the container to be interactive or accept input streams. If we do not mention this option, then no input will be taken. However, for calculators we require inputs. Hence we are using "-it" mode.

   The same steps go for any other app that has been written in any other language. We just have to specify the parent image name in the Dockerfile correctly, so that the right package will be installed and used.

```
PS D:\Semester VI Stuff\srid> docker run -it your-image-name
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Enter choice (1/2/3/4/5): 1
Enter first number: 100
Enter second number: 200
100.0 + 200.0 = 300.0
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Enter choice (1/2/3/4/5): 5
Exiting the calculator.
PS D:\Semester VI Stuff\srid>
```

5. **Results and other conclusions**

   Other command executed:

   - **docker ps**: lists out the currently running containers. It has the attributes:

     CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

   - **docker ps -a**: it shows all the containers that have been exited and currently running.

```
PS D:\Semester VI Stuff\srid> docker ps -a
CONTAINER ID   IMAGE             COMMAND           CREATED             STATUS                      PORTS     NAMES
5568cc18f814   your-image-name   "python app.py"   56 minutes ago      Exited (1) 8 minutes ago              angry_mahavira
9f95540a3c62   your-image-name   "python app.py"   About an hour ago   Exited (0) 14 minutes ago             busy_gould
c8c959bbb8fa   your-image-name   "python app.py"   About an hour ago   Exited (1) 20 minutes ago             charming_curie
```

   - **docker stop image_name_or_id:** This will stop the container id mentioned.

- **docker info**: It will provide all information about docker. We may check the status and other details:

```
Client:
 Version:    24.0.7
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.12.0-desktop.2
    Path:     C:\Program Files\Docker\cli-plugins\docker-buildx.exe
  compose: Docker Compose (Docker Inc.)
    Version:  v2.23.3-desktop.2
    Path:     C:\Program Files\Docker\cli-plugins\docker-compose.exe
  dev: Docker Dev Environments (Docker Inc.)
    Version:  v0.1.0
    Path:     C:\Program Files\Docker\cli-plugins\docker-dev.exe
  extension: Manages Docker extensions (Docker Inc.)
    Version:  v0.2.21
    Path:     C:\Program Files\Docker\cli-plugins\docker-extension.exe
  feedback: Provide feedback, right in your terminal! (Docker Inc.)
    Version:  0.1
    Path:     C:\Program Files\Docker\cli-plugins\docker-feedback.exe
  init: Creates Docker-related starter files for your project (Docker Inc.)
    Version:  v0.1.0-beta.10
    Path:     C:\Program Files\Docker\cli-plugins\docker-init.exe
  sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc.)
    Version:  0.6.0
    Path:     C:\Program Files\Docker\cli-plugins\docker-sbom.exe
  scan: Docker Scan (Docker Inc.)
    Version:  v0.26.0
    Path:     C:\Program Files\Docker\cli-plugins\docker-scan.exe
  scout: Docker Scout (Docker Inc.)
    Version:  v1.2.0
    Path:     C:\Program Files\Docker\cli-plugins\docker-scout.exe
```

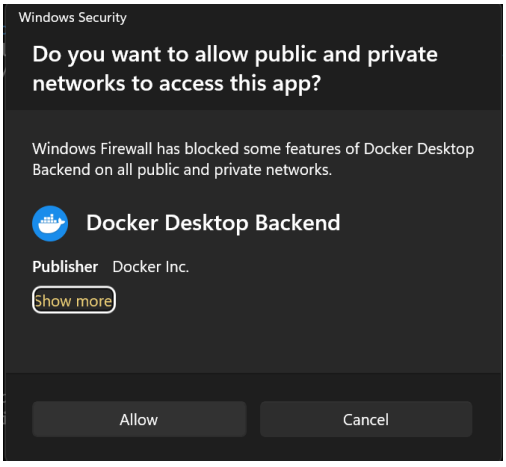
```
Server:
 Containers: 3
  Running: 0
  Paused: 0
  Stopped: 3
 Images: 1
 Server Version: 24.0.7
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: d8f198a4ed8892c764191ef7b3b06d8a2eeb5c7f
 runc version: v1.1.10-0-g18a0cb0
 init version: de40ad0
 Security Options:
  seccomp
   Profile: unconfined
 Kernel Version: 5.15.133.1-microsoft-standard-WSL2
 Operating System: Docker Desktop
 OSType: linux
 Architecture: x86_64
 CPUs: 12
 Total Memory: 7.439GiB
 Name: docker-desktop
 ID: 9f8bed89-ad01-462f-896e-045ae5fbfaf7
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 HTTP Proxy: http.docker.internal:3128
 HTTPS Proxy: http.docker.internal:3128
 No Proxy: hubproxy.docker.internal
 Experimental: false
 Insecure Registries:
  hubproxy.docker.internal:5555
  127.0.0.0/8
 Live Restore Enabled: false
```

Pulling images like python, node, and cpp_gcc

Command: *docker pull node*, *docker pull python*, *docker pull eclipse/cpp_gcc*

Detached mode

Command: *docker run -d -p 4000:80 your-image-name*




```
PS C:\Users\gssou> docker run -d -p 4000:80 your-image-name
ee4c534615c28b1b3f31435b7109aa705c1426b24894b1872ff3594dcbe8f74a
PS C:\Users\gssou> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\gssou>
```

This makes the app run in the background. But, as we have typed command “docker ps” to check which processes are running, we find none. The reason is that our app requires input from the user. This makes the app end abruptly due to blockage of the input stream.

Volumes

Volumes in Docker are a way to *persist* data generated by and used by Docker containers. They provide a mechanism for sharing and storing data between the host machine and containers, as well as between different containers.

In order to create a volume, we have made changes to the Dockerfile first. The updated dockerfile is as follows:

```
# Use the official Python image as the base image
FROM python:3.8
# Set the working directory in the container
WORKDIR /app
# Create /app/results directory
RUN mkdir /app/results
# Copy the current directory contents into the container at /app
COPY . /app
#make /app/results a volume:
VOLUME /app/results
# Run app.py when the container launches
CMD ["python", "app.py"]
```

We notice that two lines are added. First, “`RUN mkdir /app/results`” creates a directory for the results to be stored. Next, “`VOLUME /app/results`” is used to make the directory as a volume. Now our dockerfile is ready to run.

But before proceeding to build the Dockerfile, there are two more things to be done. *First* we create a directory in our folder where Dockerfile is located for all results to be stored, say folder name is “results”. *Second* thing to be done is the changes are to be made to the python code, app.py, so that the code writes some results or logs to the text file. We need to

make sure that the directory or path of that text file in app.py is set as "`open('/app/results/result.txt','a')`" kind of format.

After making these changes, we have run the commands for build and run:

docker build -t sri_ volume -f Dockerfile.txt .

Here ***sri_ volume*** is the name of my container for this application. Next:

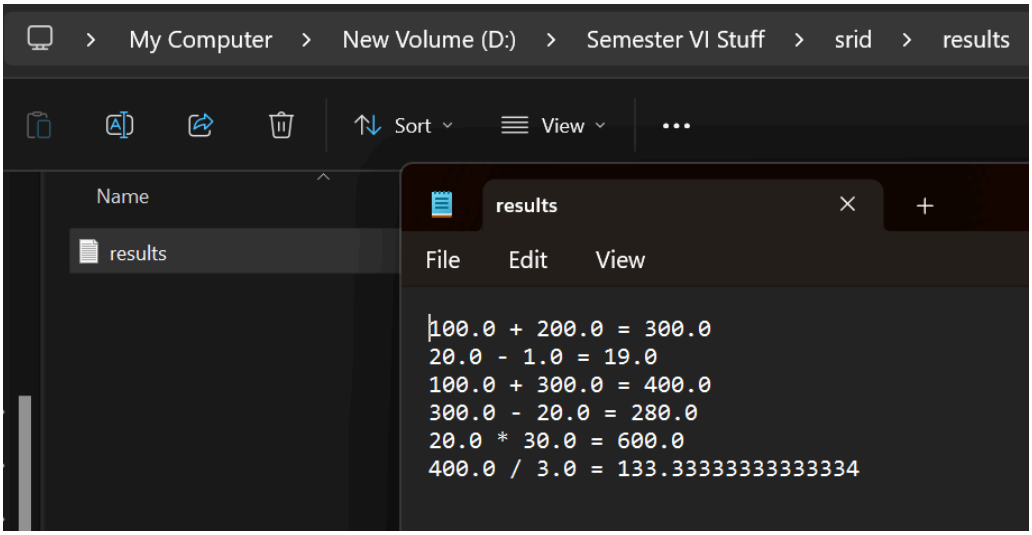
docker run -it -v "D:\Semester VI Stuff\srid\results:/app/results" -p 4000:80 sri_ volume

Here ***-it*** refers to the interactive mode, ***-v*** refers to the volume being stored in the path given after it. Since my path has some spaces in it, it needs to be put in quotation marks. Next ***-p*** defines the ports to be used. Next, we have the name of the container.

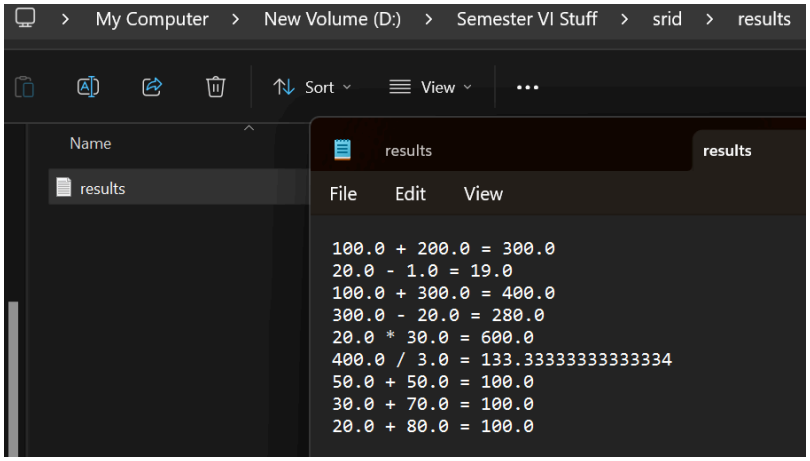
```
PS D:\Semester VI Stuff\srid> docker build -t sri_volume -f Dockerfile.txt .
[+] Building 3.7s (9/9) FINISHED                                                                                    docker:default
 => [internal] load build definition from Dockerfile.txt                                                                      0.0s
 => => transferring dockerfile: 436B                                                                                          0.0s
 => [internal] load .dockerignore                                                                                             0.0s
 => => transferring context: 2B                                                                                               0.0s
 => [internal] load metadata for docker.io/library/python:3.8                                                                 3.6s
 => [1/4] FROM docker.io/library/python:3.8@sha256:22b8bf2be7a50eeb14b01322e580485681017032a720604baab1643a90a6b794            0.0s
 => [internal] load build context                                                                                             0.0s
 => => transferring context: 177B                                                                                             0.0s
 => CACHED [2/4] WORKDIR /app                                                                                                 0.0s
 => CACHED [3/4] RUN mkdir /app/results                                                                                       0.0s
 => [4/4] COPY . /app                                                                                                         0.0s
 => exporting to image                                                                                                        0.0s
 => => exporting layers                                                                                                       0.0s
 => => writing image sha256:af3987269a9e48dbb315fec642bd7ed4e7bd9fe6ba19285751e2f161aeada385                                  0.0s
 => => naming to docker.io/library/sri_volume                                                                                 0.0s

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout quickview
PS D:\Semester VI Stuff\srid> docker run -it -v "D:\Semester VI Stuff\srid\results:/app/results" -p 4000:80 sri_volume
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Enter choice (1/2/3/4/5): 1
Enter first number: 100
Enter second number: 300
```

After executing a few operations on the calculator application, and ending the program. We can see that the results.txt has been created:

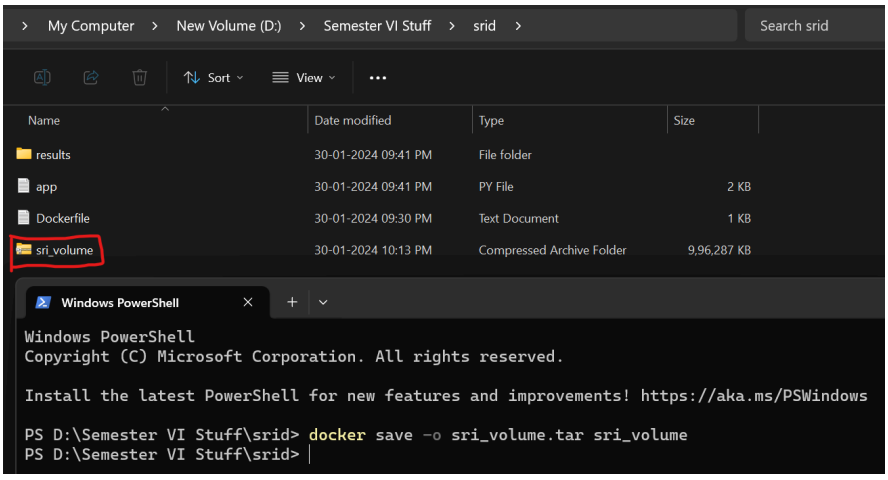


I could see all my operations in the file specified as above. So we have now stored the operations and data, so that it doesn't get lost. But, we need to pay attention to the fact that on *running the build command again*, it creates a fresh file from the start. But if we are simply running the created image of the application, we can see that the operations stack up. After closing the terminal, opening a new one and running the image directly with some operations made changes to the file, but previous ones are not lost. This was not the case when I ran the build and run commands. On building, the results file is totally new, with the new operations only.



Transferring an image from my laptop to my friend's laptop

We may transfer images by saving and loading. We will first save the image as a .tar file.



docker save -o sri_ volume.tar sri_ volume

Next we transfer it to our friend's laptop through usb, then we can unzip the file through load command→ *docker load -i your-image-name.tar*

This will load out our image. Now we can run that image.

- There is another easy way to do this. We can do this through **Docker Hub**, which is a public registry where we can store and share Docker images. We can push it to Docker Hub, and our friend can pull it from there.

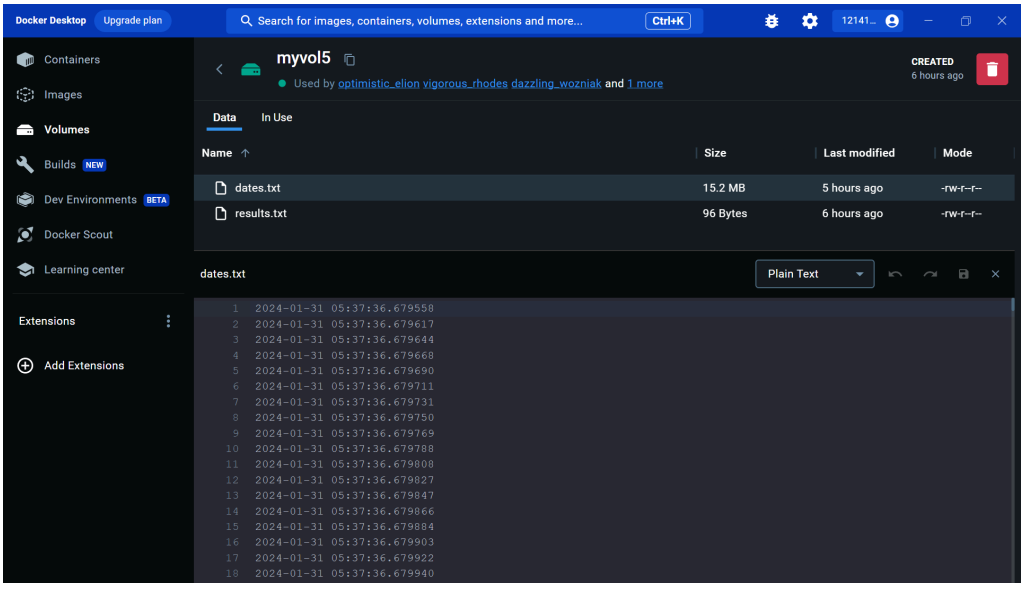
Using Volumes in DockerDesktop

Till now we have used a folder on a local PC as a storage volume. Now we want to store it on the docker desktop itself. Thus we need to create a volume and then push our data that is being generated inside it. In order to do so, I have made no change to the python app.py file. The changes are also not made in the Dockerfile. Commands:

docker volume create volume-name-here → **optional**, even if we don't execute this, our volume gets created on spot.

docker build -t image-name-here -f Dockerfile.txt .

docker run -it -v volume-name-here:/app/results image-name-here → The only difference made is in this command. Instead of putting the local path, we have put the *volume-name-here*, and the rest is the same. Now we can see the results.txt in Docker Desktop →Volumes → volume-name-here. Also, I have experimented that we can have the same volume for different containers. The results or output data gets stored at one place.

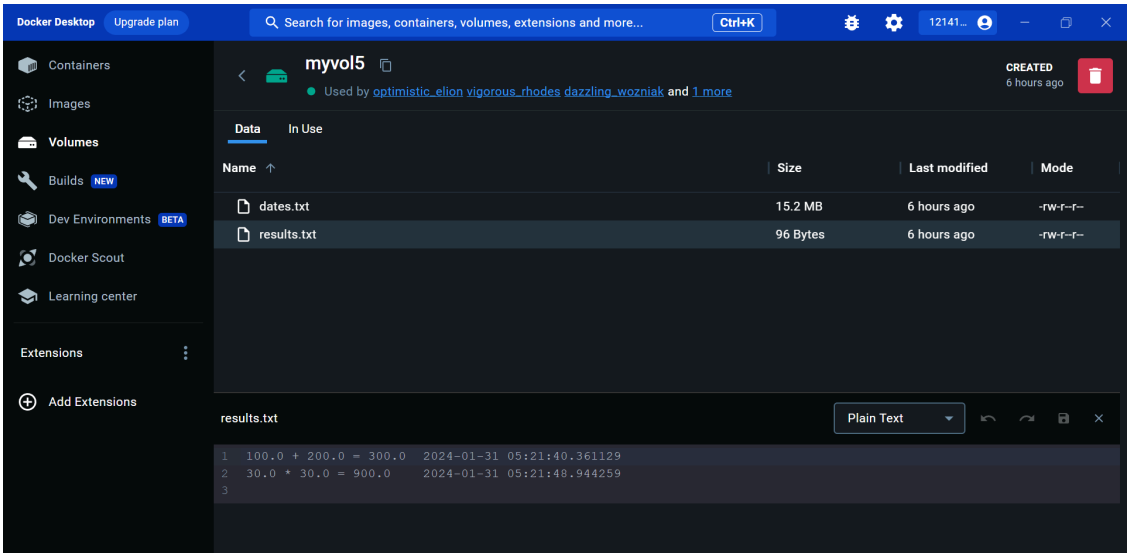


Running container in Detached Mode

Till now we have used ***-it*** for the interactive mode, now the detached mode with ***-d*** makes the program run in the background. I have tried running this app program itself in the background, but failed. This was because it needed input to run. Without that, the program faces error and stops. Unlike this app.py, I tried another app.py where we are continuously printing date and time stamps for an infinite loop. Now, when we run the app:

docker build -t det1 -f Dockerfile.txt .
docker run -d -v myvol5:/app/results det1
docker stop d831b2d3b82f0669a9c44741afcbb42f4bae044eac0e1e37ce23f4a2dfa9f886

This created a new text file, as I changed the name of the file in app.py to write the timestamps to a new text file(dates.txt, in the same volume, *myvol5*). I can see them in the DockerDesktop. Finally, we stop it manually, as it would not end.



Using private repository to transfer image and run it in another machine

A react application was created in another machine which was pushed into a private repository, created on docker hub website. After pushing data, I logged in into that account using the credentials on my machine. Next I pulled the app image.

docker login -u <username> -p <password>

docker pull 12141920/myreactapp:latest

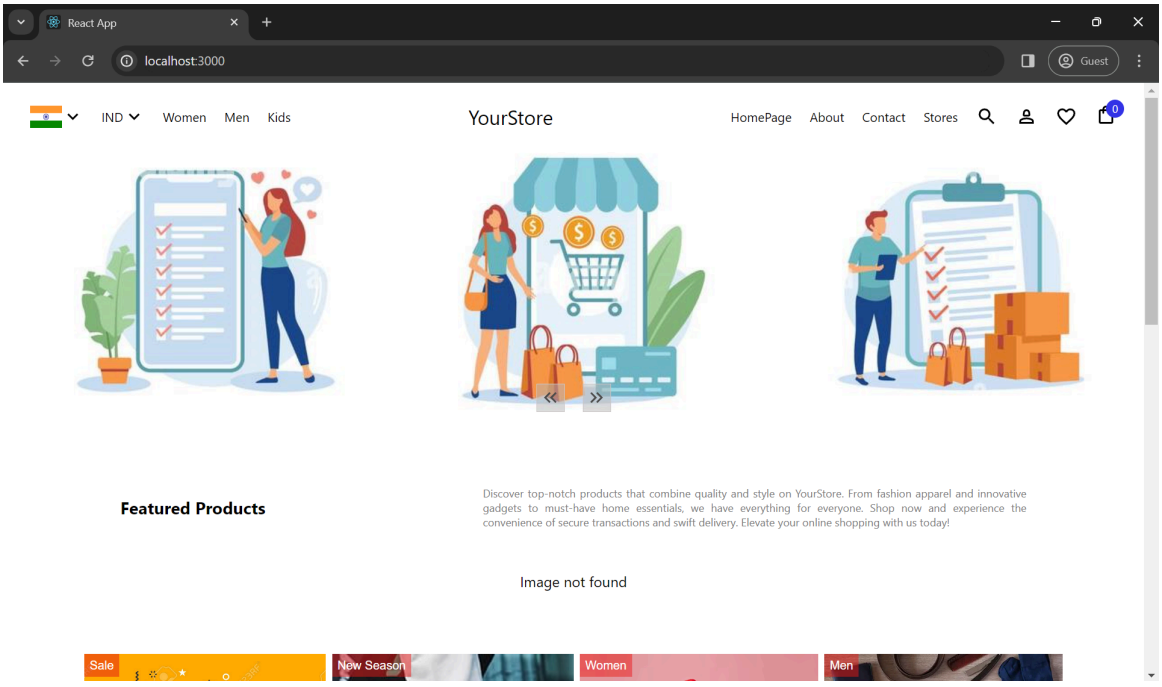
```
PS D:\Semester VI Stuff\srid3> docker pull 12141920/myreactapp:latest
latest: Pulling from 12141920/myreactapp
d52e4f012db1: Pull complete
7dd206bea61f: Pull complete
2320f9be4a9c: Pull complete
6e5565e0ba8d: Pull complete
5f1526a28cf9: Pull complete
2f0191f7d60a: Pull complete
1104f0e2cc5e: Pull complete
3f3e951e9c53: Pull complete
2eb9dd14c7ba: Pull complete
4098ee142e17: Pull complete
ff27ac532fa4: Pull complete
4bdb33389b0d: Pull complete
Digest: sha256:3eebcd3fee1ee320da58e94c5227e619b028a5b9c24766e960a148d9930f5a8e
Status: Downloaded newer image for 12141920/myreactapp:latest
docker.io/12141920/myreactapp:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview 12141920/myreactapp:latest
PS D:\Semester VI Stuff\srid3> docker run -p 3000:3000 12141920/myreactapp

> client@0.1.0 start
> react-scripts start

Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
(node:31) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use t
he 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:31) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please us
 the 'setupMiddlewares' option.
Starting the development server...
```

On google we type *localhost:3000* as the port is set to 3000 generally for react apps and also the react app Dockerfile has a line: expose 3000(setting port number).



It shows the website. This shows that we have successfully pushed an image from a machine and pulled it into another machine without facing any problems.