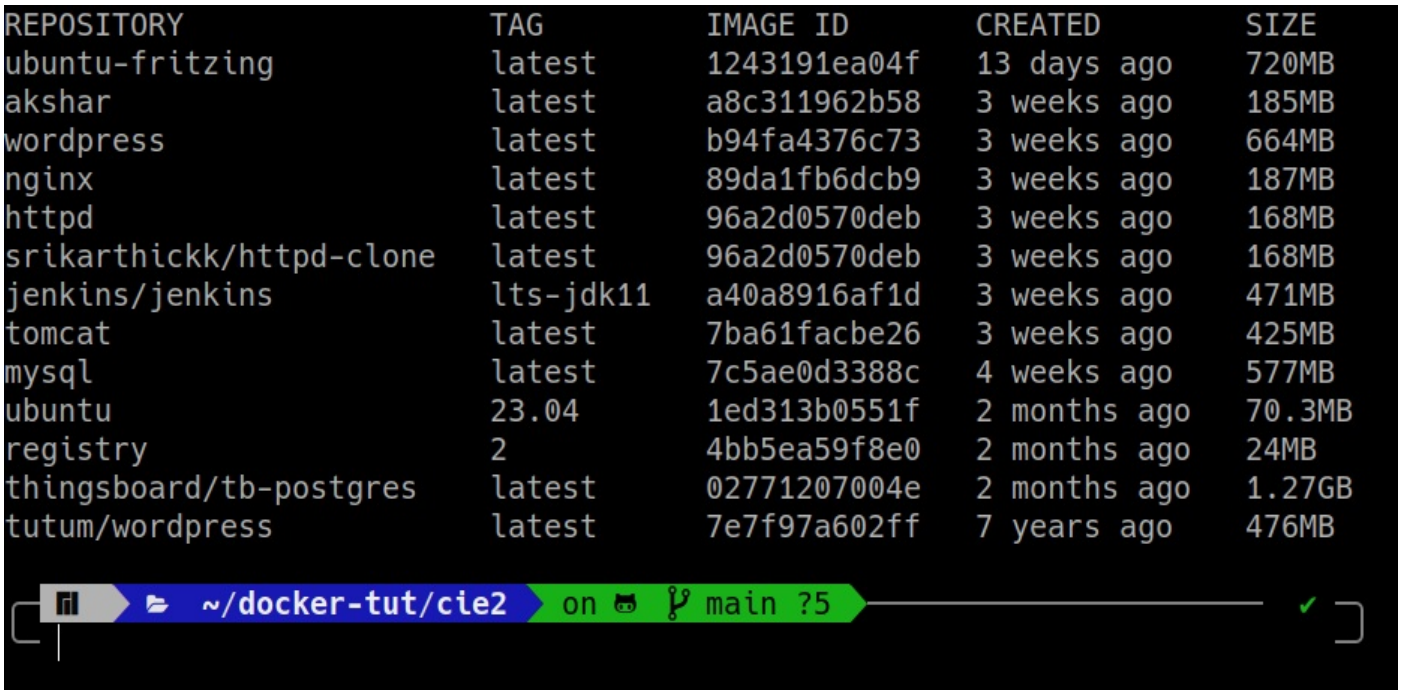
CIE 2 TECHNIQUES

Technique 15 : Save game

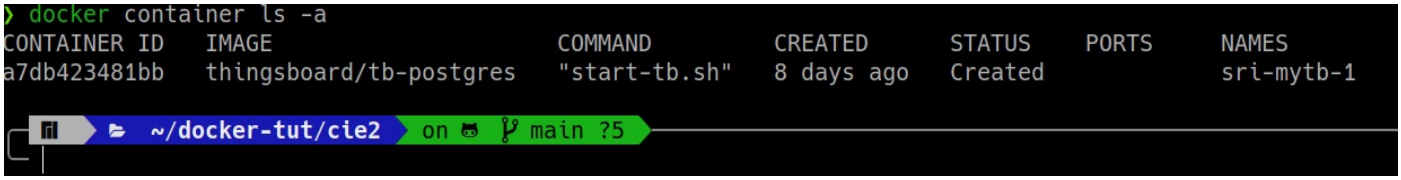
Save game is a technique of saving the state of a container as an image by commiting the container.

STEPS

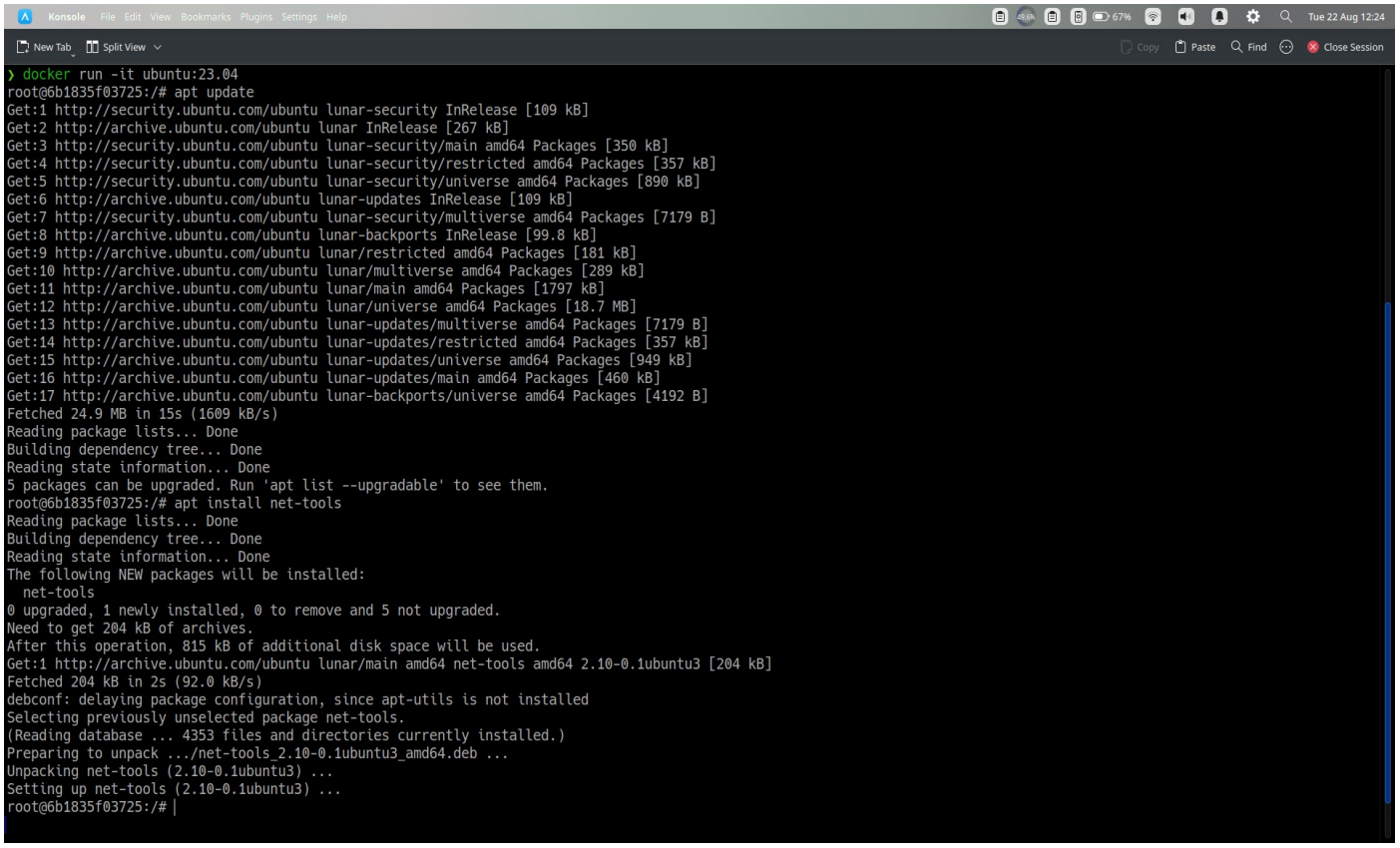
1. List all the images `docker images`



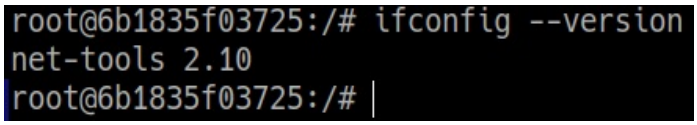
2. View containers `docker container ls -a`



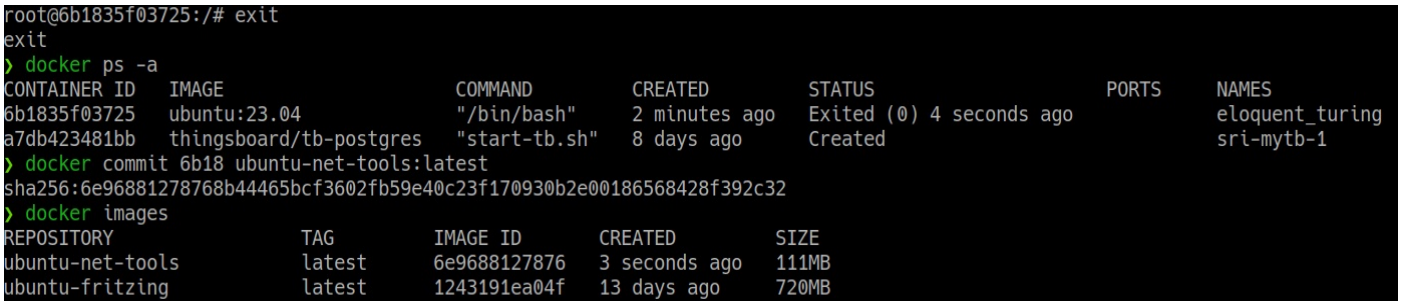
3. Run an image in interactive mode `docker run -it ubuntu:23.04`
4. Update the packages of the container `apt update`
5. Install a tool in the container `apt install net-tools`



6. Verify the installation of net-tools `ifconfig --version`



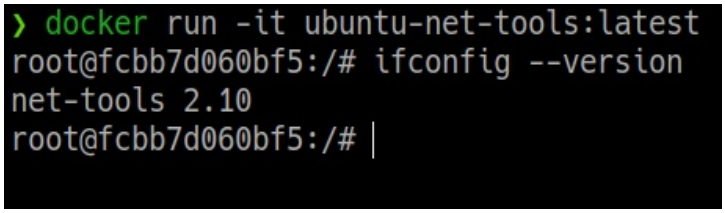
7. Exit the container `exit`
8. Display list of launched containers `docker ps -a`
9. Commit the Container with ID and provide name `docker commit <container_id> <image_name>`
10. Verify the commit `docker images`



The newly created image contains the ifconfig installed on it. All the changes made to the container would have been saved as an image when we commit the container

Additionally to verify the commit changes

1. Run the newly created image `docker run -it <image_name>:<version>`
2. Verify the installation of net-tools `ifconfig --version`

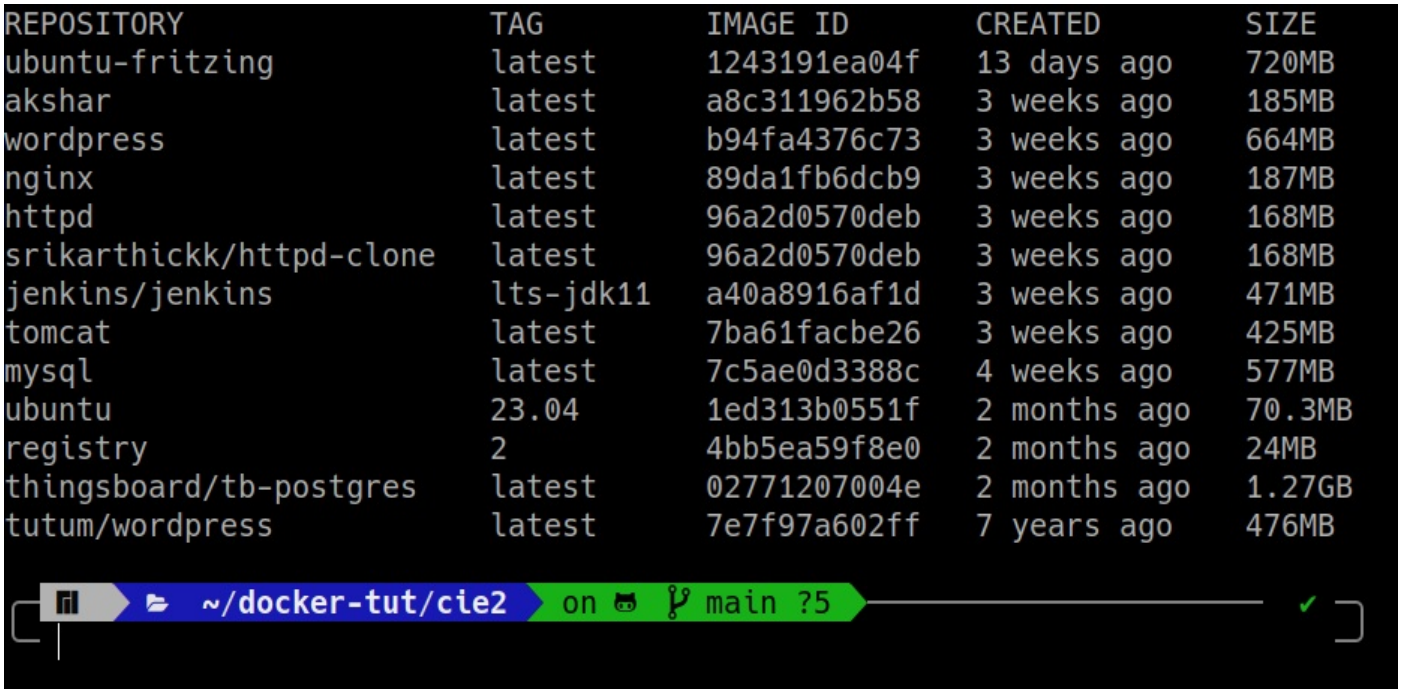


Technique 16 : Docker Tagging

Docker Tagging is a technique of assigning a label to a specific version of docker image

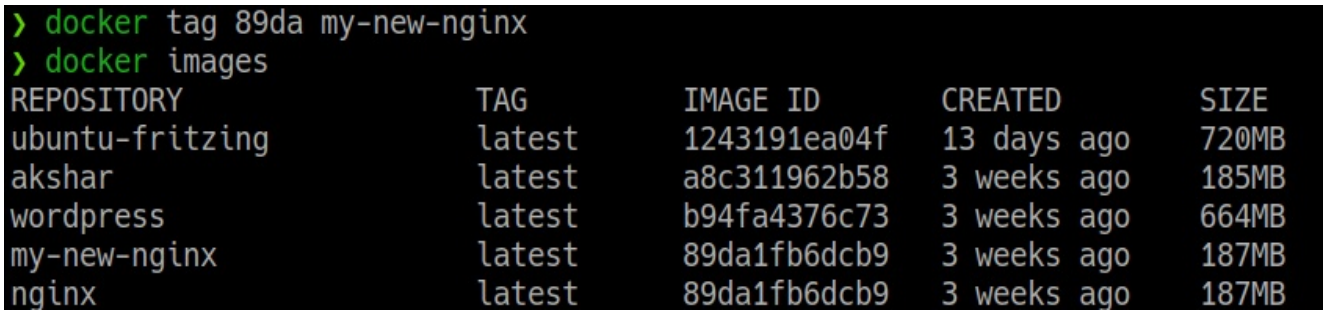
STEPS

1. List the images `docker images`



2. Specify the new name to the existing image `docker tag <existing_image_id> <new_image_name>`

3. Verify the tagged image

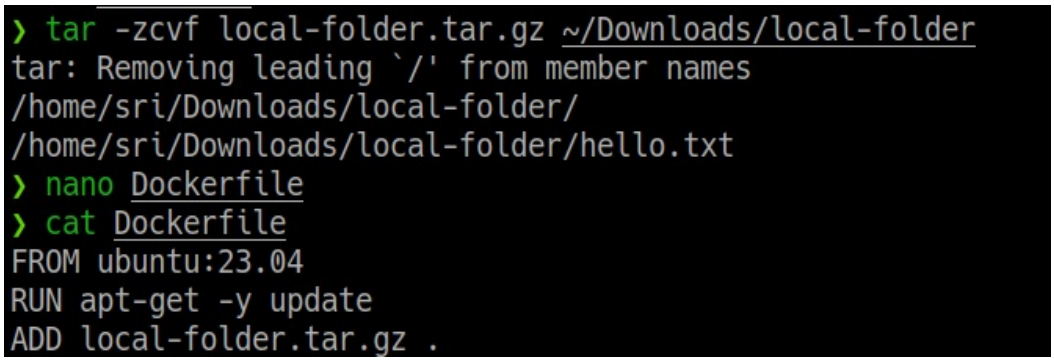


Technique 20 : Injecting Files into Image using ADD

STEPS

1. Create a Tar File `tar -zcvf local-folder.tar.gz ~/Downloads/local-folder`

2. Append the contents to a Dockerfile `FROM ubuntu:23.04 RUN apt-get -y update ADD local-folder.tar.gz .`



3. Building the Docker Image `sudo docker build -t some-image .`

4. List the images `docker images`

```
❯ sudo docker build -t some-image .

[sudo] password for sri:
[+] Building 15.8s (8/8) FINISHED
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 103B
 => [internal] load metadata for docker.io/library/ubuntu:23.04
 => CACHED [1/3] FROM docker.io/library/ubuntu:23.04
 => [internal] load build context
 => => transferring context: 262B
 => [2/3] RUN apt-get -y update
 => [3/3] ADD local-folder.tar.gz .
 => exporting to image
 => => exporting layers
 => => writing image sha256:f8949a73868018d02419e7653f0342a3366aff61ccde2c1e7cf89b56195266e0
 => => naming to docker.io/library/some-image

What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quickview
❯ docker images
REPOSITORY                TAG         IMAGE ID       CREATED          SIZE
some-image                latest      f8949a738680   18 seconds ago   110MB
```

5. Running the Docker Container `sudo docker run -it sample-image bash`

6. Verifying the Extraction `ls`

```
❯ docker run -it some-image bin/bash
root@a90e3c296321:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@a90e3c296321:/# cd home
root@a90e3c296321:/home# ls
sri  ubuntu
root@a90e3c296321:/home# cd sri
root@a90e3c296321:/home/sri# ls
Downloads
root@a90e3c296321:/home/sri# cd Downloads
root@a90e3c296321:/home/sri/Downloads# ls
local-folder
root@a90e3c296321:/home/sri/Downloads# cd local-folder/
root@a90e3c296321:/home/sri/Downloads/local-folder# cat hello.txt
Hello world from outside the container
root@a90e3c296321:/home/sri/Downloads/local-folder#
```

Technique 21 : Rebuilding without cache

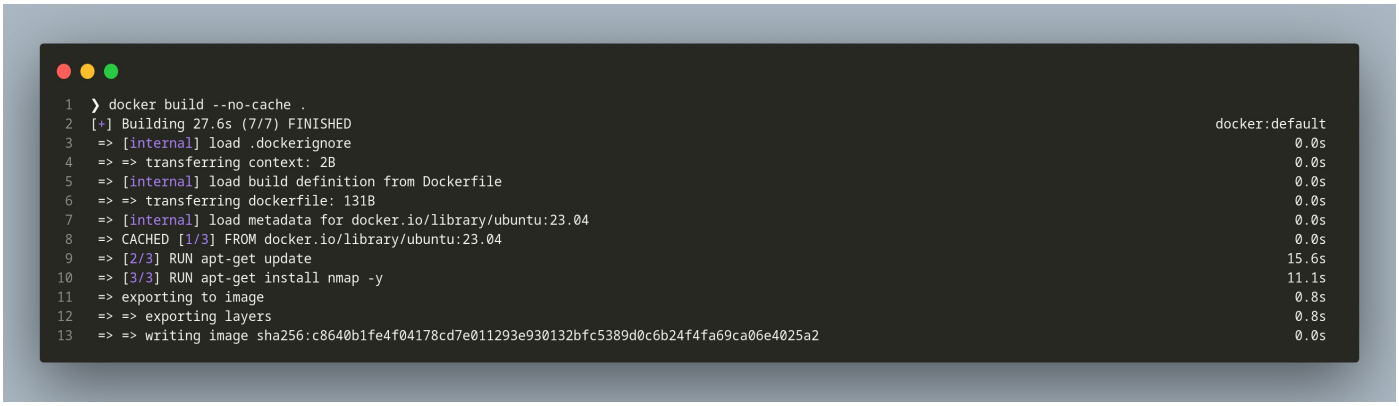
STEPS

1. Create a Docker file and append the following

```
1  FROM ubuntu:23.04
2  RUN apt-get update
3  RUN apt-get install nmap -y
4  CMD [ "nmap", "--version" ]
5
```

1. Run the build command with –no-cache option

```
docker build --no-cache .
```



```
❯ docker build --no-cache .
[+] Building 27.6s (7/7) FINISHED                                                                                  docker:default
 => [internal] load .dockerignore                                                                                           0.0s
 => => transferring context: 2B                                                                                             0.0s
 => [internal] load build definition from Dockerfile                                                                        0.0s
 => => transferring dockerfile: 131B                                                                                        0.0s
 => [internal] load metadata for docker.io/library/ubuntu:23.04                                                             0.0s
 => CACHED [1/3] FROM docker.io/library/ubuntu:23.04                                                                        0.0s
 => [2/3] RUN apt-get update                                                                                               15.6s
 => [3/3] RUN apt-get install nmap -y                                                                                      11.1s
 => exporting to image                                                                                                      0.8s
 => => exporting layers                                                                                                     0.8s
 => => writing image sha256:c8640b1fe4f04178cd7e011293e930132bfc5389d0c6b24f4fa69ca06e4025a2                                0.0s
```

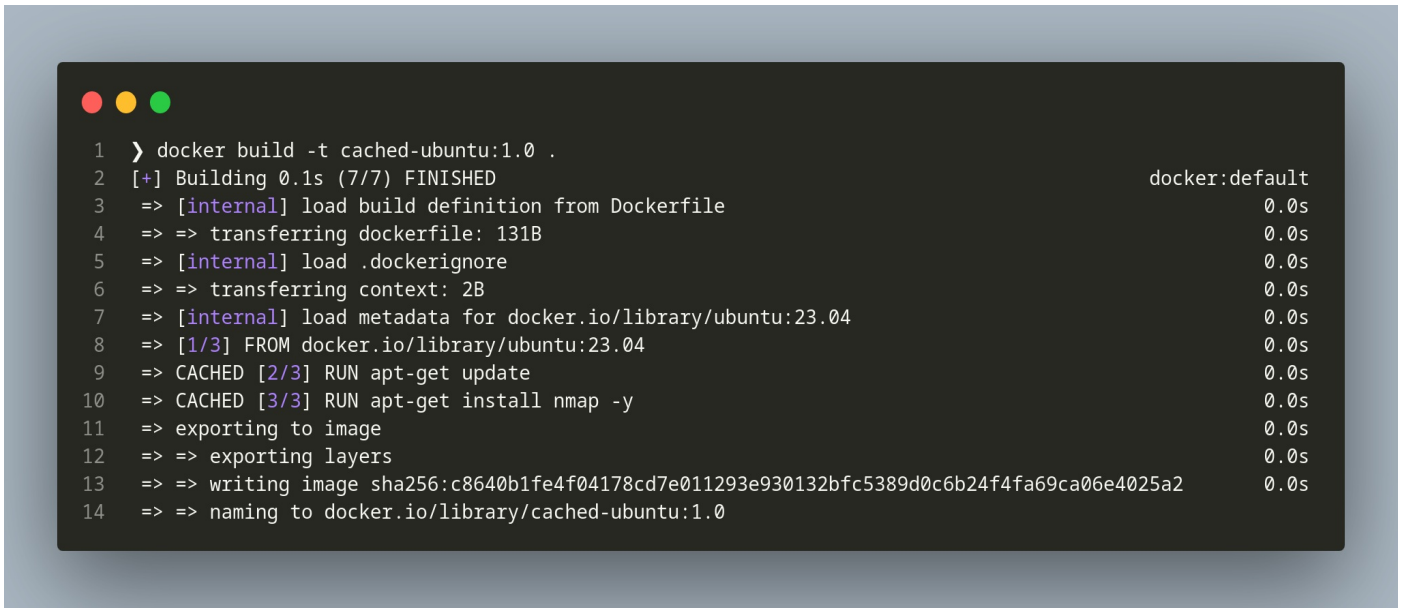
Technique 22 : Busting the cache (Comparison of build with and without caching)

STEPS

1. Creating a Dockerfile that pulls some image that has already been pulled previously and update the container

2. Build the Dockerfile

```
docker build -t cached-ubuntu:1.0 .
```



The above image was built in 0.1s, it cached previously ran update and install nmap

3. Now if there is a need to build an image without cache, it can be achieved by using –no-cache argument

```
docker build --no-cache -t cached-ubuntu:2.0 .
```

```
❯ docker build --no-cache -t cached-ubuntu:2.0 .
[+] Building 26.2s (7/7) FINISHED                                                docker:default
 => [internal] load build definition from Dockerfile                                      0.0s
 => => transferring dockerfile: 131B                                                      0.0s
 => [internal] load .dockerignore                                                         0.0s
 => => transferring context: 2B                                                           0.0s
 => [internal] load metadata for docker.io/library/ubuntu:23.04                           0.0s
 => CACHED [1/3] FROM docker.io/library/ubuntu:23.04                                      0.0s
 => [2/3] RUN apt-get update                                                             14.5s
 => [3/3] RUN apt-get install nmap -y                                                    10.6s
 => exporting to image                                                                    1.0s
 => => exporting layers                                                                   1.0s
 => => writing image sha256:ba1f083304b2c43c6680560420efe591689b2fbe17730a2e790e7d64544c33ad  0.0s
 => => naming to docker.io/library/cached-ubuntu:2.0                                      0.0s
```

The image took a while to build because it didn't cache update and install commands from RUN command

Teqnique 23 : Intelligent Cache-Busting using Build-Args

STEPS

1. Creating a Dockerfile that has build arguments to bust the cache

```
FROM ubuntu:23.04
RUN apt-get update && apt-get -y upgrade

# Custom cache invalidation
ARG CACHEBUST=1
# Welcome Text
RUN echo ["Hello...."]
```

2. Build the image with cache bust as an argument to set different value making all the layers to rebuild.

   `docker build -t ubuntu-test:2.0 --build-arg CACHEBUST=$(date +%s) .`

   `$(date +%s)` is used to ensure that each time the build command is run, unique value is generated based on date and the Image is rebuilt compulsorily

```
❯ docker build -t ubuntu-test:2.0 --build-arg CACHEBUST=$(date +%s) .
[+] Building 19.4s (7/7) FINISHED                                                                docker:default
 => [internal] load build definition from Dockerfile                                                      0.0s
 => => transferring dockerfile: 180B                                                                      0.0s
 => [internal] load .dockerignore                                                                         0.0s
 => => transferring context: 2B                                                                           0.0s
 => [internal] load metadata for docker.io/library/ubuntu:23.10                                           0.9s
 => [1/3] FROM docker.io/library/ubuntu:23.10@sha256:4ece736cc64e12426819d31e82f17f81555148adc309         5.3s
 => => resolve docker.io/library/ubuntu:23.10@sha256:4ece736cc64e12426819d31e82f17f81555148adc309         0.0s
 => => sha256:4ece736cc64e12426819d31e82f17f81555148adc3093474908ca1df00222d62 1.13kB / 1.13kB           0.0s
 => => sha256:fe7c495a4f1d2e1d86a9644ea5963370674a9894e273e34e96896bcde63441ce 424B / 424B               0.0s
 => => sha256:1909bc6bc56b0a4ba4a50a33b4c27844fcab405b08eeb26b754ff4f1fcc36471 2.30kB / 2.30kB           0.0s
 => => sha256:dd9cd860ef0ae602a40fdac6afd89d42f87a1a93dd70a90c07ae53f12f5fb022 27.10MB / 27.10MB         2.6s
 => => extracting sha256:dd9cd860ef0ae602a40fdac6afd89d42f87a1a93dd70a90c07ae53f12f5fb022                 2.6s
 => [2/3] RUN apt-get update && apt-get -y upgrade                                                       12.2s
 => [3/3] RUN echo ["Hello...."]                                                                          0.5s
 => exporting to image                                                                                    0.5s
 => => exporting layers                                                                                   0.5s
 => => writing image sha256:5ec425489289569d578432d0edc495663911dde3d3be17f4e959ce043806c6e0              0.0s
 => => naming to docker.io/library/ubuntu-test:2.0                                                        0.0s
```

Teqnique 24 : Intelligent Cache-Busting using ADD directive

STEPS

1. Creating a Dockerfile with an update command and a ADD instruction to download a file to a directory

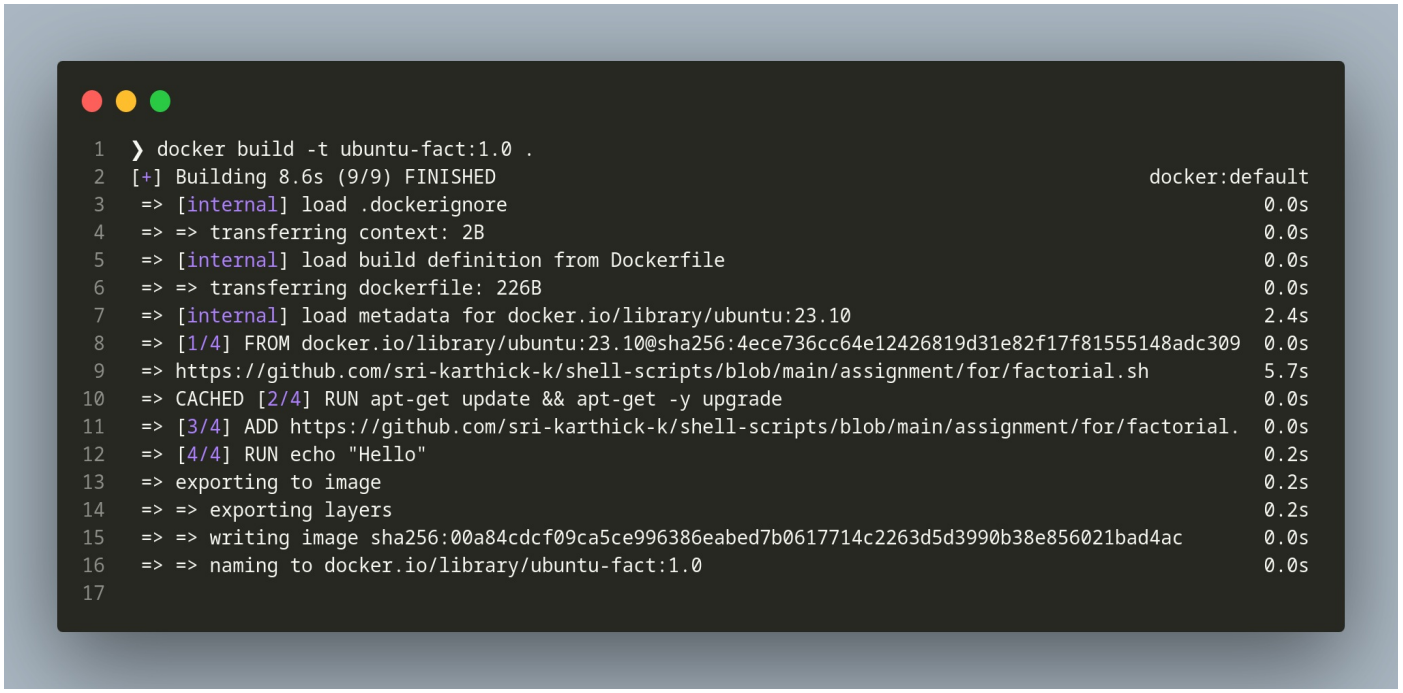
   `ADD <Online_Link /location/to_be_downloaded_to/`

```
FROM ubuntu:23.10
RUN apt-get update && apt-get -y upgrade

ADD https://github.com/sri-karthick-k/shell-scripts/blob/main/assignment/for/factorial.sh /home/devp/projects/
RUN echo "Hello"
```

2. Build image from the Dockerfile

   `docker build -t ubuntu-fact:1.0 .`



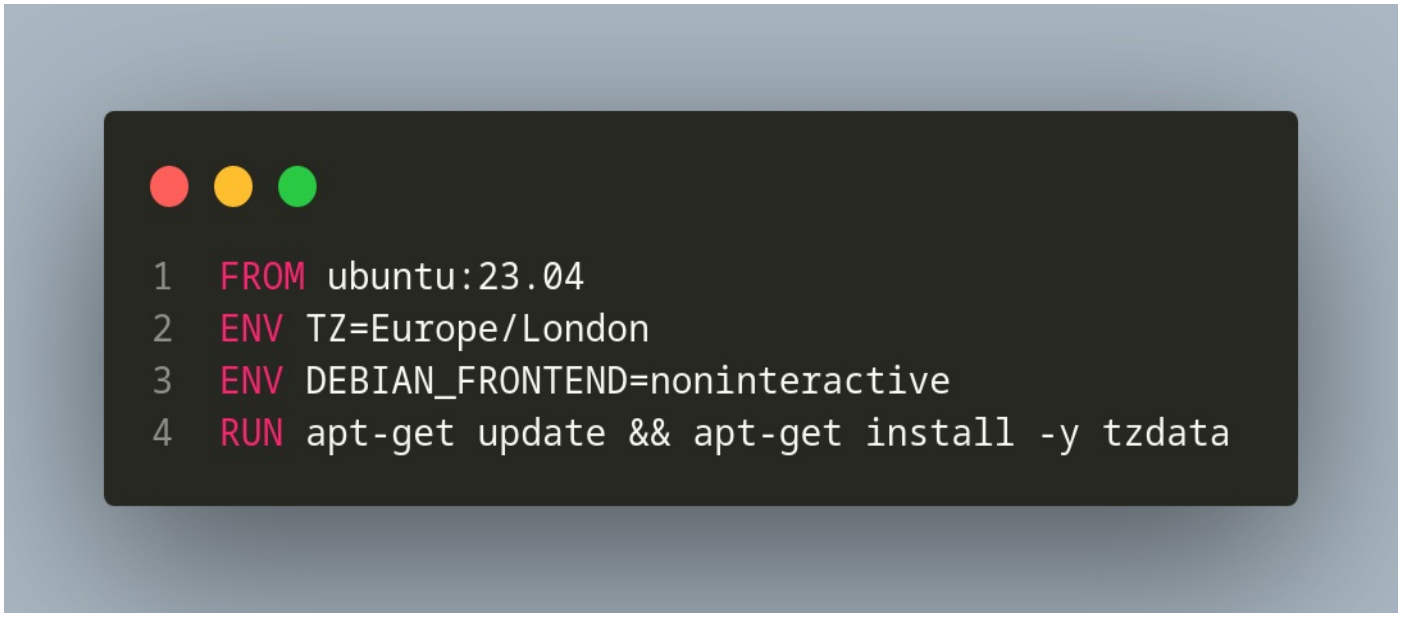
3. Listing the images

`docker images`

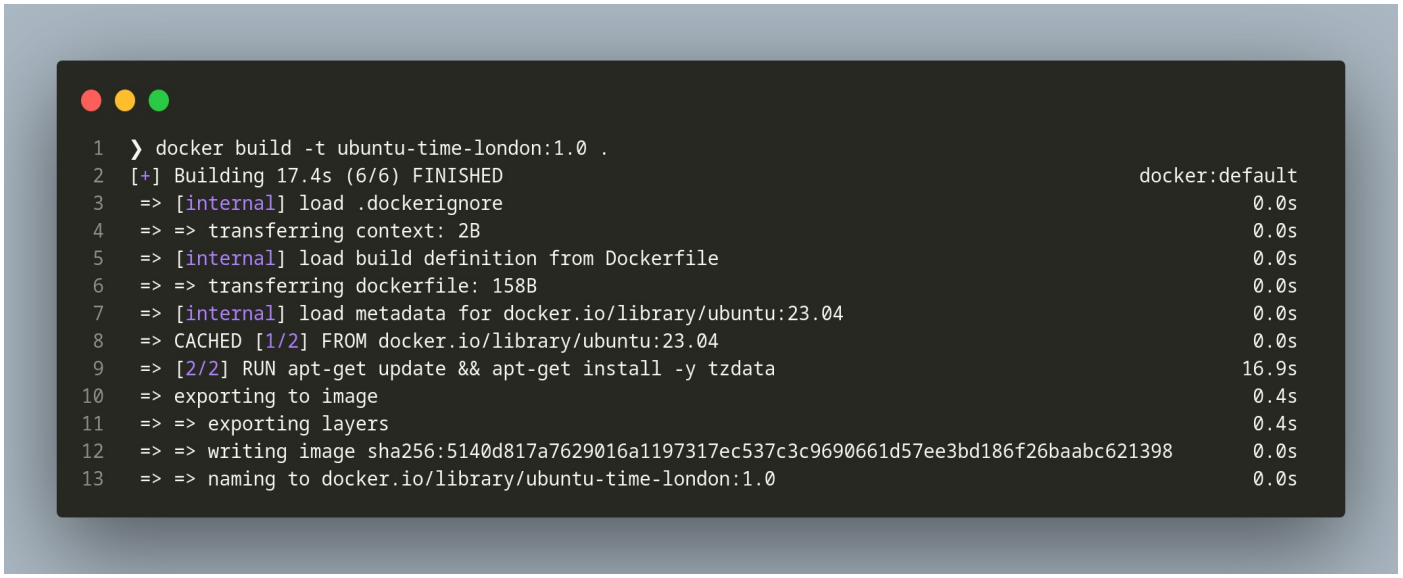
Teqnique 25 : Setting the Right Time Zone in the Containers

STEPS

1. Create a Dockerfile with an Environment



2. Build the Dockerfile `docker build -t ubuntu-time-london:1.0 .`

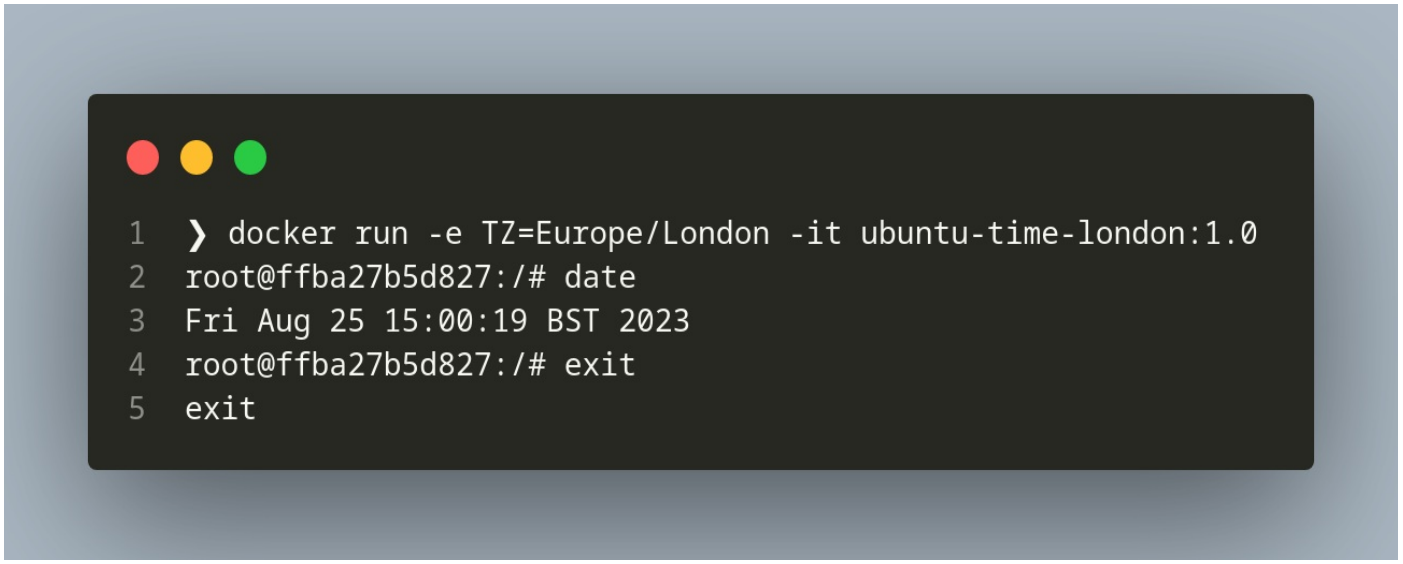


```
❯ docker build -t ubuntu-time-london:1.0 .
[+] Building 17.4s (6/6) FINISHED                                                docker:default
 => [internal] load .dockerignore                                                          0.0s
 => => transferring context: 2B                                                            0.0s
 => [internal] load build definition from Dockerfile                                       0.0s
 => => transferring dockerfile: 158B                                                       0.0s
 => [internal] load metadata for docker.io/library/ubuntu:23.04                            0.0s
 => CACHED [1/2] FROM docker.io/library/ubuntu:23.04                                       0.0s
 => [2/2] RUN apt-get update && apt-get install -y tzdata                                 16.9s
 => exporting to image                                                                     0.4s
 => => exporting layers                                                                    0.4s
 => => writing image sha256:5140d817a7629016a1197317ec537c3c9690661d57ee3bd186f26baabc621398   0.0s
 => => naming to docker.io/library/ubuntu-time-london:1.0                                  0.0s
```

3. Run the dockerfile with the command `docker run -e TZ=Europe/London -it ubuntu-time-london:1.0`

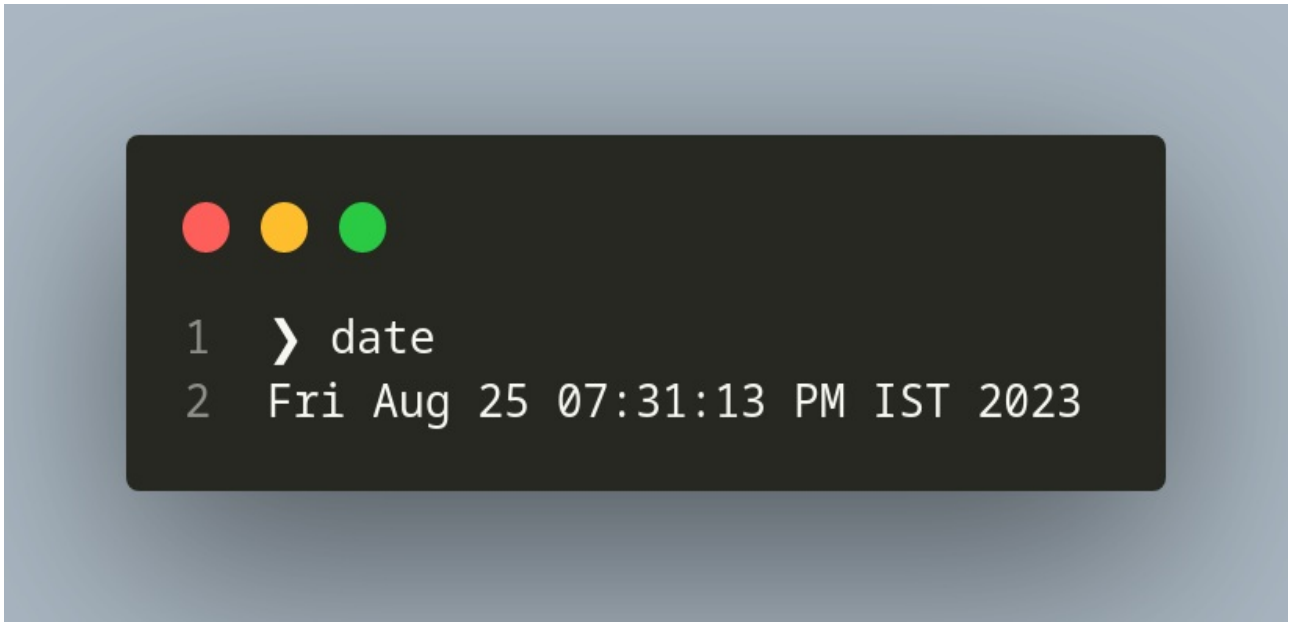
4. Check the date in the interactive mode date

5. Exit the container exit



```
❯ docker run -e TZ=Europe/London -it ubuntu-time-london:1.0
root@ffba27b5d827:/# date
Fri Aug 25 15:00:19 BST 2023
root@ffba27b5d827:/# exit
exit
```

6. Checking Local machine's and Date



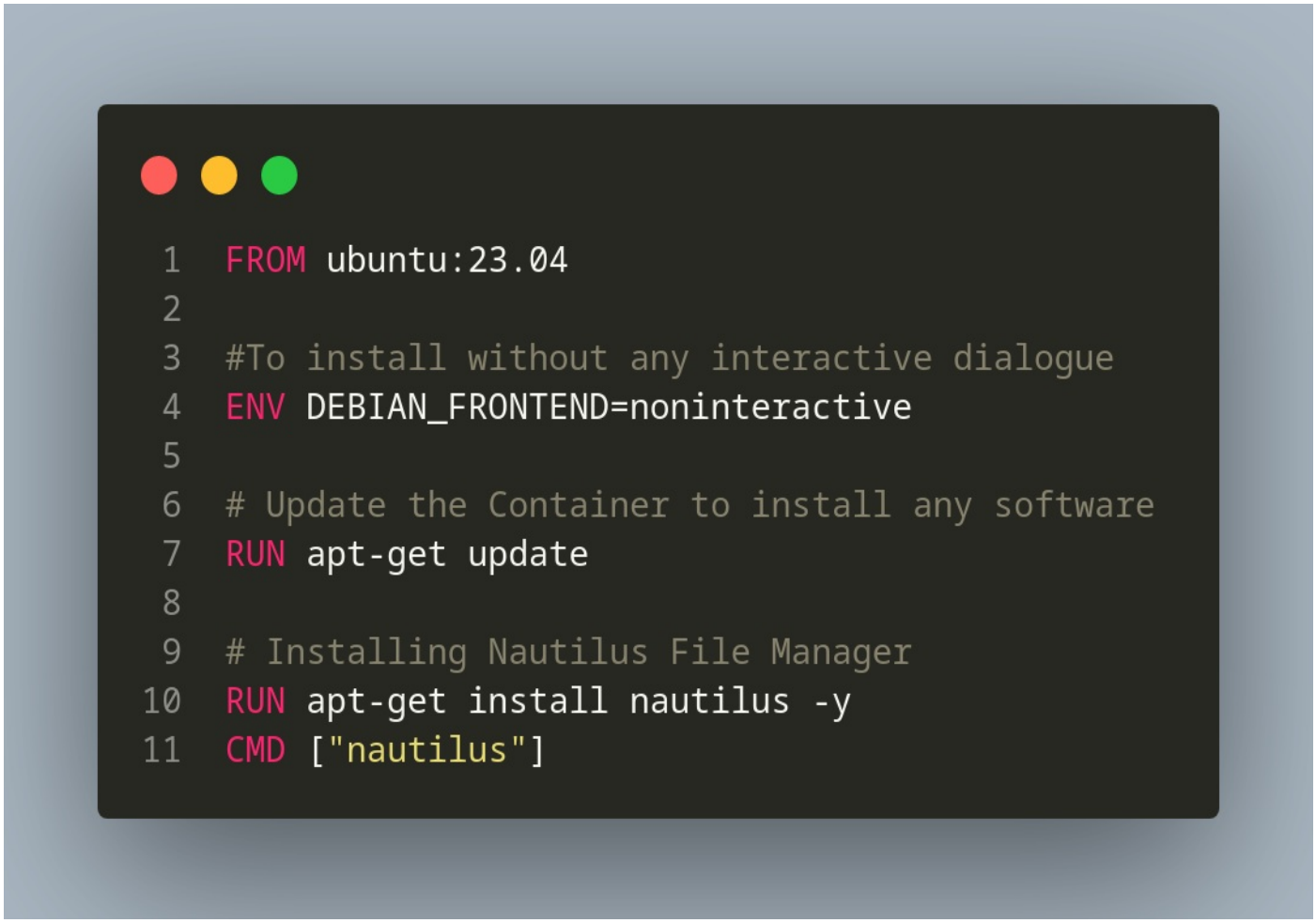
```
❯ date
Fri Aug 25 07:31:13 PM IST 2023
```

The Environment is running with the Time zone of London while the Local Machine is running with the Time Zone of Kolkata

Technique 29 : Running GUIs within Docker

STEPS

1. Create a Dockerfile with an Environment



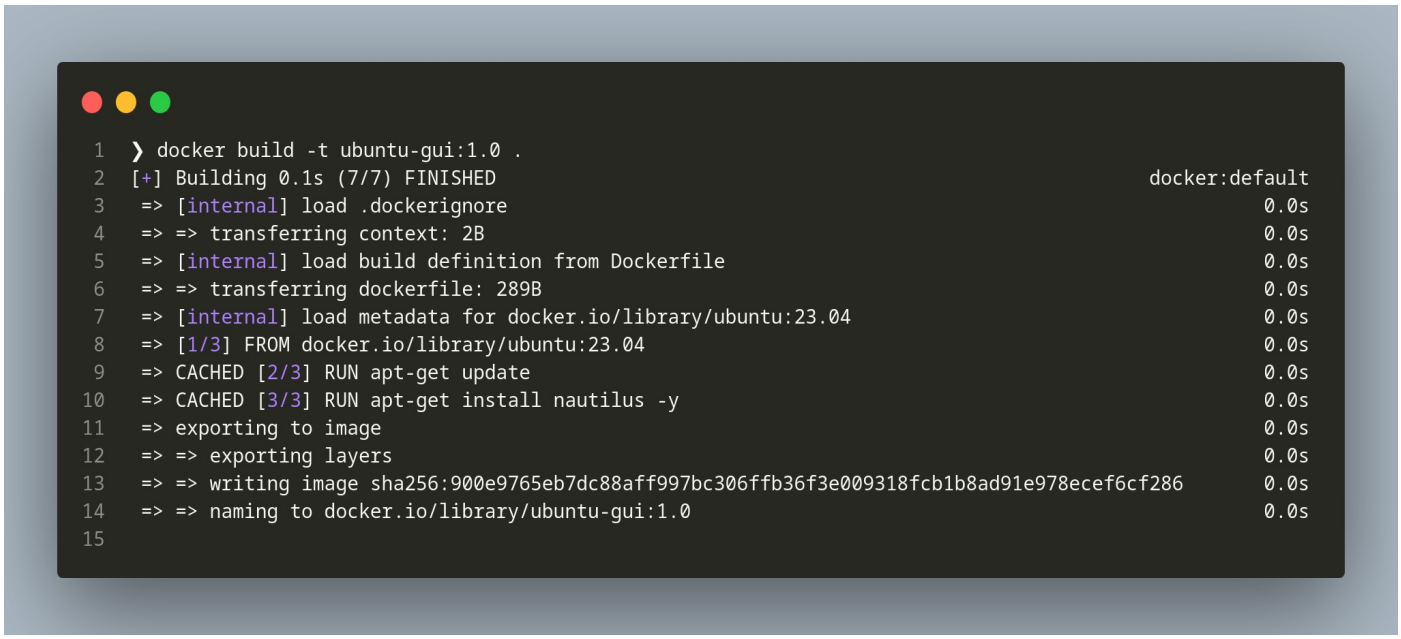
```
FROM ubuntu:23.04

#To install without any interactive dialogue
ENV DEBIAN_FRONTEND=noninteractive

# Update the Container to install any software
RUN apt-get update

# Installing Nautilus File Manager
RUN apt-get install nautilus -y
CMD ["nautilus"]
```

2. Build the image docker build -t ubuntu-gui:1.0 .



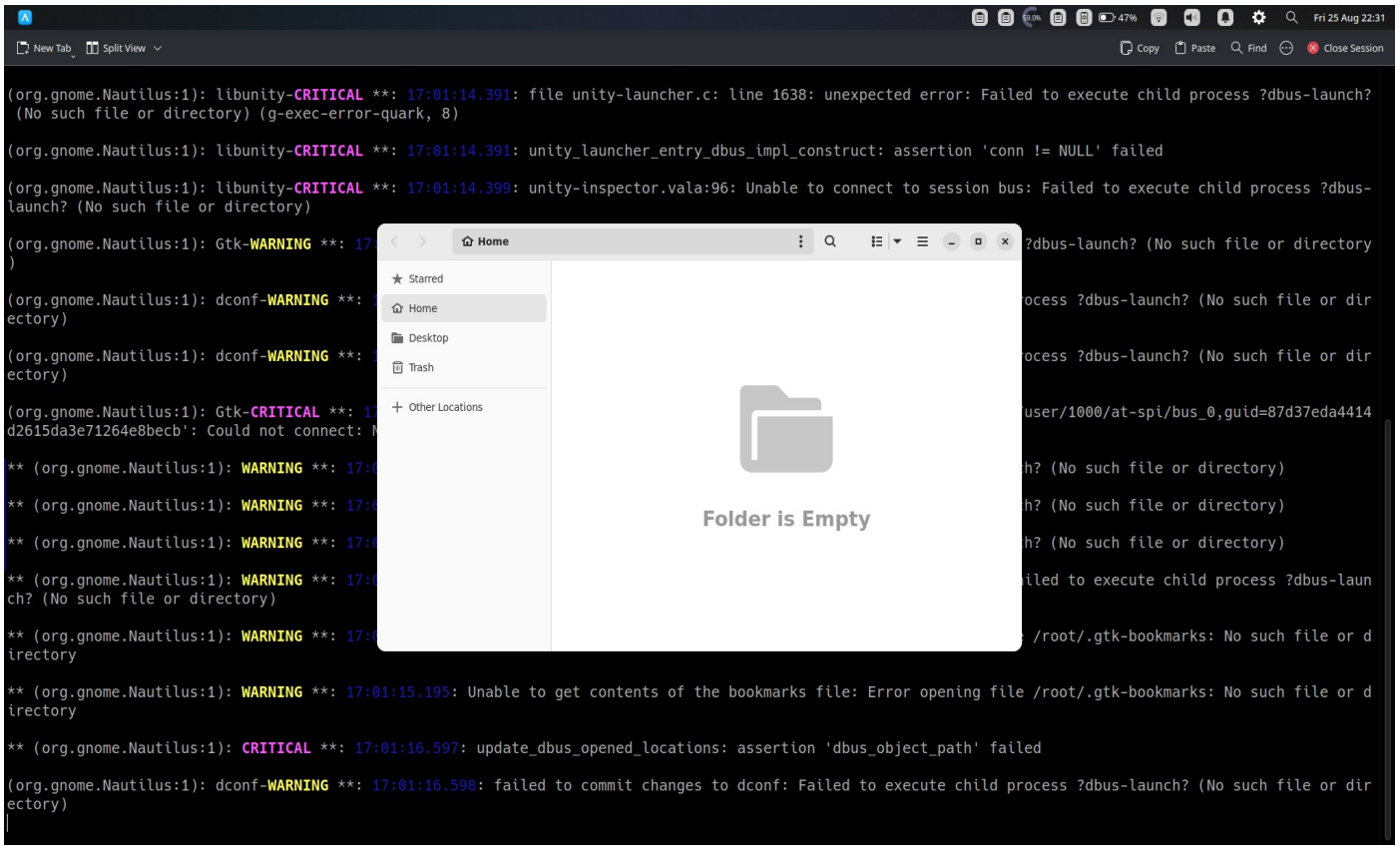
```
❯ docker build -t ubuntu-gui:1.0 .
[+] Building 0.1s (7/7) FINISHED                                                docker:default
 => [internal] load .dockerignore                                                         0.0s
 => => transferring context: 2B                                                           0.0s
 => [internal] load build definition from Dockerfile                                      0.0s
 => => transferring dockerfile: 289B                                                      0.0s
 => [internal] load metadata for docker.io/library/ubuntu:23.04                           0.0s
 => [1/3] FROM docker.io/library/ubuntu:23.04                                             0.0s
 => CACHED [2/3] RUN apt-get update                                                       0.0s
 => CACHED [3/3] RUN apt-get install nautilus -y                                          0.0s
 => exporting to image                                                                    0.0s
 => => exporting layers                                                                   0.0s
 => => writing image sha256:900e9765eb7dc88aff997bc306ffb36f3e009318fcb1b8ad91e978ecef6cf286   0.0s
 => => naming to docker.io/library/ubuntu-gui:1.0                                         0.0s

```

3. Allow docker to access X Server to run GUI applications from docker container

```
xhost +local:docker
```

4. Run the image with the command `docker run -it --rm -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix ubuntu-gui:1.0`



Technique 30 : Inspecting Containers

STEPS

1. Run a container

   `docker run -it ubuntu-gui:1.0 bash`

2. Open another terminal and retrieve the container ID

   `docker ps`

3. Inspect the container

   `docker inspect --size 4e1`

Technique 31 : Cleanly Killing the containers

STEPS

1. Run a container

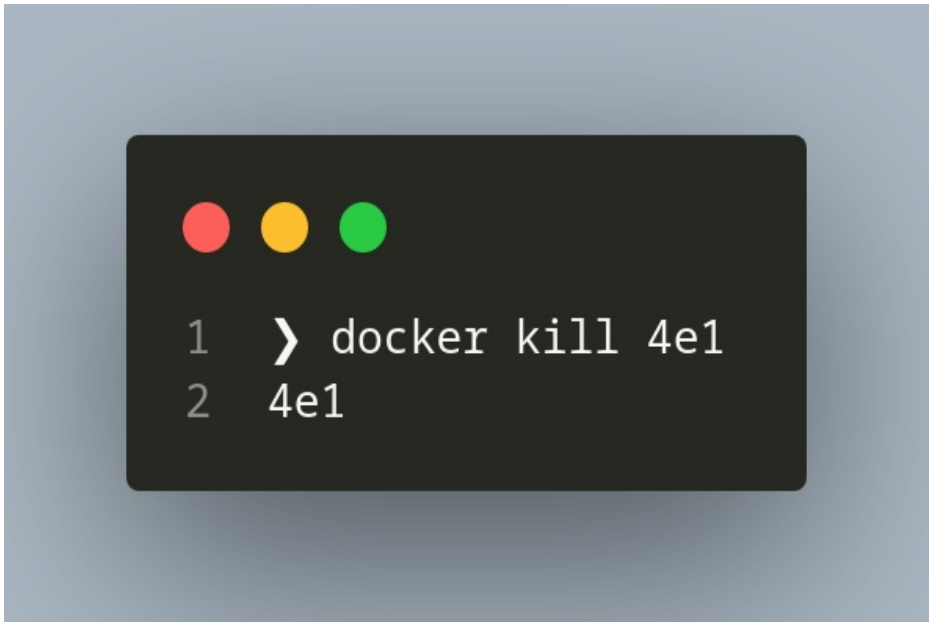
   `docker run -it ubuntu-gui:1.0 bash`

2. Open another terminal and retrieve the Container ID

   `docker ps`

3. Kill the container

   `docker kill 4e1`



The interactive terminal would have been exited and the docker container is killed, which can be verified by using `docker ps` command which will not list that container as running

Technique 32 : Using Docker Machine to provision Docker Hosts

STEPS

1. Install Docker Machine https://github.com/docker/machine/releases
2. Check the version docker-machine –version
3. Using docker-machine to run docker hosts docker-machine ls
4. Create a Docker VM using the command docker-machine create
5. Check the Virtual Box to verify the creation of VM
6. Verify the docker-machine to run docker hosts docker-machine ls

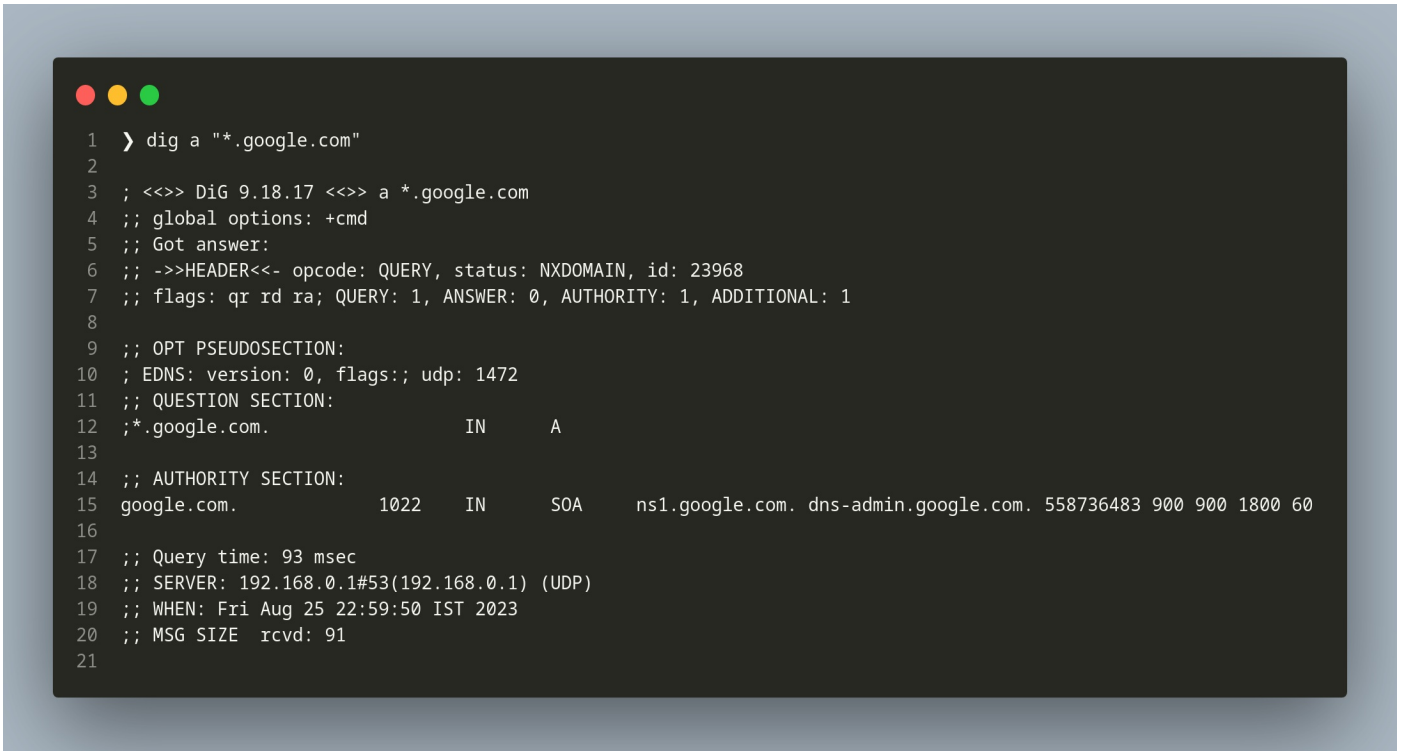
Technique 33 : Wildcard DNS

STEPS

Usage of static IP is recommended to differentiate different containers. Normally the containers take up ip-address from 172.18.0.1. If the order in which the container is run each time is differing, then each container will get different ip address differently. So, to avoid this conflict assigning static ip and referring to ip by using DNS is a recommended practice

1. To lookup the ip-address

```
dig a "*.google.com
```



Technique 61 : Using DockerHub workflow

STEPS

1. Create your repository on GitHub or BitBucket.
2. Clone the new Git repository.
3. Add code to your Git repository.
4. Commit the source.
5. Push the Git repository.
6. Create a new repository on the Docker Hub.
7. Link the Docker Hub repository to the Git repository.
8. Wait for the Docker Hub build to complete.
9. Commit and push a change to the source.
10. Wait for the second Docker Hub build to complete.

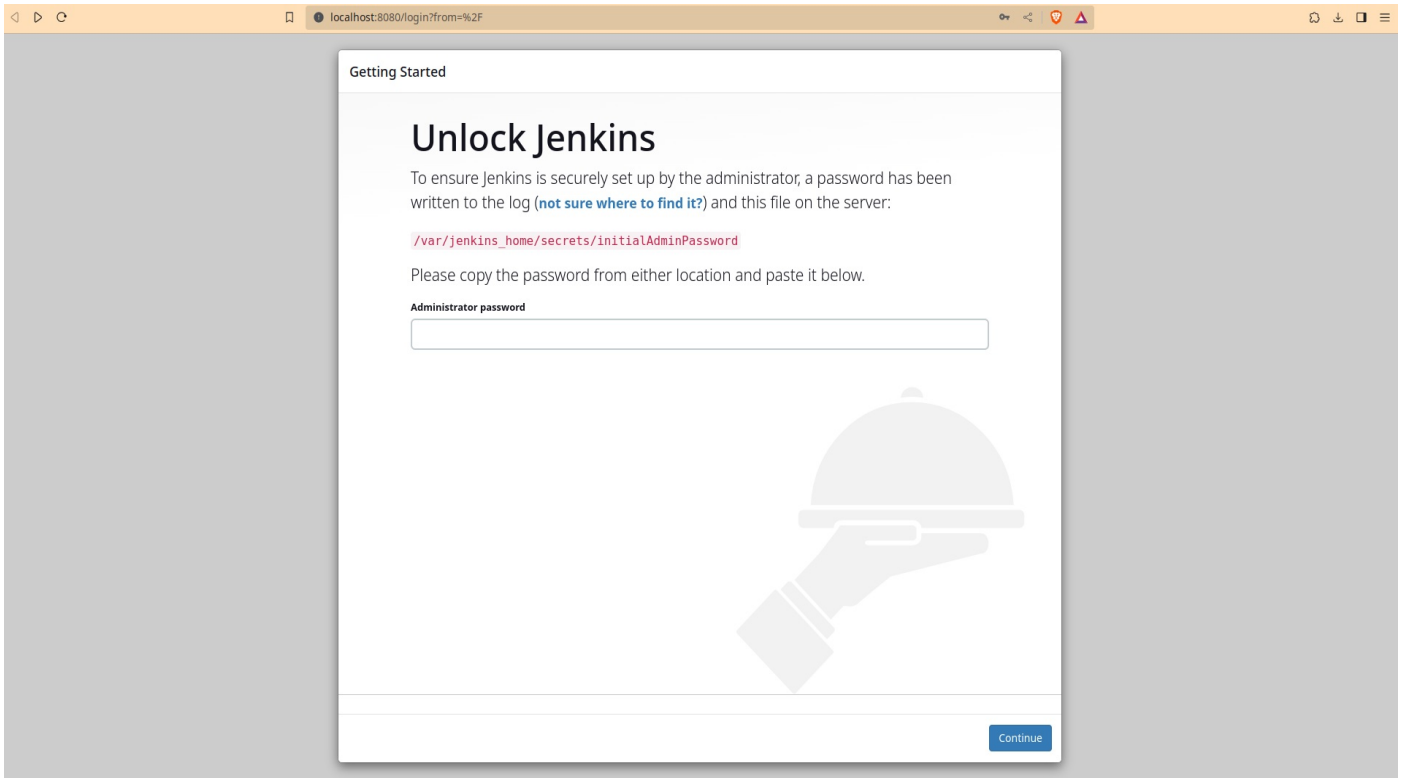
Technique 66 : Running the Jenkins Master within a Docker Container

STEPS

1. Use the Jenkins Docker Image to install and use Jenkins Server

   ```
   docker run -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts-jdk11
   ```

2. View running Containers `docker ps`
3. Navigate to the link http://localhost:8080 to verify the installation of Jenkins http://localhost:8080



Technique 70 : The Docker contract: Reducing friction

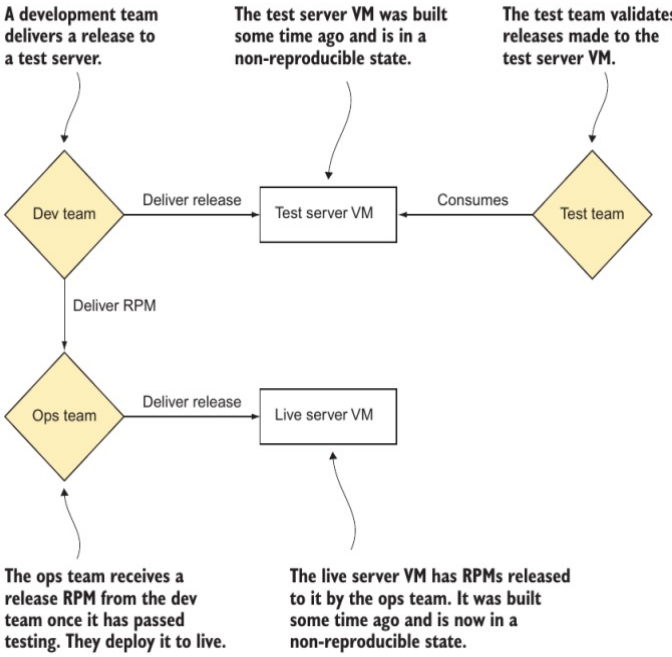


Figure 9.2 Before: a typical software workflow

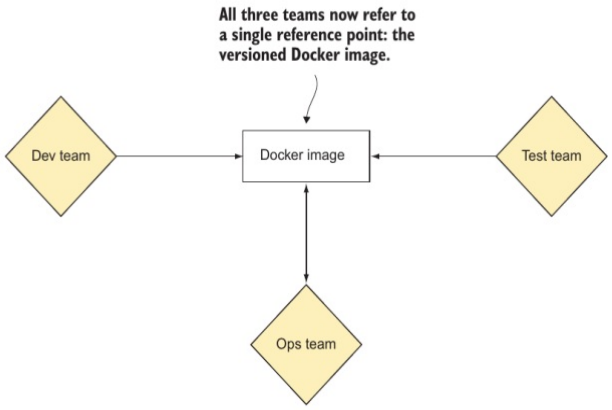


Figure 9.3 After: the Docker contract

Technique 71 : Manually mirroring registry images

1. Pull the image from the hub

   ```
   docker pull httpd
   ```

2. View Images

   ```
   docker images
   ```

3. Retag the image

   ```
   docker tag httpd:latest srikarthickk/httpd:1.0
   ```

4. Login to docker hub in terminal using,

   `sudo docker login`

5. Push to the docker hub

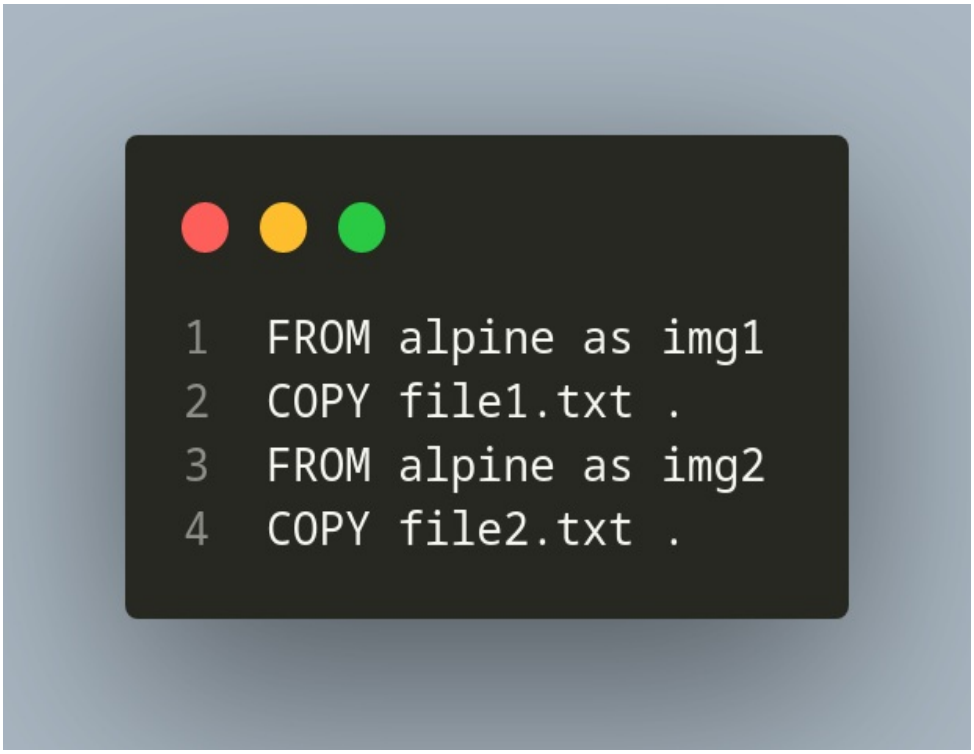
   `sudo docker push srikarthickk/httpd:1.0`

Technique 72 : Delivering Images Over Constrained Connections

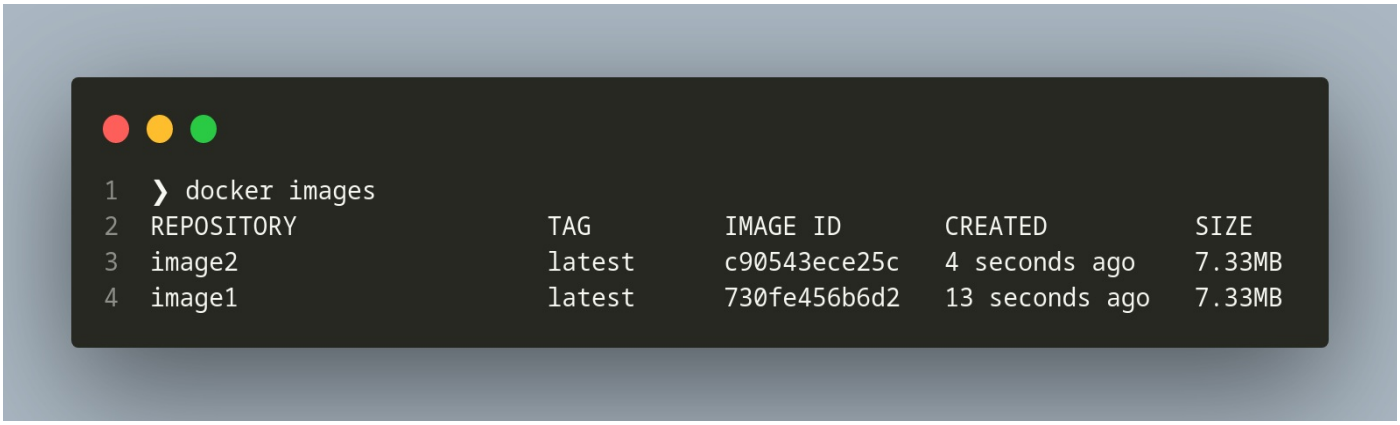
STEPS

1.

   Create a Dockerfile with multiple images



2. Build the images respectively `docker build -t image1 --target img1 .` `docker build -t image2 --target img2 .`

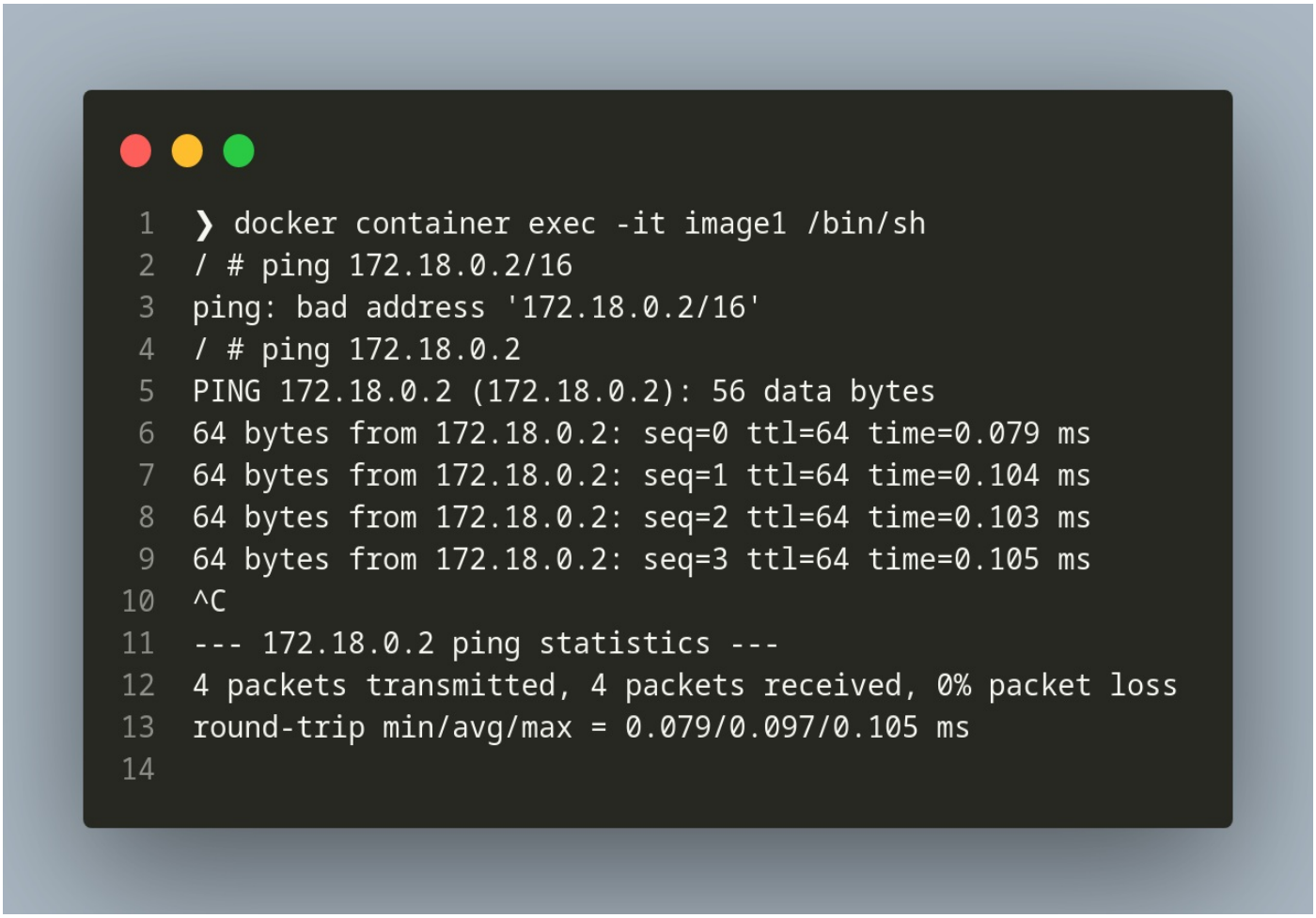


3. Run the images in the detached mode

   `docker run -it --name image1 -d image1:latest` `docker run -it --name image2 -d image2:latest`

4. Inspect the bridge in the network to get the IP addresses `docker network inspect bridge`

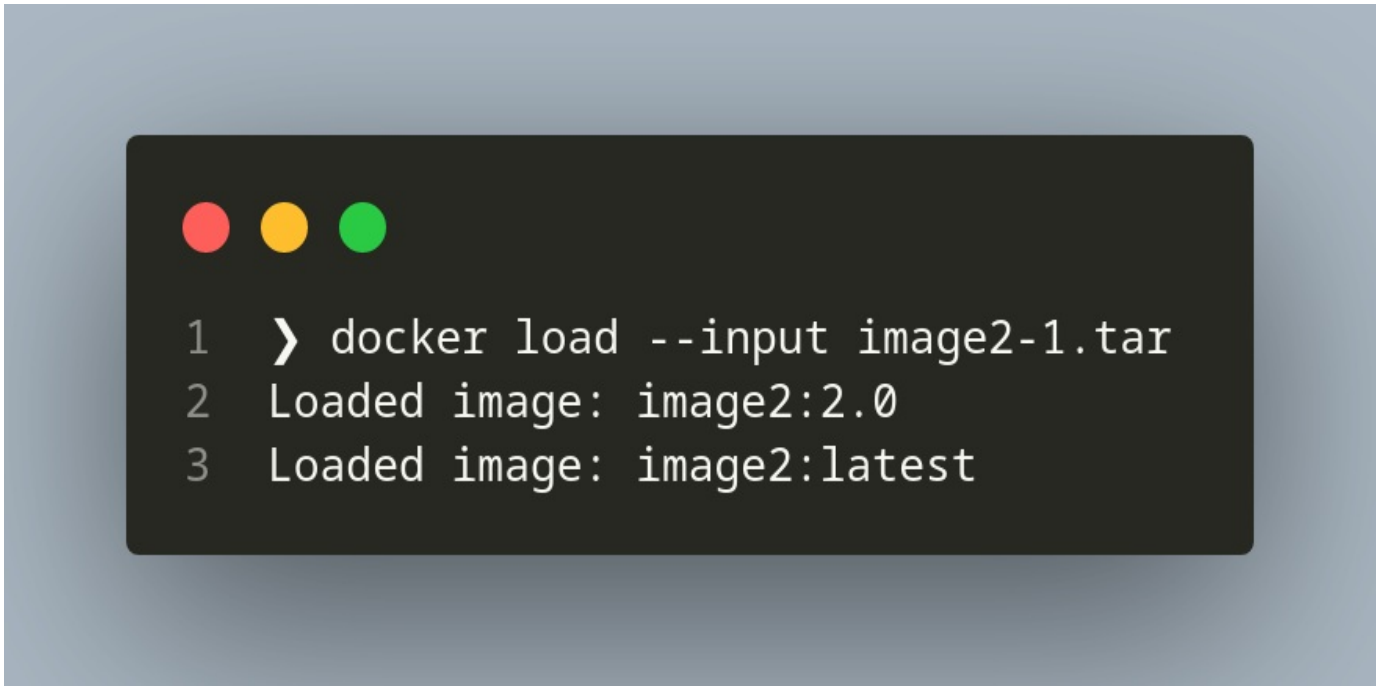
5. Execute a container and ping the other `docker container exec -it image1 /bin/sh`



Technique 73 : Sharing Docker objects as TAR files

STEPS

1. Export the Container with the Container name or ID `docker export edae > image2.tar`
2. Import the created tar file as a new Image `docker import - image2:2.0 < image2.tar`
3. List all the images `docker images`
4. Save the created image as a new Tar File `docker save -o image2-1.tar image2:2.0`
5. Load the Tar File `docker load --input image2-1.tar`



Technique 74 : Informing your containers with etcd

STEPS

1. Check the running processes of Docker

   `docker ps`

2. Pull and run the etcd image to create an Etcd Container with TLS support and etcdctl

```
docker run --rm -d -P --name etcd elcolio/etcd
```

1. Verify the running processes of Docker `docker ps`
2. List all the exposed ports by etcd

   `docker port etcd`

3. Run the command to set a key value pair

   `docker exec etcd etcdctl set my_name SRI_KARTHICK`

4. Retrieve a key value pair

   `docker exec etcd etcdctl get my_name`