

# CIE 2 TECHNIQUES

## Technique 15 : Save game

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

### STEPS

1. List all the images docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu-fritzing	latest	1243191ea04f	13 days ago	720MB
akshar	latest	a8c311962b58	3 weeks ago	185MB
wordpress	latest	b94fa4376c73	3 weeks ago	664MB
nginx	latest	89da1fb6dcb9	3 weeks ago	187MB
httpd	latest	96a2d0570deb	3 weeks ago	168MB
srikarthickk/httpd-clone	latest	96a2d0570deb	3 weeks ago	168MB
jenkins/jenkins	lts-jdk11	a40a8916af1d	3 weeks ago	471MB
tomcat	latest	7ba61facbe26	3 weeks ago	425MB
mysql	latest	7c5ae0d3388c	4 weeks ago	577MB
ubuntu	23.04	1ed313b0551f	2 months ago	70.3MB
registry	2	4bb5ea59f8e0	2 months ago	24MB
thingsboard/tb-postgres	latest	02771207004e	2 months ago	1.27GB
tutum/wordpress	latest	7e7f97a602ff	7 years ago	476MB



- 2.

View containers docker container ls -a

docker container ls -a						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a7db423481bb	thingsboard/tb-postgres	"start-tb.sh"	8 days ago	Created		sri-mytb-1



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

```

Konsole File Edit View Bookmarks Plugins Settings Help
New Tab Split View
> docker run -it ubuntu:23.04
root@6b1835f03725:/# apt update
Get:1 http://security.ubuntu.com/ubuntu lunar-security InRelease [109 kB]
Get:2 http://archive.ubuntu.com/ubuntu lunar InRelease [267 kB]
Get:3 http://security.ubuntu.com/ubuntu lunar-security/main amd64 Packages [350 kB]
Get:4 http://security.ubuntu.com/ubuntu lunar-security/restricted amd64 Packages [357 kB]
Get:5 http://security.ubuntu.com/ubuntu lunar-security/universe amd64 Packages [890 kB]
Get:6 http://archive.ubuntu.com/ubuntu lunar-updates InRelease [109 kB]
Get:7 http://archive.ubuntu.com/ubuntu lunar-security/multiverse amd64 Packages [7179 B]
Get:8 http://archive.ubuntu.com/ubuntu lunar-backports InRelease [99.8 kB]
Get:9 http://archive.ubuntu.com/ubuntu lunar/restricted amd64 Packages [181 kB]
Get:10 http://archive.ubuntu.com/ubuntu lunar/multiverse amd64 Packages [289 kB]
Get:11 http://archive.ubuntu.com/ubuntu lunar/main amd64 Packages [1797 kB]
Get:12 http://archive.ubuntu.com/ubuntu lunar/universe amd64 Packages [18.7 MB]
Get:13 http://archive.ubuntu.com/ubuntu lunar-updates/multiverse amd64 Packages [7179 B]
Get:14 http://archive.ubuntu.com/ubuntu lunar-updates/restricted amd64 Packages [357 kB]
Get:15 http://archive.ubuntu.com/ubuntu lunar-updates/universe amd64 Packages [949 kB]
Get:16 http://archive.ubuntu.com/ubuntu lunar-updates/main amd64 Packages [460 kB]
Get:17 http://archive.ubuntu.com/ubuntu lunar-backports/universe amd64 Packages [4192 B]
Fetched 24.9 MB in 15s (1609 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
5 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@6b1835f03725:/# apt install net-tools
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
net-tools
0 upgraded, 1 newly installed, 0 to remove and 5 not upgraded.
Need to get 204 kB of archives.
After this operation, 815 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu lunar/main amd64 net-tools amd64 2.10-0.1ubuntu3 [204 kB]
Fetched 204 kB in 2s (92.0 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package net-tools.
(Reading database ... 4353 files and directories currently installed.)
Preparing to unpack .../net-tools_2.10-0.1ubuntu3_amd64.deb ...
Unpacking net-tools (2.10-0.1ubuntu3) ...
Setting up net-tools (2.10-0.1ubuntu3) ...
root@6b1835f03725:/#

```

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

```

root@6b1835f03725:/# ifconfig --version
net-tools 2.10
root@6b1835f03725:/#

```

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

```

root@6b1835f03725:/# exit
exit
> docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS           PORTS     NAMES
6b1835f03725   ubuntu:23.04   "/bin/bash"   2 minutes ago  Exited (0) 4 seconds ago
a7db423481bb   thingsboard/tb-postgres "start-tb.sh" 8 days ago   Created
> docker commit 6b18 ubuntu-net-tools:latest
sha256:6e96881278768b44465bcf3602fb59e40c23f170930b2e00186568428f392c32
> docker images
REPOSITORY          TAG      IMAGE ID      CREATED      SIZE
ubuntu-net-tools    latest   6e9688127876   3 seconds ago  111MB
ubuntu-fritzng     latest   1243191ea04f   13 days ago   720MB

```

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

```

> docker run -it ubuntu-net-tools:latest
root@fcbb7d060bf5:/# ifconfig --version
net-tools 2.10
root@fcbb7d060bf5:/#

```

## 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

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu-fritzing	latest	1243191ea04f	13 days ago	720MB
akshar	latest	a8c311962b58	3 weeks ago	185MB
wordpress	latest	b94fa4376c73	3 weeks ago	664MB
nginx	latest	89da1fb6dc9	3 weeks ago	187MB
httpd	latest	96a2d0570deb	3 weeks ago	168MB
srikarthickk/httpd-clone	latest	96a2d0570deb	3 weeks ago	168MB
jenkins/jenkins	lts-jdk11	a40a8916af1d	3 weeks ago	471MB
tomcat	latest	7ba61facbe26	3 weeks ago	425MB
mysql	latest	7c5ae0d3388c	4 weeks ago	577MB
ubuntu	23.04	1ed313b0551f	2 months ago	70.3MB
registry	2	4bb5ea59f8e0	2 months ago	24MB
thingsboard/tb-postgres	latest	02771207004e	2 months ago	1.27GB
tutum/wordpress	latest	7e7f97a602ff	7 years ago	476MB



2. Specify the new name to the existing image docker tag <existing\_image\_id> <new\_image\_name>

3. Verify the tagged image

```
> docker tag 89da my-new-nginx
> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu-fritzing	latest	1243191ea04f	13 days ago	720MB
akshar	latest	a8c311962b58	3 weeks ago	185MB
wordpress	latest	b94fa4376c73	3 weeks ago	664MB
my-new-nginx	latest	89da1fb6dc9	3 weeks ago	187MB
nginx	latest	89da1fb6dc9	3 weeks ago	187MB

## 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 .

```
> tar -zcvf local-folder.tar.gz ~/Downloads/local-folder
tar: Removing leading `/' from member names
/home/sri/Downloads/local-folder/
/home/sri/Downloads/local-folder/hello.txt
> nano Dockerfile
> cat 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 .
```

```
1 docker build --no-cache .
2 [+] Building 27.6s (7/7) FINISHED
3 => [internal] load .dockerignore
4 => => transferring context: 2B
5 => [internal] load build definition from Dockerfile
6 => => transferring dockerfile: 131B
7 => [internal] load metadata for docker.io/library/ubuntu:23.04
8 => CACHED [1/3] FROM docker.io/library/ubuntu:23.04
9 => [2/3] RUN apt-get update
10 => [3/3] RUN apt-get install nmap -y
11 => exporting to image
12 => => exporting layers
13 => => writing image sha256:c8640b1fe4f04178cd7e011293e930132bfc5389d0c6b24f4fa69ca06e4025a2
```

	docker:default
0.0s	0.0s
15.6s	0.0s
11.1s	0.8s
0.8s	0.8s
0.0s	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



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

## 2. Build the Dockerfile

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



```
1 ➜ docker build -t cached-ubuntu:1.0 .
2 [+] Building 0.1s (7/7) FINISHED
3 => [internal] load build definition from Dockerfile
4 => => transferring dockerfile: 131B
5 => [internal] load .dockerignore
6 => => transferring context: 2B
7 => [internal] load metadata for docker.io/library/ubuntu:23.04
8 => [1/3] FROM docker.io/library/ubuntu:23.04
9 => CACHED [2/3] RUN apt-get update
10 => CACHED [3/3] RUN apt-get install nmap -y
11 => exporting to image
12 => => exporting layers
13 => => writing image sha256:c8640b1fe4f04178cd7e011293e930132bfc5389d0c6b24f4fa69ca06e4025a2
14 => => naming to docker.io/library/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 .
```

```
1  > docker build --no-cache -t cached-ubuntu:2.0 .
2  [+] Building 26.2s (7/7) FINISHED
3  => [internal] load build definition from Dockerfile
4  => => transferring dockerfile: 131B
5  => [internal] load .dockerignore
6  => => transferring context: 2B
7  => [internal] load metadata for docker.io/library/ubuntu:23.04
8  => CACHED [1/3] FROM docker.io/library/ubuntu:23.04
9  => [2/3] RUN apt-get update
10 => [3/3] RUN apt-get install nmap -y
11 => exporting to image
12 => => exporting layers
13 => => writing image sha256:ba1f083304b2c43c6680560420efe591689b2fbe17730a2e790e7d64544c33ad
14 => => naming to docker.io/library/cached-ubuntu:2.0
```

	docker:default
0.0s	0.0s
14.5s	14.5s
10.6s	10.6s
1.0s	1.0s
1.0s	1.0s
0.0s	0.0s
0.0s	0.0s

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

## Technique 23 : Intelligent Cache-Busting using Build-Args

### STEPS

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

```
1  FROM ubuntu:23.04
2  RUN apt-get update && apt-get -y upgrade
3
4  # Custom cache invalidation
5  ARG CACHEBUST=1
6  # Welcome Text
7  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

```
1  docker build -t ubuntu-test:2.0 --build-arg CACHEBUST=$(date +%s) .
2  [+] Building 19.4s (7/7) FINISHED                                            docker:default
3   => [internal] load build definition from Dockerfile                      0.0s
4   => => transferring dockerfile: 180B                                         0.0s
5   => [internal] load .dockerignore                                         0.0s
6   => => transferring context: 2B                                           0.0s
7   => [internal] load metadata for docker.io/library/ubuntu:23.10           0.9s
8   => [1/3] FROM docker.io/library/ubuntu:23.10@sha256:4ece736cc64e12426819d31e82f17f81555148adc309 5.3s
9   => => resolve docker.io/library/ubuntu:23.10@sha256:4ece736cc64e12426819d31e82f17f81555148adc309 0.0s
10  => => sha256:4ece736cc64e12426819d31e82f17f81555148adc3093474908ca1df00222d62 1.13kB / 1.13kB 0.0s
11  => => sha256:fe7c495a4f1d2e1d86a9644ea5963370674a9894e273e34e96896bcde63441ce 424B / 424B 0.0s
12  => => sha256:1909bc6bc56b0a4ba4a50a33b4c27844fcab405b08eeb26b754ff4f1fcc36471 2.30kB / 2.30kB 0.0s
13  => => sha256:dd9cd860ef0ae602a40fdac6af89d42f87a1a93dd70a90c07ae53f12f5fb022 27.10MB / 27.10MB 2.6s
14  => => extracting sha256:dd9cd860ef0ae602a40fdac6af89d42f87a1a93dd70a90c07ae53f12f5fb022 2.6s
15  => [2/3] RUN apt-get update && apt-get -y upgrade                         12.2s
16  => [3/3] RUN echo ["Hello...."]                                              0.5s
17  => exporting to image                                                       0.5s
18  => => exporting layers                                                       0.5s
19  => => writing image sha256:5ec425489289569d578432d0edc495663911dde3d3be17f4e959ce043806c6e0 0.0s
20  => => naming to docker.io/library/ubuntu-test:2.0                           0.0s
```

## Technique 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/
```

```
1  FROM ubuntu:23.10
2  RUN apt-get update && apt-get -y upgrade
3
4  ADD https://github.com/sri-karthick-k/shell-scripts/blob/main/assignment/for/factorial.sh /home/devp/projects/
5  RUN echo "Hello"
```

2. Build image from the Dockerfile

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

```
1 > docker build -t ubuntu-fact:1.0 .
2 [+] Building 8.6s (9/9) FINISHED
3 => [internal] load .dockerignore
4 => => transferring context: 2B
5 => [internal] load build definition from Dockerfile
6 => => transferring dockerfile: 226B
7 => [internal] load metadata for docker.io/library/ubuntu:23.10
8 => [1/4] FROM docker.io/library/ubuntu:23.10@sha256:4ece736cc64e12426819d31e82f17f81555148adc309
9 => https://github.com/sri-karthick-k/shell-scripts/blob/main/assignment/for/factorial.sh
10 => CACHED [2/4] RUN apt-get update && apt-get -y upgrade
11 => [3/4] ADD https://github.com/sri-karthick-k/shell-scripts/blob/main/assignment/for/factorial.
12 => [4/4] RUN echo "Hello"
13 => exporting to image
14 => => exporting layers
15 => => writing image sha256:00a84cdcf09ca5ce996386eabed7b0617714c2263d5d3990b38e856021bad4ac
16 => => naming to docker.io/library/ubuntu-fact:1.0
17
```

### 3. Listing the images

```
docker images
```

```
1 > docker images
2 REPOSITORY          TAG      IMAGE ID   CREATED    SIZE
3 ubuntu-fact         1.0      00a84cdcf09c  55 seconds ago  108MB
4
```

## Technique 25 : Setting the Right Time Zone in the Containers

### STEPS

1. Create a Dockerfile with an Environment

```
1 FROM ubuntu:23.04
2 ENV TZ=Europe/London
3 ENV DEBIAN_FRONTEND=noninteractive
4 RUN apt-get update && apt-get install -y tzdata
```

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

```
1 > docker build -t ubuntu-time-london:1.0 .
2 [+] Building 17.4s (6/6) FINISHED
3 => [internal] load .dockerignore
4 => => transferring context: 2B
5 => [internal] load build definition from Dockerfile
6 => => transferring dockerfile: 158B
7 => [internal] load metadata for docker.io/library/ubuntu:23.04
8 => CACHED [1/2] FROM docker.io/library/ubuntu:23.04
9 => [2/2] RUN apt-get update && apt-get install -y tzdata
10 => exporting to image
11 => => exporting layers
12 => => writing image sha256:5140d817a7629016a1197317ec537c3c9690661d57ee3bd186f26baabc621398
13 => => naming to docker.io/library/ubuntu-time-london:1.0
```

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

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

6. Checking Local machine's and Date

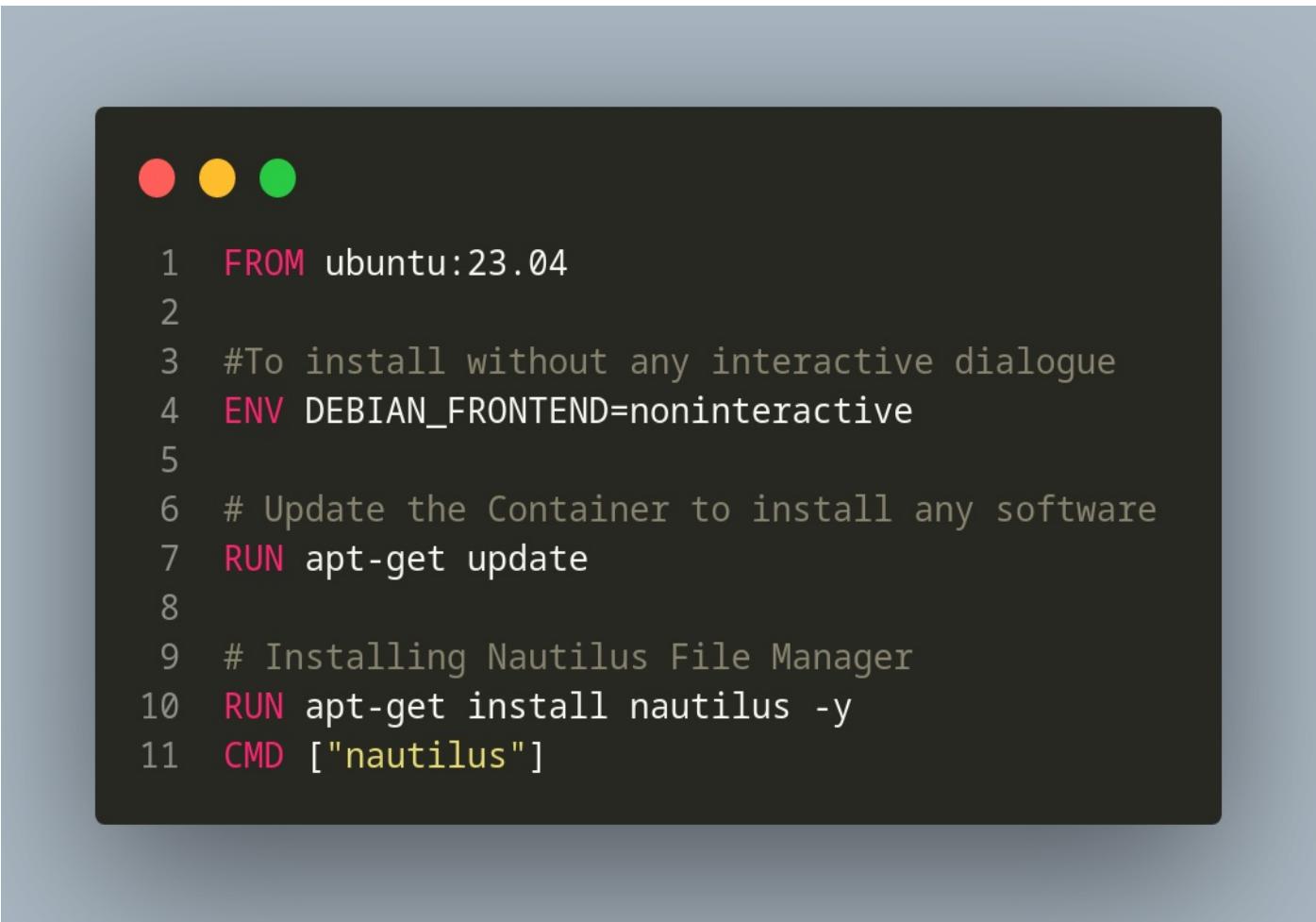
```
1 > date
2 Fri Aug 25 07:31:13 PM IST 2023
```

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

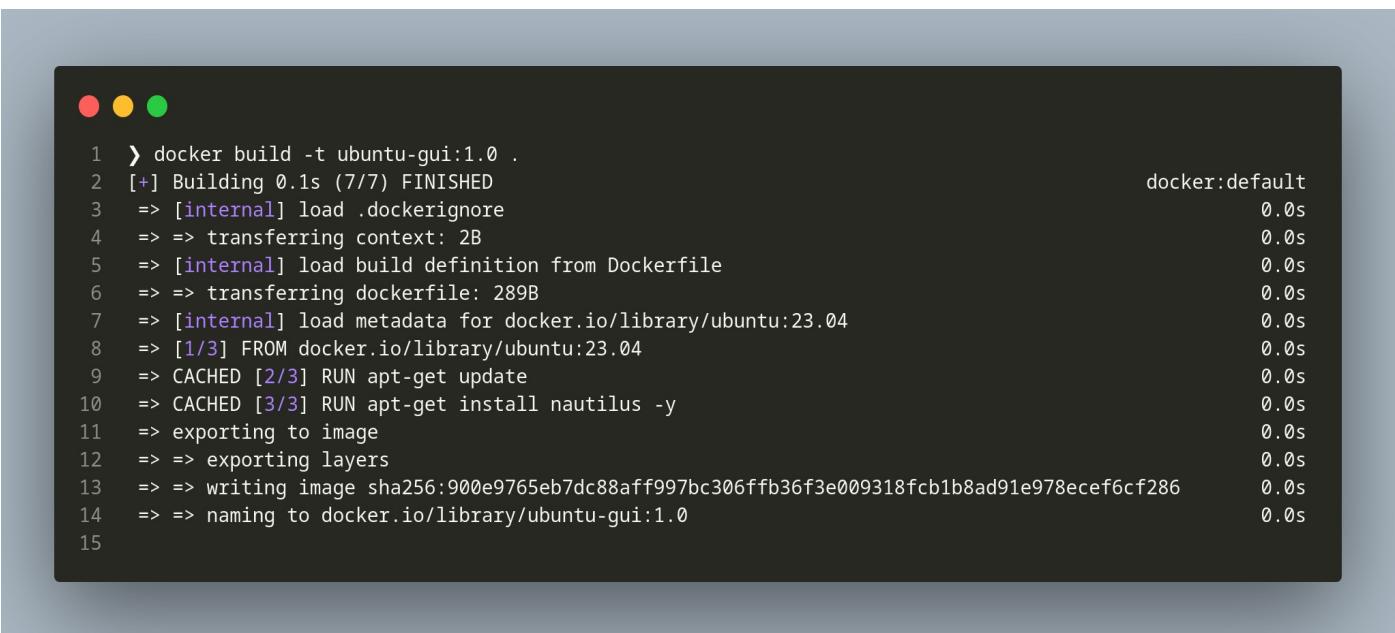
1. Create a Dockerfile with an Environment



A screenshot of a terminal window with three colored dots (red, yellow, green) at the top. The terminal displays a Dockerfile:

```
1 FROM ubuntu:23.04
2
3 #To install without any interactive dialogue
4 ENV DEBIAN_FRONTEND=noninteractive
5
6 # Update the Container to install any software
7 RUN apt-get update
8
9 # Installing Nautilus File Manager
10 RUN apt-get install nautilus -y
11 CMD ["nautilus"]
```

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



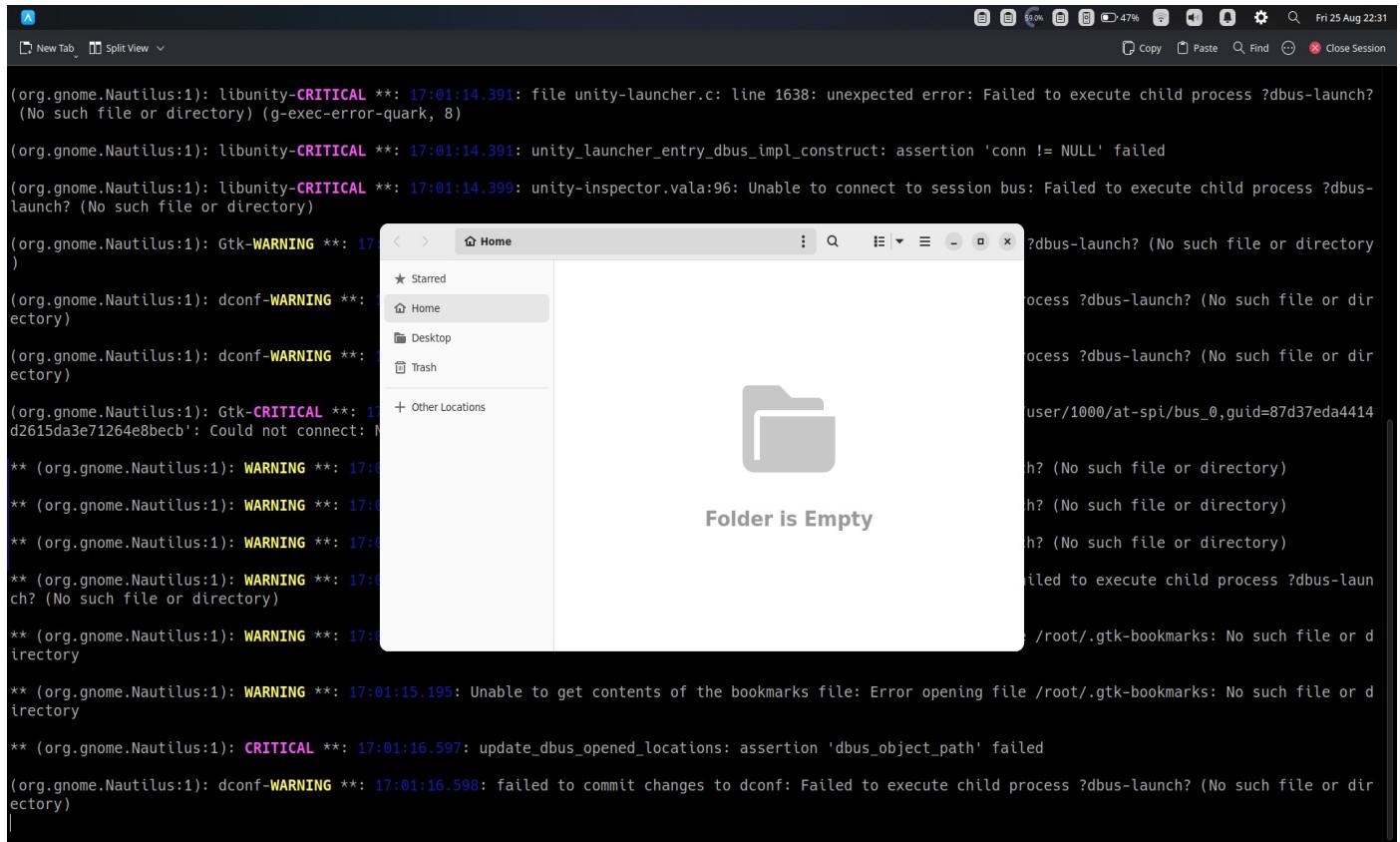
A screenshot of a terminal window with three colored dots (red, yellow, green) at the top. The terminal shows the output of a Docker build command:

```
1 ➜ docker build -t ubuntu-gui:1.0 .
2 [+] Building 0.1s (7/7) FINISHED
3 => [internal] load .dockerignore
4 => => transferring context: 2B
5 => [internal] load build definition from Dockerfile
6 => => transferring dockerfile: 289B
7 => [internal] load metadata for docker.io/library/ubuntu:23.04
8 => [1/3] FROM docker.io/library/ubuntu:23.04
9 => CACHED [2/3] RUN apt-get update
10 => CACHED [3/3] RUN apt-get install nautilus -y
11 => exporting to image
12 => => exporting layers
13 => => writing image sha256:900e9765eb7dc88aff997bc306ffb36f3e009318fcb1b8ad91e978ecef6cf286
14 => => naming to docker.io/library/ubuntu-gui:1.0
15
```

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

- Run a container

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

- Open another terminal and retrieve the container ID

```
docker ps
```

1	2	3	COMMAND	CREATED	STATUS	PORTS	NAMES
1	2	3	> docker ps	IMAGE	"bash"	12 seconds ago	Up 11 seconds
				ubuntu-gui:1.0			awesome_heyrovsky

- Inspect the container

```
docker inspect --size 4e1
```

```
1 "SizeRw": 0,
2     "SizeRootFs": 1382303225,
3     "Mounts": [],
4     "Config": {
5         "Hostname": "4e1374321ec5",
6         "Domainname": "",
7         "User": "",
8         "AttachStdin": true,
9         "AttachStdout": true,
10        "AttachStderr": true,
11        "Tty": true,
12        "OpenStdin": true,
13        "StdinOnce": true,
14        "Env": [
15            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
16            "DEBIAN_FRONTEND=noninteractive"
17        ],
18    }
19 }
```

## Technique 31 : Cleanly Killing the containers

### STEPS

1. Run a container

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

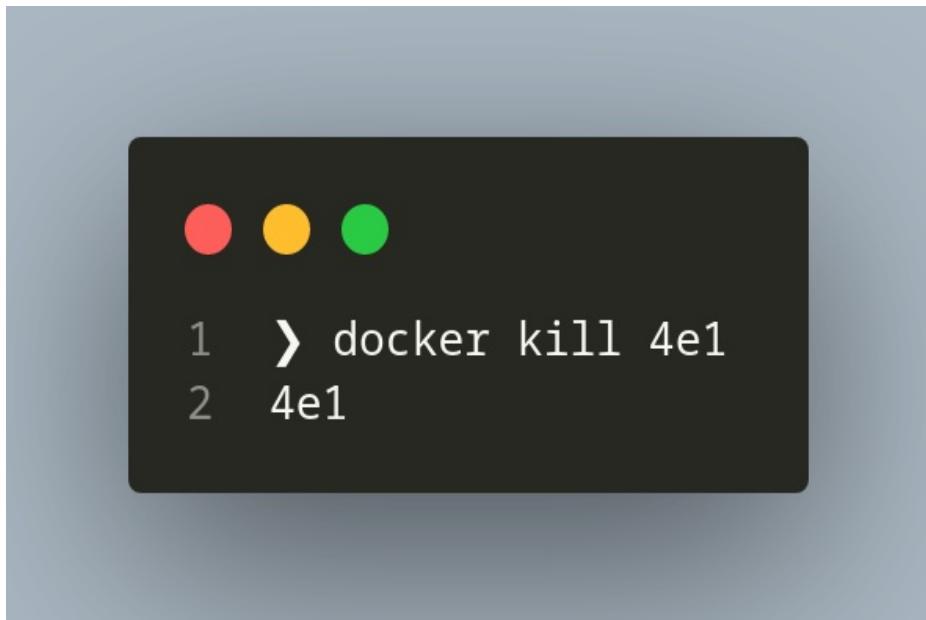
2. Open another terminal and retrieve the Container ID

```
docker ps
```

```
1 ➜ docker ps
2 CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS      PORTS     NAMES
3 4e1374321ec5   ubuntu-gui:1.0   "bash"        12 seconds ago   Up 11 seconds   0.0.0.0:22->22   awesome_heyrovsky
```

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
```

```
1  > dig a "*.google.com"
2
3  ; <>> DiG 9.18.17 <>> a *.google.com
4  ;; global options: +cmd
5  ;; Got answer:
6  ;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 23968
7  ;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
8
9  ;; OPT PSEUDOSECTION:
10 ; EDNS: version: 0, flags:; udp: 1472
11 ;; QUESTION SECTION:
12 ;*.google.com.           IN      A
13
14 ;; AUTHORITY SECTION:
15 google.com.          1022    IN      SOA     ns1.google.com. dns-admin.google.com. 558736483 900 900 1800 60
16
17 ;; Query time: 93 msec
18 ;; SERVER: 192.168.0.1#53(192.168.0.1) (UDP)
19 ;; WHEN: Fri Aug 25 22:59:50 IST 2023
20 ;; MSG SIZE  rcvd: 91
21
```

## 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

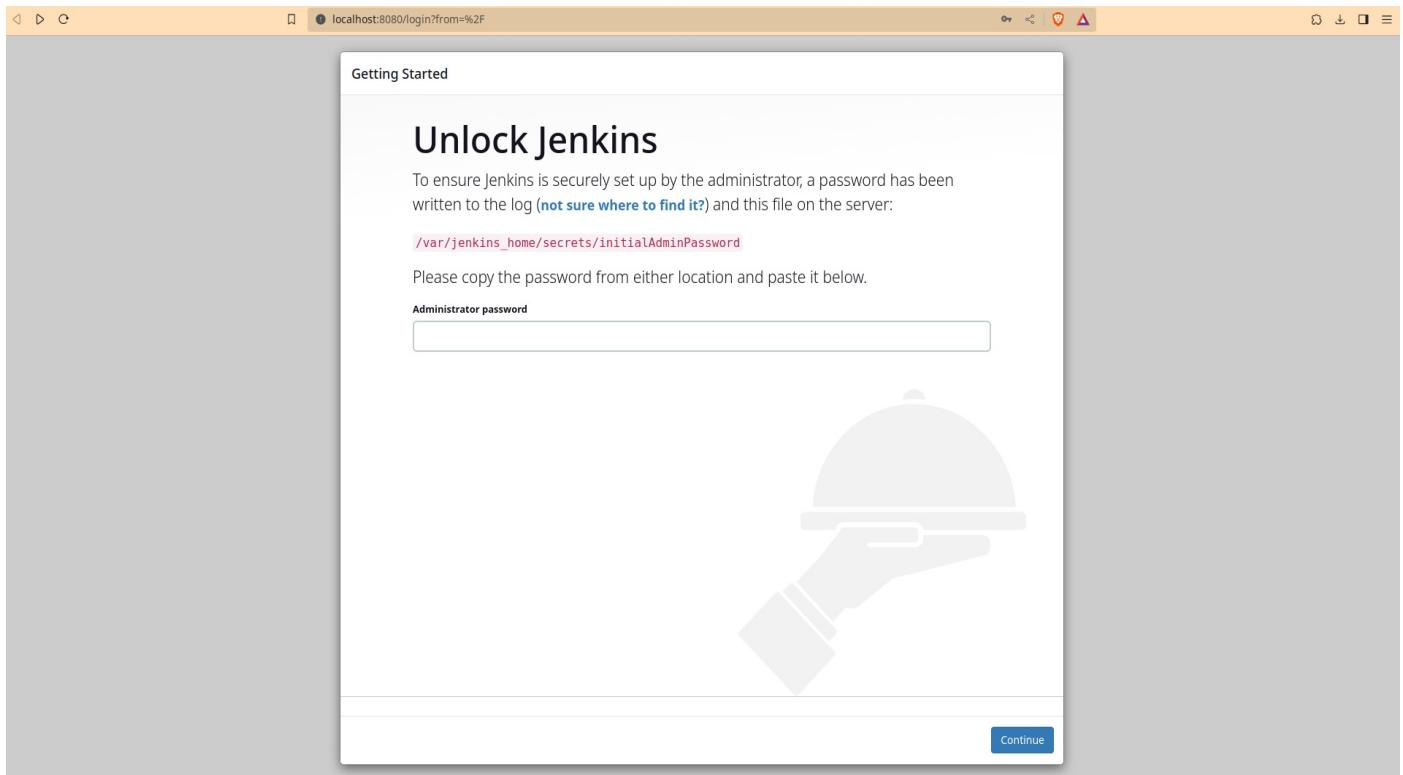
### STEPS

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

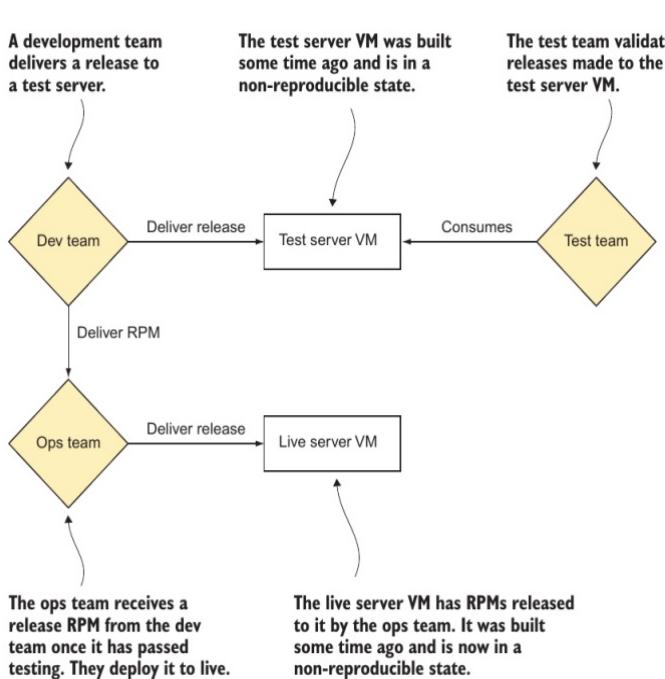


Figure 9.2 Before: a typical software workflow

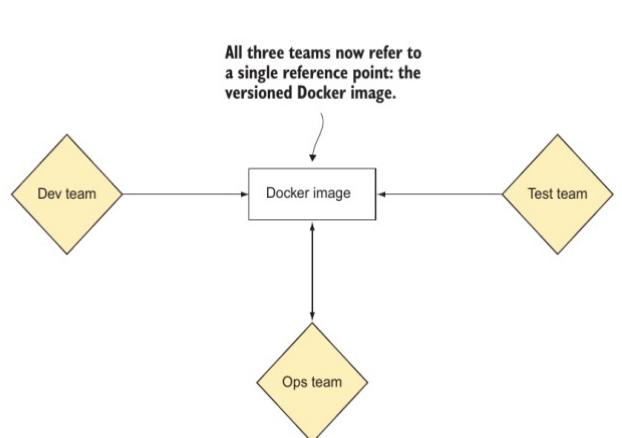


Figure 9.3 After: the Docker contract

## 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

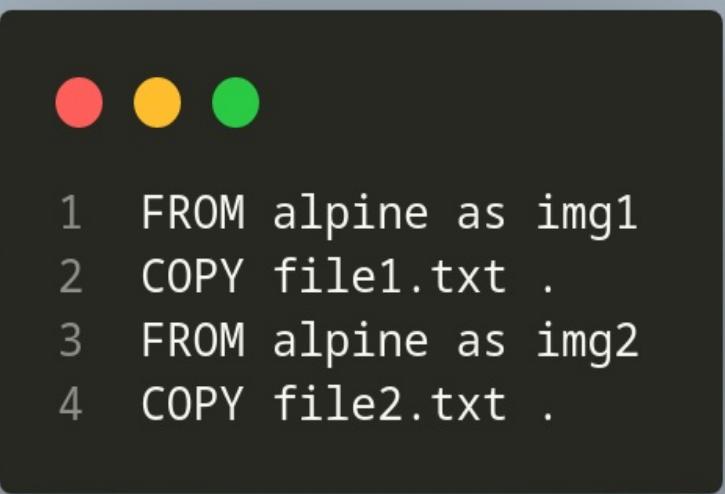
```
sudo docker push srikarthickk/httpd:1.0
```

## Technique 72 : Delivering Images Over Constrained Connections

### STEPS

1.

Create a Dockerfile with multiple images



```
1 FROM alpine as img1
2 COPY file1.txt .
3 FROM alpine as img2
4 COPY file2.txt .
```

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

```
1  > docker images
2  REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
3  image2              latest   c90543ece25c  4 seconds ago  7.33MB
4  image1              latest   730fe456b6d2  13 seconds ago 7.33MB
```

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

```
1  > docker container exec -it image1 /bin/sh
2  / # ping 172.18.0.2/16
3  ping: bad address '172.18.0.2/16'
4  / # ping 172.18.0.2
5  PING 172.18.0.2 (172.18.0.2): 56 data bytes
6  64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.079 ms
7  64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.104 ms
8  64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.103 ms
9  64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.105 ms
10 ^C
11 --- 172.18.0.2 ping statistics ---
12 4 packets transmitted, 4 packets received, 0% packet loss
13 round-trip min/avg/max = 0.079/0.097/0.105 ms
14
```

## 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



```
1  docker load --input image2-1.tar
2  Loaded image: image2:2.0
3  Loaded image: image2:latest
```

## 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
```



```
1  docker exec etcd etcdctl set my_name SRI_KARTHICK
2  SRI_KARTHICK
3  docker exec etcd etcdctl get my_name
4  SRI_KARTHICK
```