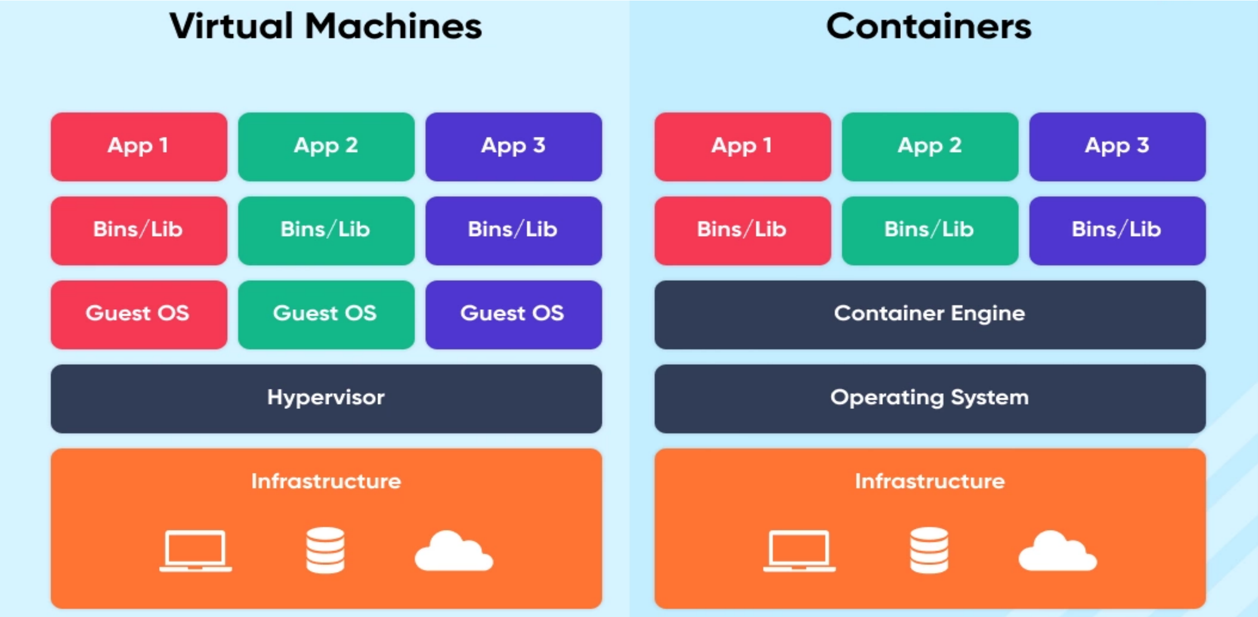
# Table of Contents

# Docker

## Why we use Container over VM

What we earlier using is virtualization on on-prem



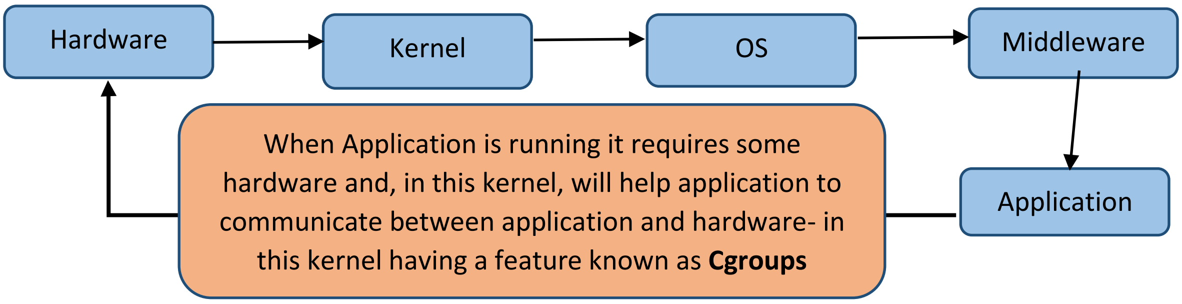
Past Scenario: - Earlier at On-Prem or Azure Cloud Data center level we are having a hardware Infrastructure on which we deploy a Hypervisor Over which we deploy different VMs with different OS and application with required lib/bin.

Problem Statement: - (Bulky VM due to Hardware used by OS- Meanwhile application is taking less space)If we are installing a different OS at different VM, That OS is consuming a lot of space and due to which Hardware utilization at datacentre on OS level is more.

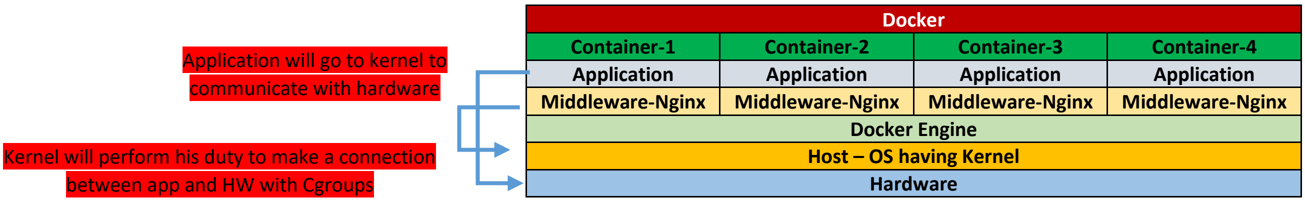
Solution: - To overcome this we need to deploy OS at Hardware level and we can create container for installation of application. So we are going to use Docker engine instead of Hypervisor. And now in each container only application and middleware will be there and VM is replaced by container.

Docker will inherit the feature of OS from Host OS.

In earlier scenario below diagram


Hardware
Kernel
OS
Middleware
Application
When Application is running it requires some hardware and, in this kernel, will help application to communicate between application and hardware- in this kernel having a feature known as **Cgroups**

Now similarly in Host OS on Docker there is some Kernel present that will help application to communicate with hardware. Same Cgroups – Linux feature is used by Docker of Host –OS kernel



Application will go to kernel to communicate with hardware

Kernel will perform his duty to make a connection between app and HW with Cgroups

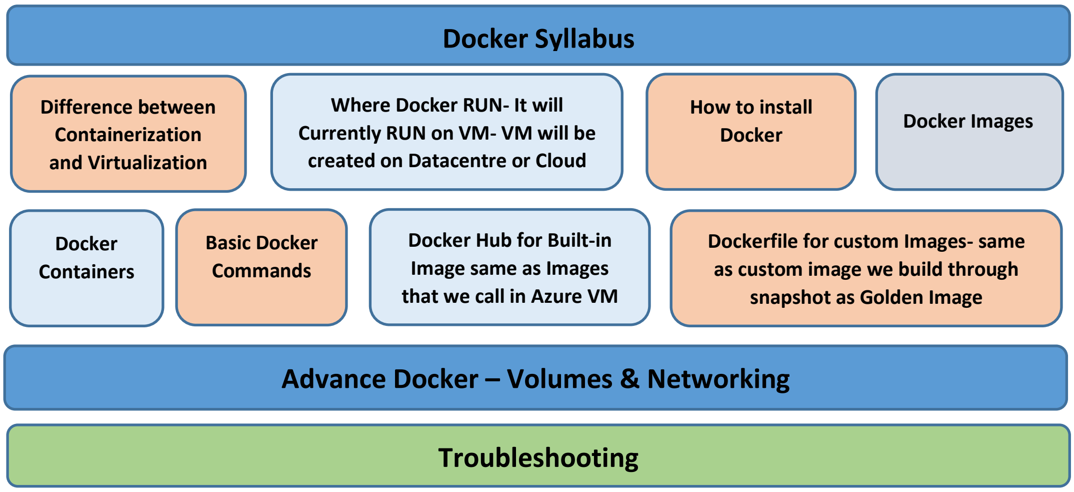
| Docker | | | |
|---|---|---|---|
| Container-1 | Container-2 | Container-3 | Container-4 |
| Application | Application | Application | Application |
| Middleware-Nginx | Middleware-Nginx | Middleware-Nginx | Middleware-Nginx |
| Docker Engine | | | |
| Host – OS having Kernel | | | |
| Hardware | | | |

## Benefits of Docker container

- Application restart fast within a fraction of second
- Faster deployment – less downtime
- Runtime Memory allocation

Previously, there was an issue when installing applications that depend on NodeJS; if we had to install them in different environments like test, non-prod, and prod, the version of NodeJS could differ in each environment. However, with containers, we can package the entire application along with its dependencies and configurations, allowing us to deploy it all at once, which speeds up the deployment process.

## Docker Syllabus



Docker Syllabus

Difference between Containerization and Virtualization

Where Docker RUN- It will Currently RUN on VM- VM will be created on Datacentre or Cloud

How to install Docker

Docker Images

Docker Containers

Basic Docker Commands

Docker Hub for Built-in Image same as Images that we call in Azure VM

Dockerfile for custom Images- same as custom image we build through snapshot as Golden Image

Advance Docker – Volumes & Networking

Troubleshooting

### How to Install Docker

### For Windows

Search on google Docker Desktop >>>>> Download >>>> Run Exe file >>>>next >>>>RUN Docker

https://docs.docker.com/desktop/install/windows-install/

If you are doing it on your laptop/desktop than it may slow your system operation it’s better to install it on a Windows VM over the cloud.

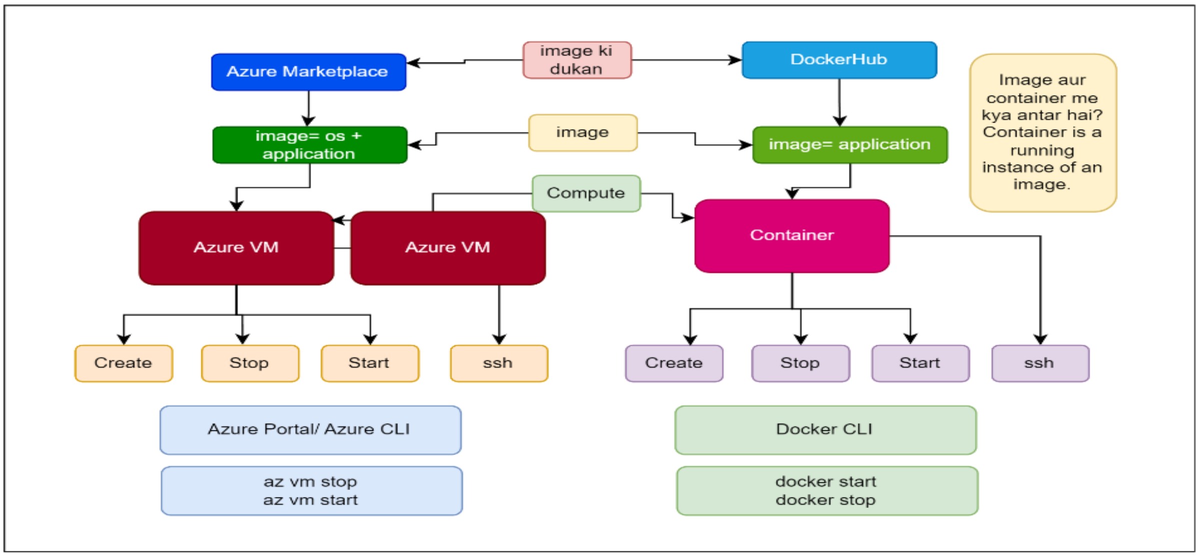
For Linux

https://docs.docker.com/desktop/install/linux/

### What all process we can do on container

Docker will run through CLI. There are some commands from which we will perform different jobs

From Docker –help



# Commands

**Run:** - create and run a new container from an image

Now you may think from where this image comes. Just take an example of VM where when we select Image of OS type while creating VM and it will pull image from marketplace or custom image of VM snapshot. Similarly Docker will pull image from **DockerHub** and Docker images (custom image we create through writing a **Dockerfile**)

Note: - VM always run on installed OS that we select from Image but in case of Docker container will run on Host-OS no special OS required. Image itself contains the OS required

Note: - Image (a package with configurations of application) is that contain application to be run and container is a running instance of an image. Simply Image requires a container to run an application.

**Problem statement: -** How to create and run a container?? How to stop, start, ssh on container

**Solution with example: -** we will install nginx on container and run it on website

Some docker command we can find running **docker --help**

## Common Commands:

| | |
|---|---|
| **run** | Create and run a new container from an image |
| **exec** | Execute a command in a running container |
| **ps** | List containers |
| **build** | Build an image from a Dockerfile |
| **pull** | Download an image from a registry |
| **push** | Upload an image to a registry |
| **images** | List images |
| **login** | Log in to a registry |
| **logout** | Log out from a registry |
| **search** | Search Docker Hub for images |
| **version** | Show the Docker version information |
| **info** | Display system-wide information |

### Management Commands:

| | |
|---|---|
| **builder** | Manage builds |
| **container** | Manage containers |
| **context** | Manage contexts |

| | |
|---|---|
| **image** | Manage images |
| **manifest** | Manage Docker image manifests and manifest lists |
| **network** | Manage networks |
| **plugin** | Manage plugins |
| **system** | Manage Docker |
| **trust** | Manage trust on Docker images |
| **volume** | Manage volumes |

### Swarm Commands:

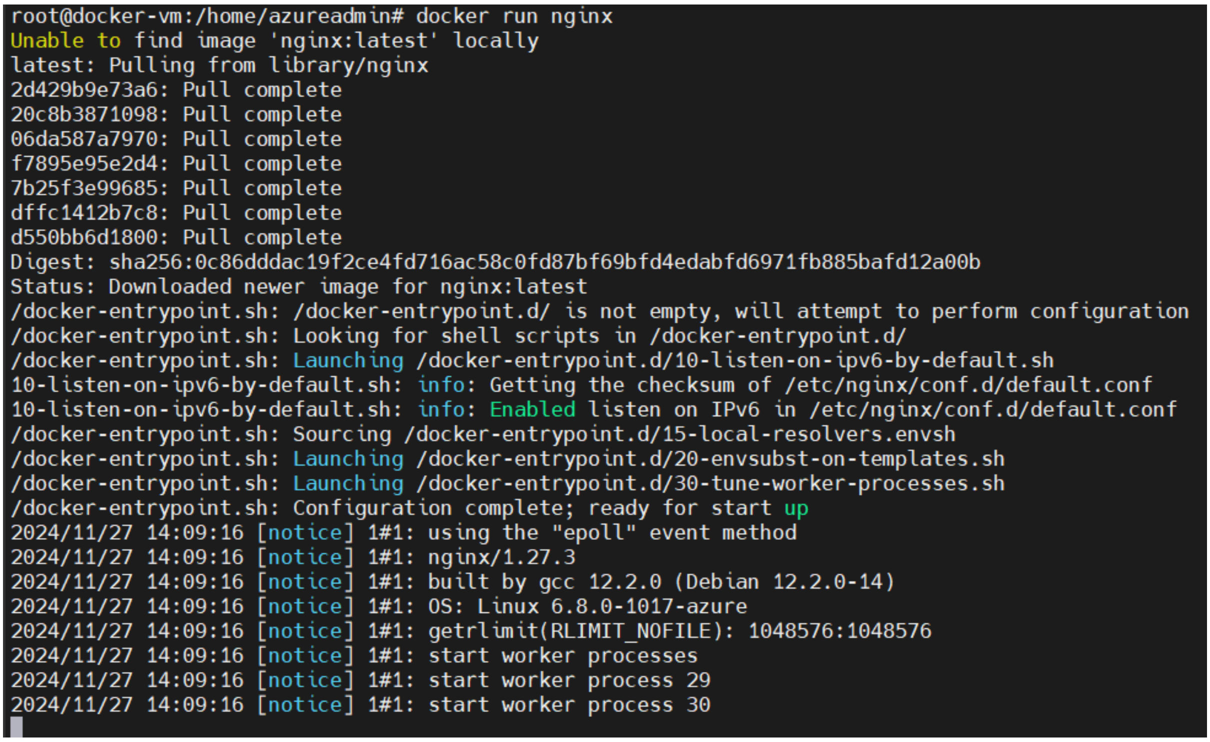
| | |
|---|---|
| **swarm** | Manage Swarm |

### Commands:

| | |
|---|---|
| **attach** | Attach local standard input, output, and error streams to a running container |
| **commit** | Create a new image from a container's changes |
| **cp** | Copy files/folders between a container and the local filesystem |
| **create** | Create a new container |
| **diff** | Inspect changes to files or directories on a container's filesystem |
| **events** | Get real time events from the server |
| **export** | Export a container's filesystem as a tar archive |
| **history** | Show the history of an image |
| **import** | Import the contents from a tarball to create a filesystem image |
| **inspect** | Return low-level information on Docker objects |
| **kill** | Kill one or more running containers |
| **load** | Load an image from a tar archive or STDIN |
| **logs** | Fetch the logs of a container |
| **pause** | Pause all processes within one or more containers |
| **port** | List port mappings or a specific mapping for the container |
| **rename** | Rename a container |
| **restart** | Restart one or more containers |
| **rm** | Remove one or more containers |
| **rmi** | Remove one or more images |
| **save** | Save one or more images to a tar archive (streamed to STDOUT by default) |
| **start** | Start one or more stopped containers |
| **stats** | Display a live stream of container(s) resource usage statistics |
| **stop** | Stop one or more running containers |
| **tag** | Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE |
| **top** | Display the running processes of a container |
| **unpause** | Unpause all processes within one or more containers |
| **update** | Update configuration of one or more containers |
| **wait** | Block until one or more containers stop, then print their exit codes |

Command for installing nginx which will be pulled from **DockerHub**

- docker run nginx



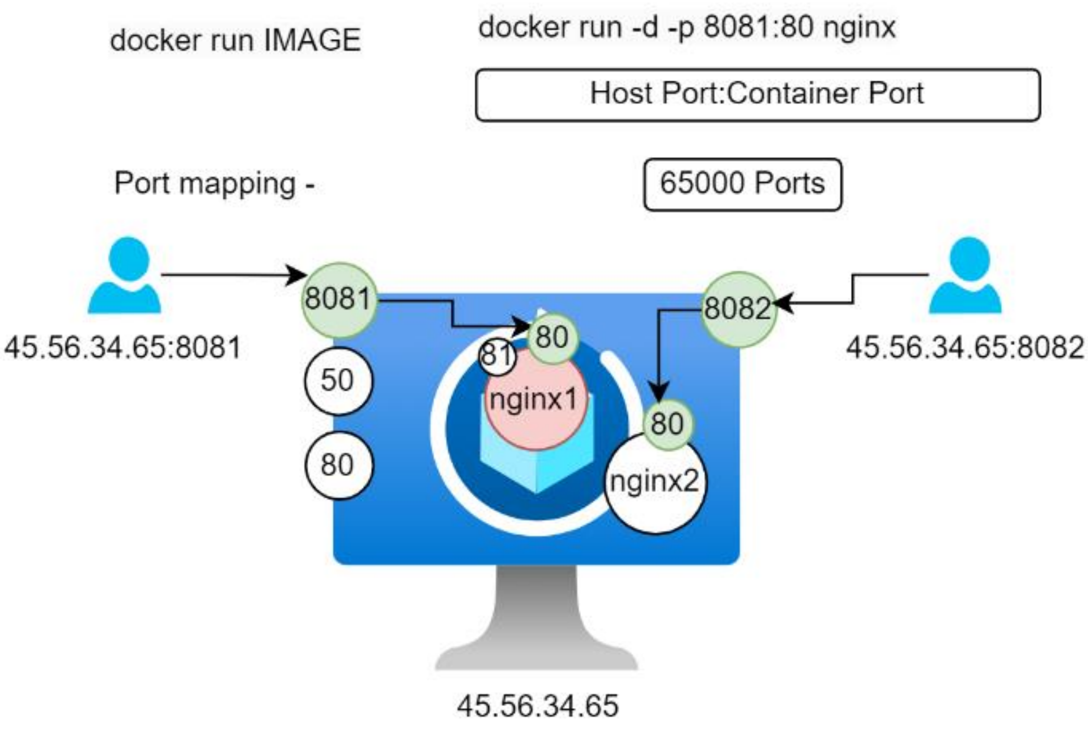
```
root@docker-vm:/home/azureadmin# docker run nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
2d429b9e73a6: Pull complete
20c8b3871098: Pull complete
06da587a7970: Pull complete
f7895e95e2d4: Pull complete
7b25f3e99685: Pull complete
dffc1412b7c8: Pull complete
d550bb6d1800: Pull complete
Digest: sha256:0c86dddac19f2ce4fd716ac58c0fd87bf69bfd4edabfd6971fb885bafd12a00b
Status: Downloaded newer image for nginx:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/11/27 14:09:16 [notice] 1#1: using the "epoll" event method
2024/11/27 14:09:16 [notice] 1#1: nginx/1.27.3
2024/11/27 14:09:16 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/11/27 14:09:16 [notice] 1#1: OS: Linux 6.8.0-1017-azure
2024/11/27 14:09:16 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/11/27 14:09:16 [notice] 1#1: start worker processes
2024/11/27 14:09:16 [notice] 1#1: start worker process 29
2024/11/27 14:09:16 [notice] 1#1: start worker process 30
```

now nginx is installed and running now how this nginx will access

Nginx is running on a container in a VM and if a user wants to access that nginx than it should first go to private/public IP of the VM and then to container where nginx installed. Here comes a concept of port mapping.

## Port Mapping: -

If we have to access nginx server we have to first access the VM PIP with some port (80, 443, etc.,) which is applied through NSG and which be mapped with to some port of container where nginx is installed.



**Host Port**: - where container is build

**Container port**: - where application is installed

Usage: docker run [options] image [command] [argument]

- **Docker run -p 81:80 nginx**

Now to access the application of container user needs to browse the URL: http://publicip:81 or https://publicip:81

Now for host we can know from NSG which port is used and for container if we want to know which port, we are using than we have to go to docker hub image we can search for port used in that document.

Similarly, if we want to run nginx on port 8082 we have to just run below command

- **Docker run -p 8082:80 nginx**

As soon as we run

- **docker run -p hostport:containerappport image**

On CLI container start running but we can perform any other task on CLI to overcome this we have to RUN the service in background by following command it also print container ID

- **Docker run -d -p 8081:80 nginx**
- **And if we want other nginx to run on other port**
- **Docker run -p 8082:80 nginx**

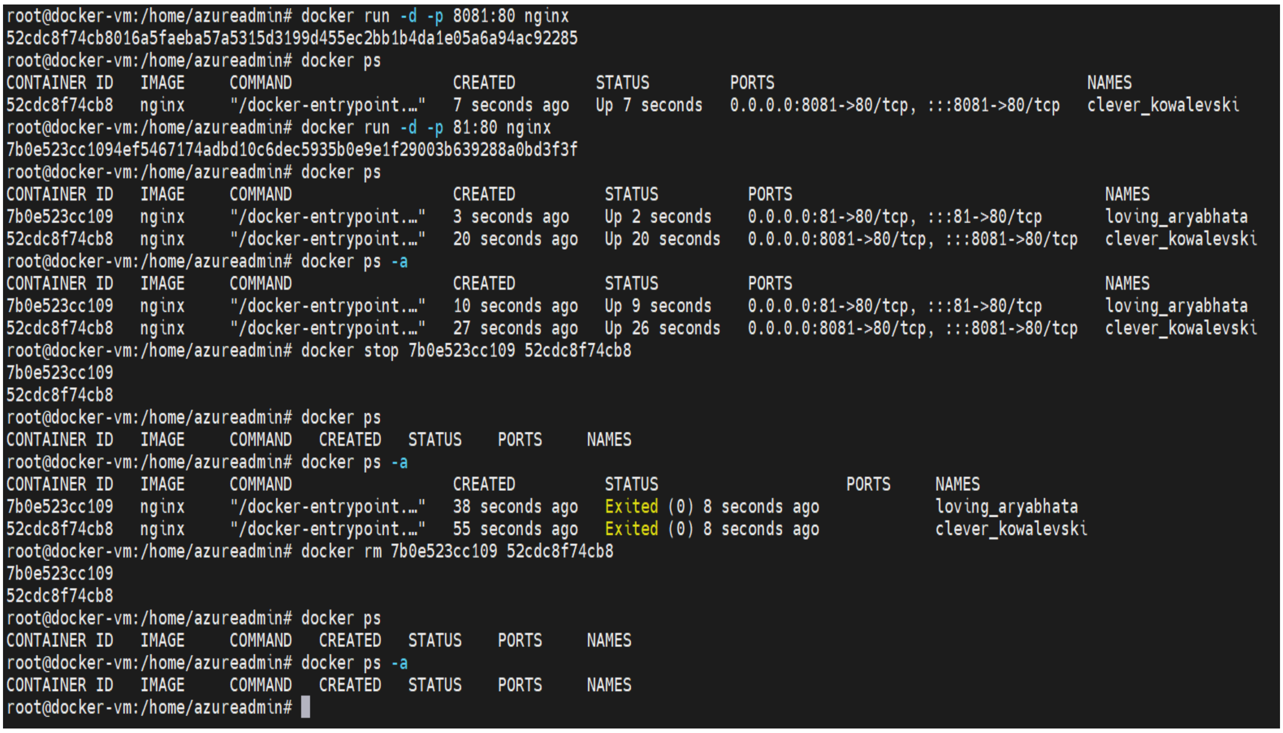
**Note:-** This will run you application always

Now if we want to check numbers of containers running

- Docker ps

And if we want to see all containers running or stopped we have to run following command

- Docker ps -a



```
root@docker-vm:/home/azureadmin# docker run -d -p 8081:80 nginx
52cdc8f74cb8016a5faeba57a5315d3199d455ec2bb1b4da1e05a6a94ac92285
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                   NAMES
52cdc8f74cb8   nginx     "/docker-entrypoint.…"   7 seconds ago   Up 7 seconds   0.0.0.0:8081->80/tcp, :::8081->80/tcp   clever_kowalevski
root@docker-vm:/home/azureadmin# docker run -d -p 81:80 nginx
7b0e523cc1094ef5467174adbd10c6dec5935b0e9e1f29003b639288a0bd3f3f
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                   NAMES
7b0e523cc109   nginx     "/docker-entrypoint.…"   3 seconds ago    Up 2 seconds    0.0.0.0:81->80/tcp, :::81->80/tcp       loving_aryabhata
52cdc8f74cb8   nginx     "/docker-entrypoint.…"   20 seconds ago   Up 20 seconds   0.0.0.0:8081->80/tcp, :::8081->80/tcp   clever_kowalevski
root@docker-vm:/home/azureadmin# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                   NAMES
7b0e523cc109   nginx     "/docker-entrypoint.…"   10 seconds ago   Up 9 seconds    0.0.0.0:81->80/tcp, :::81->80/tcp       loving_aryabhata
52cdc8f74cb8   nginx     "/docker-entrypoint.…"   27 seconds ago   Up 26 seconds   0.0.0.0:8081->80/tcp, :::8081->80/tcp   clever_kowalevski
root@docker-vm:/home/azureadmin# docker stop 7b0e523cc109 52cdc8f74cb8
7b0e523cc109
52cdc8f74cb8
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@docker-vm:/home/azureadmin# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                     PORTS     NAMES
7b0e523cc109   nginx     "/docker-entrypoint.…"   38 seconds ago   Exited (0) 8 seconds ago             loving_aryabhata
52cdc8f74cb8   nginx     "/docker-entrypoint.…"   55 seconds ago   Exited (0) 8 seconds ago             clever_kowalevski
root@docker-vm:/home/azureadmin# docker rm 7b0e523cc109 52cdc8f74cb8
7b0e523cc109
52cdc8f74cb8
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@docker-vm:/home/azureadmin# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@docker-vm:/home/azureadmin# 
```

**To access this nginx server just type**

http://public-ip-of-vm:8081

http://public-ip-of-vm:81

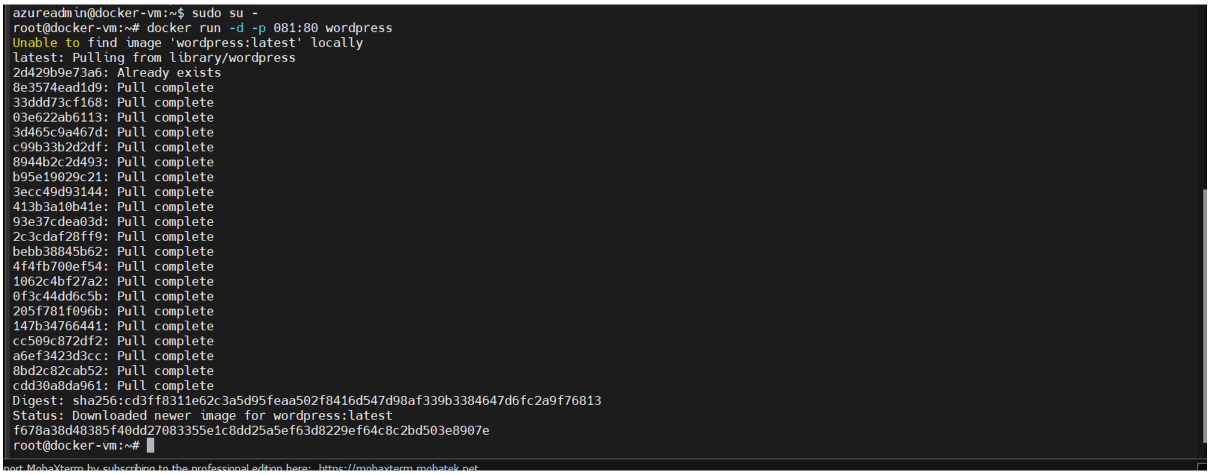
## WORDPRESS Application

Need to run an application WORDPRESS

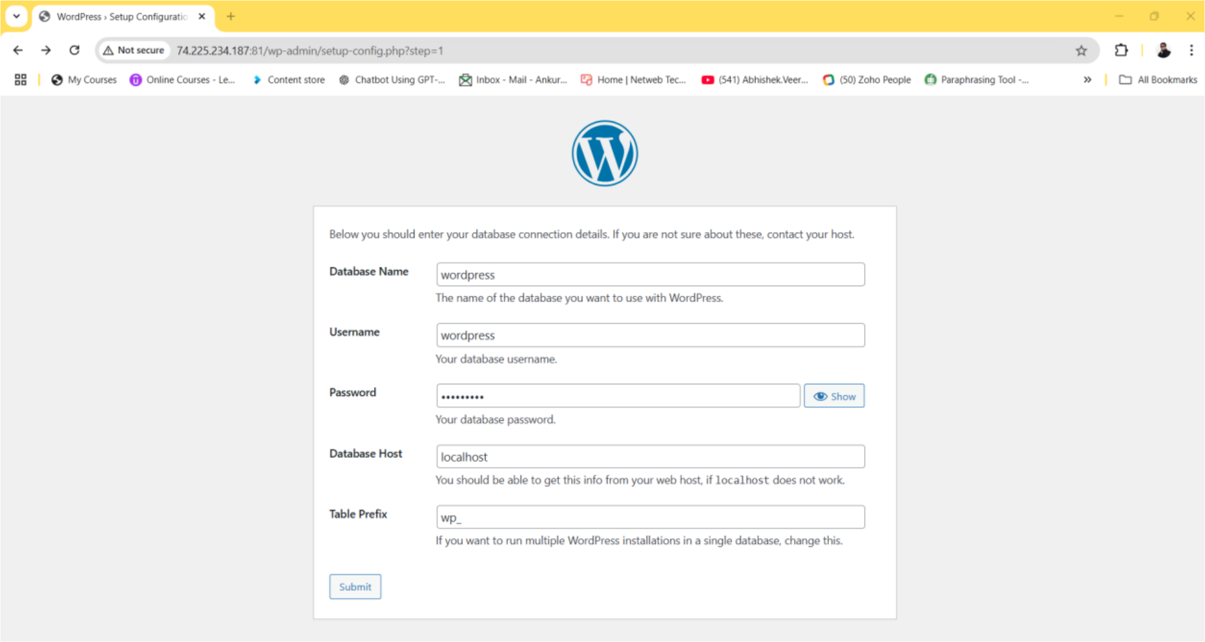
go to Docker Hub and search for WordPress and in that document search for the port on which by default WordPress is working

now just go to your Docker server VM and RUN below command to install the WordPress application

- sudo docker -d -p 81:80 wordpress



```
azureadmin@docker-vm:~$ sudo su -
root@docker-vm:~# docker run -d -p 081:80 wordpress
Unable to find image 'wordpress:latest' locally
latest: Pulling from library/wordpress
2d429b9e73a6: Already exists
8e3574ead1d9: Pull complete
33ddd73cf168: Pull complete
03e622ab6113: Pull complete
3d465c9a467d: Pull complete
c99b33b2d2df: Pull complete
8944b2c2d493: Pull complete
b95e19029c21: Pull complete
3ecc49d93144: Pull complete
413b3a10b41e: Pull complete
93e37cdea03d: Pull complete
2c3cdaf28ff9: Pull complete
bebb38845b62: Pull complete
4f4fb700ef54: Pull complete
1062c4bf27a2: Pull complete
0f3c44dd6c5b: Pull complete
205f781f096b: Pull complete
147b34766441: Pull complete
cc509c872df2: Pull complete
a6ef3423d3cc: Pull complete
8bd2c82cab52: Pull complete
cdd30a8da961: Pull complete
Digest: sha256:cd3ff8311e62c3a5d95feaa502f8416d547d98af339b3384647d6fc2a9f76813
Status: Downloaded newer image for wordpress:latest
f678a38d48385f40dd27083355e1c8dd25a5ef63d8229ef64c8c2bd503e8907e
root@docker-vm:~#
```



Below you should enter your database connection details. If you are not sure about these, contact your host.

Database Name: wordpress
The name of the database you want to use with WordPress.

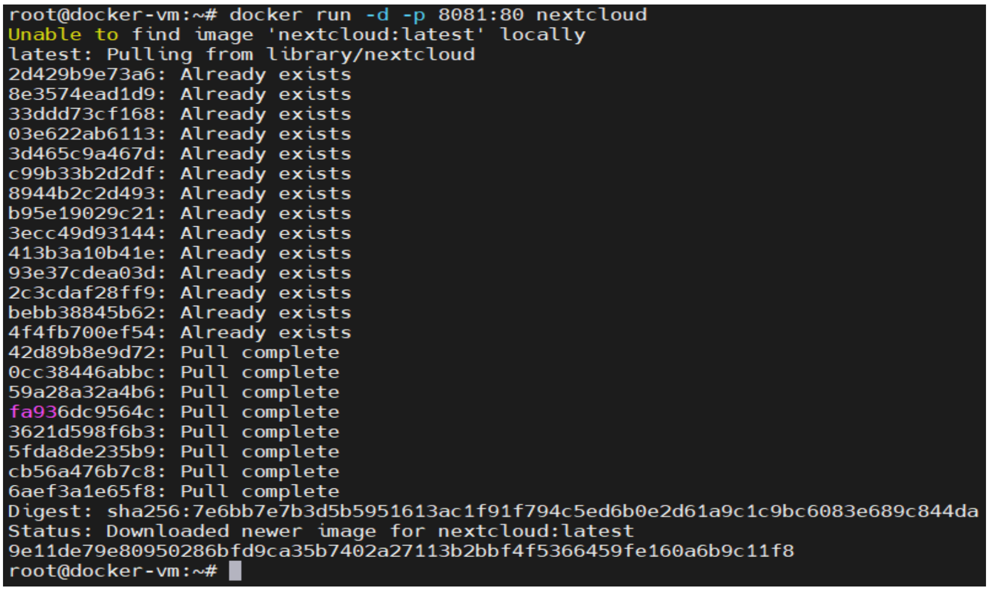
Username: wordpress
Your database username.

Password: •••••••••
Your database password.

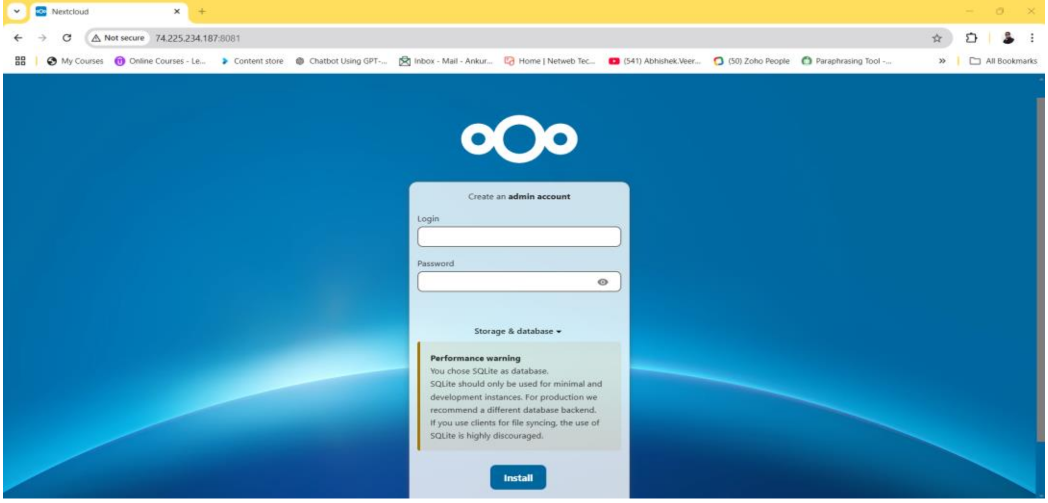
Database Host: localhost
You should be able to get this info from your web host, if `localhost` does not work.

Table Prefix: wp_
If you want to run multiple WordPress installations in a single database, change this.

## Next cloud application



```
root@docker-vm:~# docker run -d -p 8081:80 nextcloud
Unable to find image 'nextcloud:latest' locally
latest: Pulling from library/nextcloud
2d429b9e73a6: Already exists
8e3574ead1d9: Already exists
33ddd73cf168: Already exists
03e622ab6113: Already exists
3d465c9a467d: Already exists
c99b33b2d2df: Already exists
8944b2c2d493: Already exists
b95e19029c21: Already exists
3ecc49d93144: Already exists
413b3a10b41e: Already exists
93e37cdea03d: Already exists
2c3cdaf28ff9: Already exists
bebb38845b62: Already exists
4f4fb700ef54: Already exists
42d89b8e9d72: Pull complete
0cc38446abbc: Pull complete
59a28a32a4b6: Pull complete
fa936dc9564c: Pull complete
3621d598f6b3: Pull complete
5fda8de235b9: Pull complete
cb56a476b7c8: Pull complete
6aef3a1e65f8: Pull complete
Digest: sha256:7e6bb7e7b3d5b5951613ac1f91f794c5ed6b0e2d61a9c1c9bc6083e689c844da
Status: Downloaded newer image for nextcloud:latest
9e11de79e80950286bfd9ca35b7402a27113b2bbf4f5366459fe160a6b9c11f8
root@docker-vm:~#
```



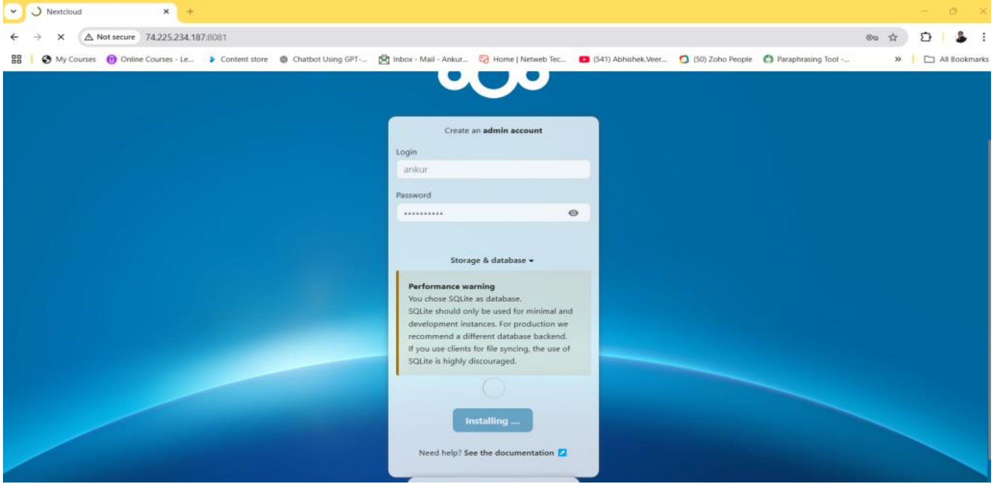
Create an **admin account**

Login

Password

Storage & database ▾

**Performance warning**
You chose SQLite as database.
SQLite should only be used for minimal and development instances. For production we recommend a different database backend.
If you use clients for file syncing, the use of SQLite is highly discouraged.



Create an **admin account**

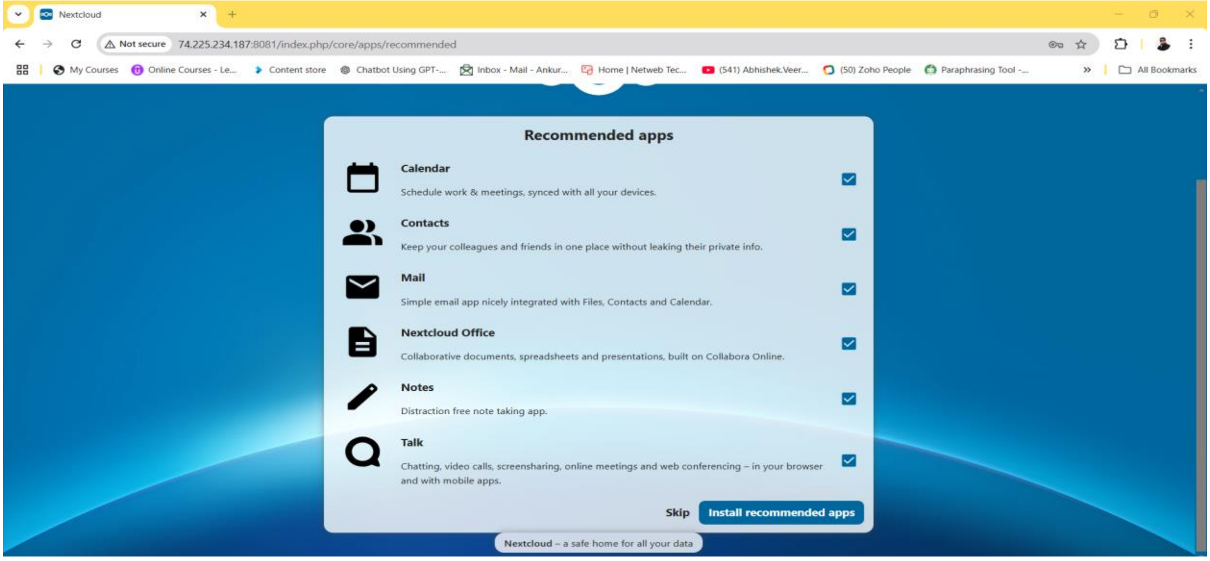
Login: ankur

Password: ••••••••••

Storage & database ▾

**Performance warning**
You chose SQLite as database.
SQLite should only be used for minimal and development instances. For production we recommend a different database backend.
If you use clients for file syncing, the use of SQLite is highly discouraged.

Need help? **See the documentation**



Recommended apps

**Calendar** ☑
Schedule work & meetings, synced with all your devices.

**Contacts** ☑
Keep your colleagues and friends in one place without leaking their private info.

**Mail** ☑
Simple email app nicely integrated with Files, Contacts and Calendar.

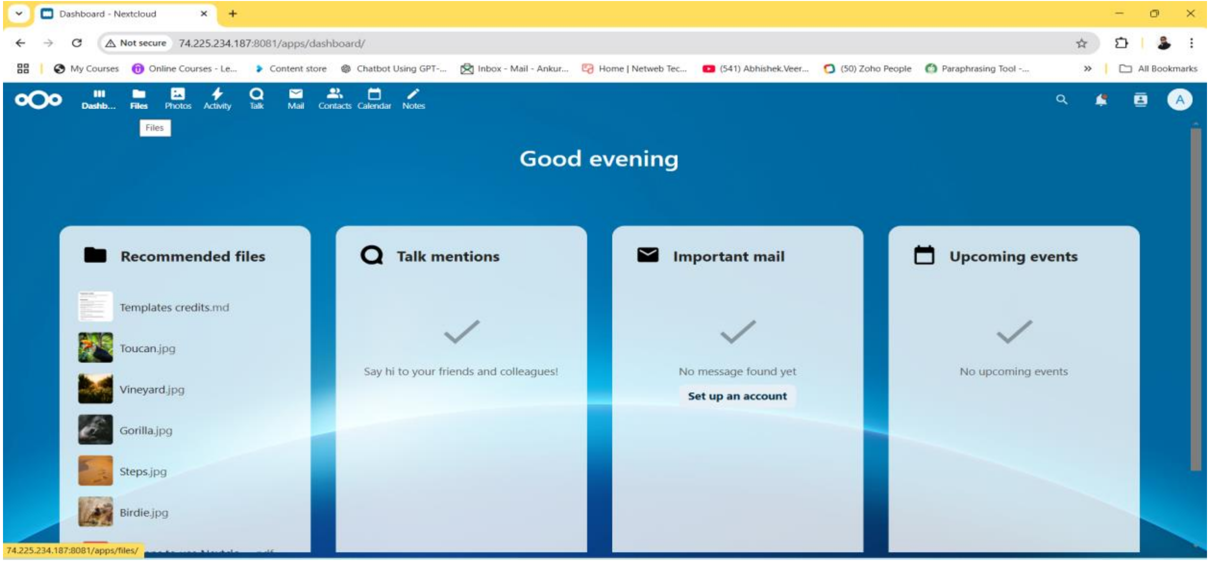
**Nextcloud Office** ☑
Collaborative documents, spreadsheets and presentations, built on Collabora Online.

**Notes** ☑
Distraction free note taking app.

**Talk** ☑
Chatting, video calls, screensharing, online meetings and web conferencing – in your browser and with mobile apps.

Skip | Install recommended apps

Nextcloud – a safe home for all your data



Good evening

**Recommended files**
Templates credits.md
Toucan.jpg
Vineyard.jpg
Gorilla.jpg
Steps.jpg
Birdie.jpg

**Talk mentions**
Say hi to your friends and colleagues!

**Important mail**
No message found yet
Set up an account

**Upcoming events**
No upcoming events

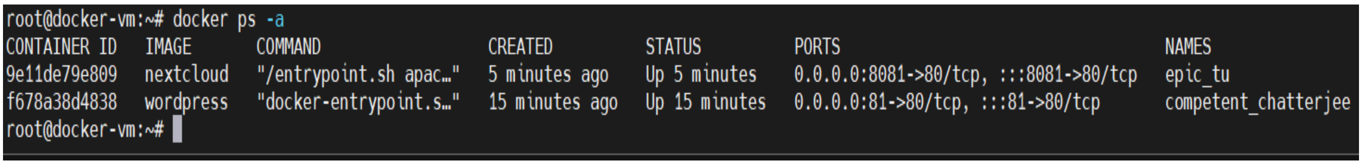
So current Linux VM having Docker installed is running with two applications on different ports like:

Wordpress – 81

Nextcloud – 8081

So to check number of containers we have and of which image we can run

- docker ps -a

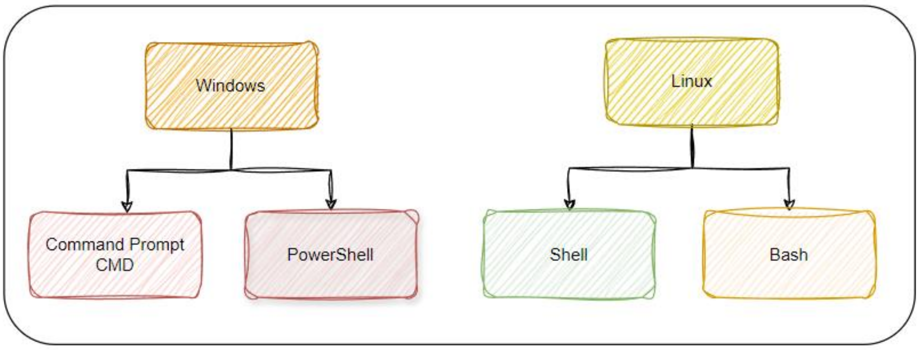

```
root@docker-vm:~# docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                     NAMES
9e11de79e809   nextcloud   "/entrypoint.sh apac…"   5 minutes ago    Up 5 minutes    0.0.0.0:8081->80/tcp, :::8081->80/tcp     epic_tu
f678a38d4838   wordpress   "docker-entrypoint.s…"   15 minutes ago   Up 15 minutes   0.0.0.0:81->80/tcp, :::81->80/tcp         competent_chatterjee
root@docker-vm:~#
```

**Container is a running instance of an image**

| **Commands** | **Details** |
|---|---|
| docker run –d –p | to create a container and -p is for port mapping like 8080:80 where first port is access port of application from public or private IP/URL and second port is the application port, -d is deattach mode if we apply this container will run in background |
| docker images | for listing of all the images you have in your server |
| docker search imagename | as we search images on DockerHub for any application similarly we can search from cli - e.g., docker search python, it will provide the output with all images present in DockerHub |
| docker rmi imagename | To delete any Image from your server but before that no container attached to that Image |
| docker container rm ContainerID | For deleting an container |
| docker container stop ContainerID | To stopping a running container |
| docker container start ContainerID | To start a container |
| docker container restart ContainerID | To restart a container |
| docker exec | ssh inside container, docker exec containerID command like -- docker exec containerID ls, this will run outside container not will go inside |

Now if we have to run any command in windows machine, we use Command Prompt or PowerShell

Similarly in Linux we can run command through Shell or Bash

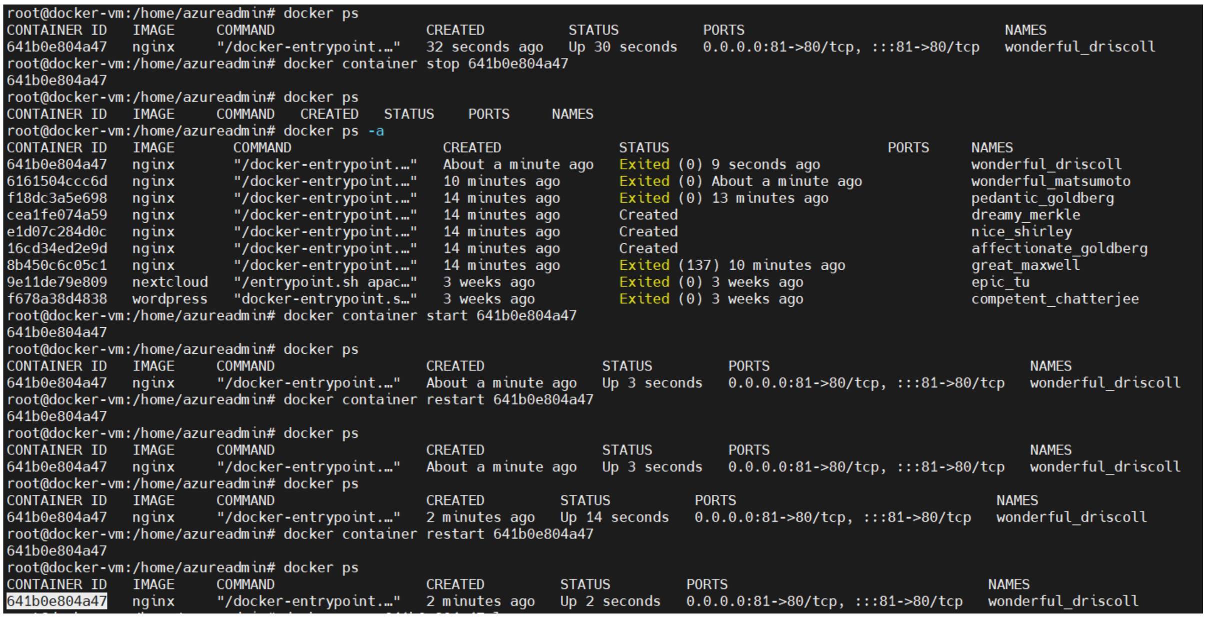


So if we have to go inside the container after ssh we have to use bash

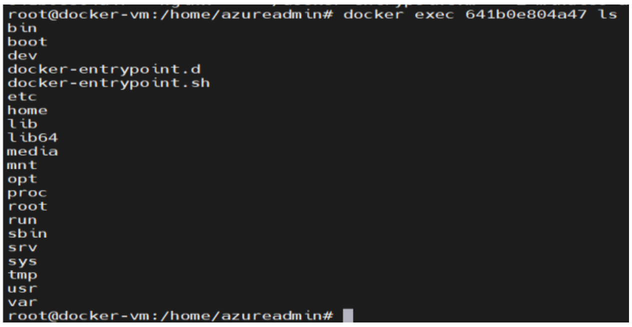
E.g.,

**docker exec -i –t containerid bash**

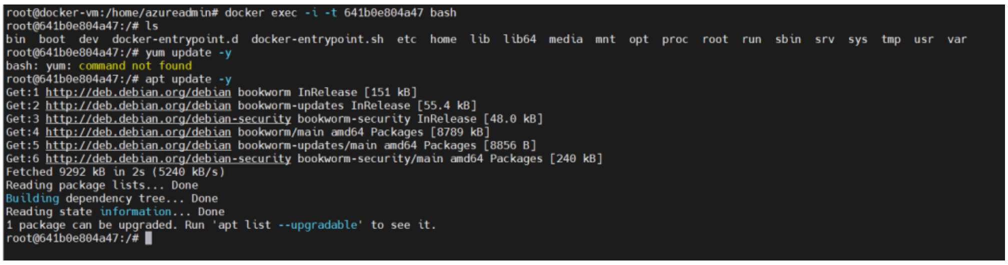
**after which you can run any Command as you work in a Linux VM**



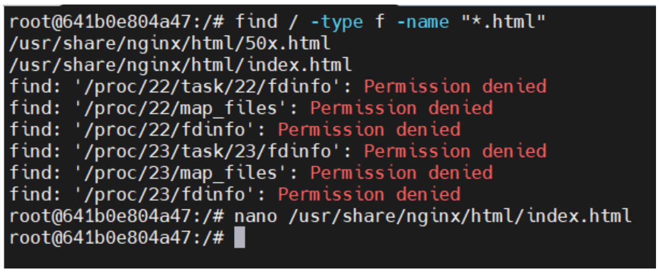
```
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                 NAMES
641b0e804a47   nginx     "/docker-entrypoint.…"   32 seconds ago   Up 30 seconds   0.0.0.0:81->80/tcp, :::81->80/tcp     wonderful_driscoll
root@docker-vm:/home/azureadmin# docker container stop 641b0e804a47
641b0e804a47
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@docker-vm:/home/azureadmin# docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED            STATUS                       PORTS     NAMES
641b0e804a47   nginx       "/docker-entrypoint.…"   About a minute ago Exited (0) 9 seconds ago               wonderful_driscoll
6161504ccc6d   nginx       "/docker-entrypoint.…"   10 minutes ago     Exited (0) About a minute ago          wonderful_matsumoto
f18dc3a5e698   nginx       "/docker-entrypoint.…"   14 minutes ago     Exited (0) 13 minutes ago              pedantic_goldberg
cea1fe074a59   nginx       "/docker-entrypoint.…"   14 minutes ago     Created                                dreamy_merkle
e1d07c284d0c   nginx       "/docker-entrypoint.…"   14 minutes ago     Created                                nice_shirley
16cd34ed2e9d   nginx       "/docker-entrypoint.…"   14 minutes ago     Created                                affectionate_goldberg
8b450c6c05c1   nginx       "/docker-entrypoint.…"   14 minutes ago     Exited (137) 10 minutes ago            great_maxwell
9e11de79e809   nextcloud   "/entrypoint.sh apac…"   3 weeks ago        Exited (0) 3 weeks ago                 epic_tu
f678a38d4838   wordpress   "docker-entrypoint.s…"   3 weeks ago        Exited (0) 3 weeks ago                 competent_chatterjee
root@docker-vm:/home/azureadmin# docker container start 641b0e804a47
641b0e804a47
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS         PORTS                                 NAMES
641b0e804a47   nginx     "/docker-entrypoint.…"   About a minute ago   Up 3 seconds   0.0.0.0:81->80/tcp, :::81->80/tcp     wonderful_driscoll
root@docker-vm:/home/azureadmin# docker container restart 641b0e804a47
641b0e804a47
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS         PORTS                                 NAMES
641b0e804a47   nginx     "/docker-entrypoint.…"   About a minute ago   Up 3 seconds   0.0.0.0:81->80/tcp, :::81->80/tcp     wonderful_driscoll
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS          PORTS                                 NAMES
641b0e804a47   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 14 seconds   0.0.0.0:81->80/tcp, :::81->80/tcp     wonderful_driscoll
root@docker-vm:/home/azureadmin# docker container restart 641b0e804a47
641b0e804a47
root@docker-vm:/home/azureadmin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                 NAMES
641b0e804a47   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 2 seconds   0.0.0.0:81->80/tcp, :::81->80/tcp     wonderful_driscoll
```



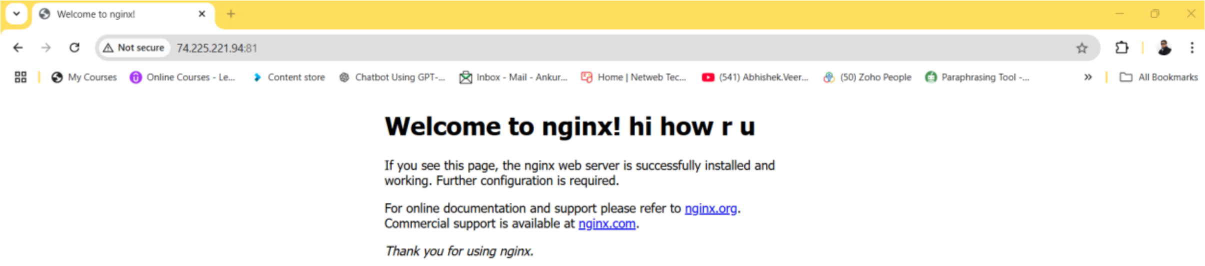
```
root@docker-vm:/home/azureadmin# docker exec 641b0e804a47 ls
bin
boot
dev
docker-entrypoint.d
docker-entrypoint.sh
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
root@docker-vm:/home/azureadmin#
```



```
root@docker-vm:/home/azureadmin# docker exec -i -t 641b0e804a47 bash
root@641b0e804a47:/# ls
bin  boot  dev  docker-entrypoint.d  docker-entrypoint.sh  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@641b0e804a47:/# yum update -y
bash: yum: command not found
root@641b0e804a47:/# apt update -y
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8789 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [8856 B]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [240 kB]
Fetched 9292 kB in 2s (5240 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
root@641b0e804a47:/#
```



```
root@641b0e804a47:/# find / -type f -name "*.html"
/usr/share/nginx/html/50x.html
/usr/share/nginx/html/index.html
find: '/proc/22/task/22/fdinfo': Permission denied
find: '/proc/22/map_files': Permission denied
find: '/proc/22/fdinfo': Permission denied
find: '/proc/23/task/23/fdinfo': Permission denied
find: '/proc/23/map_files': Permission denied
find: '/proc/23/fdinfo': Permission denied
root@641b0e804a47:/# nano /usr/share/nginx/html/index.html
root@641b0e804a47:/#
```

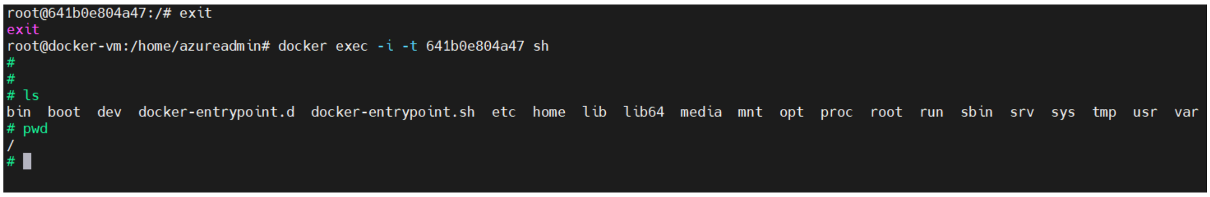


**Welcome to nginx! hi how r u**

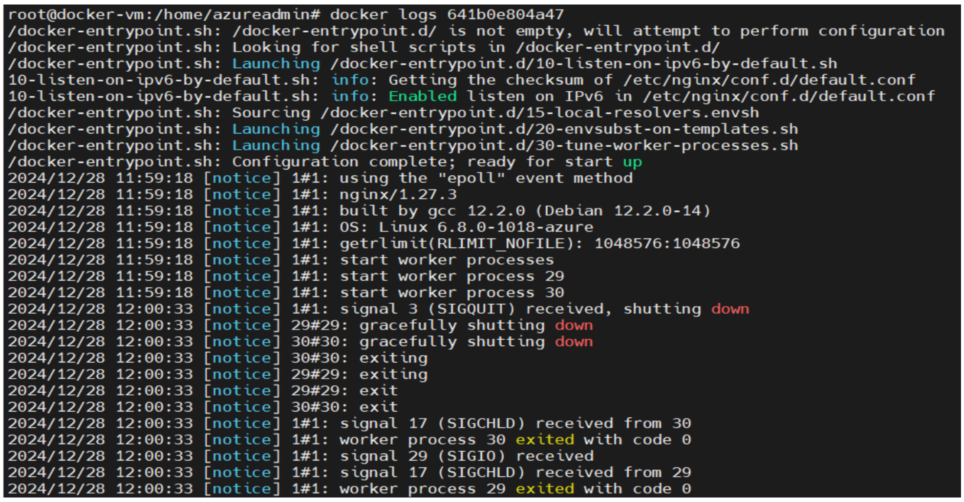
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

*Thank you for using nginx.*



```
root@641b0e804a47:/# exit
exit
root@docker-vm:/home/azureadmin# docker exec -i -t 641b0e804a47 sh
#
#
# ls
bin  boot  dev  docker-entrypoint.d  docker-entrypoint.sh  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# pwd
/
#
```

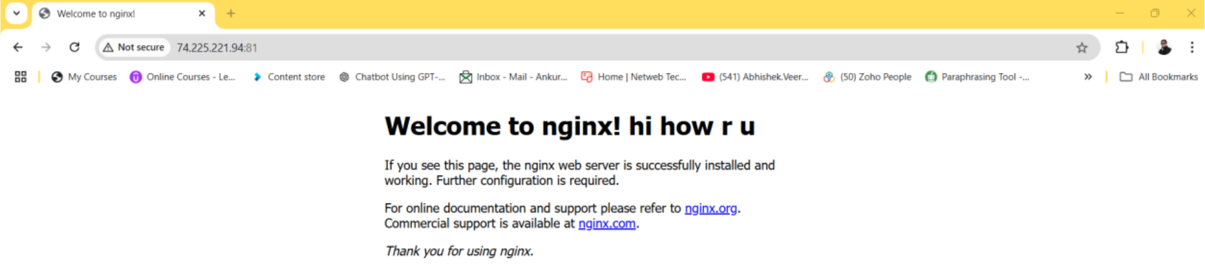


```
root@docker-vm:/home/azureadmin# docker logs 641b0e804a47
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/12/28 11:59:18 [notice] 1#1: using the "epoll" event method
2024/12/28 11:59:18 [notice] 1#1: nginx/1.27.3
2024/12/28 11:59:18 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/12/28 11:59:18 [notice] 1#1: OS: Linux 6.8.0-1018-azure
2024/12/28 11:59:18 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/12/28 11:59:18 [notice] 1#1: start worker processes
2024/12/28 11:59:18 [notice] 1#1: start worker process 29
2024/12/28 11:59:18 [notice] 1#1: start worker process 30
2024/12/28 12:00:33 [notice] 1#1: signal 3 (SIGQUIT) received, shutting down
2024/12/28 12:00:33 [notice] 29#29: gracefully shutting down
2024/12/28 12:00:33 [notice] 30#30: gracefully shutting down
2024/12/28 12:00:33 [notice] 30#30: exiting
2024/12/28 12:00:33 [notice] 29#29: exiting
2024/12/28 12:00:33 [notice] 29#29: exit
2024/12/28 12:00:33 [notice] 30#30: exit
2024/12/28 12:00:33 [notice] 1#1: signal 17 (SIGCHLD) received from 30
2024/12/28 12:00:33 [notice] 1#1: worker process 30 exited with code 0
2024/12/28 12:00:33 [notice] 1#1: signal 29 (SIGIO) received
2024/12/28 12:00:33 [notice] 1#1: signal 17 (SIGCHLD) received from 29
2024/12/28 12:00:33 [notice] 1#1: worker process 29 exited with code 0
```

## Custom Image

Consider a scenario where you have set up a virtual machine (VM) with a specific application installed, along with a customized configuration file and additional settings in other files required to run the updated application. Now, if you need multiple VMs with the exact same setup, the solution is to create an image of the current VM, capturing all the updates, configurations, and files. This image can then be used as a template to deploy any number of new VMs, ensuring consistency across all instances.

We can apply the same concept in a containerized environment. For instance, consider a scenario where changes are made to the index.html file.



**Welcome to nginx! hi how r u**

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

*Thank you for using nginx.*

To achieve the output in above snap in multiple containers, we need to create an image of the container using the following command:

- docker export containerID

In the previous scenario, we first pulled the Nginx image from Docker Hub, accessed the HTML page, and made some changes.

Now, if we want to perform all these steps starting from a custom image — such as pulling an operating system, setting up a web server, and making changes to specific files — we can accomplish all of this in a single step by creating a custom image.

We can automate this process by writing a Dockerfile, which will define the necessary steps. Using this Dockerfile, we can build a custom image. This image can then be used to deploy multiple containers on different ports as needed.

# Dockerfile

A Dockerfile is a simple text file that contains a set of instructions to build a Docker image. It defines everything needed to create an image, including the base image, software installation, environment setup, file copies, and commands to execute when the container runs.

## Dockerfile instruction

For that we can take reference from below URL

https://docs.docker.com/reference/dockerfile/

| Instruction | Description |
|---|---|
| ADD | Add local or remote files and directories. |
| ARG | Use build-time variables. |
| CMD | Specify default commands. |
| COPY | Copy files and directories. |
| ENTRYPOINT | Specify default executable. |
| ENV | Set environment variables. |
| EXPOSE | Describe which ports your application is listening on. |
| FROM | Create a new build stage from a base image. |
| HEALTHCHECK | Check a container's health on startup. |
| LABEL | Add metadata to an image. |
| MAINTAINER | Specify the author of an image. |
| ONBUILD | Specify instructions for when the image is used in a build. |
| RUN | Execute build commands. |
| SHELL | Set the default shell of an image. |
| STOPSIGNAL | Specify the system call signal for exiting a container. |
| USER | Set user and group ID. |
| VOLUME | Create volume mounts. |
| WORKDIR | Change working directory. |

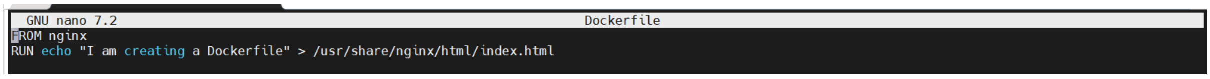
How to write a Dockerfile steps

First we have to bring os and image

- FROM image example FROM nginx

Now we want to execute some commands

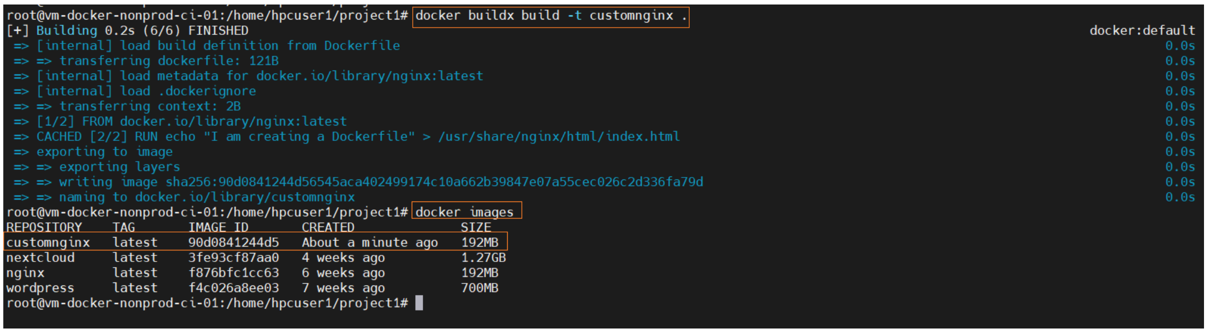
- RUN command example
    - RUN rm /usr/share/nginx/html/index.html
    - RUN echo “I am writing Dockerfile” >  /usr/share/nginx/html/index.html
    - And now we have to build an Image from this docker file



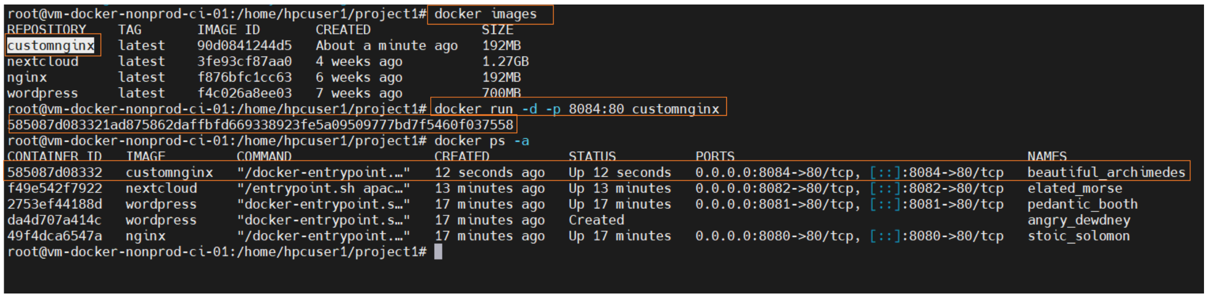
```
  GNU nano 7.2                                                 Dockerfile
FROM nginx
RUN echo "I am creating a Dockerfile" > /usr/share/nginx/html/index.html
```

- docker buildx build –t imagename .

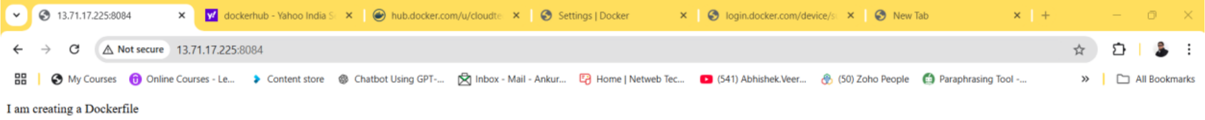
Note whenever we are running a docker build command to create an image than automatically a temporary container is creating in background and as soon as the docker build command execute that container will automatically delete.



```
root@vm-docker-nonprod-ci-01:/home/hpcuser1/project1# docker buildx build -t customnginx .
[+] Building 0.2s (6/6) FINISHED                                                                        docker:default
 => [internal] load build definition from Dockerfile                                                              0.0s
 => => transferring dockerfile: 121B                                                                              0.0s
 => [internal] load metadata for docker.io/library/nginx:latest                                                   0.0s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [1/2] FROM docker.io/library/nginx:latest                                                                     0.0s
 => CACHED [2/2] RUN echo "I am creating a Dockerfile" > /usr/share/nginx/html/index.html                         0.0s
 => exporting to image                                                                                            0.0s
 => => exporting layers                                                                                           0.0s
 => => writing image sha256:90d0841244d56545aca402499174c10a662b39847e07a55cec026c2d336fa79d                      0.0s
 => => naming to docker.io/library/customnginx                                                                    0.0s
root@vm-docker-nonprod-ci-01:/home/hpcuser1/project1# docker images
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
customnginx   latest    90d0841244d5   About a minute ago   192MB
nextcloud     latest    3fe93cf87aa0   4 weeks ago          1.27GB
nginx         latest    f876bfc1cc63   6 weeks ago          192MB
wordpress     latest    f4c026a8ee03   7 weeks ago          700MB
root@vm-docker-nonprod-ci-01:/home/hpcuser1/project1# 
```



```
root@vm-docker-nonprod-ci-01:/home/hpcuser1/project1# docker images
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
customnginx   latest    90d0841244d5   About a minute ago   192MB
nextcloud     latest    3fe93cf87aa0   4 weeks ago          1.27GB
nginx         latest    f876bfc1cc63   6 weeks ago          192MB
wordpress     latest    f4c026a8ee03   7 weeks ago          700MB
root@vm-docker-nonprod-ci-01:/home/hpcuser1/project1# docker run -d -p 8084:80 customnginx
585087d083321ad875862daffbfd669338923fe5a09509777bd7f5460f037558
root@vm-docker-nonprod-ci-01:/home/hpcuser1/project1# docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS                                   NAMES
585087d08332   customnginx   "/docker-entrypoint.…"   12 seconds ago   Up 12 seconds   0.0.0.0:8084->80/tcp, [::]:8084->80/tcp   beautiful_archimedes
f49e542f7922   nextcloud     "/entrypoint.sh apac…"   13 minutes ago   Up 13 minutes   0.0.0.0:8082->80/tcp, [::]:8082->80/tcp   elated_morse
2753ef44188d   wordpress     "docker-entrypoint.s…"   17 minutes ago   Up 17 minutes   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp   pedantic_booth
da4d707a414c   wordpress     "docker-entrypoint.s…"   17 minutes ago   Created                                                 angry_dewdney
49f4dca6547a   nginx         "/docker-entrypoint.…"   17 minutes ago   Up 17 minutes   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   stoic_solomon
root@vm-docker-nonprod-ci-01:/home/hpcuser1/project1# 
```



I am creating a Dockerfile

## Scenario – Deployment of a static website

Now consider that if we have our own index.html file at same folder

Folder (Docker)>

- Dockerfile
- index.html

And I have to use this index.html file as default page at /usr/share/nginx/html/index.html

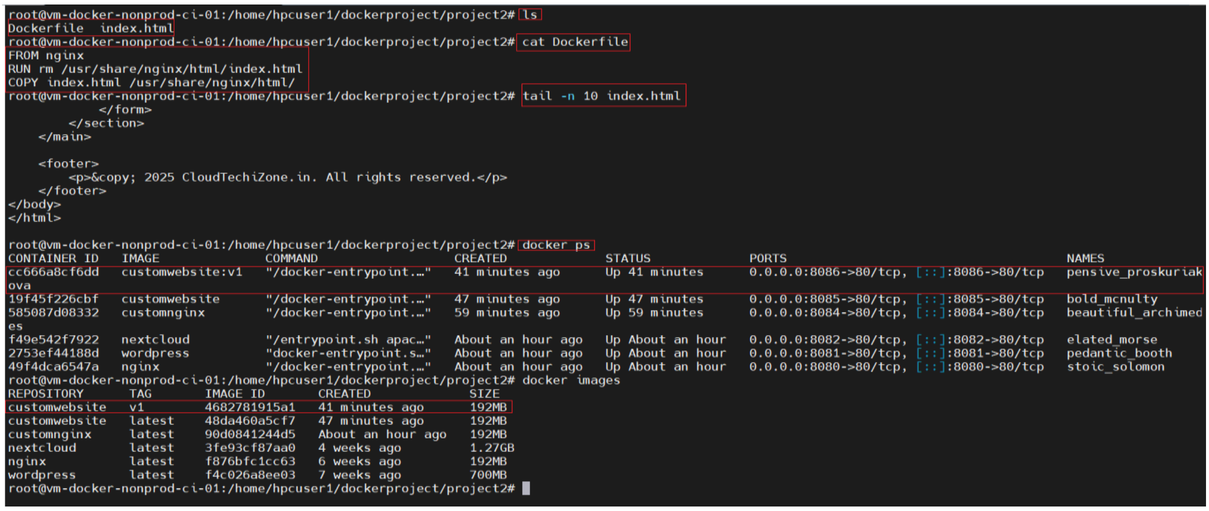
Now exactly the scenario is like that we have to copy this index.html file to the path of temporary container to path /usr/share/nginx/html/index.html

We can use to command like COPY or ADD

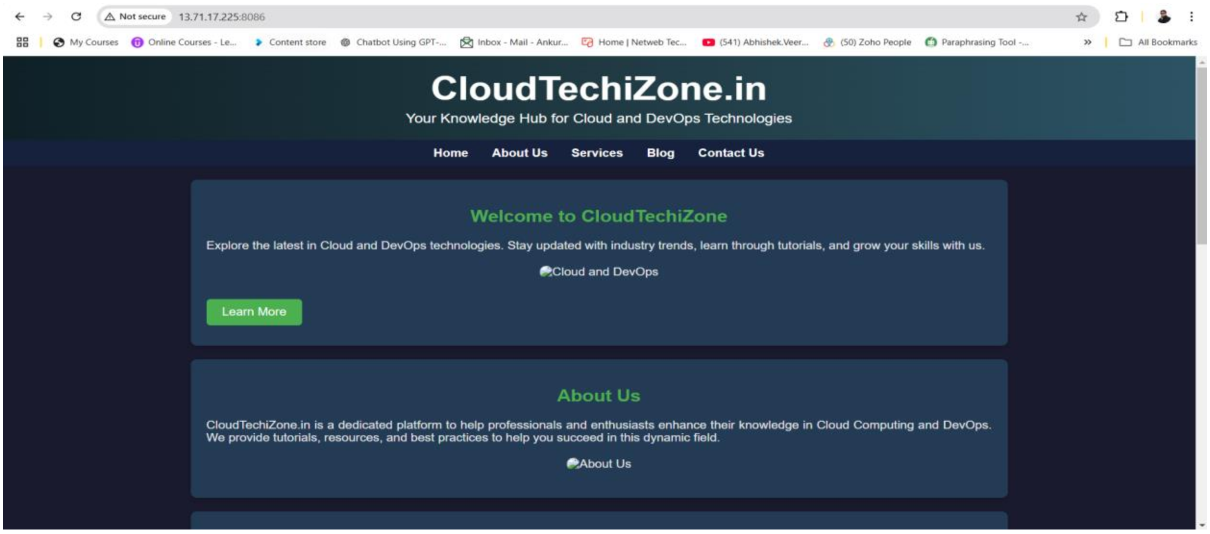
- COPY <src> path where file is kept… <dest> to the path of index.html file of temp container
- COPY index.html /usr/share/nginx/html/

**ADD or COPY**

ADD and COPY are functionally similar. COPY supports basic copying of files into the container, from the build context or from a stage in a multi-stage build. ADD supports features for fetching files from remote HTTPS and Git URLs, and extracting tar files automatically when adding files from the build context.



```
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project2# ls
Dockerfile  index.html
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project2# cat Dockerfile
FROM nginx
RUN rm /usr/share/nginx/html/index.html
COPY index.html /usr/share/nginx/html/
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project2# tail -n 10 index.html
            </form>
        </section>
    </main>

    <footer>
        <p>&copy; 2025 CloudTechiZone.in. All rights reserved.</p>
    </footer>
</body>
</html>

root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project2# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED             STATUS             PORTS                                     NAMES
cc666a8cf6dd   customwebsite:v1   "/docker-entrypoint.…"   41 minutes ago      Up 41 minutes      0.0.0.0:8086->80/tcp, [::]:8086->80/tcp   pensive_proskuriak
ova
19f45f226cbf   customwebsite      "/docker-entrypoint.…"   47 minutes ago      Up 47 minutes      0.0.0.0:8085->80/tcp, [::]:8085->80/tcp   bold_mcnulty
585087d08332   customnginx        "/docker-entrypoint.…"   59 minutes ago      Up 59 minutes      0.0.0.0:8084->80/tcp, [::]:8084->80/tcp   beautiful_archimed
es
f49e542f7922   nextcloud          "/entrypoint.sh apac…"   About an hour ago   Up About an hour   0.0.0.0:8082->80/tcp, [::]:8082->80/tcp   elated_morse
2753ef44188d   wordpress          "docker-entrypoint.s…"   About an hour ago   Up About an hour   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp   pedantic_booth
49f4dca6547a   nginx              "/docker-entrypoint.…"   About an hour ago   Up About an hour   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   stoic_solomon
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project2# docker images
REPOSITORY      TAG       IMAGE ID       CREATED             SIZE
customwebsite   v1        4682781915a1   41 minutes ago      192MB
customwebsite   latest    48da460a5cf7   47 minutes ago      192MB
customnginx     latest    90d0841244d5   About an hour ago   192MB
nextcloud       latest    3fe93cf87aa0   4 weeks ago         1.27GB
nginx           latest    f876bfc1cc63   6 weeks ago         192MB
wordpress       latest    f4c026a8ee03   7 weeks ago         700MB
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project2# 
```



CloudTechiZone.in

Your Knowledge Hub for Cloud and DevOps Technologies

Welcome to CloudTechiZone

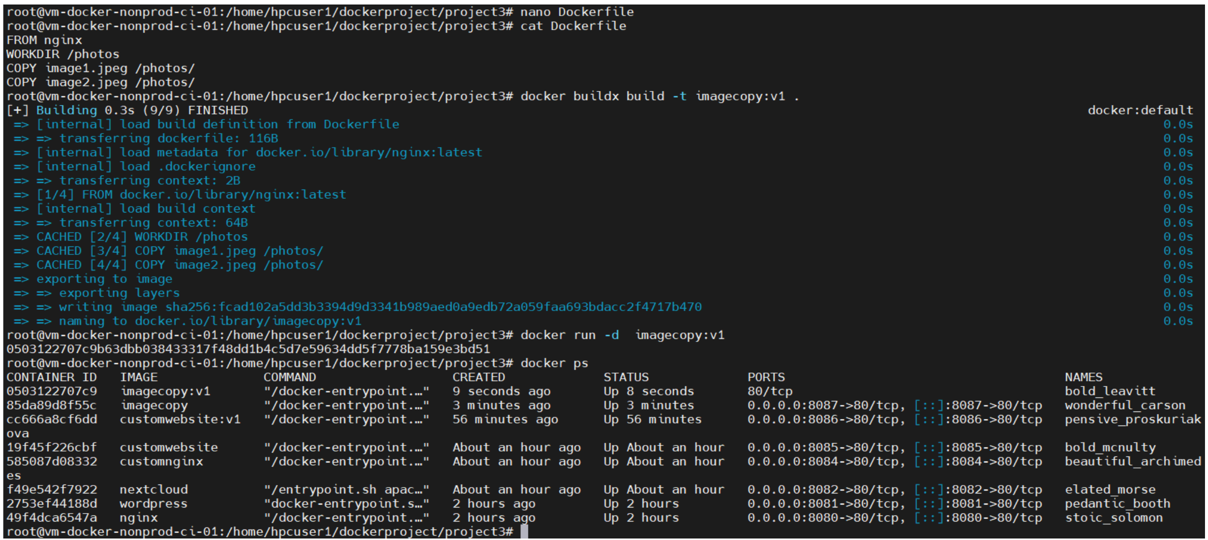
Explore the latest in Cloud and DevOps technologies. Stay updated with industry trends, learn through tutorials, and grow your skills with us.

Cloud and DevOps

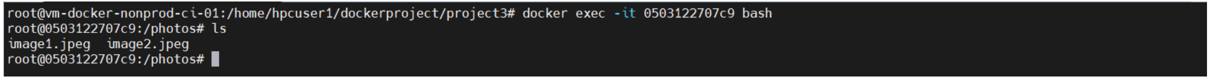
About Us

CloudTechiZone.in is a dedicated platform to help professionals and enthusiasts enhance their knowledge in Cloud Computing and DevOps. We provide tutorials, resources, and best practices to help you succeed in this dynamic field.

About Us



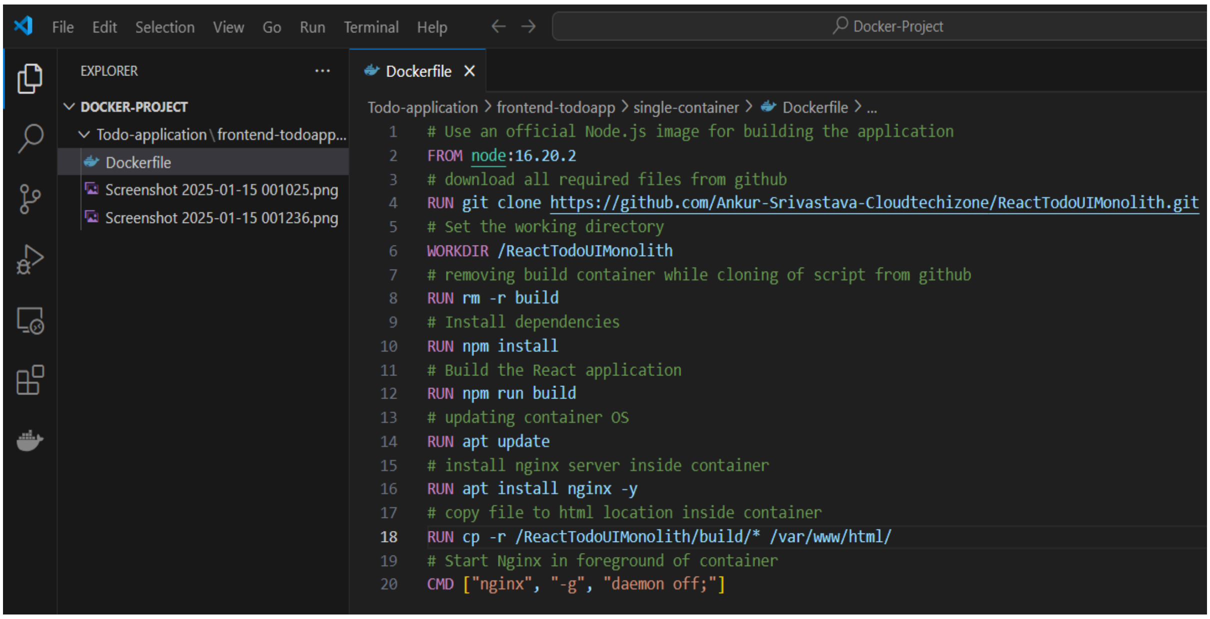
```
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project3# nano Dockerfile
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project3# cat Dockerfile
FROM nginx
WORKDIR /photos
COPY image1.jpeg /photos/
COPY image2.jpeg /photos/
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project3# docker buildx build -t imagecopy:v1 .
[+] Building 0.3s (9/9) FINISHED                                                                        docker:default
 => [internal] load build definition from Dockerfile                                                              0.0s
 => => transferring dockerfile: 116B                                                                              0.0s
 => [internal] load metadata for docker.io/library/nginx:latest                                                   0.0s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [1/4] FROM docker.io/library/nginx:latest                                                                     0.0s
 => [internal] load build context                                                                                 0.0s
 => => transferring context: 64B                                                                                  0.0s
 => CACHED [2/4] WORKDIR /photos                                                                                  0.0s
 => CACHED [3/4] COPY image1.jpeg /photos/                                                                        0.0s
 => CACHED [4/4] COPY image2.jpeg /photos/                                                                        0.0s
 => exporting to image                                                                                            0.0s
 => => exporting layers                                                                                           0.0s
 => => writing image sha256:fcad102a5dd3b3394d9d3341b989aed0a9edb72a059faa693bdacc2f4717b470                      0.0s
 => => naming to docker.io/library/imagecopy:v1                                                                   0.0s
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project3# docker run -d  imagecopy:v1
0503122707c9b63dbb038433317f48dd1b4c5d7e59634dd5f7778ba159e3bd51
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project3# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED             STATUS             PORTS                                     NAMES
0503122707c9   imagecopy:v1       "/docker-entrypoint.…"   9 seconds ago       Up 8 seconds       80/tcp                                    bold_leavitt
85da89d8f55c   imagecopy          "/docker-entrypoint.…"   3 minutes ago       Up 3 minutes       0.0.0.0:8087->80/tcp, [::]:8087->80/tcp   wonderful_carson
cc666a8cf6dd   customwebsite:v1   "/docker-entrypoint.…"   56 minutes ago      Up 56 minutes      0.0.0.0:8086->80/tcp, [::]:8086->80/tcp   pensive_proskuriak
ova
19f45f226cbf   customwebsite      "/docker-entrypoint.…"   About an hour ago   Up About an hour   0.0.0.0:8085->80/tcp, [::]:8085->80/tcp   bold_mcnulty
585087d08332   customnginx        "/docker-entrypoint.…"   About an hour ago   Up About an hour   0.0.0.0:8084->80/tcp, [::]:8084->80/tcp   beautiful_archimed
es
f49e542f7922   nextcloud          "/entrypoint.sh apac…"   About an hour ago   Up About an hour   0.0.0.0:8082->80/tcp, [::]:8082->80/tcp   elated_morse
2753ef44188d   wordpress          "docker-entrypoint.s…"   2 hours ago         Up 2 hours         0.0.0.0:8081->80/tcp, [::]:8081->80/tcp   pedantic_booth
49f4dca6547a   nginx              "/docker-entrypoint.…"   2 hours ago         Up 2 hours         0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   stoic_solomon
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project3# 
```



```
root@vm-docker-nonprod-ci-01:/home/hpcuser1/dockerproject/project3# docker exec -it 0503122707c9 bash
root@0503122707c9:/photos# ls
image1.jpeg  image2.jpeg
root@0503122707c9:/photos# 
```

## Scenario – Todo frontend application installation

First go the folder in which we have to deploy frontend app and write a Dockerfile as below


File
Edit
Selection
View
Go
Run
Terminal
Help
Docker-Project
EXPLORER
DOCKER-PROJECT
Todo-application \ frontend-todoapp...
Dockerfile
Screenshot 2025-01-15 001025.png
Screenshot 2025-01-15 001236.png
Dockerfile
Todo-application > frontend-todoapp > single-container > Dockerfile > ...

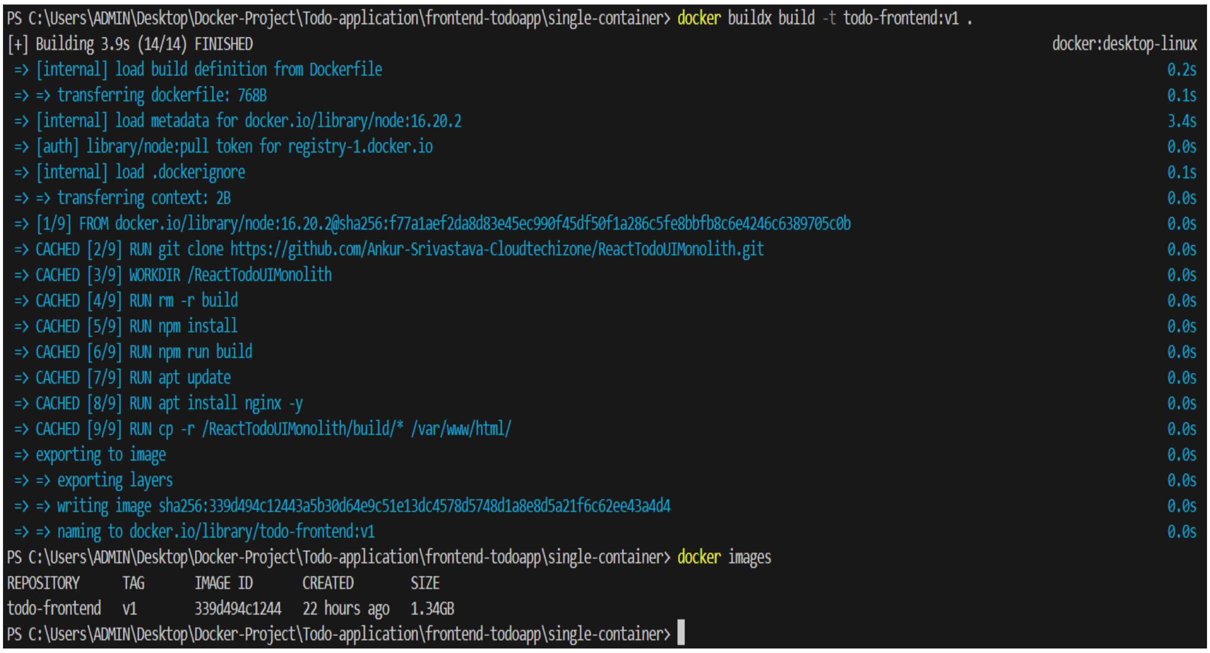
```
# Use an official Node.js image for building the application
FROM node:16.20.2
# download all required files from github
RUN git clone https://github.com/Ankur-Srivastava-Cloudtechizone/ReactTodoUIMonolith.git
# Set the working directory
WORKDIR /ReactTodoUIMonolith
# removing build container while cloning of script from github
RUN rm -r build
# Install dependencies
RUN npm install
# Build the React application
RUN npm run build
# updating container OS
RUN apt update
# install nginx server inside container
RUN apt install nginx -y
# copy file to html location inside container
RUN cp -r /ReactTodoUIMonolith/build/* /var/www/html/
# Start Nginx in foreground of container
CMD ["nginx", "-g", "daemon off;"]
```

Here's a line-by-line explanation of the Dockerfile:

1. **FROM node:16.20.2**:
   - This line specifies the base image for the Docker container, which is the Node.js version 16.20.2 image. This base image includes Node.js and npm (Node Package Manager), which are necessary to build and run the React application.
2. **RUN git clone https://github.com/Ankur-Srivastava-Cloudtechizone/ReactTodoUIMonolith.git**:
   - This command clones the ReactTodoUIMonolith Git repository from the specified URL into the container's filesystem. The repository contains the source code for the React application.
3. **WORKDIR /ReactTodoUIMonolith**:
   - This sets the working directory for subsequent commands to /ReactTodoUIMonolith, where the repository was cloned. All following commands will be executed in this directory unless otherwise specified.
4. **RUN rm -r build**:
   - This removes the existing build directory if it exists. The build directory typically contains the compiled output of a React application from previous builds.
5. **RUN npm install**:
   - This installs the dependencies listed in the package.json file of the React application. npm install downloads all the required packages into the node_modules directory.
6. **RUN npm run build**:
   - This runs the build script defined in the package.json file, which compiles the React application into static files for production. The output is typically placed in a build directory.
7. **RUN apt update**:
   - This updates the package list for the apt package manager to ensure it has the latest information about available packages and their versions.
8. **RUN apt install nginx -y**:
   - This installs the Nginx web server in the container. The -y flag automatically answers 'yes' to any prompts during the installation process.
9. **RUN cp -r /ReactTodoUIMonolith/build/* /var/www/html/**:
   - This copies all files from the build directory of the React application to the /var/www/html/ directory, which is the default root directory for serving web content in Nginx.
10. **CMD ["nginx", "-g", "daemon off;"]**:
    - This specifies the command to run when the container starts. It runs Nginx in the **foreground** (non-daemon mode) to keep the container running and serving the application.

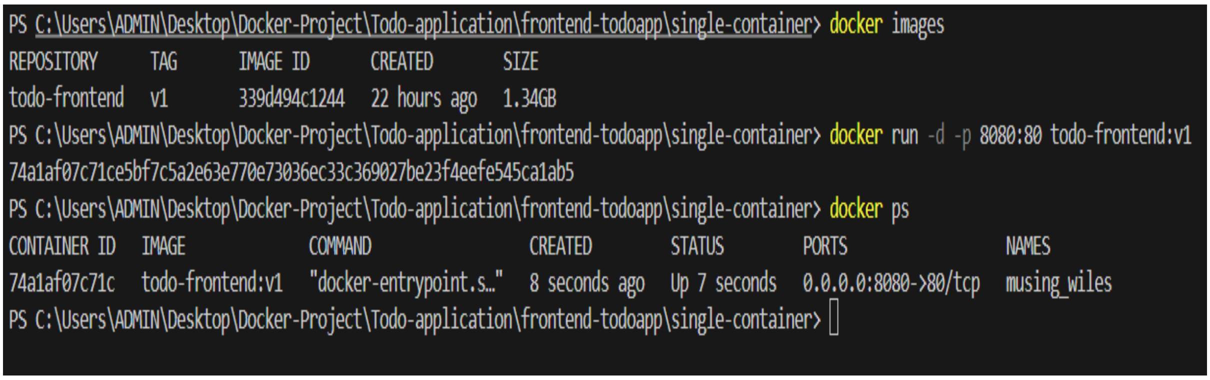
This Dockerfile sets up a containerized environment where a React application is built and served using Nginx.

After creating this Dockerfile now we will create an image from this Dockerfile



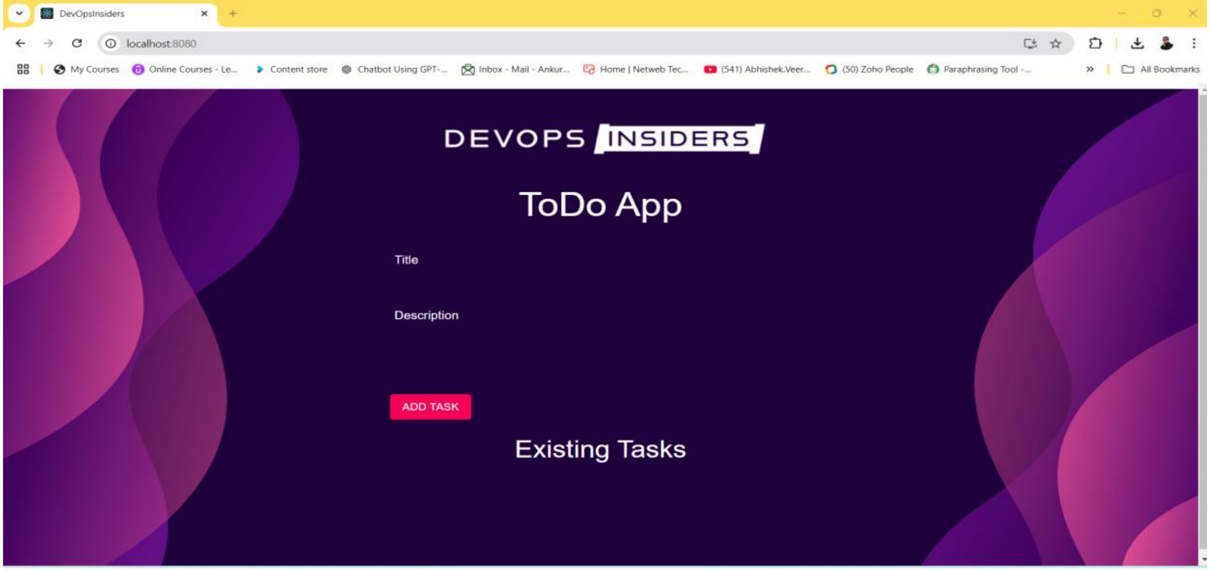
```
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\single-container> docker buildx build -t todo-frontend:v1 .
[+] Building 3.9s (14/14) FINISHED                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                0.2s
 => => transferring dockerfile: 768B                                                                                0.1s
 => [internal] load metadata for docker.io/library/node:16.20.2                                                     3.4s
 => [auth] library/node:pull token for registry-1.docker.io                                                         0.0s
 => [internal] load .dockerignore                                                                                   0.1s
 => => transferring context: 2B                                                                                     0.0s
 => [1/9] FROM docker.io/library/node:16.20.2@sha256:f77a1aef2da8d83e45ec990f45df50f1a286c5fe8bbfb8c6e4246c6389705c0b 0.0s
 => CACHED [2/9] RUN git clone https://github.com/Ankur-Srivastava-Cloudtechizone/ReactTodoUIMonolith.git           0.0s
 => CACHED [3/9] WORKDIR /ReactTodoUIMonolith                                                                       0.0s
 => CACHED [4/9] RUN rm -r build                                                                                    0.0s
 => CACHED [5/9] RUN npm install                                                                                    0.0s
 => CACHED [6/9] RUN npm run build                                                                                  0.0s
 => CACHED [7/9] RUN apt update                                                                                     0.0s
 => CACHED [8/9] RUN apt install nginx -y                                                                           0.0s
 => CACHED [9/9] RUN cp -r /ReactTodoUIMonolith/build/* /var/www/html/                                              0.0s
 => exporting to image                                                                                              0.0s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:339d494c12443a5b30d64e9c51e13dc4578d5748d1a8e8d5a21f6c62ee43a4d4                        0.0s
 => => naming to docker.io/library/todo-frontend:v1                                                                 0.0s
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\single-container> docker images
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
todo-frontend   v1        339d494c1244   22 hours ago   1.34GB
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\single-container>
```

After that we will create a container on which application is running with specified port



```
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\single-container> docker images
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
todo-frontend   v1        339d494c1244   22 hours ago   1.34GB
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\single-container> docker run -d -p 8080:80 todo-frontend:v1
74a1af07c71ce5bf7c5a2e63e770e73036ec33c369027be23f4eefe545ca1ab5
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\single-container> docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS         PORTS                  NAMES
74a1af07c71c   todo-frontend:v1   "docker-entrypoint.s…"   8 seconds ago   Up 7 seconds   0.0.0.0:8080->80/tcp   musing_wiles
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\single-container>
```

Application running browsed in a browser



**Note:-** In general, when running applications in a virtual machine (VM), they tend to run in the background. However, when using containers, the application should typically run in the foreground, which is why the CMD instruction is used in Dockerfiles. This ensures the primary process of the container remains active.

For example, the following Docker command runs a container in detached mode and executes the ls command to list items in the container's current directory:

- docker run -d -p 8080:80 containername ls

In this case, ls is the command being executed. However, running a command like ls with docker run will override the default command specified with CMD in the Dockerfile, causing the original process to stop.

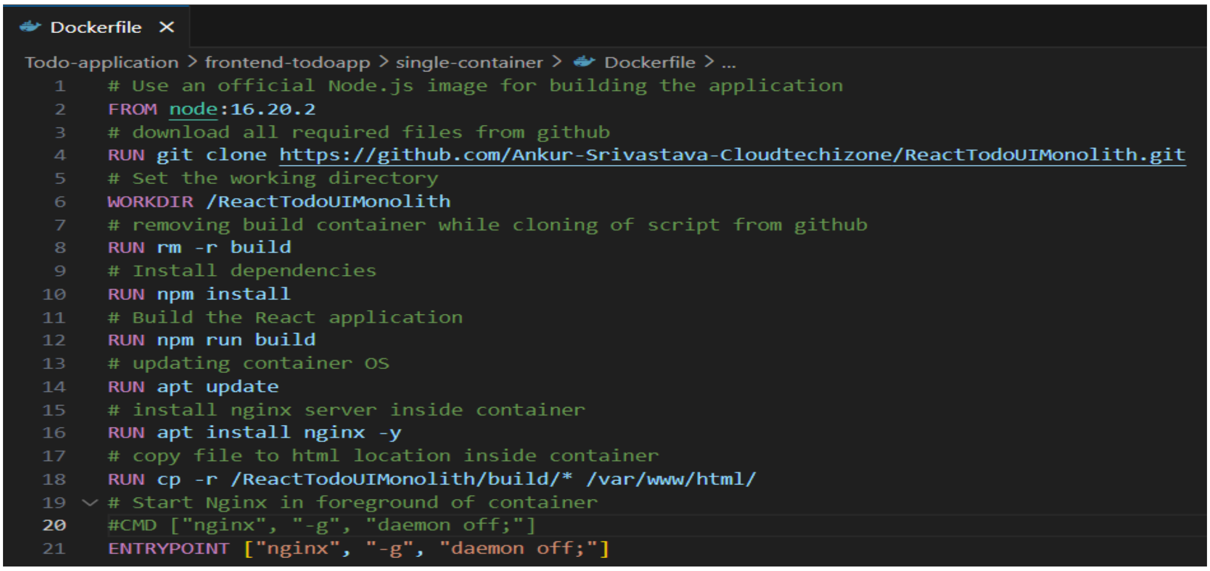
To avoid this issue and ensure that the primary process specified with CMD continues running even when additional commands are passed to docker run, you can use ENTRYPOINT. ENTRYPOINT allows you to define a fixed command or script that will always run, and any additional arguments provided with docker run will be appended to this command.

For example, in a Dockerfile:

- ENTRYPOINT ["your_command_or_script"]

With this setup, you can run:

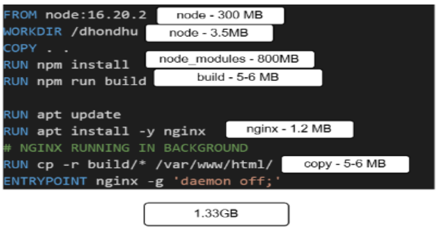
- docker run -d -p 8080:80 containername additional_command


Dockerfile ✕
Todo-application > frontend-todoapp > single-container > Dockerfile > ...

```
# Use an official Node.js image for building the application
FROM node:16.20.2
# download all required files from github
RUN git clone https://github.com/Ankur-Srivastava-Cloudtechizone/ReactTodoUIMonolith.git
# Set the working directory
WORKDIR /ReactTodoUIMonolith
# removing build container while cloning of script from github
RUN rm -r build
# Install dependencies
RUN npm install
# Build the React application
RUN npm run build
# updating container OS
RUN apt update
# install nginx server inside container
RUN apt install nginx -y
# copy file to html location inside container
RUN cp -r /ReactTodoUIMonolith/build/* /var/www/html/
# Start Nginx in foreground of container
#CMD ["nginx", "-g", "daemon off;"]
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

# Multistage Dockerfile

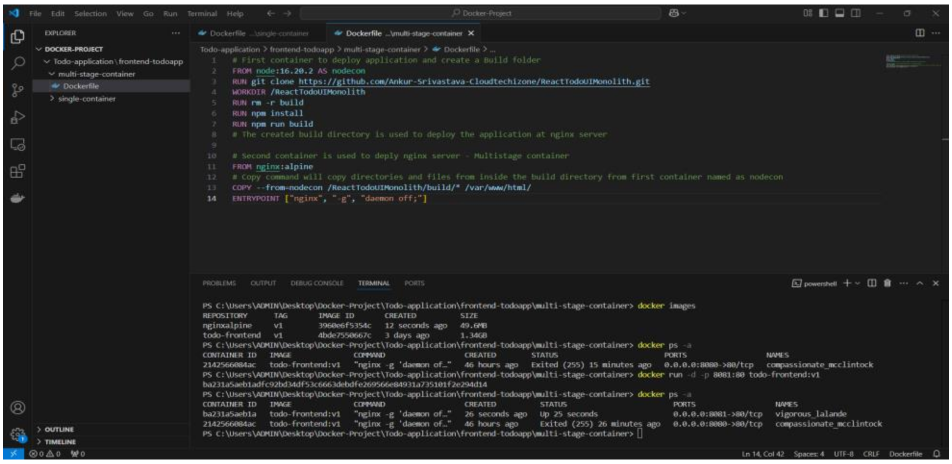
A multistage Dockerfile allows you to build a Docker image in multiple stages, optimizing the final image size by copying only necessary files from the build stages.



```
FROM node:16.20.2
WORKDIR /dhondhu
COPY . .
RUN npm install
RUN npm run build

RUN apt update
RUN apt install -y nginx
# NGINX RUNNING IN BACKGROUND
RUN cp -r build/* /var/www/html/
ENTRYPOINT nginx -g 'daemon off;'
```

The earlier application we created is of 1.33GB size which is big as per container and we are creating these container to be of small size so we will optimize the image by writing Multistage Dockerfile

In first container which is temporary and will deploy a build directory containing application and that will be copied to second container deployed with nginx server.



Todo-application > frontend-todoapp > multi-stage-container > Dockerfile > ...

```dockerfile
# First container to deploy application and create a Build folder
FROM node:16.20.2 AS nodecon
RUN git clone https://github.com/Ankur-Srivastava-Cloudtechizone/ReactTodoUIMonolith.git
WORKDIR /ReactTodoUIMonolith
RUN rm -r build
RUN npm install
RUN npm run build
# The created build directory is used to deploy the application at nginx server

# Second container is used to deply nginx server - Multistage container
FROM nginx:alpine
# Copy command will copy directories and files from inside the build directory from first container named as nodecon
COPY --from=nodecon /ReactTodoUIMonolith/build/* /var/www/html/
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

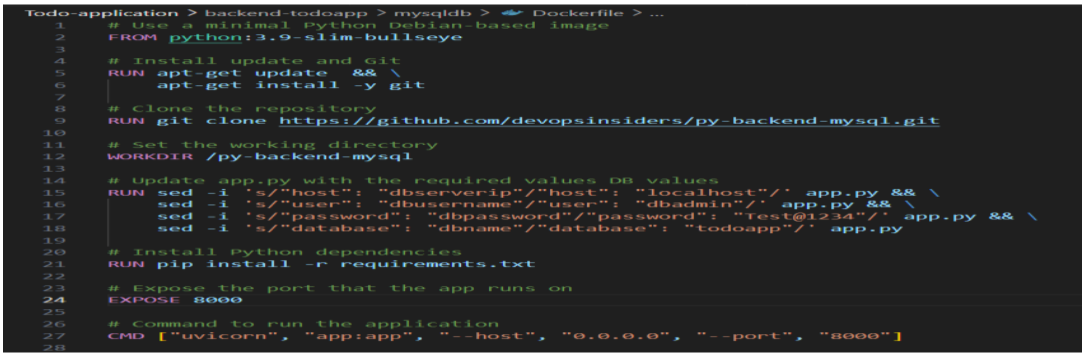
```
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\multi-stage-container> docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
nginxalpine     v1        3960e6f5354c   12 seconds ago   49.6MB
todo-frontend   v1        4bde7550667c   3 days ago       1.34GB
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\multi-stage-container> docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS                        PORTS                  NAMES
2142566084ac   todo-frontend:v1   "nginx -g 'daemon of…"   46 hours ago   Exited (255) 15 minutes ago   0.0.0.0:8080->80/tcp   compassionate_mcclintock
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\multi-stage-container> docker run -d -p 8081:80 todo-frontend:v1
ba231a5aeb1adfc92bd34df53c6663debdfe269566e84931a735101f2e294d14
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\multi-stage-container> docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS                        PORTS                  NAMES
ba231a5aeb1a   todo-frontend:v1   "nginx -g 'daemon of…"   26 seconds ago   Up 25 seconds                 0.0.0.0:8081->80/tcp   vigorous_lalande
2142566084ac   todo-frontend:v1   "nginx -g 'daemon of…"   46 hours ago     Exited (255) 26 minutes ago   0.0.0.0:8080->80/tcp   compassionate_mcclintock
PS C:\Users\ADMIN\Desktop\Docker-Project\Todo-application\frontend-todoapp\multi-stage-container>
```

File Edit Selection View Go Run Terminal Help
Docker-Project
EXPLORER
DOCKER-PROJECT
Todo-application \ frontend-todoapp
multi-stage-container
Dockerfile
single-container
Dockerfile ...\single-container
Dockerfile ...\multi-stage-container
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
OUTLINE
TIMELINE
Ln 14, Col 42 Spaces: 4 UTF-8 CRLF Dockerfile

Installing and creating a mysql DB

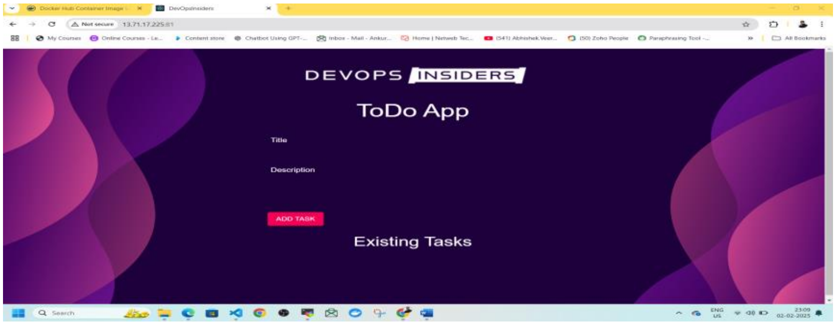
- docker run --name some-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=Test@12345 -e MYSQL_PASSWORD=Test@1234 -e MYSQL_USER=dbadmin -e MYSQL_DATABASE=todoapp -d mysql

Backend



Todo-application > backend-todoapp > mysqldb > Dockerfile > ...

```dockerfile
# Use a minimal Python Debian-based image
FROM python:3.9-slim-bullseye

# Install update and Git
RUN apt-get update  && \
    apt-get install -y git

# Clone the repository
RUN git clone https://github.com/devopsinsiders/py-backend-mysql.git

# Set the working directory
WORKDIR /py-backend-mysql

# Update app.py with the required values DB values
RUN sed -i 's/"host": "dbserverip"/"host": "localhost"/' app.py && \
    sed -i 's/"user": "dbusername"/"user": "dbadmin"/' app.py && \
    sed -i 's/"password": "dbpassword"/"password": "Test@1234"/' app.py && \
    sed -i 's/"database": "dbname"/"database": "todoapp"/' app.py

# Install Python dependencies
RUN pip install -r requirements.txt

# Expose the port that the app runs on
EXPOSE 8000

# Command to run the application
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```



As we configure both Frontend and Backend with database server individually but we need to establish a communication between frontend container and backend container. For this first we need to the concept of Docker Networking.

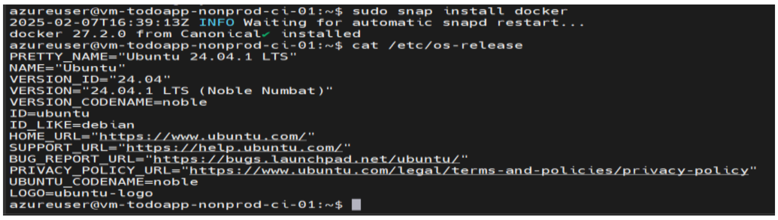
# Docker Networking

Docker networking is like setting up communication between different containers, just like how devices connect in a home or office network. Think of Docker containers as small virtual computers running different applications. These containers need a way to talk to each other or the outside world, and Docker provides different networking options to make that happen.

Here’s a simple breakdown:

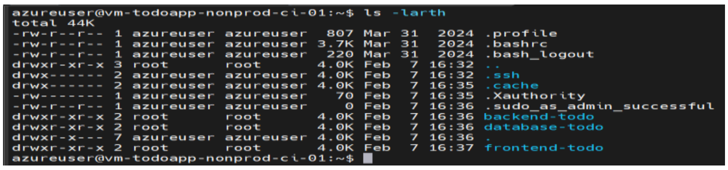
1. **Bridge Network (Default)** – Imagine a Wi-Fi router at home. When you start a new container, Docker automatically connects it to a private network (like your home Wi-Fi) so it can talk to other containers on the same network. But from outside (the internet), you need special settings (port forwarding) to access it.
2. **Host Network** – This is like directly plugging your computer into the main internet line. The container shares the same network as the host machine, so there's no isolation. This makes things faster but less secure.
3. **Overlay Network** – This is like a company VPN. It allows containers running on different physical or cloud machines to communicate as if they were on the same private network. Used mainly in Docker Swarm for multi-host networking.
4. **Macvlan Network** – Think of giving each container its own IP address, just like each device in your house gets a unique address from the internet provider. This is useful when containers need to behave like real devices on a physical network.
5. **None Network** – This is like disconnecting a device from the internet. The container is isolated and can't communicate with anything unless explicitly configured.

To understand the clear concept of Docker networking we are going to take a Ubuntu VM where I am installing Docker



```
azureuser@vm-todoapp-nonprod-ci-01:~$ sudo snap install docker
2025-02-07T16:39:13Z INFO Waiting for automatic snapd restart...
docker 27.2.0 from Canonical✓ installed
azureuser@vm-todoapp-nonprod-ci-01:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
azureuser@vm-todoapp-nonprod-ci-01:~$
```

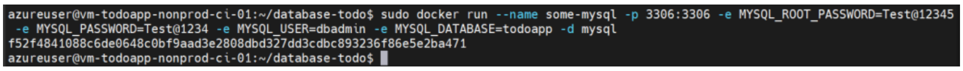
Now I will create 3 directories as Frontend, Backend & Database. Will deploy all three services as per requirement



```
azureuser@vm-todoapp-nonprod-ci-01:~$ ls -larth
total 44K
-rw-r--r-- 1 azureuser azureuser  807 Mar 31  2024 .profile
-rw-r--r-- 1 azureuser azureuser 3.7K Mar 31  2024 .bashrc
-rw-r--r-- 1 azureuser azureuser  220 Mar 31  2024 .bash_logout
drwxr-xr-x 3 root      root      4.0K Feb  7 16:32 ..
drwx------ 2 azureuser azureuser 4.0K Feb  7 16:32 .ssh
drwx------ 2 azureuser azureuser 4.0K Feb  7 16:35 .cache
-rw------- 1 azureuser azureuser   70 Feb  7 16:35 .Xauthority
-rw-r--r-- 1 azureuser azureuser    0 Feb  7 16:36 .sudo_as_admin_successful
drwxr-xr-x 2 root      root      4.0K Feb  7 16:36 backend-todo
drwxr-xr-x 2 root      root      4.0K Feb  7 16:36 database-todo
drwxr-x--- 7 azureuser azureuser 4.0K Feb  7 16:36 .
drwxr-xr-x 2 root      root      4.0K Feb  7 16:37 frontend-todo
azureuser@vm-todoapp-nonprod-ci-01:~$
```

First I will deploy a MySQL database in desired database directory

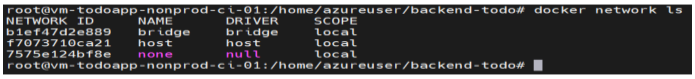
- docker run --name some-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=Test@12345 -e MYSQL_PASSWORD=Test@1234 -e MYSQL_USER=dbadmin -e MYSQL_DATABASE=todoapp -d mysql



```
azureuser@vm-todoapp-nonprod-ci-01:~/database-todo$ sudo docker run --name some-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=Test@12345
 -e MYSQL_PASSWORD=Test@1234 -e MYSQL_USER=dbadmin -e MYSQL_DATABASE=todoapp -d mysql
f52f4841088c6de0648c0bf9aad3e2808dbd327dd3cdbc893236f86e5e2ba471
azureuser@vm-todoapp-nonprod-ci-01:~/database-todo$
```

Now I have to use this DB container network IP in backend so that DB connects to backend application for this we have to see network IP of DB docker

To see current network devices installed by default is as



```
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
b1ef47d2e889   bridge    bridge    local
f7073710ca21   host      host      local
7575e124bf8e   none      null      local
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo#
```

By default, Docker creates three network drivers upon installation: **bridge**, **host**, and **none**.

- The **bridge** network driver enables communication between containers within the same Docker network. Containers connected to this network can communicate using the internal Docker network CIDR.

- The **host** network driver allows containers to share the host machine's network stack, making them directly accessible from the VM or server where Docker is running.

- The **none** network driver isolates the container from any network, effectively disabling networking for that container.

# Isolated Network

## Bridge

First, let's explore how the bridge network works.

Since we've already deployed the database (DB) container, we'll use its IP address in the backend container. Similarly, we'll use the backend container's IP in the frontend application container.

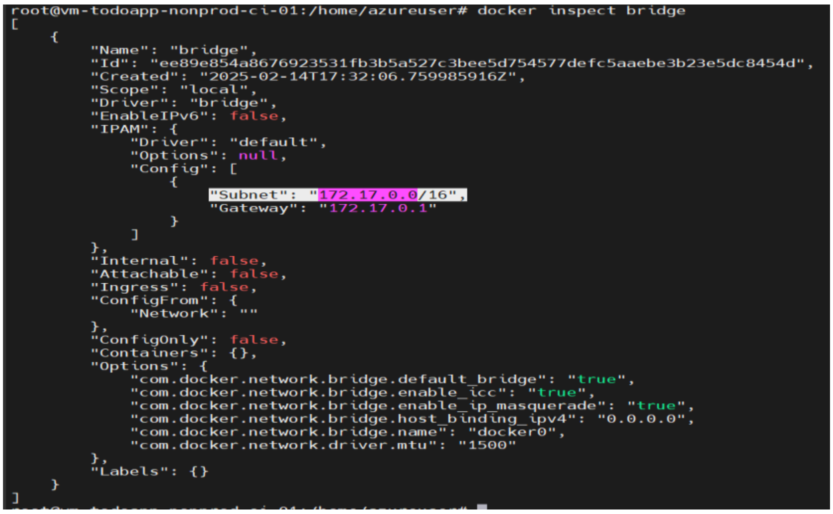
Since we're using a bridge network driver, we'll also deploy a browser container. We'll then access the application through the VM's IP on a specific port to validate connectivity.

**Note: -** Docker networks created using CIDR operate in an isolated environment. Any container connected to such a network remains within that isolated space and does not interact with external networks, such as an Azure virtual network, unless explicitly configured to do so (e.g., via port forwarding, bridge networking, or additional routing rules). Bridge network uses bridge driver.

If we want to see all IP's of container in bridge network than we can run below command

- docker inspect bridge

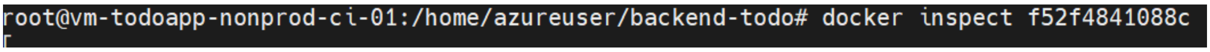
it will also give the subnet range used for that bridge network



```
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker inspect bridge
[
    {
        "Name": "bridge",
        "Id": "ee89e854a8676923531fb3b5a527c3bee5d754577defc5aaebe3b23e5dc8454d",
        "Created": "2025-02-14T17:32:06.759985916Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
root@vm-todoapp-nonprod-ci-01:/home/azureuser#
```

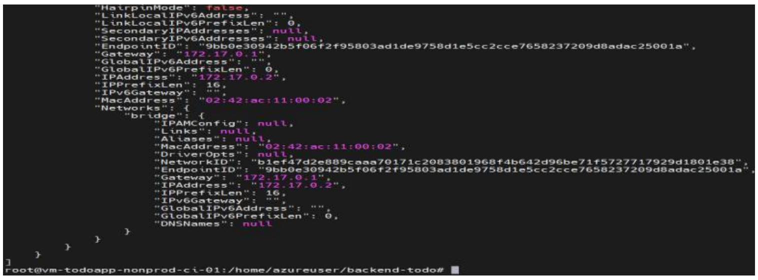
Now to see DB container IP

- docker inspect containerID



```
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo# docker inspect f52f4841088c
[
```

In last you will find the container IP



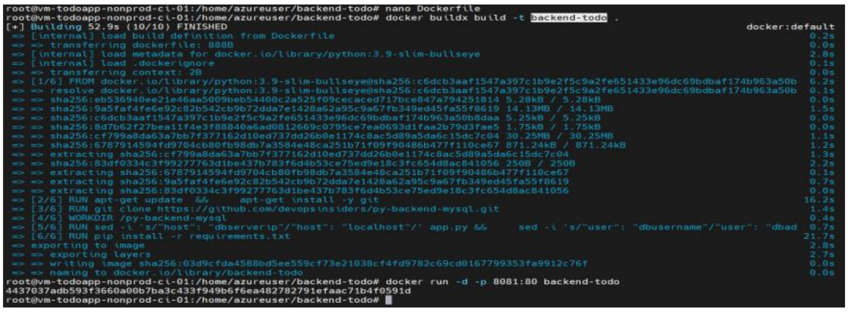
```
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "9bb0e30942b5f06f2f95803ad1de9758d1e5cc2cce7658237209d8adac25001a",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null,
                    "NetworkID": "b1ef47d2e889caaa70171c2083801968f4b642d96be71f5727717929d1801e38",
                    "EndpointID": "9bb0e30942b5f06f2f95803ad1de9758d1e5cc2cce7658237209d8adac25001a",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DNSNames": null
                }
            }
        }
    }
]
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo#
```

We will use this IP Address: 172.17.0.2 to connect DB to backend application we will deploy.

Now go inside backend application folder we will create a Dockerfile and after that create an Image and docker container

Backend Dockerfile

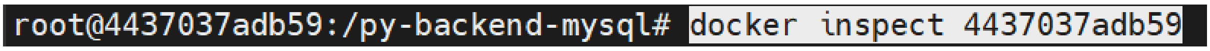
```
# Use a minimal Python Debian-based image
FROM python:3.9-slim-bullseye

# Install update and Git
RUN apt-get update  && \
        apt-get install -y git

# Clone the repository
RUN git clone https://github.com/devopsinsiders/py-backend-mysql.git

# Set the working directory
WORKDIR /py-backend-mysql

# Update app.py with the required values DB values
RUN sed -i 's/"host": "dbserverip"/"host": "172.17.0.2"/' app.py && \
    sed -i 's/"user": "dbusername"/"user": "dbadmin"/' app.py && \
    sed -i 's/"password": "dbpassword"/"password": "Test@1234"/' app.py && \
    sed -i 's/"database": "dbname"/"database": "todoapp"/' app.py

# Install Python dependencies
RUN pip install -r requirements.txt

# Expose the port that the app runs on
EXPOSE 8000

# Command to run the application
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```



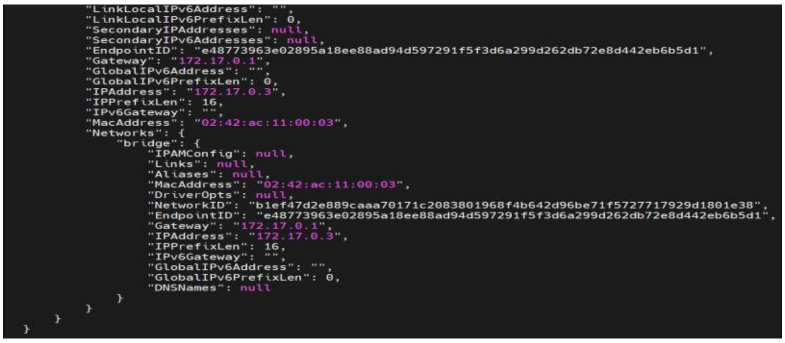
```
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo# nano Dockerfile
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo# docker buildx build -t backend-todo .
[+] Building 52.9s (10/10) FINISHED                                                                                   docker:default
 => [internal] load build definition from Dockerfile                                                                            0.2s
 => => transferring dockerfile: 888B                                                                                            0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim-bullseye                                                     2.8s
 => [internal] load .dockerignore                                                                                               0.1s
 => => transferring context: 2B                                                                                                 0.0s
 => [1/6] FROM docker.io/library/python:3.9-slim-bullseye@sha256:c6dcb3aaf1547a397c1b9e2f5c9a2fe651433e96dc69bdbaf174b963a50b   6.2s
 => => resolve docker.io/library/python:3.9-slim-bullseye@sha256:c6dcb3aaf1547a397c1b9e2f5c9a2fe651433e96dc69bdbaf174b963a50b   0.1s
 => => sha256:eb536940ee21e46aa5009beb54400c2a525f09cecaced717bce847a794251814 5.28kB / 5.28kB                                  0.0s
 => => sha256:9a5faf4fe6e92c82b542cb9b72dda7e1428a62a95c9a67fb349ed45fa55f8619 14.13MB / 14.13MB                                1.5s
 => => sha256:c6dcb3aaf1547a397c1b9e2f5c9a2fe651433e96dc69bdbaf174b963a50b8daa 5.25kB / 5.25kB                                  0.0s
 => => sha256:8d7b62f27bea11f4e3f88840a6ad0812669c0795ce7ea0693d1faa2b79d3fae5 1.75kB / 1.75kB                                  0.0s
 => => sha256:cf799a8da63a7bb7f377162d10ed737dd26b0e1174c8ac5d89a5da6c15dc7c04 30.25MB / 30.25MB                                1.1s
 => => sha256:6787914594fd9704cb80fb98db7a3584e48ca251b71f09f90486b477f110ce67 871.24kB / 871.24kB                              1.2s
 => => extracting sha256:cf799a8da63a7bb7f377162d10ed737dd26b0e1174c8ac5d89a5da6c15dc7c04                                       1.3s
 => => sha256:83df0334c3f99277763d1be437b783f6d4b53ce75ed9e18c3fc654d8ac841056 250B / 250B                                      2.2s
 => => extracting sha256:6787914594fd9704cb80fb98db7a3584e48ca251b71f09f90486b477f110ce67                                       0.1s
 => => extracting sha256:9a5faf4fe6e92c82b542cb9b72dda7e1428a62a95c9a67fb349ed45fa55f8619                                       0.7s
 => => extracting sha256:83df0334c3f99277763d1be437b783f6d4b53ce75ed9e18c3fc654d8ac841056                                       0.0s
 => [2/6] RUN apt-get update  &&     apt-get install -y git                                                                    16.2s
 => [3/6] RUN git clone https://github.com/devopsinsiders/py-backend-mysql.git                                                  1.4s
 => [4/6] WORKDIR /py-backend-mysql                                                                                             0.4s
 => [5/6] RUN sed -i 's/"host": "dbserverip"/"host": "localhost"/' app.py &&     sed -i 's/"user": "dbusername"/"user": "dbad   0.7s
 => [6/6] RUN pip install -r requirements.txt                                                                                  21.7s
 => exporting to image                                                                                                          2.8s
 => => exporting layers                                                                                                         2.7s
 => => writing image sha256:03d9cfda4588bd5ee559cf73e21038cf4fd9782c69cd0167799353fa9912c76f                                    0.0s
 => => naming to docker.io/library/backend-todo                                                                                 0.0s
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo# docker run -d -p 8081:80 backend-todo
4437037adb593f3660a00b7ba3c433f949b6f6ea482782791efaac71b4f0591d
root@vm-todoapp-nonprod-ci-01:/home/azureuser/backend-todo#
```

Now to see the network IP of backend application

- Docker inspect containerIP



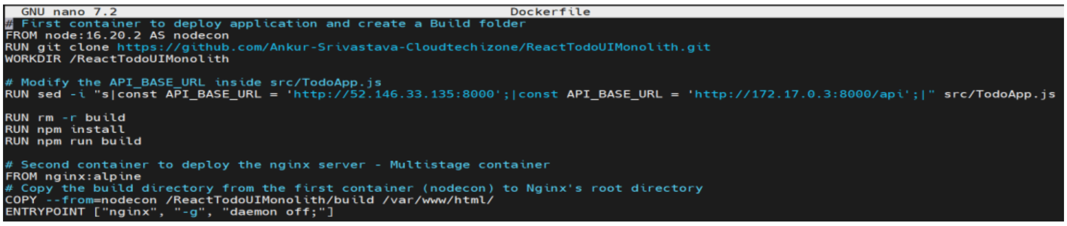
```
root@4437037adb59:/py-backend-mysql# docker inspect 4437037adb59
```



```
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "e48773963e02895a18ee88ad94d597291f5f3d6a299d262db72e8d442eb6b5d1",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.3",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:03",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "02:42:ac:11:00:03",
                    "DriverOpts": null,
                    "NetworkID": "b1ef47d2e889caaa70171c2083801968f4b642d96be71f5727717929d1801e38",
                    "EndpointID": "e48773963e02895a18ee88ad94d597291f5f3d6a299d262db72e8d442eb6b5d1",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.3",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DNSNames": null
                }
            }
        }
    }
```

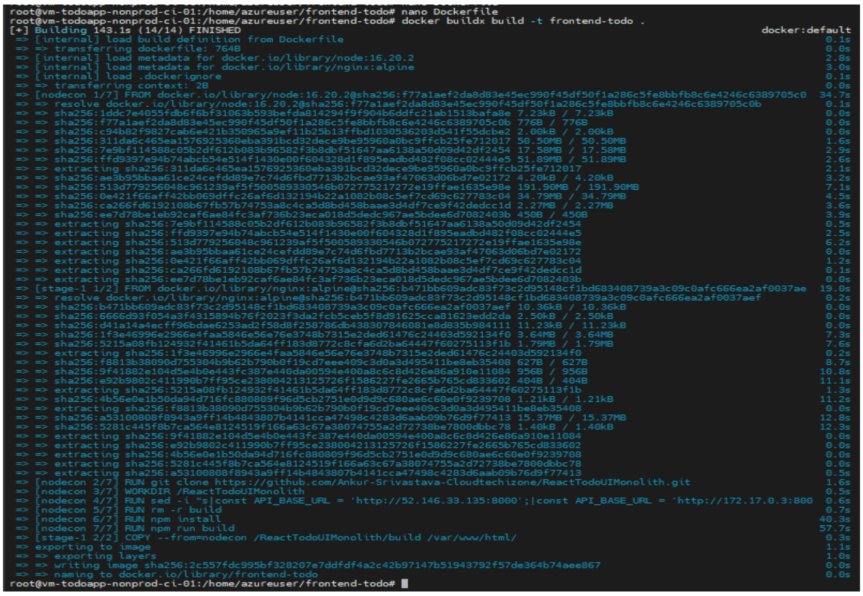
Now we establish connection between backend application and DB containers

We will deploy frontend container. Go inside the frontend directory

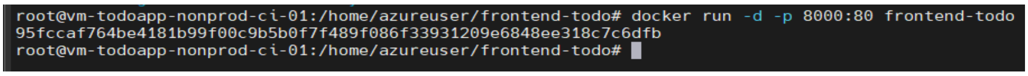


```
  GNU nano 7.2                                            Dockerfile
# First container to deploy application and create a Build folder
FROM node:16.20.2 AS nodecon
RUN git clone https://github.com/Ankur-Srivastava-Cloudtechizone/ReactTodoUIMonolith.git
WORKDIR /ReactTodoUIMonolith

# Modify the API_BASE_URL inside src/TodoApp.js
RUN sed -i "s|const API_BASE_URL = 'http://52.146.33.135:8000';|const API_BASE_URL = 'http://172.17.0.3:8000/api';|" src/TodoApp.js

RUN rm -r build
RUN npm install
RUN npm run build

# Second container to deploy the nginx server - Multistage container
FROM nginx:alpine
# Copy the build directory from the first container (nodecon) to Nginx's root directory
COPY --from=nodecon /ReactTodoUIMonolith/build /var/www/html/
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

From this Dockerfile I will create an Image and after that I will create a container for frontend app



```
root@vm-todoapp-nonprod-ci-01:/home/azureuser/frontend-todo# nano Dockerfile
root@vm-todoapp-nonprod-ci-01:/home/azureuser/frontend-todo# docker buildx build -t frontend-todo .
[+] Building 143.1s (14/14) FINISHED                                                                docker:default
 => [internal] load build definition from Dockerfile                                                          0.1s
 => => transferring dockerfile: 764B                                                                          0.0s
 => [internal] load metadata for docker.io/library/node:16.20.2                                               2.8s
 => [internal] load metadata for docker.io/library/nginx:alpine                                               3.0s
 => [internal] load .dockerignore                                                                             0.1s
 => => transferring context: 2B                                                                               0.0s
 => [nodecon 1/7] FROM docker.io/library/node:16.20.2@sha256:f77a1aef2da8d83e45ec990f45df50f1a286c5fe8bbfb8c6e4246c6389705c0  34.7s
 => => resolve docker.io/library/node:16.20.2@sha256:f77a1aef2da8d83e45ec990f45df50f1a286c5fe8bbfb8c6e4246c6389705c0b       0.1s
 => => sha256:1ddc7e4055fdb6f6bf31063b593befda814294f9f904b6ddfc21ab1513bafa8e 7.23kB / 7.23kB                0.0s
 => => sha256:f77a1aef2da8d83e45ec990f45df50f1a286c5fe8bbfb8c6e4246c6389705c0b 776B / 776B                    0.0s
 => => sha256:c94b82f9827cab6e421b350965a9ef11b25b13ffbd1030536203d541f55dcbe2 2.00kB / 2.00kB                0.0s
 => => sha256:311da6c465ea1576925360eba391bcd32dece9be95960a0bc9ffcb25fe712017 50.50MB / 50.50MB              1.6s
 => => sha256:7e9bf114588c05b2df612b083b96582f3b8dbf51647aa6138a50d09d42df2454 17.58MB / 17.58MB              2.9s
 => => sha256:ffd9397e94b74abcb54e514f1430e00f604328d1f895eadbd482f08cc02444e5 51.89MB / 51.89MB              2.6s
 => => extracting sha256:311da6c465ea1576925360eba391bcd32dece9be95960a0bc9ffcb25fe712017                     2.1s
 => => sha256:ae3b95bbaa61ce24cefdd89e7c74d6fbd7713b2bcae93af47063d06bd7e02172 4.20kB / 4.20kB                3.2s
 => => sha256:513d779256048c961239af5f500589330546b072775217272e19ffae1635e98e 191.90MB / 191.90MB            7.1s
 => => sha256:0e421f66aff42bb069dffc26af6d132194b22a1082b08c5ef7cd69c627783c04 34.79MB / 34.79MB              4.5s
 => => sha256:ca266fd6192108b67fb57b74753a8c4ca5d8bd458baae3d4df7ce9f42dedcc1d 2.27MB / 2.27MB                3.6s
 => => sha256:ee7d78be1eb92caf6ae84fc3af736b23eca018d5dedc967ae5bdee6d7082403b 450B / 450B                    3.9s
 => => extracting sha256:7e9bf114588c05b2df612b083b96582f3b8dbf51647aa6138a50d09d42df2454                     0.5s
 => => extracting sha256:ffd9397e94b74abcb54e514f1430e00f604328d1f895eadbd482f08cc02444e5                     2.5s
 => => extracting sha256:513d779256048c961239af5f500589330546b072775217272e19ffae1635e98e                     6.2s
 => => extracting sha256:ae3b95bbaa61ce24cefdd89e7c74d6fbd7713b2bcae93af47063d06bd7e02172                     0.0s
 => => extracting sha256:0e421f66aff42bb069dffc26af6d132194b22a1082b08c5ef7cd69c627783c04                     1.2s
 => => extracting sha256:ca266fd6192108b67fb57b74753a8c4ca5d8bd458baae3d4df7ce9f42dedcc1d                     0.1s
 => => extracting sha256:ee7d78be1eb92caf6ae84fc3af736b23eca018d5dedc967ae5bdee6d7082403b                     0.0s
 => [stage-1 1/2] FROM docker.io/library/nginx:alpine@sha256:b471bb609adc83f73c2d95148cf1bd683408739a3c09c0afc666ea2af0037ae  19.0s
 => => resolve docker.io/library/nginx:alpine@sha256:b471bb609adc83f73c2d95148cf1bd683408739a3c09c0afc666ea2af0037aef       0.2s
 => => sha256:b471bb609adc83f73c2d95148cf1bd683408739a3c09c0afc666ea2af0037aef 10.36kB / 10.36kB              0.0s
 => => sha256:6666d93f054a3f4315894b76f2023f3da2fcb5ceb5f8d91625cca81623edd2da 2.50kB / 2.50kB                0.0s
 => => sha256:d41a14a4ecff96bdae6253ad2f58d8f258786db438307846081e8d835b984111 11.23kB / 11.23kB              0.0s
 => => sha256:1f3e46996e2966e4faa5846e56e76e3748b7315e2ded61476c24403d592134f0 3.64MB / 3.64MB                7.3s
 => => sha256:5215a08fb124932f41461b5da64ff183d8772c8cfa6d2ba64447f60275113f1b 1.79MB / 1.79MB                7.6s
 => => extracting sha256:1f3e46996e2966e4faa5846e56e76e3748b7315e2ded61476c24403d592134f0                     0.2s
 => => sha256:f8813b38090d755304b9b62b790b0f19cd7eee409c3d0a3d495411be8eb35408 627B / 627B                    8.7s
 => => sha256:9f41882e104d5e4b0e443fc387e440da00594e400a8c6c8d426e86a910e11084 956B / 956B                   10.8s
 => => sha256:e92b9802c411990b7ff95ce238004213125726f1586227fe2665b765cd833602 404B / 404B                   11.1s
 => => extracting sha256:5215a08fb124932f41461b5da64ff183d8772c8cfa6d2ba64447f60275113f1b                     1.3s
 => => sha256:4b56e0e1b50da94d716fc880809f96d5cb2751e0d9d9c680ae6c60e0f9239708 1.21kB / 1.21kB               11.2s
 => => extracting sha256:f8813b38090d755304b9b62b790b0f19cd7eee409c3d0a3d495411be8eb35408                     0.0s
 => => sha256:a53100808f8943a9ff14b4843807b4141cca47498c4283d6aab09b76d9f77413 15.37MB / 15.37MB             12.8s
 => => sha256:5281c445f8b7ca564e8124519f166a63c67a38074755a2d72738be7800dbbc78 1.40kB / 1.40kB               12.3s
 => => extracting sha256:9f41882e104d5e4b0e443fc387e440da00594e400a8c6c8d426e86a910e11084                     0.0s
 => => extracting sha256:e92b9802c411990b7ff95ce238004213125726f1586227fe2665b765cd833602                     0.0s
 => => extracting sha256:4b56e0e1b50da94d716fc880809f96d5cb2751e0d9d9c680ae6c60e0f9239708                     0.0s
 => => extracting sha256:5281c445f8b7ca564e8124519f166a63c67a38074755a2d72738be7800dbbc78                     0.0s
 => => extracting sha256:a53100808f8943a9ff14b4843807b4141cca47498c4283d6aab09b76d9f77413                     0.5s
 => [nodecon 2/7] RUN git clone https://github.com/Ankur-Srivastava-Cloudtechizone/ReactTodoUIMonolith.git    1.6s
 => [nodecon 3/7] WORKDIR /ReactTodoUIMonolith                                                                0.5s
 => [nodecon 4/7] RUN sed -i "s|const API_BASE_URL = 'http://52.146.33.135:8000';|const API_BASE_URL = 'http://172.17.0.3:800  0.6s
 => [nodecon 5/7] RUN rm -r build                                                                             0.7s
 => [nodecon 6/7] RUN npm install                                                                            40.3s
 => [nodecon 7/7] RUN npm run build                                                                          57.7s
 => [stage-1 2/2] COPY --from=nodecon /ReactTodoUIMonolith/build /var/www/html/                               0.3s
 => exporting to image                                                                                        1.1s
 => => exporting layers                                                                                       1.0s
 => => writing image sha256:2c557fdc995bf328207e7ddfdf4a2c42b97147b51943792f57de364b74aee867                  0.0s
 => => naming to docker.io/library/frontend-todo                                                              0.0s
root@vm-todoapp-nonprod-ci-01:/home/azureuser/frontend-todo# 
```

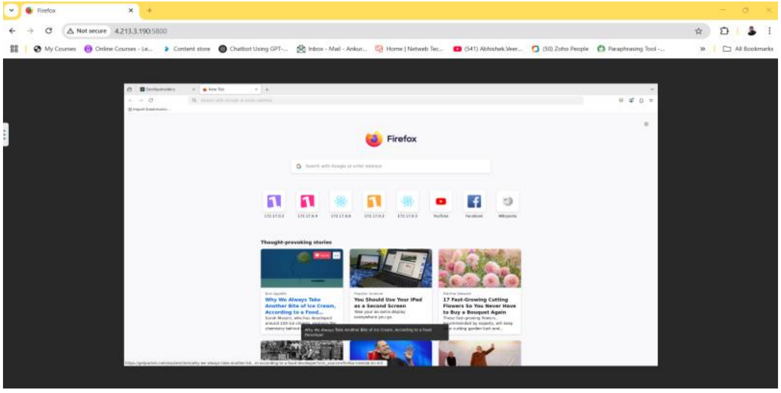


```
root@vm-todoapp-nonprod-ci-01:/home/azureuser/frontend-todo# docker run -d -p 8000:80 frontend-todo
95fccaf764be4181b99f00c9b5b0f7f489f086f33931209e6848ee318c7c6dfb
root@vm-todoapp-nonprod-ci-01:/home/azureuser/frontend-todo# 
```

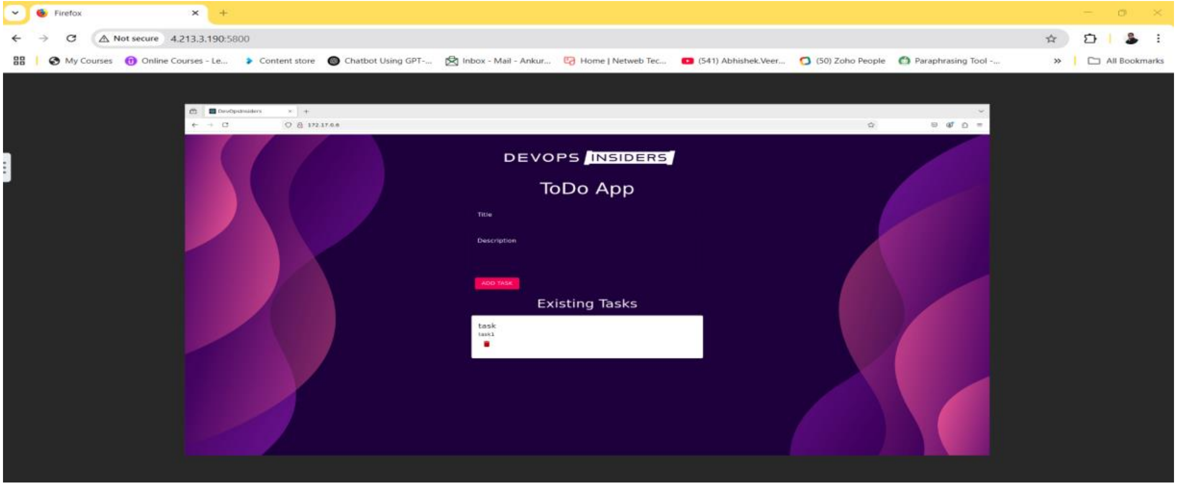
Now to access all Todoapp we require a browser for which we will deploy below container

As we are working in docker network so we will deploy a firefox container to which we will access through our VM IP. It will open a firefox browser via container

- Docker run -d -p 5800:3000 linuxserver/firefox
- `6e8226857b9e   jlesage/firefox   "/init"   46 hours ago   Up About an hour   0.0.0.0:5800->5800/tcp, :::5800->5800/tcp, 5900/tcp   firefox`



From this firewall container browser, we will now access the frontend app and try to insert data in data column.

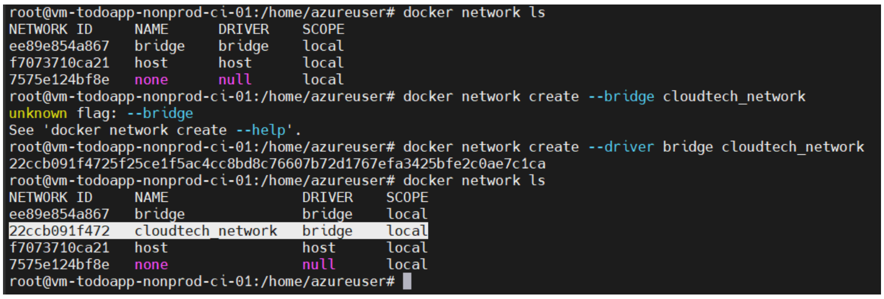


## User Defined Bridge Network

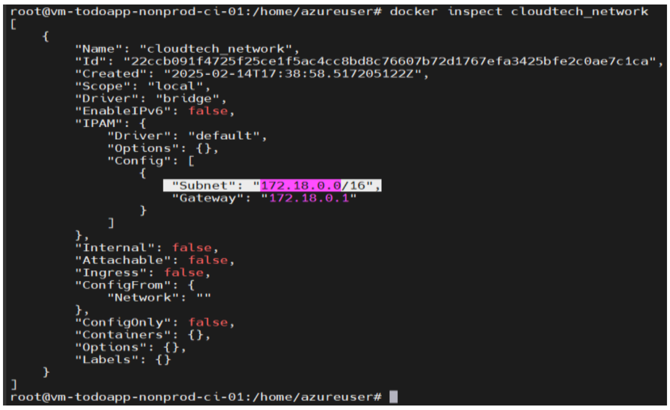
A user defined network is the network in which we can create our own CIDR range network.

We have to use command

- docker network create –driver bridge Networkname
- docker network create --bridge bridge cloudtech_network

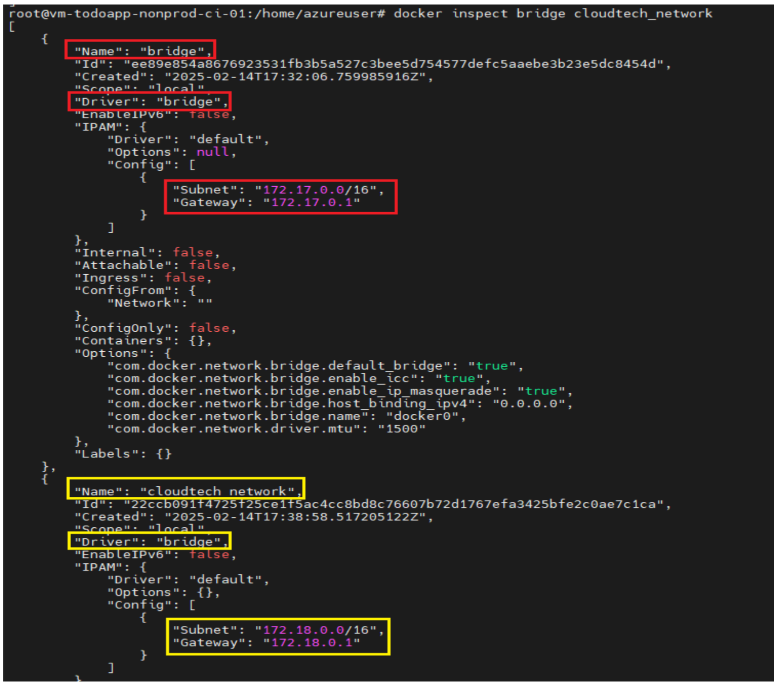


```
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
ee89e854a867   bridge    bridge    local
f7073710ca21   host      host      local
7575e124bf8e   none      null      local
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker network create --bridge cloudtech_network
unknown flag: --bridge
See 'docker network create --help'.
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker network create --driver bridge cloudtech_network
22ccb091f4725f25ce1f5ac4cc8bd8c76607b72d1767efa3425bfe2c0ae7c1ca
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
ee89e854a867   bridge              bridge    local
22ccb091f472   cloudtech_network   bridge    local
f7073710ca21   host                host      local
7575e124bf8e   none                null      local
root@vm-todoapp-nonprod-ci-01:/home/azureuser# 
```



```
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker inspect cloudtech_network
[
    {
        "Name": "cloudtech_network",
        "Id": "22ccb091f4725f25ce1f5ac4cc8bd8c76607b72d1767efa3425bfe2c0ae7c1ca",
        "Created": "2025-02-14T17:38:58.517205122Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
root@vm-todoapp-nonprod-ci-01:/home/azureuser# 
```

The network setup ensures complete isolation between the two bridge networks: bridge and cloudtech_network. While both networks use the same bridge driver, they operate on different CIDR ranges, making them distinct from each other. Additionally, these networks are fully isolated, meaning containers connected to bridge cannot communicate with those on cloudtech_network, and vice versa. There is no association between them or with the host network, ensuring strict network segmentation.

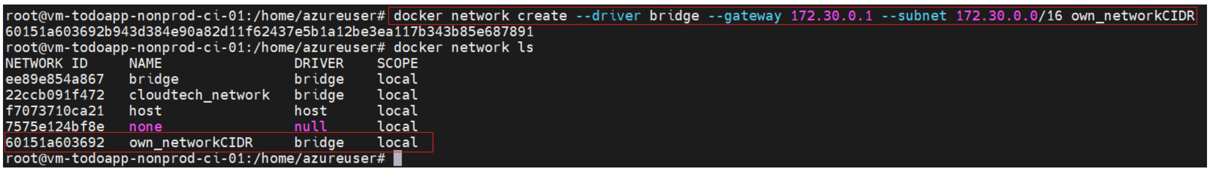


```
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker inspect bridge cloudtech_network
[
    {
        "Name": "bridge",
        "Id": "ee89e854a8676923531fb3b5a527c3bee5d754577defc5aaebe3b23e5dc8454d",
        "Created": "2025-02-14T17:32:06.759985916Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    },
    {
        "Name": "cloudtech_network",
        "Id": "22ccb091f4725f25ce1f5ac4cc8bd8c76607b72d1767efa3425bfe2c0ae7c1ca",
        "Created": "2025-02-14T17:38:58.517205122Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
```

Now if we want to create a bridge network of our own defined CIDR like 172.30.0.0/16

Command

- docker network create --driver bridge --gateway 172.30.0.1 --subnet 172.30.0.0/16

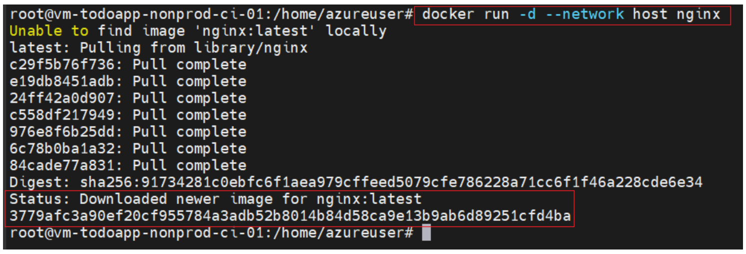


```
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker network create --driver bridge --gateway 172.30.0.1 --subnet 172.30.0.0/16 own_networkCIDR
60151a603692b943d384e90a82d11f62437e5b1a12be3ea117b343b85e687891
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
ee89e854a867   bridge            bridge    local
22ccb091f472   cloudtech_network bridge    local
f7073710ca21   host              host      local
7575e124bf8e   none              null      local
60151a603692   own_networkCIDR   bridge    local
root@vm-todoapp-nonprod-ci-01:/home/azureuser#
```

## Host

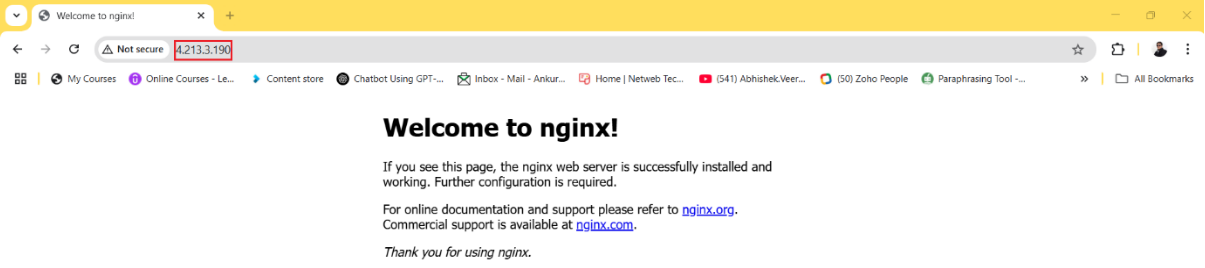
Now if we want to access the container via our laptop or VM on which Docker is installed we have to access it through port forwarding otherwise we can access the container from inside the docker network only.
but if we don't want to do port forwarding, we have to use Host network. Host network uses Host Driver.

using the host network mode in Docker means that the container will share the network stack of the host machine. This allows direct communication between the container and external networks, such as Azure or any other connected network, without the need for NAT (Network Address Translation) or port mapping. Essentially, the container behaves as if it is running directly on the host, using the host's IP address.



```
root@vm-todoapp-nonprod-ci-01:/home/azureuser# docker run -d --network host nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
c29f5b76f736: Pull complete
e19db8451adb: Pull complete
24ff42a0d907: Pull complete
c558df217949: Pull complete
976e8f6b25dd: Pull complete
6c78b0ba1a32: Pull complete
84cade77a831: Pull complete
Digest: sha256:91734281c0ebfc6f1aea979cffeed5079cfe786228a71cc6f1f46a228cde6e34
Status: Downloaded newer image for nginx:latest
3779afc3a90ef20cf955784a3adb52b8014b84d58ca9e13b9ab6d89251cfd4ba
root@vm-todoapp-nonprod-ci-01:/home/azureuser#
```

Now if I just browse my public IP of VM on any browser at my laptop it will open the application I deployed on container. It will use same network interface to VM/server(Host)



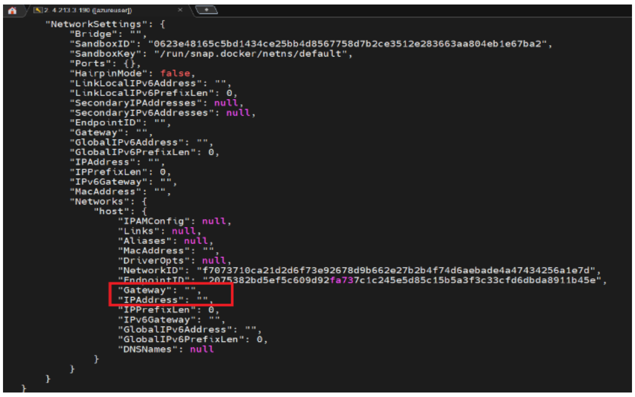
**Welcome to nginx!**

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

*Thank you for using nginx.*

Now if we check IP through docker inspect nginx



```
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "0623e48165c5bd1434ce25bb4d8567758d7b2ce3512e283663aa804eb1e67ba2",
            "SandboxKey": "/run/snap.docker/netns/default",
            "Ports": {},
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "",
            "Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "",
            "IPPrefixLen": 0,
            "IPv6Gateway": "",
            "MacAddress": "",
            "Networks": {
                "host": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "",
                    "DriverOpts": null,
                    "NetworkID": "f7073710ca21d2d6f73e92678d9b662e27b2b4f74d6aebade4a47434256a1e7d",
                    "EndpointID": "2075382bd5ef5c609d92fa737c1c245e5d85c15b5a3f3c33cfd6dbda8911b45e",
                    "Gateway": "",
                    "IPAddress": "",
                    "IPPrefixLen": 0,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DNSNames": null
                }
            }
        }
    }
```

Here no IP is showing because no bridge network is used here and application is running on host network

## None Network

A fully isolated network means that a container created using the none network mode will have no network connectivity at all. It will not be able to communicate with other containers, the host machine, or any external networks, such as the internet. Essentially, the container will exist in a network-disabled state, preventing any inbound or outbound traffic.

## Overlay Network

When multiple containers are running on different hosts and need to communicate with each other, an overlay network is used. This type of network is managed by Docker and allows containers on different physical or virtual machines to interact as if they were on the same local network. The overlay driver facilitates this communication by creating a virtual network that spans multiple Docker hosts, enabling seamless container-to-container connectivity across different nodes.