DOCKER

Reference- https://docs.docker.com/get-started/docker-overview/

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Typical use case of docker

1. Your developers write code locally and share their work with their colleagues using Docker containers.
2. They use Docker to push their applications into a test environment and run automated and manual tests.
3. When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.

4. When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

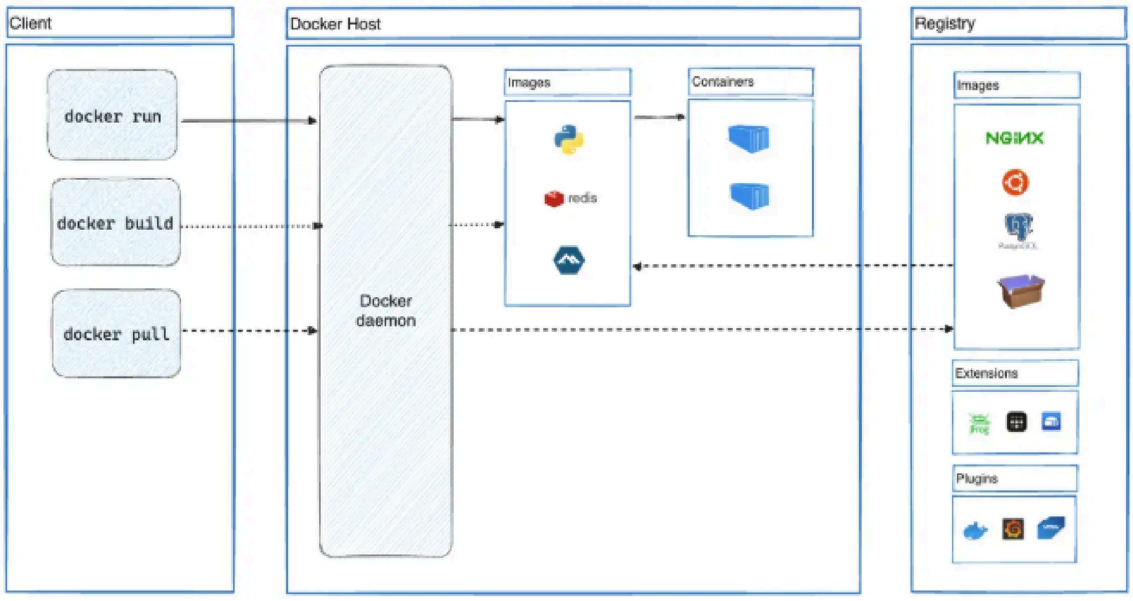
DOCKER FEATURES-

Fast, consistent delivery of your applications.

Responsive deployment and scaling.

Running more workloads on the same hardware.

DOCKER ARCHITECTURE



Docker Daemon-

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services. The docker client can send requests to the docker daemon to carry out tasks like creating images, containers etc when we use commands like docker run, docker build etc.

Docker Client-

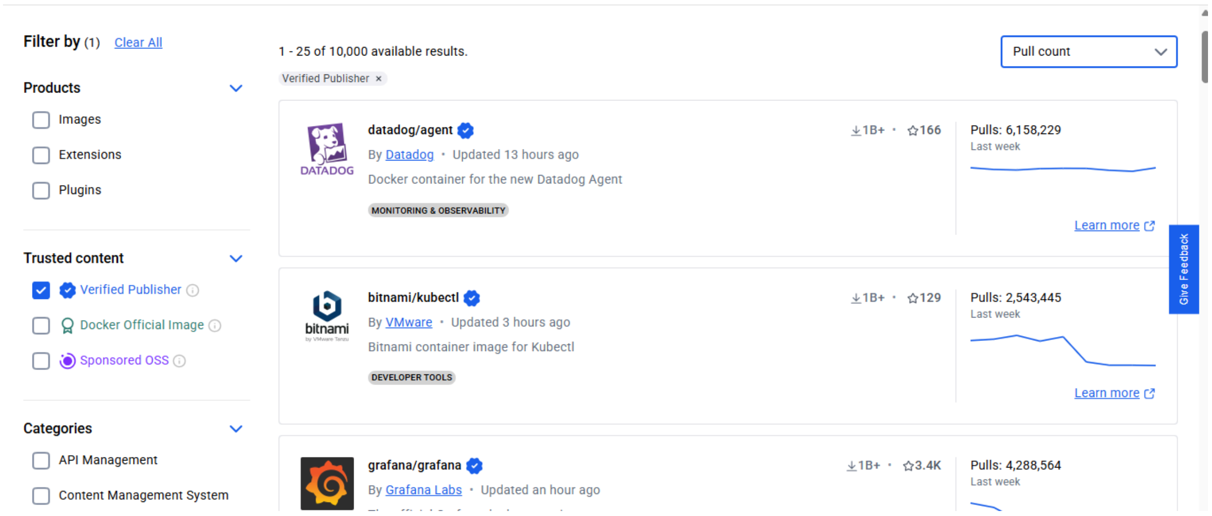
The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Registry-

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker looks for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, Docker pulls the required images from your configured registry. When you use the docker push command, Docker pushes your image to your configured registry.

DockerHub- DockerHub is an online registry hub that can be used to store your images. It can also be used to pull official images that are posted by verified publishers.


Filter by (1) Clear All

Products
- Images
- Extensions
- Plugins

Trusted content
- [✓] Verified Publisher
- Docker Official Image
- Sponsored OSS

Categories
- API Management
- Content Management System

1 - 25 of 10,000 available results.

Verified Publisher ×

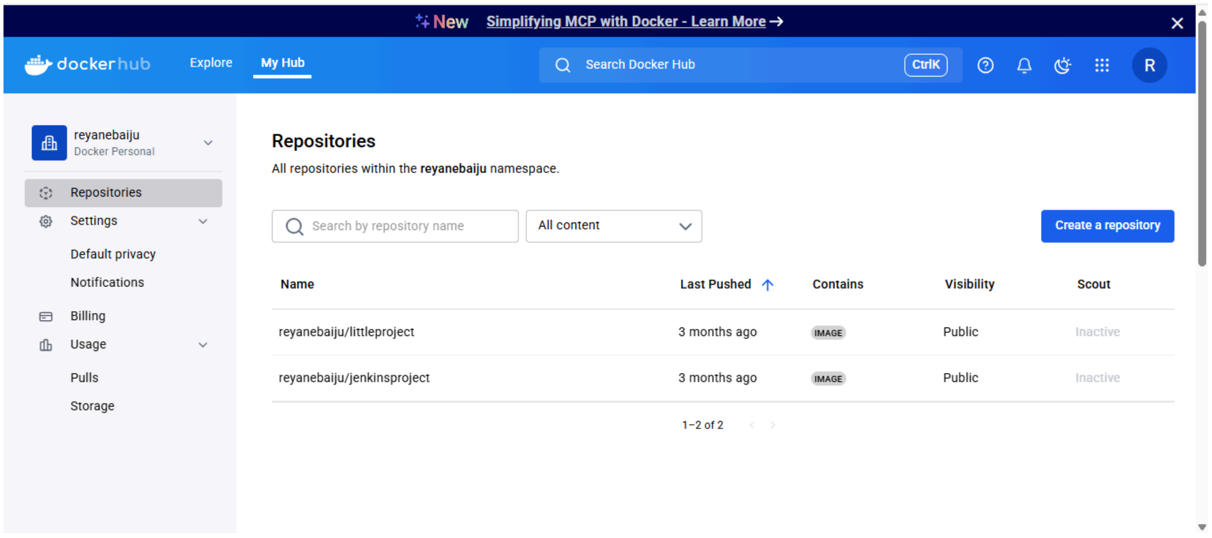
Pull count

datadog/agent
By Datadog · Updated 13 hours ago
Docker container for the new Datadog Agent
MONITORING & OBSERVABILITY
1B+ · 166
Pulls: 6,158,229
Last week
Learn more

bitnami/kubectl
By VMware · Updated 3 hours ago
Bitnami container image for Kubectl
DEVELOPER TOOLS
1B+ · 129
Pulls: 2,543,445
Last week
Learn more

grafana/grafana
By Grafana Labs · Updated an hour ago
1B+ · 3.4K
Pulls: 4,288,564
Last week

We can also have our own public or private repositories if we go to the “my hub” section.


New Simplifying MCP with Docker - Learn More →

docker hub Explore My Hub Search Docker Hub CtrlK

reyanebaiju
Docker Personal

- Repositories
- Settings
  - Default privacy
  - Notifications
- Billing
- Usage
  - Pulls
  - Storage

Repositories

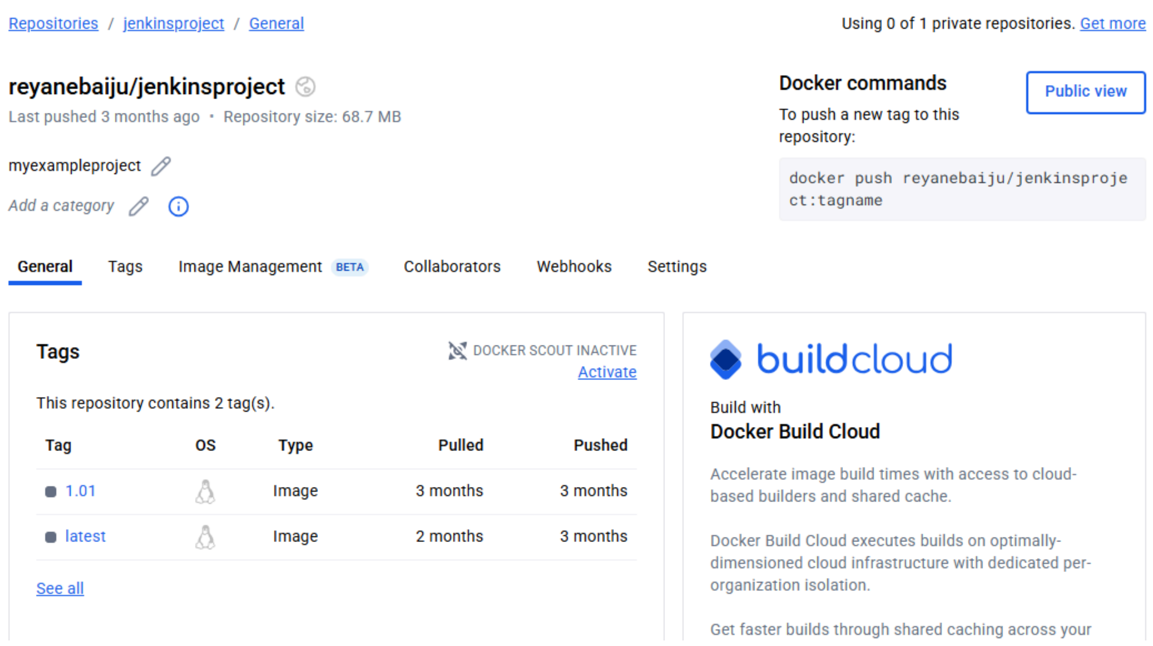
All repositories within the **reyanebaiju** namespace.

Search by repository name | All content | Create a repository

Name	Last Pushed	Contains	Visibility	Scout
reyanebaiju/littleproject	3 months ago	IMAGE	Public	Inactive
reyanebaiju/jenkinsproject	3 months ago	IMAGE	Public	Inactive

1–2 of 2

We can see all the images that we have pushed to the docker repository.


Repositories / jenkinsproject / General

Using 0 of 1 private repositories. Get more

reyanebaiju/jenkinsproject

Last pushed 3 months ago · Repository size: 68.7 MB

myexampleproject

Add a category

Docker commands

Public view

To push a new tag to this repository:

```
docker push reyanebaiju/jenkinsproje
ct:tagname
```

General | Tags | Image Management BETA | Collaborators | Webhooks | Settings

Tags

DOCKER SCOUT INACTIVE Activate

This repository contains 2 tag(s).

Tag	OS	Type	Pulled	Pushed
1.01		Image	3 months	3 months
latest		Image	2 months	3 months

See all

buildcloud

Build with

Docker Build Cloud

Accelerate image build times with access to cloud-based builders and shared cache.

Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.

Get faster builds through shared caching across your

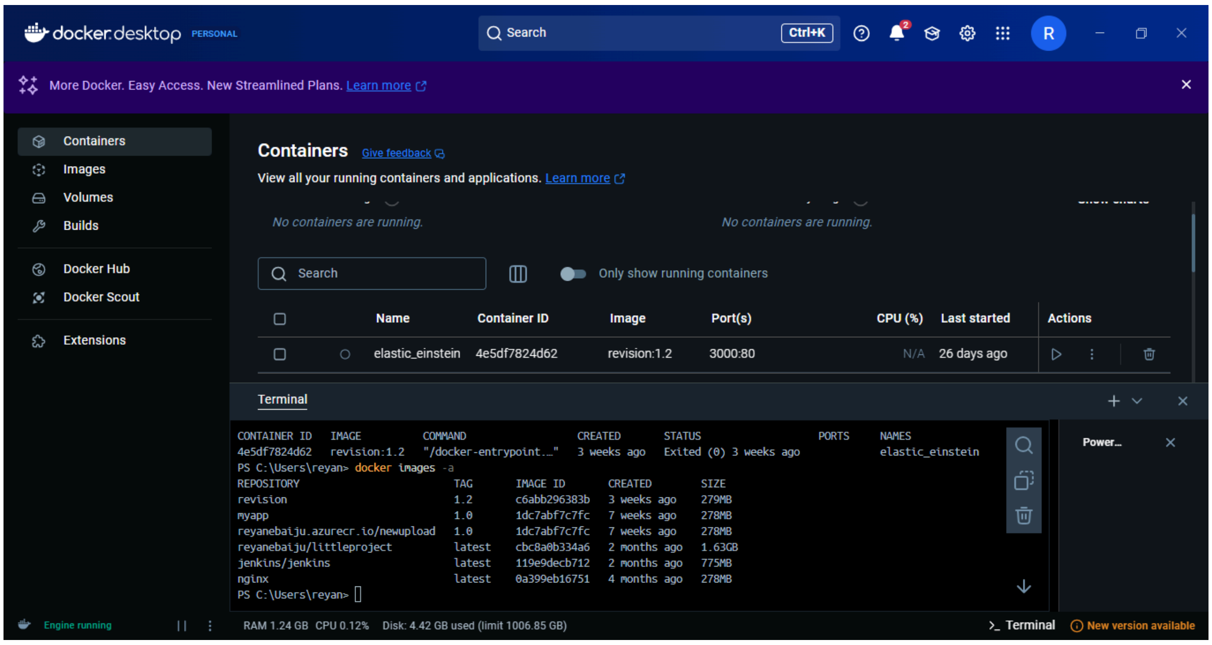
(EXTRA KNOWLEDGE- To run a container, you need to have a container runtime. The container runtime in docker is containerd).

Installing Docker

Reference- https://docs.docker.com/get-started/get-docker/

Docker can be installed in Windows, Linux and MacOS.

I have installed docker desktop on windows-

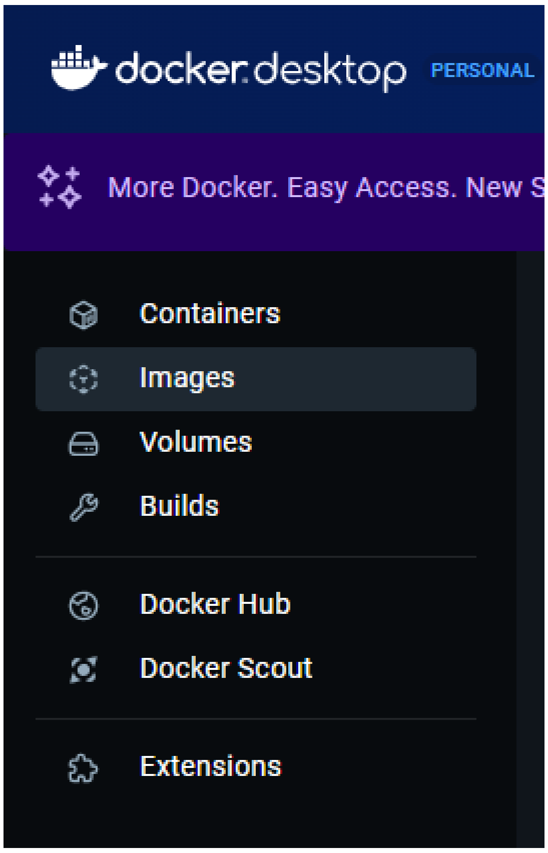

docker desktop PERSONAL
Search Ctrl+K
More Docker. Easy Access. New Streamlined Plans. Learn more
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
Containers Give feedback
View all your running containers and applications. Learn more
No containers are running.
No containers are running.
Search
Only show running containers
Engine running
RAM 1.24 GB CPU 0.12% Disk: 4.42 GB used (limit 1006.85 GB)
Terminal
New version available

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
	elastic_einstein	4e5df7824d62	revision:1.2	3000:80	N/A	26 days ago	

```
CONTAINER ID   IMAGE          COMMAND                  CREATED       STATUS                     PORTS     NAMES
4e5df7824d62   revision:1.2   "/docker-entrypoint.…"   3 weeks ago   Exited (0) 3 weeks ago               elastic_einstein
PS C:\Users\reyan> docker images -a
REPOSITORY                         TAG       IMAGE ID       CREATED        SIZE
revision                           1.2       c6abb296383b   3 weeks ago    279MB
myapp                              1.0       1dc7abf7c7fc   7 weeks ago    278MB
reyanebaiju.azurecr.io/newupload   1.0       1dc7abf7c7fc   7 weeks ago    278MB
reyanebaiju/littleproject          latest    cbc8a0b334a6   2 months ago   1.63GB
jenkins/jenkins                    latest    119e9decb712   2 months ago   775MB
nginx                              latest    0a399eb16751   4 months ago   278MB
PS C:\Users\reyan> 
```

We can start containers using images using the docker terminal.

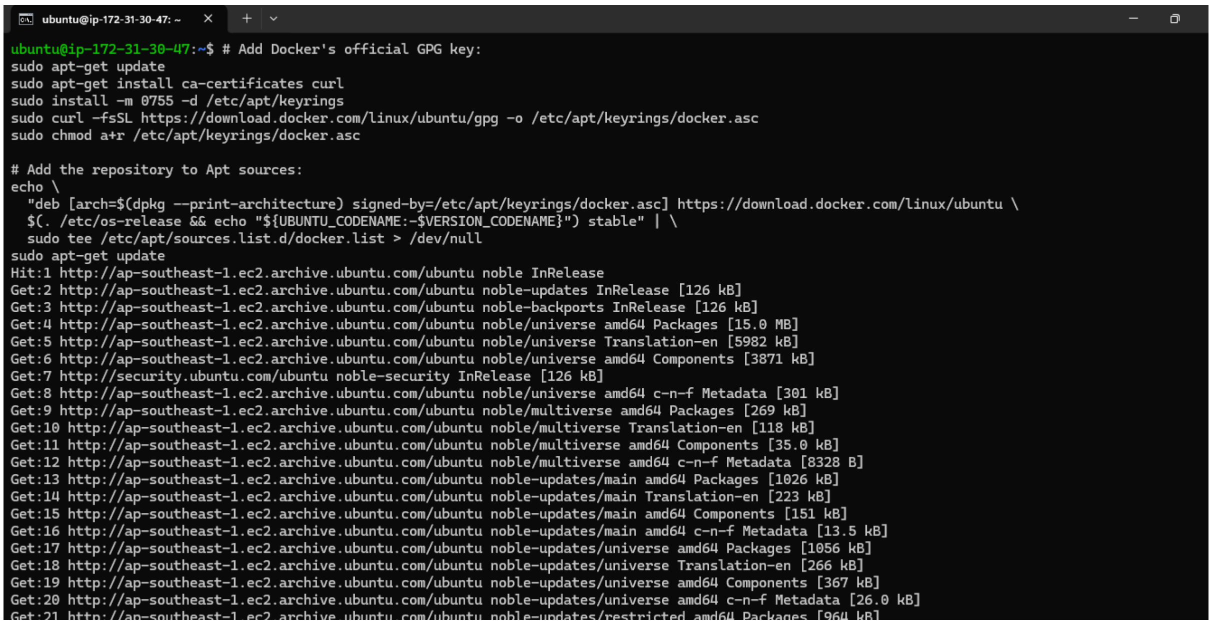
Docker for Windows gets a UI for easy management. We can easily access the volumes and builds from the UI.



Docker for Linux-

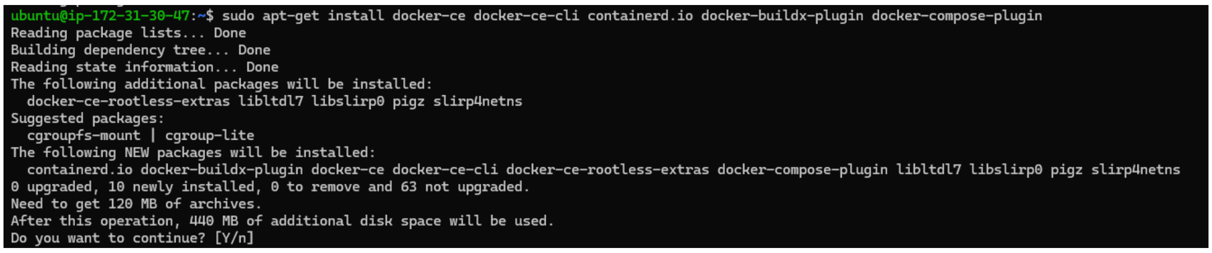
Docker for linux can be installed on our Ubuntu system by using the commands mentioned in this official site- https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository

Start by running the commands-



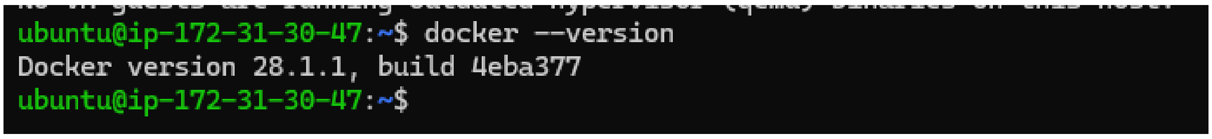
```
ubuntu@ip-172-31-30-47:~$ # Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
Hit:1 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:5 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:6 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:8 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:9 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:10 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:11 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:12 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:13 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1026 kB]
Get:14 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [223 kB]
Get:15 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [151 kB]
Get:16 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 c-n-f Metadata [13.5 kB]
Get:17 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1056 kB]
Get:18 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [266 kB]
Get:19 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [367 kB]
Get:20 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [26.0 kB]
Get:21 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [964 kB]
```

Now install docker-



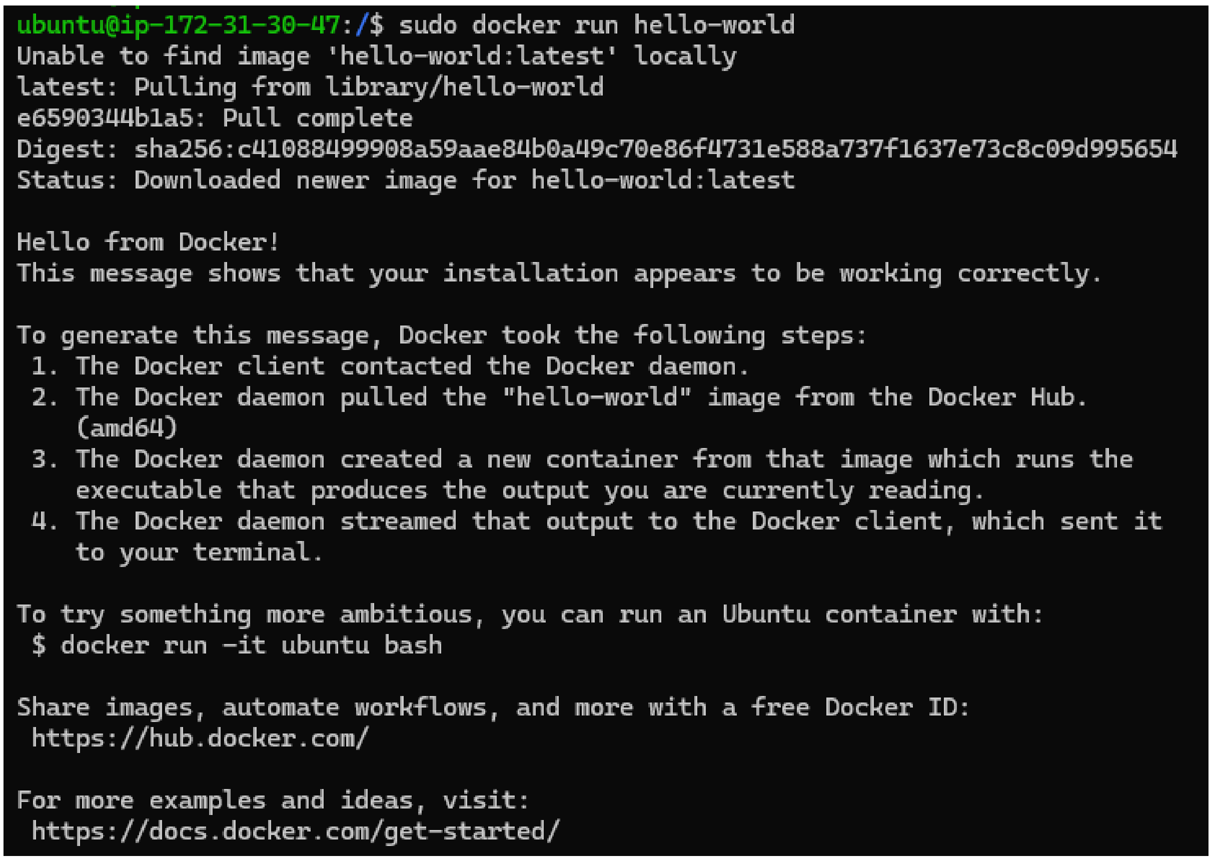
```
ubuntu@ip-172-31-30-47:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras libltdl7 libslirp0 pigz slirp4netns
Suggested packages:
  cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 63 not upgraded.
Need to get 120 MB of archives.
After this operation, 440 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Docker is installed successfully.



```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-30-47:~$ docker --version
Docker version 28.1.1, build 4eba377
ubuntu@ip-172-31-30-47:~$
```

To check if docker is working correctly, use the docker run hello-world command.



```
ubuntu@ip-172-31-30-47:/$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:c41088499908a59aae84b0a49c70e86f4731e588a737f1637e73c8c09d995654
Status: Downloaded newer image for hello-world:latest

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

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

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

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

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

All the docker images, containers, volumes are stored in the /var/lib/docker/ directory.

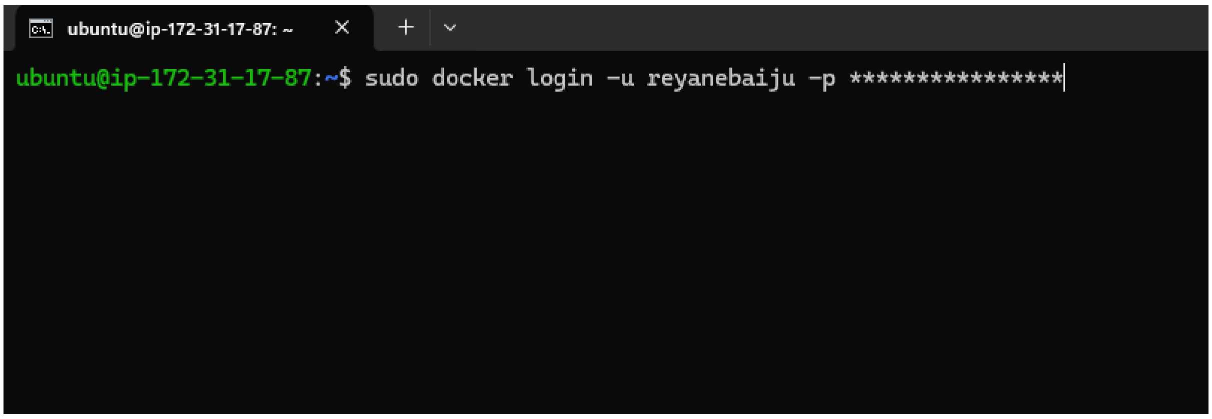
Configure Docker

Docker CLI commands reference documentation- https://docs.docker.com/reference/cli/docker/

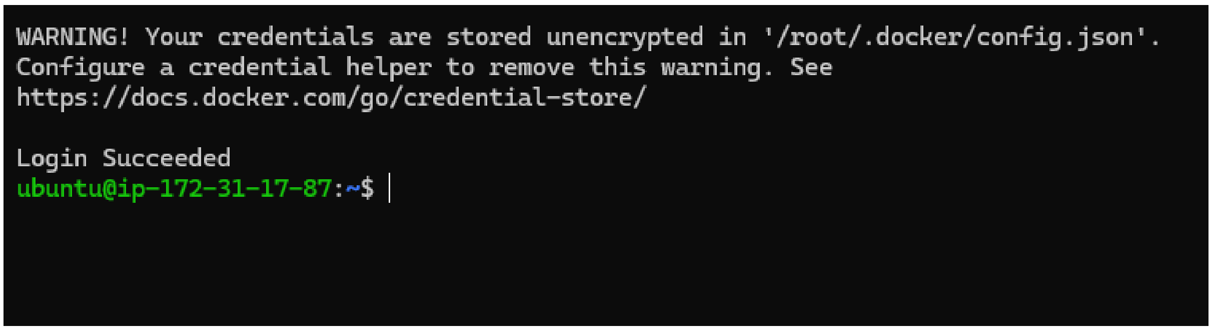
DOCKER LOGIN

We can login to docker hub registry if we wanted to by using the docker login command,

Ex- docker login -u reyanebaiju -p ************



```
ubuntu@ip-172-31-17-87:~$ sudo docker login -u reyanebaiju -p ****************
```



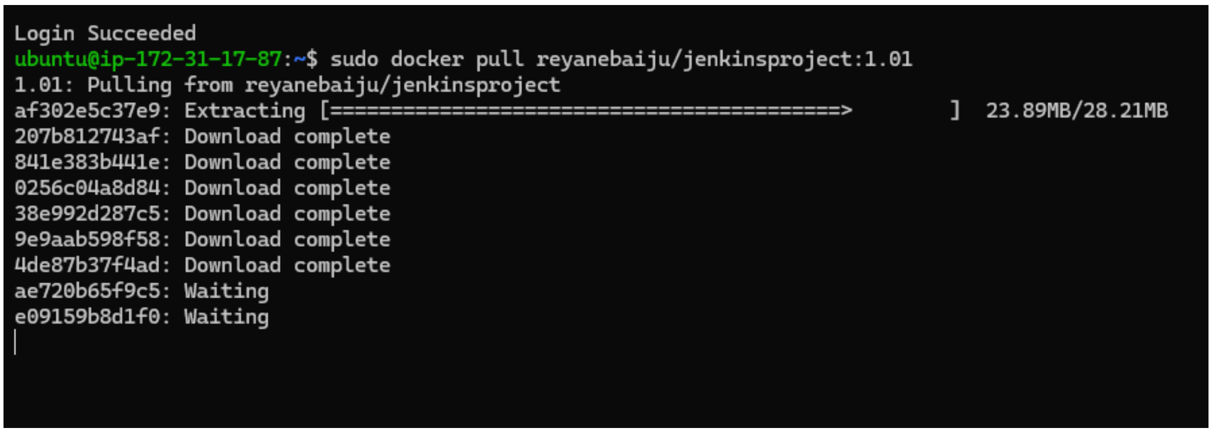
```
WARNING! Your credentials are stored unencrypted in '/root/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
ubuntu@ip-172-31-17-87:~$
```

You have now logged into DockerHub via docker CLI.

Tip- don't hardcore values, use protected variables to store username and password.

DOCKER PULL

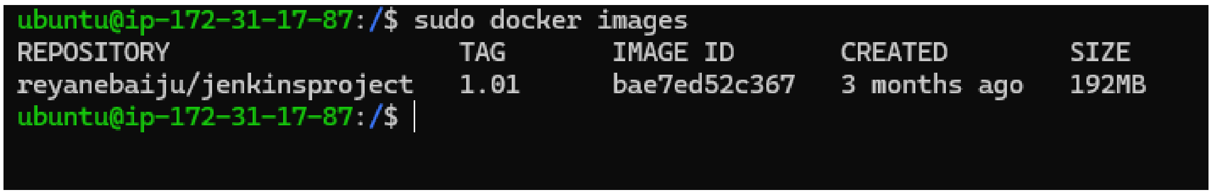
I have a docker image in my DockerHub. We can pull images using the command sudo docker pull reponame/imagename:version.



```
Login Succeeded
ubuntu@ip-172-31-17-87:~$ sudo docker pull reyanebaiju/jenkinsproject:1.01
1.01: Pulling from reyanebaiju/jenkinsproject
af302e5c37e9: Extracting [==========================================>        ]  23.89MB/28.21MB
207b812743af: Download complete
841e383b441e: Download complete
0256c04a8d84: Download complete
38e992d287c5: Download complete
9e9aab598f58: Download complete
4de87b37f4ad: Download complete
ae720b65f9c5: Waiting
e09159b8d1f0: Waiting
```

DOCKER IMAGES

To see all the docker images available in our system, type the sudo docker images command.



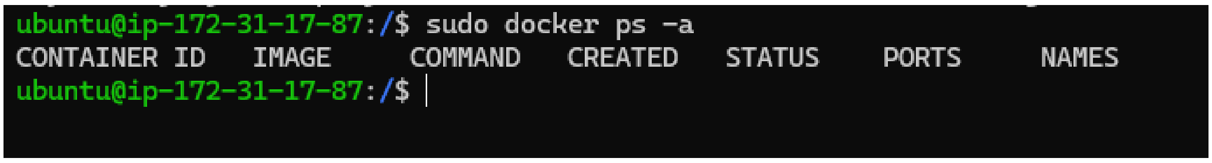
```
ubuntu@ip-172-31-17-87:/$ sudo docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
reyanebaiju/jenkinsproject    1.01      bae7ed52c367   3 months ago    192MB
ubuntu@ip-172-31-17-87:/$
```

DOCKER PS

Use this command to see all the running containers.

Sudo docker ps -a

Use the -a flag to see all of them.



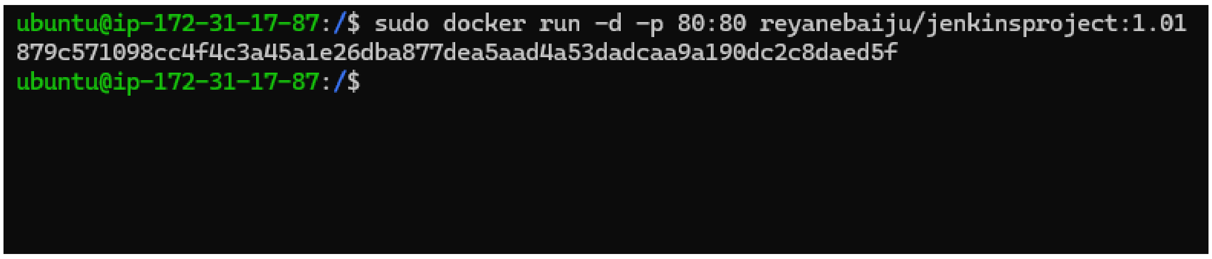
```
ubuntu@ip-172-31-17-87:/$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@ip-172-31-17-87:/$
```

DOCKER RUN

This command is used to run an image as a container.

Use the command sudo docker run name/imagename:tag

You can also directly run images as containers without pulling first.

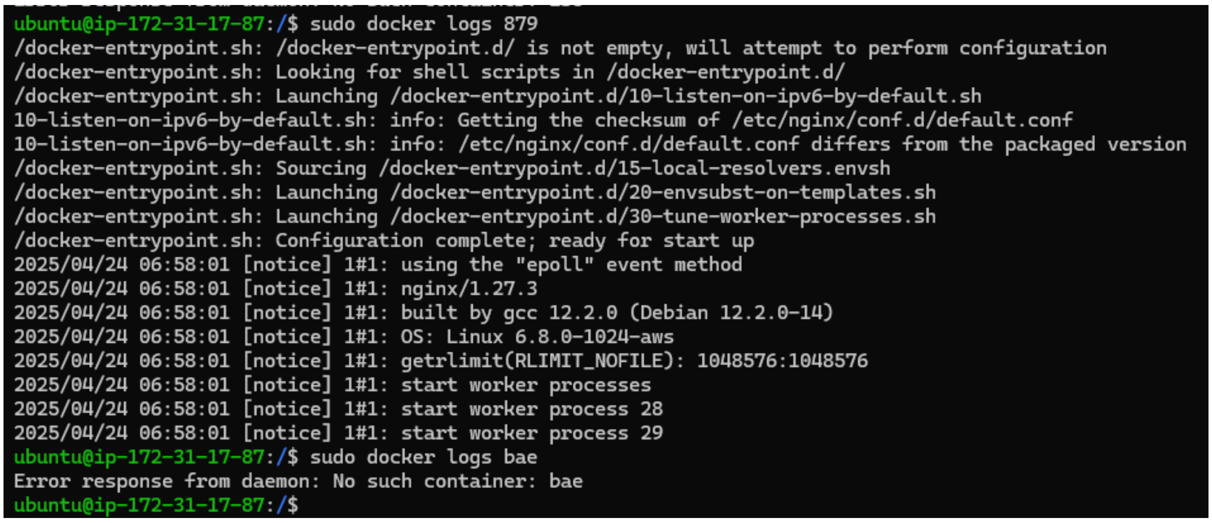


```
ubuntu@ip-172-31-17-87:/$ sudo docker run -d -p 80:80 reyanebaiju/jenkinsproject:1.01
879c571098cc4f4c3a45a1e26dba877dea5aad4a53dadcaa9a190dc2c8daed5f
ubuntu@ip-172-31-17-87:/$
```

DOCKER LOGS

You can see the logs of containers using this command-

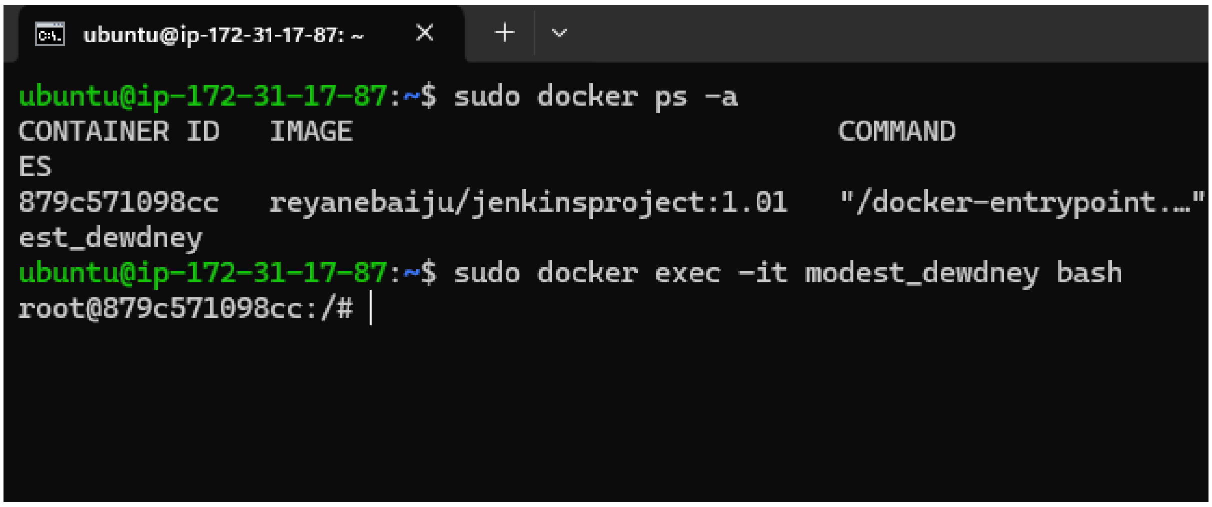
Sudo docker logs <containerID>


```
ubuntu@ip-172-31-17-87:/$ sudo docker logs 879
/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: /etc/nginx/conf.d/default.conf differs from the packaged version
/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
2025/04/24 06:58:01 [notice] 1#1: using the "epoll" event method
2025/04/24 06:58:01 [notice] 1#1: nginx/1.27.3
2025/04/24 06:58:01 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/04/24 06:58:01 [notice] 1#1: OS: Linux 6.8.0-1024-aws
2025/04/24 06:58:01 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/04/24 06:58:01 [notice] 1#1: start worker processes
2025/04/24 06:58:01 [notice] 1#1: start worker process 28
2025/04/24 06:58:01 [notice] 1#1: start worker process 29
ubuntu@ip-172-31-17-87:/$ sudo docker logs bae
Error response from daemon: No such container: bae
ubuntu@ip-172-31-17-87:/$
```

DOCKER EXEC

Docker exec command can be used to execute a command in a running container.

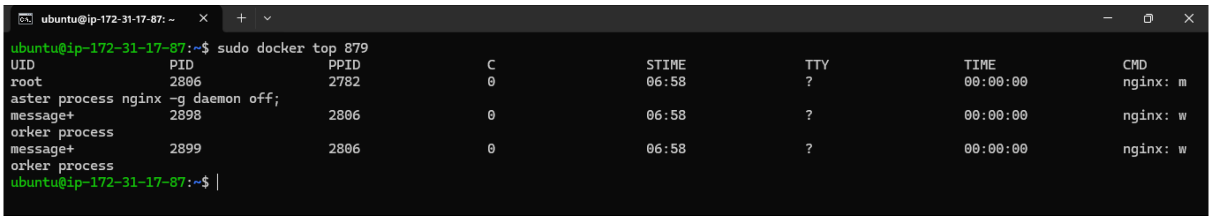
Here, I'm using the command docker exec -it <ContainerName> bash to use an interactive terminal to execute commands in the running container.



```
ubuntu@ip-172-31-17-87:~$ sudo docker ps -a
CONTAINER ID   IMAGE                              COMMAND
ES
879c571098cc   reyanebaiju/jenkinsproject:1.01    "/docker-entrypoint.…"
est_dewdney
ubuntu@ip-172-31-17-87:~$ sudo docker exec -it modest_dewdney bash
root@879c571098cc:/#
```

DOCKER TOP

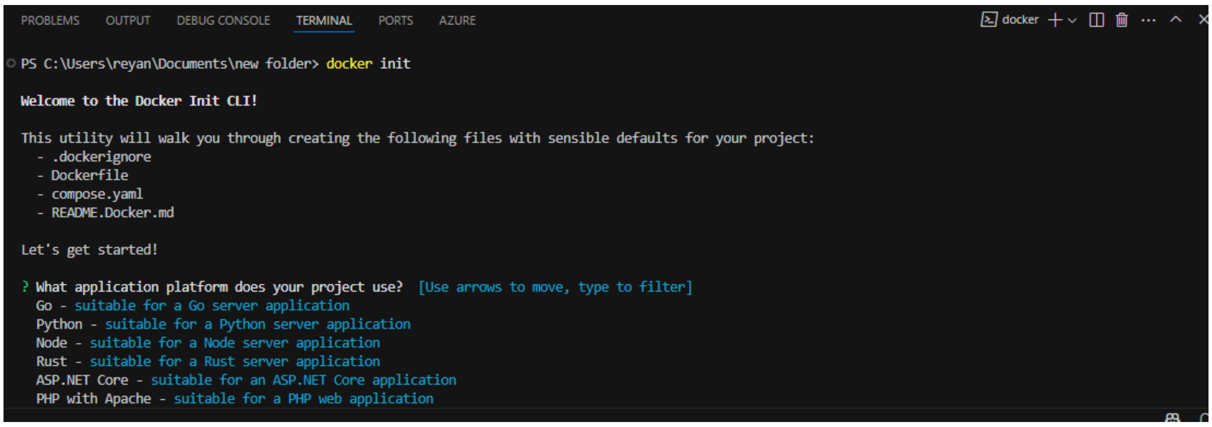
Display the running processes of a container. Use command sudo docker top <CID>


```
ubuntu@ip-172-31-17-87:~$ sudo docker top 879
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                2806                2782                0                   06:58               ?                   00:00:00            nginx: m
aster process nginx -g daemon off;
message+            2898                2806                0                   06:58               ?                   00:00:00            nginx: w
orker process
message+            2899                2806                0                   06:58               ?                   00:00:00            nginx: w
orker process
ubuntu@ip-172-31-17-87:~$ |
```

DOCKER INIT

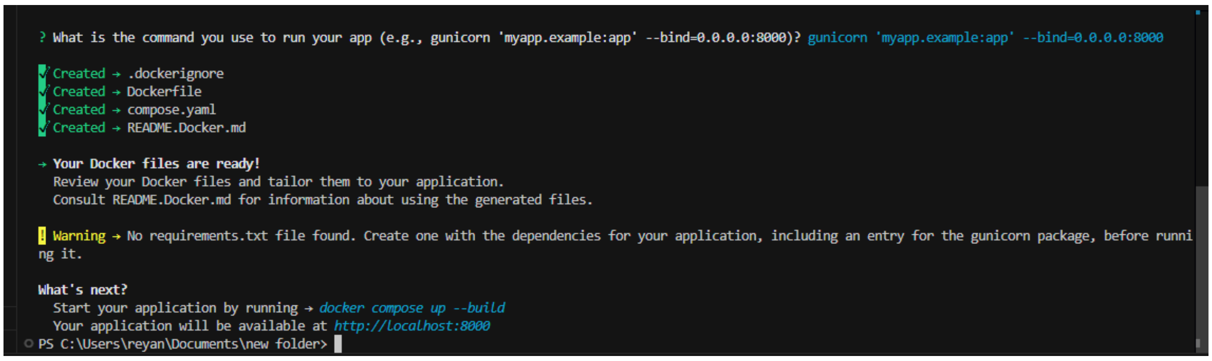
Docker init is a useful command that can be used to create the starting docker configuration files to use as a base.

Use the command docker init.

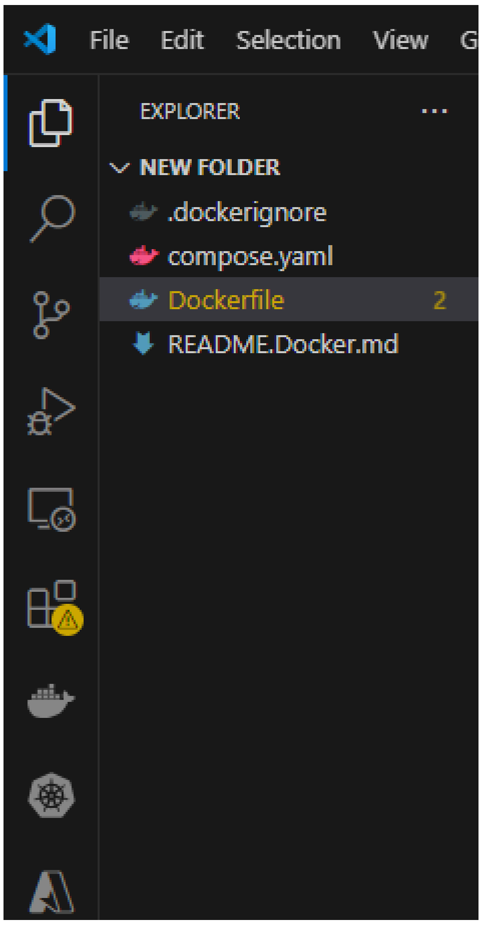


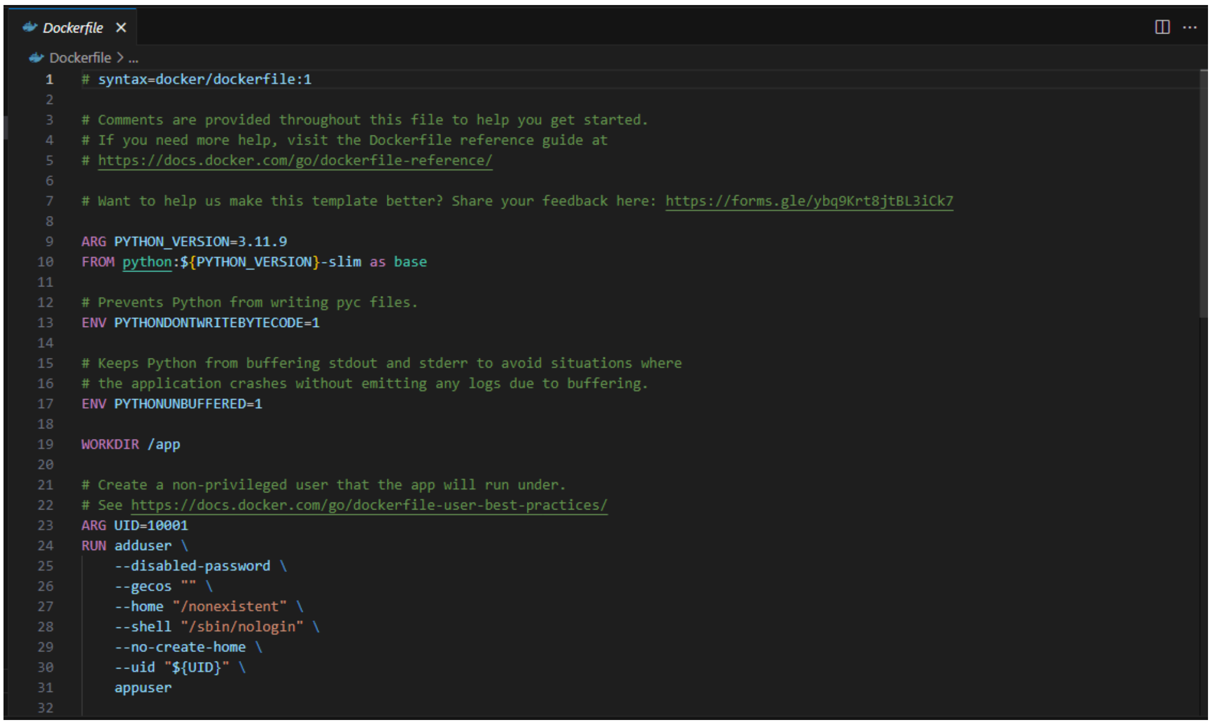
```
PS C:\Users\reyan\Documents\new folder> docker init

Welcome to the Docker Init CLI!

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

Let's get started!

? What application platform does your project use?  [Use arrows to move, type to filter]
  Go - suitable for a Go server application
  Python - suitable for a Python server application
  Node - suitable for a Node server application
  Rust - suitable for a Rust server application
  ASP.NET Core - suitable for an ASP.NET Core application
  PHP with Apache - suitable for a PHP web application
```

We can select what type of application that we are developing-


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

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

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

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

What's next?
  Start your application by running → docker compose up --build
  Your application will be available at http://localhost:8000
PS C:\Users\reyan\Documents\new folder> 
```

The files are automatically created for us.



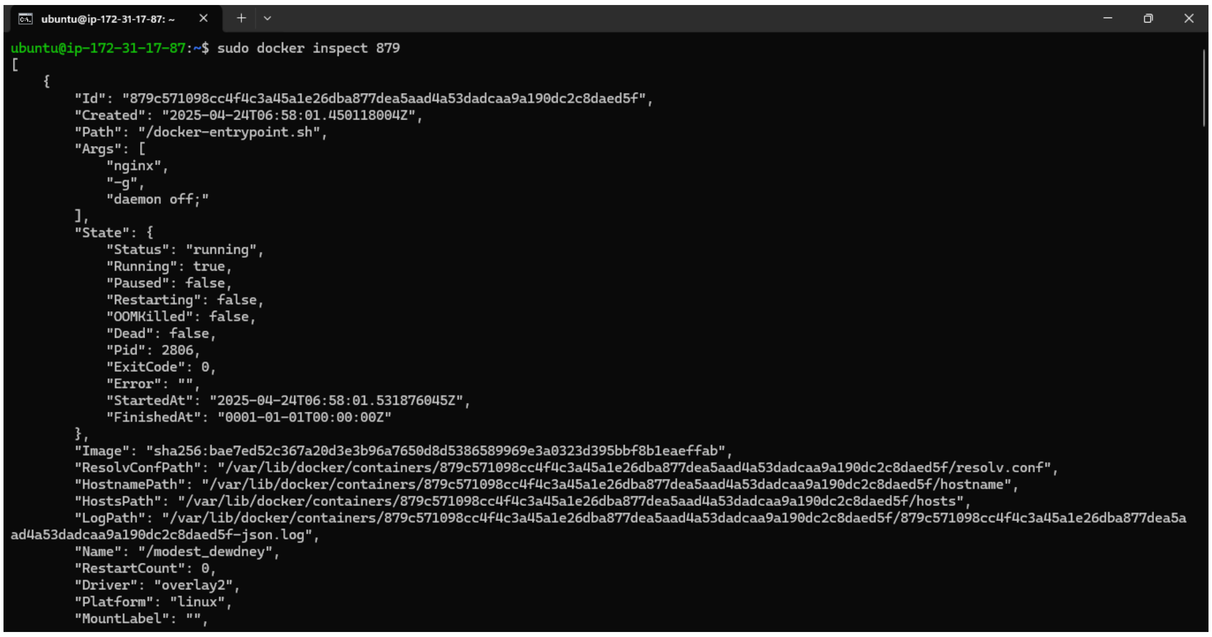


```
# syntax=docker/dockerfile:1

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

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

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

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

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

WORKDIR /app

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

```

DOCKER INSPECT

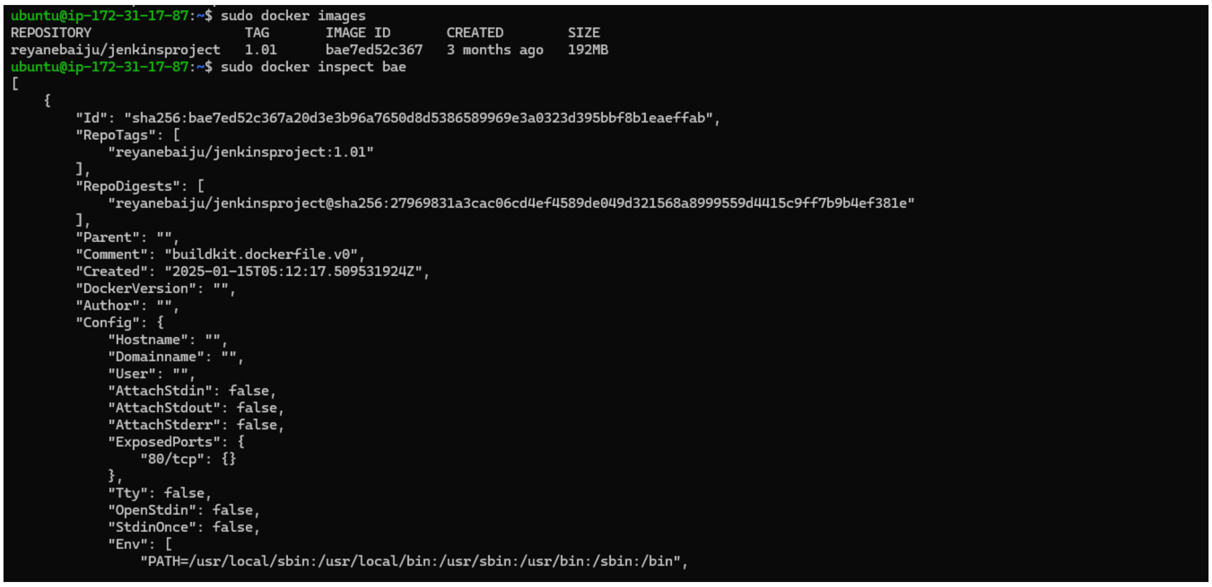
Docker inspect command can be used to return low level information.

Ex- Return info about a running container



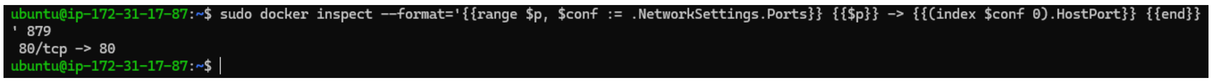
```
ubuntu@ip-172-31-17-87:~$ sudo docker inspect 879
[
    {
        "Id": "879c571098cc4f4c3a45a1e26dba877dea5aad4a53dadcaa9a190dc2c8daed5f",
        "Created": "2025-04-24T06:58:01.450118004Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 2806,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-04-24T06:58:01.531876045Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:bae7ed52c367a20d3e3b96a7650d8d5386589969e3a0323d395bbf8b1eaeffab",
        "ResolvConfPath": "/var/lib/docker/containers/879c571098cc4f4c3a45a1e26dba877dea5aad4a53dadcaa9a190dc2c8daed5f/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/879c571098cc4f4c3a45a1e26dba877dea5aad4a53dadcaa9a190dc2c8daed5f/hostname",
        "HostsPath": "/var/lib/docker/containers/879c571098cc4f4c3a45a1e26dba877dea5aad4a53dadcaa9a190dc2c8daed5f/hosts",
        "LogPath": "/var/lib/docker/containers/879c571098cc4f4c3a45a1e26dba877dea5aad4a53dadcaa9a190dc2c8daed5f/879c571098cc4f4c3a45a1e26dba877dea5a
ad4a53dadcaa9a190dc2c8daed5f-json.log",
        "Name": "/modest_dewdney",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
```

Ex- return info about an image



```
ubuntu@ip-172-31-17-87:~$ sudo docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
reyanebaiju/jenkinsproject   1.01      bae7ed52c367   3 months ago    192MB
ubuntu@ip-172-31-17-87:~$ sudo docker inspect bae
[
    {
        "Id": "sha256:bae7ed52c367a20d3e3b96a7650d8d5386589969e3a0323d395bbf8b1eaeffab",
        "RepoTags": [
            "reyanebaiju/jenkinsproject:1.01"
        ],
        "RepoDigests": [
            "reyanebaiju/jenkinsproject@sha256:27969831a3cac06cd4ef4589de049d321568a8999559d4415c9ff7b9b4ef381e"
        ],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2025-01-15T05:12:17.509531924Z",
        "DockerVersion": "",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
```

Check Port configuration of instance-

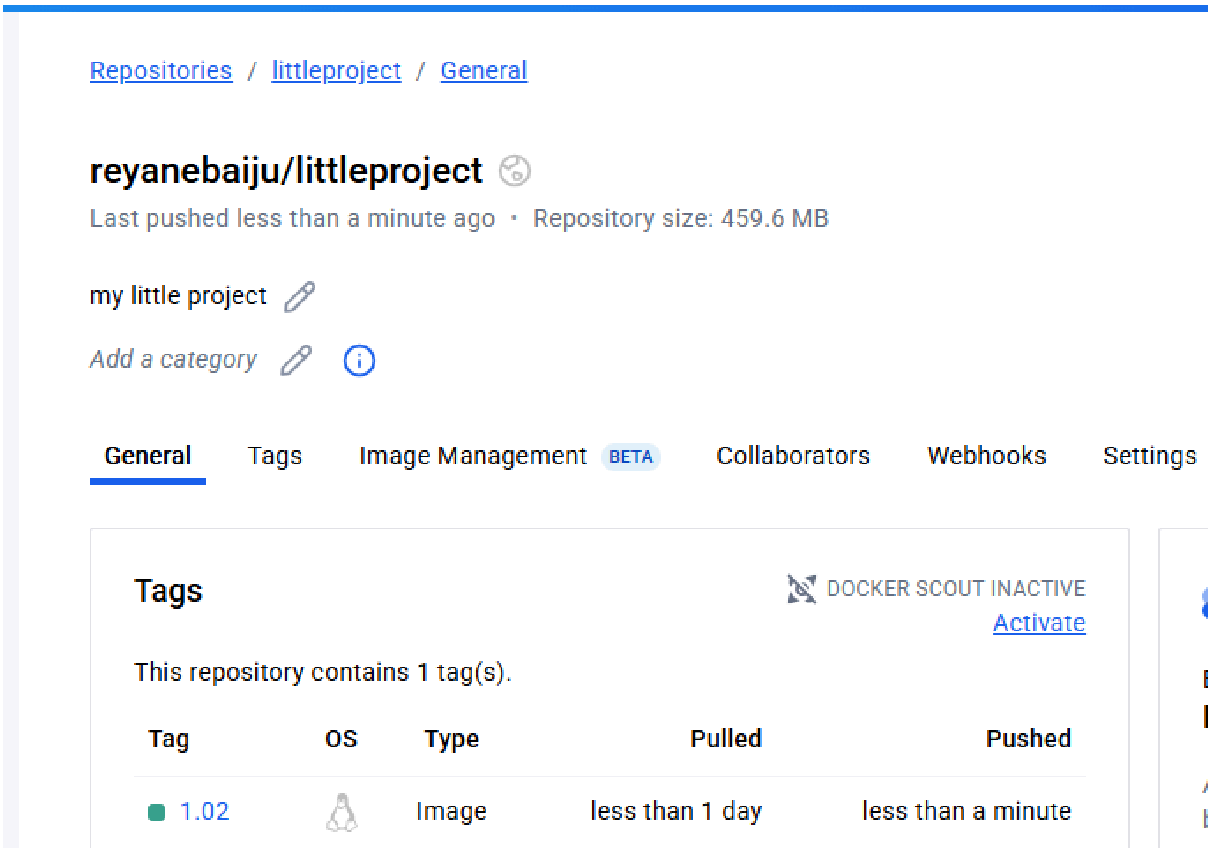


```
ubuntu@ip-172-31-17-87:~$ sudo docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}}
' 879
 80/tcp -> 80
ubuntu@ip-172-31-17-87:~$ |
```

DOCKER TAG-

The docker tag command is used to name and tag an image. After we build a docker image, we can tag an image to push it to our desired repository.

Here, I'm going to name and tag my image to be pushed to this repository. So the syntax for that is docker tag imagename:tag repo name/project name:tag


Repositories / littleproject / General

reyanebaiju/littleproject

Last pushed less than a minute ago • Repository size: 459.6 MB

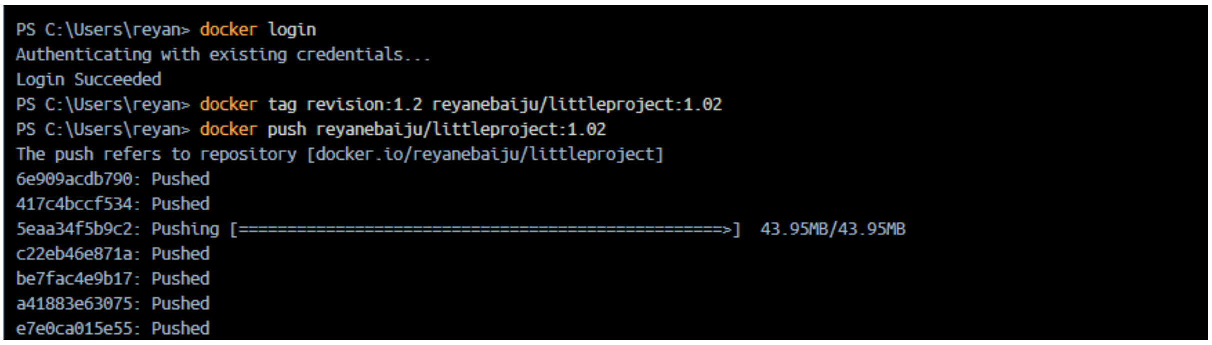
my little project

Add a category

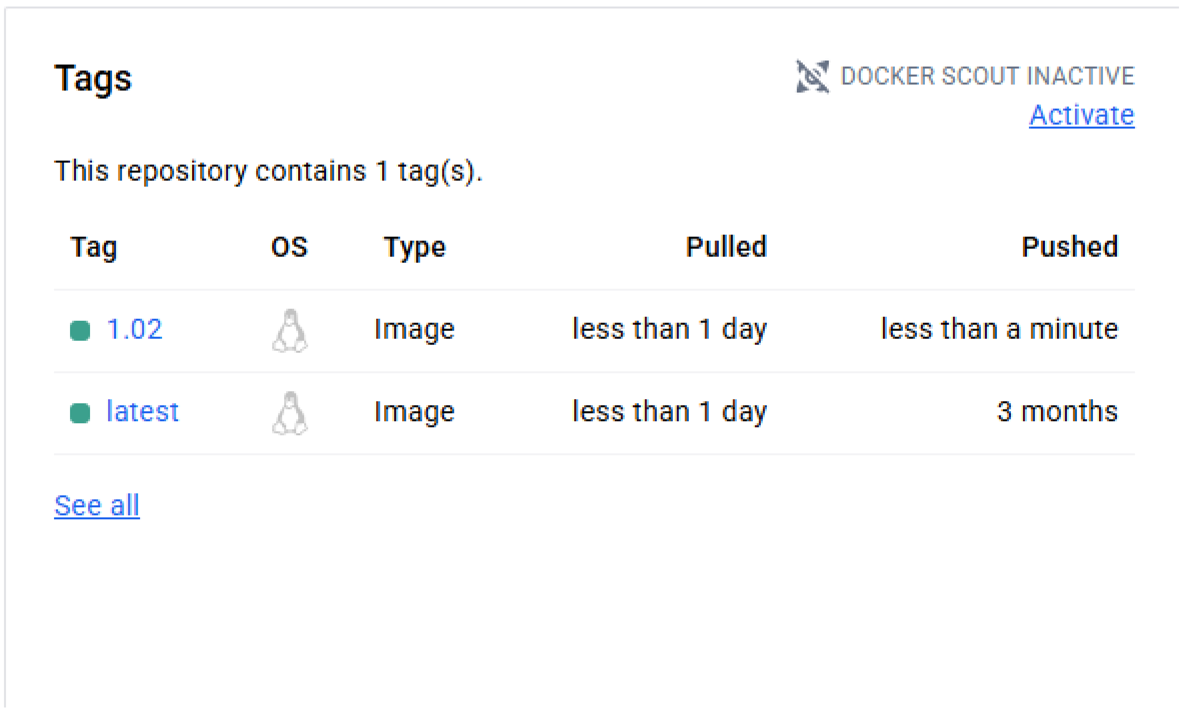
Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
🟩 1.02		Image	less than 1 day	less than a minute


```
PS C:\Users\reyan> docker login
Authenticating with existing credentials...
Login Succeeded
PS C:\Users\reyan> docker tag revision:1.2 reyanebaiju/littleproject:1.02
PS C:\Users\reyan> docker push reyanebaiju/littleproject:1.02
The push refers to repository [docker.io/reyanebaiju/littleproject]
6e909acdb790: Pushed
417c4bccf534: Pushed
5eaa34f5b9c2: Pushing [==================================================>]  43.95MB/43.95MB
c22eb46e871a: Pushed
be7fac4e9b17: Pushed
a41883e63075: Pushed
e7e0ca015e55: Pushed
```

We can see that the image is pushed successfully.



Tags

DOCKER SCOUT INACTIVE
Activate

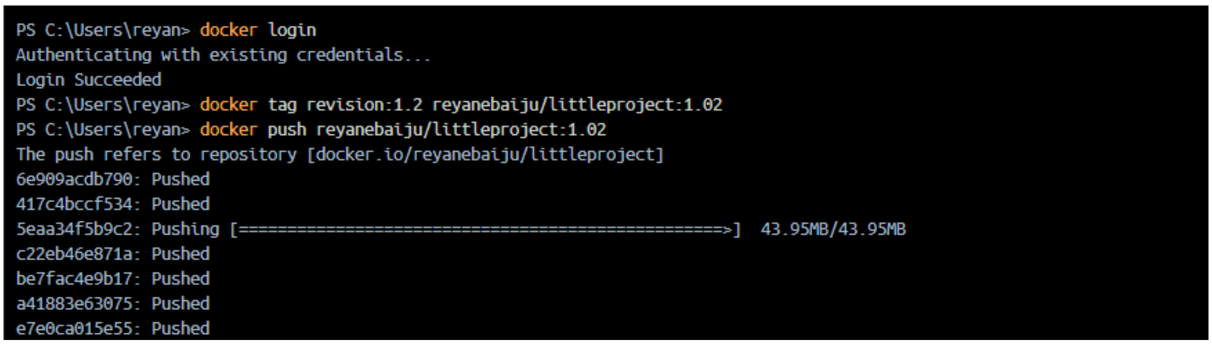
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
🟩 1.02		Image	less than 1 day	less than a minute
🟩 latest		Image	less than 1 day	3 months

See all

DOCKER PUSH

Docker push is a command used to push your docker image to your desired registry.



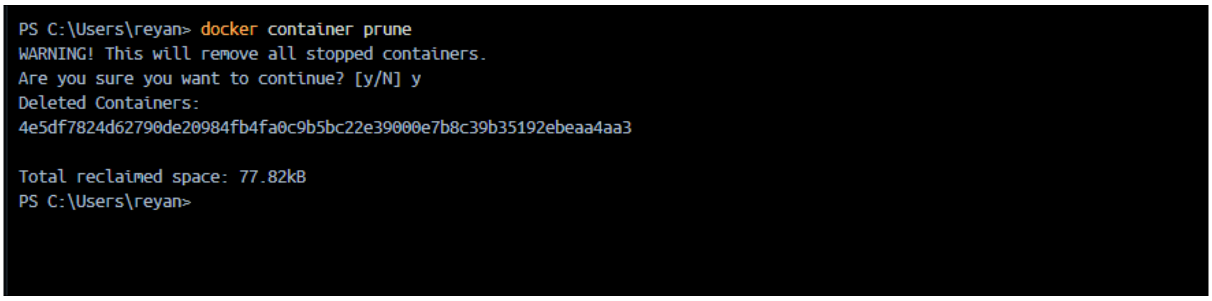
```
PS C:\Users\reyan> docker login
Authenticating with existing credentials...
Login Succeeded
PS C:\Users\reyan> docker tag revision:1.2 reyanebaiju/littleproject:1.02
PS C:\Users\reyan> docker push reyanebaiju/littleproject:1.02
The push refers to repository [docker.io/reyanebaiju/littleproject]
6e909acdb790: Pushed
417c4bccf534: Pushed
5eaa34f5b9c2: Pushing [==================================================>]  43.95MB/43.95MB
c22eb46e871a: Pushed
be7fac4e9b17: Pushed
a41883e63075: Pushed
e7e0ca015e55: Pushed
```

To push to your desired registry, you need to log in using docker cli to that registry.

DOCKER PRUNE

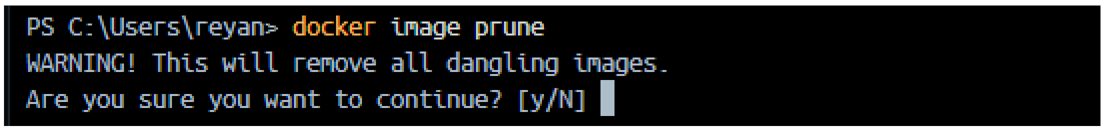
Docker prune command can be used to delete containers and images.

Docker container prune removes all stopped containers-



```
PS C:\Users\reyan> docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
4e5df7824d62790de20984fb4fa0c9b5bc22e39000e7b8c39b35192ebeaa4aa3

Total reclaimed space: 77.82kB
PS C:\Users\reyan>
```

Docker image prune is used to delete dangling images-



```
PS C:\Users\reyan> docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N]
```

DOCKER IMAGES

A container image is a standardized package that includes all of the files, binaries, libraries, and configurations to run a container.

There are two important principles of images:

1. Images are immutable. Once an image is created, it can't be modified. You can only make a new image or add changes on top of it.
2. Container images are composed of layers. Each layer represents a set of file system changes that add, remove, or modify files.

Docker images consist of a base image- A base image can be anything based on your needs. It can be a python base image, a node based image to run javascript etc.

Then it consists of our code and files, for example, it could be a simple HTML file to run in an nginx web server base image.

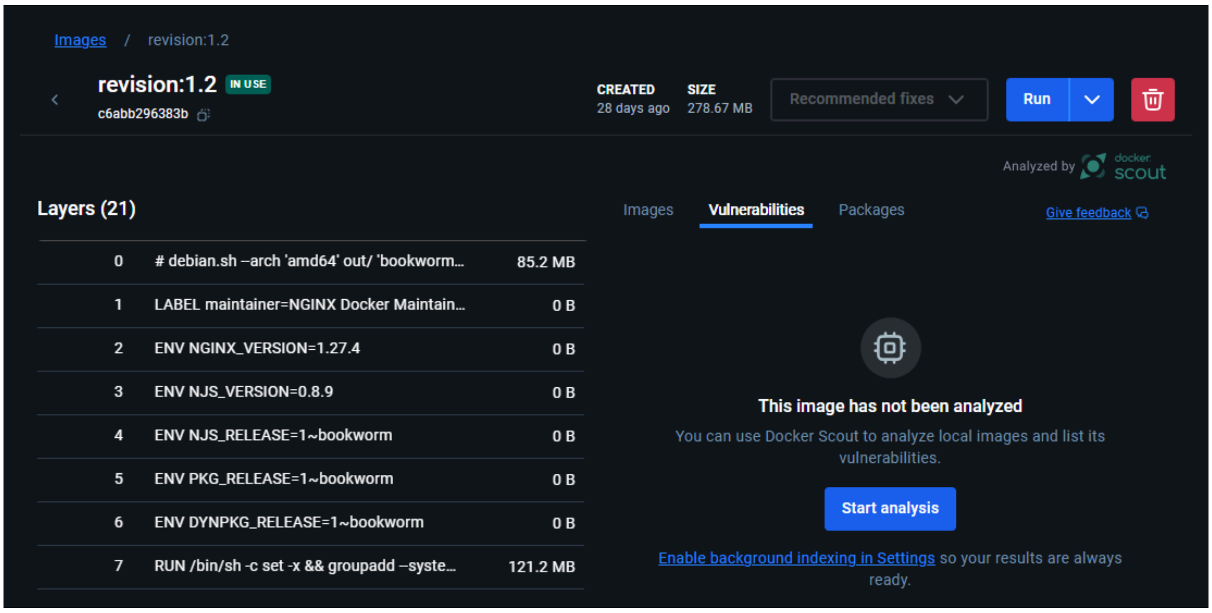
Then, it could consist of additional binaries and dependencies that we could add to support our program.

All of these are what constitutes a docker image.

We use a DOCKERFILE to create a docker image. A dockerfile is a script that tells the docker engine how to create a docker image.

Images contain name and tag for identification.

Example image-



Layers (21)

0	# debian.sh --arch 'amd64' out/ 'bookworm...	85.2 MB
1	LABEL maintainer=NGINX Docker Maintain...	0 B
2	ENV NGINX_VERSION=1.27.4	0 B
3	ENV NJS_VERSION=0.8.9	0 B
4	ENV NJS_RELEASE=1~bookworm	0 B
5	ENV PKG_RELEASE=1~bookworm	0 B
6	ENV DYNPKG_RELEASE=1~bookworm	0 B
7	RUN /bin/sh -c set -x && groupadd --syste...	121.2 MB

This image has not been analyzed

You can use Docker Scout to analyze local images and list its vulnerabilities.

Enable background indexing in Settings so your results are always ready.

To run an image, we use the sudo docker run command.

Important flags to remember-

1. '-d'

It is used to run the container in the background, meaning detached state.

2.’-p’

It is used to map ports to the container.

Ex. -p 8080:80.

8080 is the host port (entry port) and 80 is the container port.

3.’-v’

It is used to mount volumes. We can use this to mount local volumes to the container for data persistence and share files between local and container.

4.’-e’

It is used to create or pass environmental variables into the container.

Ex. -e “ABC=prod”

Or just use command “export variablename”

Then use docker run -e variablename

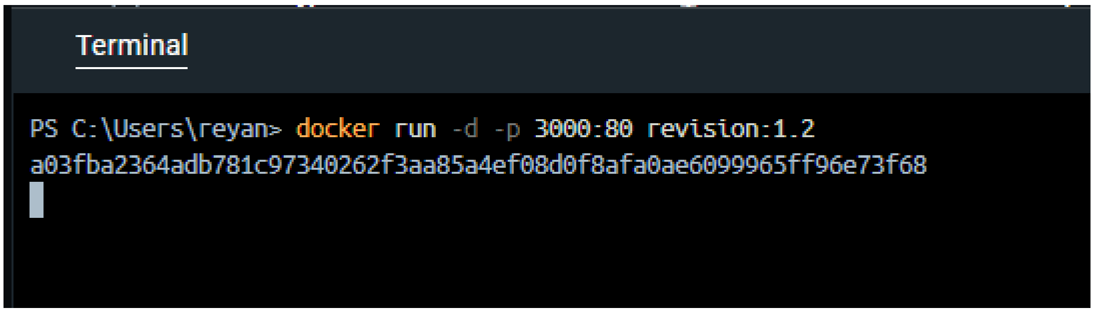
This is not safe to pass credentials.

5. ‘--env-file’

We can create a .env file with our credentials, and mention that file after this command

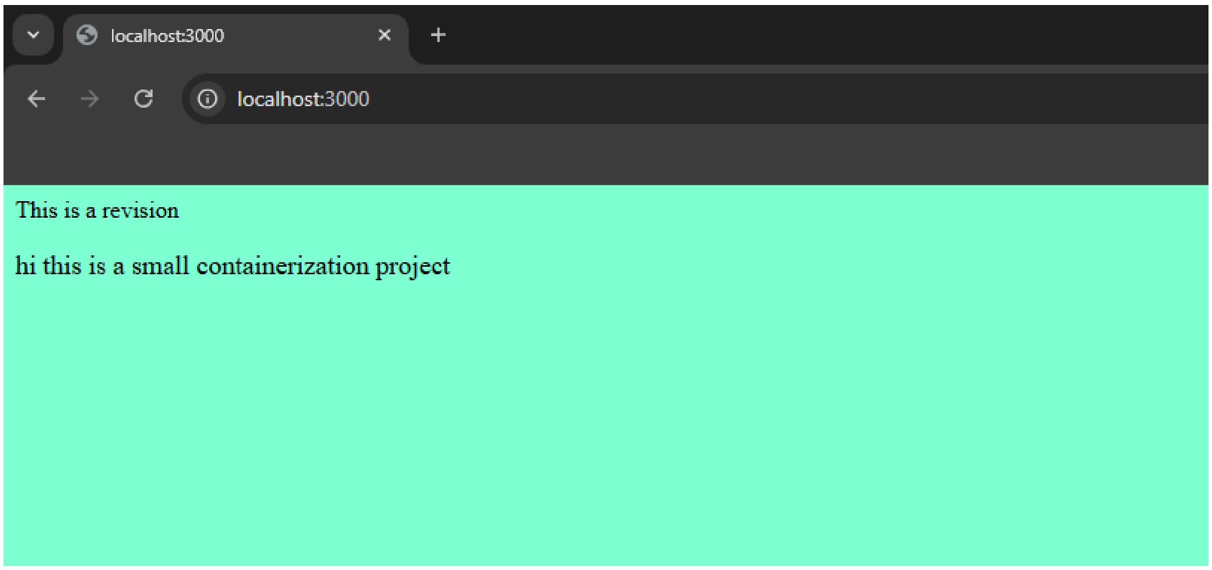
Ex- docker run –env-file abc.env

EXAMPLE OF RUNNING A CONTAINER FROM AN IMAGE-



```
PS C:\Users\reyan> docker run -d -p 3000:80 revision:1.2
a03fba2364adb781c97340262f3aa85a4ef08d0f8afa0ae6099965ff96e73f68
```

The container is running in port 3000. If we access localhost:3000 from our browser, we get this-



This is a revision

hi this is a small containerization project

(EXTRA) DOCKER MULTISTAGE

Docker multi stage is used to reduce the size of the docker image by using multiple stages of builds. The best method of using multi stage is-

Stage one should have a rich base image of Ubuntu which contains all the features like curl, wget, all the apt softwares and apt repositories. We can do all the building of the app in the first stage.

The second stage starts when we write the next FROM statement. This should be a distroless image which is very lightweight.

We can just copy the artifacts and binary from the first stage into the second stage and use CMD to run the app. Remember to alias the first base image using ‘AS’ keyboard.

Syntax of copying the binaries from stage one-

FROM Ubuntu AS build

COPY –from=build /folderorbinaryname(source) /folderorbinaryname(destination)

Link to get distroless images- https://github.com/GoogleContainerTools/distroless

DOCKER VOLUME AND BIND MOUNTS

Containers are ephemeral(short lived) in nature. We can use volume or bind mounting to have persistent data storage. This allows different containers to access this persistent file, and also allows the container to store data in the local machine for easy access to administrators.

1. BIND MOUNTS

Bind mounting means to bind or link a directory in a container to a directory in the local machine. The container directory is given access to the local directory. The local directory can exist anywhere in the host system.

The main disadvantage is that when using bind mounting, the local directory is present in the host machine only, so if we wanted to deploy the container in another host machine, we can’t access this local directory.

Ex docker run -v /path/in/local:/path/in/container

2. VOLUMES

Docker volumes are the recommended way of storing data.

Docker volumes are managed by docker and created and managed using docker CLI.

Docker volumes are located in the /var/lib/docker/volumes directory.

It can be cloned to other machines using tools.

Volumes can also be external sources like another entire host, an EC2, S3, NFS etc.

This can also be backed up.

Docker volume CLI commands reference-

https://docs.docker.com/reference/cli/docker/volume/

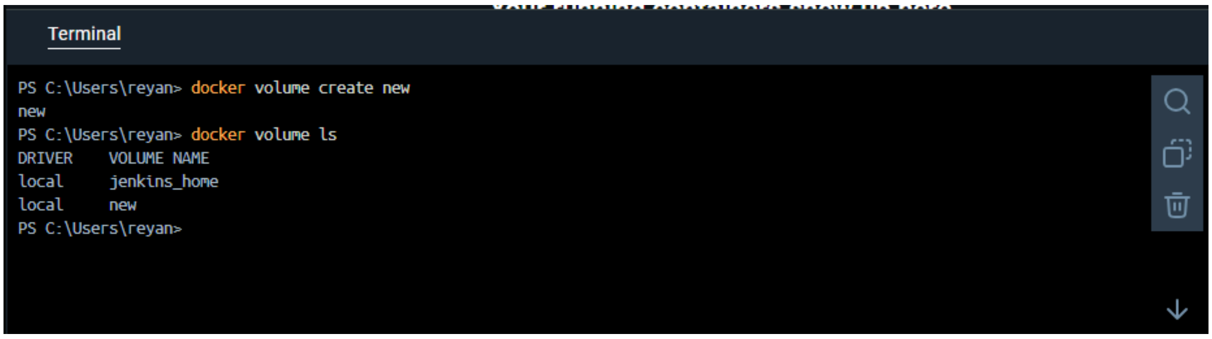
To create a docker volume, use the docker volume create <name> command, to list volumes, use docker volume ls.

To inspect a volume and return a json output, use syntax docker volume inspect myvolume

You can use the -v or –mount (verbose mode to give more details) command for bind mounting and volume.

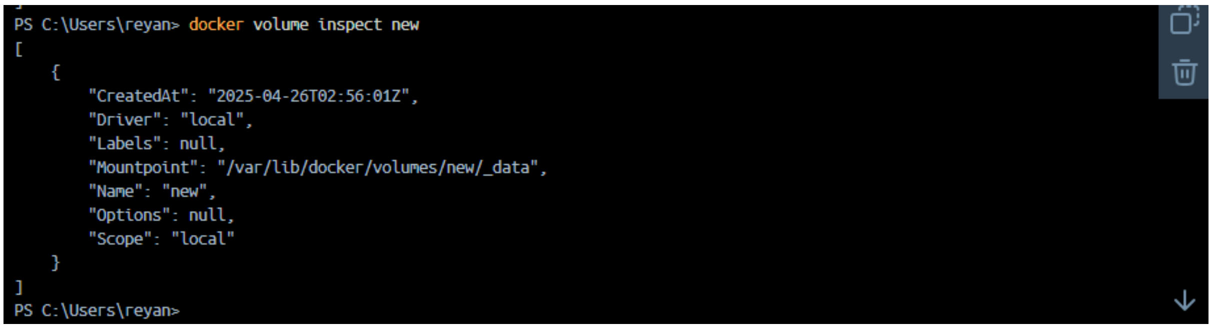
We don't use / in the start of the path to signify volume mounting.

Ex. docker run -v volumename:/path/in/container


Terminal

```
PS C:\Users\reyan> docker volume create new
new
PS C:\Users\reyan> docker volume ls
DRIVER    VOLUME NAME
local     jenkins_home
local     new
PS C:\Users\reyan>
```

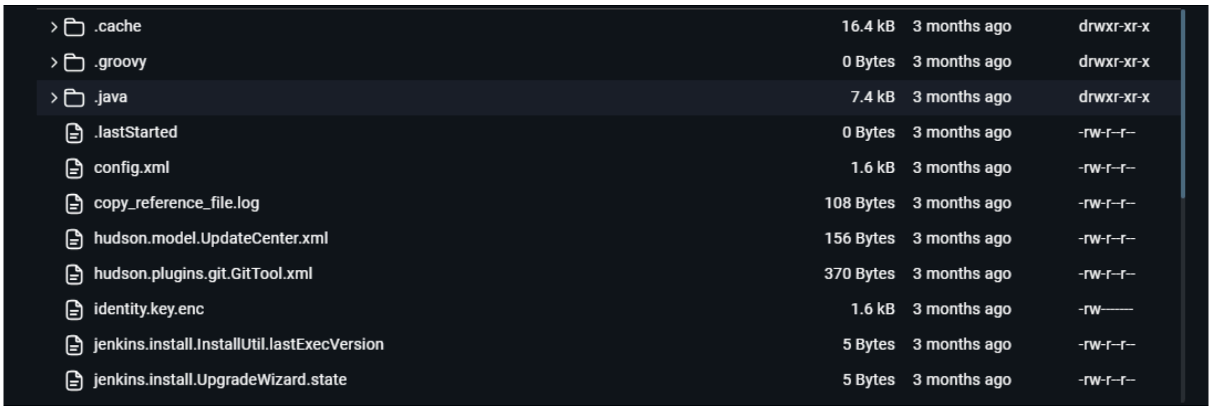
We can use the docker volume inspect <name> command to see the details of the volume.



```
PS C:\Users\reyan> docker volume inspect new
[
    {
        "CreatedAt": "2025-04-26T02:56:01Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/new/_data",
        "Name": "new",
        "Options": null,
        "Scope": "local"
    }
]
PS C:\Users\reyan>
```

To delete volume, use the command docker volume rm <name>

For example, this is a volume created for a jenkins container to store persistent data-



.cache	16.4 kB	3 months ago	drwxr-xr-x
.groovy	0 Bytes	3 months ago	drwxr-xr-x
.java	7.4 kB	3 months ago	drwxr-xr-x
.lastStarted	0 Bytes	3 months ago	-rw-r--r--
config.xml	1.6 kB	3 months ago	-rw-r--r--
copy_reference_file.log	108 Bytes	3 months ago	-rw-r--r--
hudson.model.UpdateCenter.xml	156 Bytes	3 months ago	-rw-r--r--
hudson.plugins.git.GitTool.xml	370 Bytes	3 months ago	-rw-r--r--
identity.key.enc	1.6 kB	3 months ago	-rw-------
jenkins.install.InstallUtil.lastExecVersion	5 Bytes	3 months ago	-rw-r--r--
jenkins.install.UpgradeWizard.state	5 Bytes	3 months ago	-rw-r--r--

DOCKER NETWORKING

By default, docker containers can talk to the host machine using a virtual ethernet called v.eth (docker0). This is called BRIDGED NETWORKING.

Eg. a user cannot access a container directly. He connects to the hosts which hosts the container. If the container doesnt have a working virtual ethernet, the user won't be able to connect to it.

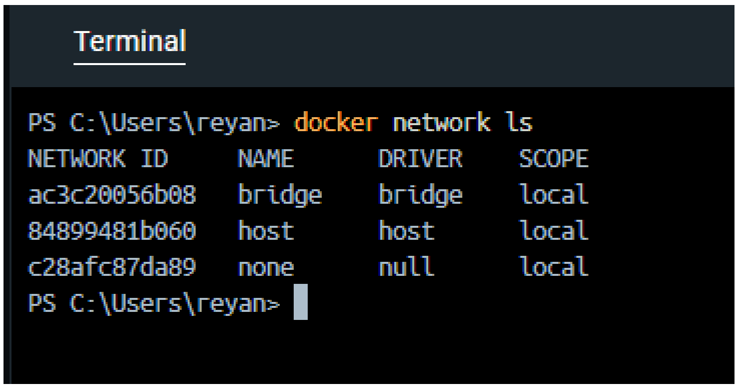
Another way of networking is HOST NETWORKING. It removes the network isolation between the host and the containers.

Host networking can be defined as the method in which the container and the host have ip addresses in the same cidr range.

This method of networking is not recommended as every container uses the same docker0 eth. So security is compromised.

The third type of networking is OVERLAY NETWORKING. This type of networking is used for Kubernetes clusters and docker swarm.

To create network with either bridge, host and overlay network, and manage it, use the reference link- https://docs.docker.com/reference/cli/docker/network/



```
PS C:\Users\reyan> docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
ac3c20056b08   bridge    bridge    local
84899481b060   host      host      local
c28afc87da89   none      null      local
PS C:\Users\reyan>
```

NETWORKS

Reference-
https://docs.docker.com/engine/network/#published-ports

By default, docker containers are not exposed outside the host system unless we use the –publish or -p command during docker run.

Ex docker run -p 8080:80

Means that if traffic comes on port 8080 of the host machine, it is redirected to port 80 of the container.

Another example- docker run -p 8080:80/tcp

Important-

If you want to make a container accessible to other containers, it isn't necessary to publish the container's ports. You can enable inter-container communication by connecting the containers to the same network, usually a bridge network.

DOCKER COMPOSE

Docker compose CLI reference- https://docs.docker.com/reference/cli/docker/compose/

Docker compose is a tool used to manage multi-container applications. It is used for local development before deploying to kubernetes, for CI/CD and testing for QE.

Docker compose solves problems with traditional docker execution style using docker build and docker run-

1. If we wanted to run multiple containers, it is time consuming to run every command.
2. It is easier to manage many container deployment and management lifecycles.

To use docker compose, we have to first build our images, and probably store those images in a repository. Then we can reference those images in our docker-compose.yml file.

Syntax of a docker-compose.yml file-

Reference-
https://docs.docker.com/reference/compose-file/services/

To start, we have to give a name to the container that we want to run-

```
services:

        Name1:



        Name2:
```

To build an image from docker compose itself, we can use the build command.

```
services:

        Name1:

                build: <dockerfile location>

        Name2:

                build: <dockerfile location>
```

To use an image, use the image command.

```
services:
    Name1:
        image: image from repo

    Name2:
        image: image from repo
```

You can use depends_on command to start containers before that particular container.

We can reference the name of the container or the hostname if we specifically mention it to reference that container in other containers for networking.

To copy files from local to docker images, use the “volume:” command.

Ex. volume:

```
./sitename.conf:/etc/nginx/sites-available
```

To use .env files where we store information, use this-

```
docker compose --env-file <file> up
```

To use variables in .env files, use this-

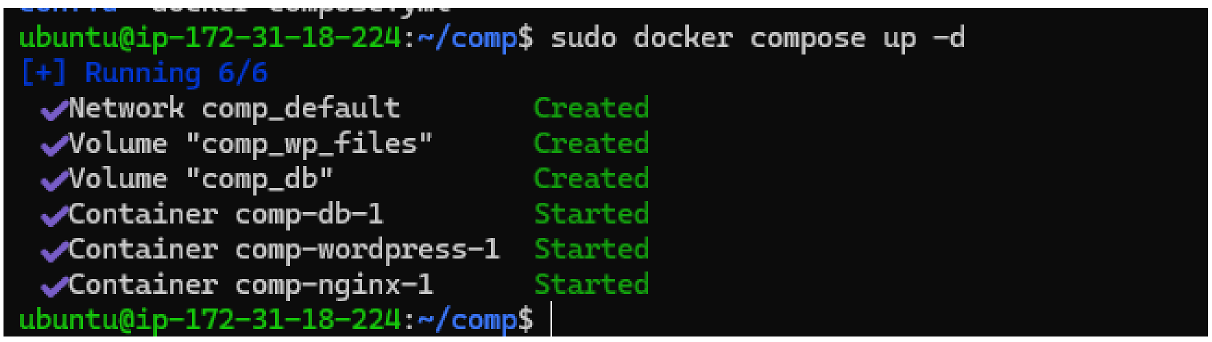
```
${Variablename}
```

For more reference-

https://www.warp.dev/terminus/docker-compose-env-file?gad_source=1&gbraid=0AAAAAoTBvvqGxN13wB4e9MIl6PgGAHls1&gclid=Cj0KCQjwzrzABhD8ARIsANlSWNOEKNpJFS17NIWrdLVYD2aGhvvdQmCaLo41MWXhTgI6vOLQp0XxHNkaArCvEALw_wcB

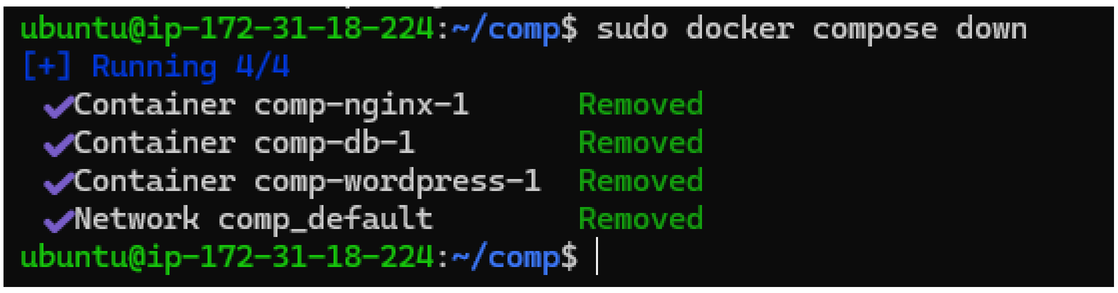
DOCKER COMPOSE CLI COMMANDS-

After creating a docker-compose.yml file, we can start our containers using the docker compose up -d command.



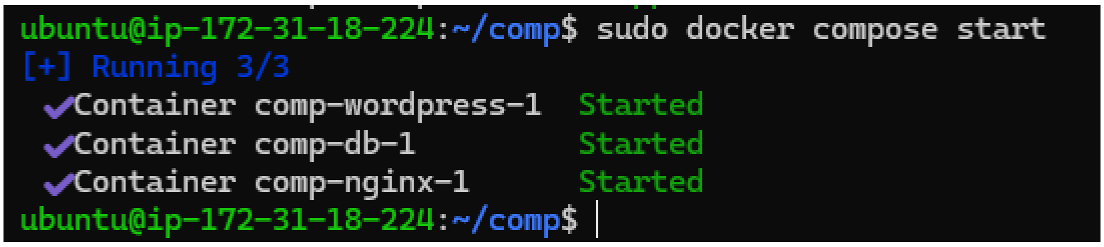
```
ubuntu@ip-172-31-18-224:~/comp$ sudo docker compose up -d
[+] Running 6/6
 ✔Network comp_default          Created
 ✔Volume "comp_wp_files"        Created
 ✔Volume "comp_db"              Created
 ✔Container comp-db-1           Started
 ✔Container comp-wordpress-1    Started
 ✔Container comp-nginx-1        Started
ubuntu@ip-172-31-18-224:~/comp$
```

To stop running the containers use the docker compose down command.

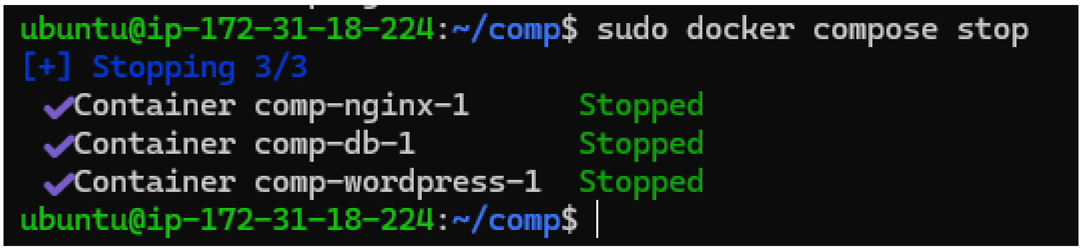


```
ubuntu@ip-172-31-18-224:~/comp$ sudo docker compose down
[+] Running 4/4
 ✔Container comp-nginx-1        Removed
 ✔Container comp-db-1           Removed
 ✔Container comp-wordpress-1    Removed
 ✔Network comp_default          Removed
ubuntu@ip-172-31-18-224:~/comp$
```

You can also start and stop containers without removing them. Use the docker compose start and stop commands.



```
ubuntu@ip-172-31-18-224:~/comp$ sudo docker compose start
[+] Running 3/3
 ✔Container comp-wordpress-1  Started
 ✔Container comp-db-1         Started
 ✔Container comp-nginx-1      Started
ubuntu@ip-172-31-18-224:~/comp$ |
```



```
ubuntu@ip-172-31-18-224:~/comp$ sudo docker compose stop
[+] Stopping 3/3
 ✔Container comp-nginx-1      Stopped
 ✔Container comp-db-1         Stopped
 ✔Container comp-wordpress-1  Stopped
ubuntu@ip-172-31-18-224:~/comp$ |
```

DOCKER COMPOSE EXAMPLE-

NGINX, WORDPRESS, MYSQL DOCKER COMPOSE DEPLOYMENT

(IMP)Reference Link- https://hub.docker.com/_/wordpress

Bonus reference- https://github.com/atif089/wordpress-docker-compose

We are going to host a wordpress application with mysql using nginx.

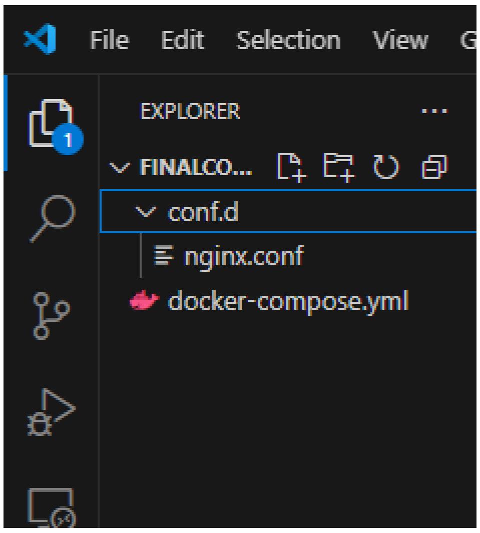
I am going to first create a nginx.conf file to correctly use the port 80 and to configure the fastcgi php-

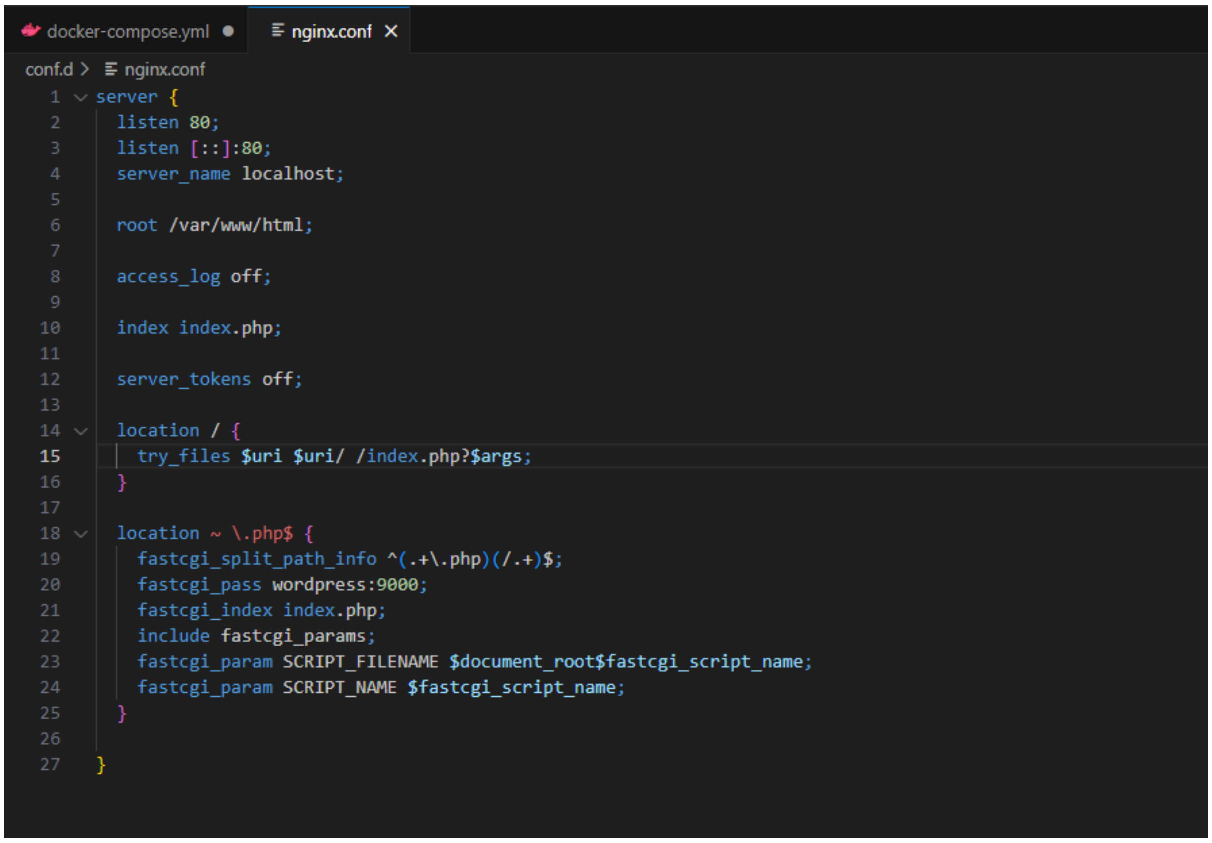
```
location ~ \.php$ {
```

```
fastcgi_split_path_info ^(.+\.php)(/.+)$;
fastcgi_pass wordpress:9000;
fastcgi_index index.php;
include fastcgi_params;
fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
fastcgi_param SCRIPT_NAME $fastcgi_script_name;
}
```

Here, fastcgi_pass wordpress:9000; the “wordpress” refers to the container name.

The complete nginx file-




docker-compose.yml ● nginx.conf ×

conf.d > nginx.conf

```
server {
    listen 80;
    listen [::]:80;
    server_name localhost;

    root /var/www/html;

    access_log off;

    index index.php;

    server_tokens off;

    location / {
      try_files $uri $uri/ /index.php?$args;
    }

    location ~ \.php$ {
      fastcgi_split_path_info ^(.+\.php)(/.+)$;
      fastcgi_pass wordpress:9000;
      fastcgi_index index.php;
      include fastcgi_params;
      fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
      fastcgi_param SCRIPT_NAME $fastcgi_script_name;
    }

}
```

Next, we are going to create a docker-compose.yml file. The docker compose is going to run three containers. Nginx, wordpress and mysql.

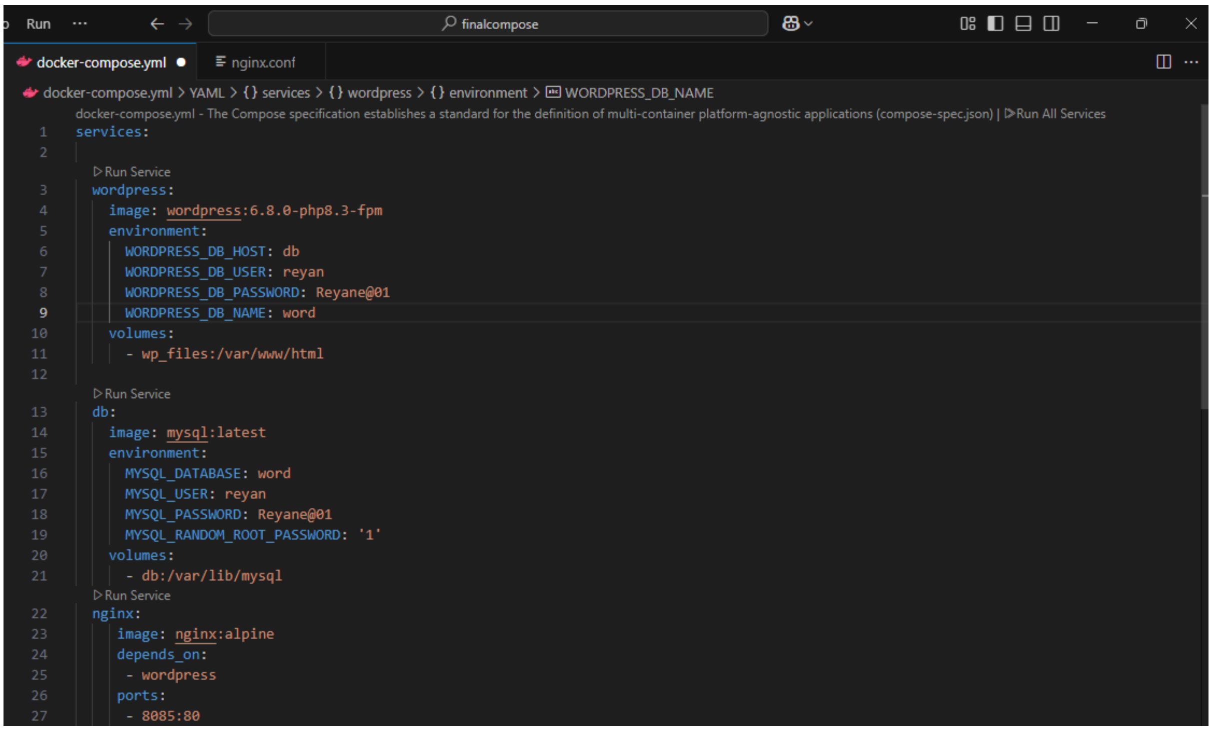
For persistent storage, we are going to use two volumes, wp_files and db.

(VERY IMPORTANT: USE ONE VOLUME TO STORE THE WORDPRESS CONTAINER /var/www/html FILES, IE THE WORDPRESS MAIN FILES. THEN USE THE SAME VOLUME AND MOUNT IT TO THE NGINX

CONTAINER, WHERE IT REFERENCES THE /var/www/html FILES.

This is because the wordpress data is being stored locally in the host machine and only connected to the wordpress container. If we don't mention the same volume, nginx tries to look inside wordpress /var/www/html and can't access the files because the files are actually in the volume in the host machine.

The docker-compose.yml file-

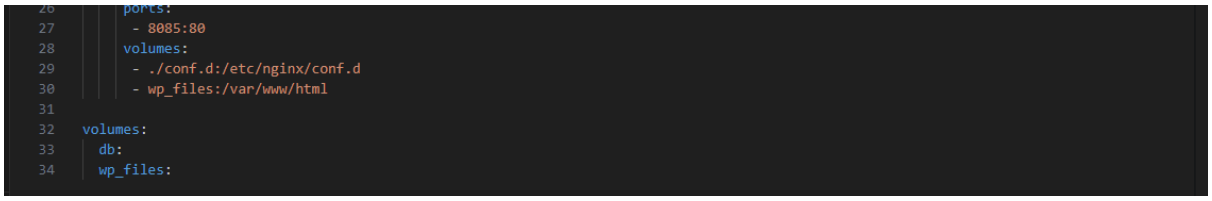


docker-compose.yml > YAML > {} services > {} wordpress > {} environment > WORDPRESS_DB_NAME

docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json) | ▷Run All Services

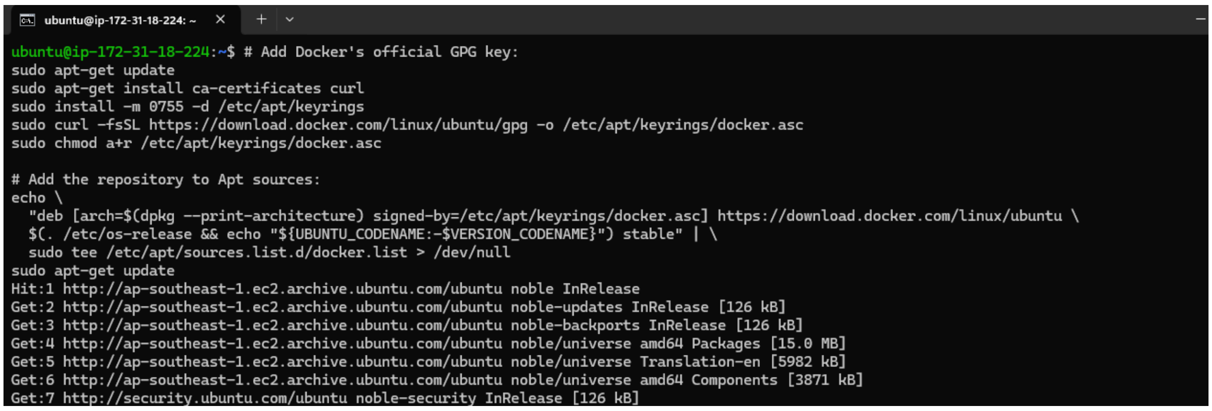
```yaml
services:

  wordpress:
    image: wordpress:6.8.0-php8.3-fpm
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: reyan
      WORDPRESS_DB_PASSWORD: Reyane@01
      WORDPRESS_DB_NAME: word
    volumes:
      - wp_files:/var/www/html

  db:
    image: mysql:latest
    environment:
      MYSQL_DATABASE: word
      MYSQL_USER: reyan
      MYSQL_PASSWORD: Reyane@01
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - db:/var/lib/mysql
  nginx:
     image: nginx:alpine
     depends_on:
      - wordpress
     ports:
      - 8085:80
```

We are going to create 2 volumes on our host machine, called wp_files and db.

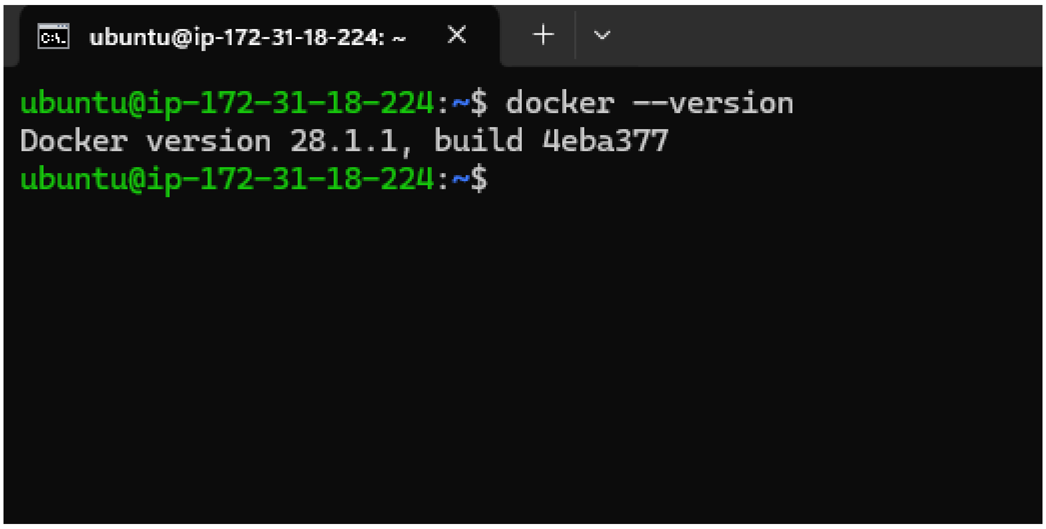


```yaml
     ports:
      - 8085:80
     volumes:
      - ./conf.d:/etc/nginx/conf.d
      - wp_files:/var/www/html

volumes:
  db:
  wp_files:
```

Install docker on our host machine-

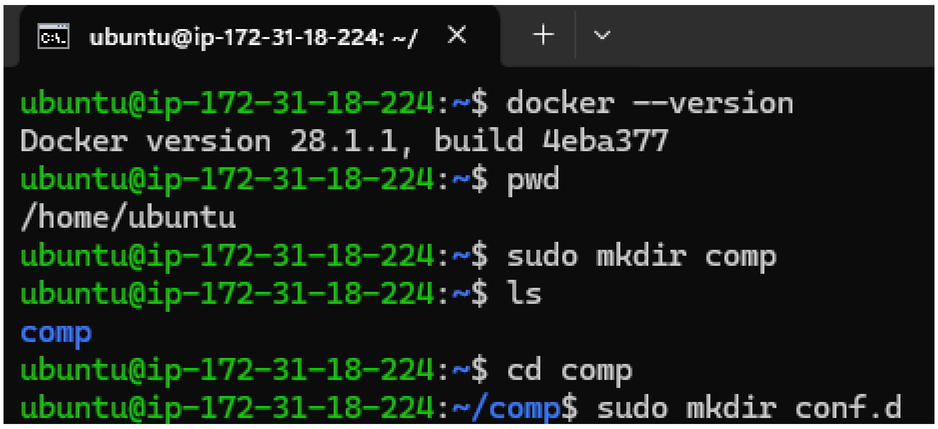


```
ubuntu@ip-172-31-18-224:~$ # Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
Hit:1 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:5 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:6 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
```



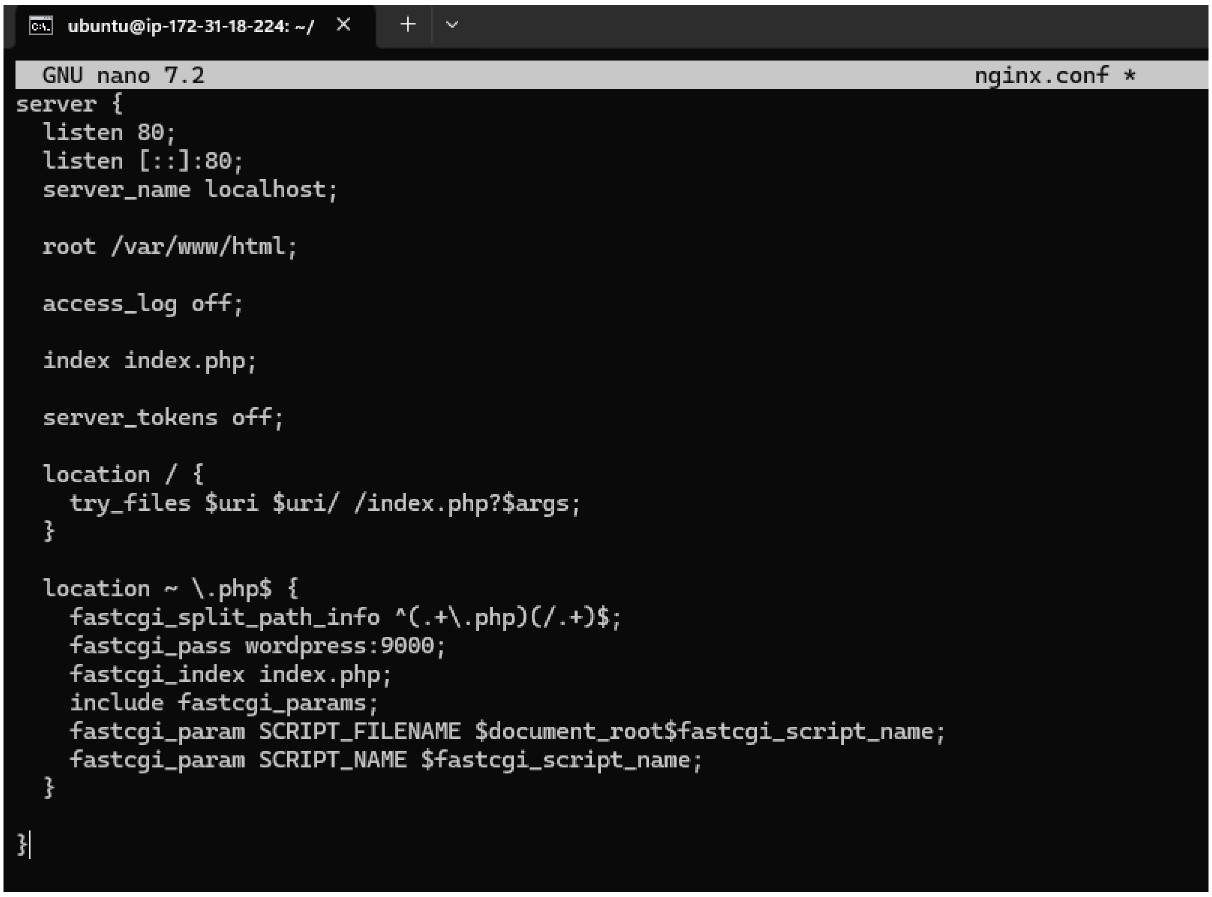
```
ubuntu@ip-172-31-18-224:~$ docker --version
Docker version 28.1.1, build 4eba377
ubuntu@ip-172-31-18-224:~$
```

After that, I am going to create a new directory called comp to set up our docker compose.



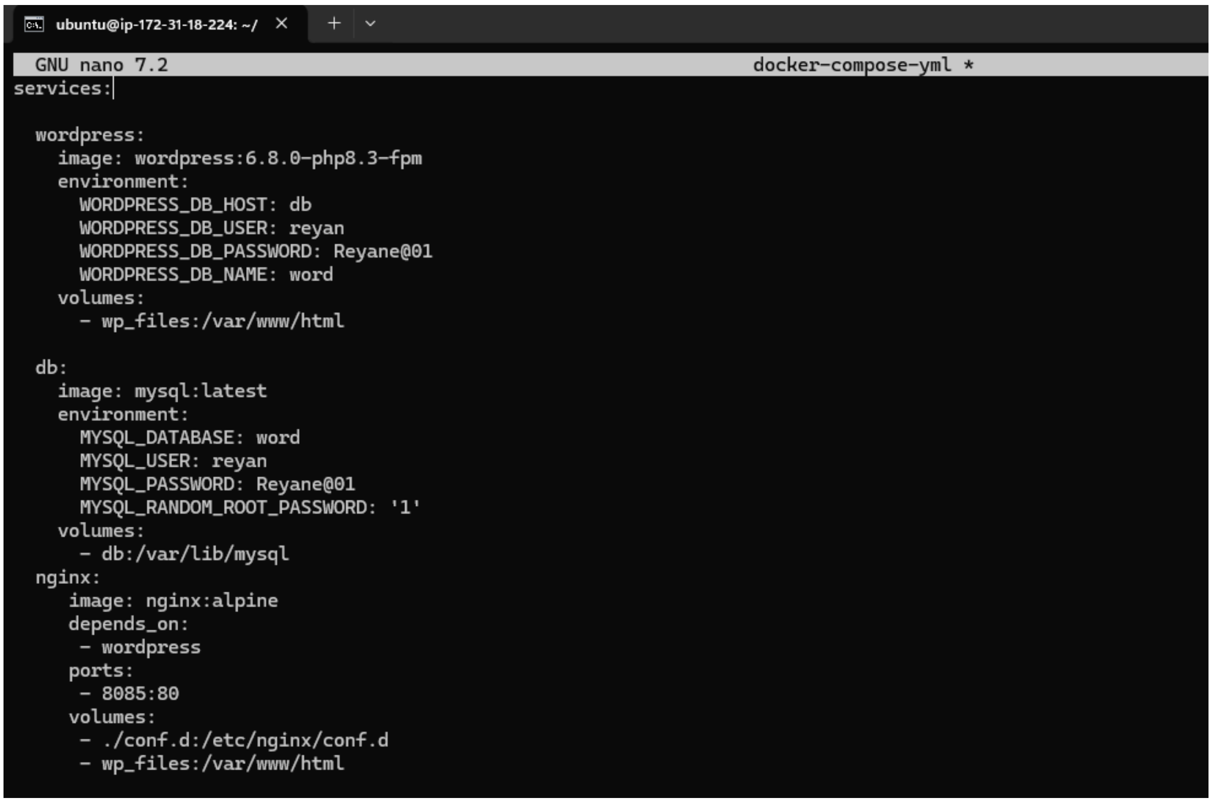
```
ubuntu@ip-172-31-18-224:~$ docker --version
Docker version 28.1.1, build 4eba377
ubuntu@ip-172-31-18-224:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-18-224:~$ sudo mkdir comp
ubuntu@ip-172-31-18-224:~$ ls
comp
ubuntu@ip-172-31-18-224:~$ cd comp
ubuntu@ip-172-31-18-224:~/comp$ sudo mkdir conf.d
```

Creating and pasting the appropriate code.



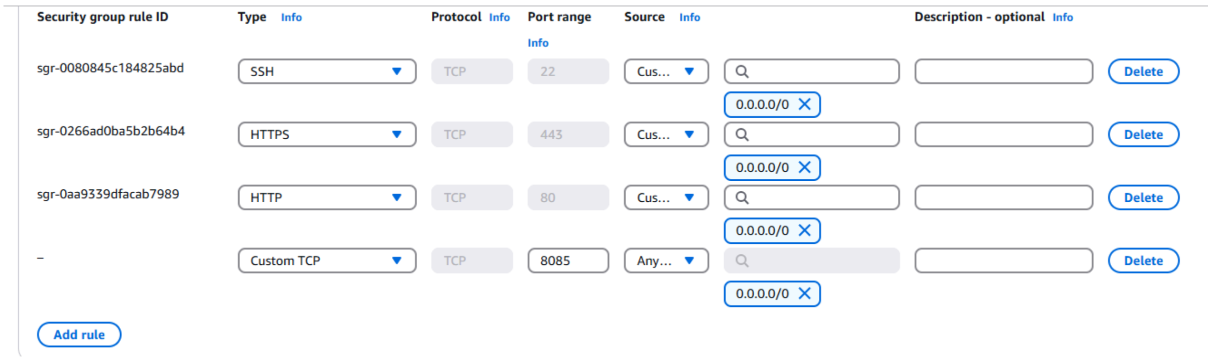
```
  GNU nano 7.2                                                         nginx.conf *
server {
  listen 80;
  listen [::]:80;
  server_name localhost;

  root /var/www/html;

  access_log off;

  index index.php;

  server_tokens off;

  location / {
    try_files $uri $uri/ /index.php?$args;
  }

  location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass wordpress:9000;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param SCRIPT_NAME $fastcgi_script_name;
  }

}
```

Compose.yml file-



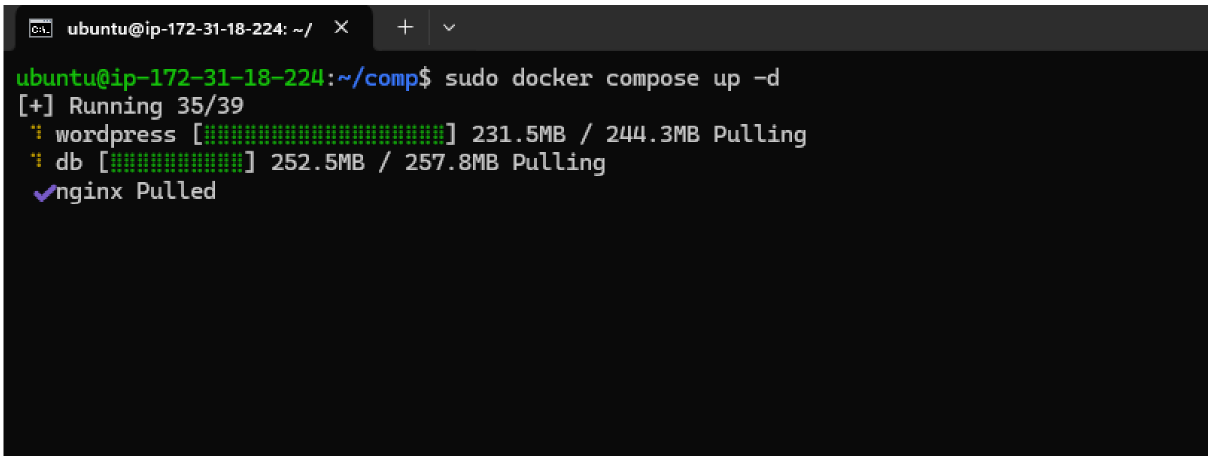
```
  GNU nano 7.2                                                     docker-compose-yml *
services:

  wordpress:
    image: wordpress:6.8.0-php8.3-fpm
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: reyan
      WORDPRESS_DB_PASSWORD: Reyane@01
      WORDPRESS_DB_NAME: word
    volumes:
      - wp_files:/var/www/html

  db:
    image: mysql:latest
    environment:
      MYSQL_DATABASE: word
      MYSQL_USER: reyan
      MYSQL_PASSWORD: Reyane@01
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - db:/var/lib/mysql
  nginx:
     image: nginx:alpine
     depends_on:
      - wordpress
     ports:
      - 8085:80
     volumes:
      - ./conf.d:/etc/nginx/conf.d
      - wp_files:/var/www/html
```

We need to open port 8085 on our aws EC2 instance.



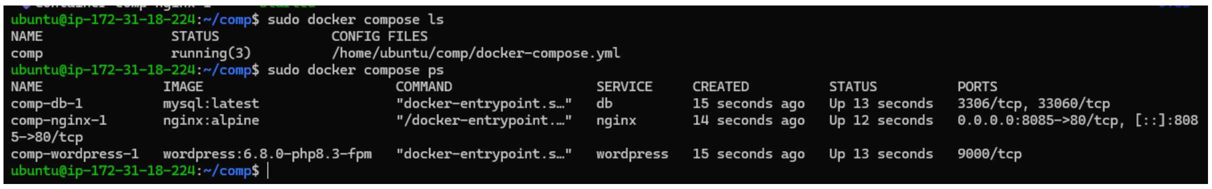
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info			Description - optional Info	
sgr-0080845c184825abd	SSH	TCP	22	Cus...	0.0.0.0/0			Delete
sgr-0266ad0ba5b2b64b4	HTTPS	TCP	443	Cus...	0.0.0.0/0			Delete
sgr-0aa9339dfacab7989	HTTP	TCP	80	Cus...	0.0.0.0/0			Delete
–	Custom TCP	TCP	8085	Any...	0.0.0.0/0			Delete

From the directory where the compose.yml file exists, use the docker compose up -d command.



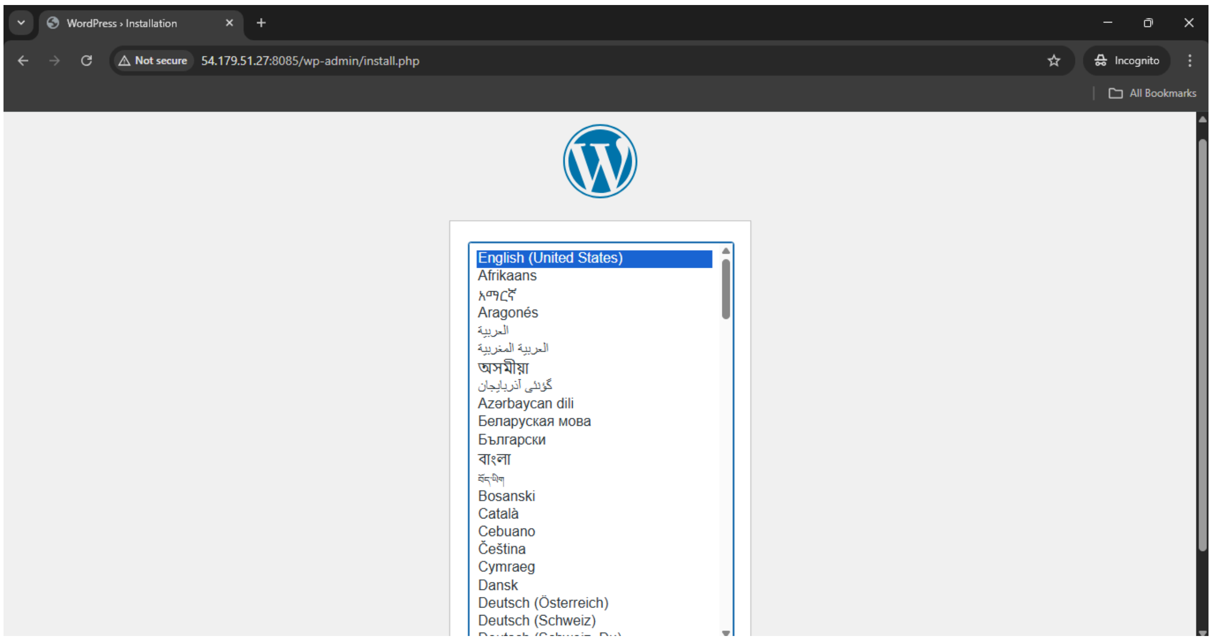
```
ubuntu@ip-172-31-18-224: ~/

ubuntu@ip-172-31-18-224:~/comp$ sudo docker compose up -d
[+] Running 35/39
 ⠸ wordpress [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿] 231.5MB / 244.3MB Pulling
 ⠸ db [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿] 252.5MB / 257.8MB Pulling
 ✔nginx Pulled
```

We can see that the containers are running successfully.

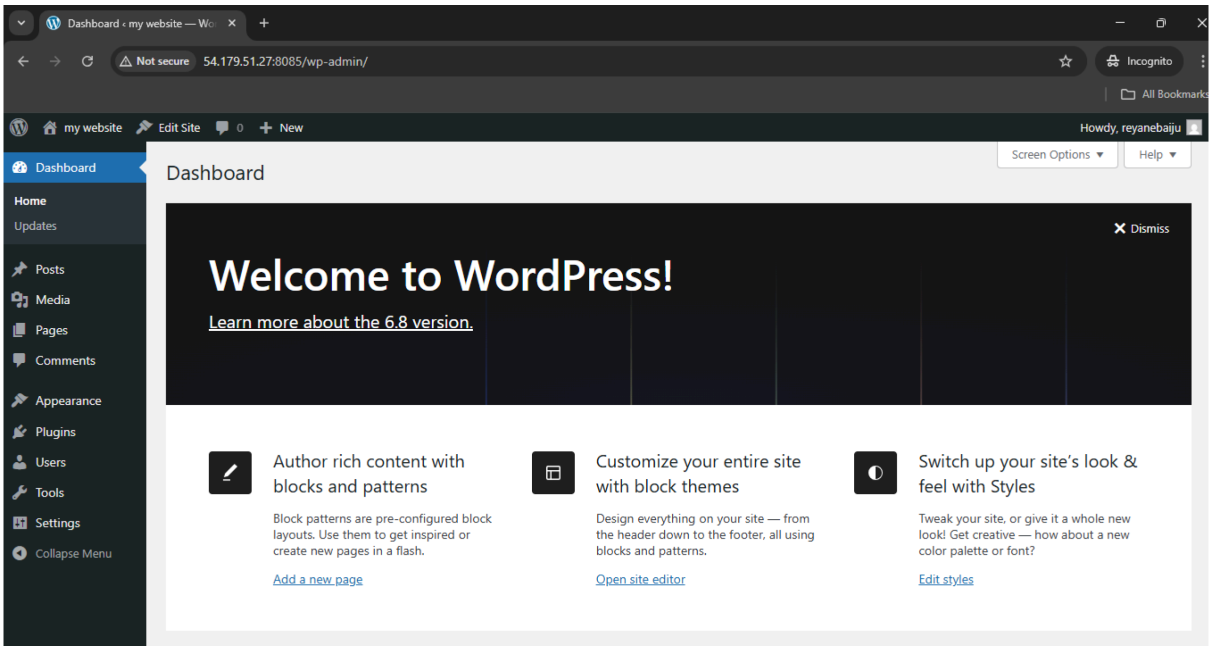


```
ubuntu@ip-172-31-18-224:~/comp$ sudo docker compose ls
NAME                STATUS              CONFIG FILES
comp                running(3)          /home/ubuntu/comp/docker-compose.yml
ubuntu@ip-172-31-18-224:~/comp$ sudo docker compose ps
NAME                IMAGE                       COMMAND                  SERVICE     CREATED          STATUS          PORTS
comp-db-1           mysql:latest                "docker-entrypoint.s…"   db          15 seconds ago   Up 13 seconds   3306/tcp, 33060/tcp
comp-nginx-1        nginx:alpine                "/docker-entrypoint.…"   nginx       14 seconds ago   Up 12 seconds   0.0.0.0:8085->80/tcp, [::]:808
5->80/tcp
comp-wordpress-1    wordpress:6.8.0-php8.3-fpm  "docker-entrypoint.s…"   wordpress   15 seconds ago   Up 13 seconds   9000/tcp
ubuntu@ip-172-31-18-224:~/comp$ |
```

If we try to access the website using http://ipaddress:8085, we are greeted with this page.



After logging in, we are greeted with this page.


Dashboard ‹ my website — Wo
Not secure
54.179.51.27:8085/wp-admin/
Incognito
All Bookmarks
my website
Edit Site
0
New
Howdy, reyanebaiju
Dashboard
Home
Updates
Posts
Media
Pages
Comments
Appearance
Plugins
Users
Tools
Settings
Collapse Menu
Screen Options
Help
Dismiss
Welcome to WordPress!
Learn more about the 6.8 version.
Author rich content with blocks and patterns
Block patterns are pre-configured block layouts. Use them to get inspired or create new pages in a flash.
Add a new page
Customize your entire site with block themes
Design everything on your site — from the header down to the footer, all using blocks and patterns.
Open site editor
Switch up your site's look & feel with Styles
Tweak your site, or give it a whole new look! Get creative — how about a new color palette or font?
Edit styles

Now, we can edit to our liking.

DOCKERFILE

The dockerfile is a file that tells the docker engine how to build a docker image. After creating a dockerfile, we can use the docker build command to create a docker image.

Ex. docker build -t myapp:1.0 .

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

A dockerfile must begin with a FROM Command-

Ex. FROM ubuntu:latest

Then we could declare a working directory using the WORKDIR command-

Ex. WORKDIR /app

After that, we can use the COPY command to copy files between the current machine into the docker image.

Syntax- COPY <source> <destination>

Ex. COPY nginx.conf /etc/nginx/conf.d

You can use the RUN command to run commands inside the container during building.

Ex. RUN apt update && apt install nginx -y

There should only be one CMD command in a dockerfile. If there are multiple, then the last one is executed. CMD command is used to execute whatever is given when running a container.

Ex. CMD [“nginx”, “-g”, “daemon off;”]

The EXPOSE command is used to expose a port in the container.

Ex. EXPOSE 80/tcp

The ENTRYPOINT command allows you to configure a container that will run as an executable.

Ex. ENTRYPOINT ["executable", "param1", "param2"]

You can use ENTRYPOINT to execute stable commands that won’t be changed, while using CMD to execute often changing commands.

The ADD command is similar to the COPY command, but ADD supports features for fetching files from remote HTTPS and Git URLs, and extracting tar files automatically when adding files from the build context.

Ex. ADD

```
--checksum=sha256:270d731bd08040c6a3228115de1f
74b91cf441c584139ff8f8f6503447cebdbb \



https://dotnetcli.azureedge.net/dotnet/Runtime
/$DOTNET_VERSION/dotnet-runtime-$DOTNET_VERSIO
N-linux-arm64.tar.gz /dotnet.tar.gz
```

The USER command sets the default user and group for the container.
Ex. USER <user>[:<group>]

Example of a dockerfile that creates a nodejs image-

```
FROM node:latest
WORKDIR /app
COPY package.json .
```

```
RUN npm install
COPY . .
CMD ["node","app.js"]
EXPOSE 3000
```

Another example of a dockerfile that creates a custom nginx image-

```
FROM nginx:1.27.3
COPY index.html /usr/share/nginx/html/index.html
COPY default.conf /etc/nginx/conf.d/default.conf
EXPOSE 80

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

We also need to look at other dockerfile types examples for python images, java, go etc.

1. PHP image

```
FROM php:8.2-fpm

WORKDIR /var/www

COPY . .

RUN apt-get update && apt-get install -y \
libpq-dev \
&& docker-php-ext-install pdo pdo_mysql

CMD ["php-fpm"]
```

Imp- This php image cannot serve this to users directly. We need to use either apache or nginx with php-fpm installed to use this. The php files are sent to the fastcgi processor and then served via http.

2. .NET

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
COPY . .

RUN dotnet publish -c Release -o out

CMD ["dotnet", "out/MyDotNetApp.dll"]
```

3. GOLANG

```
FROM golang:1.20 AS build
WORKDIR /app
COPY . .

RUN go build -o myapp

FROM alpine:latest
WORKDIR /root/
COPY --from=build /app/myapp .
CMD ["./myapp"]
```

Here, we are using a multistage dockerfile to reduce size, but we can use the image called “scratch” to run golang directly.

4. RUBY ON RAILS

```
FROM ruby:3.1
WORKDIR /app
```

```
COPY . .

RUN bundle install
CMD ["rails", "server", "-b", "0.0.0.0"]
```

5. C++

```
FROM gcc:latest
WORKDIR /app
COPY . .

RUN g++ -o myapp src/main.cpp
CMD ["./myapp"]
```

7. RUST

```
FROM rust:1.70 AS build
WORKDIR /app
COPY . .

RUN cargo build --release

FROM debian:bullseye
WORKDIR /app
COPY --from=build /app/target/release/myapp .
CMD ["./myapp"]
```

8. PYTHON

```
FROM python:3.11-slim

WORKDIR /app
```

```
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

DOCKER SWARM

Docker swarm is a container orchestration tool like k8s.
Container orchestration is the process of deploying and maintaining a large number of containers.
Docker swarm contains at least one manager node and worker nodes.
The docker manager node scales and maintains the cluster of docker host nodes.
The docker manager is responsible for the correct working of the deployed containers in the worker nodes.

Problems that Docker swarm solves-

1. Single host problem-
When we deploy containers independently using docker run or multiple containers using docker compose, we are limited to only that host machine.
If there is any issue or fault with the host, our application goes down.
Docker swarm enables us to replicate and deploy containers to multiple hosts as a cluster, to ensure high availability.
2. Autohealing
Docker swarm automatically detects faulty or crashed containers and can spin up new ones when this is detected.
3.Scaling

We can specify how many replicas of tasks that we want to have and docker will scale accordingly.

4. Resource management

We can allocate correct resources to the container by specifying them.

DOCKER SWARM CLI COMMANDS-
https://docs.docker.com/reference/cli/docker/swarm/

DOCKER SERVICE CLI COMMANDS-
https://docs.docker.com/reference/cli/docker/service/

DOCKER SWARM COMPONENTS

1.SERVICE-

Services define the tasks that need to be executed on the manager and worker nodes.

2. TASK

Tasks refer to the docker containers that are in the separate worker nodes.

3. MANAGER NODE

Manager nodes receive commands to run tasks in worker nodes, allocating IP addresses to tasks, assigning tasks to nodes, and instructing nodes to run tasks.

4. WORKER NODE

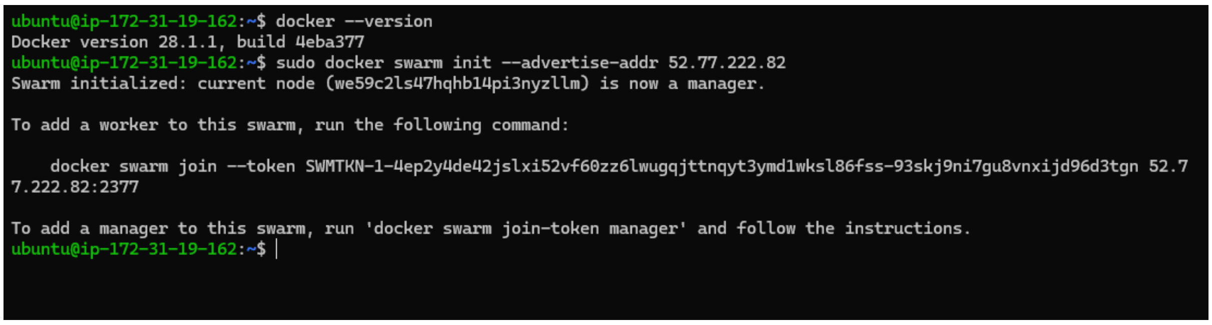
Check assigned tasks and execute containers inside them.

BUILDING A DOCKER SWARM

To build a docker swarm, we have to first initialize a node as the manager node.

We can use the command docker swarm init –advertise-addr <manager-ip>

I'm going to use an EC2 instance as our manager node.



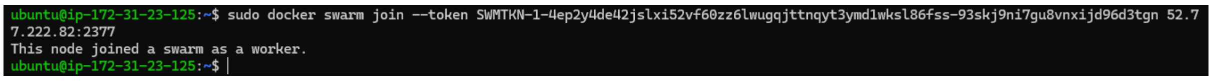
```
ubuntu@ip-172-31-19-162:~$ docker --version
Docker version 28.1.1, build 4eba377
ubuntu@ip-172-31-19-162:~$ sudo docker swarm init --advertise-addr 52.77.222.82
Swarm initialized: current node (we59c2ls47hqhb14pi3nyzllm) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-4ep2y4de42jslxi52vf60zz6lwugqjttnqyt3ymd1wksl86fss-93skj9ni7gu8vnxijd96d3tgn 52.7
7.222.82:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
ubuntu@ip-172-31-19-162:~$ |
```

We will get a command that looks like this- docker swarm join --token SWMTKN-1-4ep2y4de42jslxi52vf60zz6lwugqjttnqyt3ymd1wksl86fss-93skj9ni7gu8vnxijd96d3tgn 52.77.222.82:2377

In the last part, we can see a port number 2377. (IMP)This port should be allowed on the managed node security group.

We can then run the docker swarm join –token command to add worker nodes to this manager.

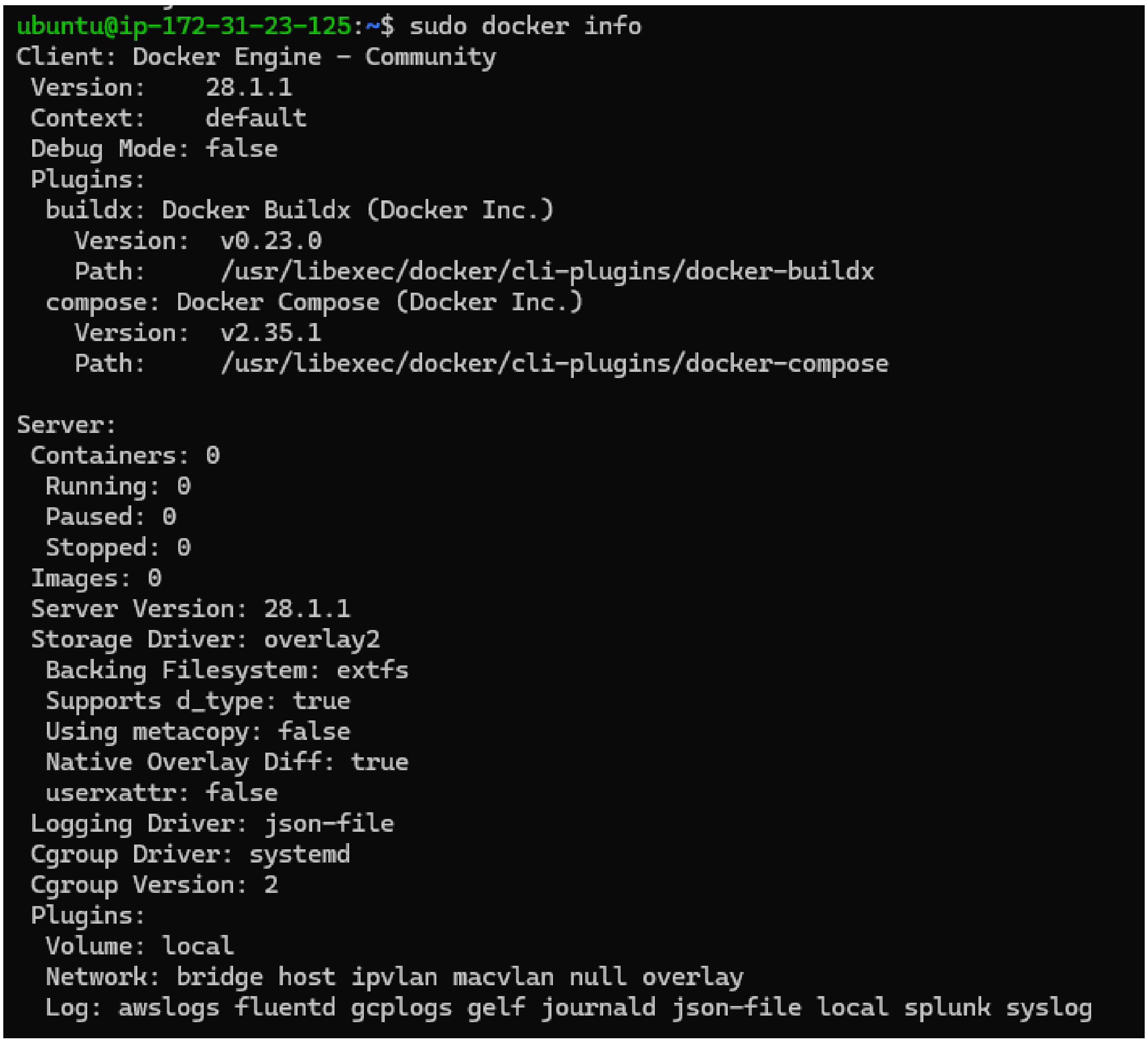
(TIP- TO GET THE JOIN TOKEN COMMAND AGAIN, USE THE COMMAND "sudo docker swarm join-token worker")



```
ubuntu@ip-172-31-23-125:~$ sudo docker swarm join --token SWMTKN-1-4ep2y4de42jslxi52vf60zz6lwugqjttnqyt3ymd1wksl86fss-93skj9ni7gu8vnxijd96d3tgn 52.7
7.222.82:2377
This node joined a swarm as a worker.
ubuntu@ip-172-31-23-125:~$ |
```

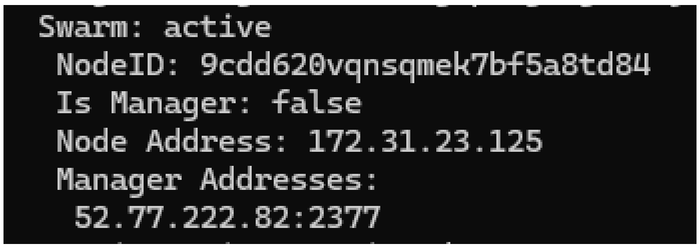
We can see that node has joined the swarm as a worker.

TO CONFIRM, USE DOCKER INFO.



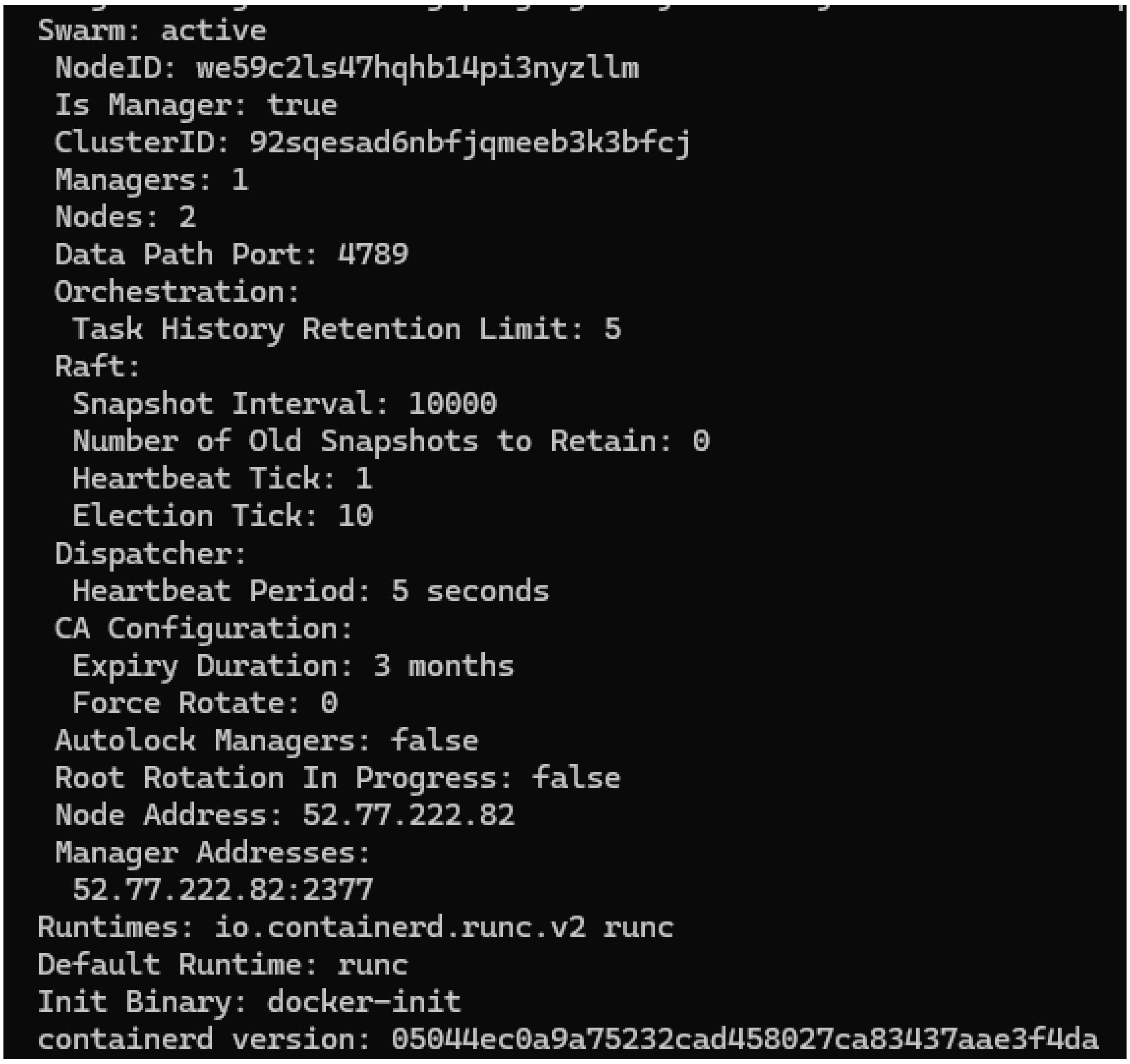
```
ubuntu@ip-172-31-23-125:~$ sudo docker info
Client: Docker Engine - Community
 Version:    28.1.1
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.23.0
    Path:     /usr/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version:  v2.35.1
    Path:     /usr/libexec/docker/cli-plugins/docker-compose

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 28.1.1
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: systemd
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local splunk syslog
```

We can see this part where it says “swarm=active”



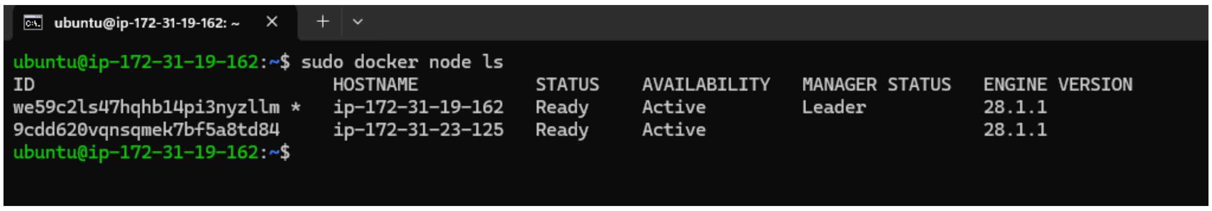
```
Swarm: active
 NodeID: 9cdd620vqnsqmek7bf5a8td84
 Is Manager: false
 Node Address: 172.31.23.125
 Manager Addresses:
  52.77.222.82:2377
```

If we check the manager node info, we can see that the node is successfully added.



```
Swarm: active
 NodeID: we59c2ls47hqhb14pi3nyzllm
 Is Manager: true
 ClusterID: 92sqesad6nbfjqmeeb3k3bfcj
 Managers: 1
 Nodes: 2
 Data Path Port: 4789
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 10
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 52.77.222.82
 Manager Addresses:
  52.77.222.82:2377
Runtimes: io.containerd.runc.v2 runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 05044ec0a9a75232cad458027ca83437aae3f4da
```

WE CAN USE THE DOCKER NODE LS COMMAND TO SEE ALL THE MANAGER AND WORKER NODES.

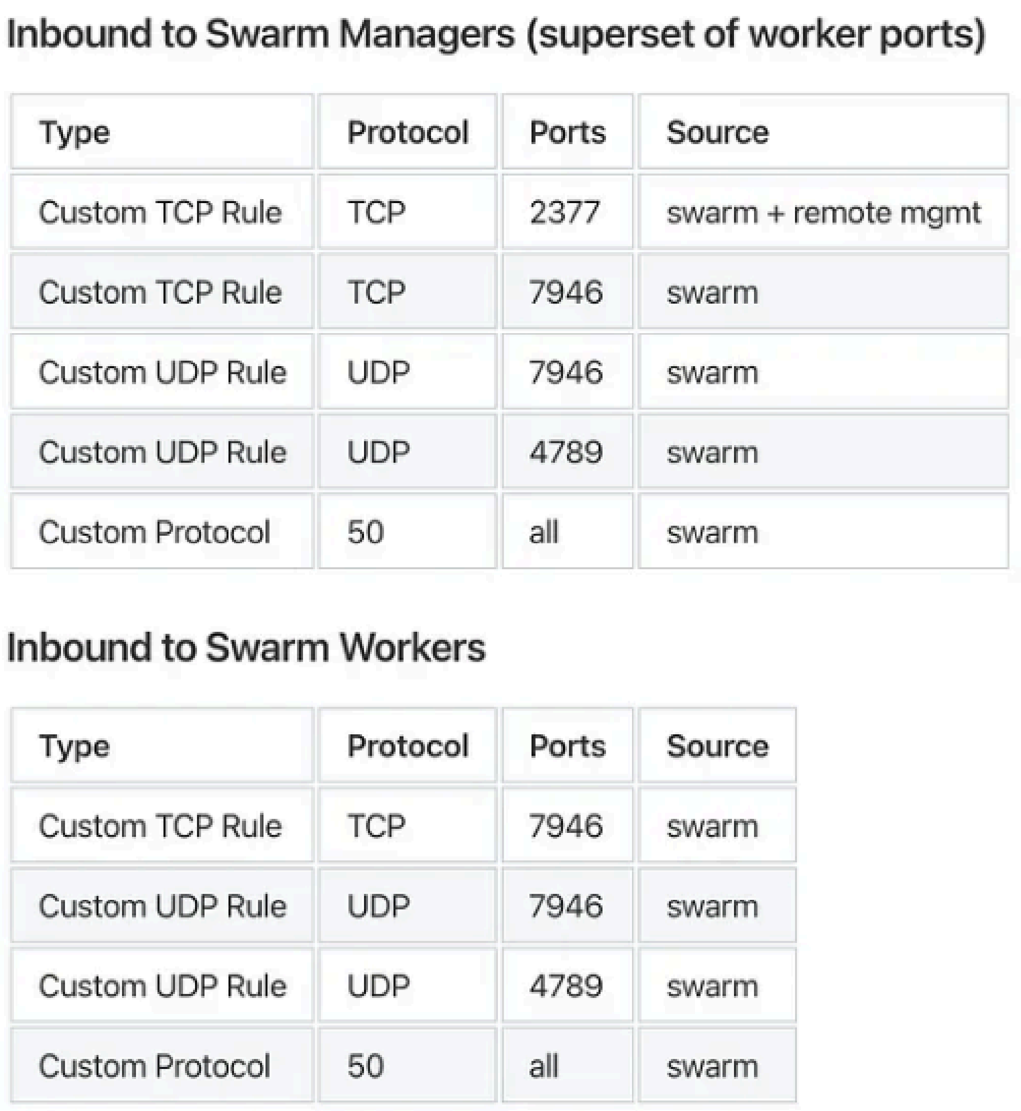


```
ubuntu@ip-172-31-19-162:~$ sudo docker node ls
ID                            HOSTNAME           STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
we59c2ls47hqhb14pi3nyzllm *   ip-172-31-19-162   Ready     Active         Leader           28.1.1
9cdd620vqnsqmek7bf5a8td84     ip-172-31-23-125   Ready     Active                          28.1.1
ubuntu@ip-172-31-19-162:~$
```

If we want a node to leave a swarm, use the docker swarm leave command, you can also use the –force flag.

We can also remove nodes from the manager node using the docker node rm <node-ID>

Also we have to open up some ports in the manager and worker node as docker swarm manages the cluster using different ports for different functions.
Important port configurations(check this)-
https://www.bretfisher.com/docker-swarm-firewall-ports/



Inbound to Swarm Managers (superset of worker ports)

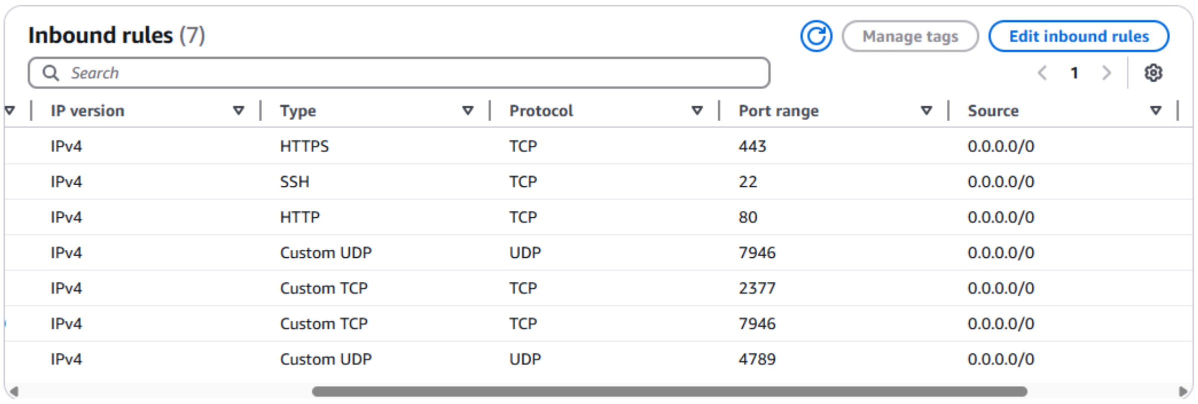
Type	Protocol	Ports	Source
Custom TCP Rule	TCP	2377	swarm + remote mgmt
Custom TCP Rule	TCP	7946	swarm
Custom UDP Rule	UDP	7946	swarm
Custom UDP Rule	UDP	4789	swarm
Custom Protocol	50	all	swarm

Inbound to Swarm Workers

Type	Protocol	Ports	Source
Custom TCP Rule	TCP	7946	swarm
Custom UDP Rule	UDP	7946	swarm
Custom UDP Rule	UDP	4789	swarm
Custom Protocol	50	all	swarm

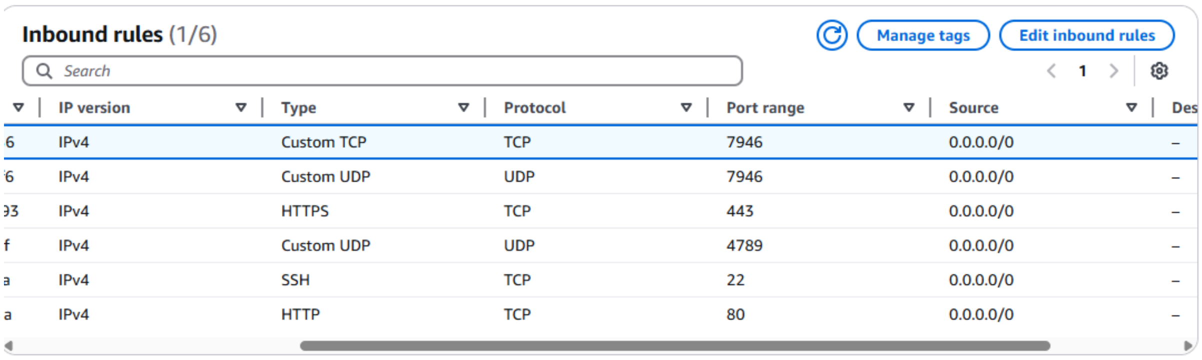
I have configure the security group of my manager and worker node like this-

Manager node-


Inbound rules (7)

IP version	Type	Protocol	Port range	Source
IPv4	HTTPS	TCP	443	0.0.0.0/0
IPv4	SSH	TCP	22	0.0.0.0/0
IPv4	HTTP	TCP	80	0.0.0.0/0
IPv4	Custom UDP	UDP	7946	0.0.0.0/0
IPv4	Custom TCP	TCP	2377	0.0.0.0/0
IPv4	Custom TCP	TCP	7946	0.0.0.0/0
IPv4	Custom UDP	UDP	4789	0.0.0.0/0

Worker node-


Inbound rules (1/6)

	IP version	Type	Protocol	Port range	Source	Des
6	IPv4	Custom TCP	TCP	7946	0.0.0.0/0	–
6	IPv4	Custom UDP	UDP	7946	0.0.0.0/0	–
93	IPv4	HTTPS	TCP	443	0.0.0.0/0	–
f	IPv4	Custom UDP	UDP	4789	0.0.0.0/0	–
a	IPv4	SSH	TCP	22	0.0.0.0/0	–
a	IPv4	HTTP	TCP	80	0.0.0.0/0	–

SERVICE

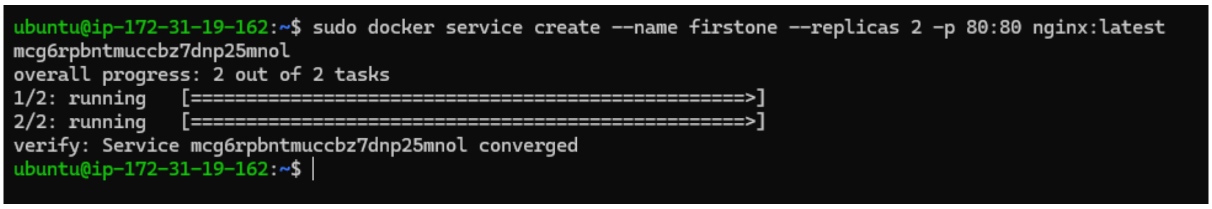
A service is used to deploy an application image.

To create a docker service, use the docker service create command.

EX SYNTAX-

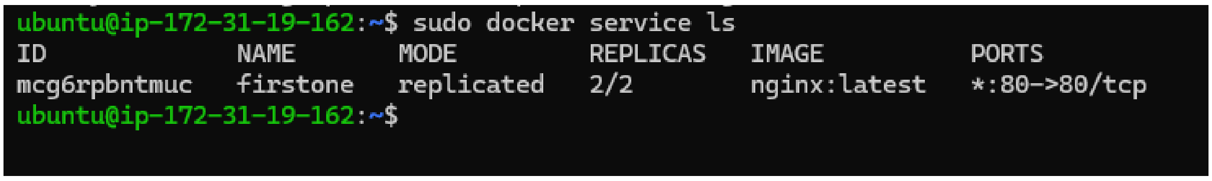
docker service create –name <name-of-service> –replicas <replica-no> –publish <port-mapping> <image-name>

I’m creating a service that deploys nginx. Using this command-sudo docker service create --name firstone --replicas 2 -p 80:80 nginx:latest



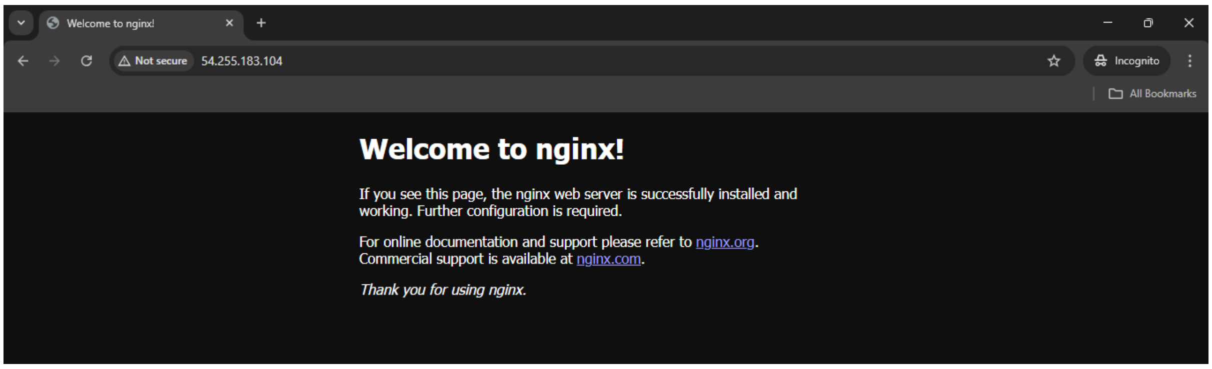
```
ubuntu@ip-172-31-19-162:~$ sudo docker service create --name firstone --replicas 2 -p 80:80 nginx:latest
mcg6rpbntmuccbz7dnp25mnol
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service mcg6rpbntmuccbz7dnp25mnol converged
ubuntu@ip-172-31-19-162:~$ |
```

You can use the docker service ls to see the live services.



```
ubuntu@ip-172-31-19-162:~$ sudo docker service ls
ID             NAME       MODE         REPLICAS   IMAGE          PORTS
mcg6rpbntmuc   firstone   replicated   2/2        nginx:latest   *:80->80/tcp
ubuntu@ip-172-31-19-162:~$
```

If we go to the ip address of the worker node, we can see that nginx is working successfully.



Welcome to nginx!

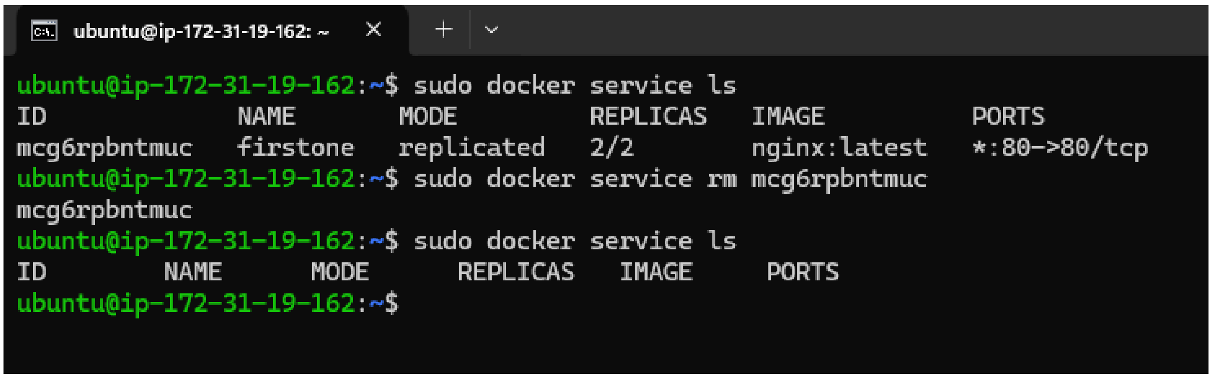
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

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

Thank you for using nginx.

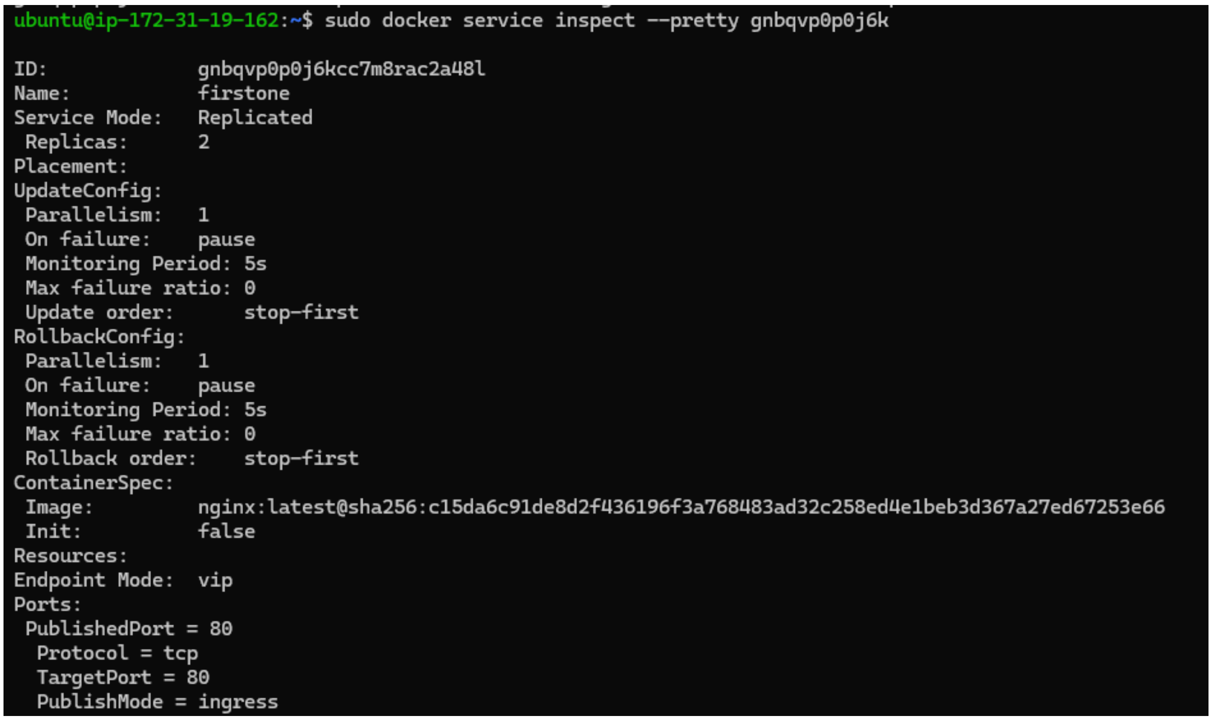
If the container in the worker node was to go down, docker swarm will redeploy it immediately.
TO REMOVE A SERVICE, WE CAN USE THE COMMAND-

‘Docker service rm <service-ID>’



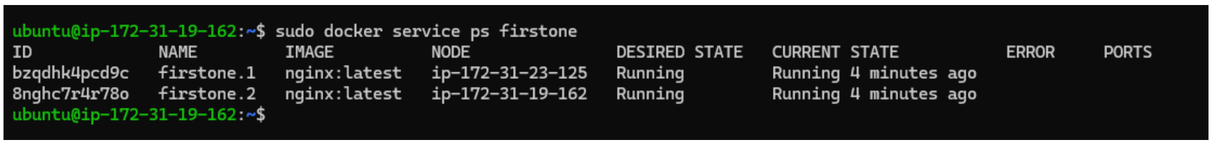
```
ubuntu@ip-172-31-19-162:~$ sudo docker service ls
ID             NAME       MODE         REPLICAS   IMAGE          PORTS
mcg6rpbntmuc   firstone   replicated   2/2        nginx:latest   *:80->80/tcp
ubuntu@ip-172-31-19-162:~$ sudo docker service rm mcg6rpbntmuc
mcg6rpbntmuc
ubuntu@ip-172-31-19-162:~$ sudo docker service ls
ID        NAME      MODE      REPLICAS   IMAGE     PORTS
ubuntu@ip-172-31-19-162:~$
```

To inspect a service, use the docker service inspect –pretty <service-ID>



```
ubuntu@ip-172-31-19-162:~$ sudo docker service inspect --pretty gnbqvp0p0j6k

ID:             gnbqvp0p0j6kcc7m8rac2a48l
Name:           firstone
Service Mode:   Replicated
 Replicas:      2
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         nginx:latest@sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66
 Init:          false
Resources:
Endpoint Mode:  vip
Ports:
 PublishedPort = 80
  Protocol = tcp
  TargetPort = 80
  PublishMode = ingress
```

To check the status and deployed nodes of each service, use the docker service ps <name-of-service>



```
ubuntu@ip-172-31-19-162:~$ sudo docker service ps firstone
ID             NAME         IMAGE          NODE               DESIRED STATE   CURRENT STATE           ERROR     PORTS
bzqdhk4pcd9c   firstone.1   nginx:latest   ip-172-31-23-125   Running         Running 4 minutes ago
8nghc7r4r78o   firstone.2   nginx:latest   ip-172-31-19-162   Running         Running 4 minutes ago
ubuntu@ip-172-31-19-162:~$
```

DEPLOYMENTS IN DOCKER SWARM

To deploy in docker swarm, we can use the “stack” command.

Docker stack CLI commands reference-
https://docs.docker.com/reference/cli/docker/stack/

The docker stack function uses a YAML file to deploy multiple services at once.

Syntax of docker stack command example-

```
docker stack deploy -c <filename.yml> <stack-name>
```

We can specify the number of replicas that we want in the YAML file itself.

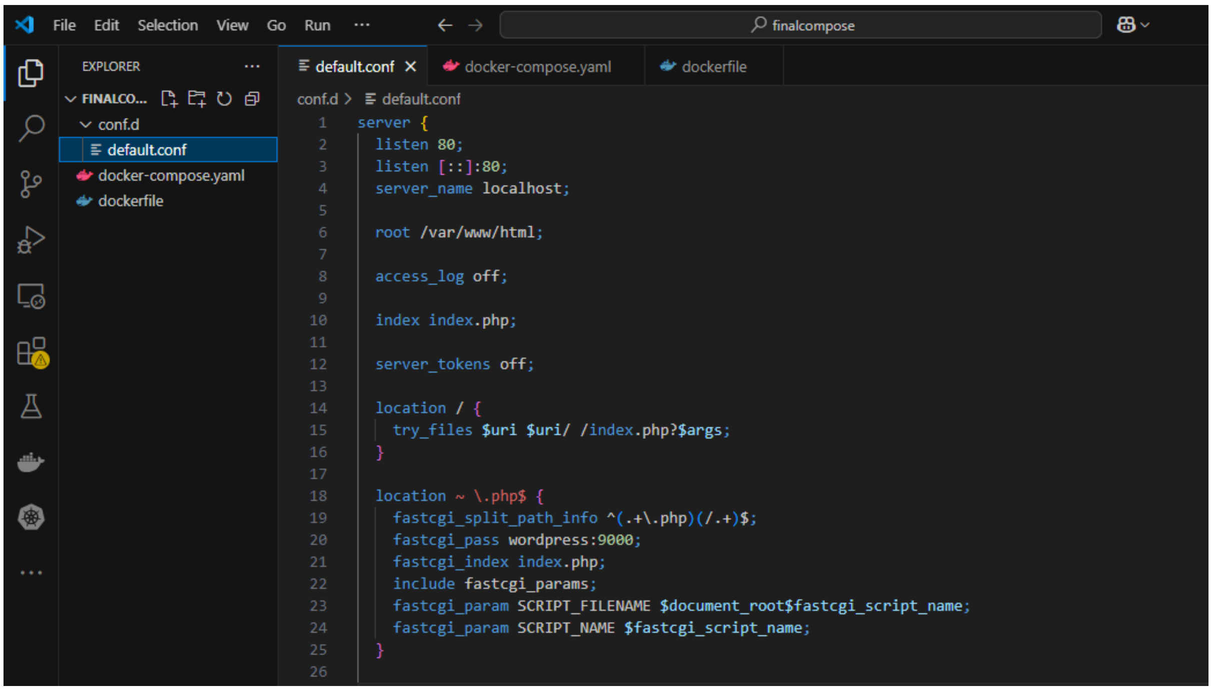
After the container name, give it like this-

```
Deploy:
   Replicas:3
```

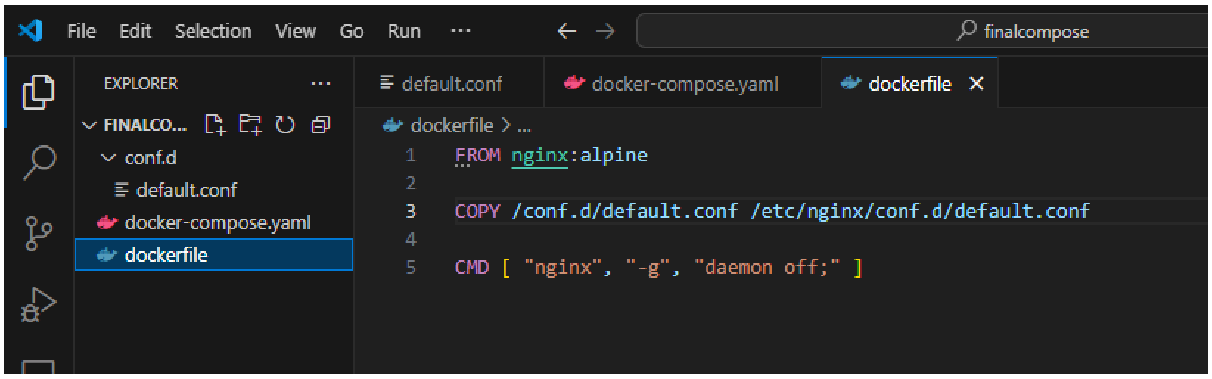
EXAMPLE-

I’m going to create a YAML file to deploy containers of a wordpress stack with nginx and mysql.

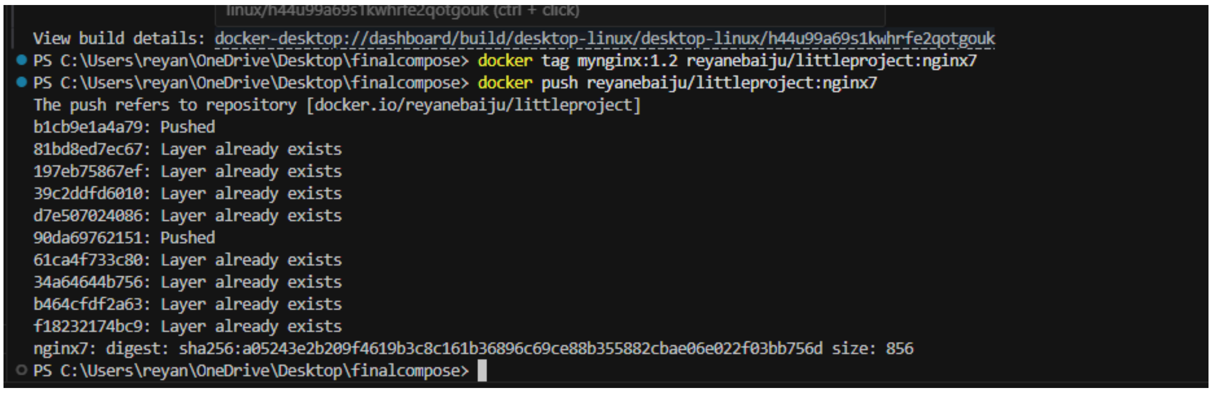
First, we have to create a default.conf file that tells nginx to use fastcgi php processing of the wordpress container to serve the webpage. And create a docker image of nginx.


```
server {
  listen 80;
  listen [::]:80;
  server_name localhost;

  root /var/www/html;

  access_log off;

  index index.php;

  server_tokens off;

  location / {
    try_files $uri $uri/ /index.php?$args;
  }

  location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass wordpress:9000;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param SCRIPT_NAME $fastcgi_script_name;
  }

```

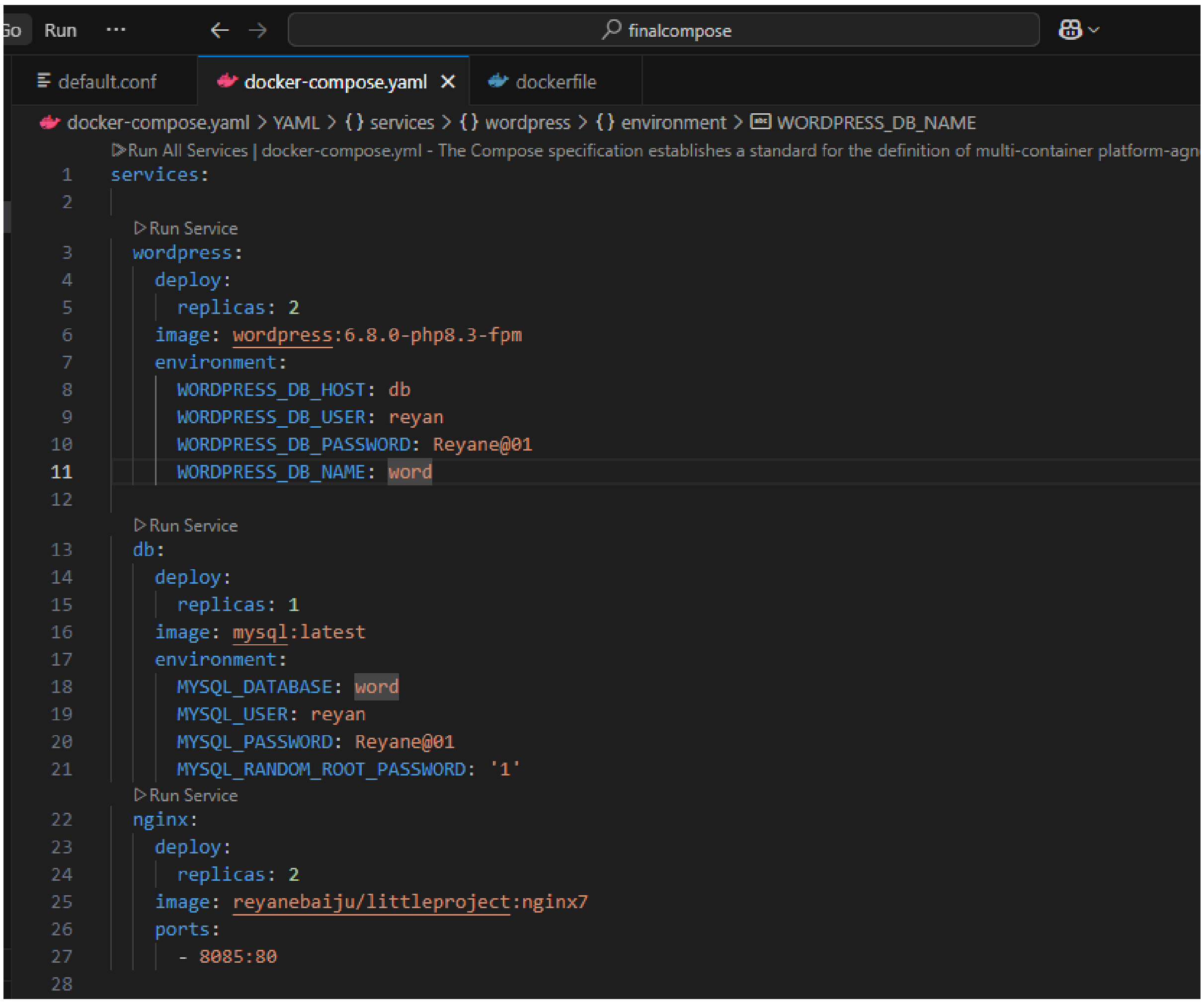
I have created a dockerfile to create a docker image and push it to the docker hub repository.


```
FROM nginx:alpine

COPY /conf.d/default.conf /etc/nginx/conf.d/default.conf

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

Building and pushing-

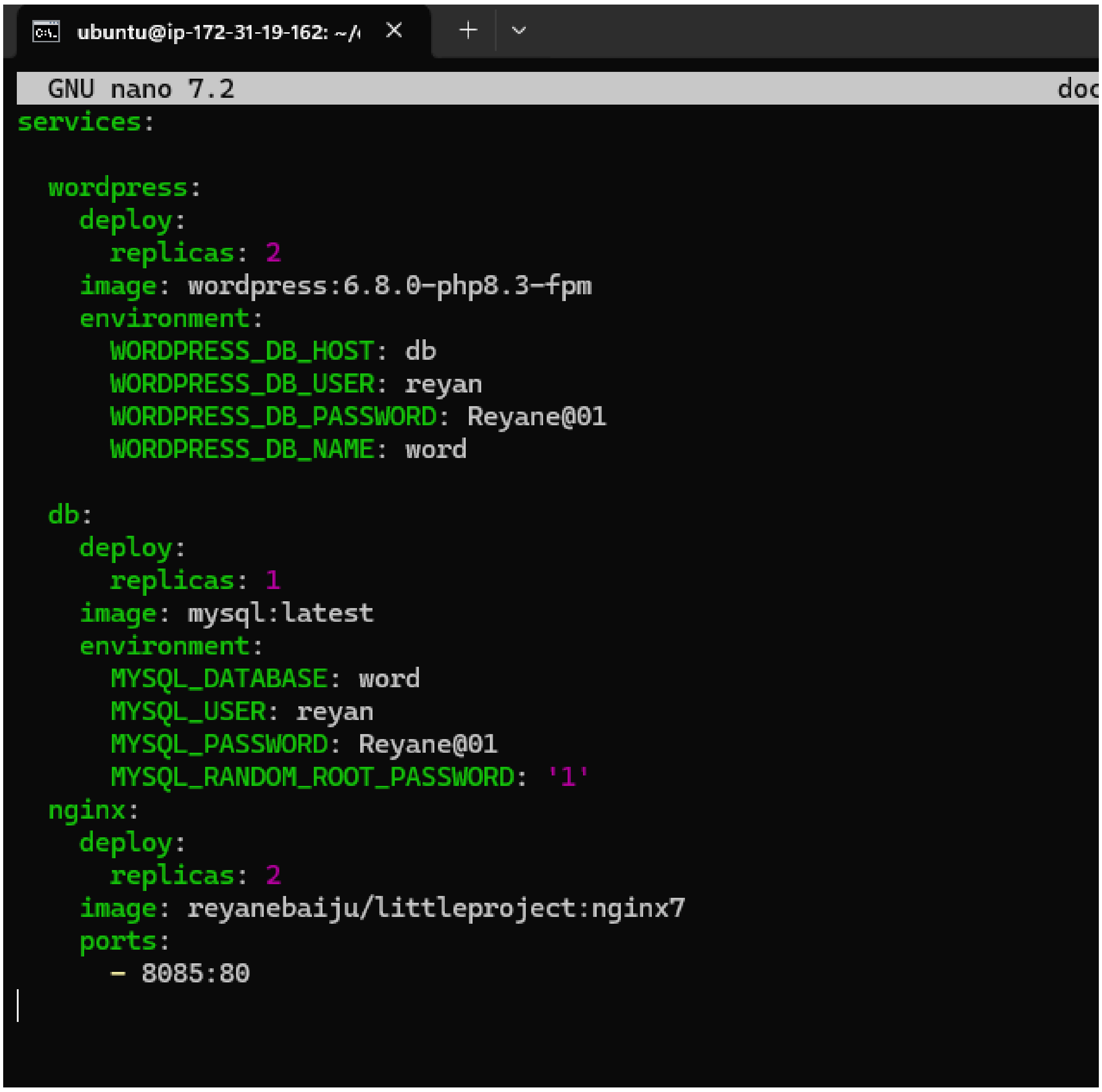

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/h44u99a69s1kwhrfe2qotgouk
PS C:\Users\reyan\OneDrive\Desktop\finalcompose> docker tag mynginx:1.2 reyanebaiju/littleproject:nginx7
PS C:\Users\reyan\OneDrive\Desktop\finalcompose> docker push reyanebaiju/littleproject:nginx7
The push refers to repository [docker.io/reyanebaiju/littleproject]
b1cb9e1a4a79: Pushed
81bd8ed7ec67: Layer already exists
197eb75867ef: Layer already exists
39c2ddfd6010: Layer already exists
d7e507024086: Layer already exists
90da69762151: Pushed
61ca4f733c80: Layer already exists
34a64644b756: Layer already exists
b464cfdf2a63: Layer already exists
f18232174bc9: Layer already exists
nginx7: digest: sha256:a05243e2b209f4619b3c8c161b36896c69ce88b355882cbae06e022f03bb756d size: 856
PS C:\Users\reyan\OneDrive\Desktop\finalcompose>
```

Now, we can create our yaml file that we can deploy in the swarm.


Go Run ···
finalcompose
default.conf
docker-compose.yaml
dockerfile
docker-compose.yaml > YAML > {} services > {} wordpress > {} environment > WORDPRESS_DB_NAME
▷Run All Services | docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agn
▷Run Service

```yaml
services:

  wordpress:
    deploy:
      replicas: 2
    image: wordpress:6.8.0-php8.3-fpm
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: reyan
      WORDPRESS_DB_PASSWORD: Reyane@01
      WORDPRESS_DB_NAME: word

  db:
    deploy:
      replicas: 1
    image: mysql:latest
    environment:
      MYSQL_DATABASE: word
      MYSQL_USER: reyan
      MYSQL_PASSWORD: Reyane@01
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
  nginx:
    deploy:
      replicas: 2
    image: reyanebaiju/littleproject:nginx7
    ports:
      - 8085:80

```

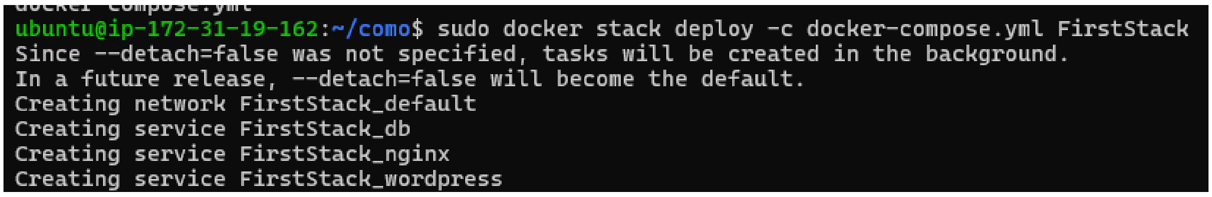
Copying to our manager node-



```
  GNU nano 7.2                                                          doc
services:

  wordpress:
    deploy:
      replicas: 2
    image: wordpress:6.8.0-php8.3-fpm
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: reyan
      WORDPRESS_DB_PASSWORD: Reyane@01
      WORDPRESS_DB_NAME: word

  db:
    deploy:
      replicas: 1
    image: mysql:latest
    environment:
      MYSQL_DATABASE: word
      MYSQL_USER: reyan
      MYSQL_PASSWORD: Reyane@01
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
  nginx:
    deploy:
      replicas: 2
    image: reyanebaiju/littleproject:nginx7
    ports:
      - 8085:80
```

We are going to deploy this-

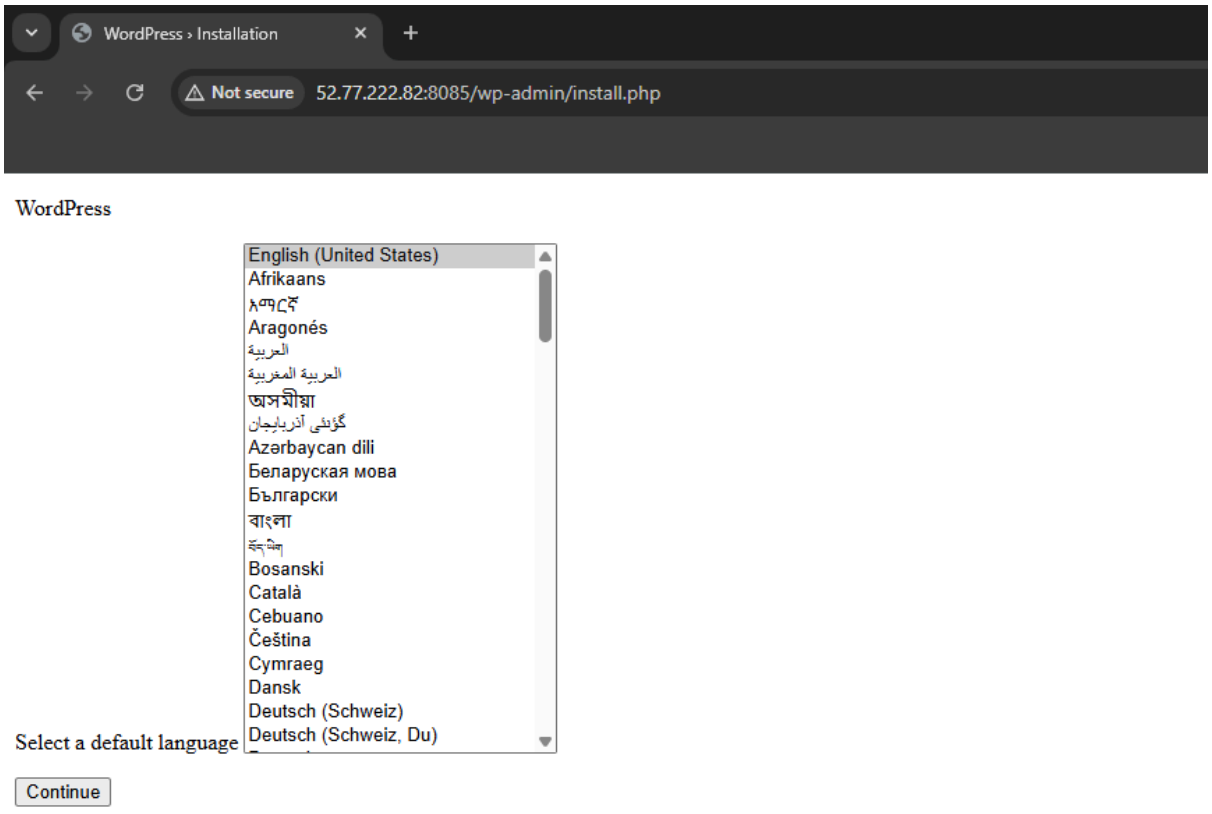
sudo docker stack deploy -c docker-compose.yml FirstStack



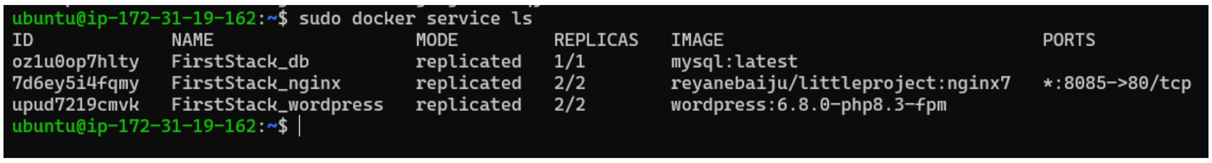
```
ubuntu@ip-172-31-19-162:~/como$ sudo docker stack deploy -c docker-compose.yml FirstStack
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network FirstStack_default
Creating service FirstStack_db
Creating service FirstStack_nginx
Creating service FirstStack_wordpress
```

It was created successfully.

Now, we can see the swarm in action.



We can see that the webpage is not loading properly due to the issue of storage. Nginx is failing to load the webpage properly due to this issue.



```
ubuntu@ip-172-31-19-162:~$ sudo docker service ls
ID             NAME                  MODE         REPLICAS   IMAGE                                 PORTS
oz1u0op7hlty   FirstStack_db         replicated   1/1        mysql:latest
7d6ey5i4fqmy   FirstStack_nginx      replicated   2/2        reyanebaiju/littleproject:nginx7      *:8085->80/tcp
upud7219cmvk   FirstStack_wordpress  replicated   2/2        wordpress:6.8.0-php8.3-fpm
ubuntu@ip-172-31-19-162:~$
```

We can see that the services were replicated correctly.

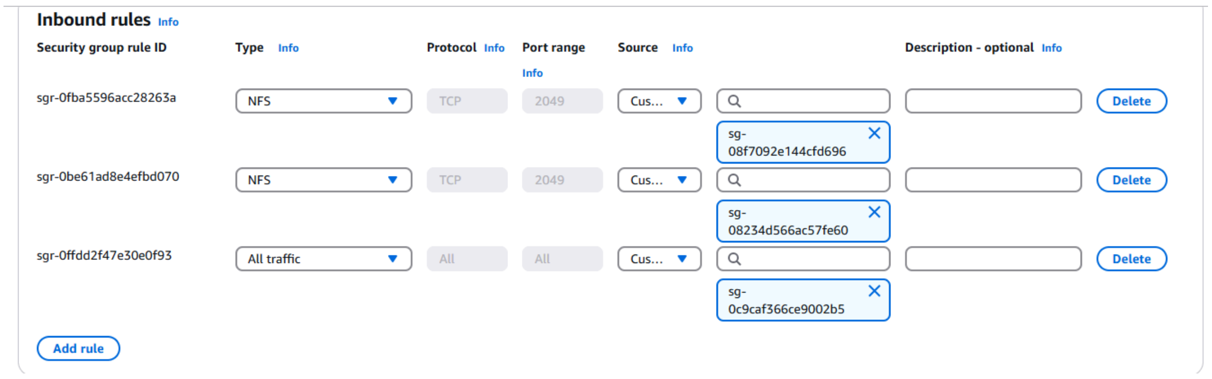
Persistent storage

The biggest problem with docker swarm is the storage issue. If we wanted to use volume or bind mounting, it is situated locally in each node, so we can't achieve consistent storage consistency throughout the cluster.

To solve this issue, we could use Ceph, GlusterFS, EFS, EBS etc.

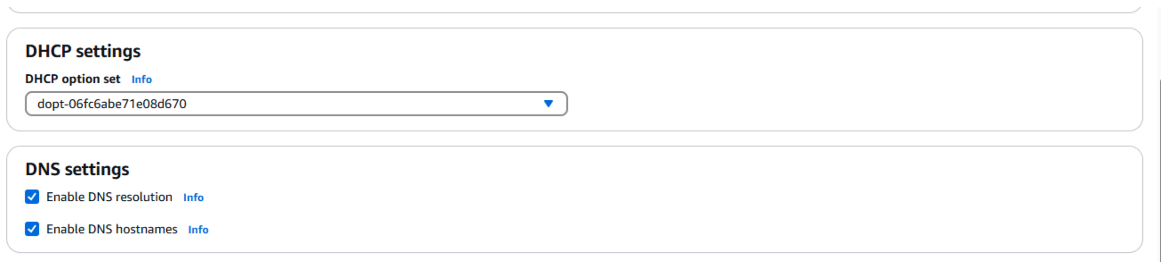
I'm going to use the solution of EFS.

EFS is a cloud based file storage system provided by AWS. To configure this, we need to follow some things. EFS should be deployed in the same subnet as the EC2 nodes, and should be given appropriate rules (NFS) in the security group to the security group of the EC2 instances.


Inbound rules

Security group rule ID	Type	Protocol	Port range	Source		Description - optional
sgr-0fba5596acc28263a	NFS	TCP	2049	Cus...	sg-08f7092e144cfd696	
sgr-0be61ad8e4efbd070	NFS	TCP	2049	Cus...	sg-08234d566ac57fe60	
sgr-0ffdd2f47e30e0f93	All traffic	All	All	Cus...	sg-0c9caf366ce9002b5	

Also we need to set DNS resolution and DHCP for VPC.

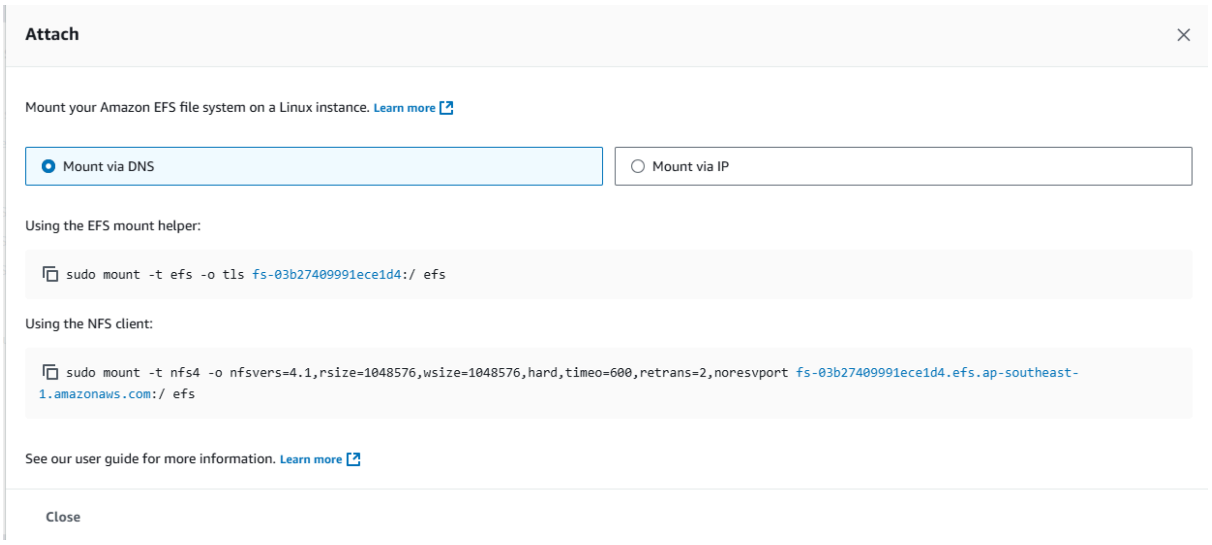

DHCP settings

DHCP option set: dopt-06fc6abe71e08d670

DNS settings

- [x] Enable DNS resolution
- [x] Enable DNS hostnames

We do this during the creation of EFS. After that, we need to use the "mount via DNS" option and mount our file system.


Attach

Mount your Amazon EFS file system on a Linux instance. Learn more

[✓ Mount via DNS] Mount via IP

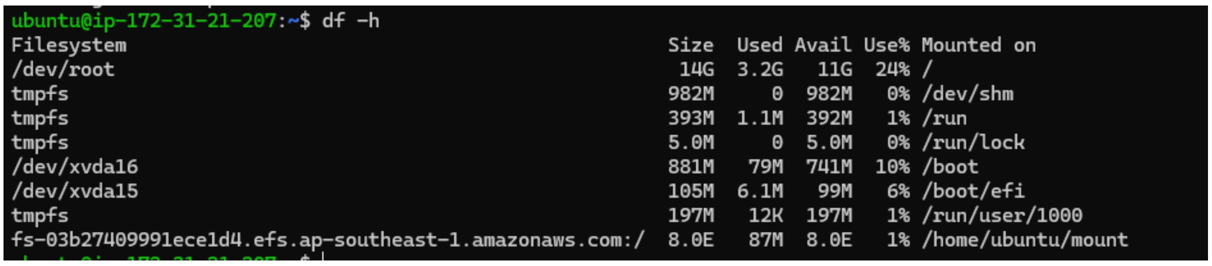
Using the EFS mount helper:

```
sudo mount -t efs -o tls fs-03b27409991ece1d4:/ efs
```

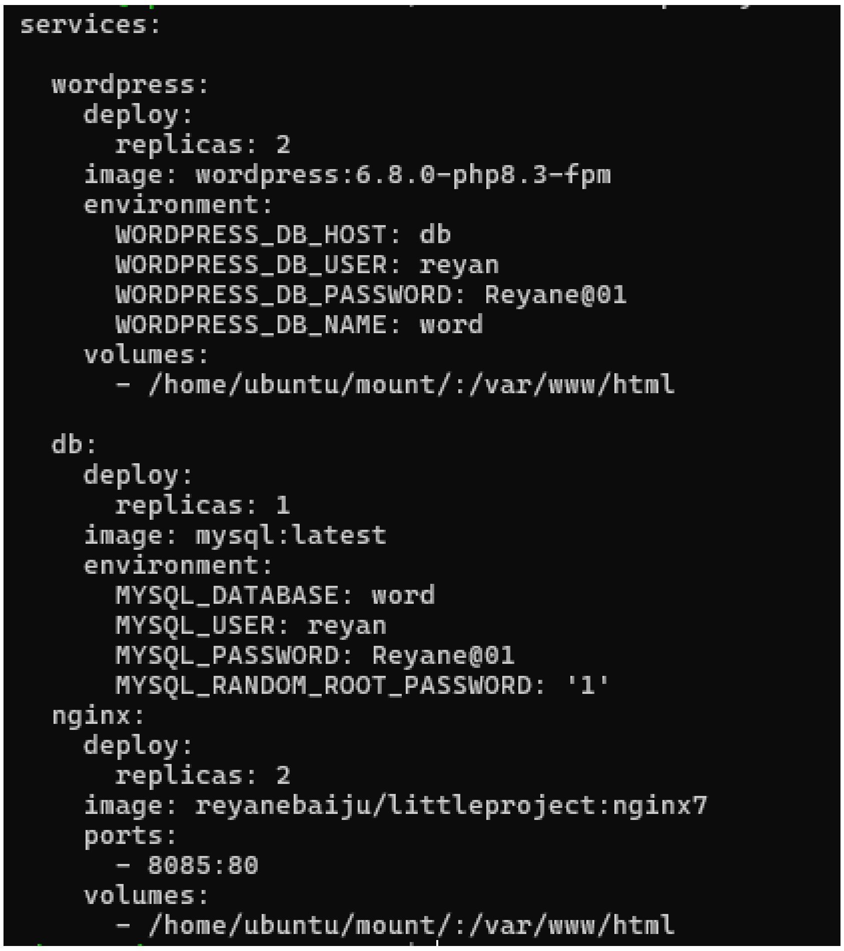
Using the NFS client:

```
sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-03b27409991ece1d4.efs.ap-southeast-
1.amazonaws.com:/ efs
```

See our user guide for more information. Learn more

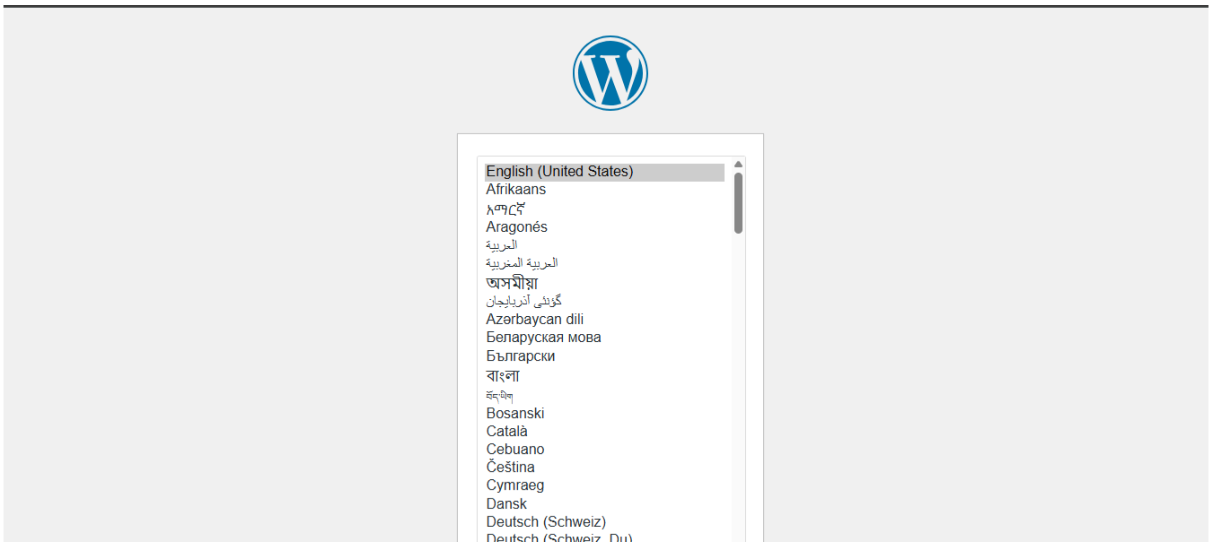

```
ubuntu@ip-172-31-21-207:~$ df -h
Filesystem                                             Size  Used Avail Use% Mounted on
/dev/root                                               14G  3.2G   11G  24% /
tmpfs                                                  982M     0  982M   0% /dev/shm
tmpfs                                                  393M  1.1M  392M   1% /run
tmpfs                                                  5.0M     0  5.0M   0% /run/lock
/dev/xvda16                                            881M   79M  741M  10% /boot
/dev/xvda15                                            105M  6.1M   99M   6% /boot/efi
tmpfs                                                  197M   12K  197M   1% /run/user/1000
fs-03b27409991ece1d4.efs.ap-southeast-1.amazonaws.com:/  8.0E   87M  8.0E   1% /home/ubuntu/mount
```

Now, we can use bind mounting or custom volume with this directory and reference it in our YAML file.



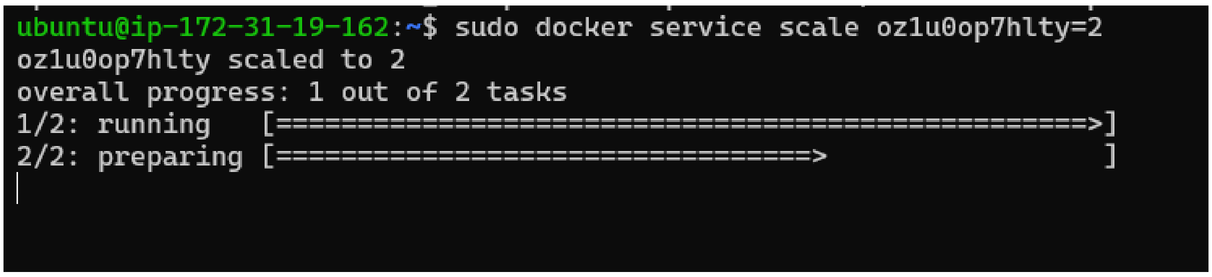
```
services:

  wordpress:
    deploy:
      replicas: 2
    image: wordpress:6.8.0-php8.3-fpm
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: reyan
      WORDPRESS_DB_PASSWORD: Reyane@01
      WORDPRESS_DB_NAME: word
    volumes:
      - /home/ubuntu/mount/:/var/www/html

  db:
    deploy:
      replicas: 1
    image: mysql:latest
    environment:
      MYSQL_DATABASE: word
      MYSQL_USER: reyan
      MYSQL_PASSWORD: Reyane@01
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
  nginx:
    deploy:
      replicas: 2
    image: reyanebaiju/littleproject:nginx7
    ports:
      - 8085:80
    volumes:
      - /home/ubuntu/mount/:/var/www/html
```

When we deploy this, we can see that it works perfectly.



Scaling deployed container services-

To manually scale up containers, we can use the command
sudo docker service scale <service-id>=<no-of-replicas>



```
ubuntu@ip-172-31-19-162:~$ sudo docker service scale oz1u0op7hlty=2
oz1u0op7hlty scaled to 2
overall progress: 1 out of 2 tasks
1/2: running   [==================================================>]
2/2: preparing [=================================>                 ]
```

ROLLING UPDATES

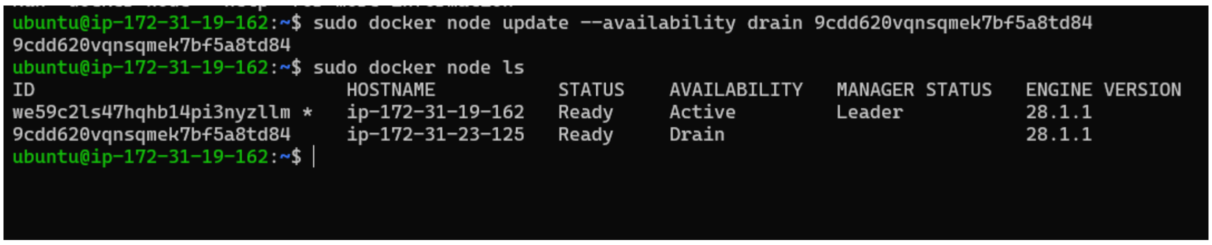
Rolling updates means changing/updating an image in a container while it is being run.

docker service update –image <image-to-update-name:tag> <container-name>

DRAIN STATUS

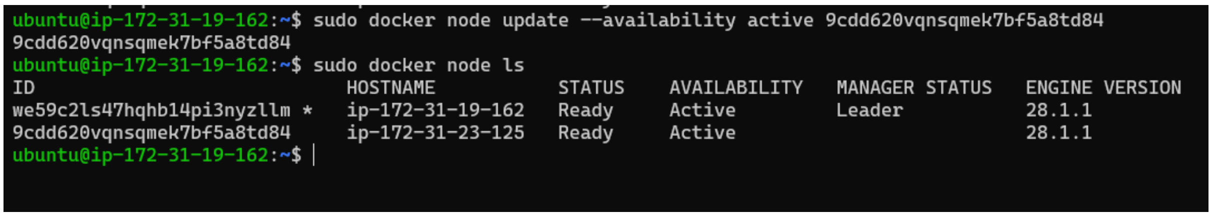
Drain status prevents a node from receiving new tasks.

docker node update –availability drain <node-name>



```
ubuntu@ip-172-31-19-162:~$ sudo docker node update --availability drain 9cdd620vqnsqmek7bf5a8td84
9cdd620vqnsqmek7bf5a8td84
ubuntu@ip-172-31-19-162:~$ sudo docker node ls
ID                            HOSTNAME           STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
we59c2ls47hqhb14pi3nyzllm *   ip-172-31-19-162   Ready     Active         Leader           28.1.1
9cdd620vqnsqmek7bf5a8td84     ip-172-31-23-125   Ready     Drain                           28.1.1
ubuntu@ip-172-31-19-162:~$
```

Use –availability active to return it to the previous state.



```
ubuntu@ip-172-31-19-162:~$ sudo docker node update --availability active 9cdd620vqnsqmek7bf5a8td84
9cdd620vqnsqmek7bf5a8td84
ubuntu@ip-172-31-19-162:~$ sudo docker node ls
ID                            HOSTNAME           STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
we59c2ls47hqhb14pi3nyzllm *   ip-172-31-19-162   Ready     Active         Leader           28.1.1
9cdd620vqnsqmek7bf5a8td84     ip-172-31-23-125   Ready     Active                          28.1.1
ubuntu@ip-172-31-19-162:~$ |
```

To create a network use the docker network create command.

After creating, use the –network <network-name> flag to use it when deploying a service.

TYPES OF SWARM SERVICES

They are two- replicated and global.

In replicated mode, we can define how many containers should be replicated.
But in global mode, the container is deployed in every possible node.
(EX use case- if we want to install monitoring agents, or antivirus).

We can create a global mode deployment by using the –mode flag.
Ex. docker service create –name <service-name> –mode global <image-name>