Docker

1. What is Docker?

1. **Definition**: Docker is an open-source platform for developing, shipping, and running applications using containerization. It ensures that applications work seamlessly across various environments, from development to production.
2. **Importance in DevOps**: Docker enhances efficiency by allowing developers to build once and run anywhere. Containers help in:
    - Reducing inconsistencies between environments.
    - Faster deployment cycles.
3. **Virtual Machines vs Containers**:
    - **VMs**: Use a hypervisor, each VM includes a full OS, which makes them heavier.
    - **Containers**: Share the host OS kernel, making them lightweight and faster.

---

2. Docker Architecture

Docker operates on a client-server model. The key components are:

4. **Docker Engine**: The runtime that builds, runs, and manages containers.
5. **Docker CLI**: A command-line interface for interacting with Docker.
6. **Docker Hub**: A public registry where Docker images are stored and shared.
7. **Docker Images**: Read-only templates for creating containers.
8. **Containers**: Runtime instances of Docker images.
9. **How Docker Works**:
    - The Docker CLI sends commands to the Docker Engine.
    - The engine pulls the image from the Docker Hub (if not available locally).
    - The container is created and managed.

Benefits:

- **Portability**: Containers can run on any system that supports Docker, whether it's a developer's laptop or a production cloud environment.
- **Consistency**: Because Docker containers package everything an app needs to run, they ensure consistency between different environments (development, staging, production).
- **Isolation**: Each container runs in isolation, so applications inside them don't interfere with each other.

- **Efficiency**: Containers are lightweight compared to virtual machines (VMs) because they share the host system's OS kernel, making them faster to start and consume fewer resources.

In short, Docker simplifies the development and deployment process by making applications more portable and easier to manage.

---

3. Installing Docker

Steps:

1. Update your package index: `sudo dnf update` (for RHEL 9).
2. Install prerequisites: `dnf -y install dnf-plugins-core`
3. Add Docker's GPG key and repository:
   `dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo`
4. Install Docker: `sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin`
5. Verify: `docker --version`.

`# docker info`

The `docker info` command provides detailed information about the Docker installation on your system. This includes information such as:

- Docker version
- Operating system details
- Number of containers (running, stopped, paused)
- Number of images
- Storage driver in use
- Network settings
- Security options

`# docker images`

The `docker images` command is used to list the images stored on your local Docker host. This command provides an overview of all available images, including their tags and sizes.

Basic Usage

To list all Docker images, simply run:

```
docker images
```

Example Output

The output will typically look like this:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
my-image            latest              abc123456789        2 days ago
150MB
another-image       v1.0                def987654321        1 week ago
250MB
```

Les’s pull centos images

```
[root@docker ~]# docker pull centos
Using default tag: latest
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest:
sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b17
7
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
[root@docker ~]#
[root@docker ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
centos       latest    5d0da3dc9764   3 years ago   231MB
[root@docker ~]#
```

docker.io/library/centos:latest
1 2 3 4

1. registry server url
2. account (library public accout)
3. image name but actually this is a dire name nd image stored in this dire in layer formate
4. tag (version)

Now let's create a container using centos images.

```
[root@docker ~]# docker run -it --name con1 centos
[root@1c9b2f0a206b /]#
[root@1c9b2f0a206b /]# ls
bin  etc   lib    lost+found  mnt  proc  run   srv  tmp  var
dev  home  lib64  media       opt  root  sbin  sys  usr
[root@1c9b2f0a206b /]#
```

```
[root@docker ~]# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
1c9b2f0a206b   centos    "/bin/bash"   3 minutes ago   Up 3 minutes             con1
[root@docker ~]#
```

The `docker ps` command is used to list running containers on your Docker host. It provides information such as the container ID, image used, command executed, creation time, status, ports, and names.

```
[root@docker ~]# docker stop con1
con1
[root@docker ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS                      PORTS
 NAMES
1c9b2f0a206b   centos    "/bin/bash"   4 minutes ago   Exited (127) 6 seconds ago
 con1
[root@docker ~]#
```

The `docker ps -a` command lists all containers on your Docker host, regardless of their state (running, exited, etc.). This is useful for getting a complete overview of all containers you have created.

```
[root@docker ~]# docker rm -f con1
con1
[root@docker ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@docker ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@docker ~]#
```

Now create a container in detach mode.

```
[root@docker ~]# docker run -itd --name con2 centos
```

```
[root@docker ~]# docker run -itd --name con2 centos
f8ca238ac808627b439d56569c8f2dbd15dd292d8f24e9d30d8d210ac56c5b06
[root@docker ~]#
[root@docker ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
f8ca238ac808   centos    "/bin/bash"   7 seconds ago   Up 7 seconds             con2
[root@docker ~]#
```

Now login to con2 on existing shell.

```
[root@docker ~]# docker attach con2
```

```
[root@docker ~]# docker attach con2
[root@f8ca238ac808 /]#
[root@f8ca238ac808 /]# ls
bin  etc   lib    lost+found  mnt  proc  run   srv  tmp  var
dev  home  lib64  media       opt  root  sbin  sys  usr
[root@f8ca238ac808 /]#
[root@f8ca238ac808 /]# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.4  12052  3200 pts/0    Ss   17:23   0:00 /bin/bash
root          16  0.0  0.5  47588  4096 pts/0    R+   17:26   0:00 ps -aux
[root@f8ca238ac808 /]#
```

Let's take a safe exit from the container.

Ctrl + p + q

```
[root@f8ca238ac808 /]# read escape sequence
[root@docker ~]#
[root@docker ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
f8ca238ac808   centos    "/bin/bash"   4 minutes ago   Up 4 minutes             con2
[root@docker ~]#
```

Now we are going to login con2 with a new shell

```
[root@docker ~]# docker exec -it con2 bash
[root@f8ca238ac808 /]#
[root@f8ca238ac808 /]# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.4  12052  3200 pts/0    Ss+  17:23   0:00 /bin/bash
root          17  0.3  0.4  12052  3200 pts/1    Ss   17:30   0:00 bash
root          31  0.0  0.5  47588  4096 pts/1    R+   17:31   0:00 ps -aux
[root@f8ca238ac808 /]#
[root@f8ca238ac808 /]# exit
exit
[root@docker ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
f8ca238ac808   centos    "/bin/bash"   7 minutes ago   Up 7 minutes             con2
[root@docker ~]#
```

Now we are going to copy content from host machine to container and the crosscheck it without login to container.

Let's start.

```
[root@docker ~]# ls
docker.sh  test.txt
[root@docker ~]#
[root@docker ~]# docker cp test.txt con2:/root
Successfully copied 1.54kB to con2:/root
[root@docker ~]#
[root@docker ~]# docker exec -it con2  ls /root/
anaconda-ks.cfg  anaconda-post.log  original-ks.cfg  test.txt
[root@docker ~]#
```

As we can see we have a test.txt file on docker host and copied to con2 in /root.

Now let's deploy web server in con2.

For that 1st we need to login to con2 with a new shell

```
[root@docker ~]# docker exec -it con2 bash
[root@f8ca238ac808 /]#
[root@f8ca238ac808 /]# cd /etc/yum.repos.d/
[root@f8ca238ac808 yum.repos.d]#
[root@f8ca238ac808 yum.repos.d]# ls
CentOS-Linux-AppStream.repo          CentOS-Linux-FastTrack.repo
CentOS-Linux-BaseOS.repo             CentOS-Linux-HighAvailability.repo
CentOS-Linux-ContinuousRelease.repo  CentOS-Linux-Media.repo
CentOS-Linux-Debuginfo.repo          CentOS-Linux-Plus.repo
CentOS-Linux-Devel.repo              CentOS-Linux-PowerTools.repo
CentOS-Linux-Extras.repo             CentOS-Linux-Sources.repo
[root@f8ca238ac808 yum.repos.d]#
[root@f8ca238ac808 yum.repos.d]# rm -f  *
[root@f8ca238ac808 yum.repos.d]#
[root@f8ca238ac808 yum.repos.d]#
```

Remove all existing repos.

Now let's create a new repo to install required packages.

```
[root@f8ca238ac808 yum.repos.d]#
[root@f8ca238ac808 yum.repos.d]#
[root@f8ca238ac808 yum.repos.d]# vi local.repo
```

[App]
name = this is appstream repo
baseurl = https://repo.almalinux.org/almalinux/8/AppStream/x86_64/os
enabled = 1
gpgcheck = 0

[Base]
name = this is appstream repo
baseurl = https://repo.almalinux.org/almalinux/8/BaseOS/x86_64/os
enabled = 1
gpgcheck = 0

Install httpd & vim package in con2

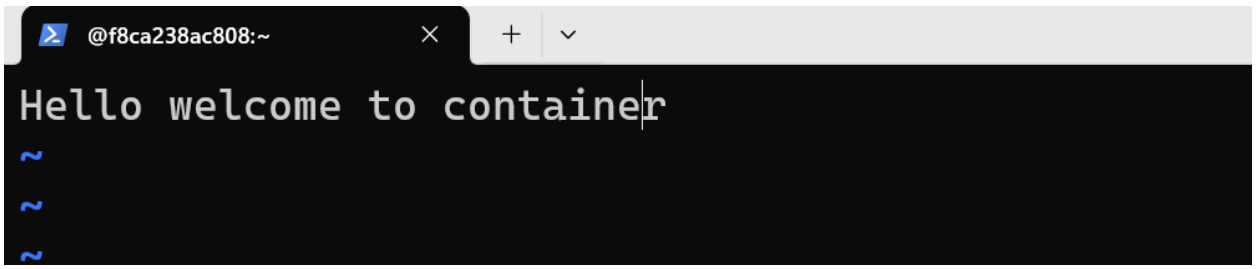
```
[root@f8ca238ac808 ~]# yum install -y httpd vim
Failed to set locale, defaulting to C.UTF-8
this is appstream repo                                  13 MB/s |  13 MB     00:00
this is appstream repo                                  19 MB/s | 6.2 MB     00:00
Last metadata expiration check: 0:00:01 ago on Thu Sep 19 17:43:59 2024.
Dependencies resolved.
================================================================================
 Package                Arch     Version                                Repo    Size
================================================================================
Installing:
 httpd                  x86_64   2.4.37-65.module_el8.10.0+3874+c2064c23.2    App    1.4 M
 vim-enhanced           x86_64   2:8.0.1763-19.el8_6.4                       App    1.4 M
Installing dependencies:
```

As we know that in centos base image no systemd so we need start httpd.service manually.

```
[root@f8ca238ac808 ~]# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.3  12052  2560 pts/0    Ss+  17:23   0:00 /bin/bash
root          38  0.0  0.3  12156  2816 pts/1    Ss   17:36   0:00 bash
root         116  0.0  0.5  47588  4096 pts/1    R+   17:45   0:00 ps -aux
[root@f8ca238ac808 ~]#
[root@f8ca238ac808 ~]# /usr/sbin/httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.17.0.2. Set the 'ServerName' directive globally to suppress this message
[root@f8ca238ac808 ~]#
[root@f8ca238ac808 ~]# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.3  12052  2560 pts/0    Ss+  17:23   0:00 /bin/bash
root          38  0.0  0.3  12156  2944 pts/1    Ss   17:36   0:00 bash
root         118  0.0  0.9 258064  7536 ?        Ss   17:45   0:00 /usr/sbin/httpd
apache       119  0.0  1.0 260644  8452 ?        S    17:45   0:00 /usr/sbin/httpd
apache       120  0.0  1.5 1318436 11792 ?       Sl   17:45   0:00 /usr/sbin/httpd
apache       121  0.0  1.5 1318436 11792 ?       Sl   17:45   0:00 /usr/sbin/httpd
apache       122  0.0  1.7 1449564 14036 ?       Sl   17:45   0:00 /usr/sbin/httpd
root         334  0.0  0.5  47588  3968 pts/1    R+   17:45   0:00 ps -aux
[root@f8ca238ac808 ~]#
```

Now create a index.html to documentroot /var/www/html.

```
[root@f8ca238ac808 ~]# vim  /var/www/html/index.html
```



Exit from container and access web-server.

```
[root@f8ca238ac808 ~]#
[root@f8ca238ac808 ~]# exit
exit
[root@docker ~]#
[root@docker ~]# docker inspect con2
[
    {
        "Id": "f8ca238ac808627b439d56569c8f2dbd15dd292d8f24e9d30d8d210ac56c5b06",
        "Created": "2024-09-19T17:23:44.698079953Z",
        "Path": "/bin/bash",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
```

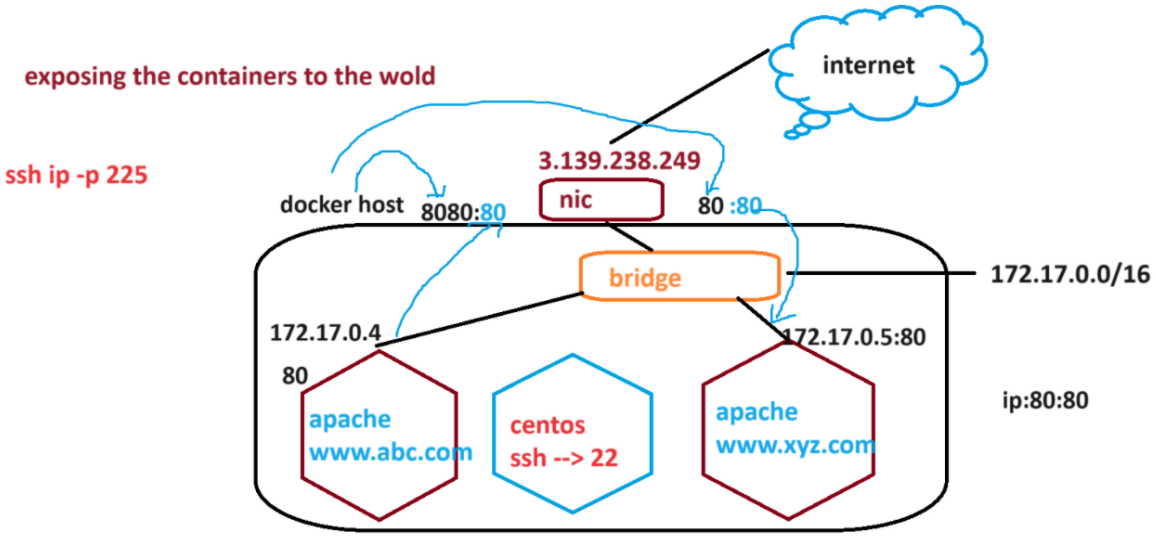
```
0f0a16",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DNSNames": null
                }
            }
        }
    }
]
[root@docker ~]#
```

Container ip is 172.17.0.2

```
[root@docker ~]# curl  http://172.17.0.2
Hello welcome to container
[root@docker ~]#
[root@docker ~]#
```

Ok at this time we can access this web-server only from the docker host. To access from browser we need to expose the 80 port.

But we can not expose a port on an existing container. We can do this task only while creating a container.



Ok now we will expose 80 port for another container. Before that we understand what is Dockerfile and how we can create an image using Dockerfile.

A Dockerfile is a text file that contains a set of instructions for building a Docker image. It specifies how the image should be constructed, including what base image to use, what files to include, and what commands to run. This allows you to automate the process of image creation, ensuring consistency and repeatability.

Lest create Dockerfile

```
[root@docker ~]# vim Dockerfile
[root@docker ~]#
[root@docker ~]# vim local.repo
[root@docker ~]#
[root@docker ~]# ls
Dockerfile  docker.sh  local.repo  test.txt
[root@docker ~]#
```

FROM centos:latest

RUN rm -f /etc/yum.repos.d/*

WORKDIR /etc/yum.repos.d

```
COPY ./local.repo /etc/yum.repos.d/

RUN yum install -y httpd zip wget

WORKDIR /var/www/html
RUN rm -rf ./*

RUN wget
https://www.free-css.com/assets/files/free-css-templates/download/page290/wave-cafe.zip

RUN unzip wave-cafe.zip

RUN rm -f wave-cafe.zip &&\
   cp -rf 2121_wave_cafe/* . &&\
      rm -rf 2121_wave_cafe
EXPOSE 80

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

Lets Build image

```
[root@docker ~]# docker image build . -t wave-web
```

```
[root@docker ~]# docker image build . -t wave-web
[+] Building 17.6s (15/15) FINISHED                                    docker:default
 => [internal] load build definition from Dockerfile                             0.0s
 => => transferring dockerfile: 555B                                             0.0s
 => [internal] load metadata for docker.io/library/centos:latest                 0.0s
 => [internal] load .dockerignore                                                0.0s
 => => transferring context: 2B                                                  0.0s
 => [ 1/10] FROM docker.io/library/centos:latest                                 0.0s
 => [internal] load build context                                                0.1s
 => => transferring context: 298B                                                0.0s
 => [ 2/10] RUN rm -f /etc/yum.repos.d/*                                         0.2s
 => [ 3/10] WORKDIR /etc/yum.repos.d                                             0.0s
 => [ 4/10] COPY ./local.repo /etc/yum.repos.d/                                  0.0s
 => [ 5/10] RUN yum install -y httpd zip wget                                   12.8s
 => [ 6/10] WORKDIR /var/www/html                                                0.0s
 => [ 7/10] RUN rm -rf ./*                                                       0.2s
 => [ 8/10] RUN wget https://www.free-css.com/assets/files/free-css-templates/download  2.8s
 => [ 9/10] RUN unzip wave-cafe.zip                                              0.3s
 => [10/10] RUN rm -f wave-cafe.zip &&    cp -rf 2121_wave_cafe/* . &&        rm -rf 2  0.2s
 => exporting to image                                                           0.8s
 => => exporting layers                                                          0.7s
 => => writing image sha256:a27640f5ee2a1b33643f9495a811ccc1b5fc705a684a8700a877898d60  0.0s
 => => naming to docker.io/library/wave-web                                      0.0s
[root@docker ~]#
```

```
[root@docker ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
wave-web     latest    a27640f5ee2a   About a minute ago   333MB
centos       latest    5d0da3dc9764   3 years ago          231MB
[root@docker ~]#
```

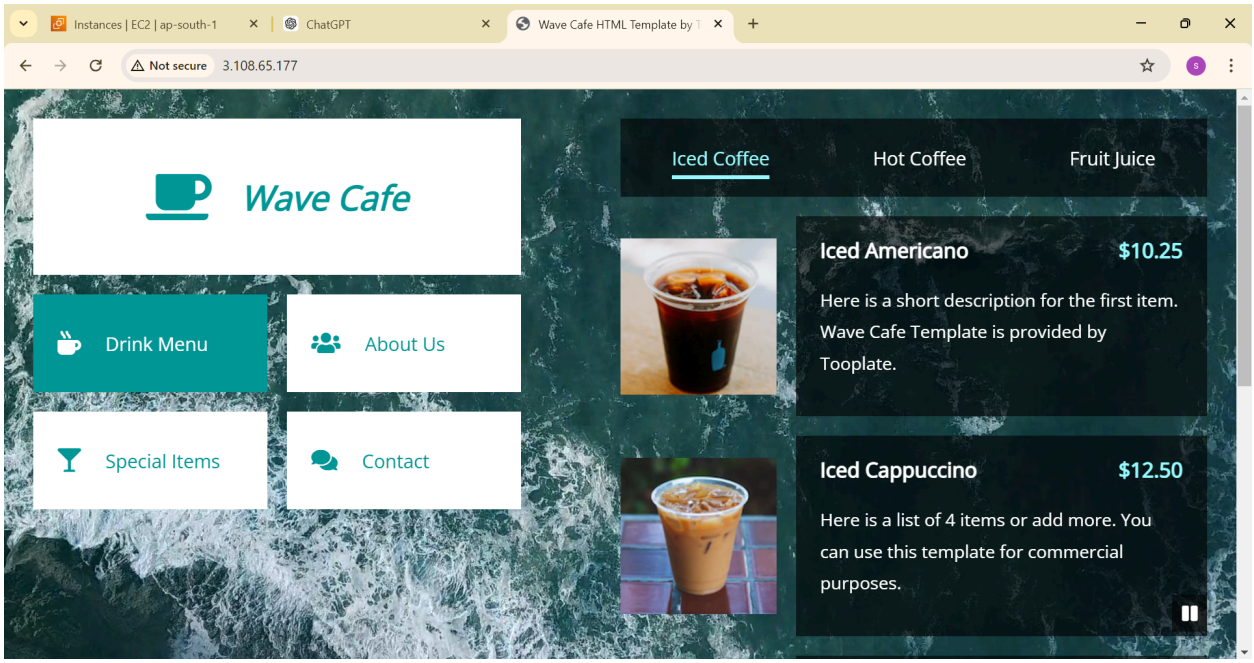
Ok now create a container using wave-web image with exposing host machine 80 port with container.

```
[root@docker ~]# docker run -itd  --name web -p 80:80 wave-web
```

```
[root@docker ~]# docker run -itd  --name web -p 80:80 wave-web
00504bbf8ac8648a6cfd3bed8b5e9063ae389c8e21758a1161d63d06a1d96166
[root@docker ~]#
[root@docker ~]# docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED             STATUS             PORTS
                          NAMES
00504bbf8ac8   wave-web   "/usr/sbin/httpd -D …"   5 seconds ago       Up 4 seconds       0.0.0.0:
80->80/tcp, :::80->80/tcp   web
f8ca238ac808   centos     "/bin/bash"              About an hour ago   Up About an hour
                          con2
[root@docker ~]#
```

Now we need to open 80/tcp port in security group and then just type.

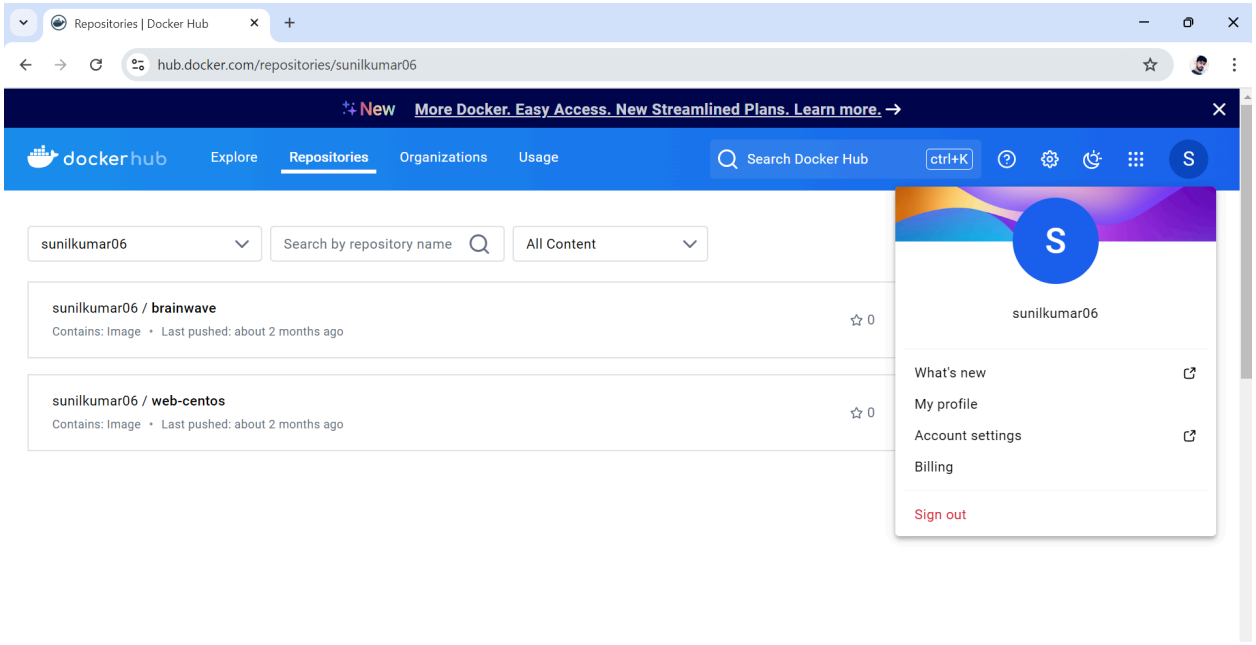
http://instance-pub-ip



Yeah.. Its done.. ;>)

Ok now push this image to docker hub repository.

For that 1st we need to take console login of docker hub.



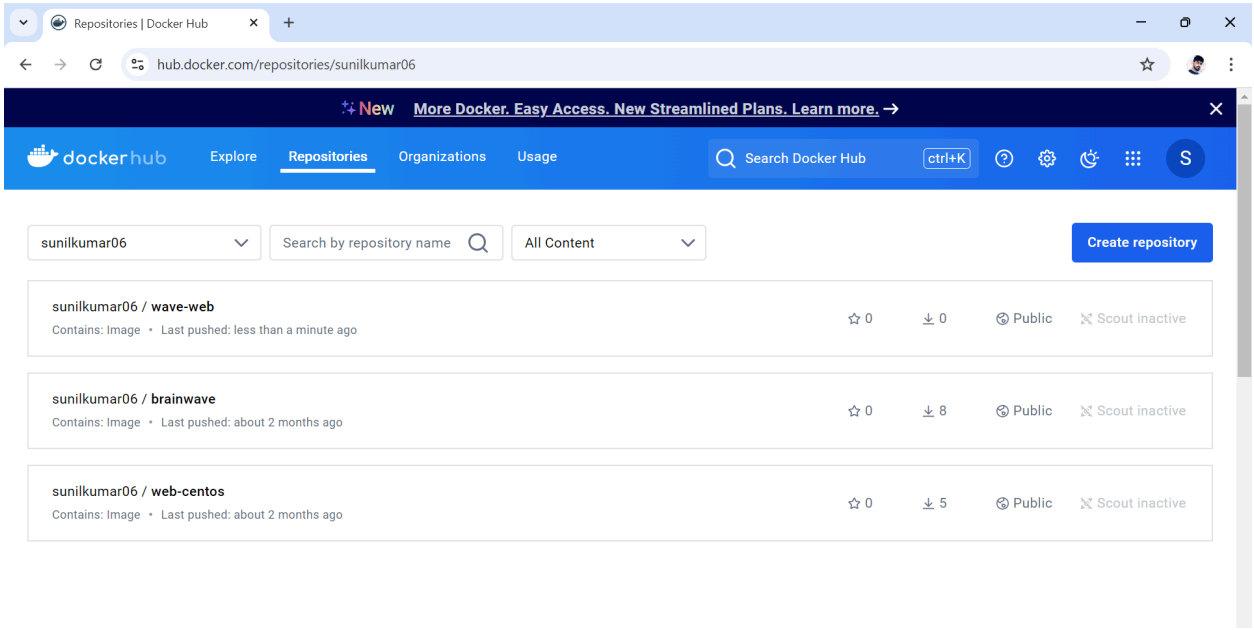

```
[root@docker ~]# docker login -u sunilkumar06
```

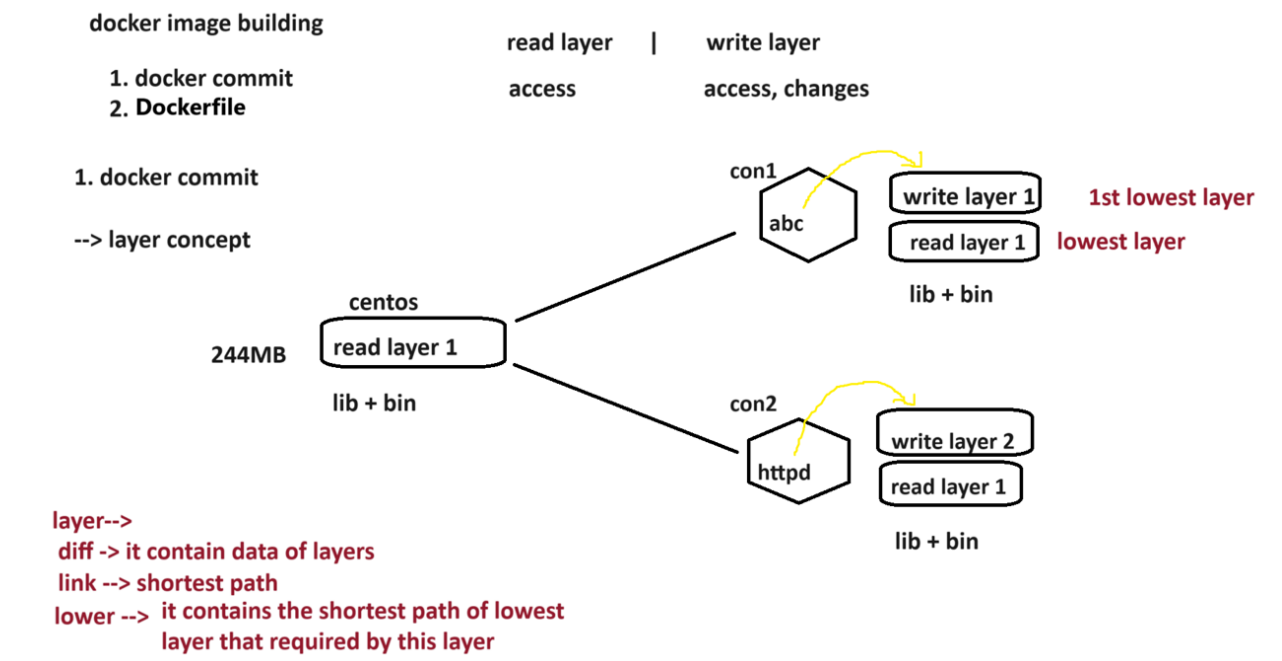
```
[root@docker ~]# docker login -u sunilkumar06
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
[root@docker ~]#
```

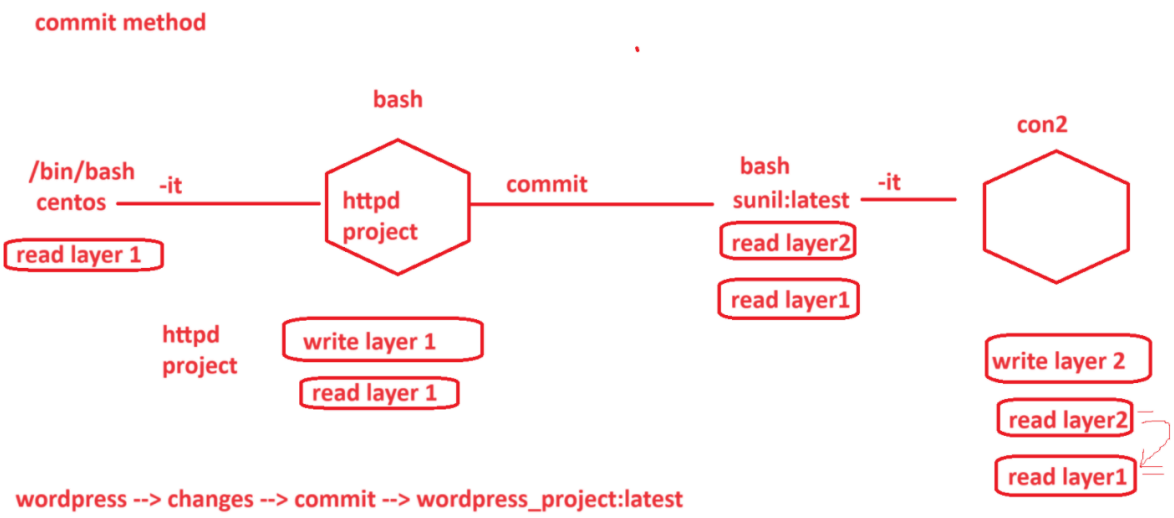
Before push images we required to define tag this images.

```
[root@docker ~]# docker tag wave-web:latest sunilkumar06/wave-web:latest
[root@docker ~]# docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
wave-web                latest    a27640f5ee2a   20 minutes ago   333MB
sunilkumar06/wave-web   latest    a27640f5ee2a   20 minutes ago   333MB
centos                  latest    5d0da3dc9764   3 years ago      231MB
[root@docker ~]#
```

```
[root@docker ~]# docker push sunilkumar06/wave-web:latest
The push refers to repository [docker.io/sunilkumar06/wave-web]
892bf6f7e476: Pushed
93c0d8d83835: Pushed
b5612f73cbe4: Pushed
5f70bf18a086: Pushed
ec3742cd2878: Pushed
4100e7462803: Pushed
a0a31b019674: Pushed
74ddd0ec08fa: Mounted from library/centos
latest: digest: sha256:e5881459afba367df3b24dba1768bdbcb0bcb49841d18f29e8095727cc25f577 size: 2409
[root@docker ~]#
```







Docker layered storage mechanism

When you use **Docker commit** to create a new image, Docker internally uses a **layered storage mechanism**:

1. **Base Layers (CentOS):**
   - The CentOS image's layers become part of the new image's layer stack.
   - Docker stores a **hash reference** for these layers, which uniquely identifies them. If you delete the CentOS image, Docker only removes its **metadata** (e.g., tag information), but not the layers themselves, as long as another image (like your new one) still references them.
2. **Layer Reuse Mechanism:**
   - When you remove the CentOS image (`docker rmi centos`), Docker keeps the underlying layers intact because your new image depends on them.
   - This ensures your new image remains functional and self-contained.

NOTE:- Your new image works independently because Docker retains the required layers even after deleting the CentOS image.

If the CentOS image is 233 MB in size and the new image I created from the container is 333 MB, is it taking up an additional 333 MB of space?

No, the **entire 333 MB does not take up new space**. Docker's layered architecture ensures **disk space optimization** by reusing existing layers.

Breakdown:

1. **CentOS Base Image (233 MB):**

    - These are the original image layers, which are part of your new image's layer stack. Docker reuses these layers already stored on disk and does not duplicate them.
2. **Writable Layer Changes (100 MB):**
    - Any modifications you make in the container (e.g., adding files, modifying configurations) are stored in the writable layer.
    - After running `docker commit`, this writable layer is converted into a new read-only layer and becomes part of the new image.
3. **New Image Size (333 MB):**
    - The total size reported by the new image is **233 MB (CentOS layers) + 100 MB (your changes)**.
    - However, on disk, Docker only uses an additional **100 MB** for the new layer. The base image layers (233 MB) are reused.