Graded Assignment on Dockerizing a Plain HTML Page With Nginx

Course 10: Containerization **| DevOps B4**

Table of Contents

Introduction to Containerization

Containerization is a technology that allows multiple isolated environments, called containers, to run on a single host operating system, sharing the same kernel and resources. Containers provide a lightweight and portable way to deploy applications, ensuring consistency and reliability across different environments.

KEY CONCEPTS:

Containers: Isolated environments run on top of a host operating system, providing a self-contained space for an application to run.

Container Runtime: The software that manages and runs containers, such as Docker, etc.

Host Operating System: The underlying operating system that runs the container runtime and provides resources to the containers.

How Containerization Works

Container Creation

A container is created from a Docker image, which includes the application code, libraries, and dependencies.

Resource Isolation

The container runtime isolates the container from the host operating system and other containers, using kernel namespaces and groups.

Process Isolation

The container runtime runs the application process inside the container, providing an isolated environment for the application to run.

Resource Allocation

The container runtime allocates resources, such as CPU, memory, and network, to the container based on the host operating system's resources.

Benefits of Containerization

Lightweight

Containers are much lighter than traditional virtual machines, making them faster to spin up and down.

Portable

Containers are portable across different environments, ensuring consistency and reliability.

Efficient

Containers use fewer resources than traditional virtual machines, making them more efficient.

Isolated

Containers provide a high level of isolation, ensuring that applications running in different containers do not interfere with each other.

What is Virtualization?

Virtualization is a technology that creates a virtual version of a physical resource, such as a server, storage device, or network. It allows multiple virtual machines (VMs) to run on a single physical machine, each with its own operating system, resources, and applications.

KEY CONCEPTS:

- **Virtual Machines (VMs):** Self-contained operating system instances that run on top of a physical machine.
- **Hypervisor:** The software that creates and manages VMs, such as VMware, VirtualBox, or Hyper-V.
- **Physical Machine:** The underlying hardware that runs the hypervisor and VMs.

How Virtualization Works

- **Hypervisor Installation -** The hypervisor is installed on the physical machine.
- **VM Creation -** A VM is created, including its own operating system, resources, and applications.
- **Resource Allocation -** The hypervisor allocates resources, such as CPU, memory, and storage, to each VM.
- **VM Execution -** The VM runs its own operating system and applications, isolated from other VMs.

Benefits of Virtualization

Hardware Virtualization

Multiple VMs can run on a single physical machine, maximizing hardware utilization.

Isolation

VMs are isolated from each other, ensuring that if one VM crashes, it won't affect others.

Flexibility

VMs can be easily created, cloned, and deleted as needed.

Portability

VMs are portable across different physical machines, making it easy to move them between environments.

Comparison with Virtualization

Containerization is often compared to virtualization, but there are key differences:

Virtualization: Creates a complete, self-contained operating system instance, including the kernel, for each virtual machine.

Containerization: Runs multiple isolated environments on top of a single host operating system, sharing the same kernel.

Introduction to Docker

Docker is a software platform that allows you to build, test, and deploy applications quickly using containers. It's a containerization platform that enables developers to package their applications and all their dependencies into a standardized unit called a container. This container includes everything the application needs to run, such as code, libraries, system tools, and runtime.

Docker provides a client-server architecture, where the Docker host is a physical or virtual machine running Linux (or another Docker-Engine compatible OS). The Docker Engine is a client-server application consisting of the Docker daemon, a Docker API that interacts with the daemon, and a command-line interface (CLI) that talks to the daemon.

Docker objects are components of a Docker deployment that help package and distribute applications. They include images, containers, networks, volumes, plug-ins, and more. Docker images contain executable application source code and all the tools, libraries, and dependencies the application code needs to run as a container. Docker containers are the live, running instances of Docker images.

Docker is an open-source platform, free to download. There is also Docker Inc., the company that sells the commercial version of Docker. Docker comes with a command-line interface (CLI), using which you can do all of the operations that the platform provides.

Here are some Docker terminology:

Images: The blueprints of our application which form the basis of containers. These contain all of the configuration settings that define the isolated environment.

Containers: These are instances of a Docker image and are what run the actual application.

Docker Daemon: That background service running on the host that listens to API calls (via the Docker client), manages images, and building, running, and distributes containers.

Docker Client: The command line tool that allows the user to interact with the daemon.

Docker Hub: A registry of Docker images containing all available Docker images. A user can have their registry, from which they can pull images.

Introduction to Nginx

Nginx is a versatile and powerful web server that can be used in various scenarios. Its high performance, efficiency, and flexibility make it a popular choice for many web applications and infrastructure setups.

Nginx (pronounced "engine-x") is a popular open-source web server known for its high performance, efficiency, and scalability. It's widely used by various organizations, including tech giants like Netflix, Dropbox, and Airbnb.

Key Features and Benefits:

- **High Performance:** Nginx is designed to handle a large number of concurrent connections efficiently, making it ideal for high-traffic websites and applications.
- **Asynchronous Architecture:** It uses an asynchronous event-driven architecture, allowing it to handle multiple requests simultaneously without blocking.
- **Reverse Proxy:** Nginx can act as a reverse proxy, load-balancing requests across multiple backend servers and providing caching capabilities.
- **Static File Serving:** It efficiently serves static content like HTML, CSS, JavaScript, and images.
- **Flexibility:** Nginx offers a modular architecture with various modules available for different functionalities, such as HTTP/2, WebSocket, and geoIP.
- **Low Resource Consumption:** Nginx is known for its lightweight design and low resource usage, making it suitable for environments with limited resources.

Common Use Cases:

- **Web Server:** Nginx can serve as a standalone web server for static content and dynamic applications.
- **Reverse Proxy:** It can be used as a reverse proxy to load balance traffic across multiple application servers or to provide caching and security features.
- **API Gateway**: Nginx can act as an API gateway, managing and routing requests to different backend services.
- **Load Balancer:** Nginx can distribute traffic across multiple servers based on various load-balancing algorithms.

Basic Nginx Configuration:

An Nginx configuration file (usually named nginx.conf) defines the server blocks, listening ports, and other settings for Nginx. Here's a simplified example:

```
http {
  server {
    listen 80;
    server_name example.com;
    location / {
      root /var/www/html;
      index index.html index.htm;
    }
  }
}
```

This configuration:

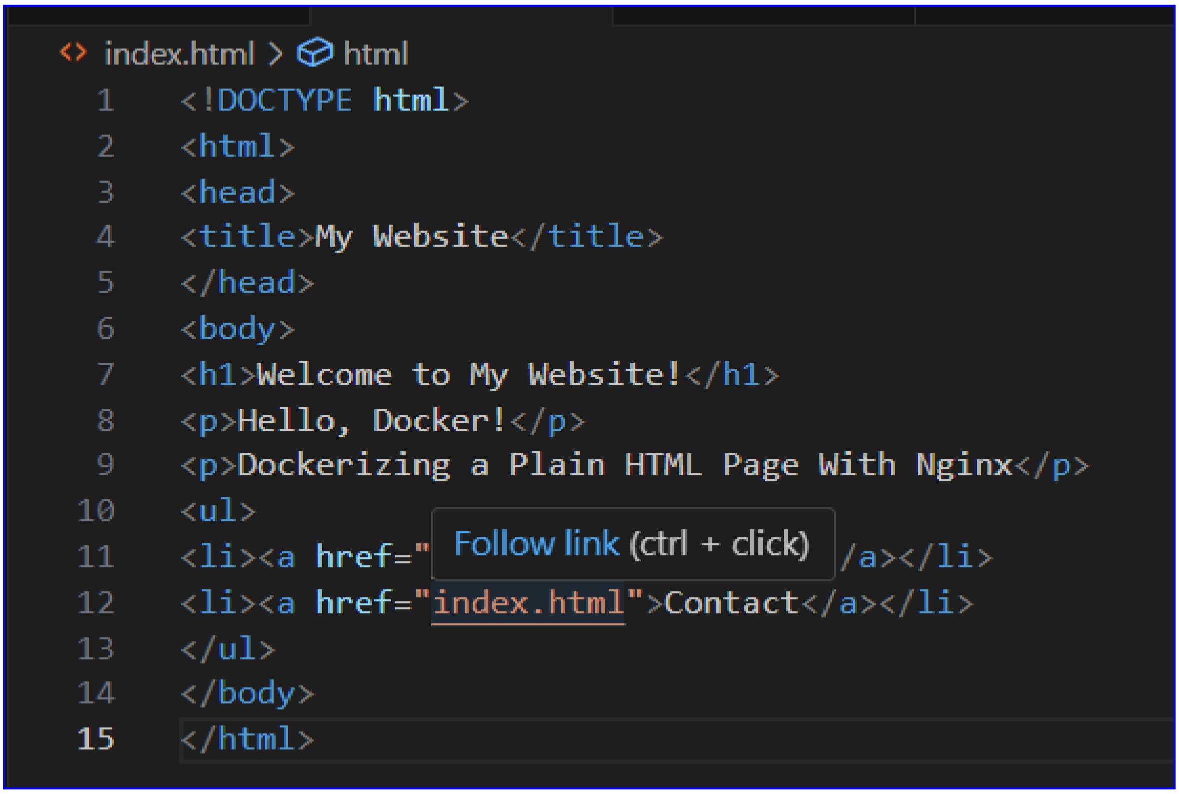
- Listens on port **80**.
- Defines a virtual host for **example.com**.
- Serves static files from the **/var/www/htm**l directory.
- Sets **index.html** and **index.htm** as the default files to serve.

Dockerizing a Plain HTML Page With Nginx

Purpose of Files

- index.html

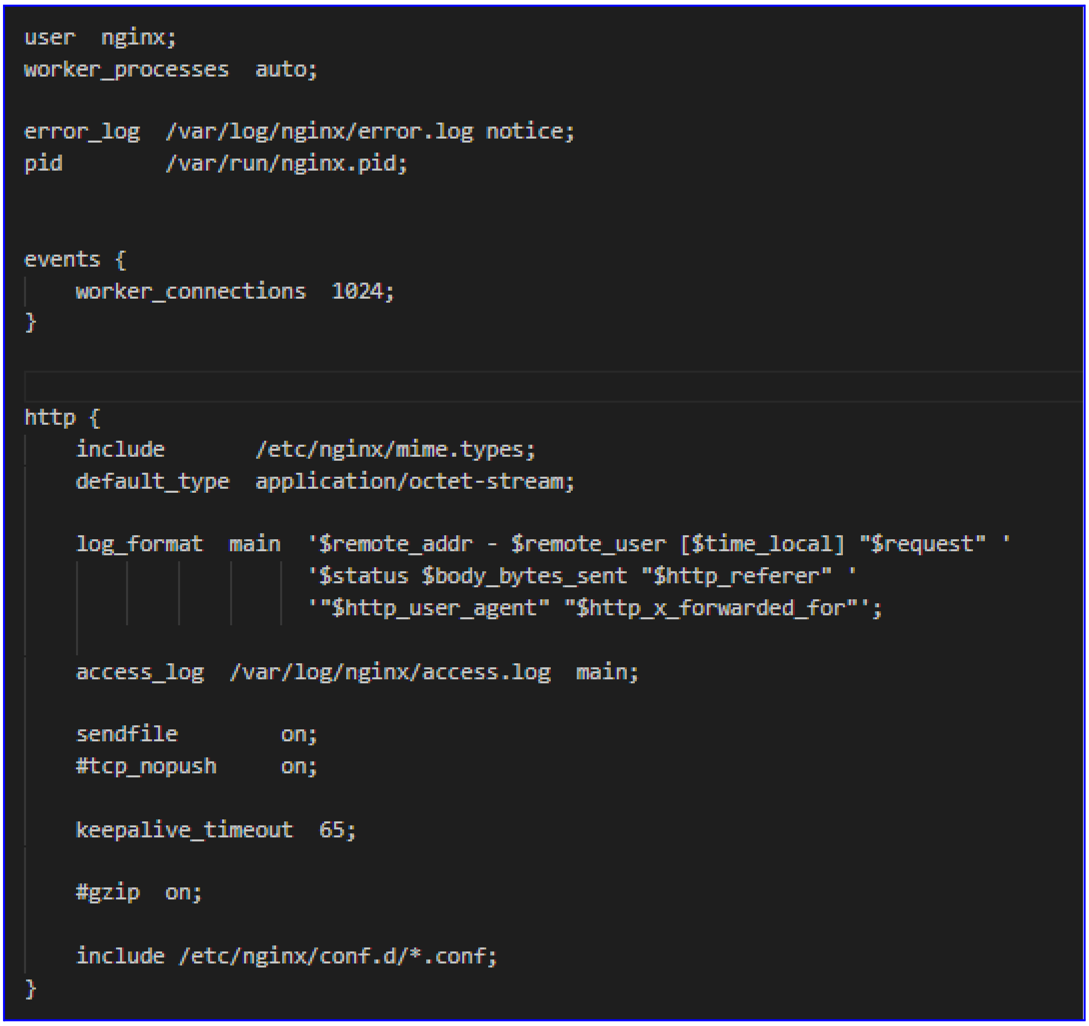
When a user enters a website's domain name without specifying a file, the web server automatically looks for an index.html file in the root directory and serves its content.



```
<!DOCTYPE html>
<html>
<head>
<title>My Website</title>
</head>
<body>
<h1>Welcome to My Website!</h1>
<p>Hello, Docker!</p>
<p>Dockerizing a Plain HTML Page With Nginx</p>
<ul>
<li><a href="                   /a></li>
<li><a href="index.html">Contact</a></li>
</ul>
</body>
</html>
```

- nginx.conf

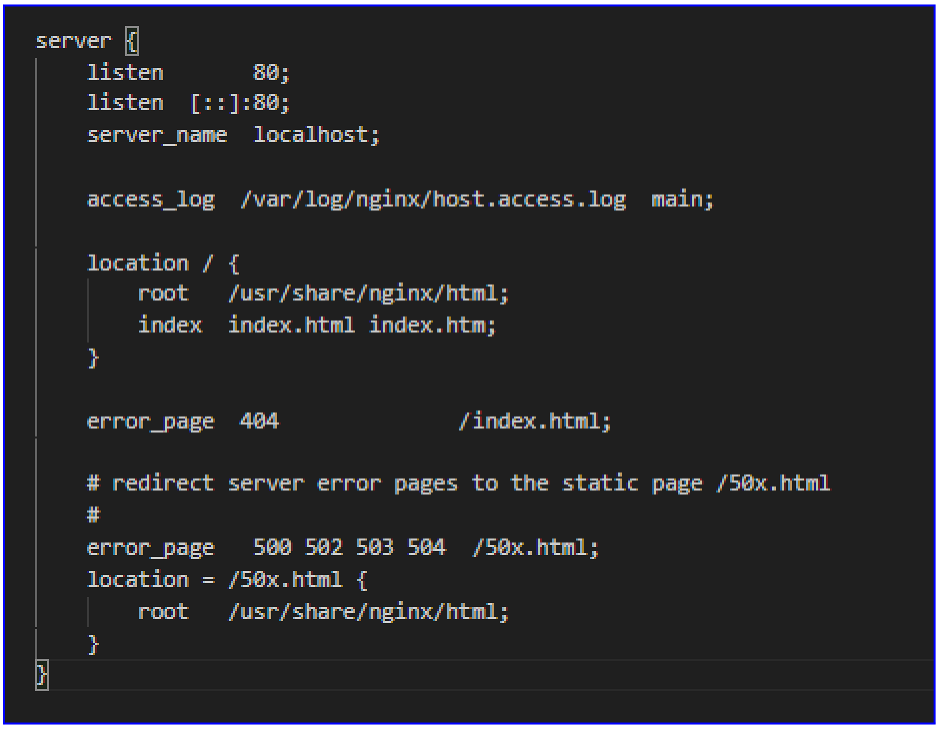
An Nginx configuration file is a text-based file that defines the rules and settings for how the Nginx web server should operate. It determines how Nginx handles incoming requests, processes them, and sends responses to clients.



```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

- default.conf

The defaults.conf file in Nginx serves as a global configuration file, providing default settings that apply to all server blocks within the main configuration file (typically nginx.conf). It allows you to define common parameters and configurations that can be reused across multiple virtual hosts.



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

    access_log  /var/log/nginx/host.access.log  main;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    error_page  404              /index.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

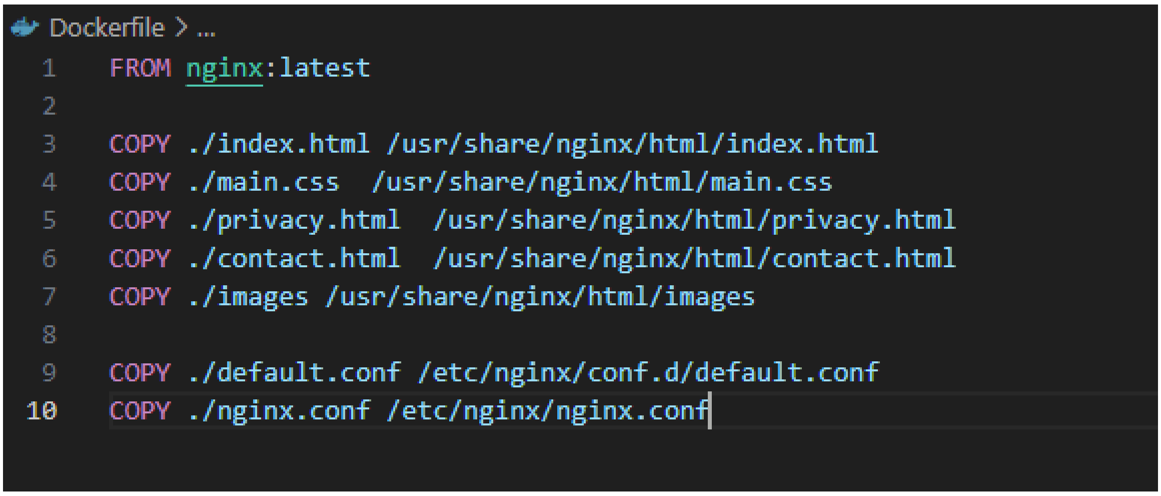
- Dockerfile

The Dockerfile i.e. Blueprint for Container Images is a text document that contains a series of instructions or commands used to build Docker images. These instructions specify the base image to use, the files and directories to add, and the commands to run within the container.

Here are some key functions of a Dockerfile -

1. Base Image Specification: The first line of a Dockerfile usually specifies the base image to use as a starting point for building the new image. This base image can be an official image from Docker Hub or a custom-built image.
2. File and Directory Operations: You can use commands like COPY and ADD to add files and directories to the image. These commands can be used to include application code, libraries, and dependencies.
3. Command Execution: You can execute commands within the container using the RUN command. This allows you to install packages, configure the environment, or perform other tasks necessary for your application.
4. Environment Variables: You can set environment variables using the ENV command. These variables can be accessed within the container and used to configure the application's behavior.
5. Exposing Ports: If your application needs to listen on specific ports, you can use the EXPOSE command to expose those ports. This allows other containers or hosts to connect to the application.
6. Workdir: The WORKDIR command sets the working directory within the container. This directory will be the current directory for subsequent commands.

Building the Docker Image



```
FROM nginx:latest

COPY ./index.html /usr/share/nginx/html/index.html
COPY ./main.css  /usr/share/nginx/html/main.css
COPY ./privacy.html  /usr/share/nginx/html/privacy.html
COPY ./contact.html  /usr/share/nginx/html/contact.html
COPY ./images /usr/share/nginx/html/images

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

Building the Docker Image

A Docker image is a lightweight, standalone, and executable package of software that includes everything an application needs to run, such as code, libraries, dependencies, and settings. It's essentially a snapshot of a container's filesystem at a particular point in time.

Docker images are created using a Dockerfile, which is a text file that contains a series of instructions for building an image. The Dockerfile specifies the base image, copies files, sets environment variables, and defines commands to run during the build process.

Components of a Docker Image:

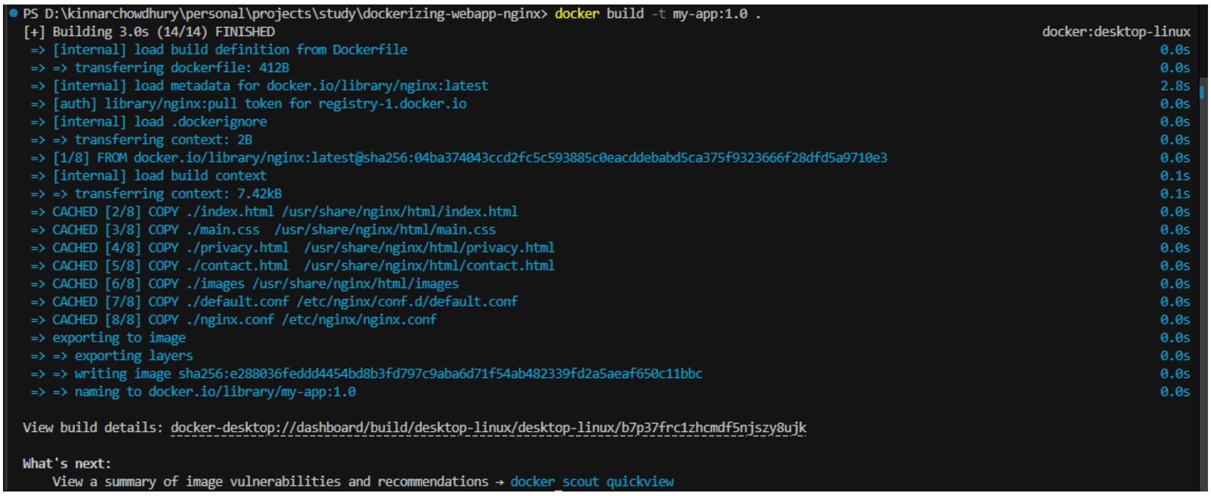
- Base Image: The starting point for the image, which can be another Docker image or a base operating system.
- Layers: A series of read-only layers that make up the image, each representing a change or addition to the previous layer.
- Container Configuration: Metadata that defines how the container should be run, such as environment variables, ports, and volumes.

Benefits of Docker Images:

- Portability: Docker images can be run on any system that supports Docker, without worrying about compatibility issues.
- Efficiency: Docker images are lightweight and use fewer resources than traditional virtual machines.
- Consistency: Docker images ensure consistency across different environments, making it easier to develop, test, and deploy applications.

```
#! Go to the Project Directory. Before creating your docker image please ensure the docker engine is running
# ------ For Windows ------
# Check Docker Desktop Application in your Windows or install it from https://docs.docker.com/desktop/install/windows-install/
# ------------------------
# ------- For Linux -------
# sudo systemctl status docker
# ========================
```

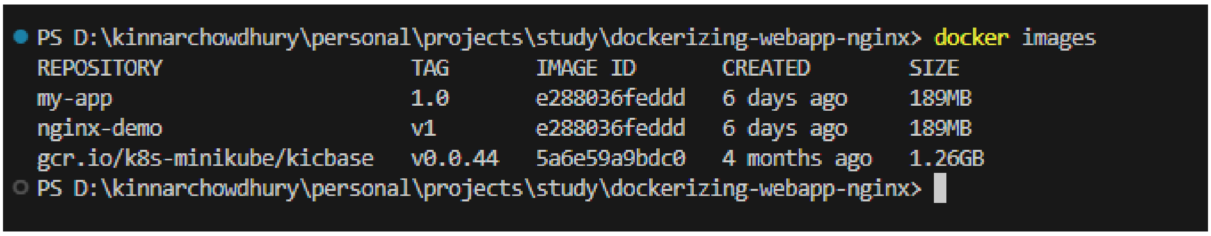
```
cd <project-directory>

docker build -t <image-name>:<tag-id> .
# e.g: docker build -t my-app:1.0 .
```



```
PS D:\kinnarchowdhury\personal\projects\study\dockerizing-webapp-nginx> docker build -t my-app:1.0 .
[+] Building 3.0s (14/14) FINISHED                                                                 docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                              0.0s
 => => transferring dockerfile: 412B                                                                              0.0s
 => [internal] load metadata for docker.io/library/nginx:latest                                                   2.8s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                      0.0s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [1/8] FROM docker.io/library/nginx:latest@sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3  0.0s
 => [internal] load build context                                                                                 0.1s
 => => transferring context: 7.42kB                                                                               0.1s
 => CACHED [2/8] COPY ./index.html /usr/share/nginx/html/index.html                                               0.0s
 => CACHED [3/8] COPY ./main.css  /usr/share/nginx/html/main.css                                                  0.0s
 => CACHED [4/8] COPY ./privacy.html  /usr/share/nginx/html/privacy.html                                          0.0s
 => CACHED [5/8] COPY ./contact.html  /usr/share/nginx/html/contact.html                                          0.0s
 => CACHED [6/8] COPY ./images /usr/share/nginx/html/images                                                       0.0s
 => CACHED [7/8] COPY ./default.conf /etc/nginx/conf.d/default.conf                                               0.0s
 => CACHED [8/8] COPY ./nginx.conf /etc/nginx/nginx.conf                                                          0.0s
 => exporting to image                                                                                            0.0s
 => => exporting layers                                                                                           0.0s
 => => writing image sha256:e288036feddd4454bd8b3fd797c9aba6d71f54ab482339fd2a5aeaf650c11bbc                      0.0s
 => => naming to docker.io/library/my-app:1.0                                                                     0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/b7p37frc1zhcmdf5njszy8ujk

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

To List your docker images

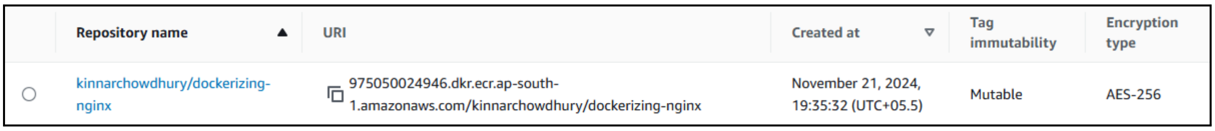
docker images



```
PS D:\kinnarchowdhury\personal\projects\study\dockerizing-webapp-nginx> docker images
REPOSITORY                    TAG       IMAGE ID       CREATED        SIZE
my-app                        1.0       e288036feddd   6 days ago     189MB
nginx-demo                    v1        e288036feddd   6 days ago     189MB
gcr.io/k8s-minikube/kicbase   v0.0.44   5a6e59a9bdc0   4 months ago   1.26GB
PS D:\kinnarchowdhury\personal\projects\study\dockerizing-webapp-nginx>
```

Push the image on ECR

Create an Private Image Repository in Amazon Elastic Container Registry (ECR) and use the following steps to authenticate and push an image to your repository.



	Repository name ▲	URI	Created at ▽	Tag immutability	Encryption type
○	kinnarchowdhury/dockerizing-nginx	975050024946.dkr.ecr.ap-south-1.amazonaws.com/kinnarchowdhury/dockerizing-nginx	November 21, 2024, 19:35:32 (UTC+05.5)	Mutable	AES-256

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

```
aws ecr get-login-password --region ap-south-1 | docker login --username AWS
--password-stdin 975050024946.dkr.ecr.ap-south-1.amazonaws.com
```

Note: If you receive an error using the AWS CLI, ensure you have installed the latest version of the AWS CLI and Docker.

2. Build your Docker image using the following command. You can skip this step if your image is already built (InMy Case, My Image name is my-app:1.0):

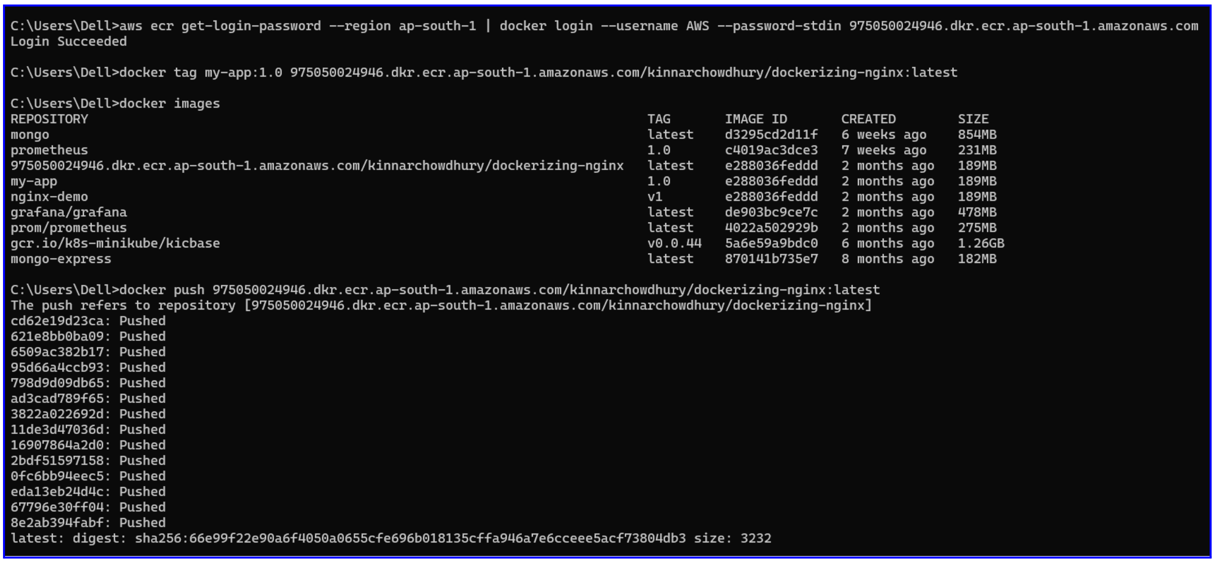
```
docker build -t kinnarchowdhury/dockerizing-nginx .
```

3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag my-app:1.0
975050024946.dkr.ecr.ap-south-1.amazonaws.com/kinnarchowdhury/dockerizing-nginx:latest
```

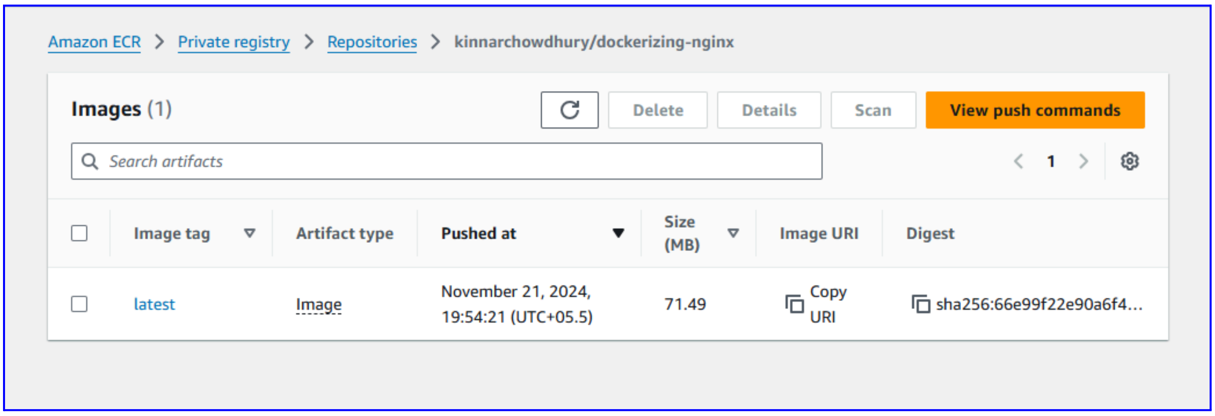
4. Run the following command to push this image to your newly created AWS repository:

```
docker push
975050024946.dkr.ecr.ap-south-1.amazonaws.com/kinnarchowdhury/dockerizing-nginx:latest
```



```
C:\Users\Dell>aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 975050024946.dkr.ecr.ap-south-1.amazonaws.com
Login Succeeded

C:\Users\Dell>docker tag my-app:1.0 975050024946.dkr.ecr.ap-south-1.amazonaws.com/kinnarchowdhury/dockerizing-nginx:latest

C:\Users\Dell>docker images
REPOSITORY                                                                         TAG       IMAGE ID       CREATED        SIZE
mongo                                                                              latest    d3295cd2d11f   6 weeks ago    854MB
prometheus                                                                         1.0       c4019ac3dce3   7 weeks ago    231MB
975050024946.dkr.ecr.ap-south-1.amazonaws.com/kinnarchowdhury/dockerizing-nginx   latest    e288036feddd   2 months ago   189MB
my-app                                                                             1.0       e288036feddd   2 months ago   189MB
nginx-demo                                                                         v1        e288036feddd   2 months ago   189MB
grafana/grafana                                                                    latest    de903bc9ce7c   2 months ago   478MB
prom/prometheus                                                                    latest    4022a502929b   2 months ago   275MB
gcr.io/k8s-minikube/kicbase                                                        v0.0.44   5a6e59a9bdc0   6 months ago   1.26GB
mongo-express                                                                      latest    870141b735e7   8 months ago   182MB

C:\Users\Dell>docker push 975050024946.dkr.ecr.ap-south-1.amazonaws.com/kinnarchowdhury/dockerizing-nginx:latest
The push refers to repository [975050024946.dkr.ecr.ap-south-1.amazonaws.com/kinnarchowdhury/dockerizing-nginx]
cd62e19d23ca: Pushed
621e8bb0ba09: Pushed
6509ac382b17: Pushed
95d66a4ccb93: Pushed
798d9d09db65: Pushed
ad3cad789f65: Pushed
3822a022692d: Pushed
11de3d47036d: Pushed
16907864a2d0: Pushed
2bdf51597158: Pushed
0fc6bb94eec5: Pushed
eda13eb24d4c: Pushed
67796e30ff04: Pushed
8e2ab394fabf: Pushed
latest: digest: sha256:66e99f22e90a6f4050a0655cfe696b018135cffa946a7e6cceee5acf73804db3 size: 3232
```

After Pushing the Image you can validate the image in the ECR whether it is available in your repository.



Amazon ECR > Private registry > Repositories > kinnarchowdhury/dockerizing-nginx
Images (1)
Delete
Details
Scan
View push commands
Search artifacts
1

☐	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
☐	latest	Image	November 21, 2024, 19:54:21 (UTC+05.5)	71.49	Copy URI	sha256:66e99f22e90a6f4...

References:

1. https://www.google.com/
2. https://docs.docker.com/
3. https://docs.aws.amazon.com/ecr/
4. https://vlearnv.herovired.com/program-timeline/186