

Name: Rohit Jain

Roll No.: 333048

Prn No.: 22010315

Division: TY-IT-C2

Assignment No 7

Title: Deploy a static website using Docker.

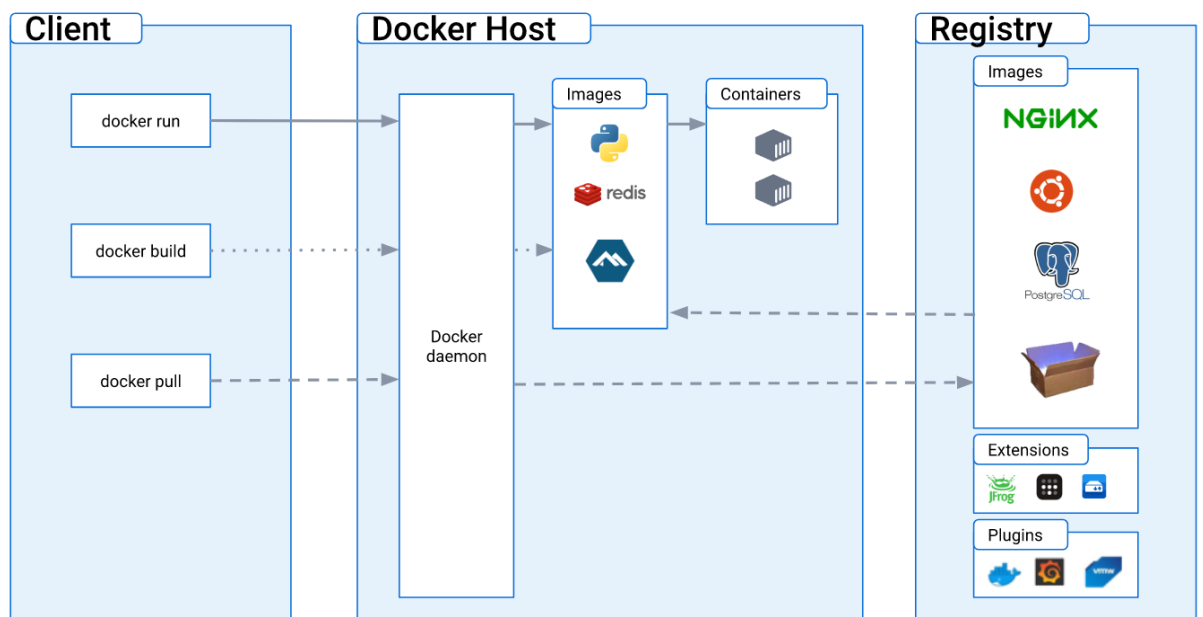
Theory:

1) What is Docker?

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 quickly, you can significantly reduce the delay between writing code and running it in production.

2) Docker Architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



1. **Docker Daemon:** The Docker daemon is the core component of the Docker architecture. It runs on the host machine and manages the lifecycle of Docker containers and images.
2. **Docker Client:** The Docker client is a command-line interface (CLI) tool that allows developers to interact with the Docker daemon. It can be used to build, run, and manage Docker containers and images.
3. **Docker Registries:** Docker registries are repositories where Docker images are stored and shared. The most common Docker registry is Docker Hub, which is a public repository that hosts thousands of pre-built Docker images.
4. **Docker Images:** Docker images are the building blocks of Docker containers. They contain everything that is needed to run an application, including the application code, dependencies, and runtime environment.
5. **Docker Containers:** Docker containers are instances of Docker images that are running on a host machine. They are isolated from the host environment and from other containers, providing a consistent runtime environment for the application.
6. **Docker Compose:** Docker Compose is a tool that allows developers to define and manage multi-container Docker applications. It uses YAML files to define the services, networks, and volumes that make up the application.

Overall, the Docker architecture provides a powerful and flexible platform for building, packaging, and deploying applications in a consistent and scalable way.

3) Difference between Docker and Virtual machine

Docker and virtual machines (VMs) are two different technologies used for creating isolated environments to run applications. While they share some similarities, there are also some key differences between them.

1. **Architecture:** Docker uses containerization technology to create isolated environments, while VMs use hypervisor technology to create virtualized environments.
2. **Resource Utilization:** Docker containers share the host operating system's kernel, which allows them to use fewer resources and start up faster than VMs. In contrast, VMs require their own complete operating system, which can consume more resources and take longer to start up.
3. **Portability:** Docker containers are more portable than VMs because they can be easily moved between hosts and platforms, while VMs are tied to a specific host system architecture.
4. **Security:** Docker containers are generally considered to be less secure than VMs because they share the host operating system's kernel, which can potentially expose them to security vulnerabilities. VMs, on the other hand, are more isolated and provide a higher level of security.
5. **Management:** Docker containers are easier to manage than VMs because they can be managed with command-line tools, APIs, and container orchestration platforms like Kubernetes. VMs require more complex management tools, such as virtual machine managers and hypervisors.

Overall, Docker and VMs have different use cases and trade-offs. Docker is often used for lightweight, portable applications that require high scalability and resource utilization, while VMs are better suited for applications that require higher levels of security and isolation.

4) Docker Commands

Here are some of the most used Docker commands:

1. **docker run**: This command is used to create and run a Docker container. For example, `docker run -it ubuntu:latest` will start a new container running the latest version of the Ubuntu image.
2. **docker ps**: This command lists all the running Docker containers.
3. **docker images**: This command lists all the Docker images that are available locally on your machine.
4. **docker pull**: This command is used to download a Docker image from a registry. For example, `docker pull nginx:latest` will download the latest version of the Nginx image.
5. **docker build**: This command is used to build a Docker image from a Dockerfile. For example, `docker build -t myimage:latest .` will build an image named "myimage" from the Dockerfile in the current directory.
6. **docker stop**: This command is used to stop a running Docker container. For example, `docker stop container_id` will stop the container with the specified ID.
7. **docker rm**: This command is used to remove a Docker container. For example, `docker rm container_id` will remove the container with the specified ID.
8. **docker rmi**: This command is used to remove a Docker image. For example, `docker rmi image_name` will remove the image with the specified name.
9. **docker exec**: This command is used to execute a command inside a running Docker container. For example, `docker exec -it container_id /bin/bash` will open a shell inside the container with the specified ID.

5) Dockerfile

A Dockerfile is a text file that contains instructions for building a Docker image. It provides a way to automate the creation of Docker images, which can then be used to run containers. Here is an example of a simple Dockerfile:

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y curl

CMD ["/usr/bin/curl", "http://example.com"]
```

Here's what each line of the Dockerfile does:

- **FROM** specifies the base image that the new image will be built on. In this case, we are using the latest version of the Ubuntu image as our base image.
- **RUN** executes a command during the build process. In this case, we are using `apt-get` to update the package index and install the `curl` package.
- **CMD** specifies the default command that will be executed when a container is started from the image. In this case, we are using `curl` to make a GET request to `http://example.com`.

To build an image from this Dockerfile, save it as "Dockerfile" in a directory on your machine, navigate to that directory in a terminal, and run the following command:

`docker build -t myimage .`

This will build a Docker image named "myimage" using the instructions in the Dockerfile. The `.` at the end of the command specifies that the build context is the current directory.

Once the image has been built, you can use the `docker run` command to start a container from it.

6) Docker-Compose and Docker-swarm

Docker Compose is a tool that allows you to define and run multi-container Docker applications. It uses a YAML file to define the services, networks, and volumes that make up an application. With Docker Compose, you can easily start and stop a set of containers and manage their configuration and dependencies.

Docker Swarm, on the other hand, is a native clustering tool for Docker that turns a group of Docker hosts into a single, virtual host. It allows you to orchestrate the deployment of Docker containers across multiple hosts, automatically scaling them up and down as needed. Swarm uses a declarative approach to describe the desired state of the system, and continuously monitors and reconciles any differences between the desired state and the current state.

Docker Compose is a tool for managing multi-container Docker applications on a single host, while Docker Swarm is a tool for orchestrating Docker containers across multiple hosts in a cluster.

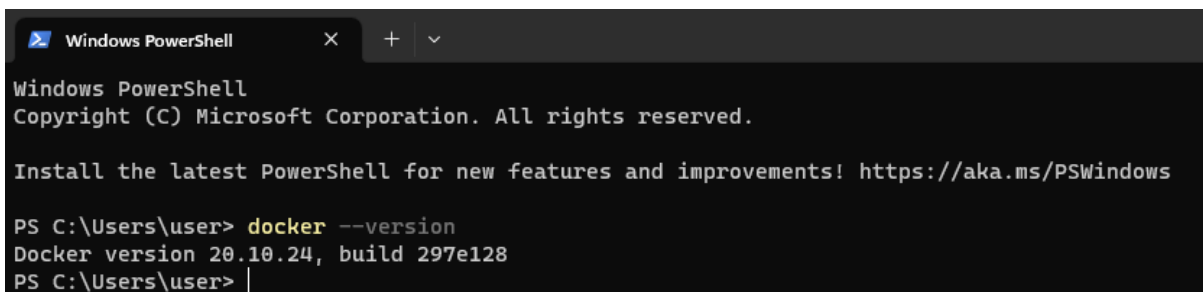
Implementation:

Step 1: Install nginx on windows follow the link:

<http://nginx.org/en/docs/windows.html>

```
cd c:\
unzip nginx-1.23.4.zip
cd nginx-1.23.4
start nginx
```

Step 2: Copy the sample-website in "C:\nginx\html\" folder

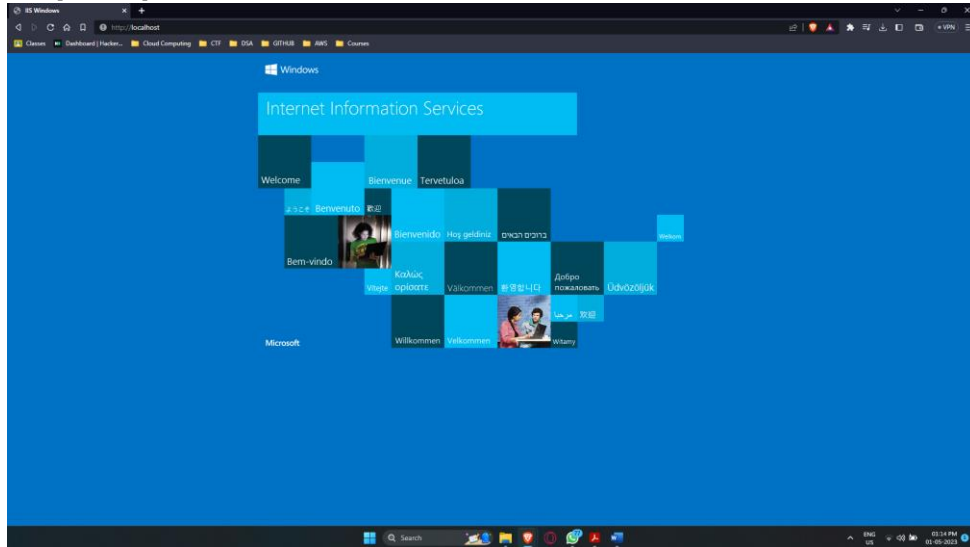
A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The text inside shows the copyright notice for Microsoft Corporation, a link to install PowerShell, and the command `docker --version` being executed. The output of the command is "Docker version 20.10.24, build 297e128".

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user> docker --version
Docker version 20.10.24, build 297e128
PS C:\Users\user> |
```

Step 3: open browser and run "localhost:80"



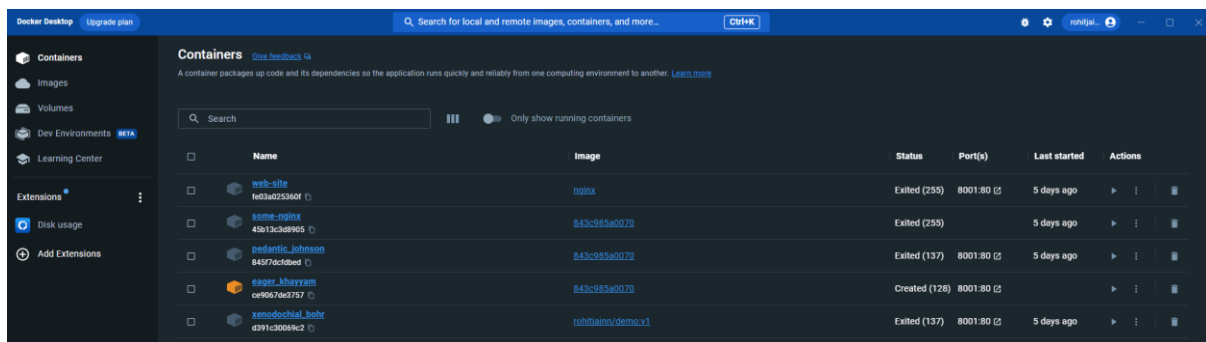
Step 4: Download Docker for windows, follow the link

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

Step 5: Start Docker Desktop

Docker Desktop does not start automatically after installation. To start Docker Desktop:

1. Search for Docker, and select **Docker Desktop** in the search results.



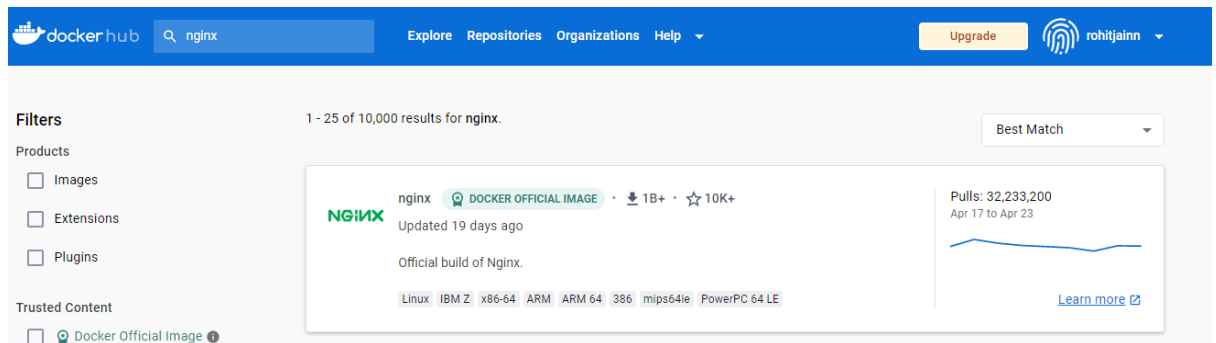
Step 6: Open Powershell and check Docker installation using commands:

- a. `docker -version`
 - b. `docker info`
 - c. `docker version --format '{{json .}}'`
-
-

- **Steps to run the “Sample website” in Docker container**

Step 1) visit to Docker hub web site: <https://hub.docker.com/>

Step 2) search for “nginx” image on site



**Step 3) pull the latest image of nginx using command
“docker pull nginx”**

```
Using default tag: latest
latest: Pulling from library/nginx
26c5c85e47da: Pulling fs layer
4f3256bdf66b: Pulling fs layer
2019c71d5655:
26c5c85e47da: Pull complete
4f3256bdf66b: Pull complete
2019c71d5655: Pull complete
8c767bdbc9ae: Pull complete
78e14bb05fd3: Pull complete
75576236abf5: Pull complete
Digest: sha256:63b44e8ddb83d5dd8020327c1f40436e37a6fffd3ef2498a6204
df23be6e7e94
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

**Step 4) check the docker images on your desktop by using command:
“docker images”**

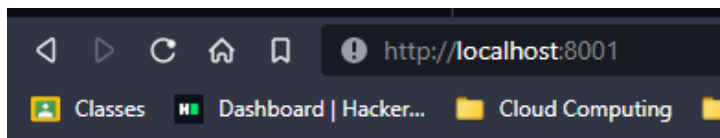
```
PS D:\TY SEM 2\hello_docker> docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
rohitjainn/demo     v1         843c985a0070  5 days ago    142MB
my-app              v1         843c985a0070  5 days ago    142MB
nginx                latest     6efc10a0510f  2 weeks ago   142MB
PS D:\TY SEM 2\hello_docker> |
```

Step 5) go in the "SampleWebsite" folder and then Create a container using the docker command and sync the "SampleWebsite" folder with folder inside the container folder. (This is called Mount Bind")

"docker run -d -p 8001:80 -v \${PWD}:/usr/share/nginx/html --name web-site nginx"

```
PS D:\TY SEM 2\hello_docker> docker run -d -p 8001:80 -v ${PWD}:/usr/share/nginx/html --name index.html nginx
b21bfb8bf02af3f9a5cffeb00dba01596f21e1a9950c614014107936d998ba1f
PS D:\TY SEM 2\hello_docker> docker build -t my-app:v1
"docker build" requires exactly 1 argument.
See 'docker build --help'.
```

Step 6)verify the website open browser and chec "localhost:8001". Now this website is running inside your container.



My First Heading

My first paragraph.

DockerFile

Step 1) Create a Directory structure like

```
App
├── SampleWebSite
└── Dockerfile
```

Step 2) Write a following script into "Dockerfile"

```
Dockerfile X
Dockerfile > ...
1 FROM nginx:latest
2 COPY index.html /usr/share/nginx/html/
3 EXPOSE 80
4
```

Step 3) build image from docker file using command

"docker build -t my-app:v1 ."

```
Build an image from a Dockerfile
PS D:\TY SEM 2\hello_docker> docker build -t my-app:v1 .
[+] Building 0.1s (7/7) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 108B                                              0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/nginx:latest                 0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 31B                                                  0.0s
=> [1/2] FROM docker.io/library/nginx:latest                                   0.0s
=> CACHED [2/2] COPY index.html /usr/share/nginx/html/                        0.0s
=> exporting to image                                                            0.0s
=> => exporting layers                                                            0.0s
=> => writing image sha256:eedfc7698030d98e25fb5fa0c941d259b07457b189ecf6e92459cbb0360760f9 0.0s
=> => naming to docker.io/library/my-app:v1                                    0.0s
```

Step 4) check images using command: docker images

```
PS D:\TY SEM 2\hello_docker> docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
rohitjainn/demo     v1           843c985a0070     5 days ago      142MB
my-app              v1           843c985a0070     5 days ago      142MB
nginx               latest       6efc10a0510f     2 weeks ago     142MB
PS D:\TY SEM 2\hello_docker> |
```

PUSH Image to "DockerHub"

Step 1) login to docker hub using command

1) docker login -u vishalmeshram

```
PS D:\TY SEM 2\hello_docker> docker login -u rohitjainn
Password:
Login Succeeded
```

2) docker images


```
PS D:\TY SEM 2\hello_docker> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
rohitjainn/demo      v1                 843c985a0070       5 days ago        142MB
my-app               v1                 843c985a0070       5 days ago        142MB
nginx                latest             6efc10a0510f       2 weeks ago       142MB
PS D:\TY SEM 2\hello_docker> |
```

3) docker tag (old image name) vishalmeshram/newname

docker tag my-web:v1 vishalmeshram/newapp

```
PS D:\TY SEM 2\hello_docker> docker tag my-app:v1 rohitjainn/my-app:v1
PS D:\TY SEM 2\hello_docker> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
rohitjainn/demo      v1                 843c985a0070       5 days ago        142MB
my-app               v1                 eedfc7698030       5 days ago        142MB
rohitjainn/my-app    v1                 eedfc7698030       5 days ago        142MB
nginx                latest             6efc10a0510f       2 weeks ago       142MB
PS D:\TY SEM 2\hello_docker> |
```

4) docker push vishalmeshram/newapp

```
PS D:\TY SEM 2\hello_docker> docker push rohitjainn/my-app:v1
The push refers to repository [docker.io/rohitjainn/my-app]
749d85311f23: Pushed
9d907f11dc74: Mounted from library/nginx
79974a1a12aa: Mounted from library/nginx
f12d4345b7f3: Mounted from library/nginx
935b5bd454e1: Mounted from library/nginx
fb6d57d46ad5: Mounted from library/nginx
ed7b0ef3bf5b: Mounted from library/nginx
v1: digest: sha256:7f6d0bf7ee1e3114650f386787beeb6412b5ef79fa552efd9b4c9b4fcf16d56d size: 1777
PS D:\TY SEM 2\hello_docker> |
```

Step 5) Login to Docker Hub and check the repository

rohitjainn / my-app

Contains: Image | Last pushed: in a few seconds

Inactive

0

0

Public