Lab 9: Docker

Docker:

Docker is a platform for developing, shipping, and running applications in lightweight, portable containers. It allows developers to package applications along with their dependencies, ensuring consistency across different environments.

How Docker Ensures Consistency Across Different Environments:

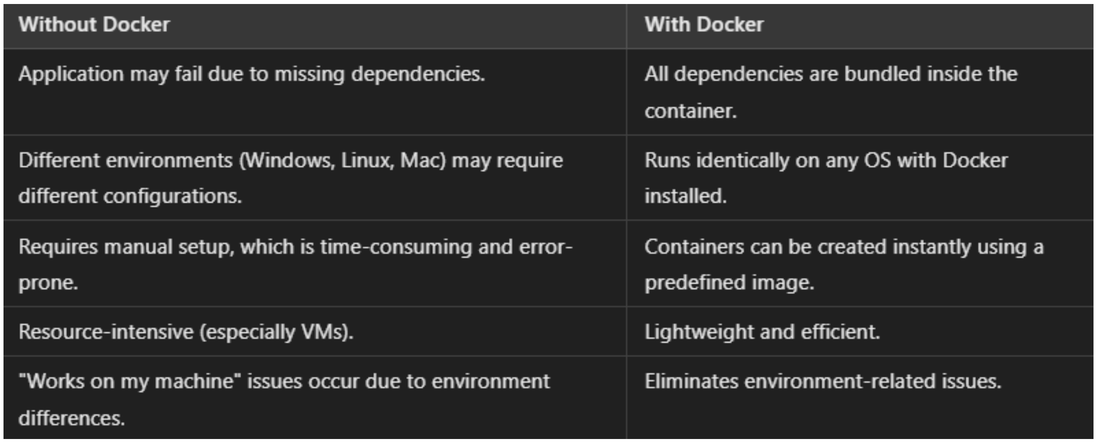
Docker ensures consistency by packaging the application along with all its dependencies, runtime, configurations, and libraries into a **Docker image**. This image is then used to create **containers**, which run identically on any system that has Docker installed—whether it's a developer’s laptop, a testing server, or a production machine.

Key Features That Ensure Consistency:

1. **Isolation** → The application runs in an isolated environment without being affected by host system configurations.
2. **Dependency Management** → The same versions of libraries and dependencies are bundled inside the container.
3. **Works Everywhere** → If a container runs on one machine, it will run the same way on another, regardless of OS differences.
4. **Eliminates "Works on My Machine" Problem** → Developers no longer face issues where an application works on one machine but not on another.

Why Use Docker When We Can Work Without It:

Before Docker, software was typically deployed using **manual installation** or **virtual machines (VMs)**. These approaches had several issues:



Without Docker	With Docker
Application may fail due to missing dependencies.	All dependencies are bundled inside the container.
Different environments (Windows, Linux, Mac) may require different configurations.	Runs identically on any OS with Docker installed.
Requires manual setup, which is time-consuming and error-prone.	Containers can be created instantly using a predefined image.
Resource-intensive (especially VMs).	Lightweight and efficient.
"Works on my machine" issues occur due to environment differences.	Eliminates environment-related issues.

Docker Components:

1. **Docker Image:**

   A Docker image is a blueprint for a container. It includes the application code, runtime, libraries, dependencies, and configurations. Images are immutable and are used to create containers.

2. **Docker Container:**

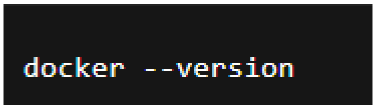
   A Docker container is a running instance of a Docker image. It provides an isolated environment for applications to run.

3. **DockerHub:**

   DockerHub is a cloud-based repository for storing, sharing, and distributing Docker images. It allows developers to pull public images or push their own custom images.

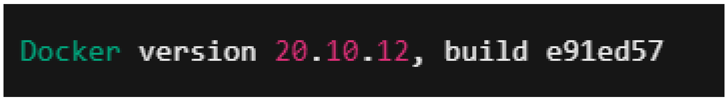
Important Docker Commands:

1. **Check Docker Version:**



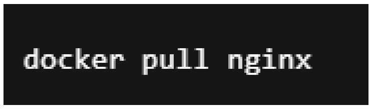
```
docker --version
```

Output:



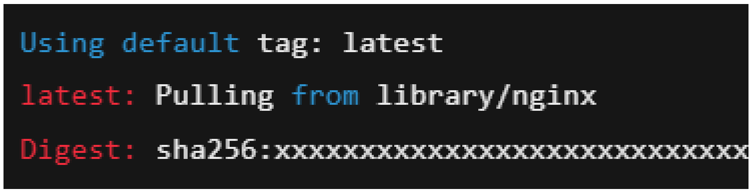
```
Docker version 20.10.12, build e91ed57
```

2. **Pull an Image from DockerHub:**



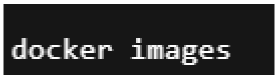
```
docker pull nginx
```

Output:



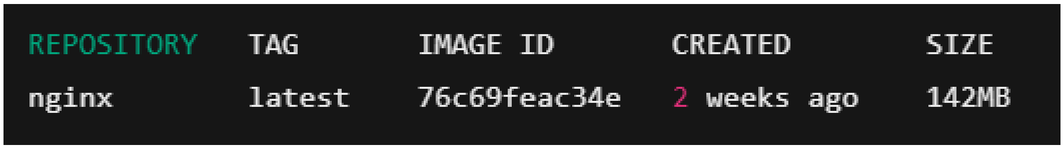
```
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

3. **List Downloaded Images:**



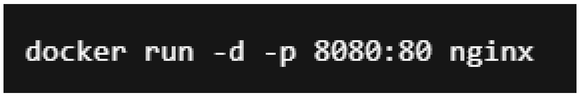
```
docker images
```

Output:



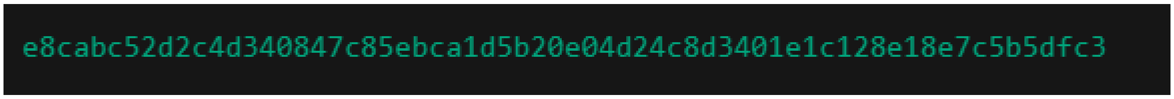
```
REPOSITORY   TAG      IMAGE ID       CREATED        SIZE
nginx        latest   76c69feac34e   2 weeks ago    142MB
```

4. Run a Container:



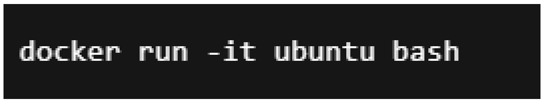
```
docker run -d -p 8080:80 nginx
```

Output:



```
e8cabc52d2c4d340847c85ebca1d5b20e04d24c8d3401e1c128e18e7c5b5dfc3
```

5. Running a Container Interactively:



```
docker run -it ubuntu bash
```

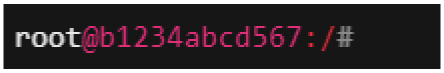
-i (interactive): Keeps STDIN open, allowing you to interact with the container.

-t (terminal): Allocates a pseudo-TTY (terminal), making it behave like a normal terminal.

ubuntu: The base image being used.

bash: The command to execute inside the container (opens a Bash shell).

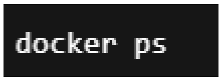
Output:



```
root@b1234abcd567:/#
```

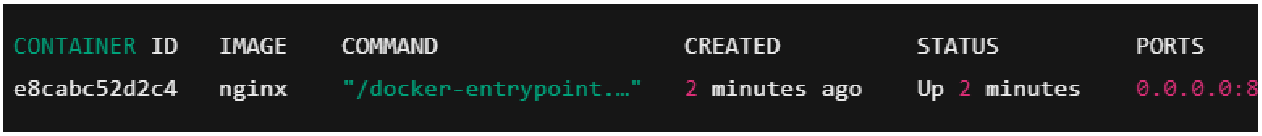
You are now inside the container and can execute commands like ls, pwd, apt update, etc.

6. List Running Containers:



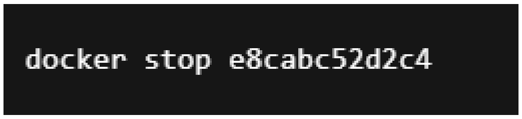
```
docker ps
```

Output:



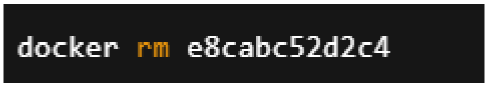
```
CONTAINER ID   IMAGE   COMMAND                  CREATED         STATUS         PORTS
e8cabc52d2c4   nginx   "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   0.0.0.0:8
```

7. **Stop a Running Container:**



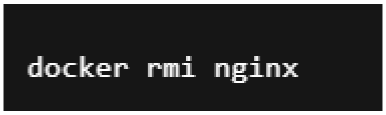
```
docker stop e8cabc52d2c4
```

8. **Remove a Container:**

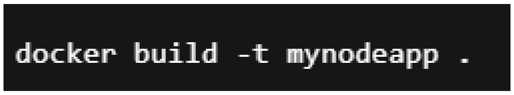


```
docker rm e8cabc52d2c4
```

9. **Remove an Image:**



```
docker rmi nginx
```



```
docker build -t mynodeapp .
```

Task: Pulling and running mongodb inside docker

Step 1. Pull the MongoDB Image

Step 2. Run the MongoDB Container Interactively

Step 3. Connect to the MongoDB Shell

Step 4. Check the MongoDB version

Step 5. List available databases

Step 6. Create a new database on your name

Step 7. Create a new collection named `students` and insert some documents

Step 8. Retrieve all documents from the `students` collection

Step 9. Update a student’s age

Step 10. Delete a student record

Step 11. Check the list of collections

Step 12. Exit the MongoDB shell