**COURSE CODE:** DJS22ITL601 **DATE:**

**COURSE NAME: Software Engineering Laboratory** **CLASS: T.Y.BTech**

**EXPERIMENT NO. 9**

**CO/LO** Analyze real world problem using software engineering principles.

**AIM / OBJECTIVE**: To install and configure Docker.

**DESCRIPTION OF EXPERIMENT:**

**Introduction to 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.

**Why is Docker used?**

Docker is a basic tool, like git or java, that you should start incorporating into your daily development and ops practices.

o Use Docker as version control system for your entire app's operating system

o Use Docker when you want to distribute/collaborate on your app's operating system with a team o Use Docker to run your code on your laptop in the same environment as you have on your server (try the building tool)

o Use Docker whenever your app needs to go through multiple phases of development (dev/test/qa/prod, try Drone or Shippable, both do Docker CI/CD)

o Use Docker with your Chef Cookbooks and Puppet Manifests (remember, Docker doesn't do configuration management)

**What role docker plays in dev-ops?**

Docker is just another tool available to DevOps Engineers. What Docker does is it encapsulates code and code dependencies in a single unit (a container) that can be run anywhere where the Docker engine is installed.

**Why is this useful?**

For multiple reasons; but in terms of CI/CD it can help Engineers separate Configuration from Code, decrease the amount of time spent doing dependency management etc., can use it to scale (with the help of some other tools of course). The list goes on. For example: If I had a single code repository, in my build script I could pull in environment specific dependencies to create a Container that functionally behaves the same in each environment, as I'm building from the same source repository, but it can contain a set of environment specific certificates and configuration files etc. Having said all of that, there really is a great deal you can do to utilize Docker in your CI/CD Pipelines.

**Containerization and its features:**

Containers are made possible by operating system (OS) process isolation and virtualization, which enable multiple application components to share the resources of a single instance of an OS kernel in much the same way that machine virtualization enables multiple virtual machines (VMs) to share the resources of a single hardware server. Containers offer all the benefits of VMs, including application isolation, cost-effective scalability, and disposability. But the additional layer of abstraction (at the OS level) offers important additional advantages:

o Lighter weight: Unlike VMs, containers don’t carry the payload of an entire OS instance—they include only the OS processes and dependencies necessary to execute the code.

o Greater resource efficiency: With containers, you can run several times as many copies of an application on the same hardware as you can using VMs. This can reduce your cloud spending.

o Improved developer productivity: Compared to VMs, containers are faster and easier to deploy, provision, and restart. This makes them ideal for use in continuous integration and continuous delivery (CI/CD) pipelines and a better fit for development teams adopting Agile and DevOps practices.

**What is docker image?**

A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform. It provides a convenient way to package up applications and preconfigured server environments, which you can use for your own private use or share publicly with other Docker users.

**Is docker image and VM same?**

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient. Containers Virtual Machines Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems. Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot

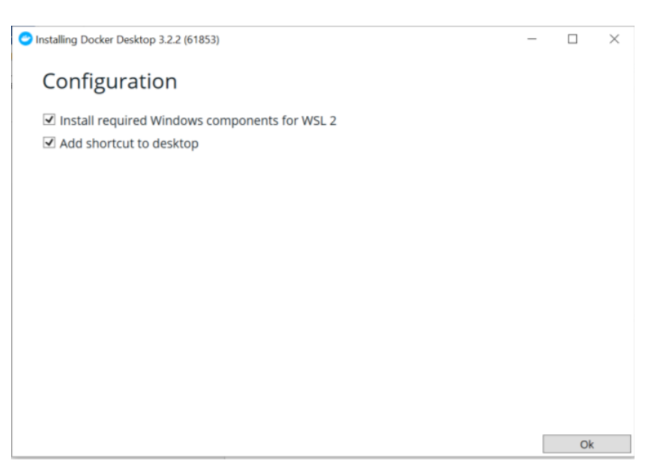
**Steps of Installation:**

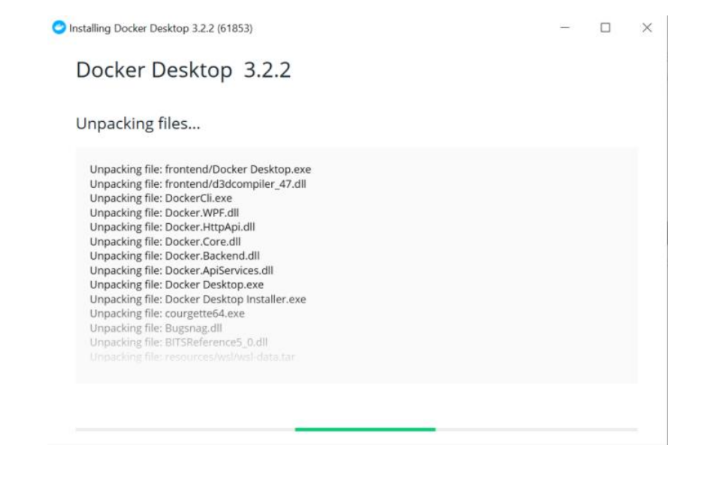
* The following hardware prerequisites are required to successfully run WSL 2 on Windows 10 or Windows 11:
  + 64-bit processor with [Second Level Address Translation (SLAT)](https://en.wikipedia.org/wiki/Second_Level_Address_Translation)
  + 4GB system RAM
  + BIOS-level hardware virtualization support must be enabled in the BIOS settings. For more information, see [Virtualization](https://docs.docker.com/desktop/troubleshoot/topics/#virtualization).
* Download and install the [Linux kernel update package](https://docs.microsoft.com/windows/wsl/wsl2-kernel).

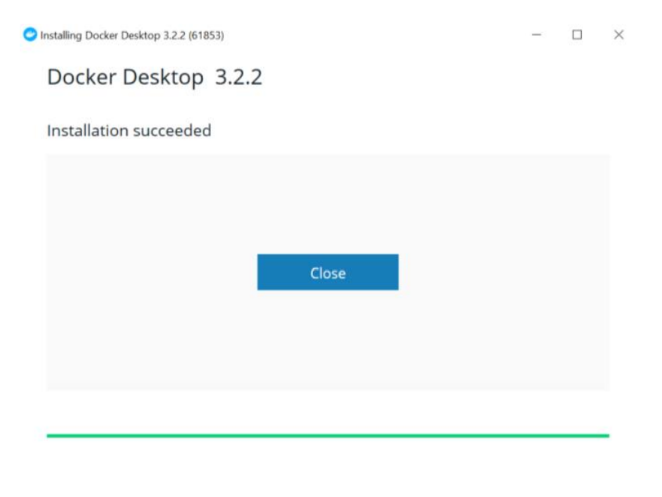
**Install Docker hub from**

https://docs.docker.com/docker-for-windows/install/ and the follow

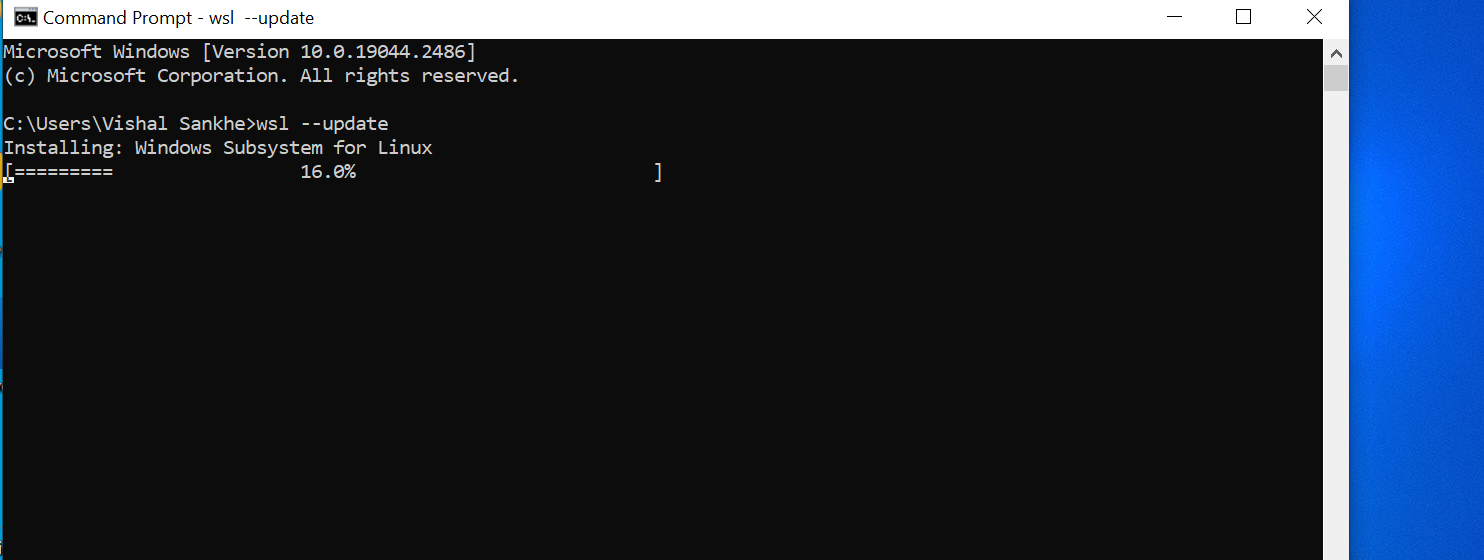
the steps

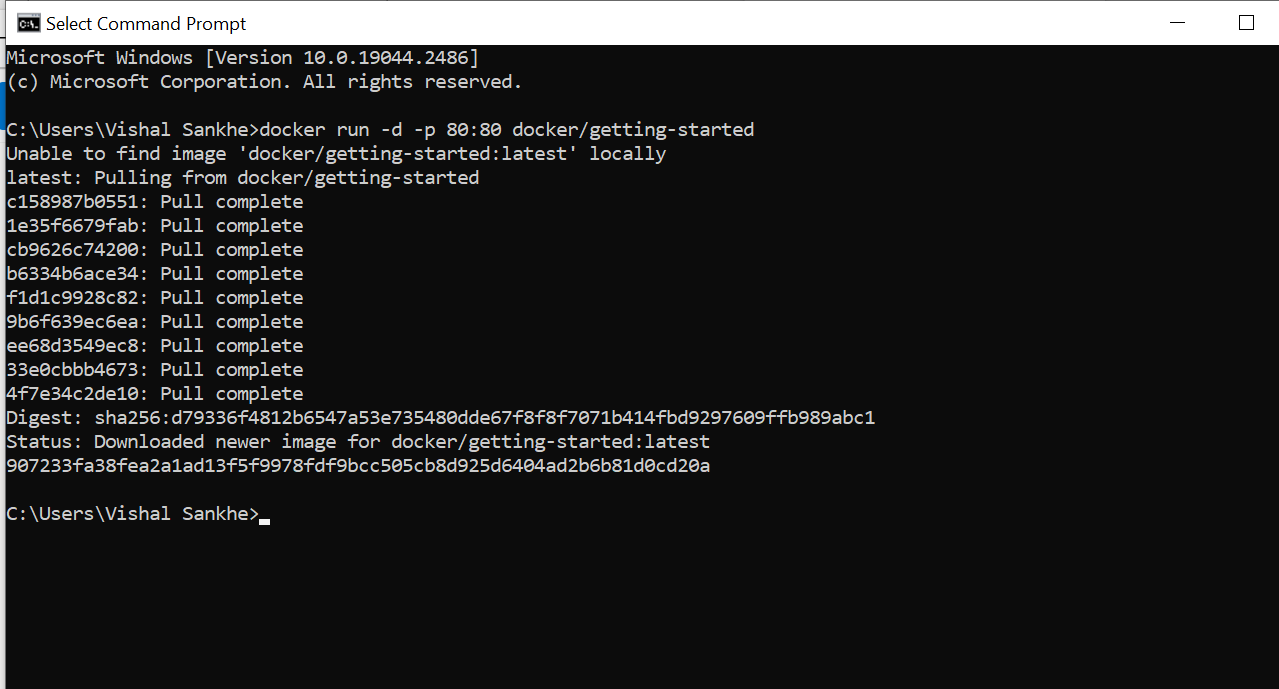




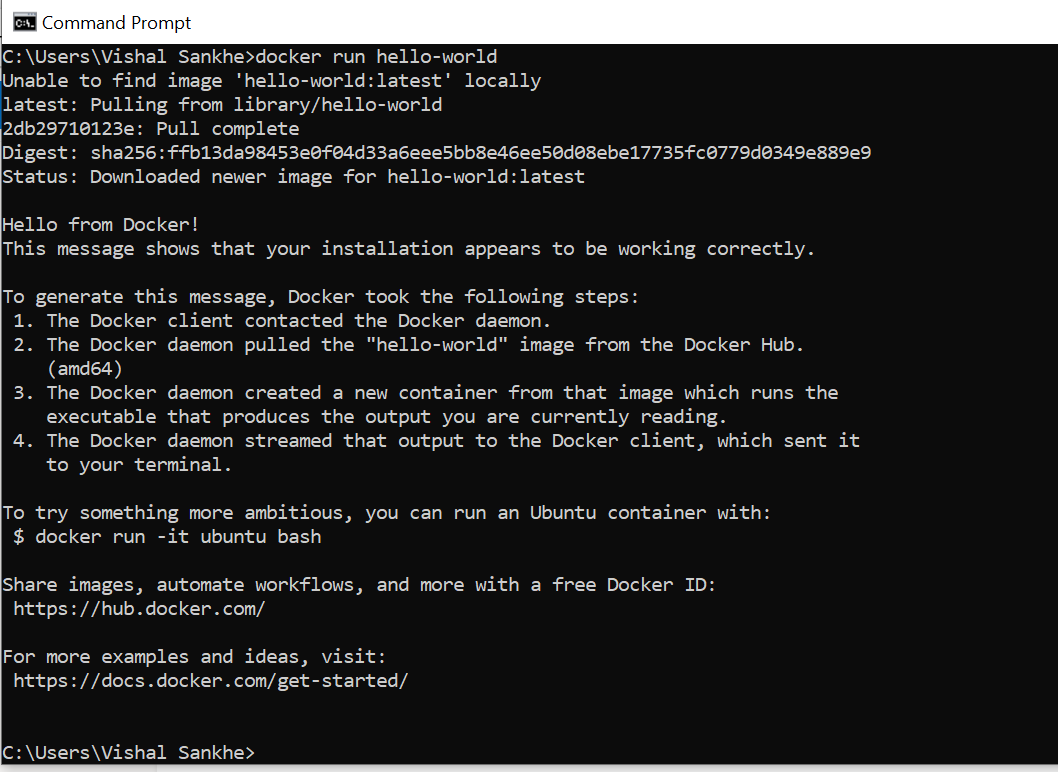


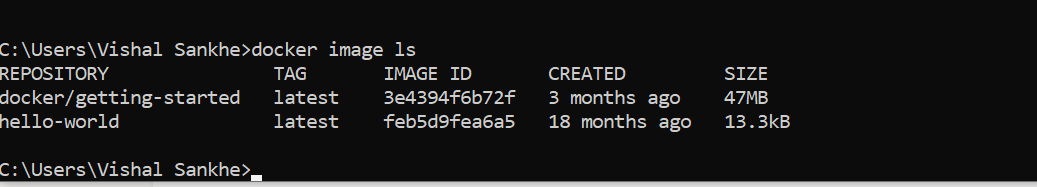
After installation in command prompt execute this command if you are getting error for linux kernel

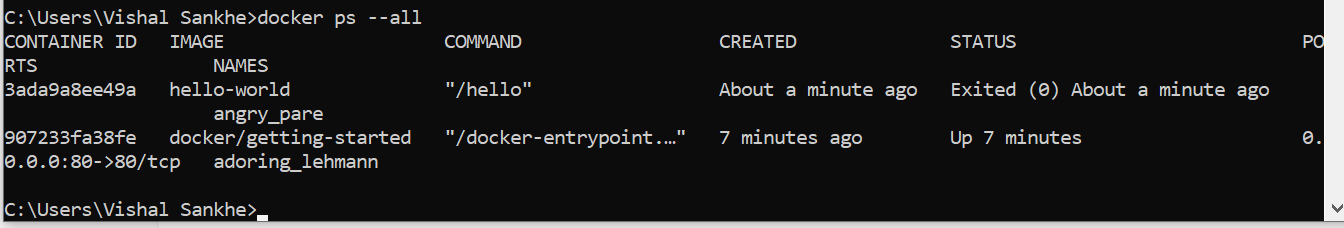




Execute the first docker container using command given below





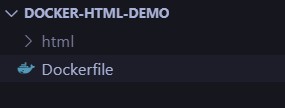


QUESTIONS:

1. Explain the concept of a Docker container and its key components.
2. How does Docker facilitate the development and deployment of applications?

OUTPUT:

**3.Implementation**



DockerFile

FROM nginx:alpine

COPY html /usr/share/nginx/html EXPOSE 80

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Welcome to Dockerized Webpage</title>

<style>

:root {

--primary: #1e90ff;

--accent: #ffb347;

--bg: #f7f9fa;

--text: #222;

--header-bg: #fff;

}

body { margin: 0; font-family: 'Segoe UI', Arial, sans-serif; background: var(--bg); color: var(--text); line-height: 1.6;

}

header { background: var(--header-bg); padding: 3rem 1rem 2rem 1rem;

text-align: left; box-shadow: 0 2px 8px rgba(30,144,255,0.04); display: flex; flex-direction: column; align-items: flex-start; min-height: 60vh; justify-content: center;

}

header h1 {

font-size: 3rem; margin: 0 0 1rem 0; color: var(--primary); font-weight: 700;

letter-spacing: -1px;

}

header p { font-size: 1.25rem; margin-bottom: 2rem;

color: #444; max-width: 500px;

}

.cta-btn { background: var(--primary); color: #fff; padding: 0.9em 2em; border: none; border-radius: 4px; font-size: 1.1rem; cursor: pointer; font-weight: 600; transition: background 0.2s, transform 0.2s; box-shadow: 0 2px 4px rgba(30,144,255,0.1);

}

.cta-btn:hover { background: var(--accent); color: #222;

transform: translateY(-2px) scale(1.03);

}

nav { margin-top: 2rem;

} nav a { text-decoration: none; color: var(--primary); font-weight: 500; margin-right: 2rem; transition: color 0.2s;

} nav a:hover { color: var(--accent);

}

main {

padding: 2rem 1rem; max-width: 900px; margin: auto; background: #fff; border-radius: 12px; box-shadow: 0 4px 24px rgba(30,144,255,0.07); margin-top: -3rem; position: relative; z-index: 2;

}

main h2 { color: var(--primary); font-size: 2rem; margin-bottom: 1rem;

} main ul { list-style: disc inside; margin-bottom: 2rem;

}

main img { max-width: 100%; border-radius: 8px;

margin-bottom: 1rem; box-shadow: 0 1px 8px rgba(0,0,0,0.07);

}

footer { text-align: center; padding: 2rem 1rem 1rem 1rem; color: #888; font-size: 1rem; background: var(--header-bg);

margin-top: 3rem;

}

@media (max-width: 600px) { header h1 { font-size: 2rem;

} main { padding: 1rem 0.5rem; margin-top: -1.5rem;

}

}

</style>

</head>

<body>

<header>

<h1>🚀 Deployed with Docker</h1>

<p>

This modern, responsive webpage is running inside a Docker container using Nginx. Experience the power of containerization and modern web design!

</p>

<button class="cta-btn" onclick="alert('Thank you for visiting!')">Get Started</button>

<nav>

<a href="#features">Features</a>

<a href="#about">About</a>

</nav>

</header>

<main>

<section id="features">

<h2>✨ Features</h2>

<ul>

<li>Containerized deployment with Docker & Nginx</li>

<li>Responsive and accessible design</li>

<li>Modern UI/UX with clear visual hierarchy</li>

<li>Fast loading and optimized for all devices</li>

<li>Micro-interactions and hover effects</li>

</ul>

<img src="https://images.unsplash.com/photo-1461749280684dccba630e2f6?auto=format&fit=crop&w=800&q=80" alt="Modern web design illustration">

</section>

<section id="about">

<h2>About This Demo</h2>

<p>

This webpage demonstrates best practices in modern web design for 2025, including minimalism, whitespace, accessibility, and engaging microinteractions. It’s built to be simple, fast, and visually appealing, making it a perfect candidate for containerized deployment.

</p>

</section>

</main>

<footer>

&copy; 2025 Docker Web Demo &mdash; Crafted for containerization experiments.

</footer>

</body>

</html>

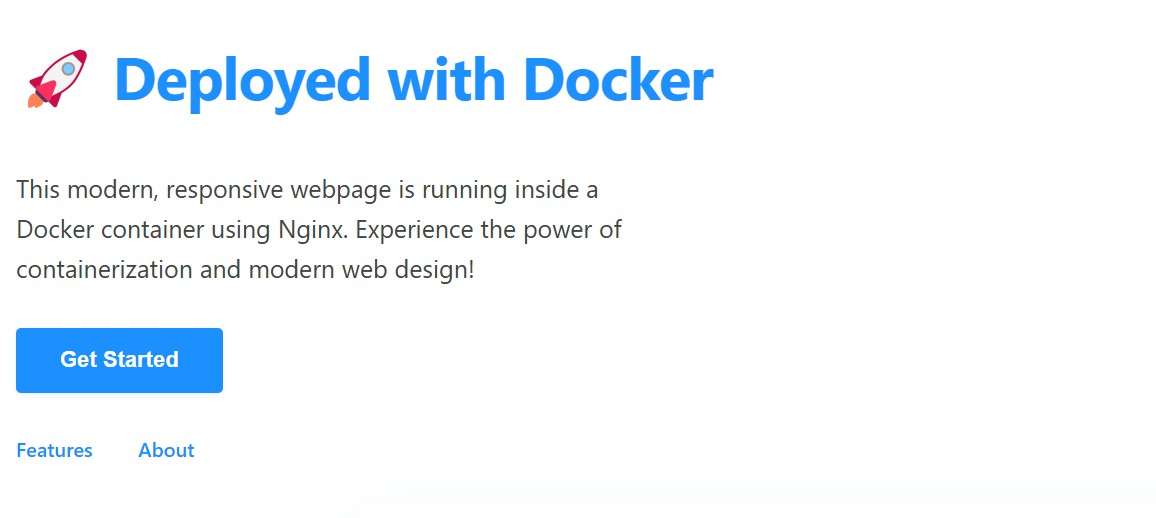
docker build -t my-nginx-html .

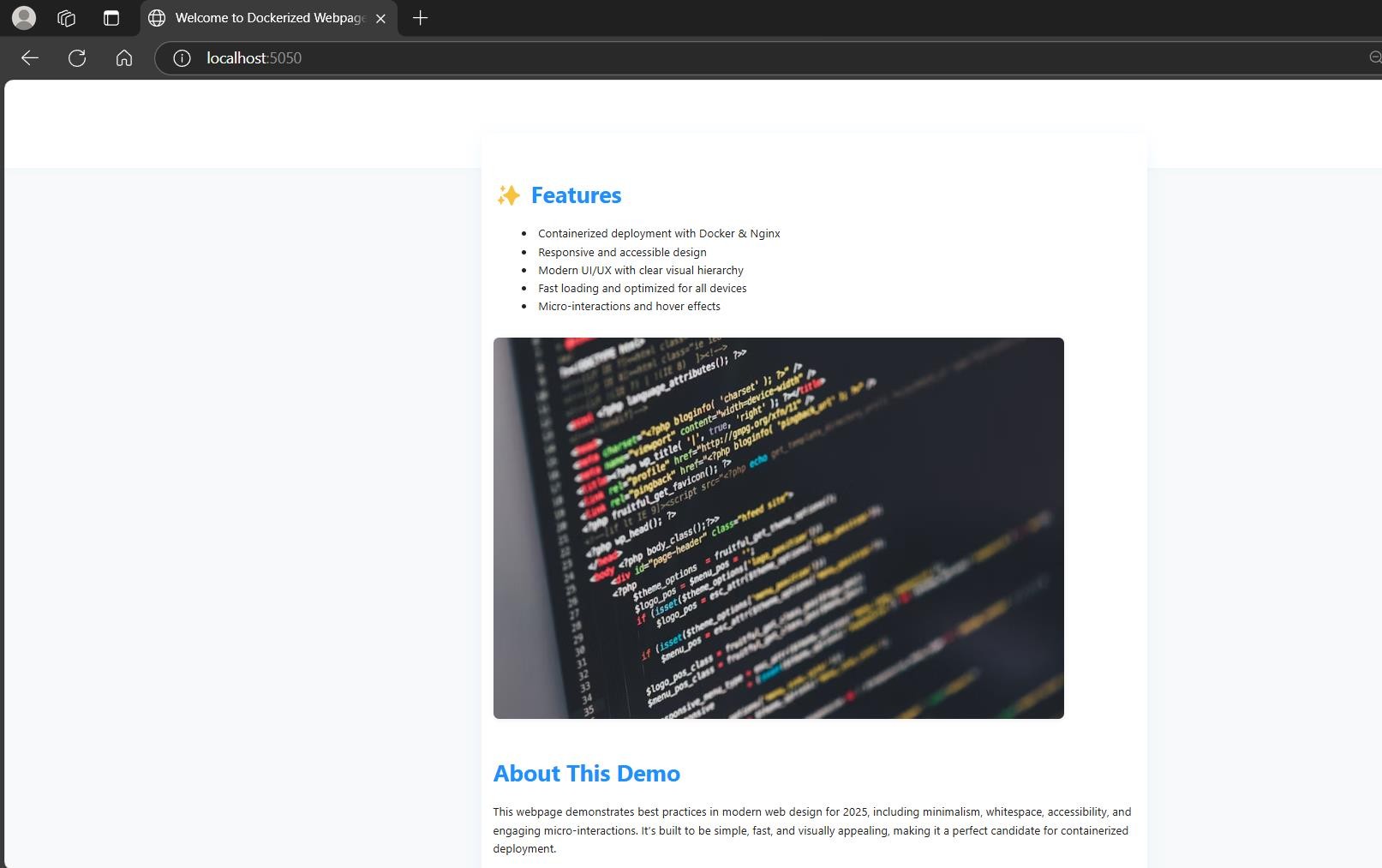


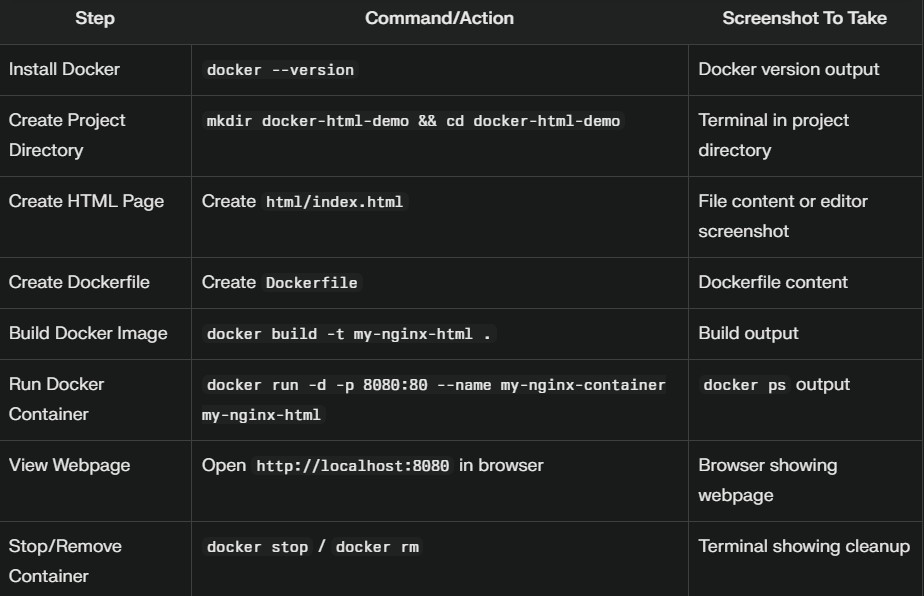
docker run -d -p 5050:80 --name my-nginx-container my-nginx-html



Localhost:5050







**Conclusion**

Containerization with Docker enables consistent, portable deployment of web applications by packaging both code and dependencies together. Deploying a basic HTML webpage in a Docker container demonstrates how easily environments can be replicated across systems. This approach streamlines development, testing, and deployment, ensuring reliability and efficiency

REFERNCE:

https://www.tutorialspoint.com/