VIDYAVARDHAKA COLLEGE OF ENGINEERING

**(Autonomous, affiliated to VTU)**

#### DEPARTMENT OF

**INFORMATION SCIENCE AND ENGINEERING**



# Devops Laboratory

**Open ended Experiment Report on**

# “Website using Docker and Apache”

Submitted in partial fulfillment of the requirement for the completion of VII semester of

**BACHELOR OF ENGINEERING**

### Submitted by:

**KAILASH M 4VV20IS047**

### GAGAN K 4VV20IS050 LIKHITH M 4VV20IS054

**SUBMITTED TO**

Asst Prof. Vidyashree K P

&

## Asst Prof. Shivani T J

Dept.of ISE

***Devops Laboratory – Open ended Experiment***

##### Tools Used

1. Docker
2. Apache HTTP Server
3. Visual Studio Code

# Introduction:

The continuous evolution of web development practices has led to a burgeoning demand for solutions that can streamline the deployment and management of web applications. In response to these challenges, the integration of Docker, a leading containerization platform, with the time-tested Apache HTTP Server has emerged as a robust and efficient solution. This report delves into the symbiotic relationship between Docker and Apache, aiming to provide a comprehensive understanding of the tools used, their individual benefits, and the methodology employed to establish a Dockerized Apache environment.

The contemporary landscape of web development is marked by a need for agility, scalability, and consistency across different environments. Docker, a containerization platform, addresses these requirements by encapsulating applications and their dependencies into self-contained units known as containers. These containers are portable, lightweight, and ensure that applications run consistently across various development, testing, and production environments. Docker has become instrumental in facilitating DevOps practices and accelerating the software development lifecycle.

Complementing Docker, the Apache HTTP Server, commonly referred to as Apache, stands as a stalwart in the realm of web hosting. Renowned for its reliability, extensibility, and modular architecture, Apache has been a cornerstone of the web since its inception. It continues to power a significant portion of the world's websites, handling diverse workloads with stability and efficiency. With features such as virtual hosting, SSL/TLS support, and URL redirection, Apache provides a versatile and customizable platform for hosting web applications.

The convergence of Docker and Apache offers a compelling solution to the challenges faced by modern web developers. By leveraging Docker's containerization capabilities and Apache's robust web server functionalities, organizations can achieve a harmonious balance between consistency, scalability, and reliability in their web deployment strategies.

##### Introduction to the tools used:

1. **Apache HTTP Server-**

##### Overview:

Apache, an open-source web server, is renowned for its reliability, stability, and extensive feature set. It plays a pivotal role in serving static and dynamic content over the World Wide Web.

##### Key Features:

Modular Architecture: Enables users to extend and customize functionality through modules, enhancing flexibility.

##### Virtual Hosting:

Supports hosting multiple websites on a single server, optimizing resource utilization. SSL/TLS Support: Ensures secure communication by providing robust support for encryption.

##### Community Support:

Apache benefits from a vibrant and extensive community of developers and users, ensuring continuous contributions and updates.

##### Key Benefits:

Reliability: Established reputation for stable and reliable performance, making it a trusted choice for critical web applications.

Extensibility: Modular architecture allows users to tailor the server to specific application requirements without sacrificing performance.

Community Support: Active community ensures ongoing development, bug fixes, and access to a wealth of resources.

##### Docker

Docker is a platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient units that encapsulate an application and its dependencies. Docker provides a consistent environment across different development, testing, and deployment stages, allowing applications to run reliably in various computing environments. Key components and concepts of Docker include:

* + **Docker Image:** An image is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools.
  + **Container:** A container is an instance of a Docker image. It runs in isolation from the host system and other containers, ensuring consistency and preventing conflicts between applications and their dependencies.
  + **Dockerfile:** A Dockerfile is a script that contains instructions for building a Docker image. It defines the base image, sets up the environment, installs dependencies, and configures the application.
  + **Docker Hub:** Docker Hub is a cloud-based registry service that allows users to share and distribute Docker images. It provides a centralized repository for Docker images, both official and user-created.
  + **Docker Compose:** Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file to configure the application's services, networks, and volumes, allowing for easy orchestration of complex applications.
  + **Container Orchestration (e.g., Kubernetes):** Docker can be integrated with container orchestration platforms like Kubernetes to manage and scale containers in a clustered environment. Orchestration tools handle deployment, scaling, and load balancing of containers.

##### Benefits of Docker include:

* + **Portability:** Docker containers run consistently across different environments, from a developer's laptop to a production server, ensuring that the application behaves the same way everywhere.
  + **Isolation:** Containers provide process isolation, enabling multiple applications to run independently on the same host without interfering with each other.
  + **Resource Efficiency:** Containers share the host OS kernel, making them lightweight and resource-efficient compared to traditional virtualization.

##### Architecture of Pipeline:

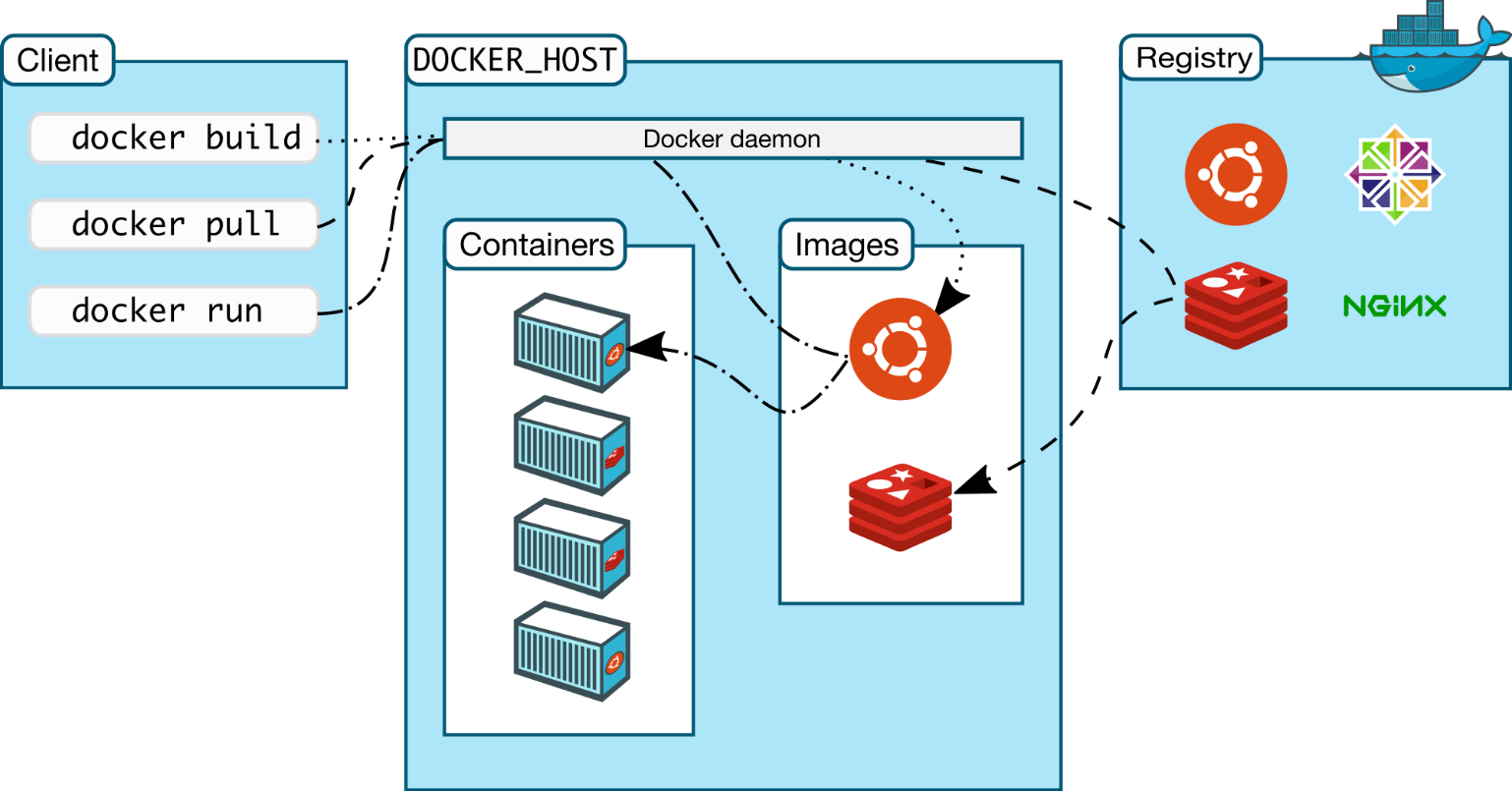


Fig.1 Pipeline

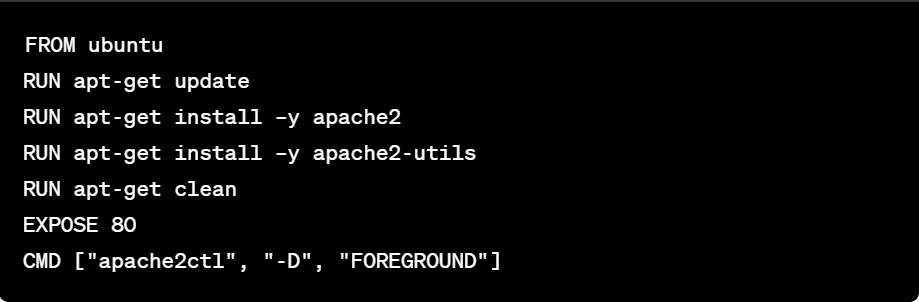
# Implementation:

#### Step 1: Building the Docker Image

To create a Docker image for our Apache web server, follow these steps:

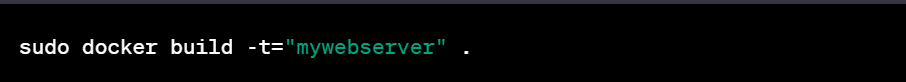
#### 1.1 Dockerfile Creation

Use a text editor, such as vim, to create a Dockerfile with the following content:



#### Step 2: Building the Docker Image

Execute the following command to build the Docker image:

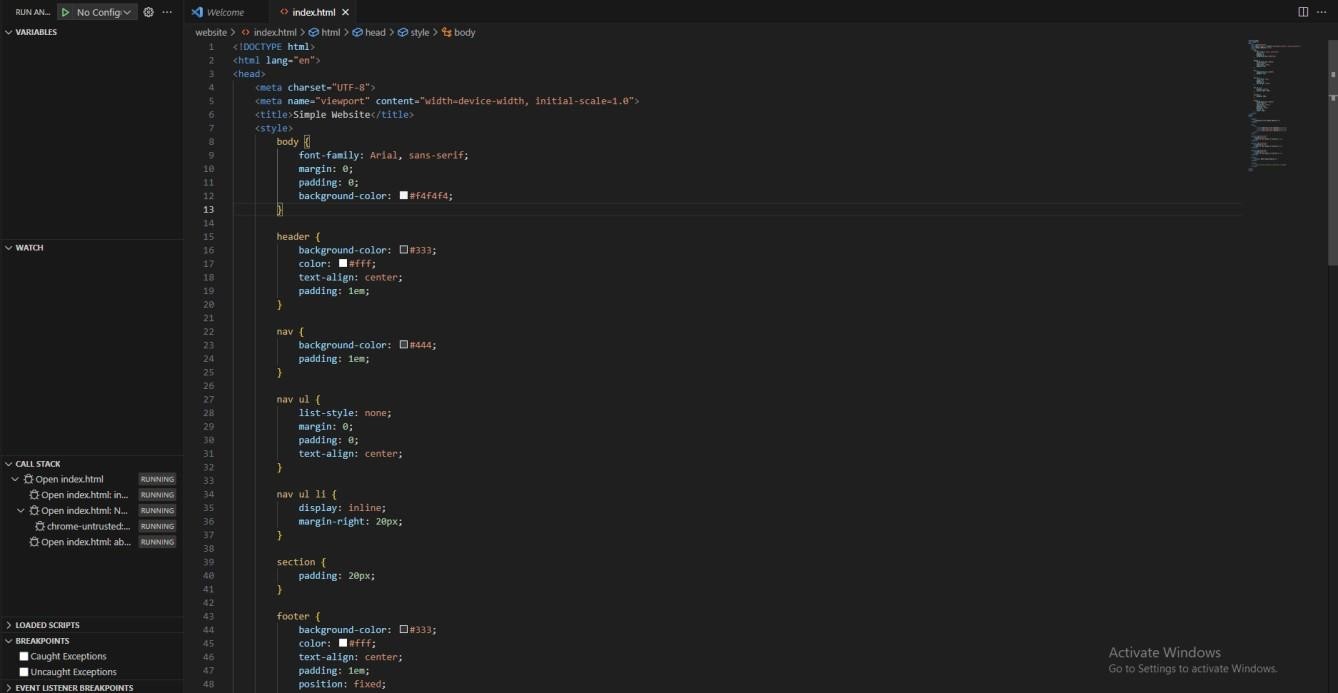


#### Step 3: Creating a Docker Container

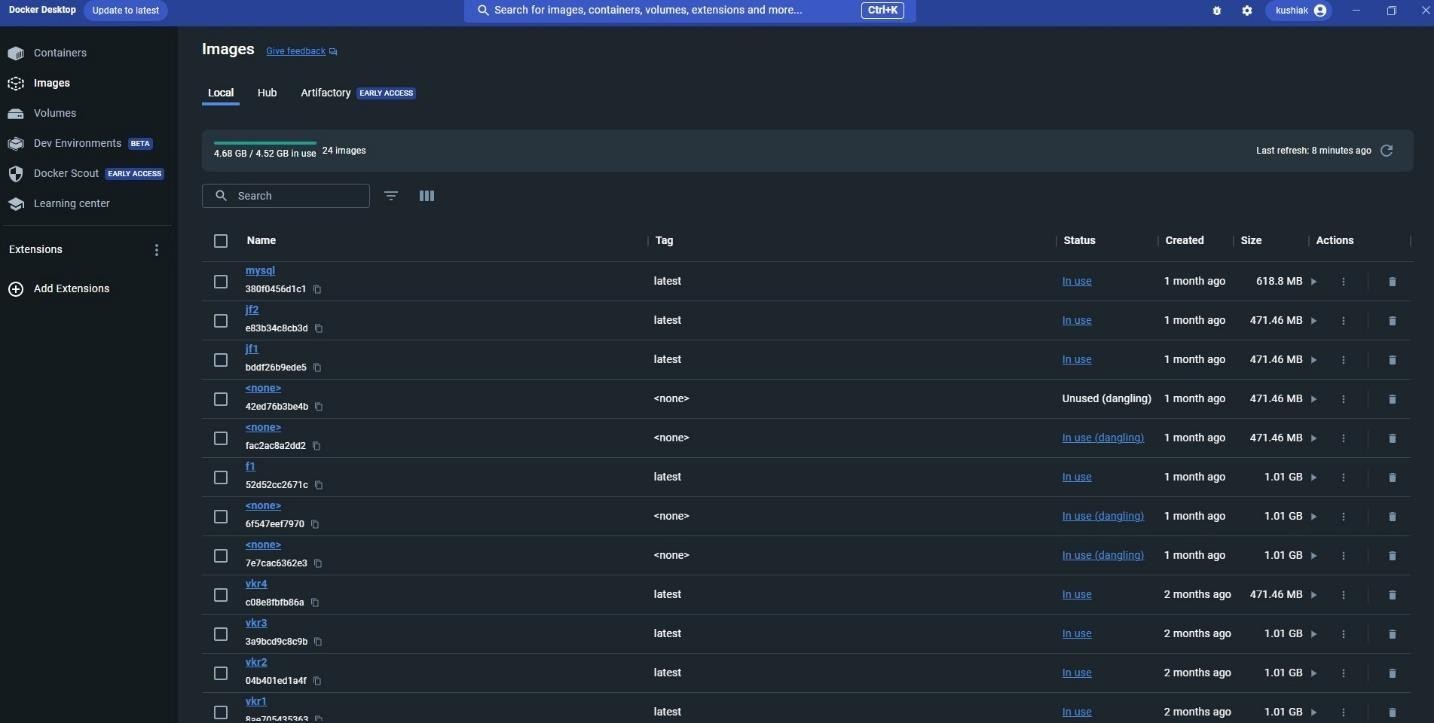
Now that the Docker image is built, proceed to create a container with the following command:



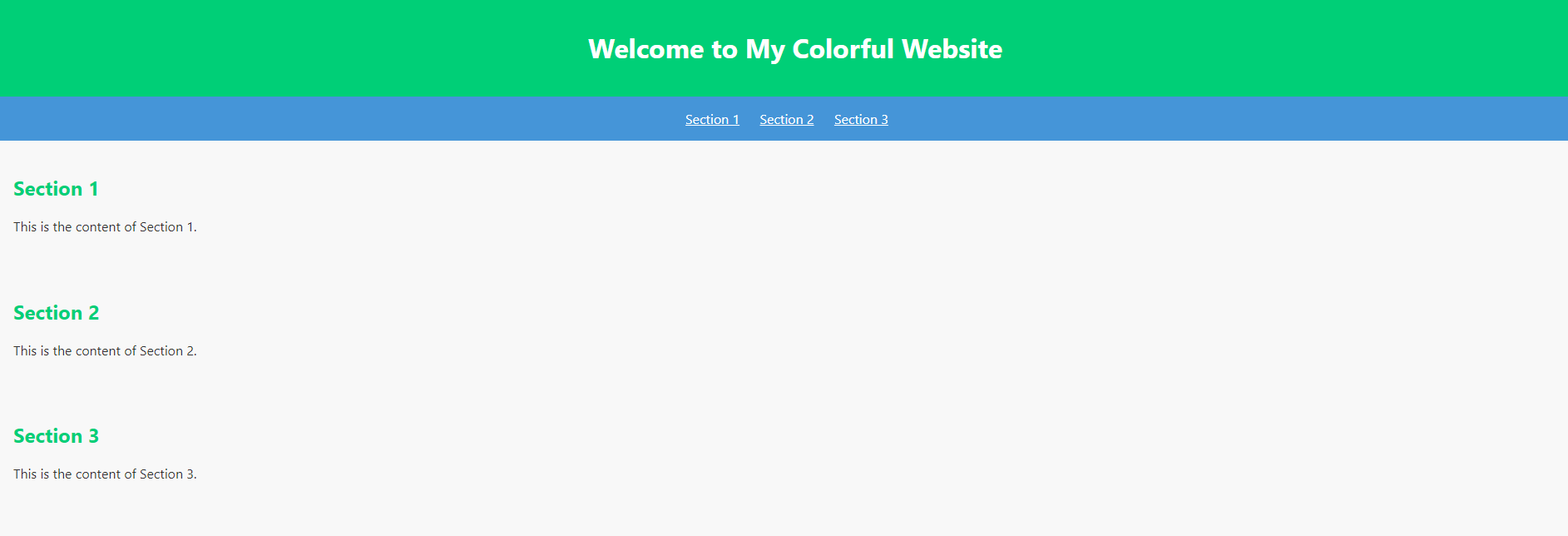
**OUTPUT:**



The above screenshot shows website code executed in vscode



The above screenshot shows the docker login successful and containers



The above screenshot shows the local host executed using docker

### Conclusion:

In conclusion, the process of building and deploying an Apache web server using Docker offers a streamlined and efficient solution for web application hosting. The integration of Docker provides a containerized environment, ensuring consistency and portability across various deployment scenarios.

##### Dockerfile Creation:

The Dockerfile serves as a blueprint for the Docker image. In our case, it begins by selecting the Ubuntu base image, updating packages, installing Apache2, its utility packages, and cleaning unnecessary files. The Dockerfile concludes by exposing port 80 and starting Apache2 in the foreground. This sequence of commands ensures the creation of a Docker image with a functional Apache web server.

##### Image Building:

Executing the `docker build` command with the appropriate tagging (`-t`) results in the creation of a Docker image named "mywebserver." This process encapsulates the configuration and dependencies defined in the Dockerfile, providing a reproducible and versioned image ready for deployment.

##### Container Deployment:

The final step involves creating a Docker container from the built image. Using the `docker run` command, we specify that the container should run in the background (`-d`), map port 80 from the container to the host (`-p 80:80`), and utilize the "mywebserver" image. This instantiates an isolated instance of the Apache web server within a Docker container.