**SEPM**

**Experiment 9**

**Aim:**

To learn Dockerfile instructions, build an image for a sample web application using Dockerfile.

**Theory:**

Docker enables efficiency and reduces operational overheads so that any developer, in any development environment, can build stable and reliable applications. Docker provides the ability to package and run an application in a loosely isolated environment called a container.

Isolation and security allow you to run many containers simultaneously on a given host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker Container Lifecycle Management

Images  
 An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image that is based on the ubuntu image but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast when compared to other virtualization technologies.

Docker can build images automatically by reading the instructions from a Docker file. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. This page describes the commands you can use in a Dockerfile.

The instruction is not case-sensitive. However, the convention is for them to be UPPERCASE to distinguish them from arguments more easily.

Docker runs instructions in a Dockerfile in order. A Dockerfile must begin with a FROM instruction. This may be after parser directives, comments, and globally scoped ARGs. The FROM instruction specifies the Parent Image from which you are building. FROM may only be preceded by one or more ARG instructions, which declare arguments that are used in FROM lines in the Dockerfile.

Docker treats lines that begin with # as a comment unless the line is a valid parser directive. A # marker anywhere else in a line is treated as an argument.

**Implementation:**

**Pre-requisites:** **Simple Flask Application.**

App.py

*from flask import Flask, jsonify*

*app = Flask(\_\_name\_\_)*

*@app.route('/')*

*def hello\_world():*

*return jsonify({*

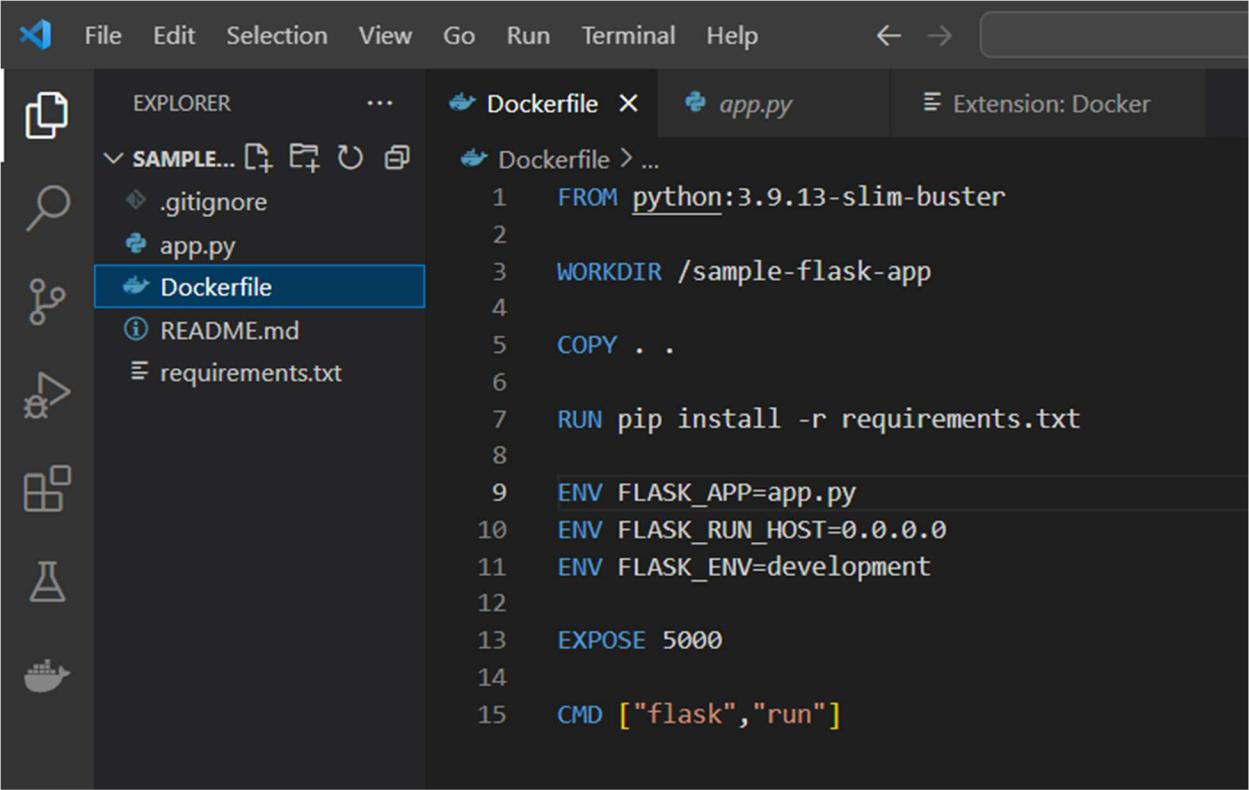
*"message": "Hello This Is Heramb"*

*})*

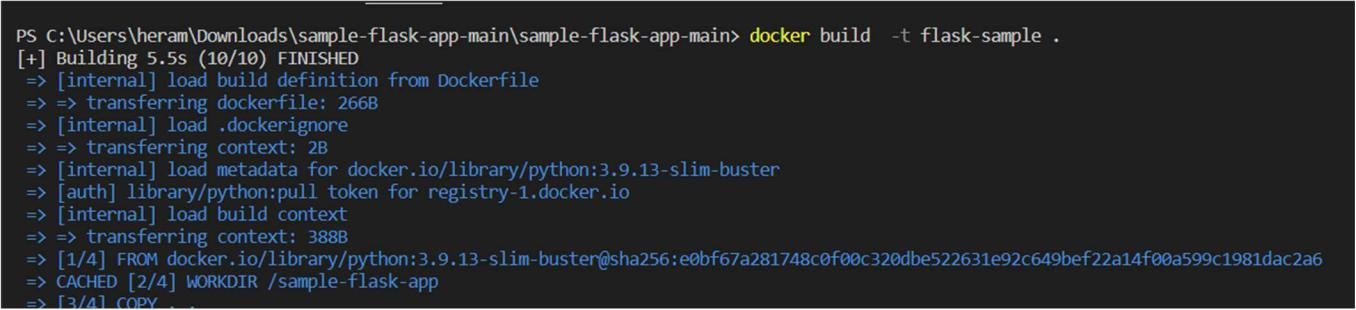
*if \_\_name\_\_ == '\_\_main\_\_':*

*app.run()*

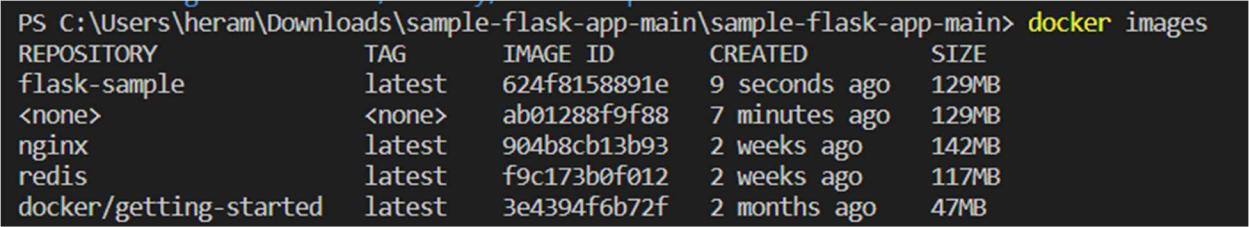
Creating A Docker File And Adding It In the Same Folder



Creating Image File for the Flask Application:



Checking If the Image File is created or not:



Running the container:



**Final Output:**

{

“Message”: “Hello this is Yash!”

}

**Conclusion:**

We have learned Dockerfile instructions and their benefits. We also studied and installed Docker, along with building an image for a sample web application using Dockerfile