Assignment - DevOps:

Version- Draft

Prepared by

Shreyank Srinivasan Harisha

|  |  |  |
| --- | --- | --- |
| Shreyank Srinivasan Harisha | 1RVU22CSE152 | Shreyanksh.btech22@rvu.edu.in |

|  |  |
| --- | --- |
| Instructor: | *CVSN Reddy* |
| Course: | Agile Software Engineering |
| Lab Section: | *BTech 3rd Semester ‘A’* |
| Teaching Assistant: | *Divya M* |
| Date: | 31/10/2023 |

**Contents**

Contents 1

Revisions 2

1 What is meant by DevOps? Write about following DevOps tools 1

1.1Maven 1

1.2 GIT 1

1.3 GITHUB 1

1.4 GIT Actions 1

1.5 PYUNIT 2

1.6 Selenium and RPA 2

1.7 JENKINS 2

1.8 Nagios, Puppet, Ansible and CHEF 2

1.9GITCO-PILOT 2

2 Explain following: 3

2.1 VM vs. Docker 3

2.2 Docker, Docker hub, Docker file, Docker Image 3

2.3 Difference between Docker and Kubernetes 4

2.4 DevOps pipeline 5

2.5 What is CI/CB/CT/CD 4

2.6 What are the DevOps tools available in MS-Azure and AWS 5

3 Demonstrating PyUnit and Pylint with Python 6

3.1 INTRODUCTION TO PYUNIT 6

3.2 DEMONSTRATING PYUNIT 7

3.3 INTRODUCTION TO PYLINT 8

3.4 DEMONSTRATING PYLINT 8

3.5 CONCLUSION 8

4 Demonstrating System Test Script Automation using RPA UIPath 11

4.1 INTRODUCTION TO PYUNIT 11

4.2 DEMONSTRATING PYUNIT 11

4.3 INTRODUCTION TO PYLINT 12

4.4 DEMONSTRATING PYLINT 11

4.3 CONCLUSION 12

5 Demonstrating System Test Script Automation using RPA UIPath 11

4.1 Performance Requirements 11

4.2 Safety and Security Requirements 11

4.3 Software Quality Attributes 12

6 Demonstrating System Test Script Automation using RPA UIPath 11

4.1 Performance Requirements 11

4.2 Safety and Security Requirements 11

4.3 Software Quality Attributes 12

7 Building a Docker Image and Pushing it to Docker Hub 13

7.1 INTRODUCTION 11

7.2 DOCKER FILE 11

7.3 BUILDING A DOCKER IMAGE 11

7.4 PREPARING THE DOCKER FILE 12

7.5 BUILDING THE DOCKER IMAGE USING DOCKER BUILD 11

7.6 TAGGING THE IMAGE 11

7.7 PUSHING THE IMAGE TO DOCKER HUB 12

7.8 CONCLUSION 12

**Revision**

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| Draft | Shreyank Srinivasan Harisha | N/A | 31/10/2023 |

**1.** **What is meant by DevOps? Write about following DevOps tools.**

DevOps, a portmanteau of 'development' and 'operations,' represents a methodology that encourages cooperation between development and operations teams with the goal of optimizing software development, delivery, and deployment processes.

1.1) **Maven:** This tool serves multiple purposes such as project building, dependency management, and documentation generation, as well as the capability to compress files.

1.2) **GIT:** As a local version control tool, GIT provides a means for tracking and managing changes in code locally.

1.3) **GitHub:** GitHub, on the other hand, extends GIT into the cloud environment, making it accessible from remote locations.

1.4) **Git Actions:** Git Actions functions as an orchestration tool, specifically designed for Continuous Integration and Continuous Deployment (CI/CD) purposes. It automates tasks like building, testing, and deploying software on GitHub.

1.5) **PYUNIT:** PYUNIT is a valuable tool in the DevOps toolkit, primarily used for unit testing and integration testing.

1.6) **Selenium and RPA:** These tools play a pivotal role in DevOps by facilitating system testing and automating the creation of system test scripts.

1.7) **Jenkins:** Jenkins, an orchestration tool, functions as an open-source server that enables developers to automate the entire software development process, including building, testing, and deployment.

1.8) **Nagios:** Nagios is essential for monitoring all the other DevOps tools, ensuring their optimal performance and reliability.

1.8) **Puppet, Ansible, and Chef:** These tools are instrumental in managing software updates and configuration management.

1.9) **GITCO-PILOT:** GITCO-PILOT, an AI-powered pair programmer, assists in accelerating code writing with increased efficiency.

**2.** **Explain following:**

2.1) **VM vs. Docker:**

* VM: It's like a computer within a computer, running different "virtual houses" (operating systems) on your real computer.
* Docker: A container that combines VM, OS, software, and libraries for deployment.

2.2) **Docker Terms:**

* Docker Image: A self-contained package with everything to run an app, including code, runtime, and tools.
* Docker Container: An isolated runtime instance of a Docker image.
* Docker File: Instructions to build a Docker image.
* Docker Hub: Cloud storage for Docker images.

2.3) **Docker vs. Kubernetes:** Docker is a container, while Kubernetes is a container orchestration tool.

2.4) **DevOps Pipeline:** A series of stages for building, testing, and deploying software.

2.5) **CI/CT/CD:**

* CI: Integrating code changes frequently and automating testing.
* CT: Running tests throughout the development process.
* CD: Automatically deploying changes to production.
* Continuous Delivery: Ensuring software is always release-ready.

2.6) **DevOps Tools in Azure and AWS:**

**Azure:**

* Azure DevOps
* DevTest Labs
* Kubernetes Service
* Azure Functions
* Azure Monitor and Application Insights

**AWS:**

* CodePipeline
* CodeBuild
* CodeDeploy
* CloudFormation
* CloudWatch
* Lambda

**3.1.** **Introduction to PyUnit (Unit test)**

PyUnit, also recognized as unit test, is an intrinsic Python library designed for crafting and executing unit tests. Unit tests play a crucial role in guaranteeing the dependability of your code by validating that individual components operate as intended. PyUnit offers a framework for delineating test cases, structuring them into test suites, and executing the tests.

**3.2. Demonstrating PyUnit**

First, install Pytest by executing the command "**pip install pytest**" and confirm the installation by running "**pytest --version**"

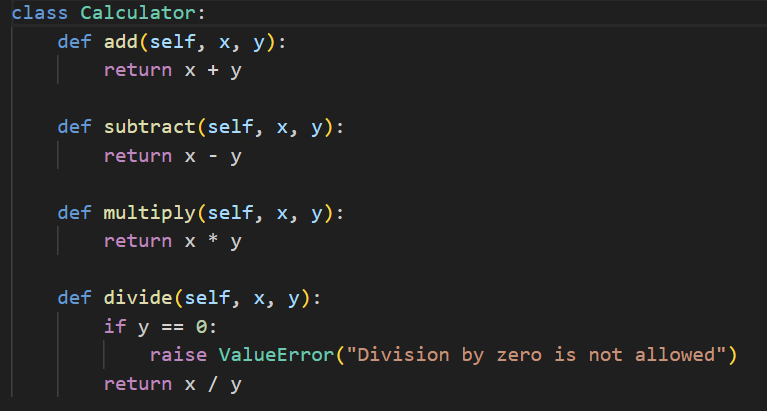
Next, let's assess a basic Python script using Pytest. Write the program in a text editor or Notepad and save it with the ".py" extension. The provided code includes two files: one containing a calculator program and another with test cases for the calculator program.

**FCalculator.py:**

* This script defines a class named Calculator encompassing four fundamental mathematical operations:

1. add: Adds two numbers.
2. subtract: Subtracts one number from another.
3. multiply: Multiplies two numbers.
4. divide: Divides one number by another, with error handling to prevent division by zero.

* These operations are implemented as methods within the Calculator class.
* You can use Pytest to test the functionality of the Calculator class and ensure it performs as expected.

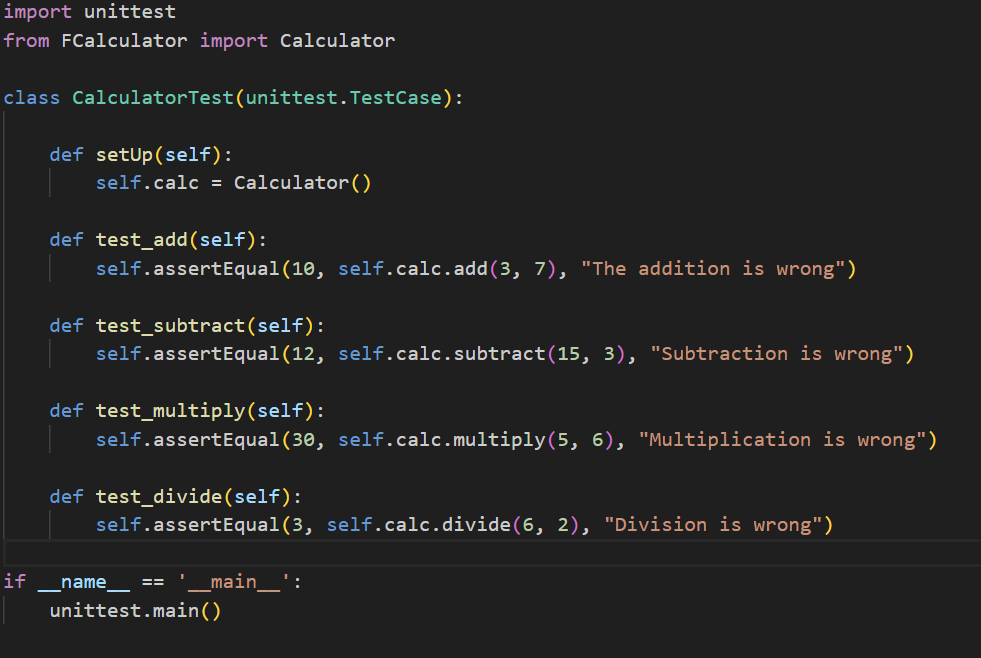


**test\_calculator.py:**

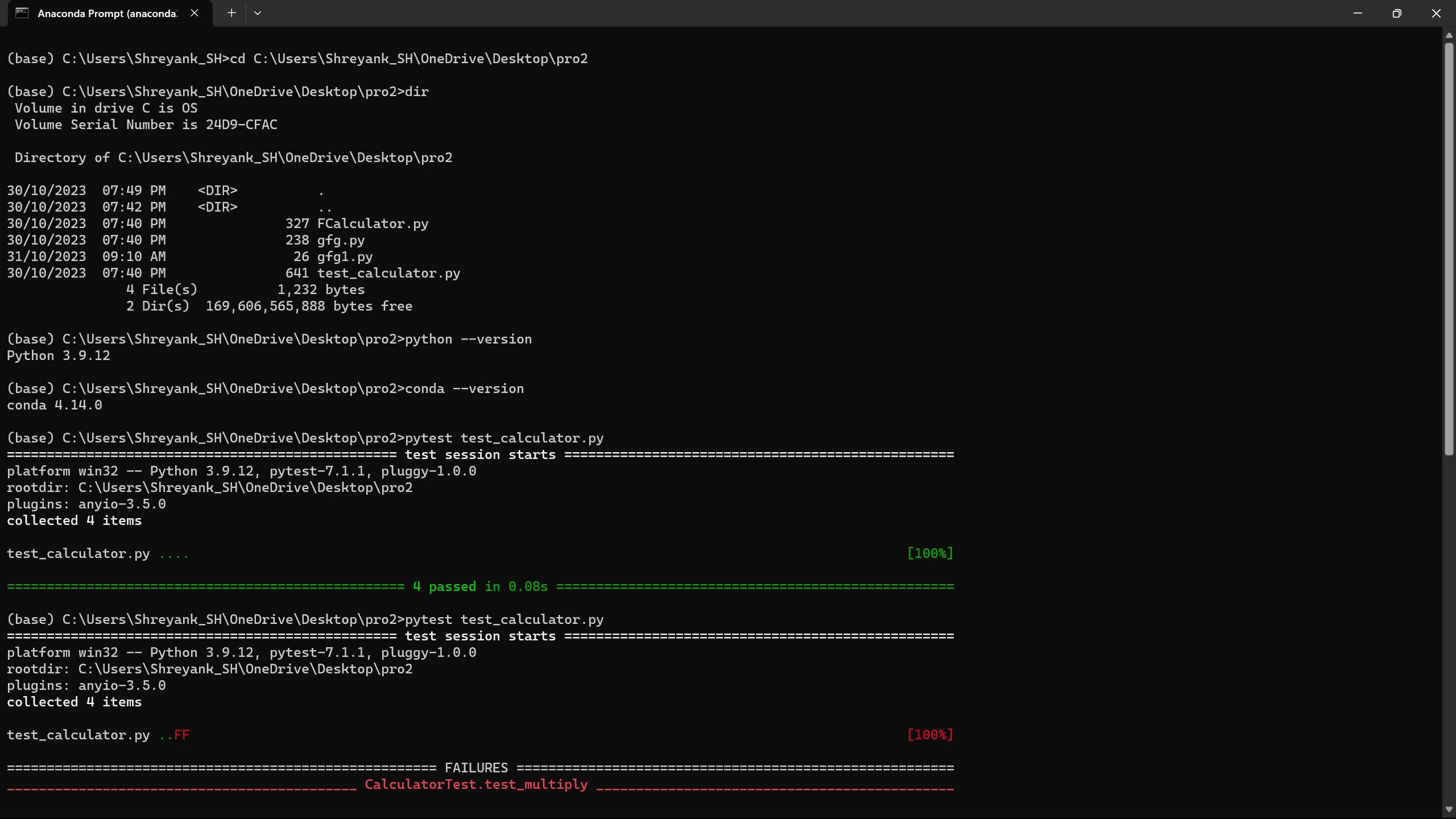
* The setUp method is utilized to create an instance of the Calculator class before running each test.
* There are four distinct test methods:

1. test\_add: Evaluates the add method of the Calculator class.
2. test\_subtract: Assesses the subtract method of the Calculator class.
3. test\_multiply: Examines the multiply method of the Calculator class.
4. test\_divide: Inspects the divide method of the Calculator class.

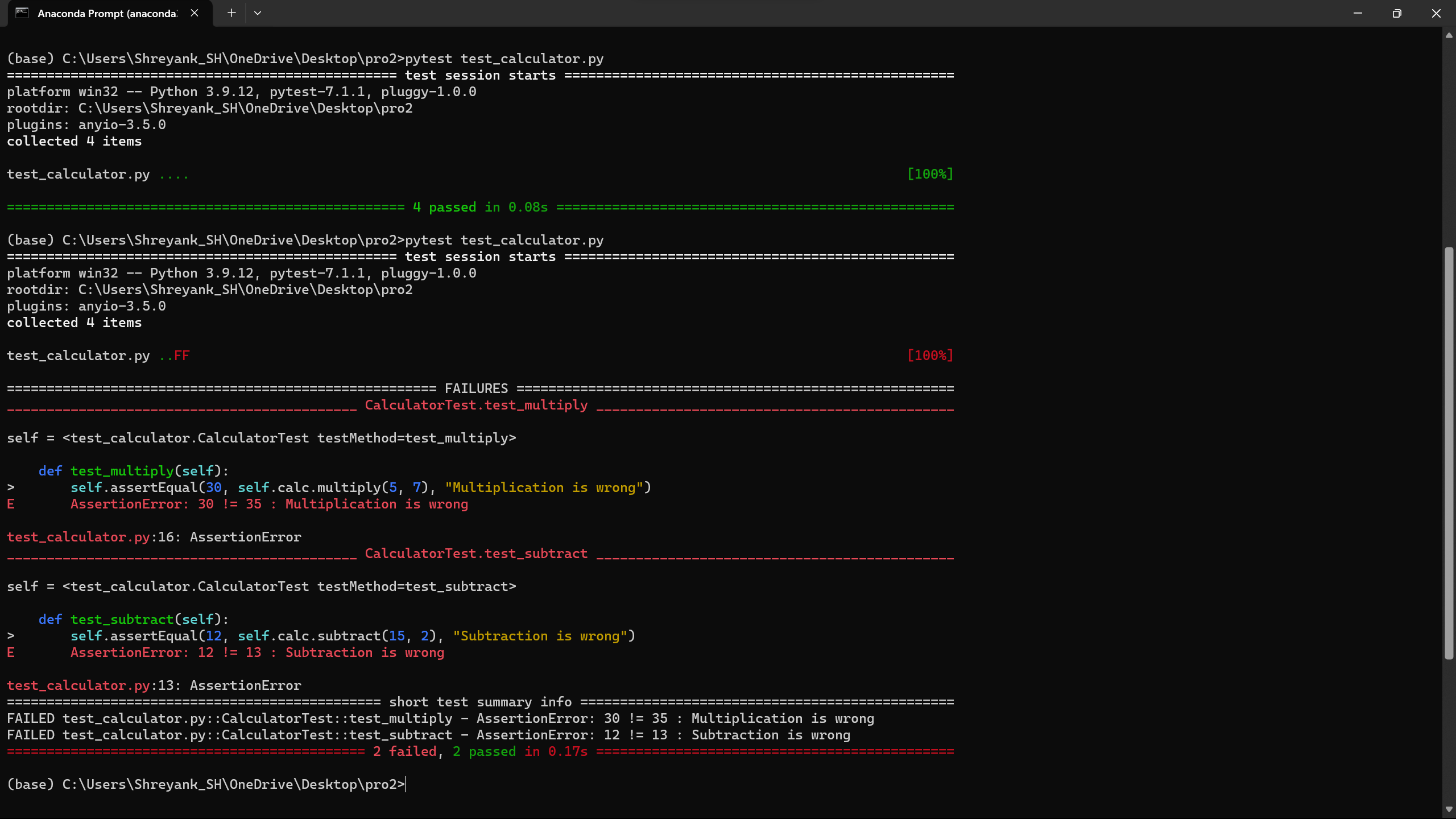
* Each test method employs the self.assertEqual assertion to compare the actual result with the expected result for a specific operation. If the assertion fails, it triggers an error message, indicating a problem in the code.
* These tests are crucial for verifying that the Calculator class functions as intended, ensuring the correctness of its mathematical operations.



To run these tests with the `pytest` framework, execute the `pytest test\_calculator.py` script. It will run each test method and report whether they pass or fail. If a test fails, the error message specified in the `self.assertEqual` statement will be displayed.



Since all the 4 operations have executed successfully, all the 4 test cases have passed successfully.



On changing the values, here, 2 test cases have passed and 2 have failed and the ‘self.assertEqual()’ prints the error message.

Thus, using **pytest** offers a concise and user-friendly way to write and run unit tests in Python. It simplifies the testing process, improves code readability, and provides a comprehensive testing ecosystem for Python projects. It's a valuable tool for writing and maintaining robust test suites.

**3.3. Introduction to Pylint**

Pylint serves as a static code analysis tool that evaluates Python code against a predefined set of coding standards. It furnishes feedback on code quality, potential concerns, and deviations from coding style, thereby assisting developers in crafting code that is more organized and easier to maintain.

**3.4. Demonstrating Pylint**

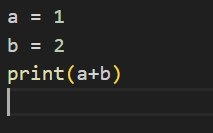
To ensure that Anaconda and Python are correctly installed, please execute the following commands:

Check Anaconda's version using "**conda --version**"

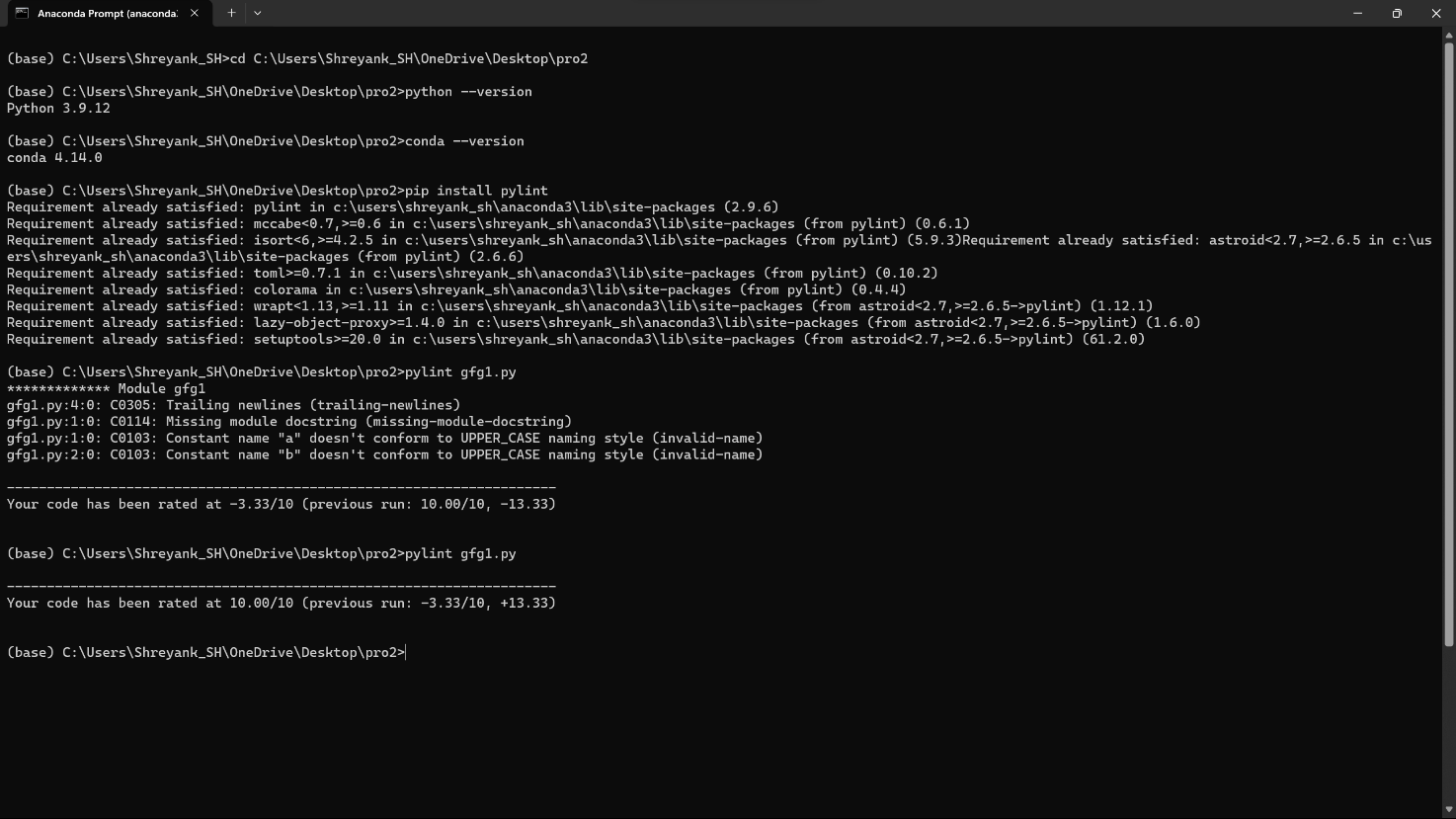
Verify the Python version with "**python --version**"

Subsequently, install Pylint by executing the command "pip install pylint," and confirm the installation by running "pylint --version."

Now, we can proceed to analyze a basic Python script using Pylint. Begin by writing the program in a text editor or Notepad and save it with the ".py" extension. This script represents a simple addition of two numbers.



To analyze this script using Pylint, please run the following command. To execute this code with Pylint, utilize the command "pylint gfg1.py" in either the command prompt or Anaconda prompt window.

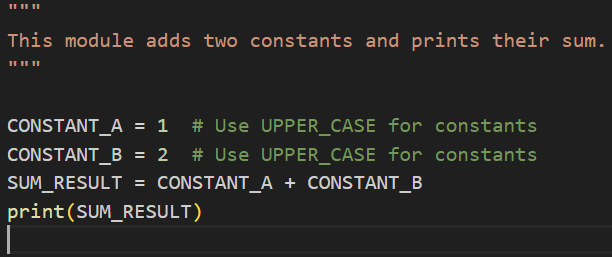


Pylint will evaluate the code and provide a report detailing aspects of code quality, possible issues, and deviations from coding style standards.

In the provided code, it currently has a rating of 0.00/10, which is relatively low. This score does not necessarily mean that the code is incorrect; rather, it reflects how easily another programmer can understand it. To enhance the code, you can incorporate the suggested improvements and continue to iterate until you achieve a rating of 10/10.

Here's a method to follow:

1. Present the code to ChatGPT.
2. Request ChatGPT to create an improved version of the code with a rating of 10/10.
3. Save the newly generated code as a .py file.
4. Re-run Pylint on this improved code.
5. Repeat this process until you attain a perfect score of 10.00/10.
6. This iterative process will help refine the code until it adheres to high coding standards.



Now, the static code analysis using Pylint is successfully completed as a 10.00/10 score is achieved. The major changes made is adding comments and ensuring the UPPER\_CASE naming style of variables.

**3.5. Conclusion**

In this report, we showcased the utilization of PyUnit for unit testing and Pylint for code analysis in Python. Both of these tools are indispensable for upholding code quality and guaranteeing the dependability of your Python applications. PyUnit, through unit testing, aids in confirming the accuracy of your code, whereas Pylint serves to identify and rectify code quality concerns. The integration of these tools into your Python development workflow is imperative for the creation of sturdy and sustainable software.

**4.1. Introduction**

**Background:** In the contemporary technology-driven world, software testing assumes a pivotal role in assuring the quality and reliability of applications. As applications grow in complexity, the imperative for automation in testing becomes increasingly apparent. Robotic Process Automation (RPA) tools, such as UIPath, provide a potent means to automate test scripts, guaranteeing swift and effective testing.

**Purpose:** The objective of this report is to present the potential of UIPath in automating system test scripts and illustrate how it can streamline the testing process, thereby reducing human error and enhancing overall productivity.

**4.2. UIPath: An Overview**

## What is UIPath?

UIPath stands out as a top-tier Robotic Process Automation (RPA) platform, empowering organizations to automate repetitive, rule-based tasks through software robots, often referred to as "bots." These bots possess the capability to interact with a diverse range of applications, mimic human actions, and execute tasks with exceptional precision.

**4.3. System Test Script Automation**

**The Importance of Test Automation:** Automating system test scripts is paramount in addressing the escalating complexity and abundance of test scenarios in contemporary software applications. Manual testing, given its time-consuming and error-prone nature, is ill-suited for iterative and agile development methodologies.

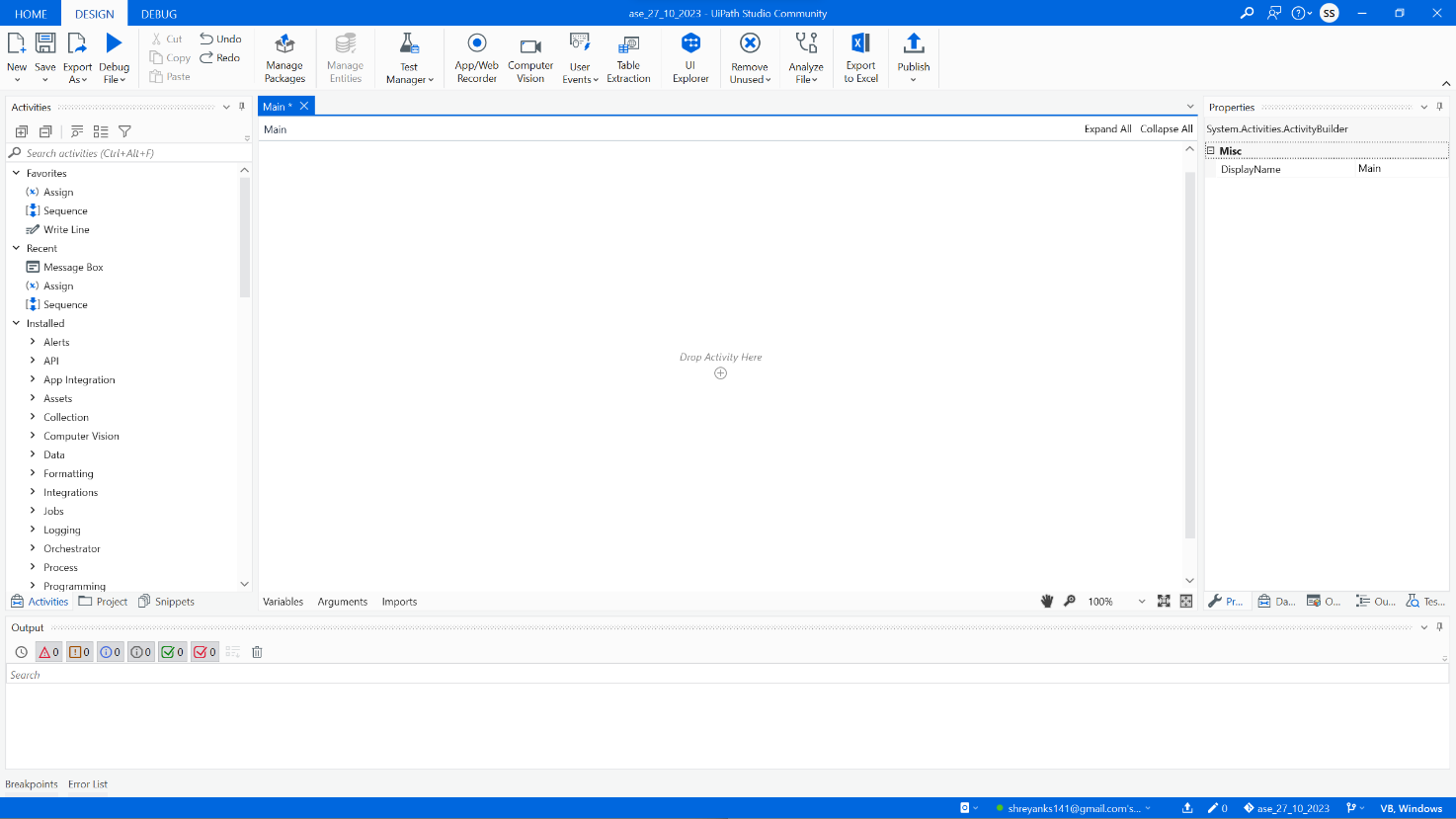
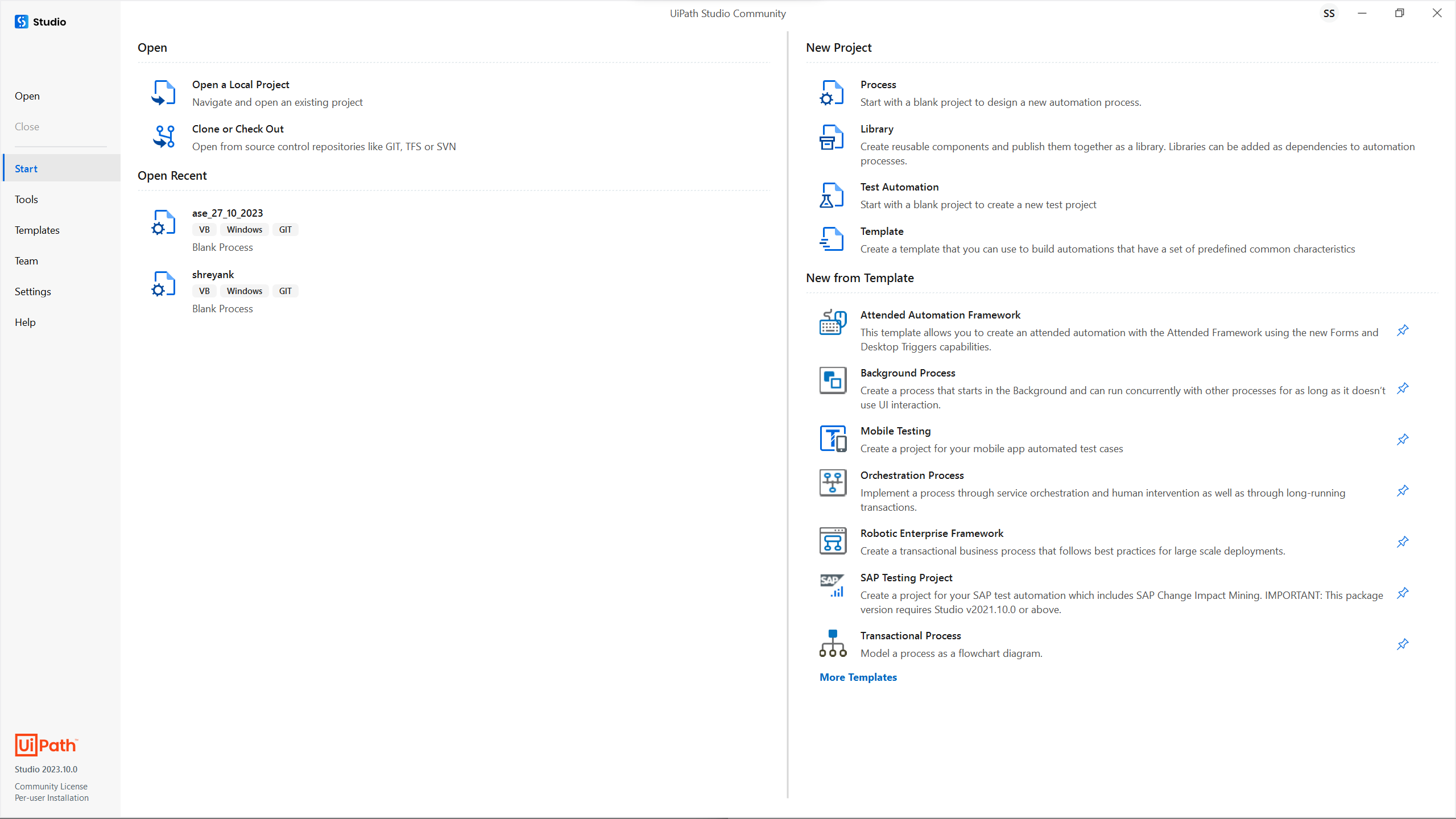
**Role of UIPath in Test Automation**: UIPath effectively tackles these challenges by furnishing a versatile platform for crafting and overseeing automated test cases. It boasts an extensive array of activities, champions automation best practices, and offers a user-friendly interface for script development.

**4.4. Demonstrations**

## Environment Setup

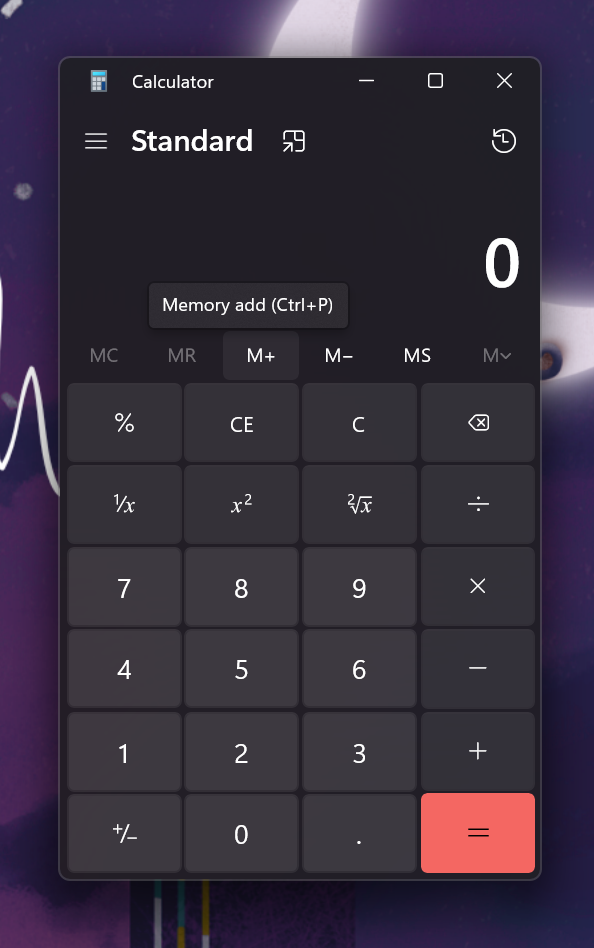
To showcase UIPath in operation, we initiated the process by establishing a test environment complete with the requisite software and tools. This involved installing UIPath Studio and configuring it to facilitate test automation.

**Step 1:** Launch UiPath Studio: Open UiPath Studio on your computer.

**Step 2:** Begin a New Process: Click on "New Process" to initiate a new automation project. Provide a descriptive name for your project and specify its location.

## Creating a Test Case:

Subsequently, we devised a test case encompassing actions such as navigating a web application, inputting data, and validating the results. Initially, this test case was executed manually. Here, we employed the calculator and manually executed the calculations.



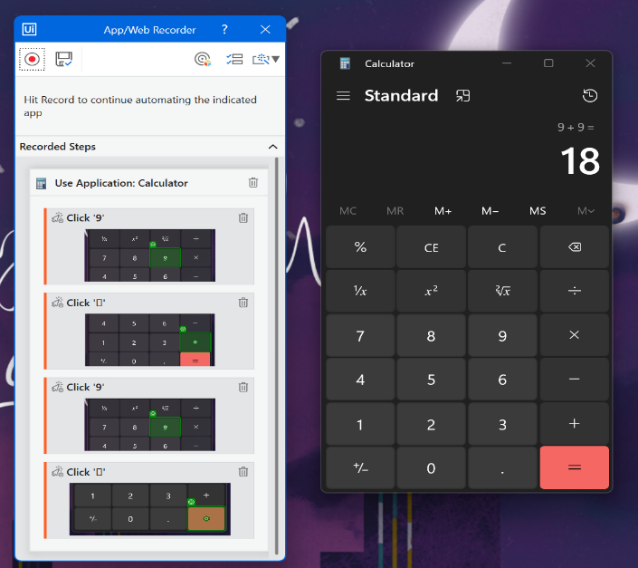
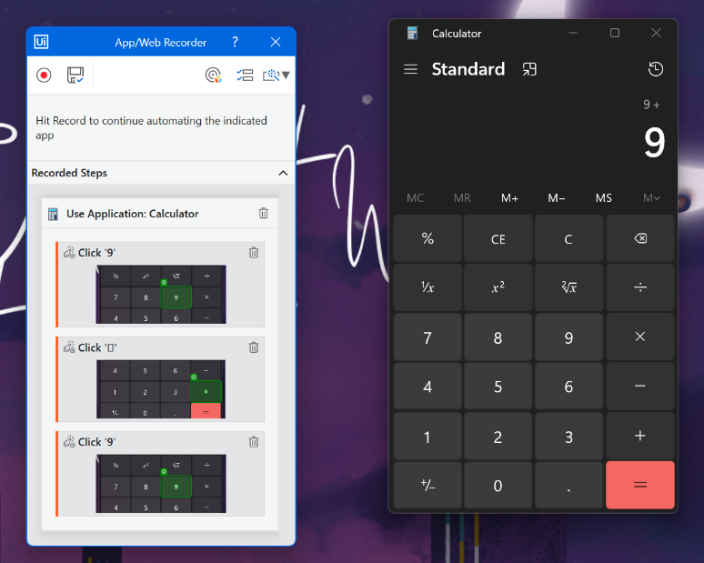
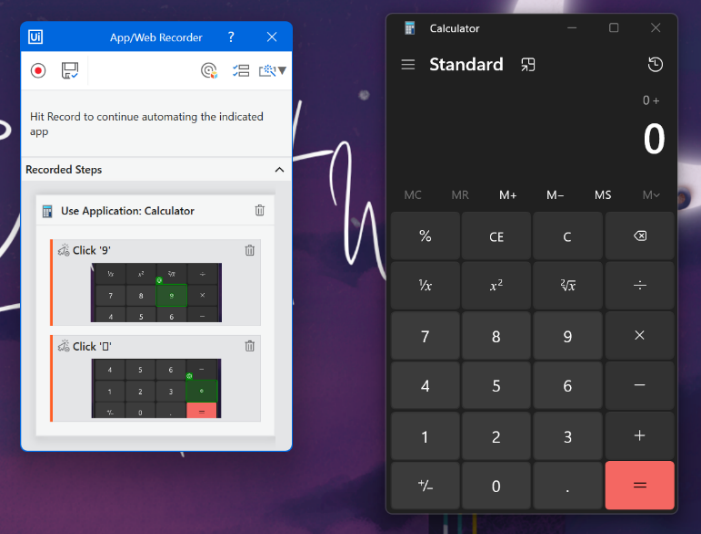
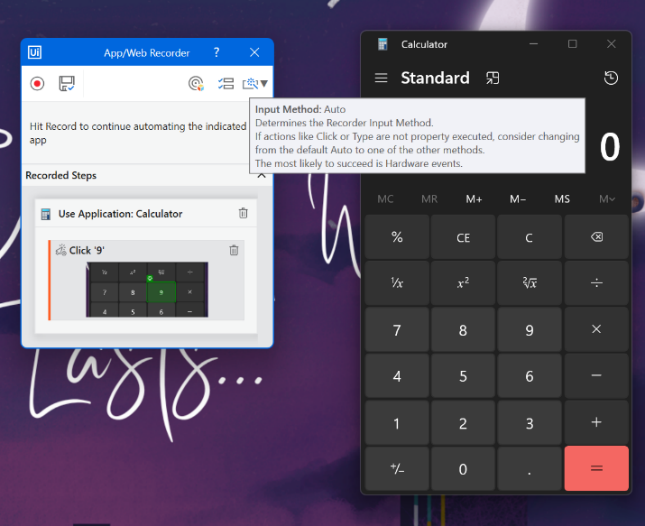
**Step 3:** Launch the Calculator and Prepare:

• Prior to commencing the recording, launch the Windows Calculator application and ensure it is readily accessible on your desktop.

## Automating the Test Script:

Using UIPath Studio, we created an automation script that replicated the steps of the manual test case. The script included activities to interact with the web application, input data, and validate results.

Will give the numbers and add them…



**Step 4:** Initiate Recording:

• Click on the "Record" option in UiPath Studio, opening a list of choices that includes "Basic Recording" and "Desktop Recording."

• Opt for "Desktop Recording" since your intent is to interact with a desktop application, such as the calculator.

**Step 5:** Execute Actions and Record:

• A recording toolbar will appear; click the "Record" button on this toolbar.

• Commence your interaction with the Calculator as you typically would. Input numbers, perform calculations, and carry out any other actions you wish to automate.

• If the calculator prompts you for confirmation, such as using a tick mark or any other action, be sure to include these actions in your recording.

**Step 6:** Pause Recording:

• When you have completed the actions you intend to automate, click on the "Pause" button in the recording toolbar.

**Step 7:** Save the Recording:

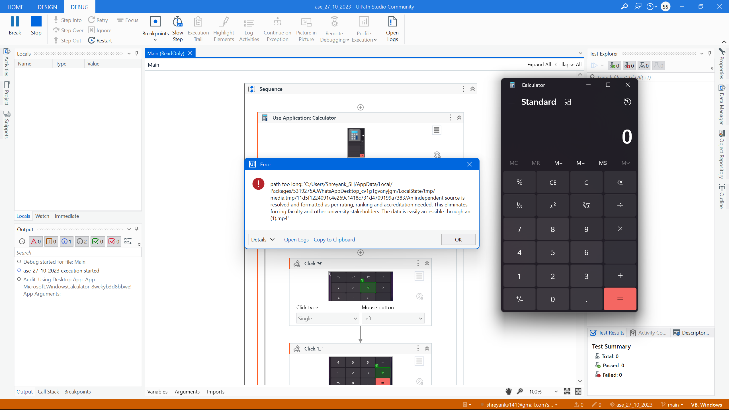
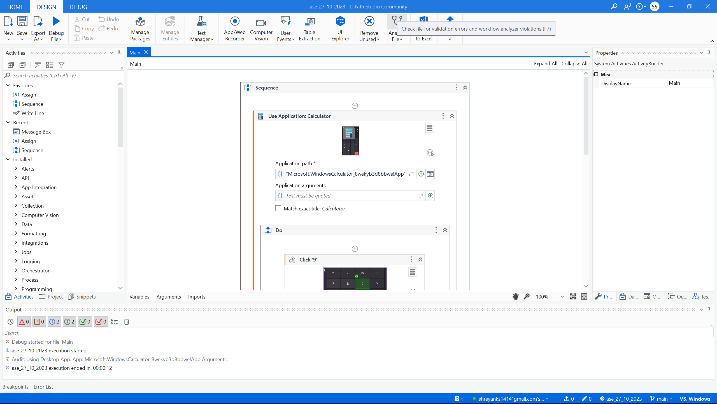
• Upon pausing, you'll receive a prompt to save the recording.

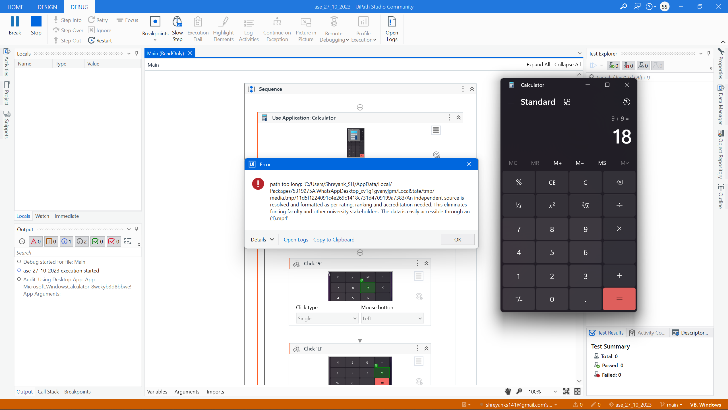
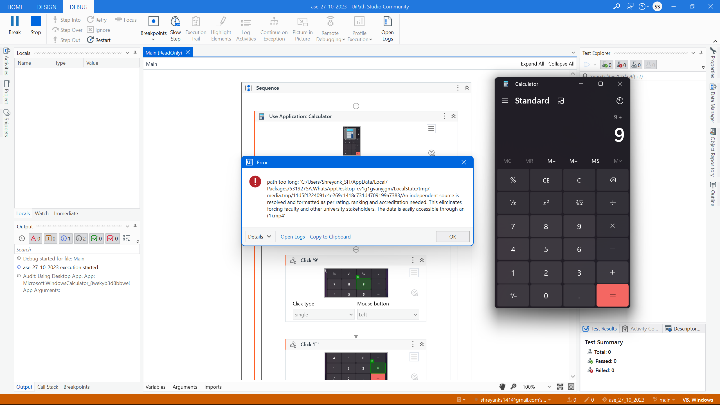
• Provide it with a meaningful name and description.

## Running and Monitoring the Test:

We proceeded to run the automated test script, closely monitoring its progress and witnessing the interaction of UIPath's bots with the application. The execution proved significantly faster compared to manual testing.

Upon inputting all the necessary data, we saved it, followed by the debugging phase. At this point, the robot will commence its tasks. The execution demonstrated a significant speed advantage over manual testing. After providing the requisite data, the next steps involve saving and debugging. Subsequently, the robot will commence its assigned tasks.





**Step 8:** Debug Your Automation:

• After saving the recording, return to UiPath Studio.

• In the top bar, on the left-hand side, locate the "Debug" button and click on it to run your automation.

**Step 9:** Implement Random Test Cases:

• UiPath Studio provides the flexibility to parameterize and introduce randomness into your automation. You can configure test cases with random data by utilizing variables, Excel files, or other data sources to enhance the versatility of your automation.

**4.5. Conclusion**

In summary, the demonstration illustrates how UIPath can optimize the automation of system test scripts, resulting in improved efficiency, accuracy, and a substantial return on investment. Despite encountering certain challenges, the benefits substantially outweigh them, rendering UIPath an invaluable tool for organizations aiming to enhance their testing processes.

**7.1. Introduction**

Docker is an open-source platform that empowers developers to automate the deployment of applications within lightweight, easily transportable containers. Docker Hub, offered by Docker as a cloud-based service, serves as a platform for sharing and disseminating Docker images. This report delineates the essential steps for constructing a Docker image and transferring it to Docker Hub, a pivotal process in contemporary software development.

**7.2. Docker File**

A Docker file is a textual document that encompasses a series of instructions for crafting a Docker container image. Docker, a widely embraced containerization platform, empowers developers to encapsulate both their applications and their associated dependencies within transportable, lightweight containers. These containers can be effortlessly deployed and executed across diverse environments, rendering Docker an indispensable tool for software development and deployment.

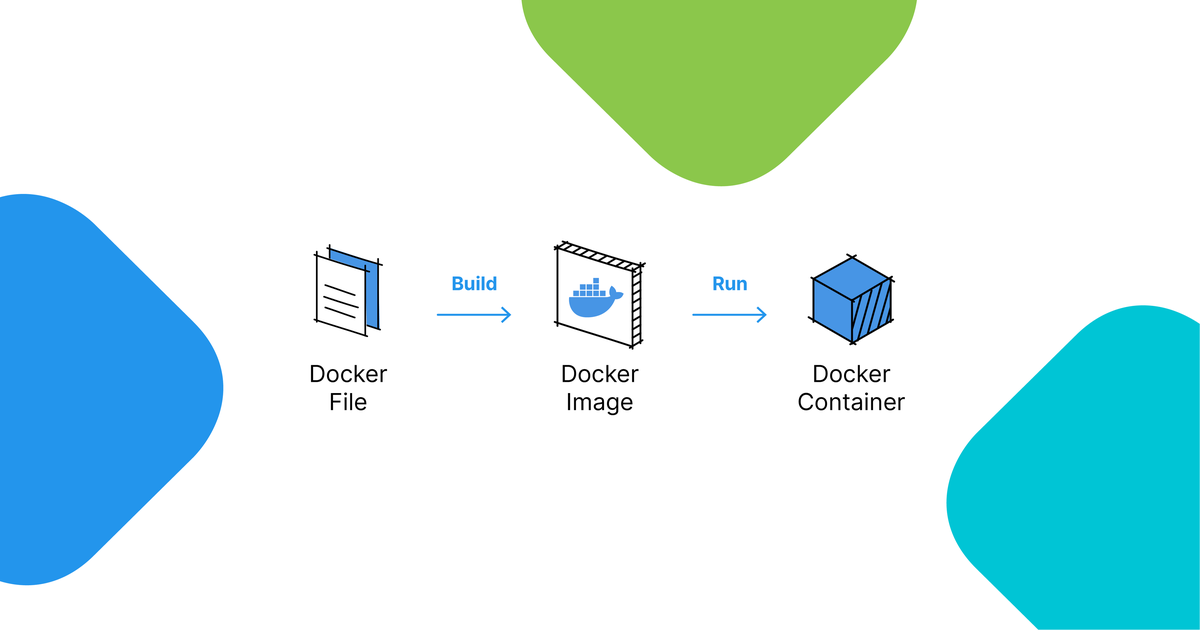


Image credit - [Picture](https://assets-global.website-files.com/622642781cd7e96ac1f66807/630e24fcf9150c8a62a09653_unnamed-1.png)

**7.3. Building a Docker Image**

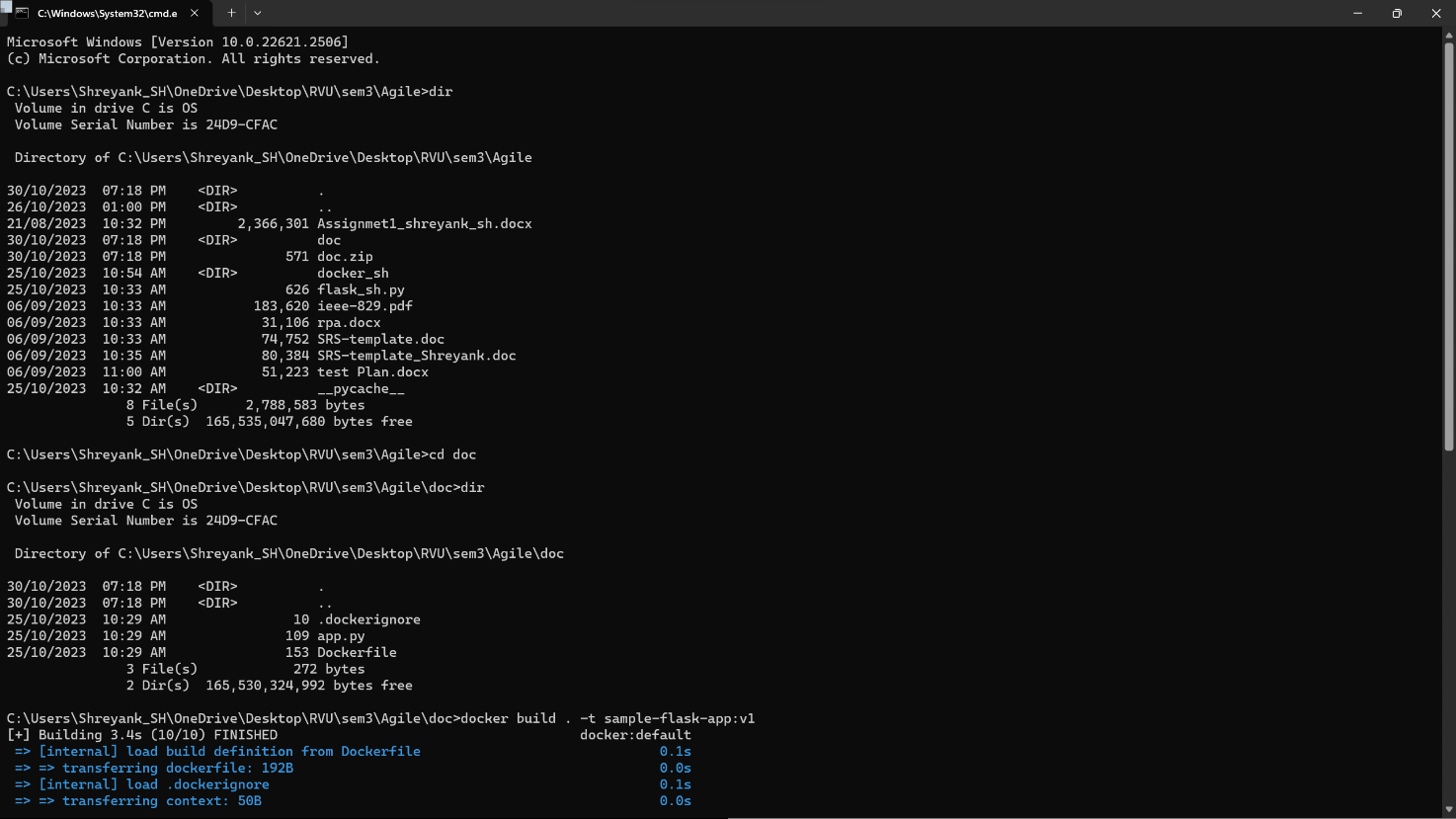
The process of building a Docker image revolves around creating a blueprint for your application and its dependencies. This blueprint is articulated within a file known as a Dockerfile, housing instructions that dictate the precise construction of the image.

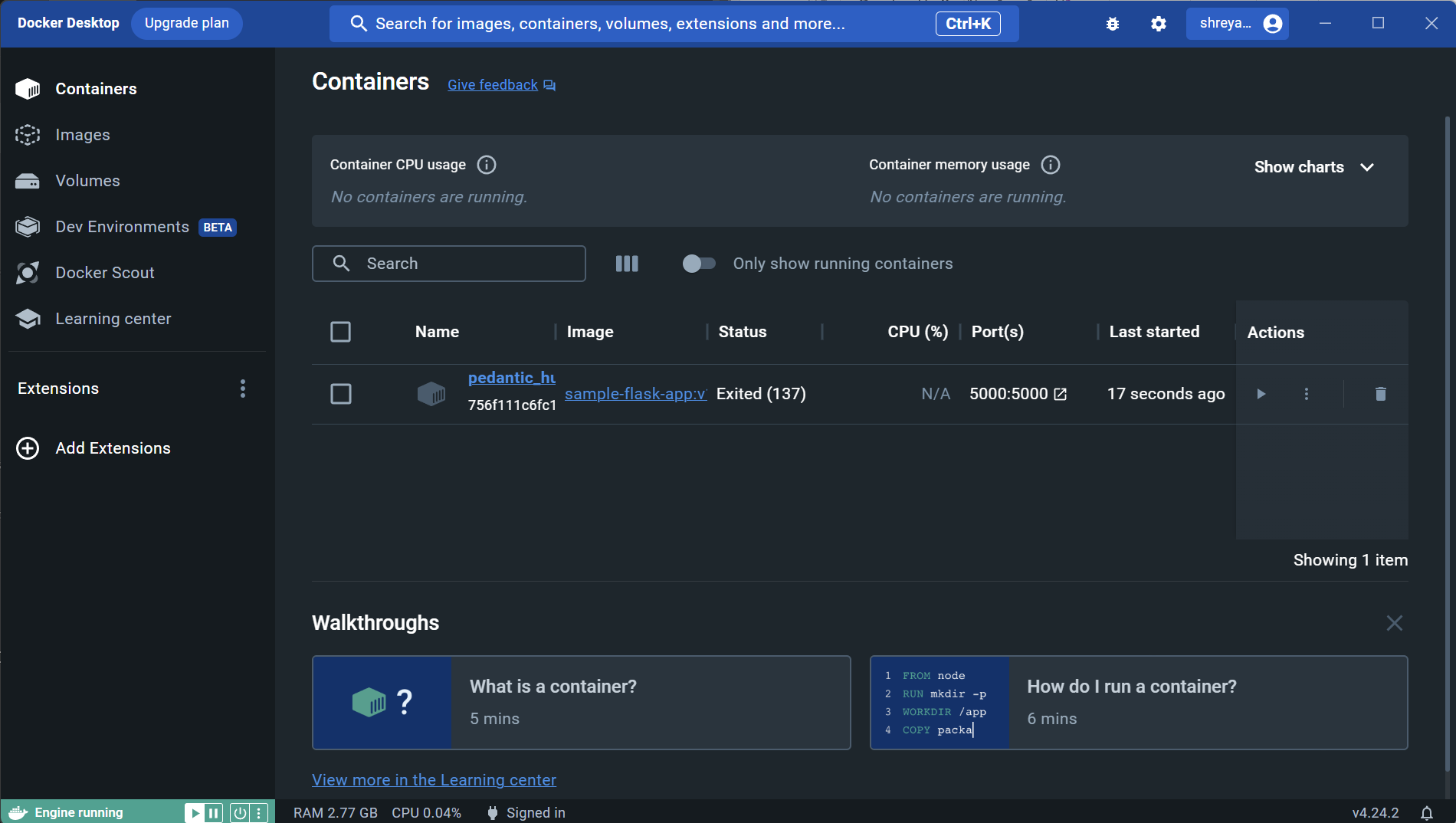
**7.4. Preparing the Docker file**

1. Begin by creating a directory for your Docker project.
2. Inside the directory, generate a text file named "Dockerfile" (without a file extension).
3. Define the base image by specifying the base operating system or another base image sourced from Docker Hub.
4. Install any necessary software, libraries, or dependencies.
5. Copy your application code into the image.
6. Specify the command that the container should execute when it starts.

**7.5.** **Building the Docker Image Using Docker Build**

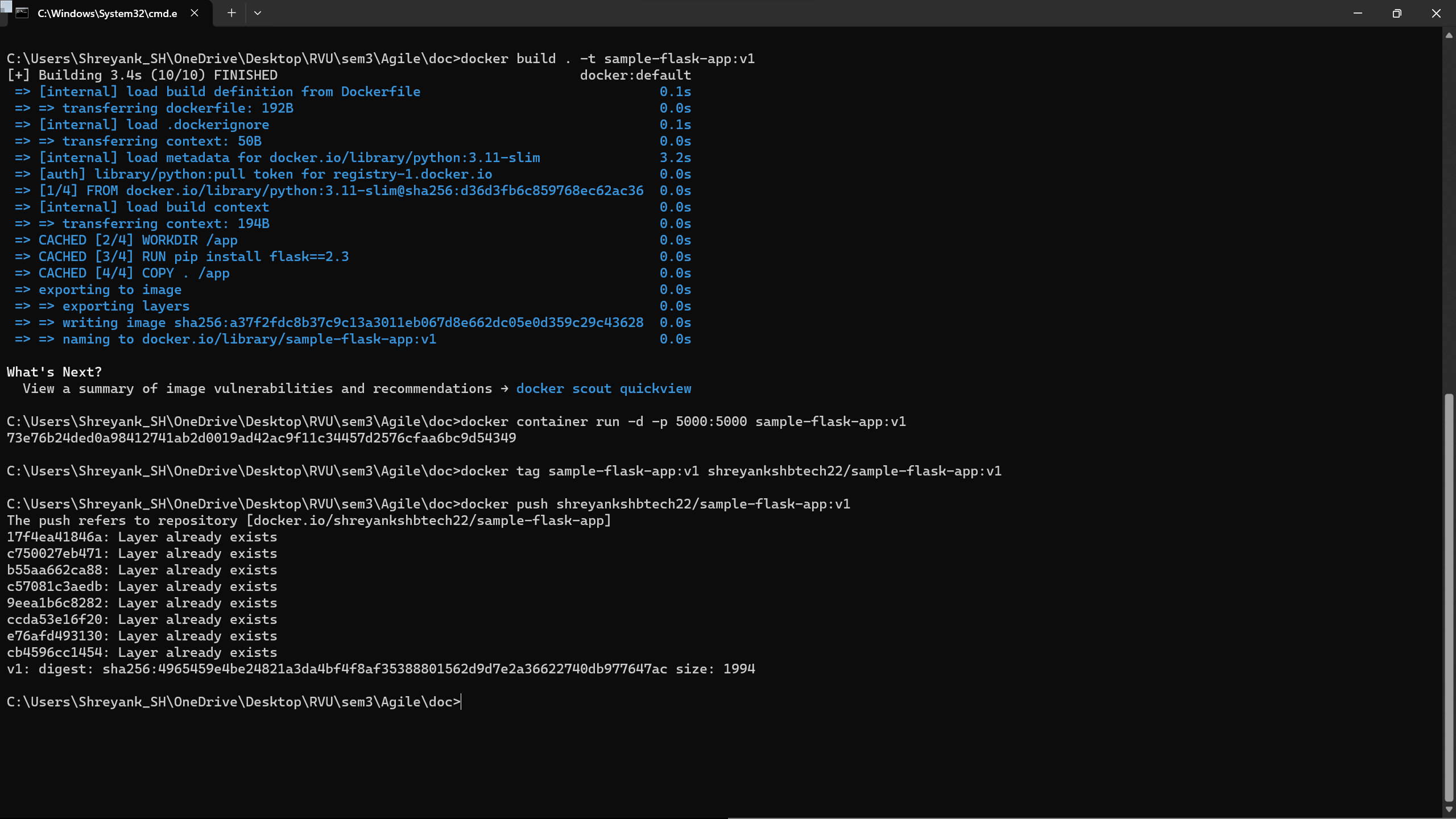
To build the Docker image, open a terminal and navigate to the directory containing your Docker file. Use the **docker build** command to build the image:





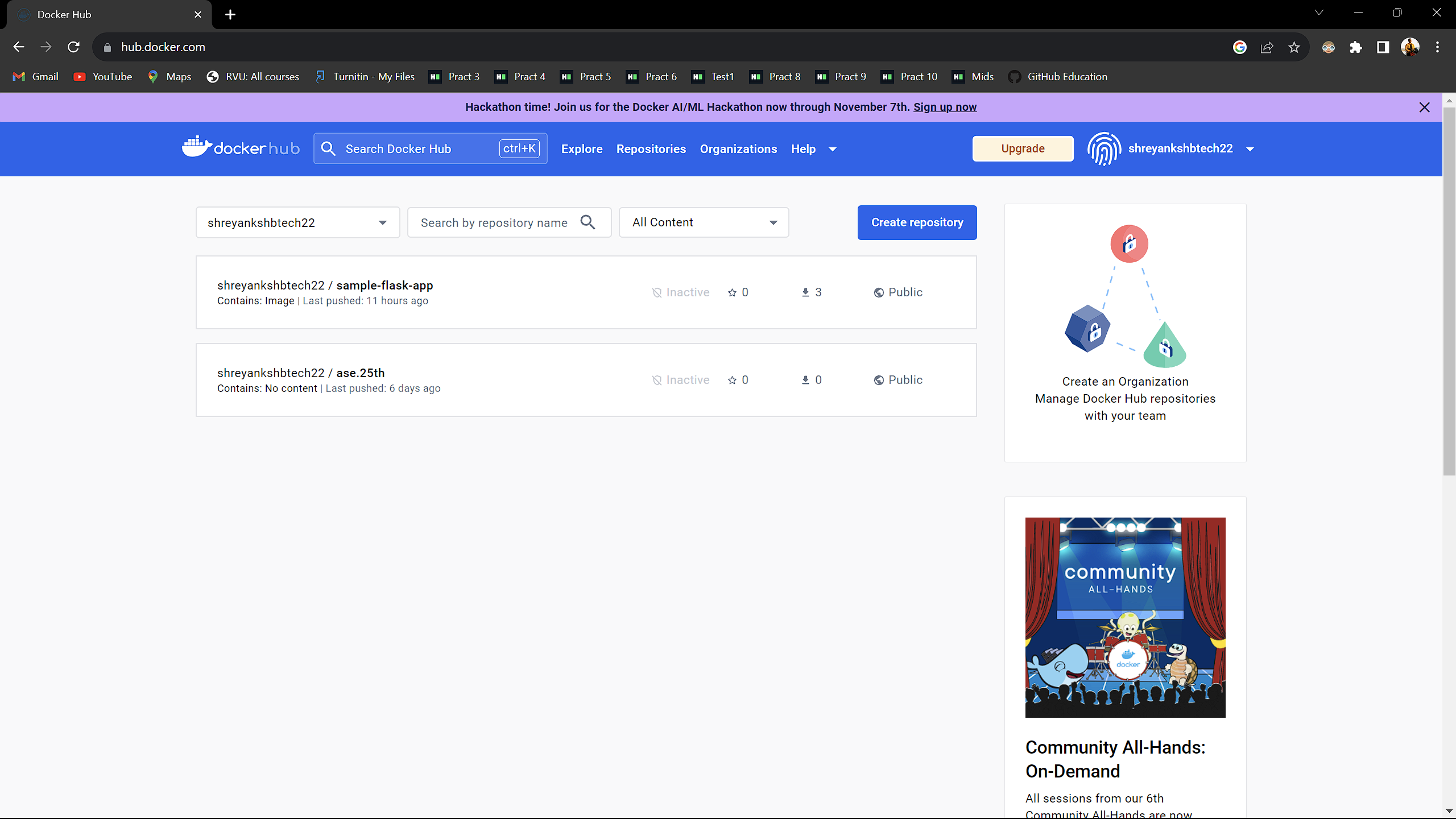
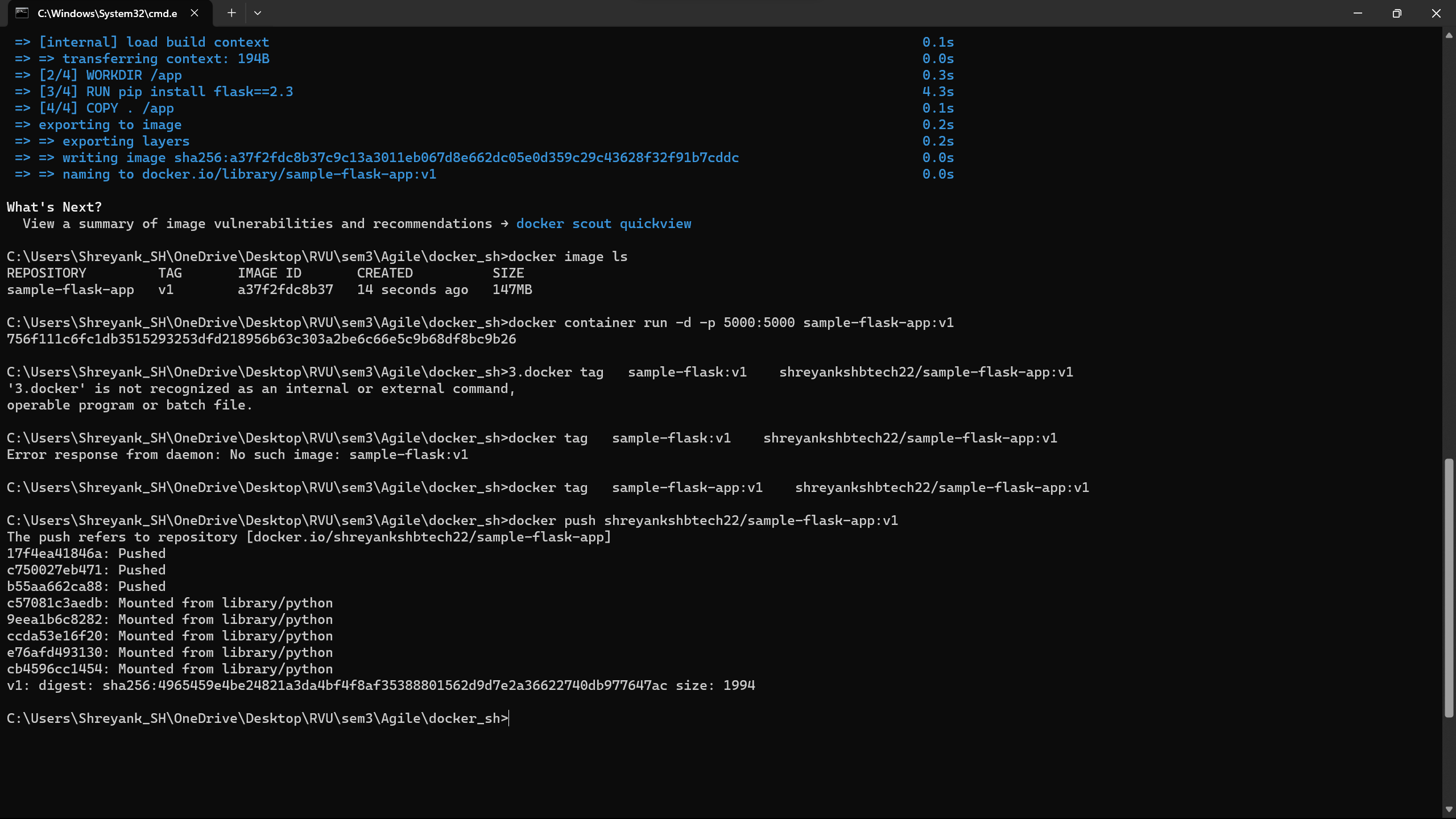
**7.6. Tagging the Image**

Once you've constructed the Docker image, it's important to tag it appropriately for preparation to upload it to Docker Hub. The tag should encompass your Docker Hub username, the image name, and its version.



**7.7. Pushing the Image to Docker Hub**

To upload your Docker image to Docker Hub, adhere to these steps: Firstly, log in to Docker Hub by executing the "docker login" command. Afterward, utilize the "docker push" command to push the image.



**7.8. Conclusion**

Building a Docker image and pushing it to Docker Hub is a crucial aspect of modern software development and deployment. Docker containers provide a consistent and reproducible environment for applications, and Docker Hub allows for easy sharing and distribution of these containers. By following the steps outlined in this report, developers can effectively create, manage, and distribute Docker images.