GITHUB: https://github.com/merlindeborah/71772118127\_Merlin

Setting Up and Pushing a Repository to GitHub Using Git Bash and GitHub Desktop

1. Create a New Repository on GitHub:

Open GitHub account.

Click on the "New" button to create a new repository.

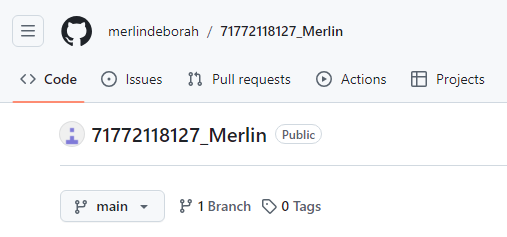
Enter the repository name as `71772118127\_Merlin`.

Provide a description if necessary.

Choose to make it public or private based on your preference here it is public.

Check the box for "Initialize this repository with a README" if you'd like to add a README file.

Click on "Create Repository" .



2. Open Git Bash:

Navigate to the Git Bash application on your desktop or in the start menu and open it.

3. Create a Local Project Directory:

Change to the needed directory on your desktop where you want to create your project. For example:

Create a new folder named `Merls\_DevOps`

Create a text file named `Merlin\_DevOps\_ExampleFile.txt` within the directory

Navigate into the newly created directory using cd:\Desktop\Merls\_DevOps



4. Initialize a Git Repository:

To create a Git repository within the `Merls\_DevOps` directory, initialize it using the following command:

git init

This will create a new subdirectory named `.git` that will contain all the necessary files for version control.



5. Add the File to the Local Repository:

Add the file `Merlin\_DevOps\_ExampleFile.txt` to the staging area:

git add Merlin\_DevOps\_ExampleFile.txt



You can check the status of your repository to confirm that the file is staged using:

git status

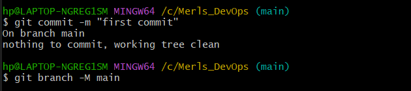


The file appears in green, indicating that it is staged and ready for commit.

6. Commit the File to the Local Repository:

To commit the file and save your changes, use the following command:

git commit -m "first commit"

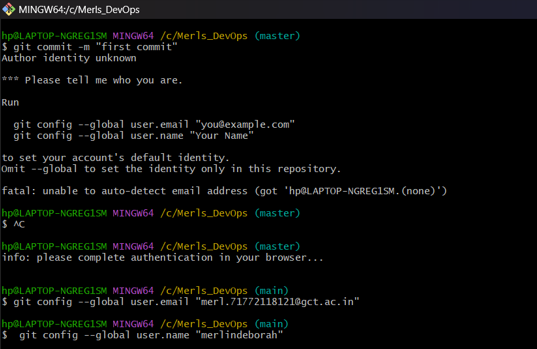


This command will create a snapshot of your file and store it in the local Git repository with the message "first commit".

Note: Authentication



If this is the first time pushing to GitHub, you might be prompted to enter your GitHub username and password.



7. Copy Your Remote Repository's URL from GitHub:

Go to your newly created repository `71772118127\_Merlin` on GitHub.

Click on the "Code" button and copy the repository URL (either HTTPS or SSH, depending on your Git configuration). For example:

https://github.com/merlindeborah/71772118127\_Merlin.git

8. Add the Remote Repository URL:

Add the copied URL as the `origin` remote to your local repository:

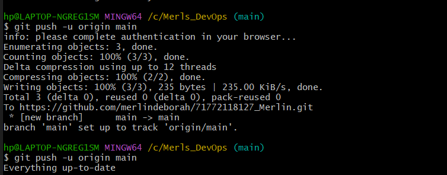
git remote add origin <https://github.com/merlindeborah/71772118127_Merlin.git>



9. Push the Code to GitHub:

Push your local commits to the remote repository on GitHub:

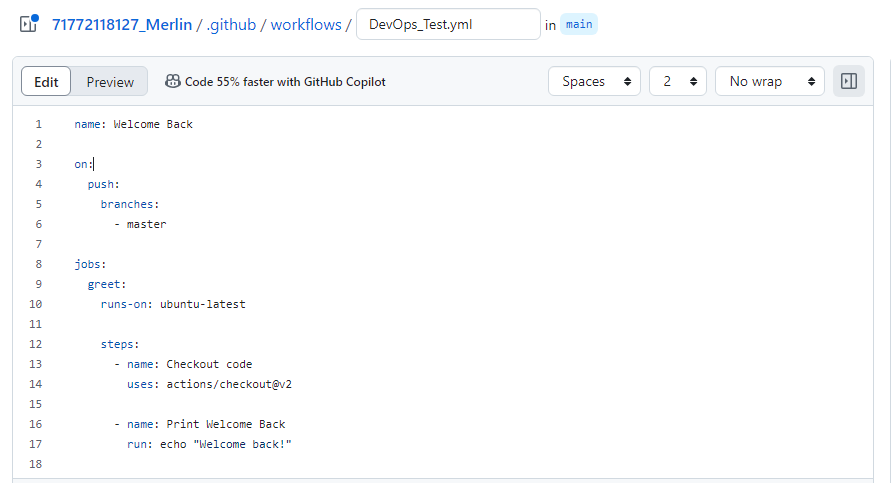
git push -u origin master

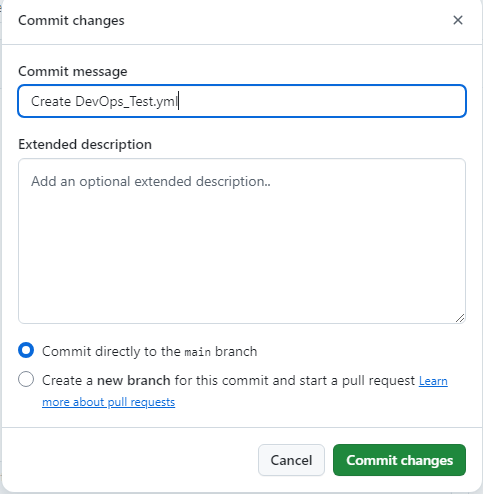


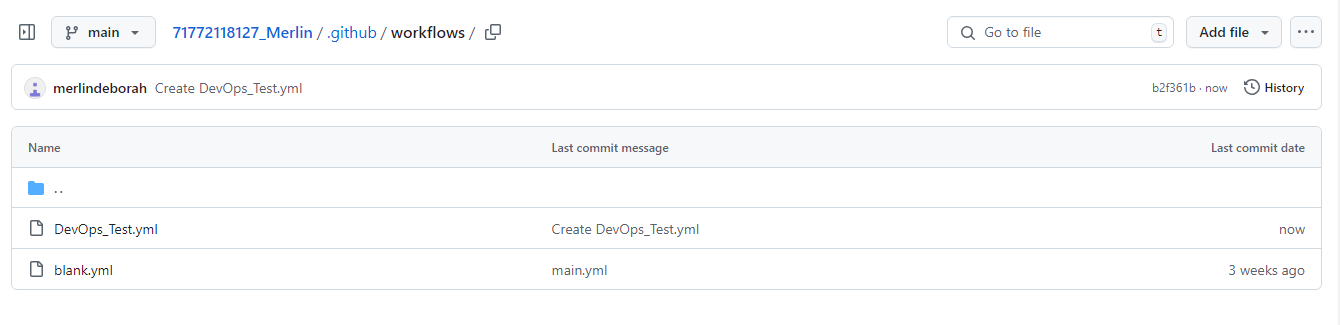
10. View Your Files in the Remote Repository:

Go to your GitHub repository `71772118127\_Merlin`.

Refresh the page to see the uploaded `Merlin\_DevOps\_ExampleFile.txt` file.







Using GitHub Desktop to Push Local Content to GitHub

1. Download and Install GitHub Desktop:

If you don’t have GitHub Desktop installed, download it from the [GitHub Desktop website](https://desktop.github.com/).

2. Set Up the Repository in GitHub Desktop:

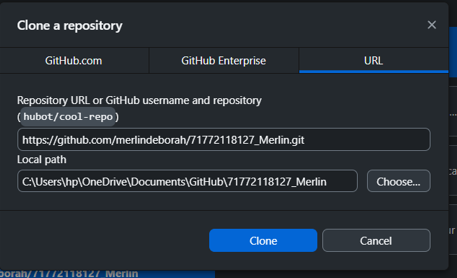
Open the GitHub Desktop application.

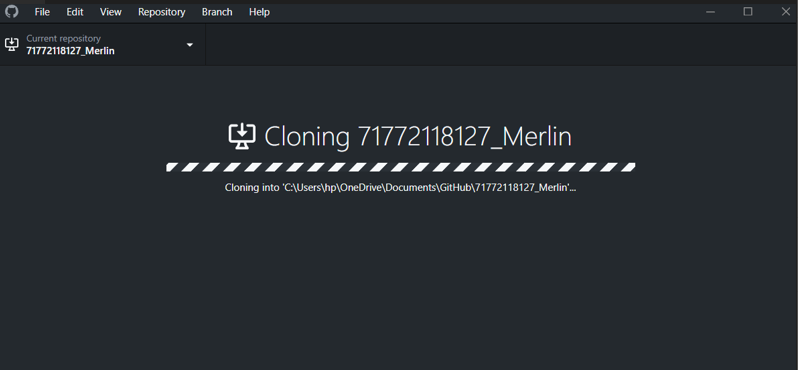
Go to "File > Clone Repository".

Choose "URL-<https://github.com/merlindeborah/71772118127_Merlin.git>" and paste the GitHub repository URL you copied earlier.

Select a local path where you want to clone the repository.

Click "Clone".





3. Copy the Required Files to the Cloned Folder:

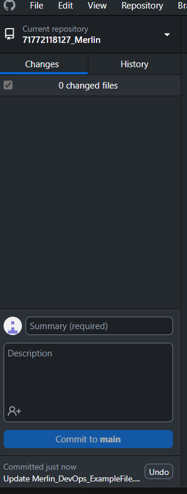
Open the cloned folder on your computer.

Copy all changed files from local `Merls\_DevOps` directory into the cloned repository folder.

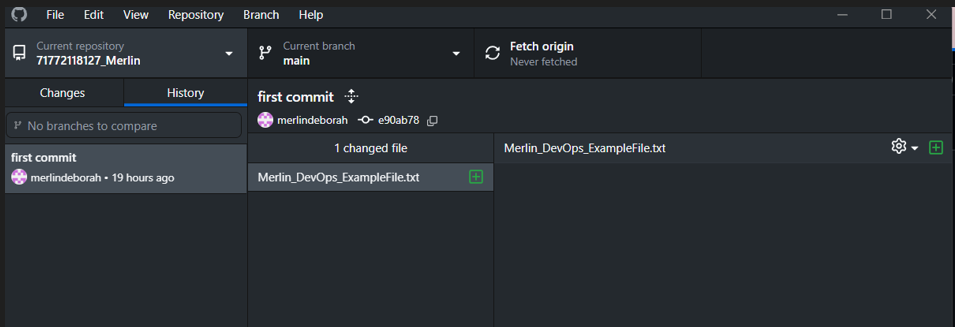
4. Commit Changes Using GitHub Desktop:

Go back to the GitHub Desktop application.

You should see the newly added files as changes.



Click on "Commit to master".



5. Publish the Branch to GitHub:

Click on "Publish Branch" to upload all files and commits to the remote GitHub repository.

After completing these steps, go to your GitHub repository `71772118127\_Merlin` and confirm that all your files have been successfully uploaded and are visible.

DOCKER LINK: <https://hub.docker.com/u/merlin71772118127>

TO SET UP DOCKER:

1. Create a Docker Hub Account

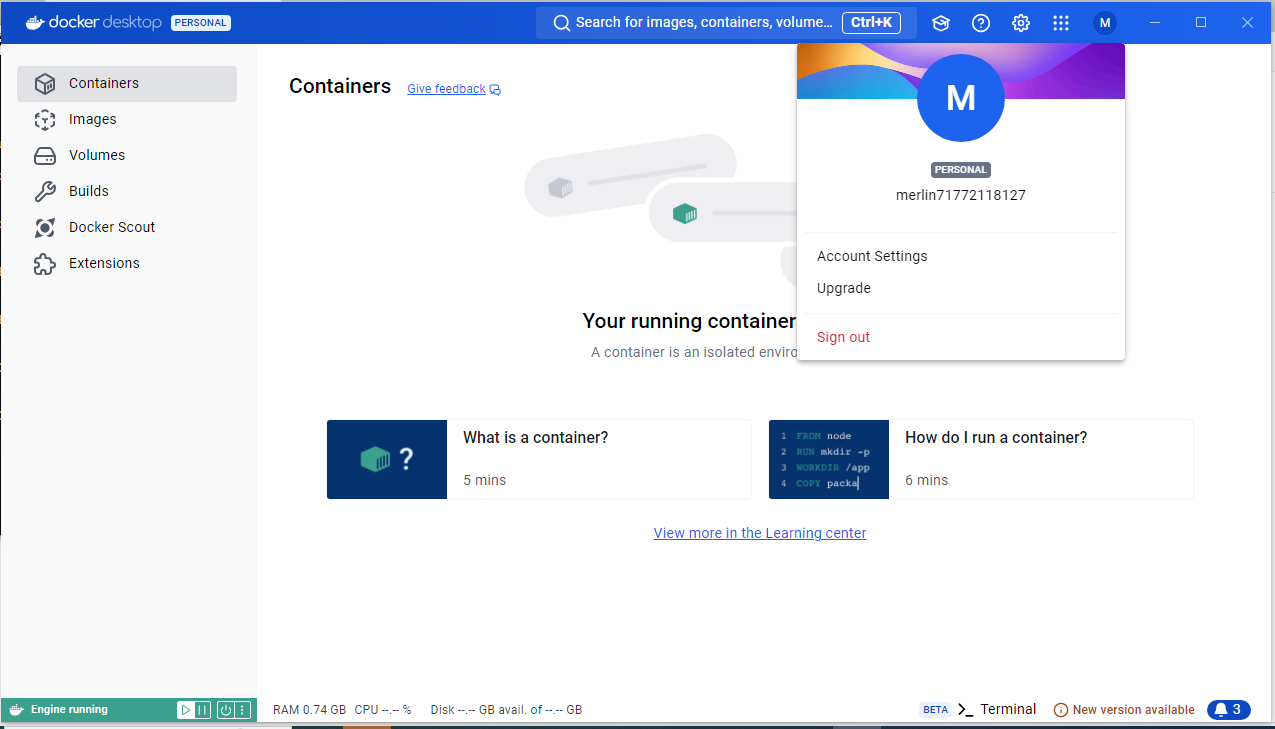
Go to the Docker Hub website.

Click on Sign Up and create a new account by providing your email, username, and password.

2. Install Docker Desktop

Download and install Docker Desktop from the Docker website.

Follow the installation instructions for your operating system.



Docker Basic Commands:

1. Check Docker Version:

docker --version

- This command checks and displays the installed Docker version along with its build number. In our case, the output shows:



Docker version 27.1.1, build 6312585

2. List Docker Images:

docker images



- This command lists all the Docker images present on your system. The output format shows columns such as REPOSITORY, TAG, IMAGE ID, CREATED, and SIZE.

- Since there are no images currently pulled, these columns will be empty.

3. List All Docker Images Including Intermediate Layers:

docker images –a



- The -a flag stands for "all", and this command lists all images, including intermediate layers that might not show up with the docker images command alone.

- The output will again show details like IMAGE ID, REPOSITORY, TAG, CREATED, and SIZE.

4. List Running Docker Containers:

docker ps



- This command lists all the currently running containers on your Docker engine.

- Columns include CONTAINER ID, IMAGE, COMMAND, CREATED, STATUS, PORTS, and NAMES.

- If no containers are running, the list will be empty.

5. List All Docker Containers (Running and Stopped):

docker ps –a



- The -a flag displays all containers, whether they are currently running, exited, or stopped.

- It provides the same columns as docker ps but includes more entries representing stopped or exited containers.

Pull Image from Docker Hub:

1.Pull the Ubuntu Image from Docker Hub:

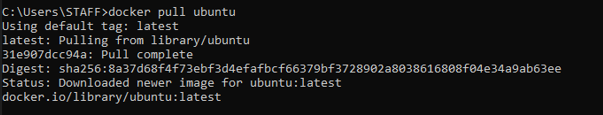
docker pull ubuntu

- This command pulls the latest Ubuntu image from the Docker Hub repository.

- Since you did not specify a tag, Docker uses the default tag latest.

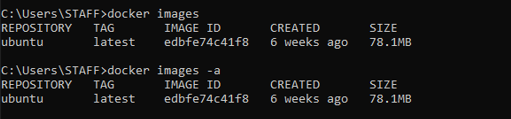
- The following steps occur during this command execution:

- latest: Pulling from library/ubuntu indicates the command is pulling from the library/ubuntu repository.

2. 2.Verify Pulled Image:

To verify that the Ubuntu image has been pulled successfully, you can use:

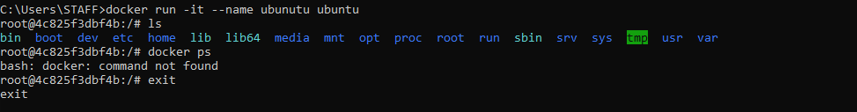
docker images



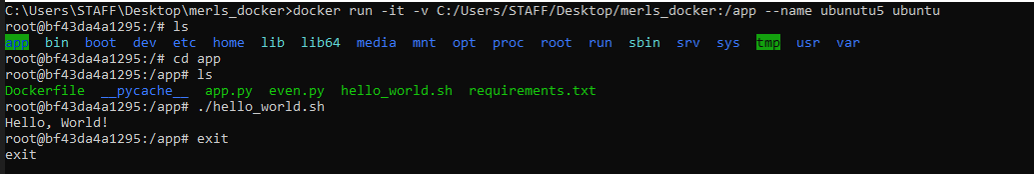
The ubuntu image should now appear in the list, with details like REPOSITORY, TAG, IMAGE ID, CREATED, and SIZE.

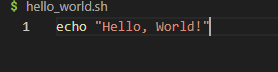
3. Running a container from pulled image:

C:\Users\STAFF>docker run -it --name ubunutu ubuntu



This command creates and starts a new container with the Ubuntu image. The -it flag allows interactive terminal access, and --name gives the container a name (ubunutu).





1. Run the Docker Container

docker run -it -V C:/Users/STAFF/Desktop/merls\_docker:/app --name ubunutu5 ubuntu

This command creates and runs an interactive Ubuntu container named ubunutu5, mounting the local directory C:/Users/STAFF/Desktop/merls\_docker to the container’s /app directory.

2. Access the Container Shell

ls

Inside the container, the ls command lists the contents of the root directory, showing standard directories like /bin, /etc, /lib, etc.

3. Navigate to the Mounted Directory

cd app

ls

Change to the /app directory, where your local files are mounted. The ls command reveals files like Dockerfile, app.py, and requirements.txt.

4. Execute a Shell Script and Exit

./hello\_world.sh

exit

Running ./hello\_world.sh executes the script, likely printing "Hello, World!" to the terminal. The exit command then closes the container shell.

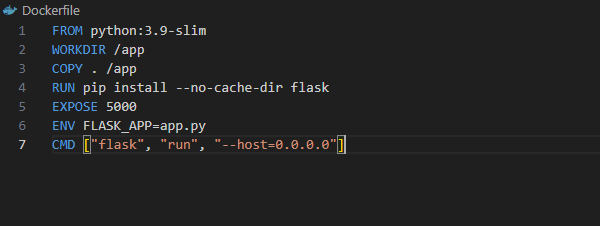
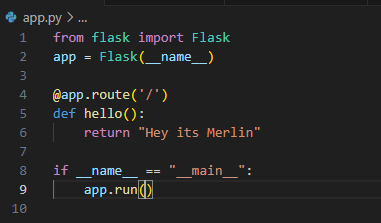
**Building a New Image**

**Step 1: Create a Dockerfile**

Open a text editor (like Visual Studio Code, Notepad++, etc.).  
Create a new file named **Dockerfile** (with no extension) in your project directory.

**Step 2: Write the Dockerfile**

Simple Dockerfile for a Python application:

**Step 3: Build the Docker Image**

Open your command prompt or terminal.  
Navigate to the directory containing your Dockerfile.  
Use the following command to build your image:

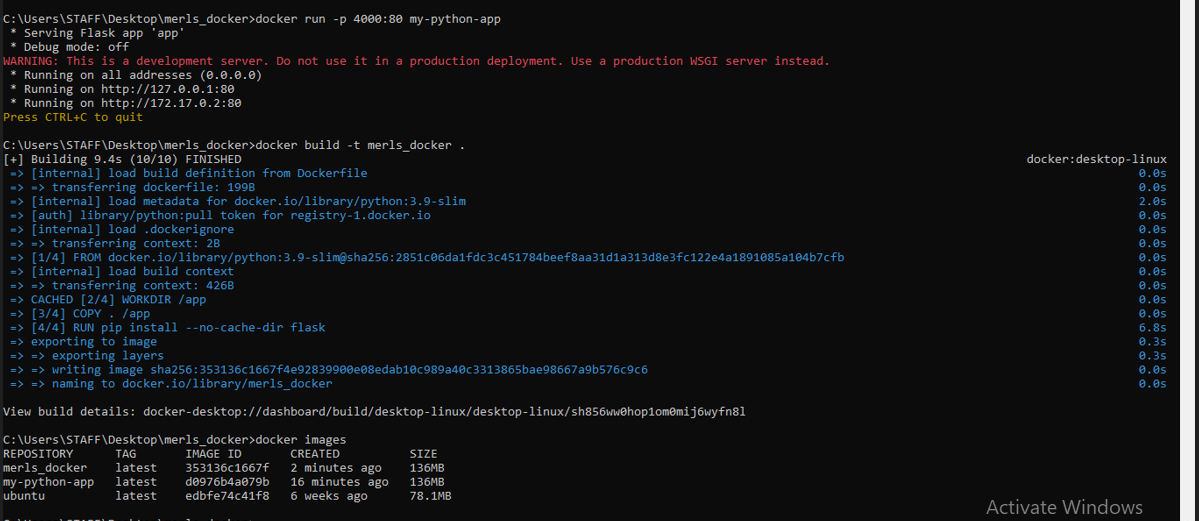
C:\Users\STAFF\Desktop\merls\_docker>docker build -t merls\_docker .

This command builds a Docker image named merls\_docker using the Dockerfile in the current directory.

### Step 4: Run Your Docker Image

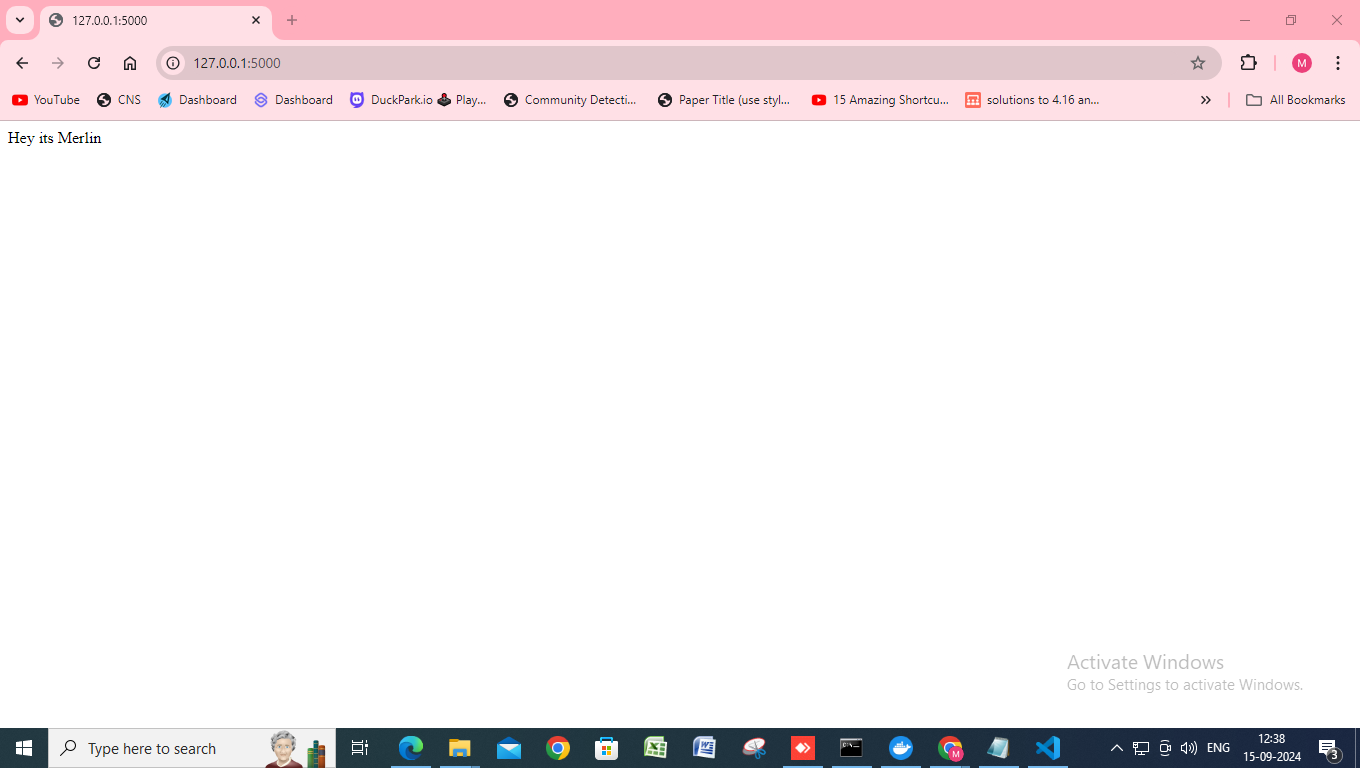
### Run the newly created Docker image, use the following command:

docker run -p 4000:80 merls\_docker



**Step 5: Access Your Application**

Our application is a web application running on 127.0.0.1:5000.



Pushing Docker Image to Docker Hub

1. Tag the Docker Image

docker tag 299de312b3fc merlin71772118127/docker-merls\_docker:latest

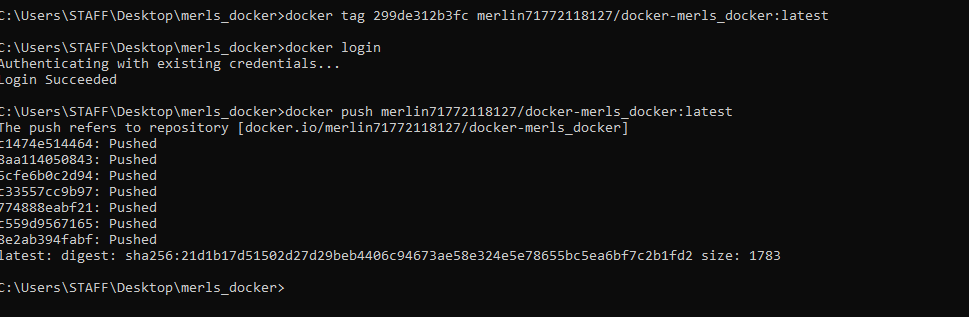
This command tags the existing Docker image with ID 299de312b3fc and assigns it a new name (merlin71772118127/docker-merls\_docker) and a version tag (latest). This prepares the image for pushing to Docker Hub.

2. Login to Docker Hub

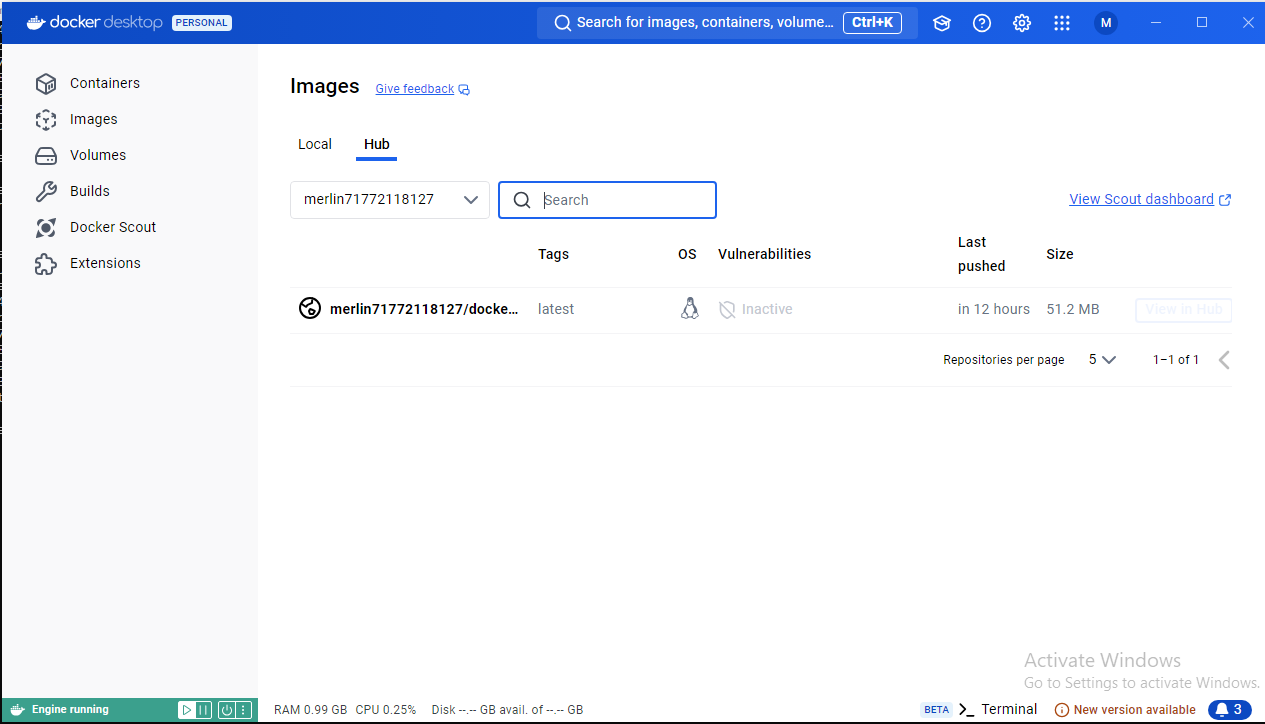
This command authenticates the user to Docker Hub using stored credentials. A successful login will confirm that you are connected to your Docker Hub account.

3. Push the Tagged Image to Docker Hub

docker push merlin71772118127/docker-merls\_docker:latest



This command uploads the tagged image (merlin71772118127/docker-merls\_docker:latest) to your Docker Hub repository. The output indicates that various layers of the image are being pushed successfully, culminating in a digest that confirms the image has been stored on Docker Hub.



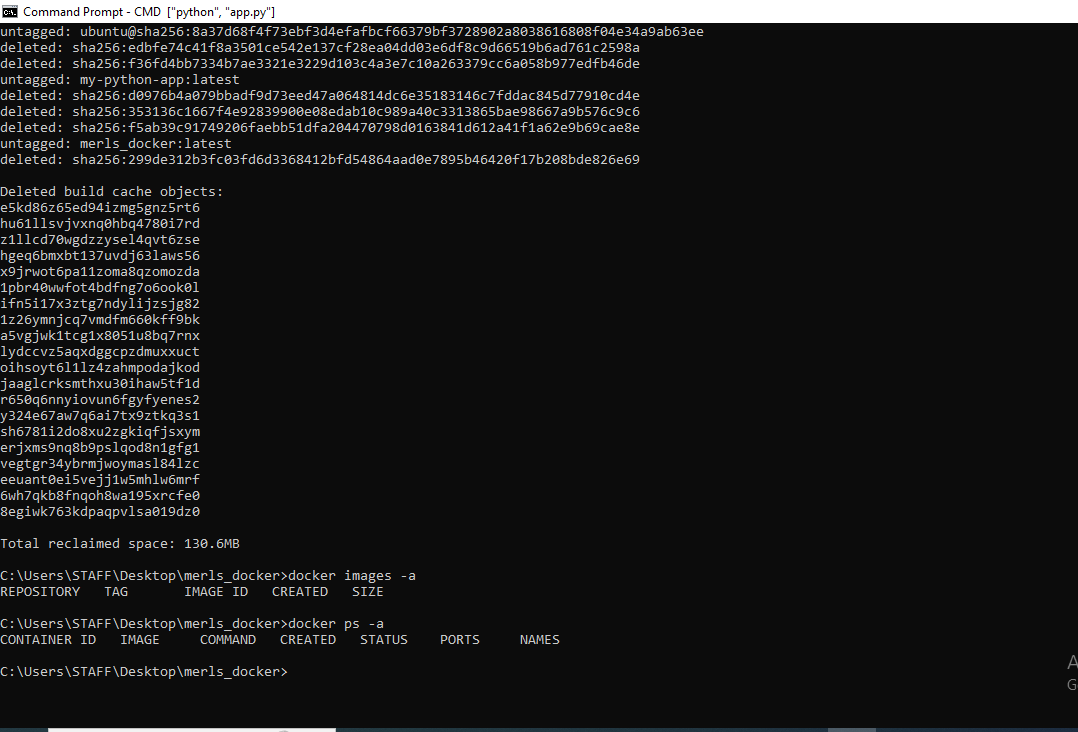
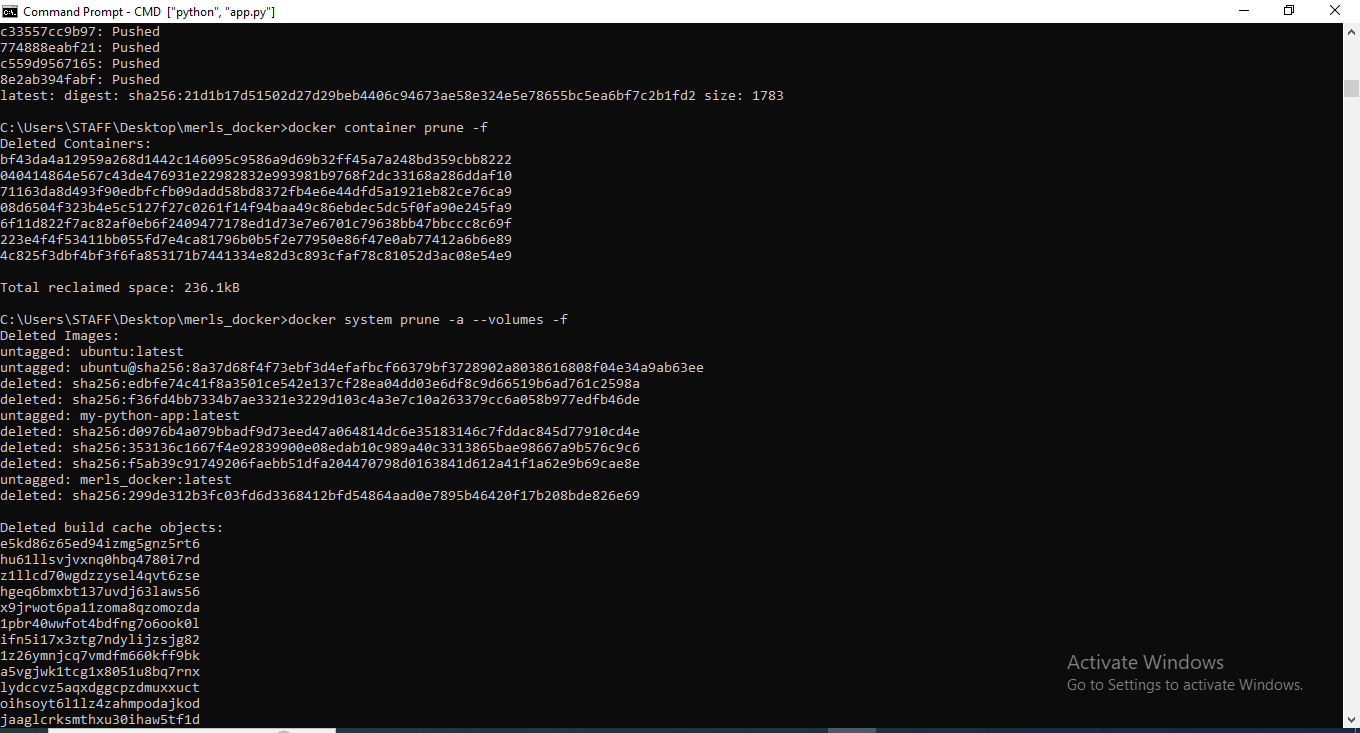
Deleting containers and images

1. Prune Unused Containers

docker container prune -f

This command removes all stopped containers, helping to free up space. The output lists the deleted containers and indicates the total amount of space reclaimed (236.1 kB in this case).

2. Prune Unused Images and Volumes

docker system prune -a --volumes -f

This command removes all unused images (not just dangling ones), all stopped containers, and optionally, all unused volumes. The output shows the deleted images and their IDs, including any untagged images and build cache objects, ensuring that unnecessary data is cleared to optimize storage space.