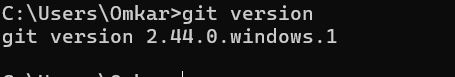
1 Exploring Git Commands through Collaborative Coding.

**Task 0 : Install Git**

**Output / Command ::**

***Verify Git installation with the version command.***

****

**Task1 : Setting Up Git Repository**

● Open the command-line interface on your computer.

● Navigate to the directory where you want to create your Git repository.

● Run the basic git commands:

git init - This initialises a new Git repository in the current directory.

Commands/output:



**Task2 : Creating and Committing Changes**

● Create a new text file named "example.txt" using any text editor.

● Add some content to the "example.txt" file.

● In the command-line interface, run the following commands:

git status - This command shows the status of your working directory, highlighting

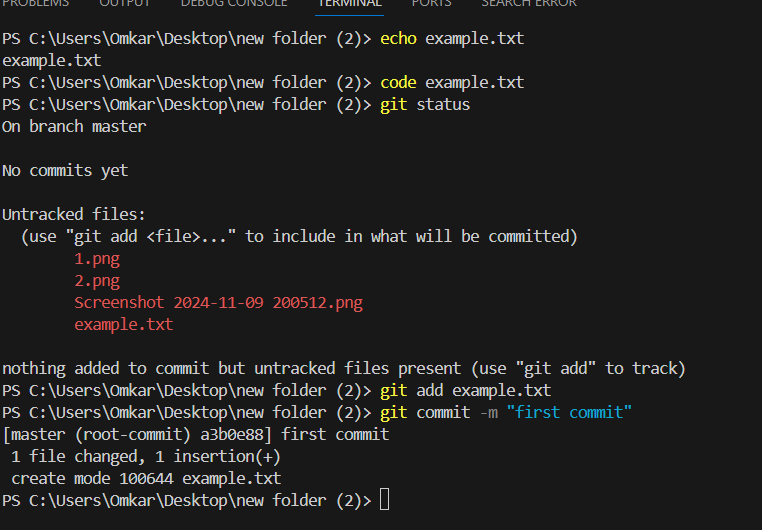
untracked files.

git add example.txt - This stages the changes of the "example.txt" file for commit.

git commit -m "Add content to example.txt" - This commits the staged changes

with a descriptive message.

Command /Output::



**Task3 : Exploring History**

Modify the content of "example.txt."

Run the following commands: git status - Notice the modified file is shown as

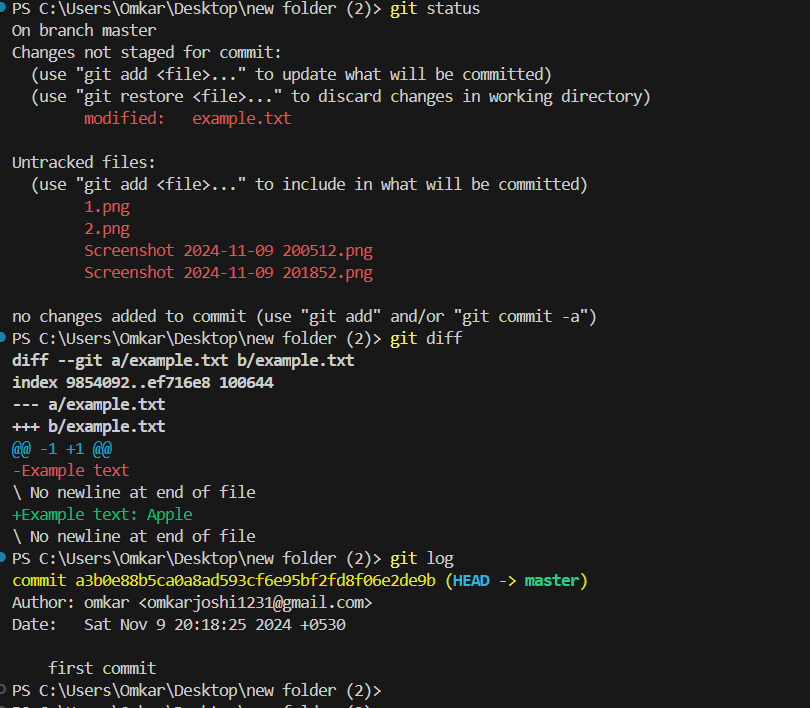
"modified."

git diff - This displays the differences between the working directory and the last

commit.

git log - This displays a chronological history of commits.

Command/output:



**Task4 : Branching and Merging**

Create a new branch named "feature" and switch to it:

git branch feature, git checkout feature

or shorthand: git checkout -b feature

● Make changes to the "example.txt" file in the "feature" branch.

● Commit the changes in the "feature" branch.

● Switch back to the "master" branch: git checkout master

● Merge the changes from the "feature" branch into the "master" branch:

git merge feature

command/output



**Task5 : Collaborating with Remote Repositories**

● Create an account on a Git hosting service like GitHub (https://github.com/).

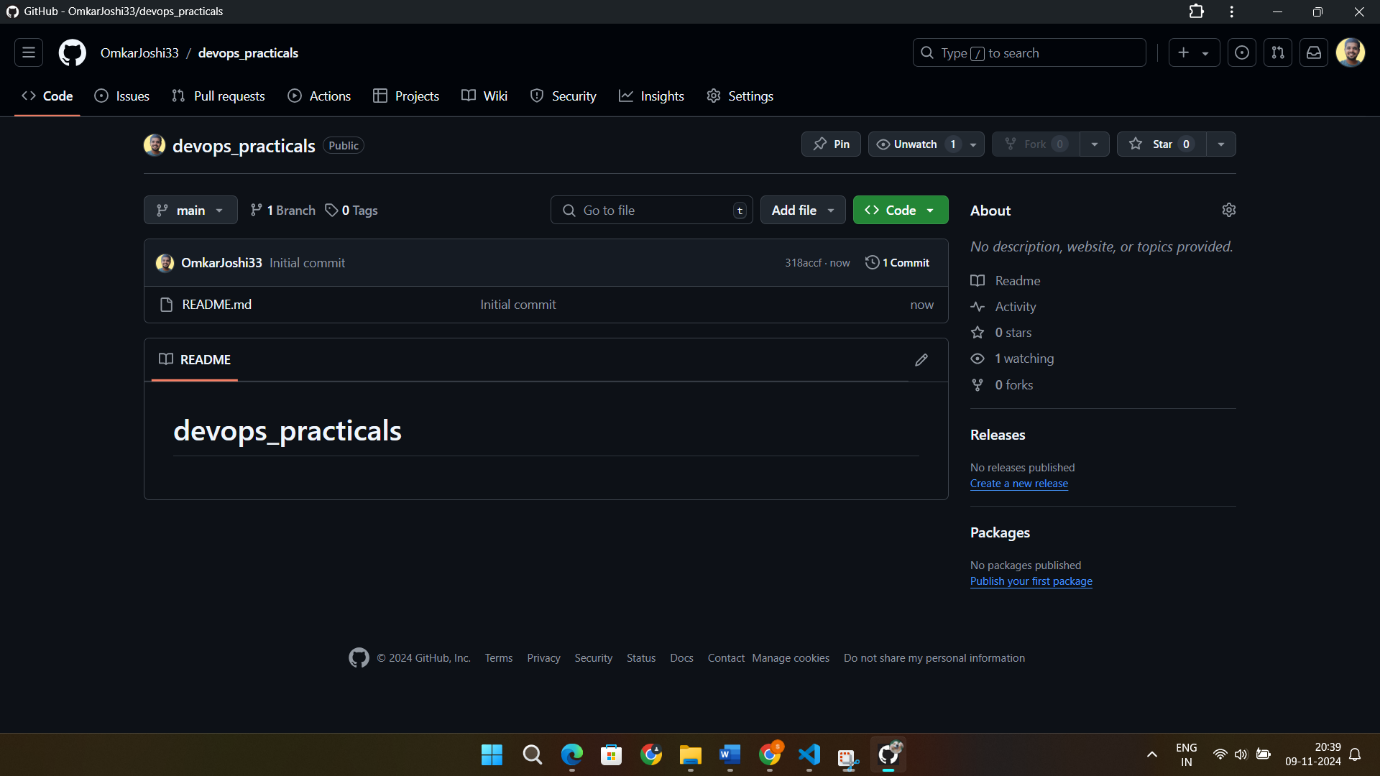
● Create a new repository on GitHub.

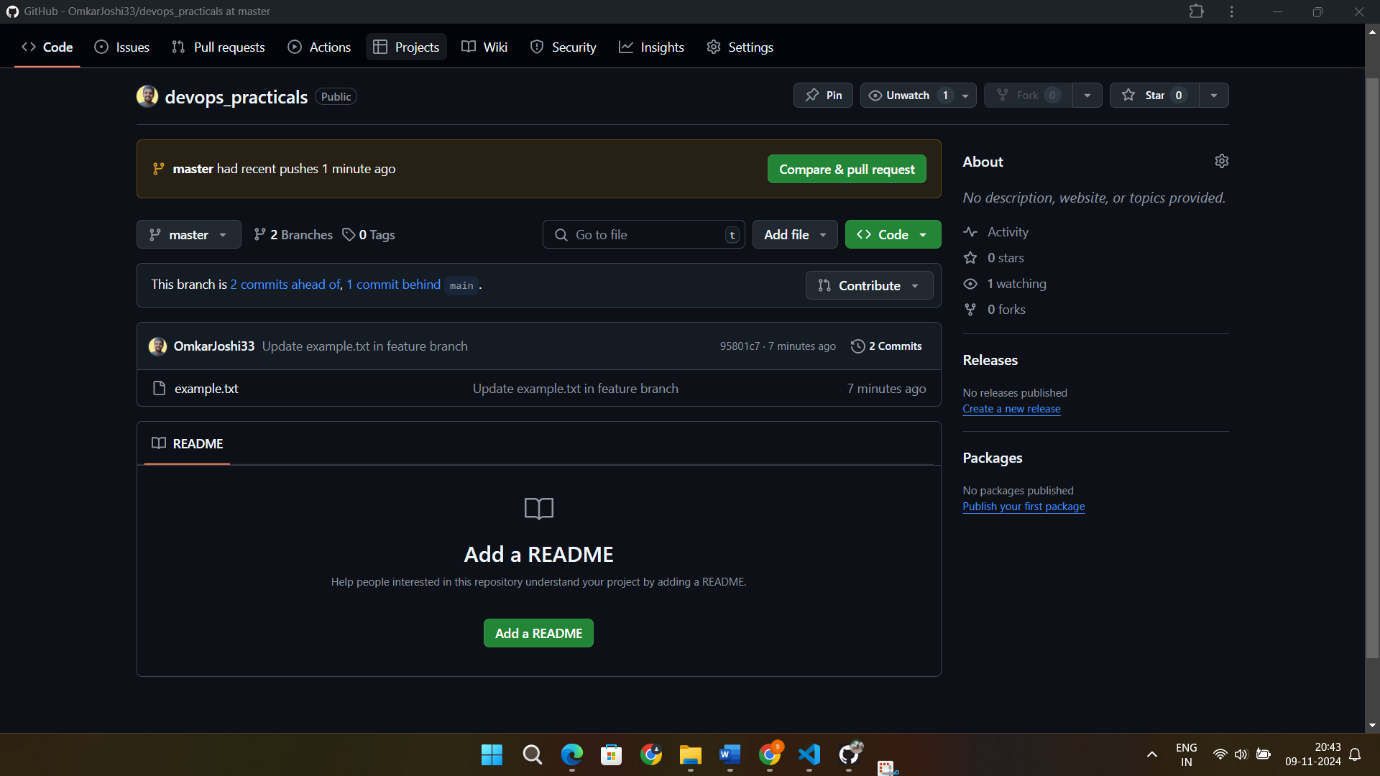
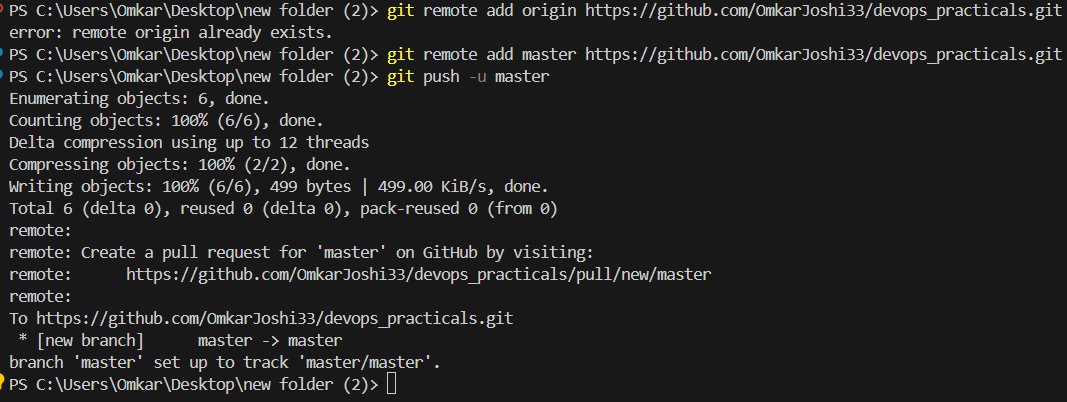
● Link your local repository to the remote repository:

git remote add origin <repository\_url>

● Push your local commits to the remote repository: git push origin master

Commands/ output::





**2 Implement GitHub Operations using Git.**

**Task 1: Cloning a Repository**

● Sign in to your GitHub account.

● Find a repository to clone (you can use a repository of your own or any public

repository).

● Click the "Code" button and copy the repository URL.

● Open your terminal or command prompt.

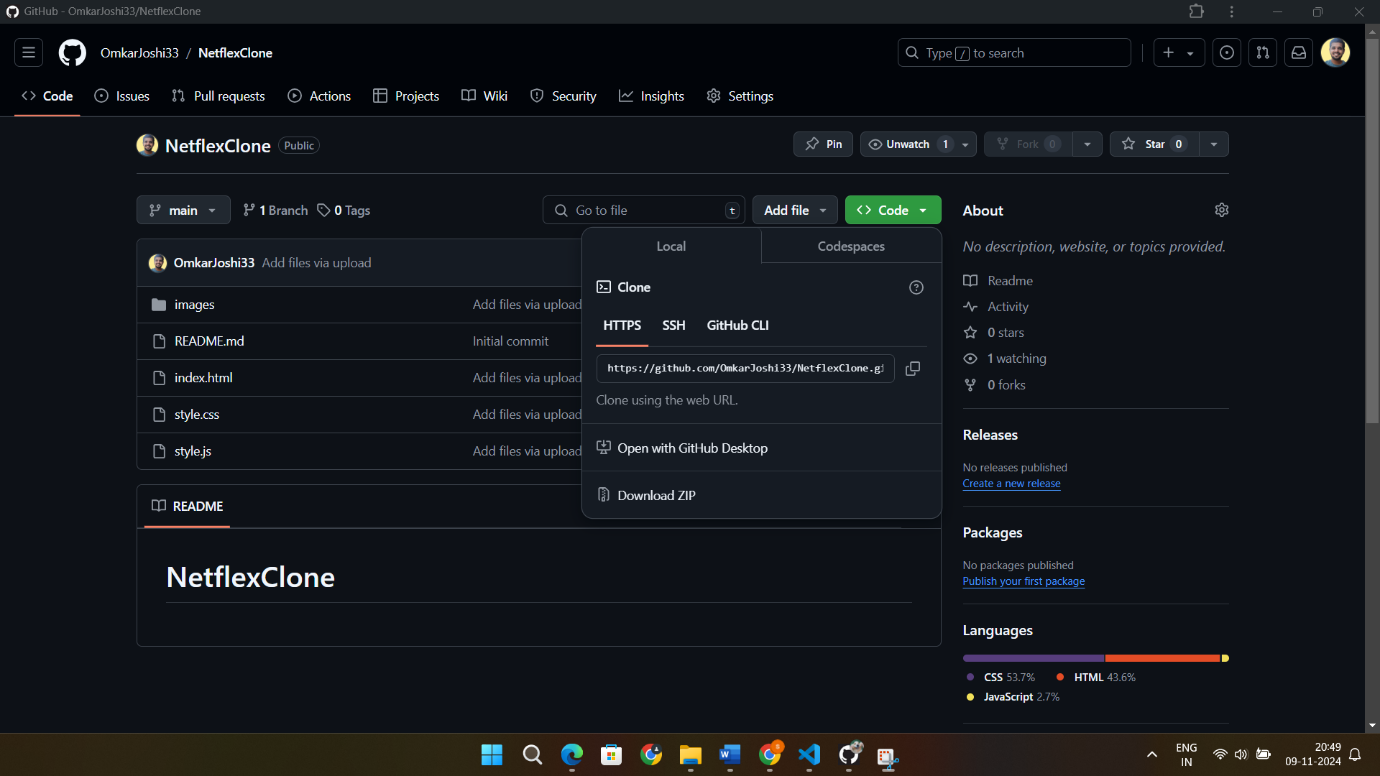
● Navigate to the directory where you want to clone the repository.

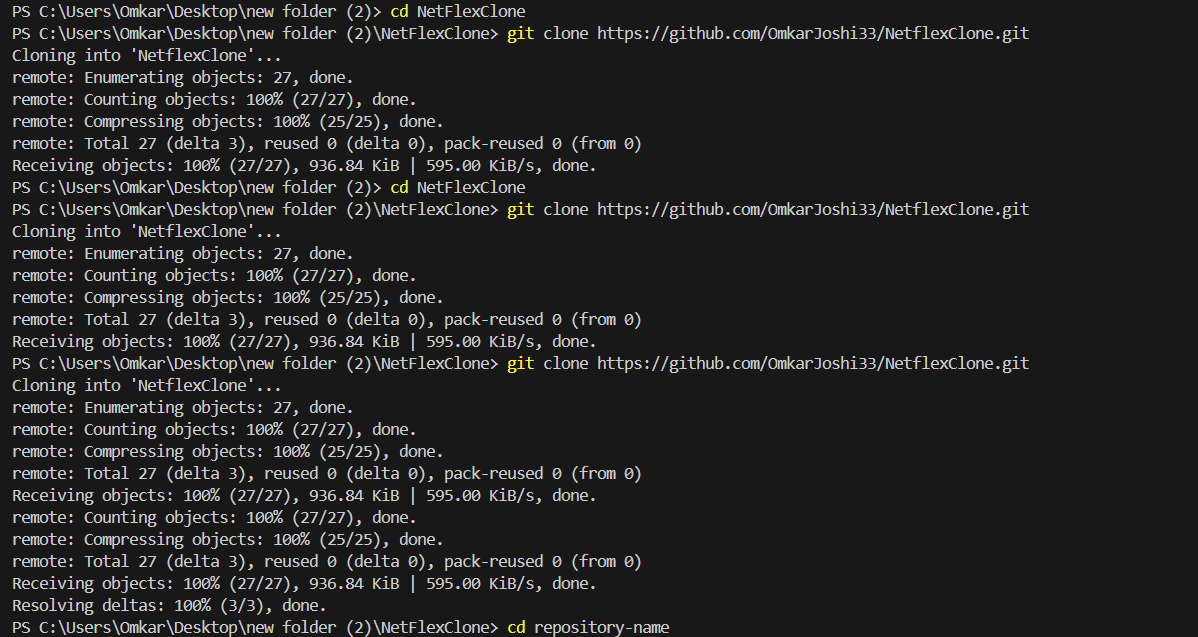
● Run the following command: git clone <repository\_url>

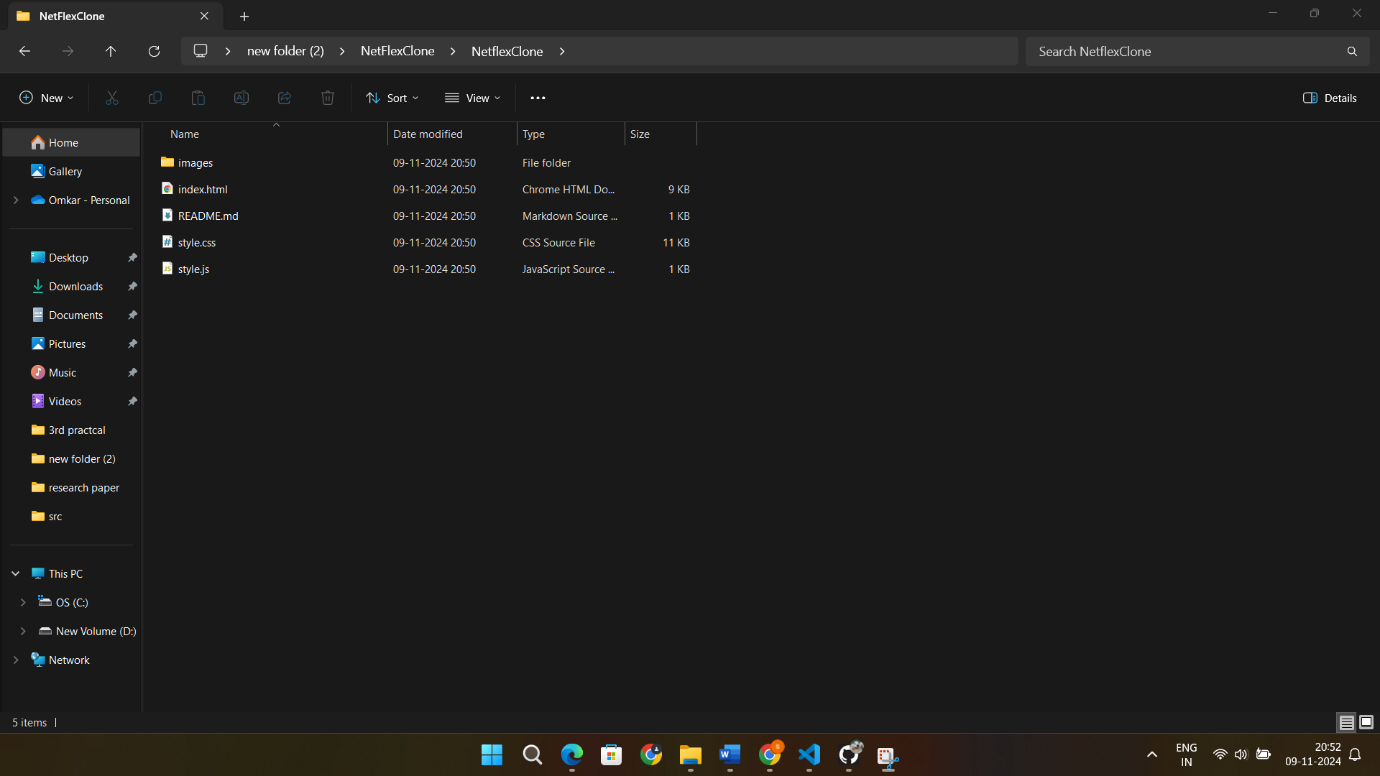
● Replace <repository\_url> with the URL you copied from GitHub.

● This will clone the repository to your local machine.

Command/output







**Task 2: Making Changes and Creating a Branch**

Navigate into the cloned repository: cd <repository\_name>

● Create a new text file named "amit.txt" using a text editor.

● Add some content to the "amit.txt" file.

● Save the file and return to the command line.

● Check the status of the repository: git status

● Stage the changes for commit: git add amit.txt

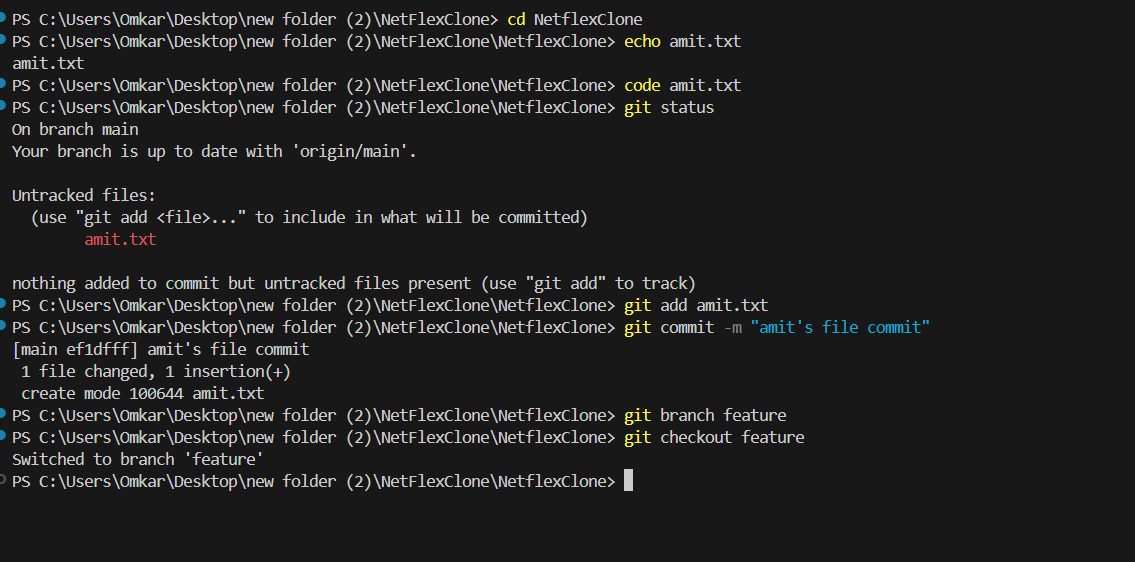
● Commit the changes with a descriptive message:

git commit -m "Add content to amit.txt"

● Create a new branch named "feature": git branch feature

● Switch to the "feature" branch: git checkout feature

Command/output:



**Task 3: Pushing Changes to GitHub**

● Add Repository URL in a variable

git remote add origin <repository\_url>

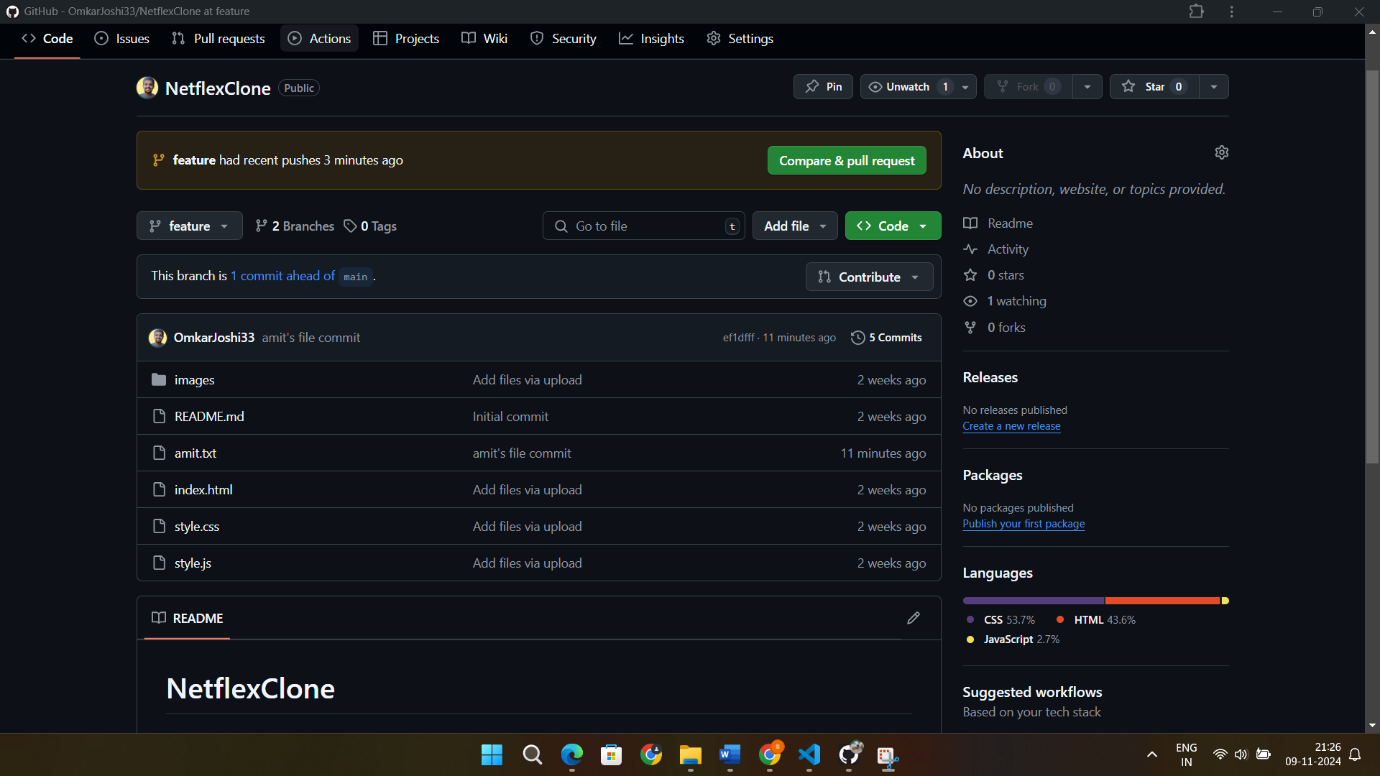
● Replace <repository\_url> with the URL you copied from GitHub.

● Push the "feature" branch to GitHub: git push origin feature

● Check your GitHub repository to confirm that the new branch "feature" is

available.

Command/output::





**Task 4: Collaborating through Pull Requests**

● After the pull request is merged, update your local repository:

git checkout main. git pull origin main

● Create a pull request on GitHub: ● Go to the repository on GitHub.

● Click on "Pull Requests" and then "New Pull Request."

● Choose the base branch (usually "main" or "master") and the compare branch

("feature").

● Review the changes and click "Create Pull Request."

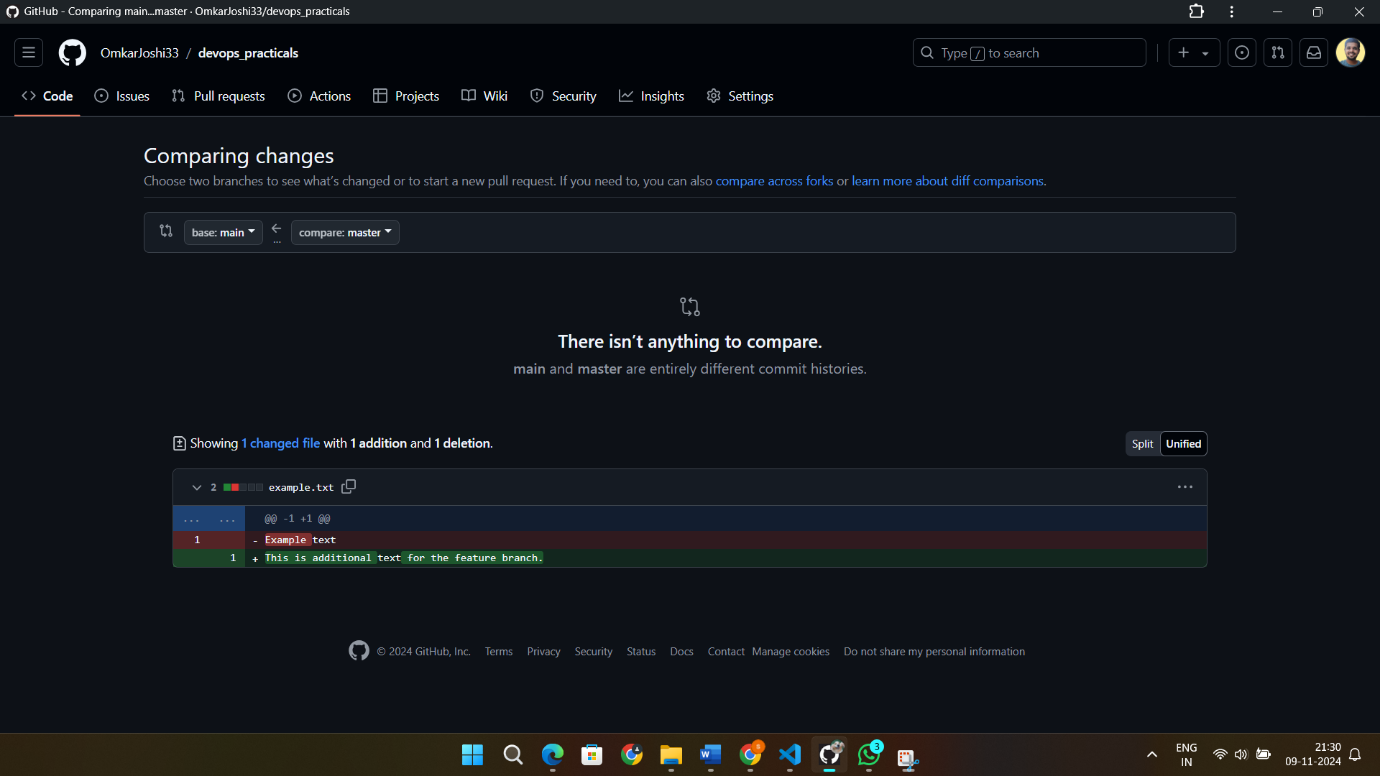
● Review and merge the pull request:

● Add a title and description for the pull request.

● Assign reviewers if needed.

● Once the pull request is approved, merge it into the base branch.

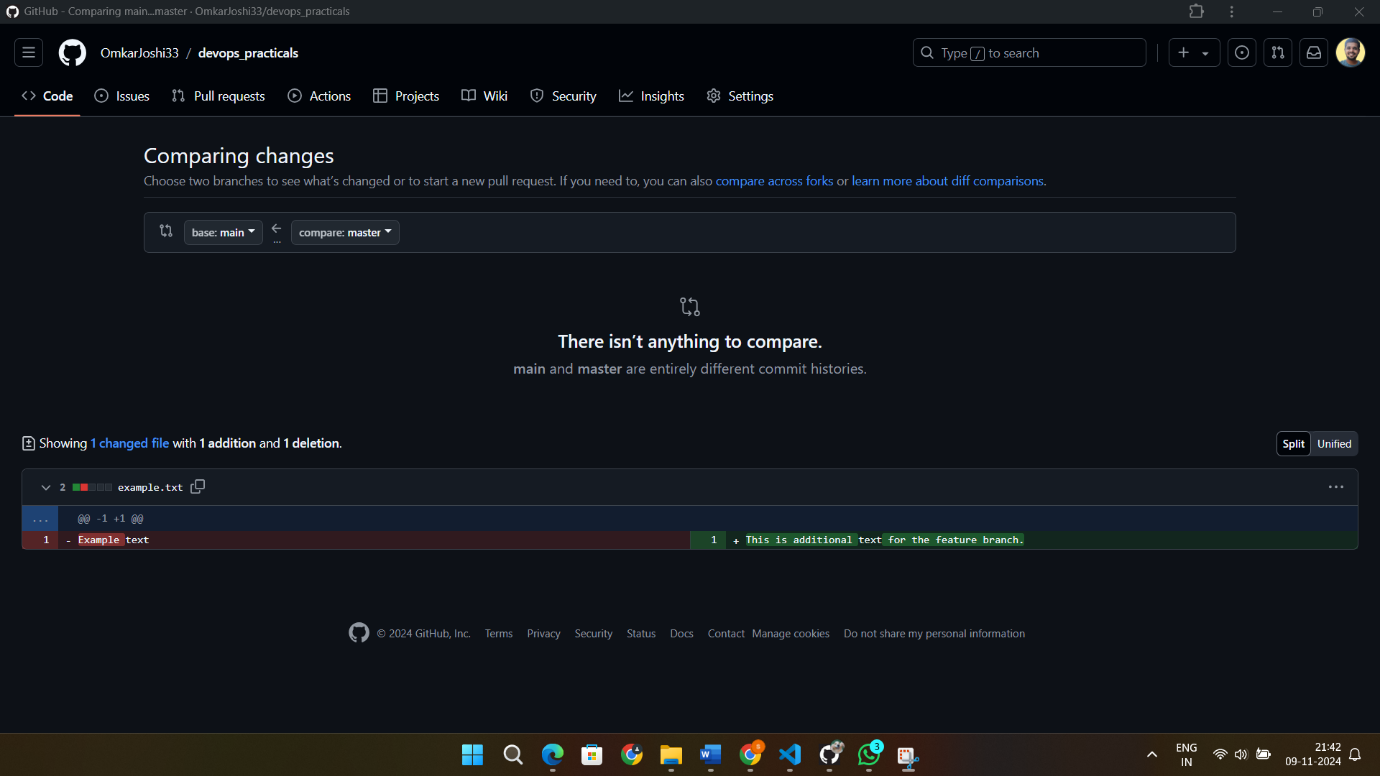
Command/output



**Task 5: Syncing Changes**

● After the pull request is merged, update your local repository: git checkout main. git pull origin main

Output::



**Implement GitLab Operations using Git.**

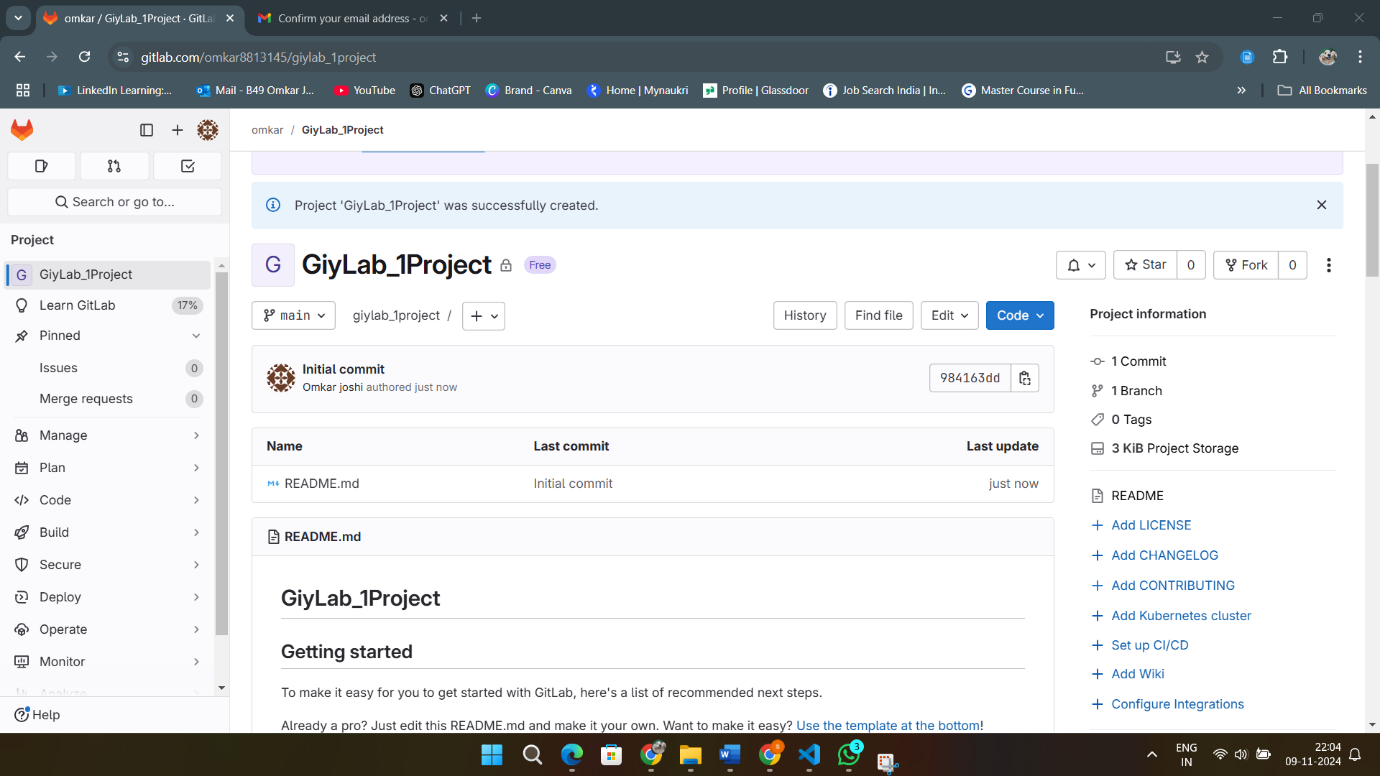
**Task 1: Creating a Repository**

● Sign in to your GitLab account.

● Click the "New" button to create a new project.

● Choose a project name, visibility level (public, private), and other settings.

● Click "Create project."

Command/output

**Task 2: Cloning a Repository**

● Open your terminal or command prompt.

● Navigate to the directory where you want to clone the repository.

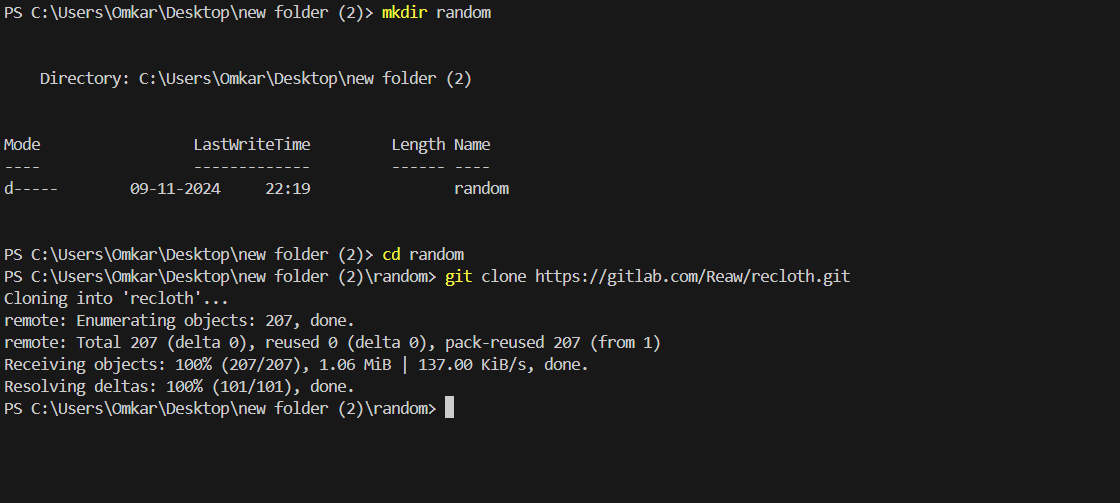
● Copy the repository URL from GitLab.

● Run the following command: git clone <repository\_url>

● Replace <repository\_url> with the URL you copied from GitLab.

● This will clone the repository to your local machine.

Command/output:



**Task 3: Making Changes and Creating a Branch**

● Navigate into the cloned repository: cd <repository\_name>

● Create a new text file named "example.txt" using a text editor.

● Add some content to the "example.txt" file.

● Save the file and return to the command line.

● Check the status of the repository: git status

● Stage the changes for commit: git add example.txt

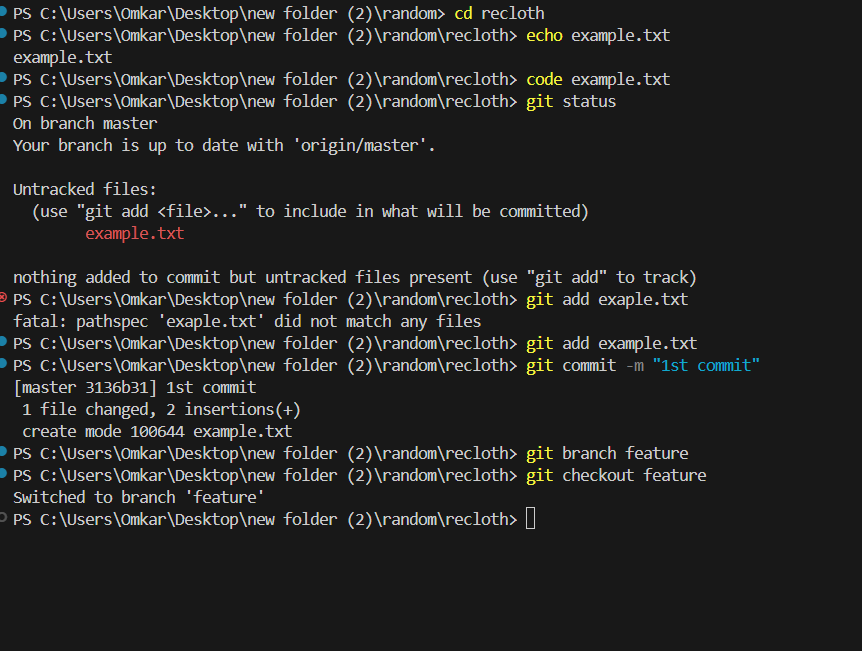
● Commit the changes with a descriptive message:

git commit -m "Add content to example.txt"

● Create a new branch named "feature": git branch feature

● Switch to the "feature" branch: git checkout feature

Command/output::



**Task 4: Pushing Changes to GitLab**

● Add Repository URL in a variable git remote add origin <repository\_url>

● Replace <repository\_url> with the URL you copied from GitLab.

● Push the "feature" branch to GitLab: git push origin feature

● Check your GitLab repository to confirm that the new branch "feature" is

available.

● After the merge request is merged, update your local repository:

git checkout main, git pull origin main

Command/output::

**Task 5: Collaborating through Merge Requests**

1. Create a merge request on GitLab:

 Go to the repository on GitLab.

 Click on "Merge Requests" and then "New Merge Request."

 Choose the source branch ("feature") and the target branch ("main" or master").

 Review the changes and click "Submit merge request."

2. Review and merge the merge request:

 Add a title and description for the merge request.

 Assign reviewers if needed.

 Once the merge request is approved, merge it into the target branch.

Task 6: Syncing Changes

4

Implement BitBucket Operations using Git.

Task 1: Creating a Repository

● Sign in to your Bitbucket account.

● Click the "Create" button to create a new repository.

● Choose a repository name, visibility (public or private), and other settings.

● Click "Create repository."

Task 2: Cloning a Repository

● Open your terminal or command prompt.

● Navigate to the directory where you want to clone the repository.

● Copy the repository URL from Bitbucket.

● Run the following command: git clone <repository\_url>

● Replace <repository\_url> with the URL you copied from Bitbucket.

● This will clone the repository to your local machine.

Task 3: Making Changes and Creating a Branch

● Navigate into the cloned repository: cd <repository\_name>

● Create a new text file named "example.txt" using a text editor.

● Add some content to the "example.txt" file.

● Save the file and return to the command line.

● Check the status of the repository: git status

33

SPPU

M.Sc. Computer Science Part-II Syllabus 2024-25

● Stage the changes for commit: git add example.txt

● Commit the changes with a descriptive message: git commit -m "Add content to

example.txt"

● Create a new branch named "feature": git branch feature

● Switch to the "feature" branch: git checkout feature

Task 4: Pushing Changes to Bitbucket

● Add Repository URL in a variable: git remote add origin <repository\_url>

● Replace <repository\_url> with the URL you copied from Bitbucket.

● Push the "feature" branch to Bitbucket: git push origin feature

● Check your Bitbucket repository to confirm that the new branch "feature" is

available.

Task 5: Collaborating through Pull Requests

1. Create a pull request on Bitbucket:

● After the pull request is merged, update your local repository:

git checkout main, git pull origin main

○ Go to the repository on Bitbucket. ○ Click on "Create pull request."

○ Choose the source branch ("feature") and the target branch ("main" or "master").

○ Review the changes and click "Create pull request."

2. Review and merge the pull request:

○ Add a title and description for the pull request.

○ Assign reviewers if needed.

○ Once the pull request is approved, merge it into the target branch.

Task 6: Syncing Changes

5 Applying CI/CD Principles to Web Development Using Jenkins, Git, and Local

HTTP Server

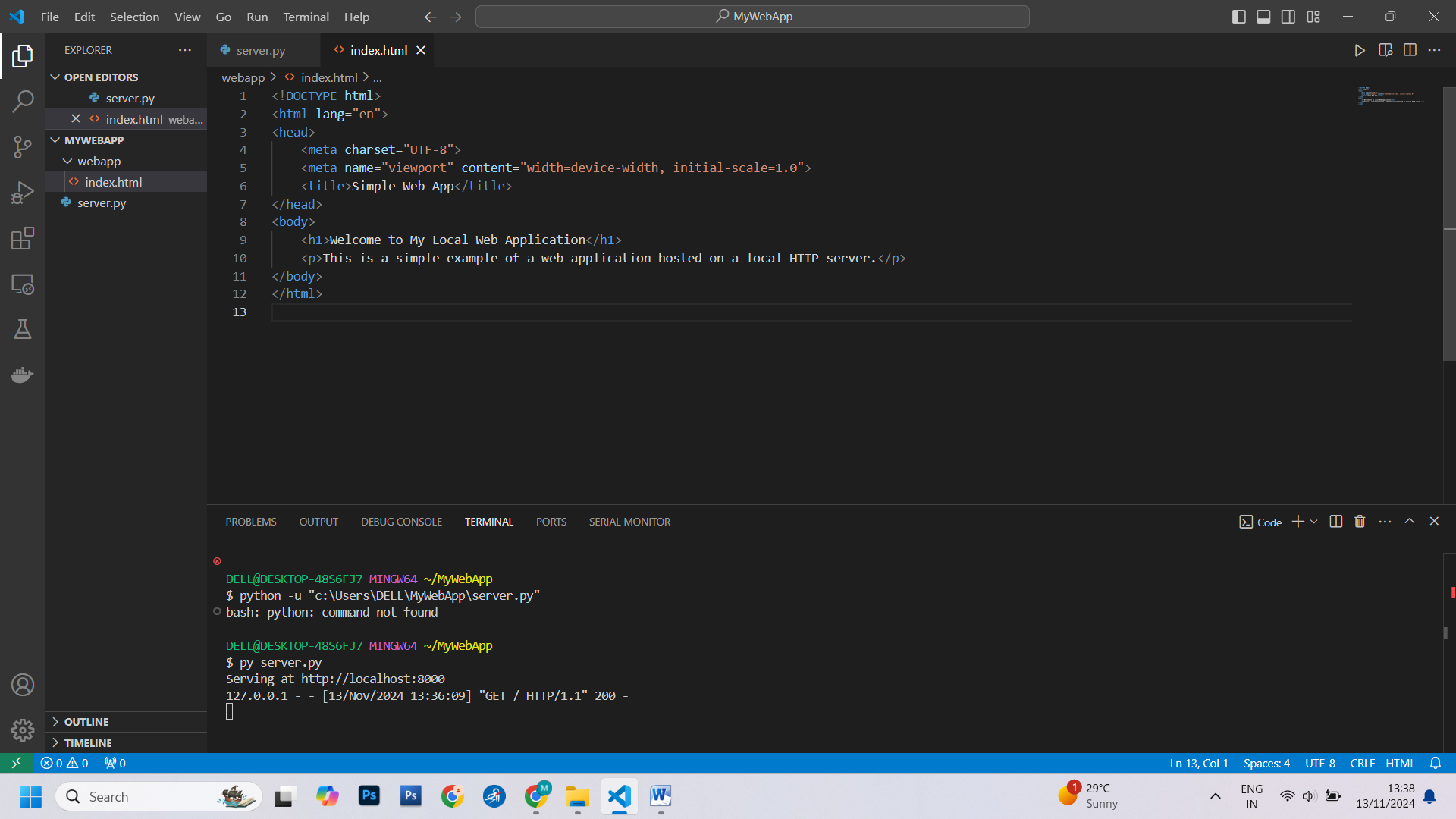
Step 1: Set Up the Web Application and Local HTTP Server

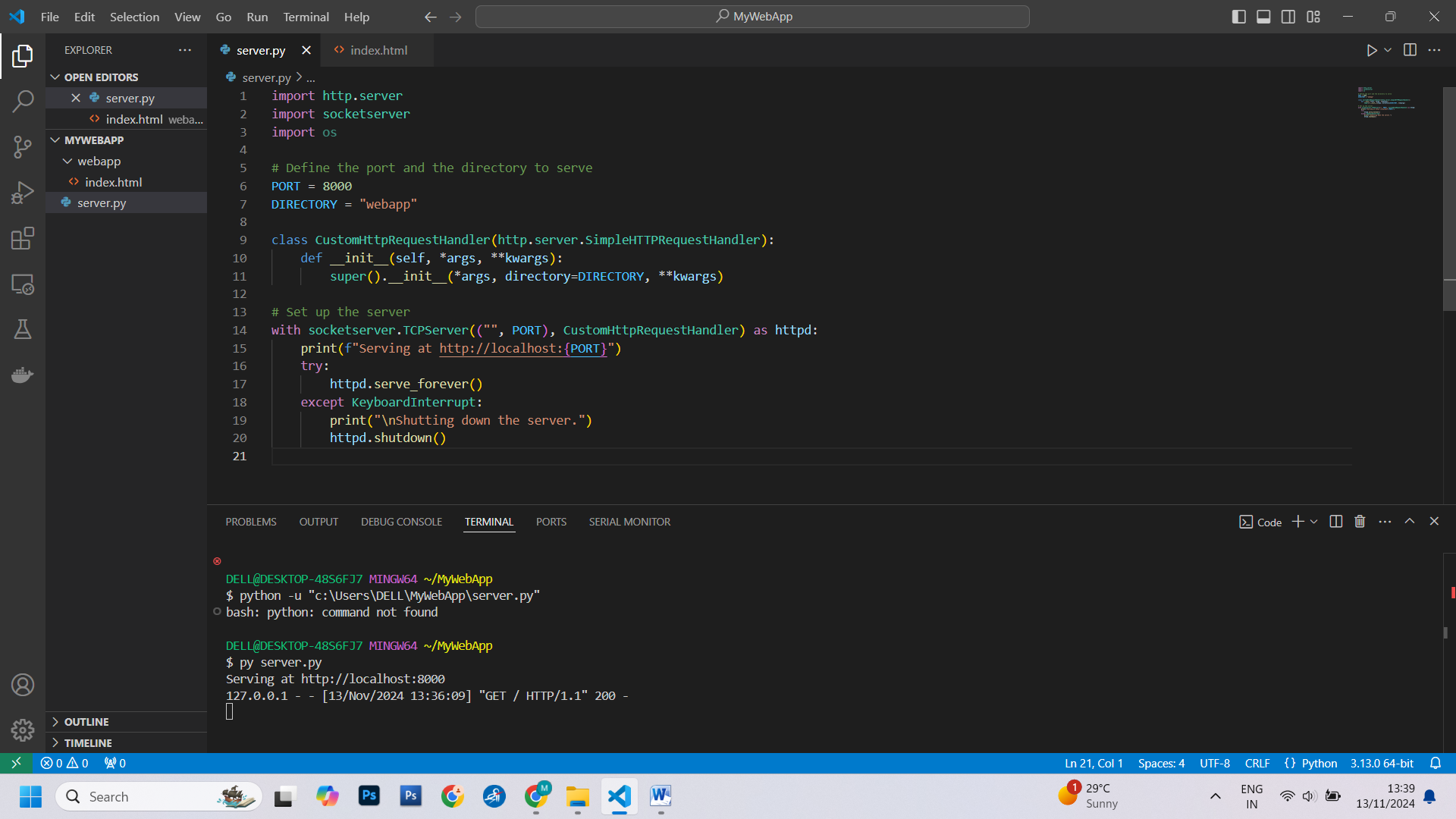
● Create a simple web application or use an existing one. Ensure it can be hosted

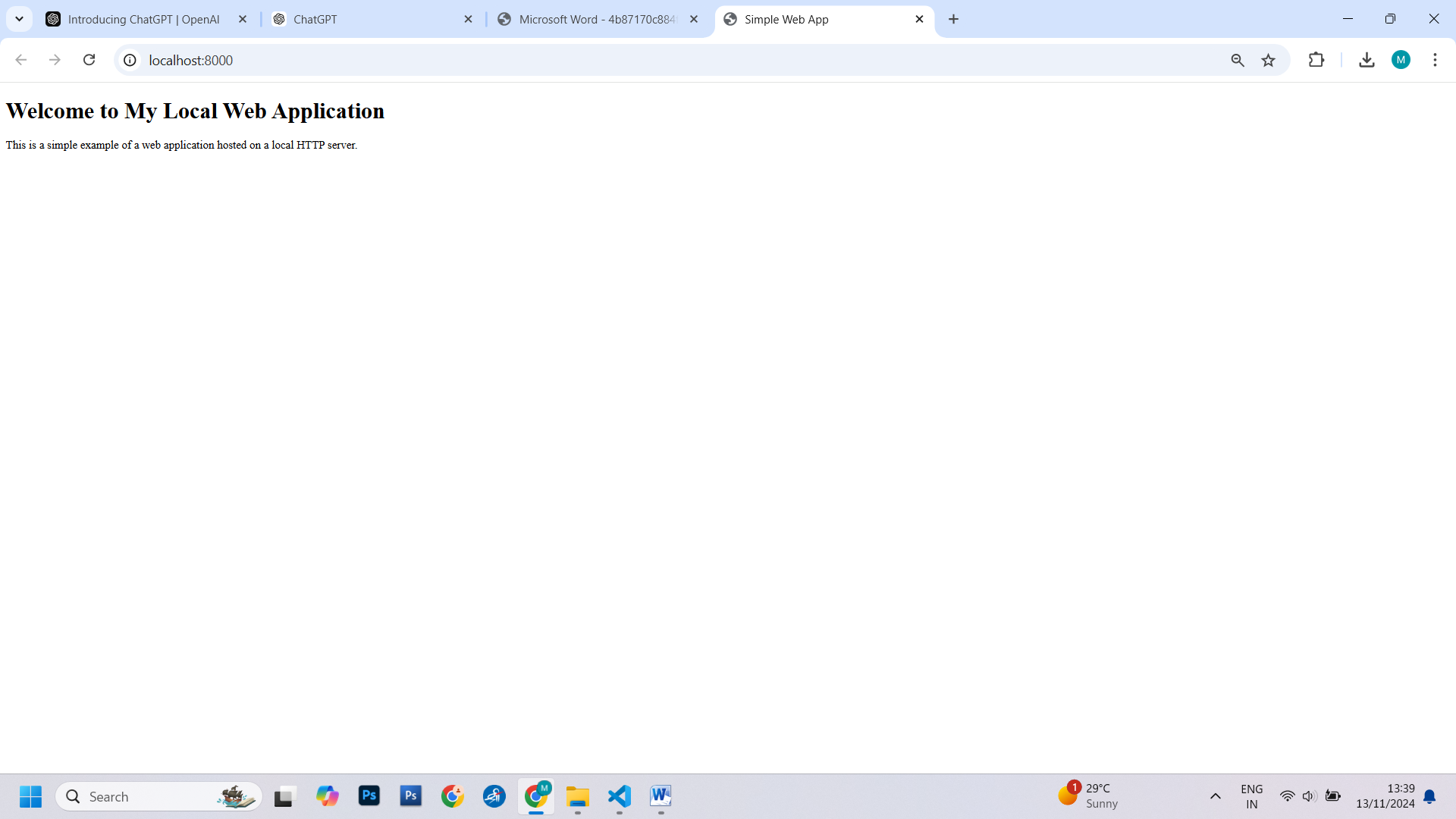
by a local HTTP server.

● Set up a local HTTP server (e.g., Apache or Nginx) to serve the web application.

Ensure it's configured correctly and running.



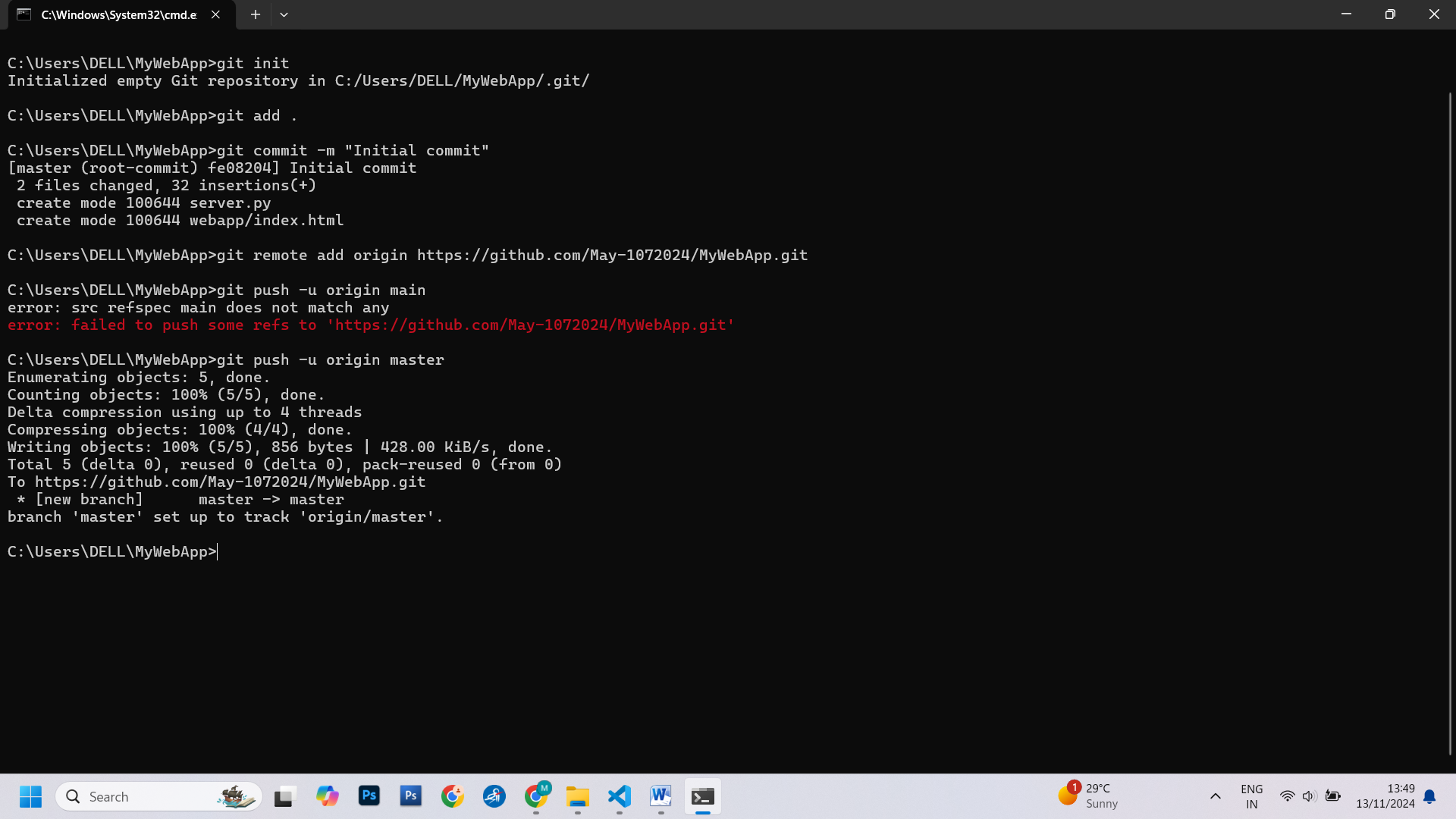




Step 2: Set Up a Git Repository

Create a Git repository for your web application. Initialize it with the following

commands: git init, git add, git commit -m "Initial commit"

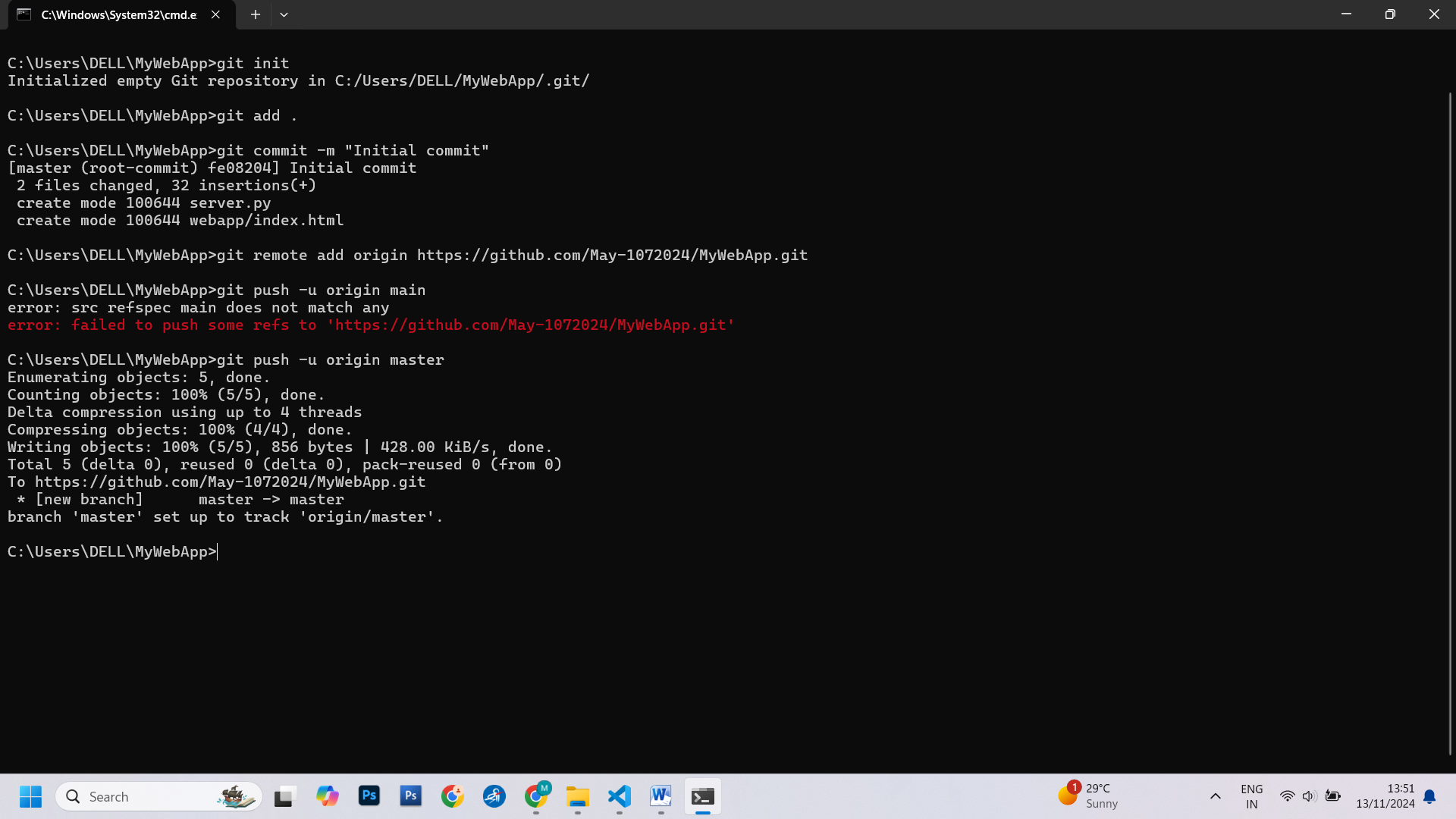


● Configure Jenkins to work with Git by setting up your Git credentials in the

Jenkins Credential Manager.

● Create a remote Git repository (e.g., on GitHub or Bitbucket) to push your code

to later.



Step 3: Install and Configure Jenkins

● Download and install Jenkins following the instructions for your operating

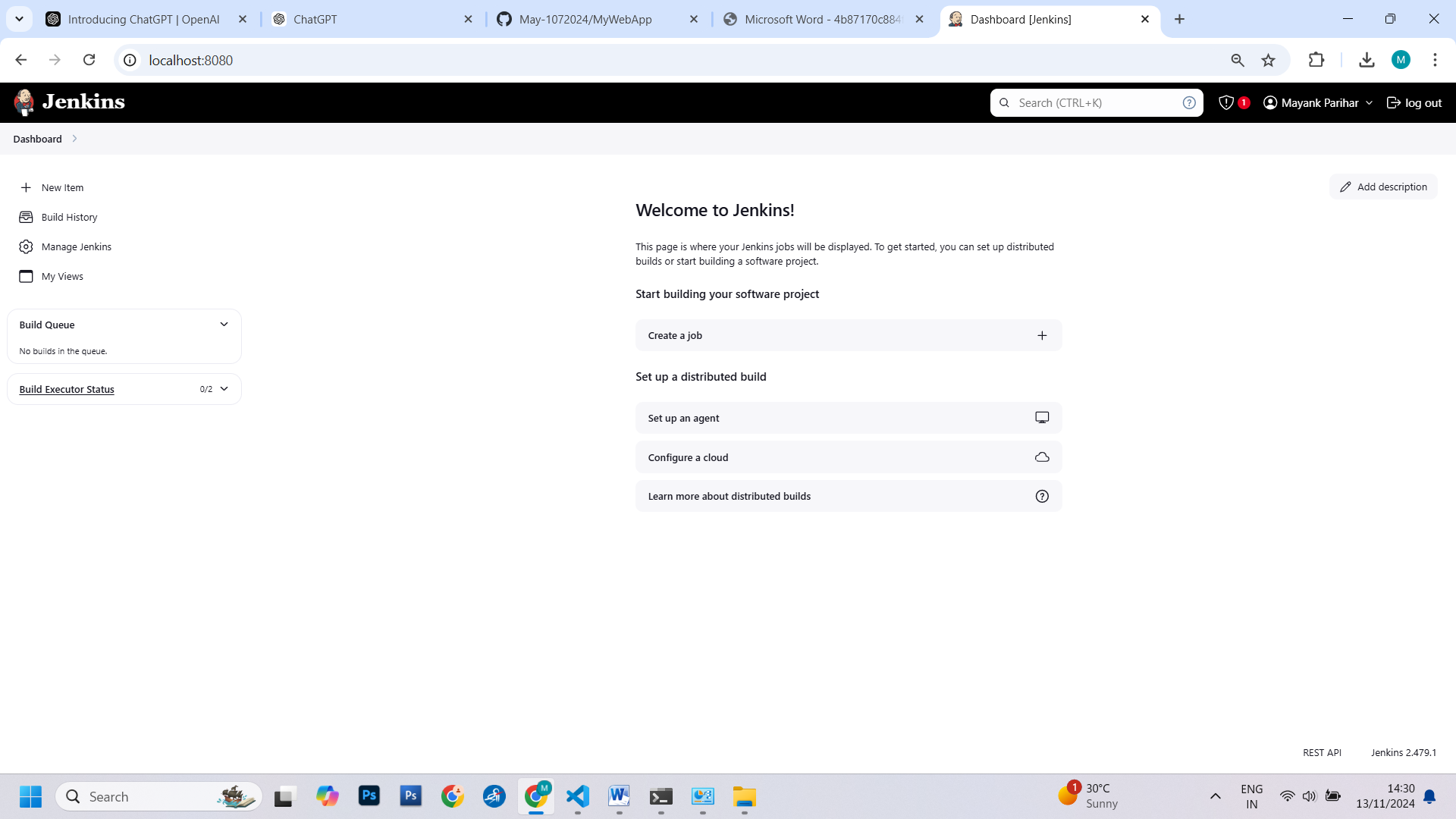
system (https://www.jenkins.io/download/).

● Open Jenkins in your web browser (usually at http://localhost:8080) and

complete the initial setup.

● Install the necessary plugins for Git integration, job scheduling, and webhook

support.



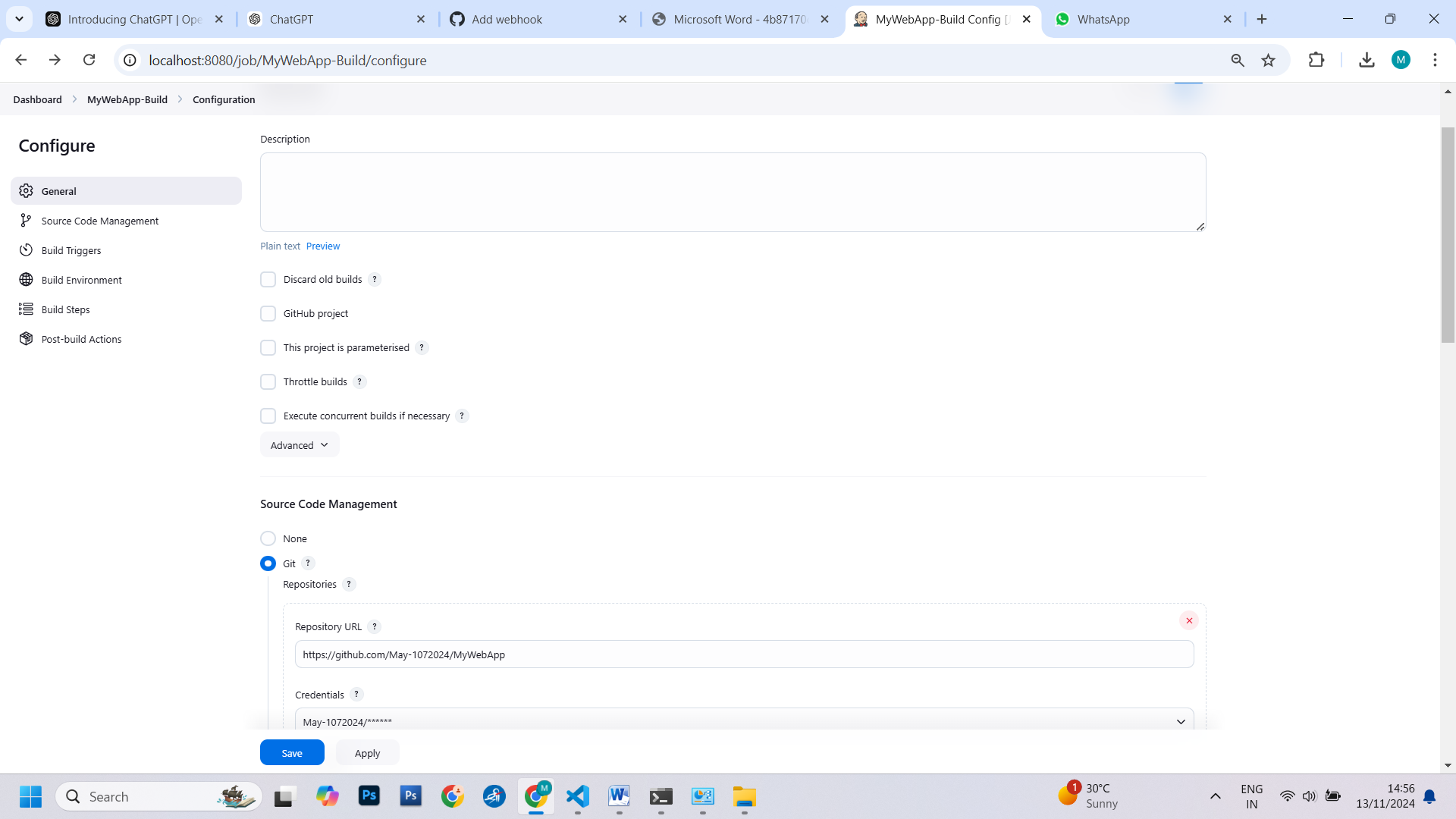
Step 4: Create a Jenkins Job

● Create a new Jenkins job using the "Freestyle project" type.

● Configure the job to use a webhook trigger. You can do this by selecting the

"GitHub

hook trigger for GITScm polling" option in the job's settings.





Step 5: Set Up the Jenkins Job (Using Execute Shell)

● In the job configuration, go to the "Build" section.

● Add a build step of type "Execute shell."

● In the "Command" field, define the build and deployment steps using shell

commands. For example:

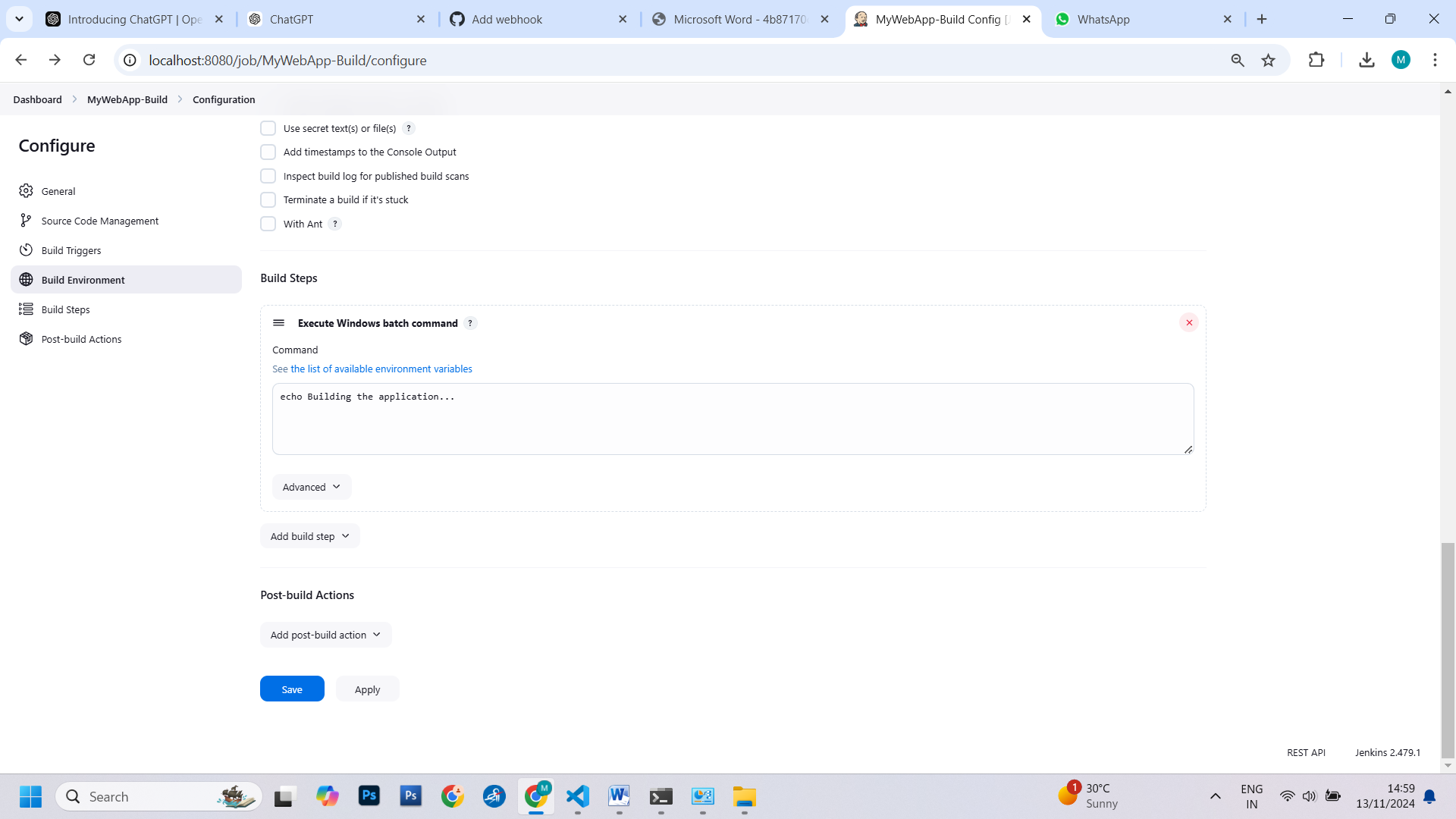
# Checkout code from Git

# Build your web application (e.g., npm install, npm run build)

# Copy the build artefacts to the local HTTP server directory

rm -rf /var/www/html/webdirectory/\*

cp -r \* /var/www/html/webdirectory/



Step 6: Set Up a Webhook in Git Repository

Visit the URL of your local HTTP server to verify that the web application has

been updated with the latest changes.

● In your Git repository (e.g., on GitHub), go to "Settings" and then "Webhooks."

● Create a new webhook, and configure it to send a payload to the Jenkins

webhook

URL (usually http://jenkins-server/github-webhook/). Make sure to set the content

type to "application/json."

● OR use “GitHub hook trigger for GITScm polling?” Under Build Trigger

Step 7: Trigger the CI/CD Pipeline

● Push changes to your Git repository. The webhook should trigger the Jenkins job

automatically, executing the build and deployment steps defined in the "Execute

Shell" build step.

● Monitor the Jenkins job's progress in the Jenkins web interface.

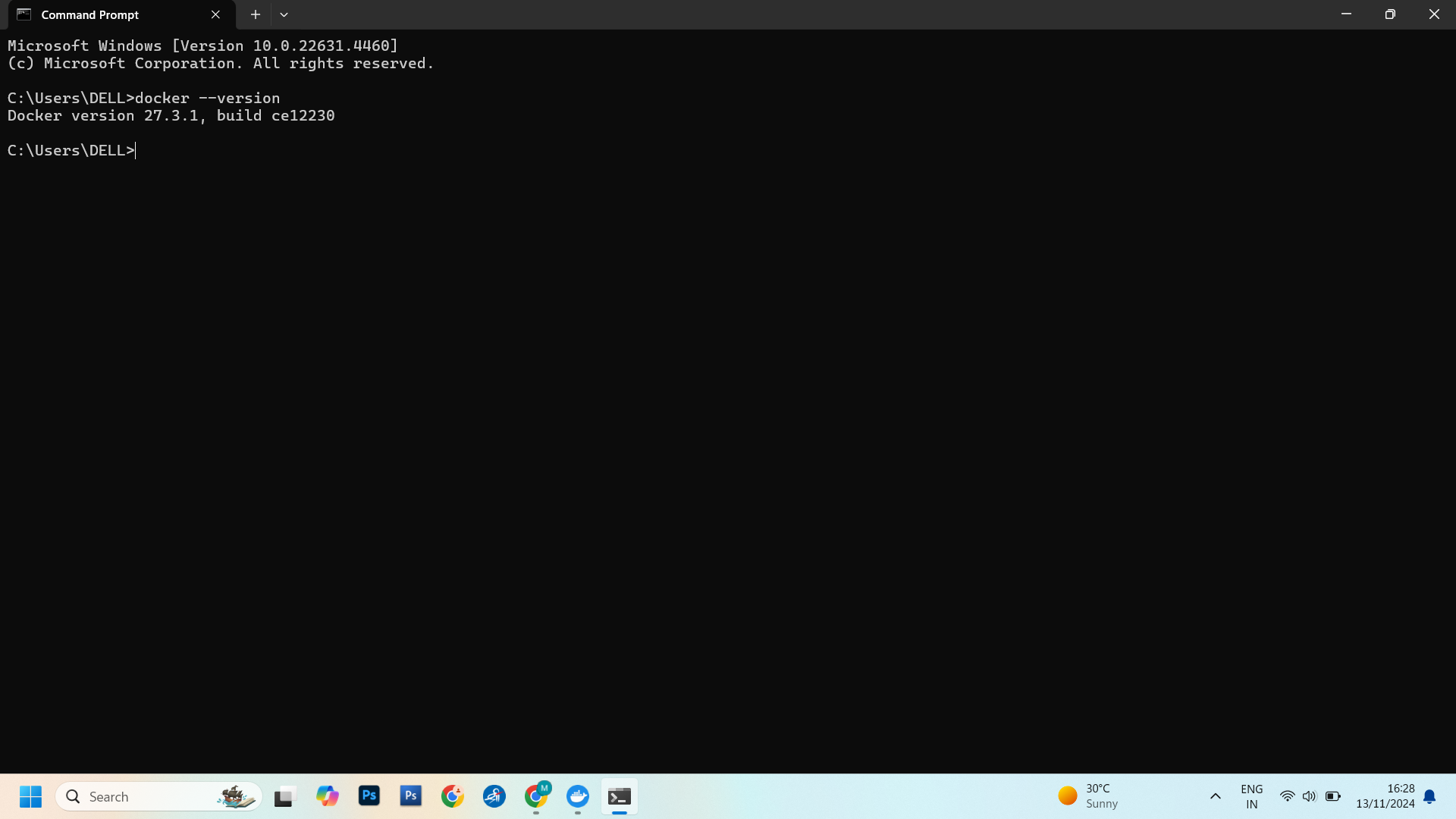
Step 8: Verify the CI/CD Pipeline

6 Exploring Containerization and Application Deployment with Docker

Task 1: Install Docker

● If you haven't already, install Docker on your computer by following the

instructions provided on the Docker website (https://docs.docker.com/get-docker/).

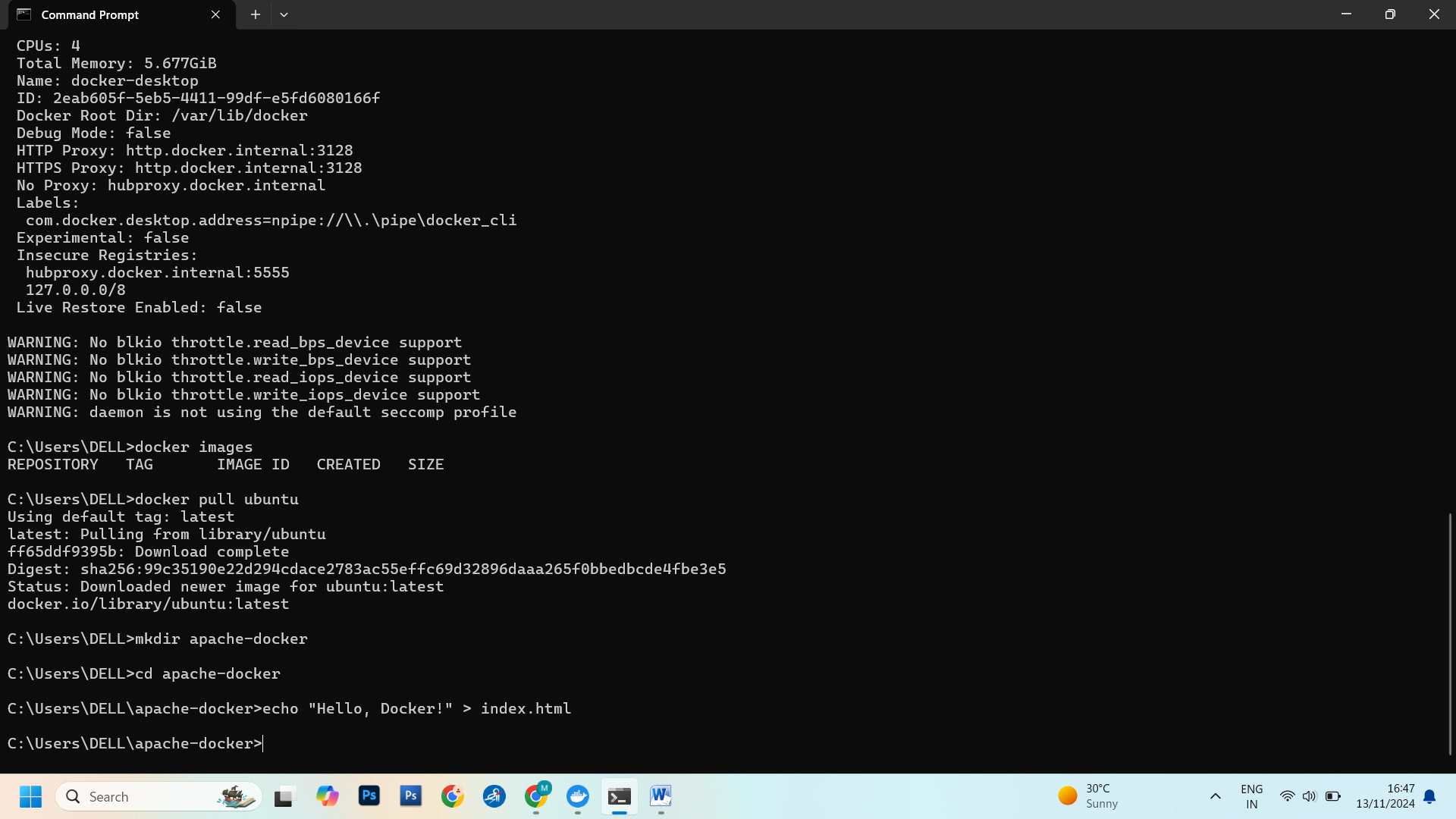


Task 2: Create a Simple HTML Page

● Create a directory for your web server project.

● Inside this directory, create a file named index.html with a simple "Hello,

Docker!" message. This will be the content served by your Apache web server.



Task 3: Create a Dockerfile

● Create a Dockerfile in the same directory as your web server project. The

Dockerfile defines how your Apache web server application will be packaged into a

Docker container.

Here's an example: //teacher can use other example for demo

Dockerfile

# Use an official Apache image as the base image

FROM httpd:2.4

# Copy your custom HTML page to the web server's document root

COPY index.html /usr/local/apache2/htdocs/

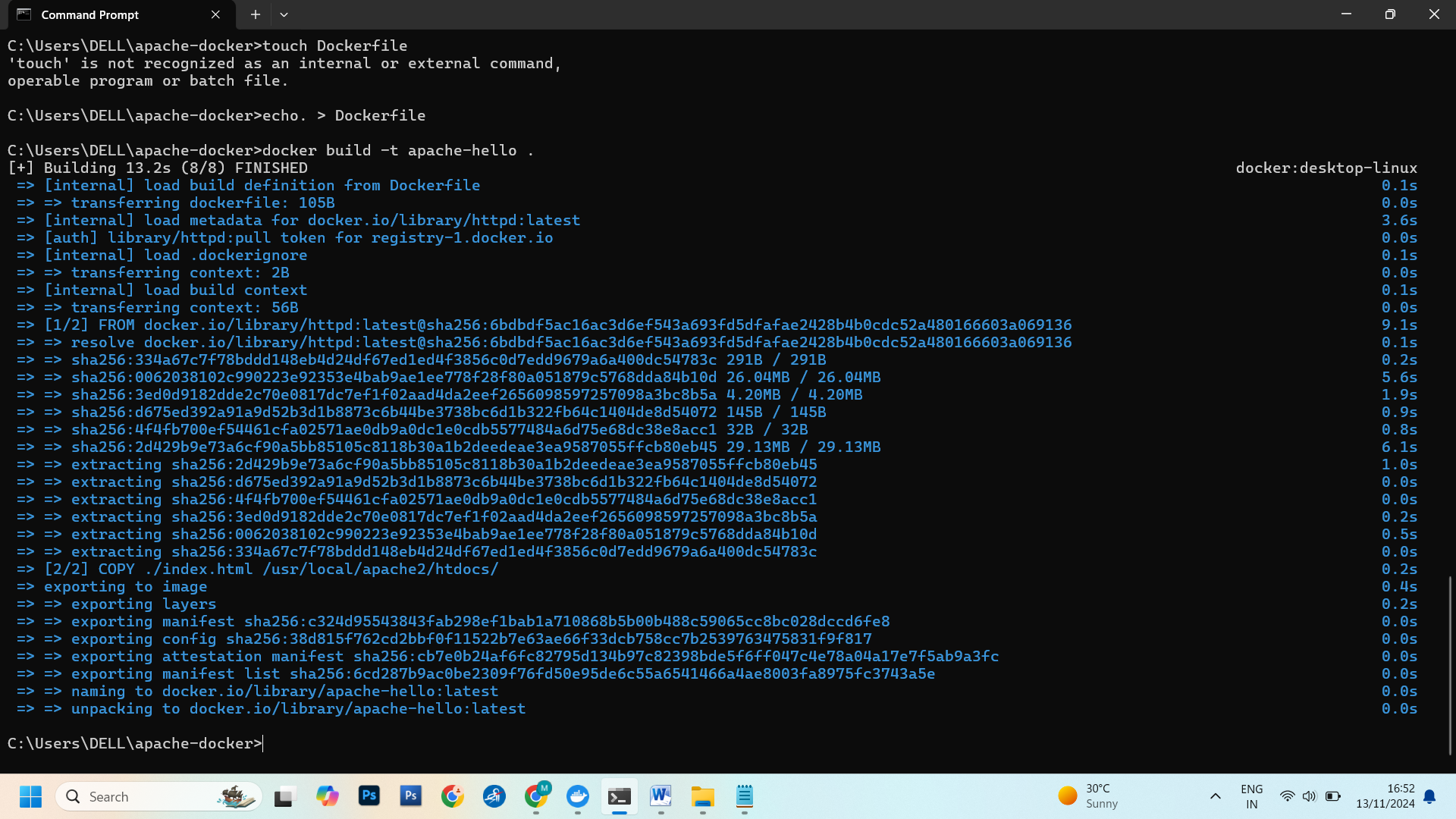


Task 4: Build the Docker Image

● Build the Docker image by running the following command in the same directory

as your Dockerfile: docker build -t my-apache-server .

● Replace my-apache-server with a suitable name for your image.



Task 5: Run the Docker Container

Start a Docker container from the image you built:

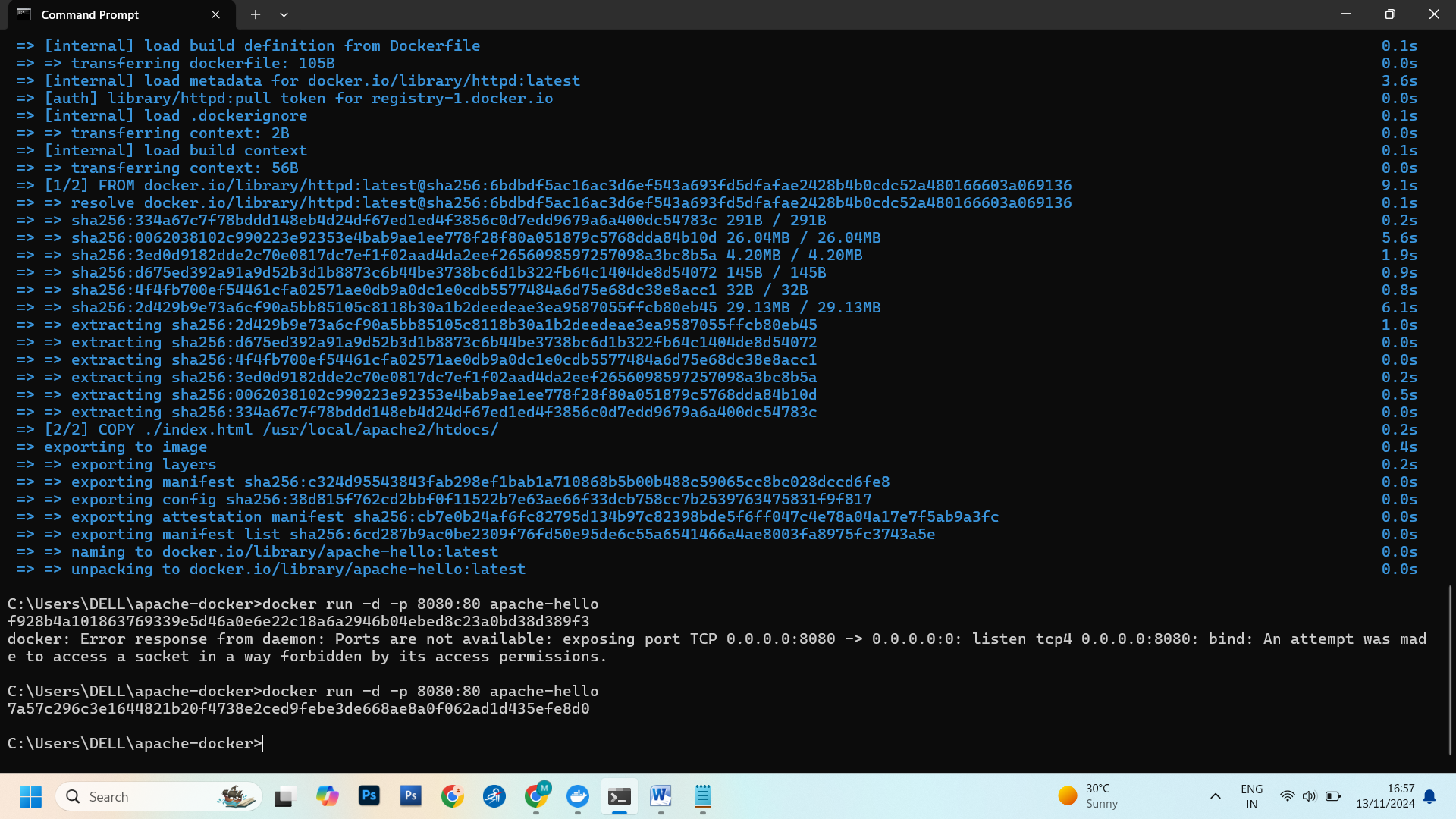
docker run -p 8080:80 -d my-apache-server

● Optionally, remove the container and the Docker image:

docker rm <container\_id>, docker rmi my-apache-server

● This command maps port 80 in the container to port 8080 on your host machine

and runs the container in detached mode.

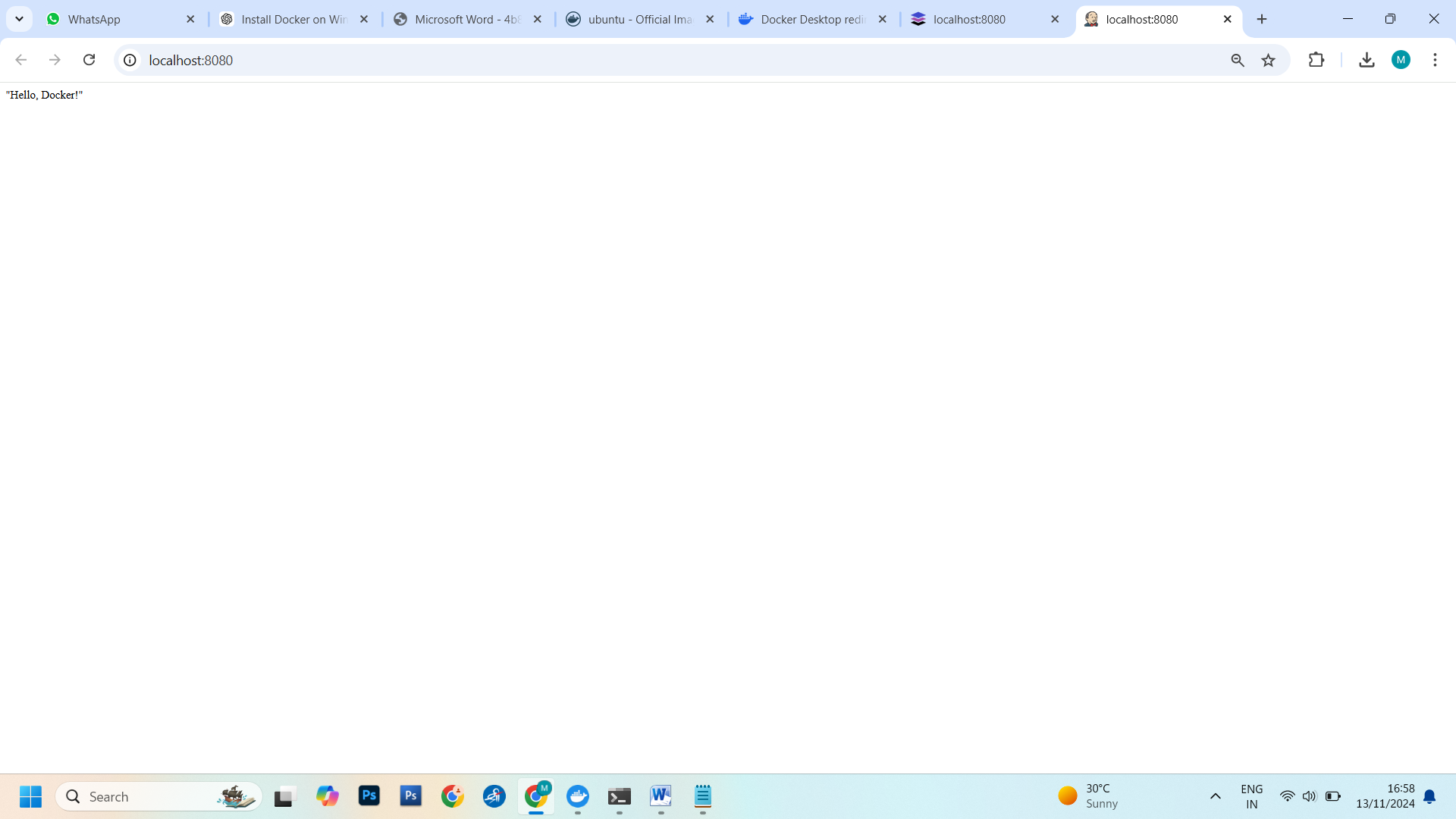


Task 6: Access Your Apache Web Server

Access your Apache web server by opening a web browser and navigating to

http://localhost:8080. You should see the "Hello, Docker!" message served by your

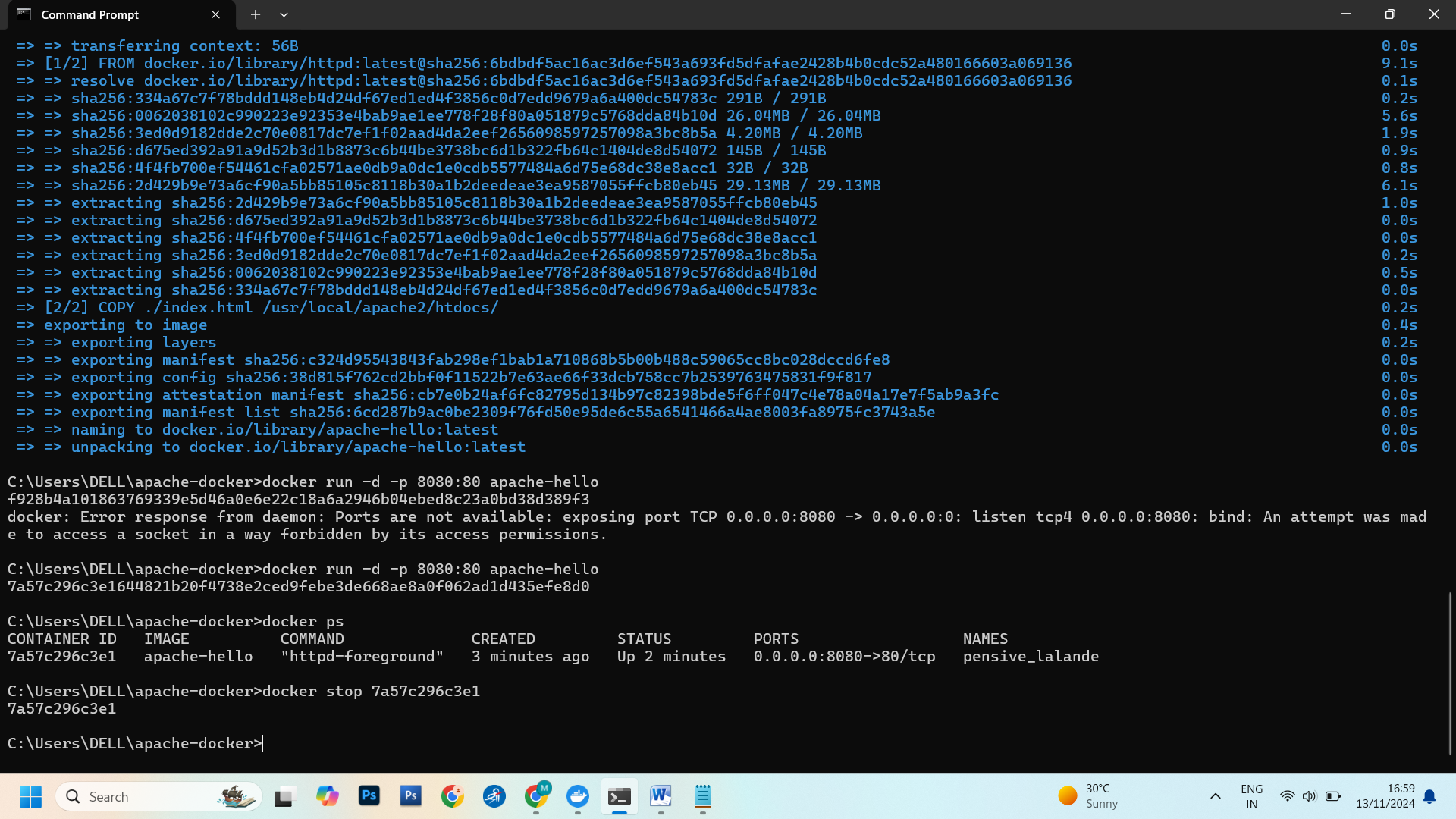
Apache web server running within the Docker container.



Task 7: Cleanup

Stop the running Docker container: docker stop <container\_id>

● Replace <container\_id> with the actual ID of your running container.



7 Applying CI/CD Principles to Web Development Using Jenkins, Git,

usingDocker Containers

Task 1: Set Up the Web Application and Git Repository

● Create a simple web application or use an existing one. Ensure it can be hosted in

a Docker container.

● Initialise a Git repository for your web application and push it to GitHub.

Task 2: Install and Configure Jenkins

● Install Jenkins on your computer or server following the instructions for your

operating system (https://www.jenkins.io/download/).

● Open Jenkins in your web browser (usually at http://localhost:8080) and

complete the initial setup, including setting up an admin user and installing

necessary plugins.

● Set Branches to build -> Branch Specifier to the working Git branch (ex

\*/master)

● Configure Jenkins to work with Git by setting up Git credentials in the Jenkins

Credential Manager.

Task 3: Create a Jenkins Job

● Create a new Jenkins job using the "Freestyle project" type.

● In the job configuration, specify a name for your job and choose "This project is

parameterized."

● Add a "String Parameter" named GIT\_REPO\_URL and set its default value to

your Git repository URL.

● In the job configuration, go to the "Build Triggers" section and select the

"GitHub hook trigger for GITScm polling" option. This enables Jenkins to listen for

GitHub webhook triggers.

Task 4: Configure Build Steps

● In the job configuration, go to the "Build" section.

● Add build steps to execute Docker commands for building and deploying the

containerized web application. Use the following commands:

# Remove the existing container if it exists: docker rm --force container1

# Build a new Docker image: docker build -t nginx-image1 .

# Run the Docker container: docker run -d -p 8081:80 --name=container1 nginx

image1

Access your web application by opening a web browser and navigating to

http://localhost:8081 (or the appropriate URL if hosted elsewhere).

● These commands remove the existing container (if any), build a Docker image

named "nginx-image1," and run a Docker container named "container1" on port

8081.

Task 5: Set Up a GitHub Webhook

● In your GitHub repository, navigate to "Settings" and then "Webhooks."

● Create a new webhook, and configure it to send a payload to the Jenkins

webhook URL (usually http://jenkins-server/github-webhook/). Set the content

type to "application/json."

Task 6: Trigger the CI/CD Pipeline

● Push changes to your GitHub repository. The webhook will trigger the Jenkins

job automatically, executing the build and deployment steps defined in the job

configuration.

● Monitor the Jenkins job's progress in the Jenkins web interface.

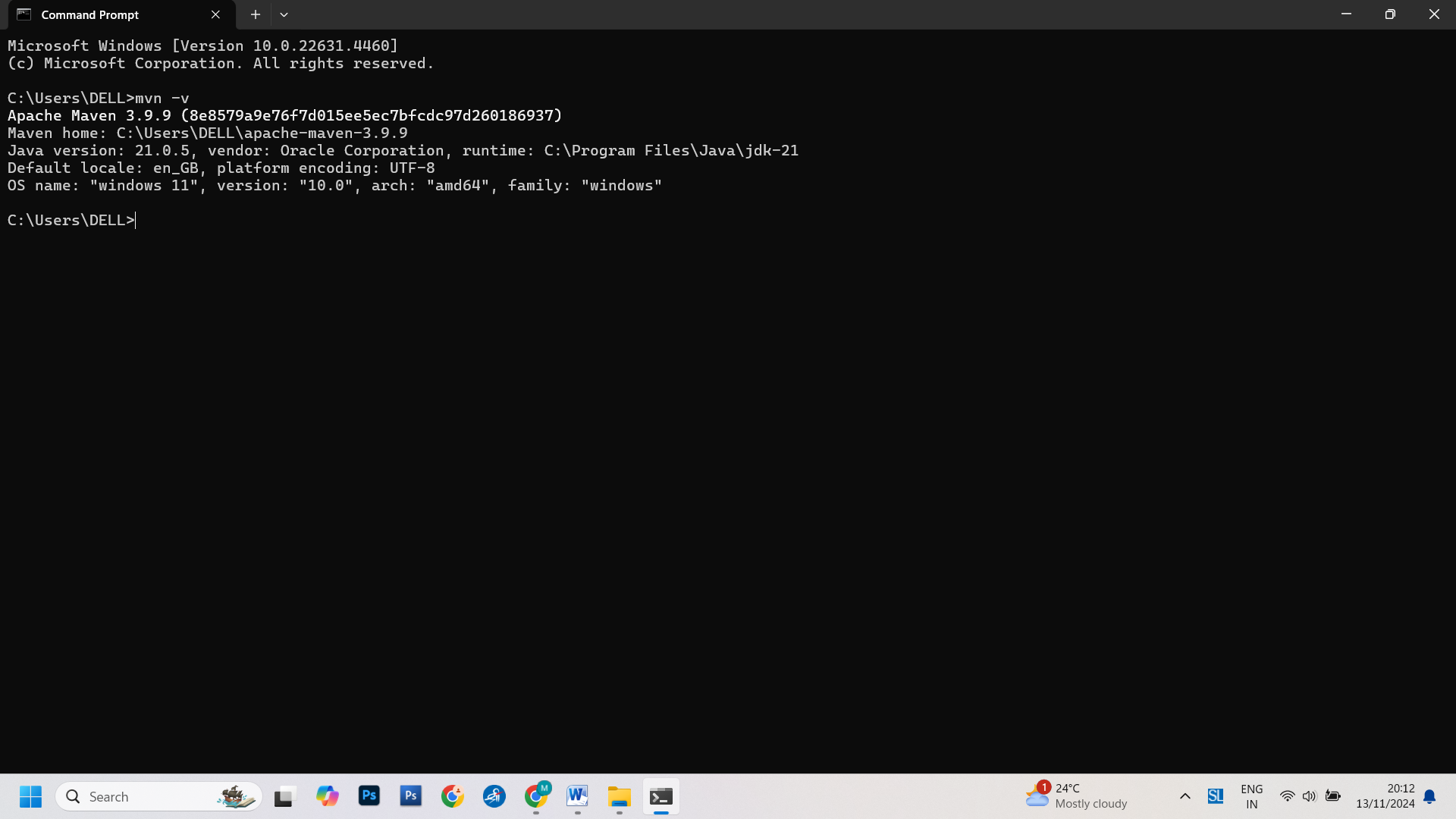
Task 7: Verify the Deployment

8 Practical Maven Assignment

Task 1: Setting up Maven Environment

 installing Maven on local machines and configuring it properly.

 understand the directory structure and how to create a basic Maven project.



Task 2: Creating a Simple Maven Project

 create a simple Java project using Maven.

 This could involve creating classes, adding dependencies, and configuring the

project's POM (Project Object Model).

Task 3: Dependency Management

 Introduce dependency management in Maven.

 Assign tasks such as adding external dependencies to the project using Maven

Central repository and managing version conflicts.

Task 4: Build Automation

 set up automated builds using Maven.

 configure Maven to compile the code, run tests, and generate artifacts like

JAR files.

