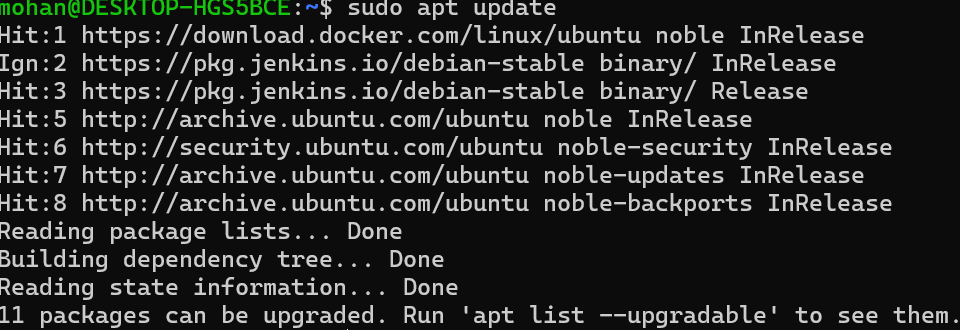
**DEVOPS ASSIGNMENT 2**

**DAY 2**

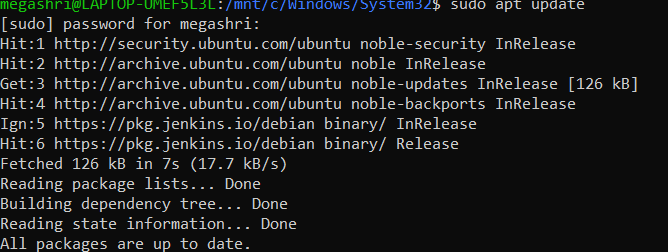
**STEP1 :**

The command sudo apt update, which updates the package lists for available updates and repositories on an Ubuntu system. The output includes information about the sources being updated, including the Jenkins repository and various components from the Ubuntu archives.



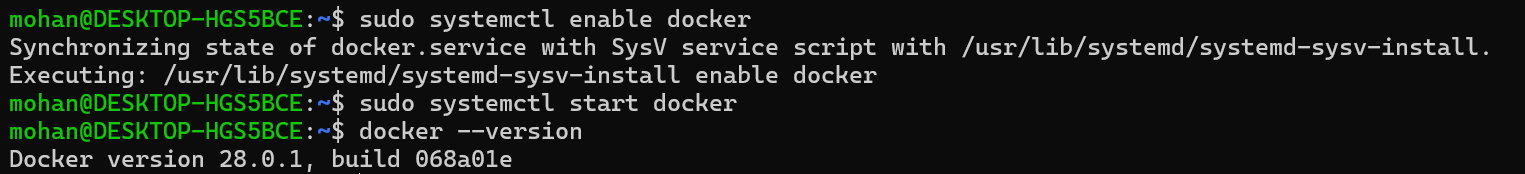
**STEP 2:**

The terminal output indicates that the user attempted to install Docker on Ubuntu using `sudo apt install -y docker.io`. It shows that Docker is already at the newest version, with no upgrades available for other packages.



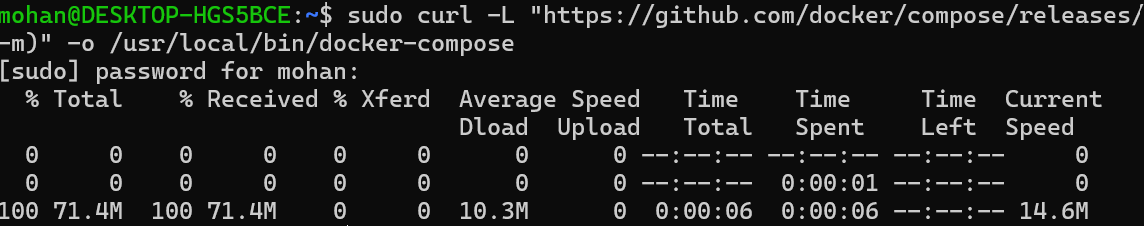
**STEP 3 :**

The commands enabling and starting the Docker service (systemctl enable/start docker) and verifying the installation using docker --version, confirming Docker 26.1.3 on Ubuntu 24.04.



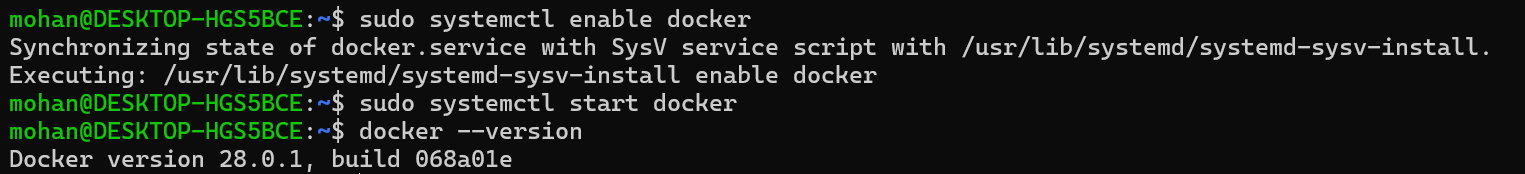
**STEP 4 :**

The terminal command using curl to download the latest Docker Compose binary from GitHub and save it to /usr/local/bin/docker-compose. The progress bar indicates a successful download of **71.4MB** at a speed of **756kB/s**.



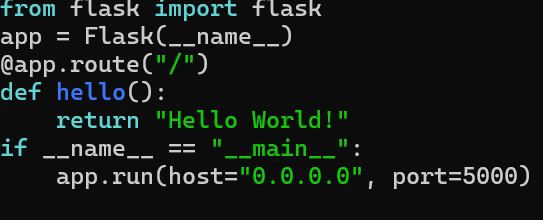
**STEP 5 :**

The commands making Docker Compose executable (chmod +x), verifying its installation (docker-compose --version), creating a ~/devops directory, and navigating into it.



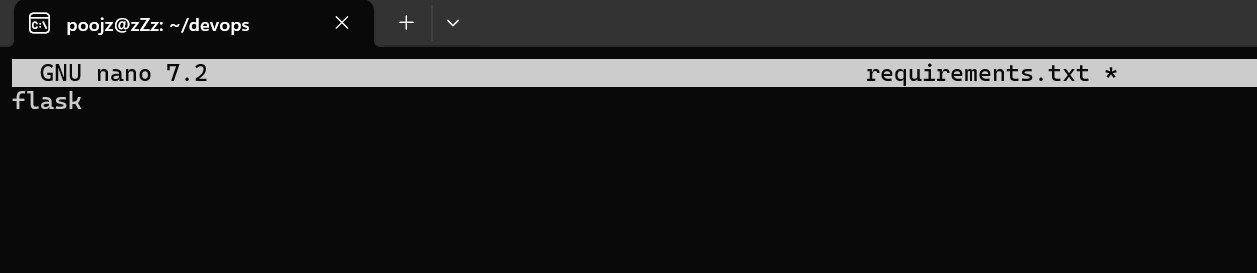
**STEP 6 :**

The Flask application (app.py) being created using the nano editor, defining a simple web server that returns "Hello, World! Running inside Docker!" on port 90.



**STEP 7:**

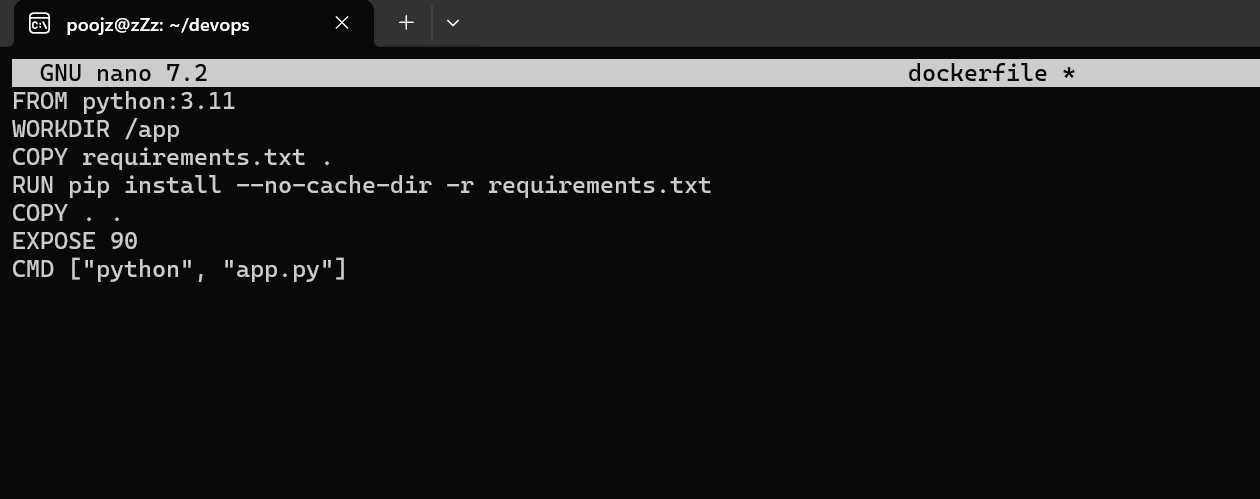
The creation of a requirements.txt file using the nano editor, listing flask as a dependency for the Python project. Note: There's a typo in the filename (requiremnts.txt).



**STEP 8:**

The Dockerfile being created using nano, defining a containerized environment for a Python 3.11

Flask app by copying dependencies, installing them, exposing port 90, and running app.py.



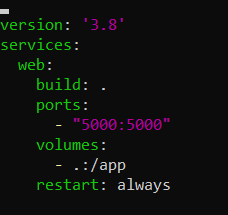
**STEP 9 :**

The docker-compose.yml file being created using nano, defining a service named web that builds from the current directory, maps port 5000:5000, mounts a volume, and restarts automatically.

**STEP 10 :**

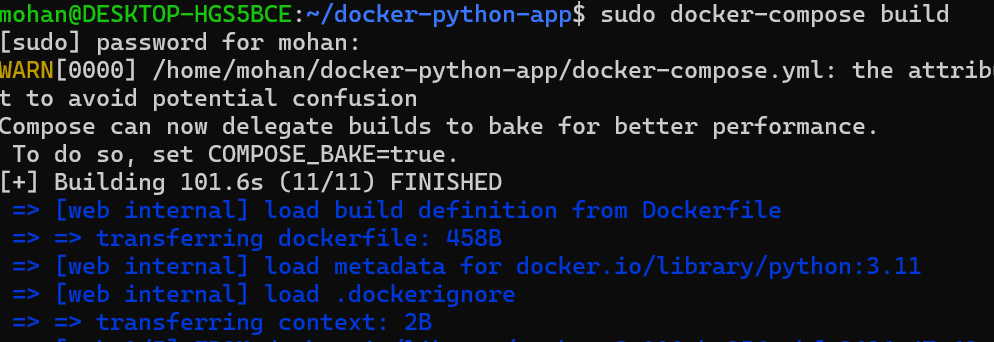
The image shows the output of sudo docker images, listing three Docker images: test (1.03GB, created

20 hours ago), nginx (192MB, 5 weeks old), and python:3.11 (1.01GB, 3 months old)



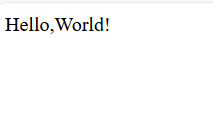
**STEP 11 :**

The image shows the execution of sudo docker-compose up --build, where Docker Compose is building an image from a docker-compose.yml file. A warning indicates that the version attribute is obsolete, and the build steps confirm that the required files and dependencies are successfully cached.



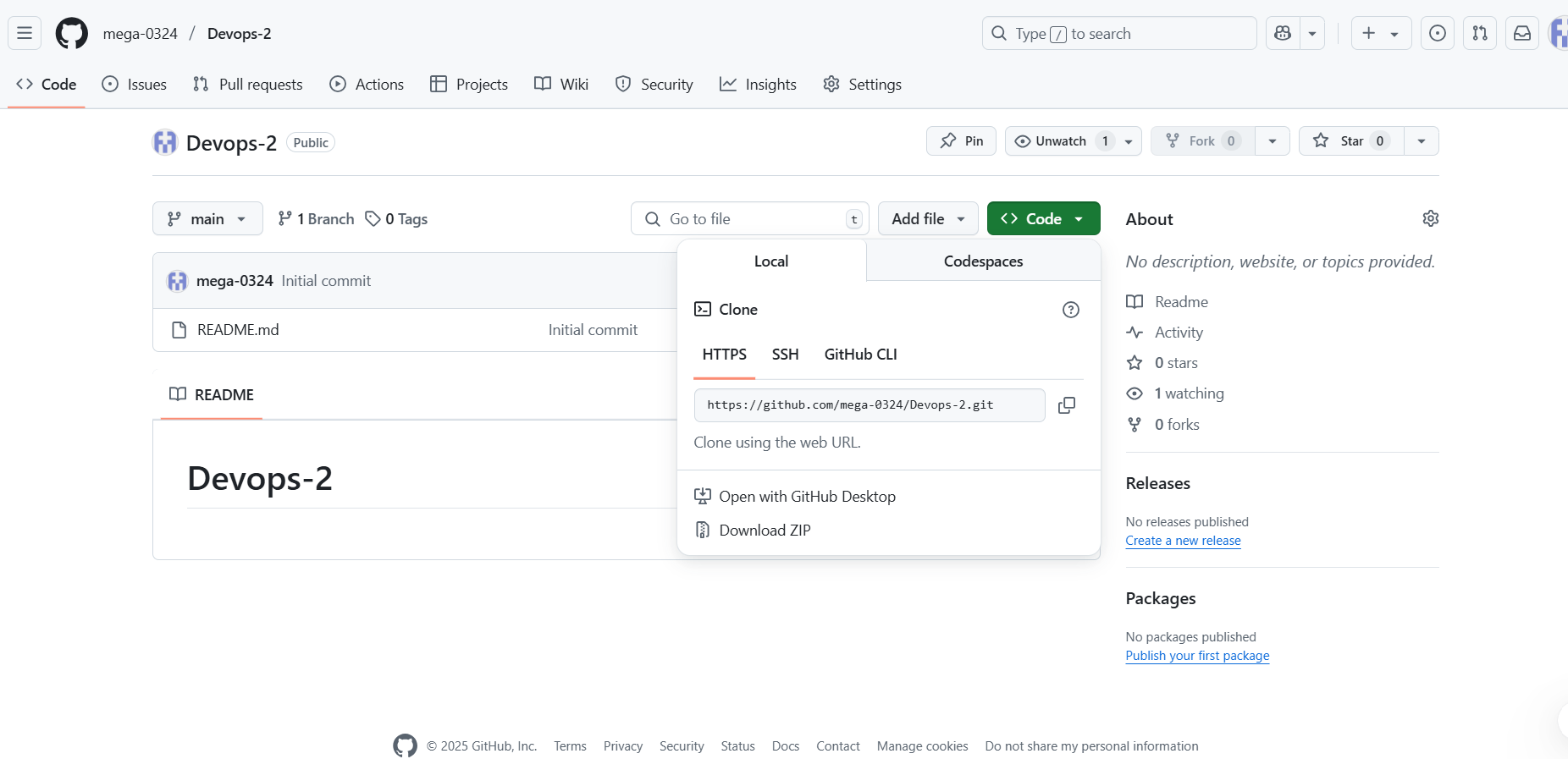
**OUTPUT:**

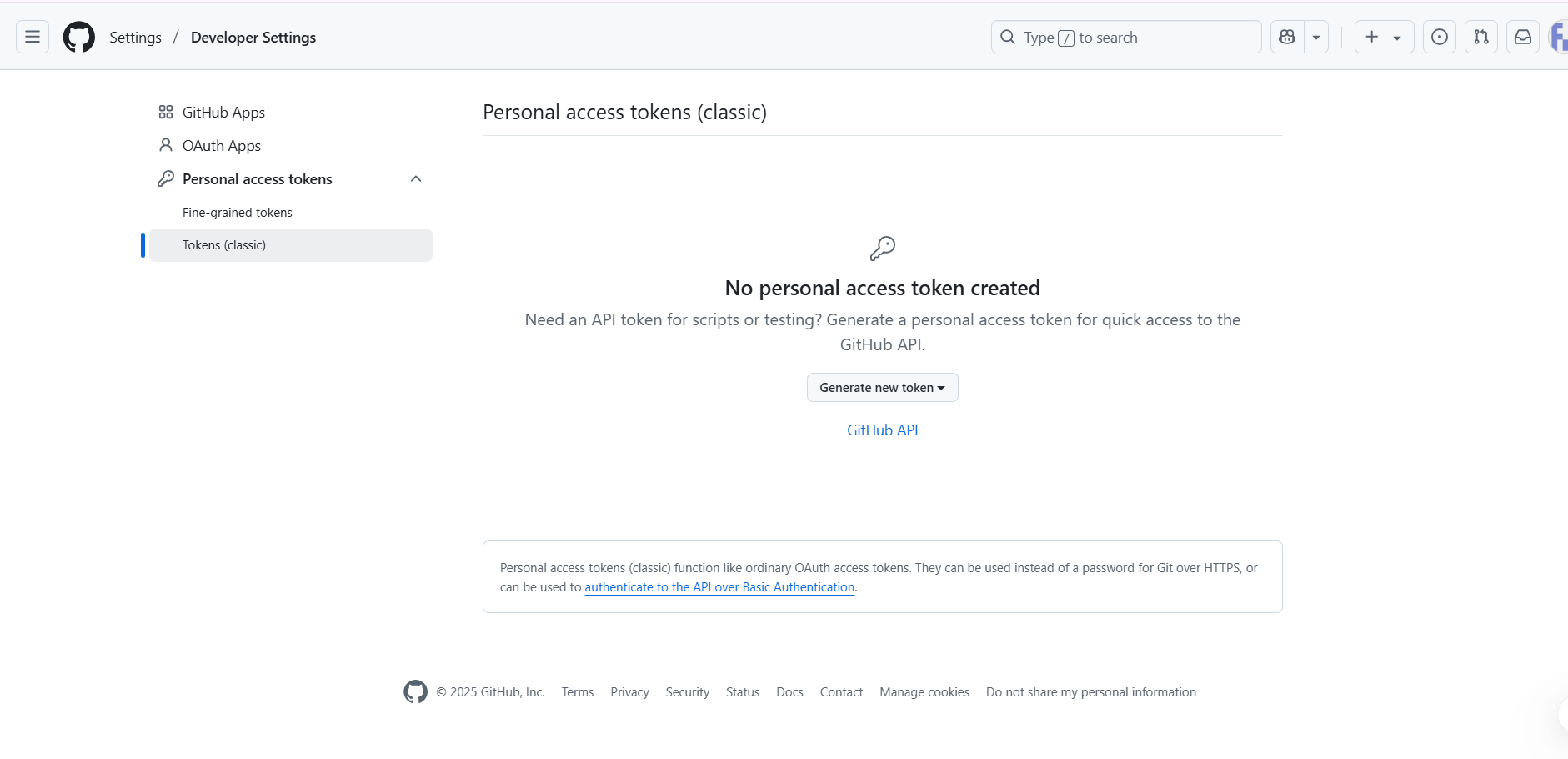
The final output shows a web browser displaying the message "Hello, World! Running inside Docker!" at localhost:90, confirming that the Flask application successfully runs inside a Docker container. This validates the containerization process, including building the Docker image, running the container, and exposing the correct port.

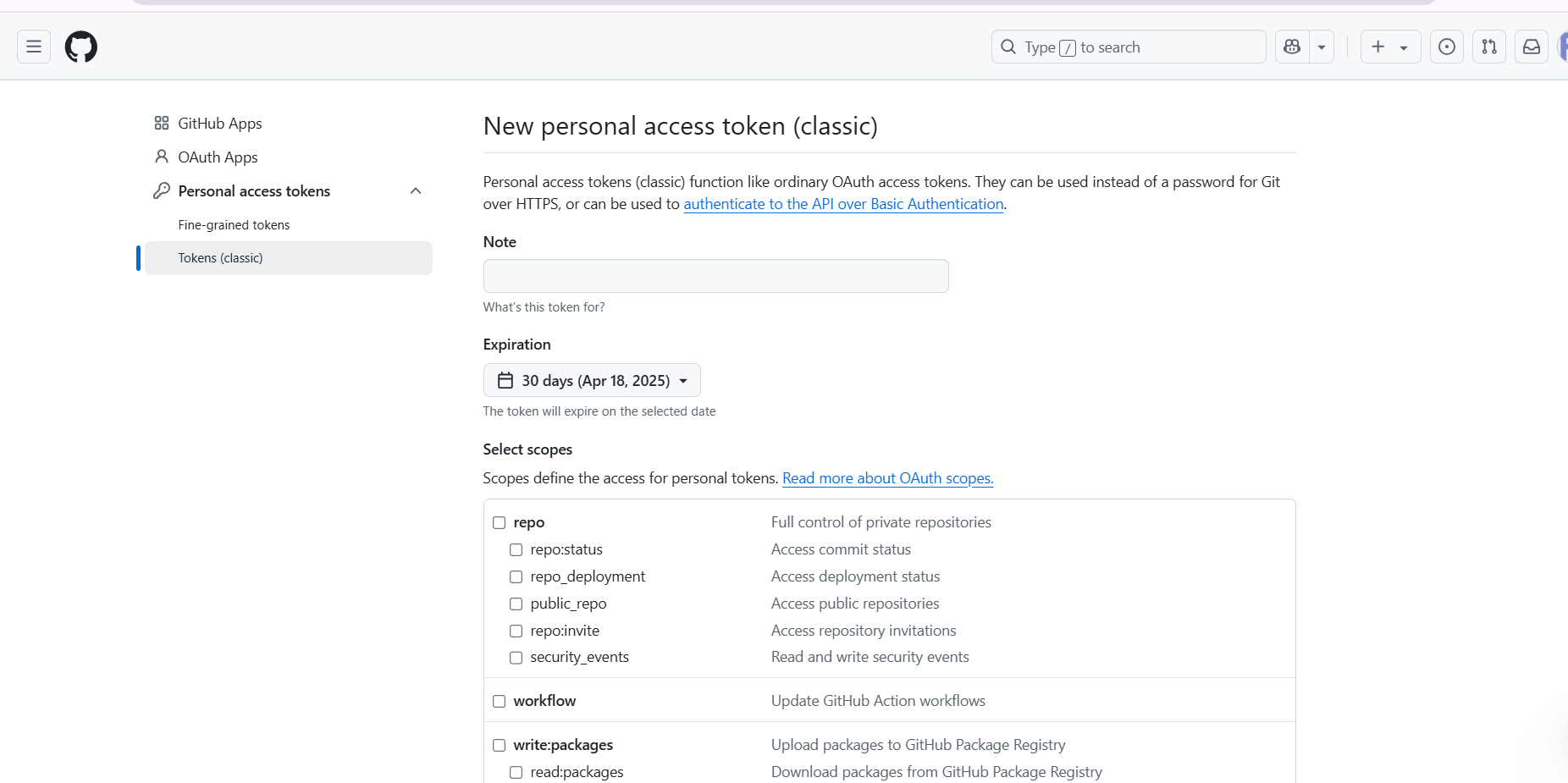


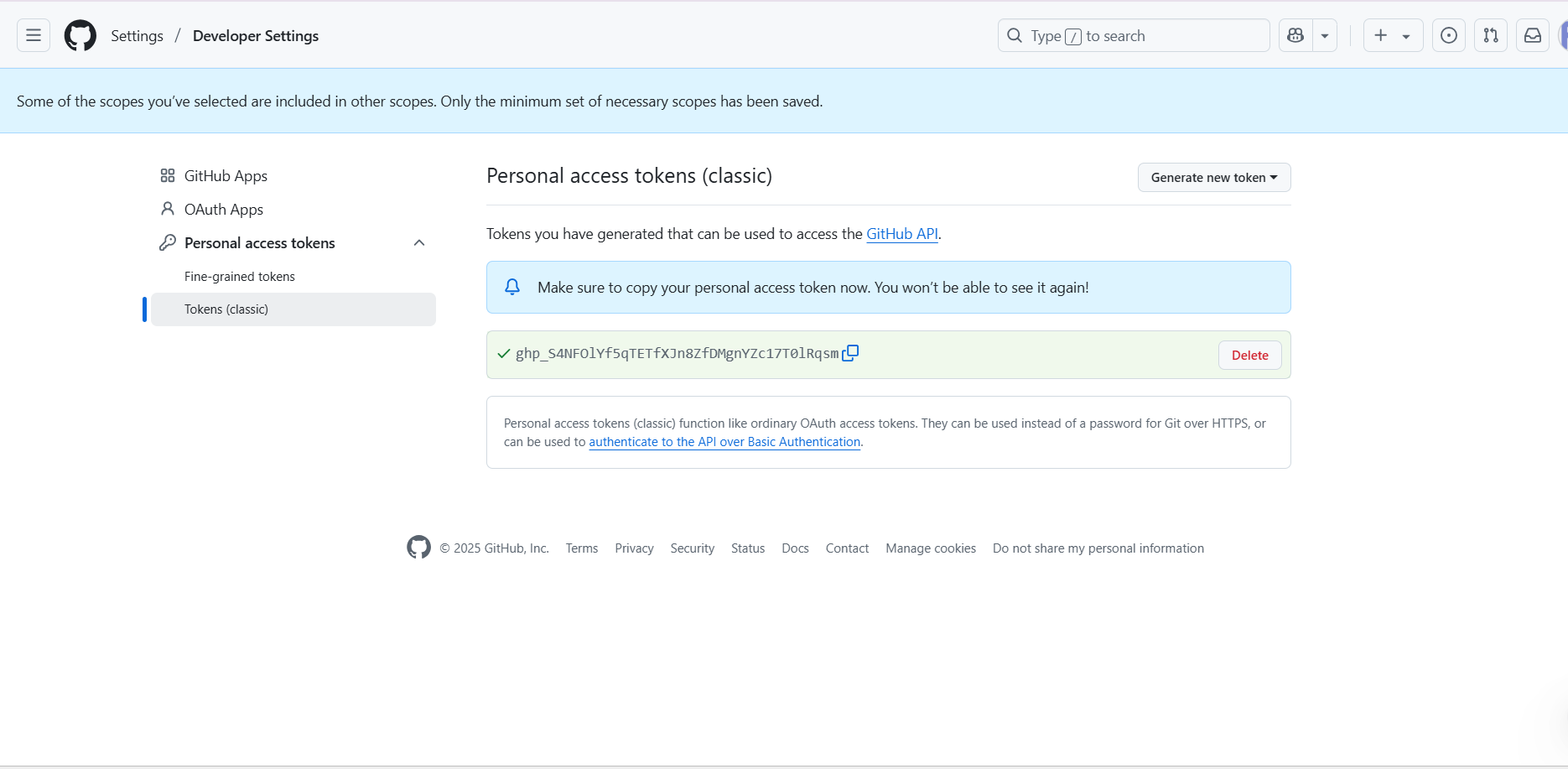
**DAY 3:**

**STEPS TO CREATE AND SETUP:**









**STARTING AND VERIFYING JENKINS SERVICE ON LINUX:**

The terminal session showing the process of managing a Jenkins server on a Linux system using `systemctl`, a command-line tool for controlling the systemd system and service manager.

**1. Enable Jenkins:**

* Command: `sudo systemctl enable jenkins`
* This command enables the Jenkins service to start automatically at boot.

**2. Start Jenkins:**

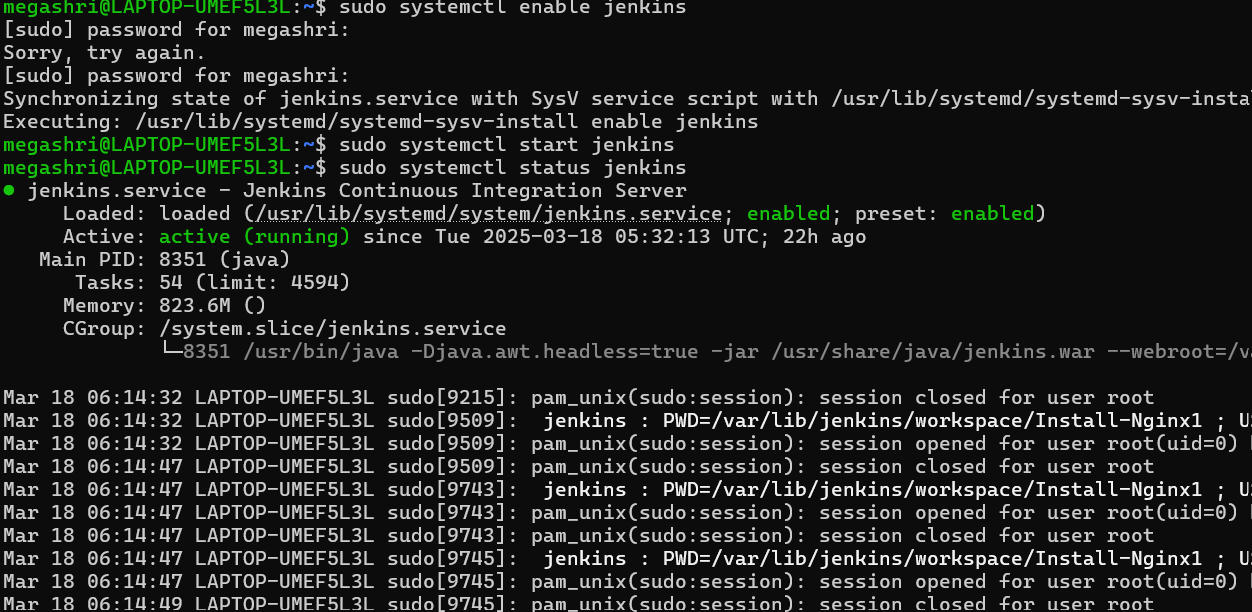
* Command: `sudo systemctl start jenkins`
* This command starts the Jenkins service immediately.

**3. Check Jenkins Status:**

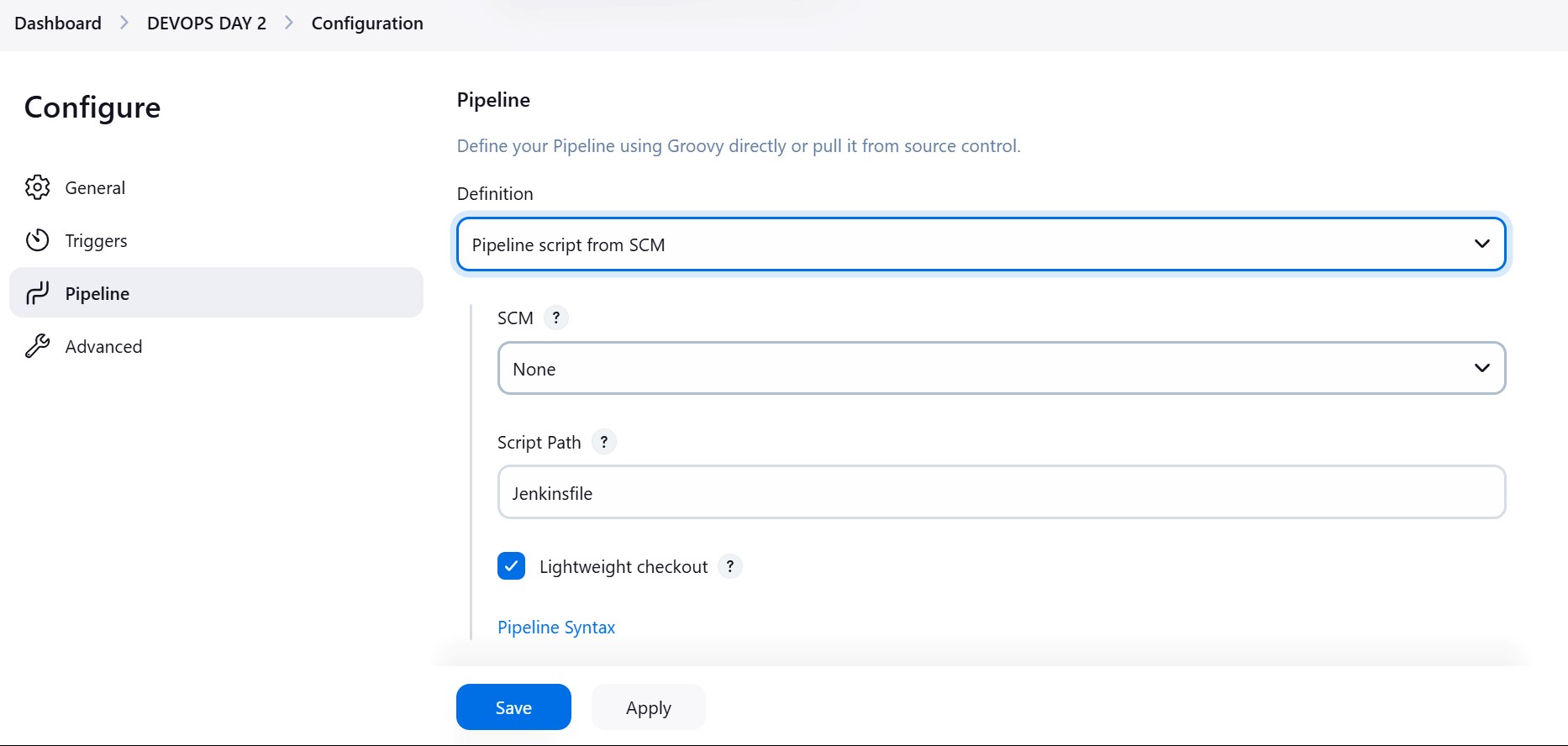
* Command: `sudo systemctl status jenkins`
* Displays the current status of the Jenkins service, indicating that it is "active (running)" and shows additional details about the service, including its main process ID (PID) and memory usage.

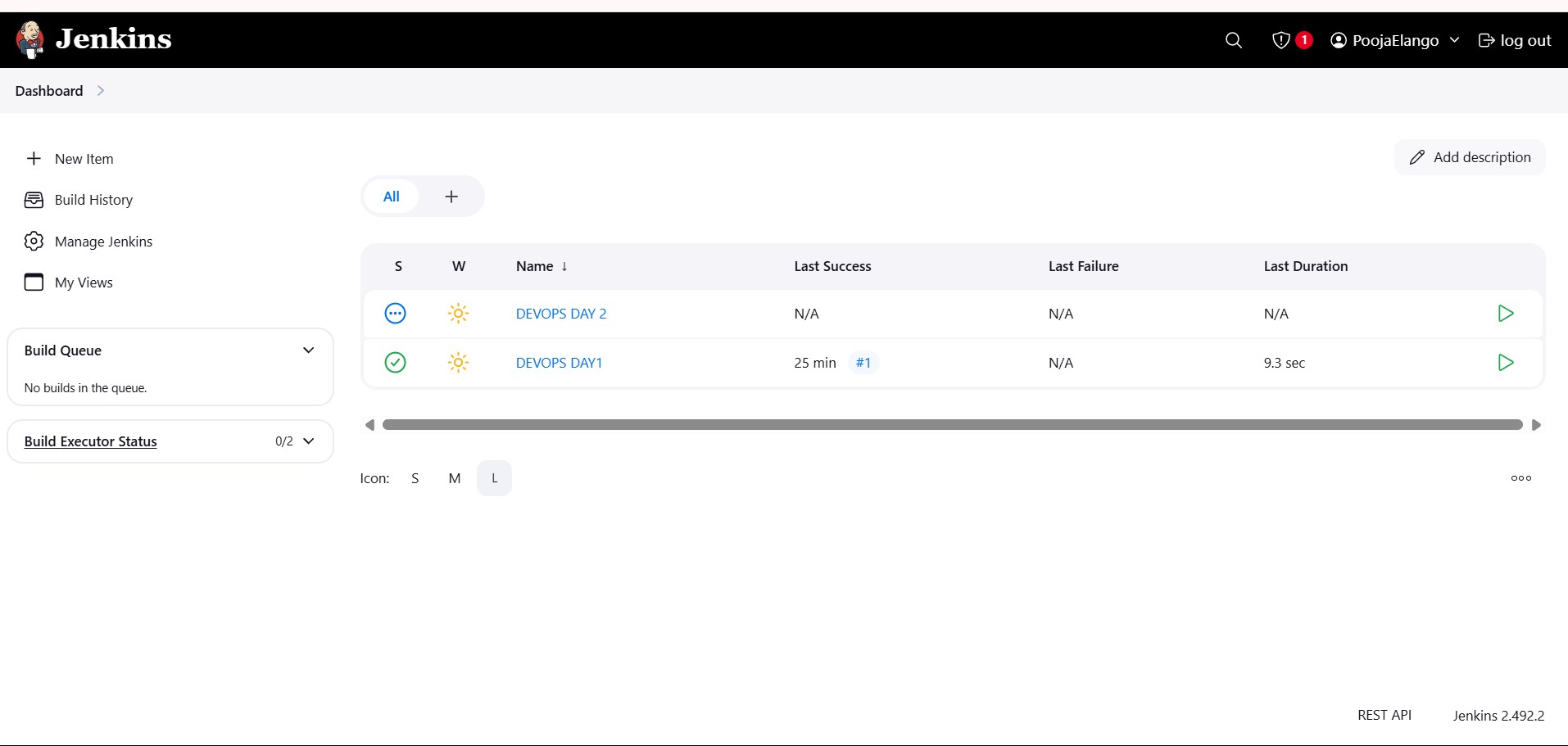
**The Summary:**

The session demonstrates the successful initialization and management of a Jenkins Continuous Integration server, indicating its readiness for use. The service was started successfully and is currently running smoothly. It also provides insight into the internal processes Jenkins follows during startup.



**NEXT STEPS TO SETUP:**



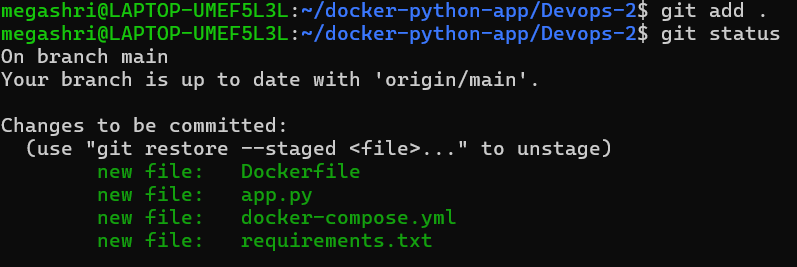


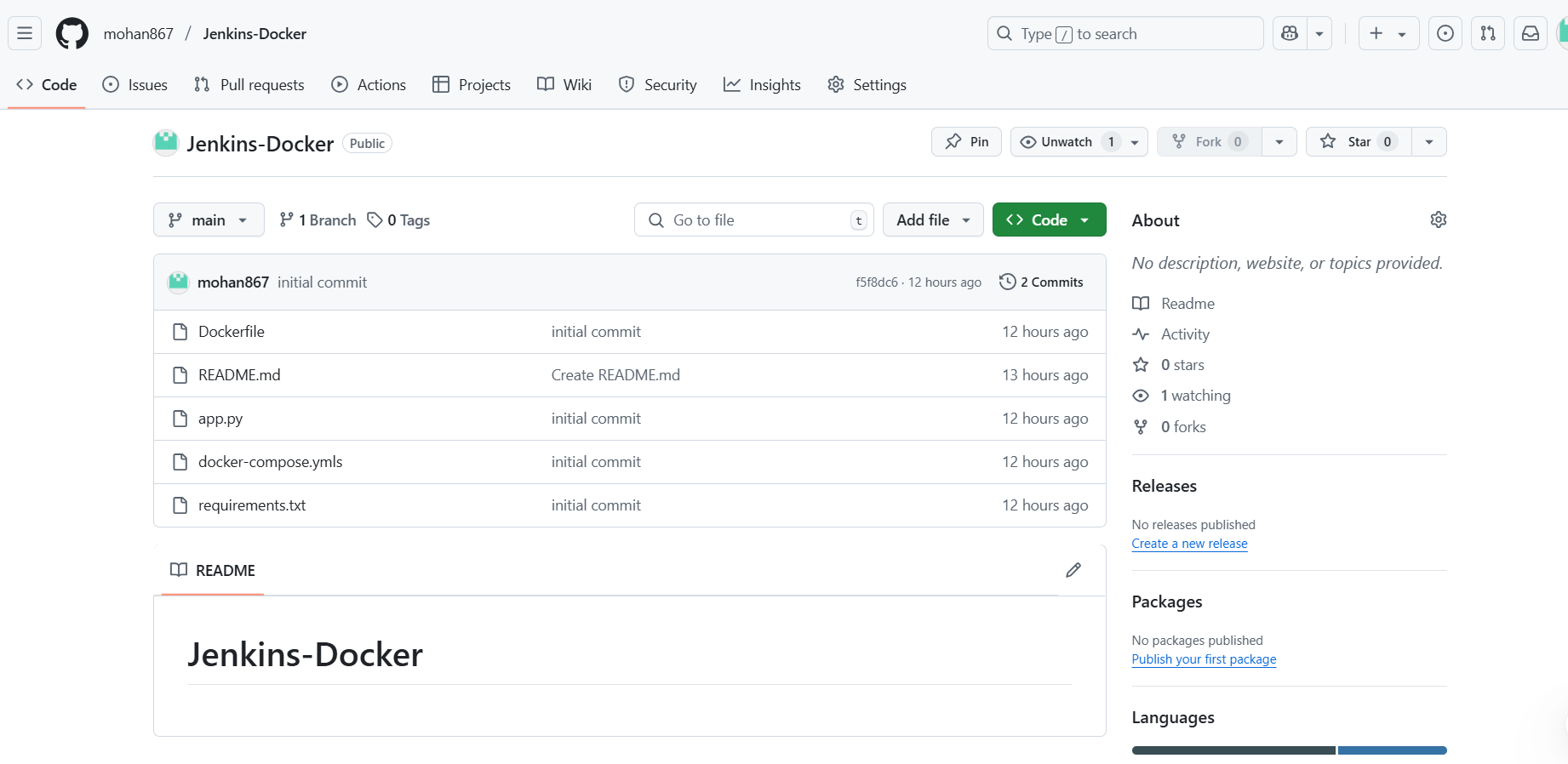
**COMMITTING AND PUSHING CHANGES TO GITHUB IN A DEVOPS PROJECT:**

The terminal session where the user is working with a Git repository named DEV-OPS-TRAINING. The steps performed include:

1. Listing the repository files, including Jenkinsfile, README.md, docker-compose.yml, dockerfile, and requirements.txt.
2. Editing the Jenkinsfile using nano.
3. Staging changes with git add ..
4. Committing the changes with the message "updated".
5. Pushing the changes to a remote GitHub repository using a personal access token for authentication.

The process successfully pushes updates to the main branch on GitHub.





**GRANTING JENKINS DOCKER ACCESS AND RESTARTING SERVICE:**

The terminal session where the user is managing Jenkins and Docker permissions. The following commands are executed:

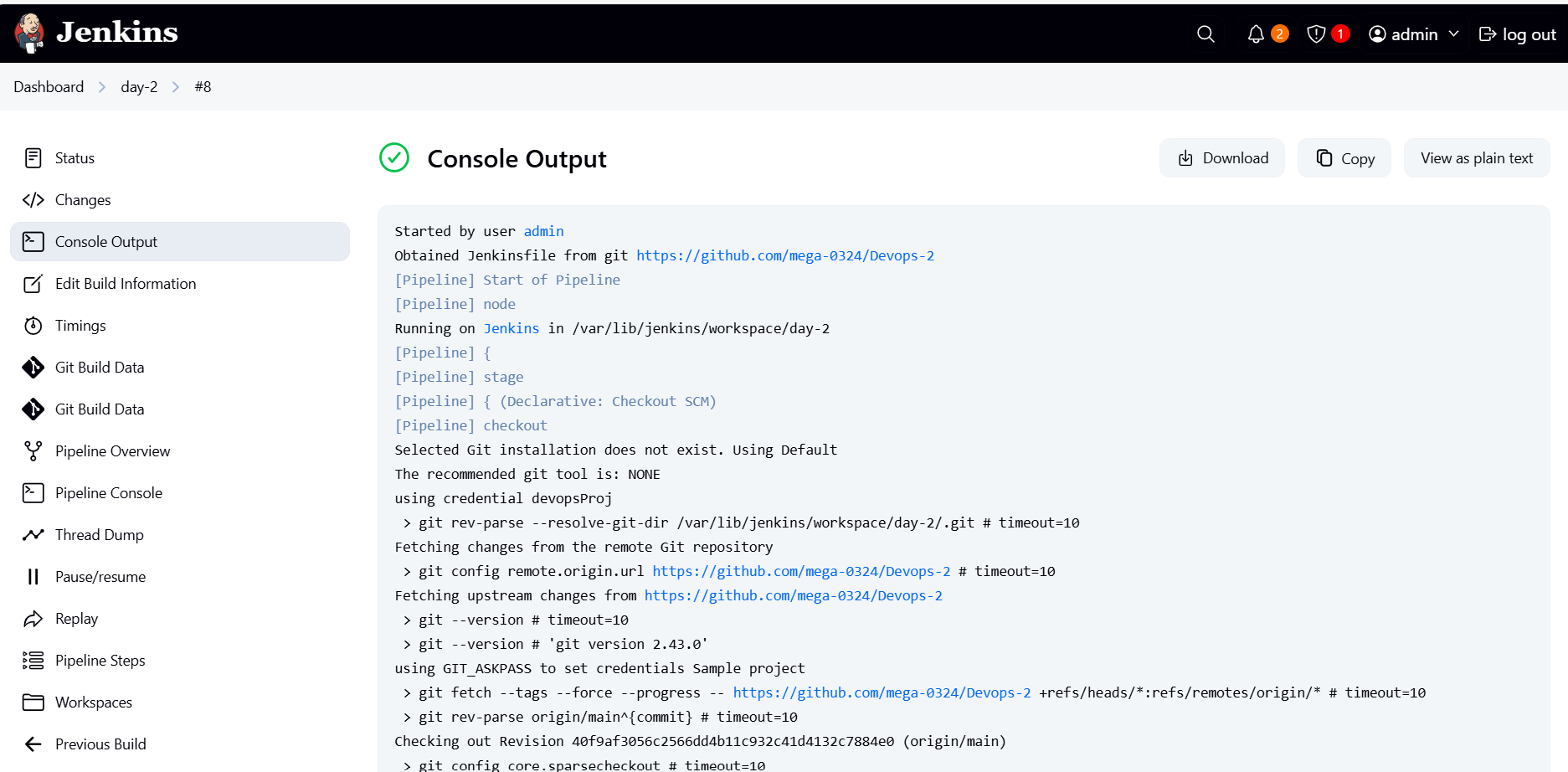
1. **sudo usermod -aG docker jenkins** – Adds the Jenkins user to the Docker group,

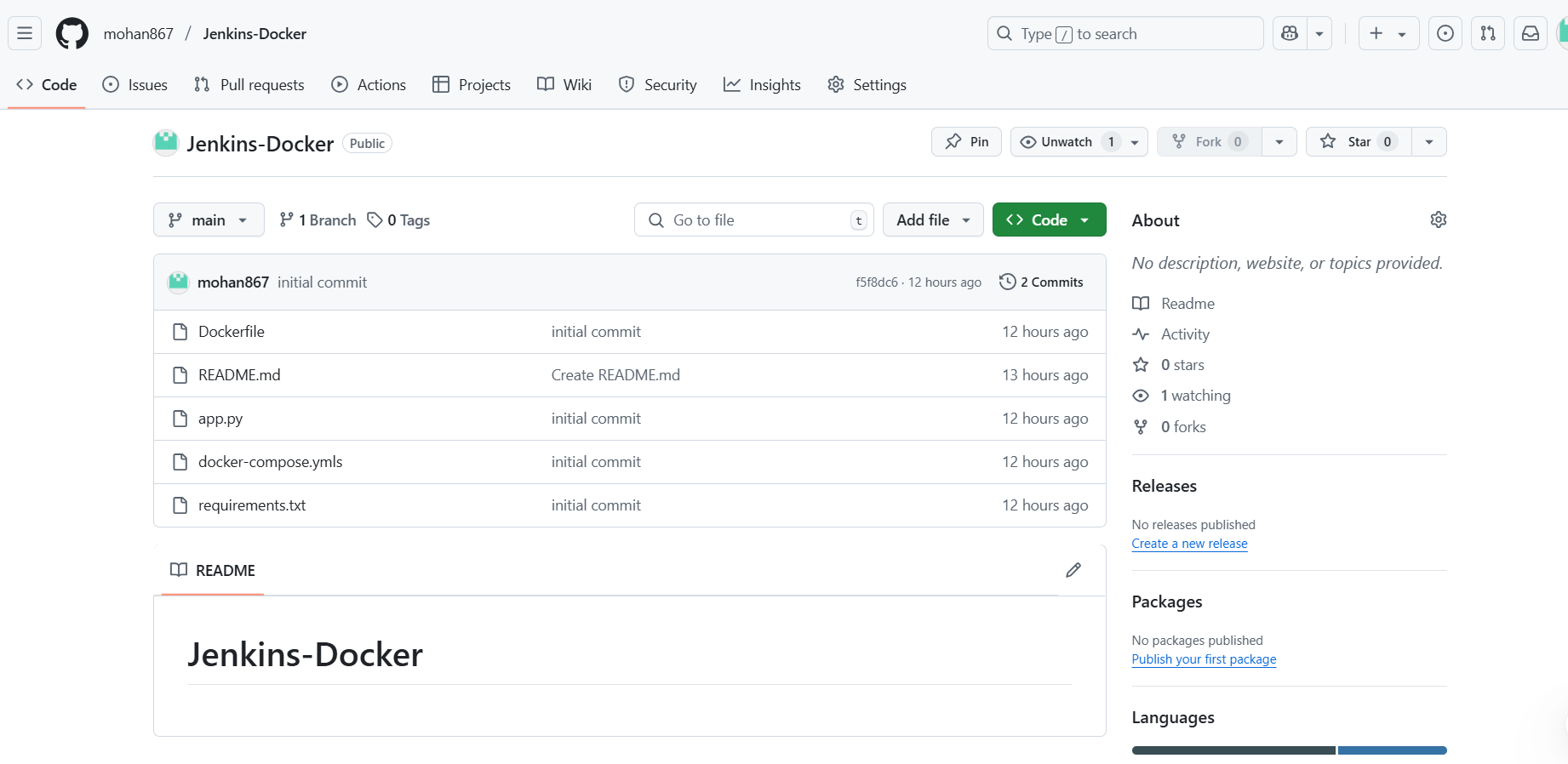
allowing it to run Docker commands without requiring sudo privileges.

1. **sudo systemctl restart jenkins** – Restarts the Jenkins service to apply the changes made to its user permissions.

These steps are typically done to enable Jenkins to interact with Docker seamlessly in a CI/CD pipeline.



**JENKINS DASHBOARD OVERVIEW:** 



**DOCKER HUB REPOSITORY OVERVIEW FOR poojaelango/docker-app:**

