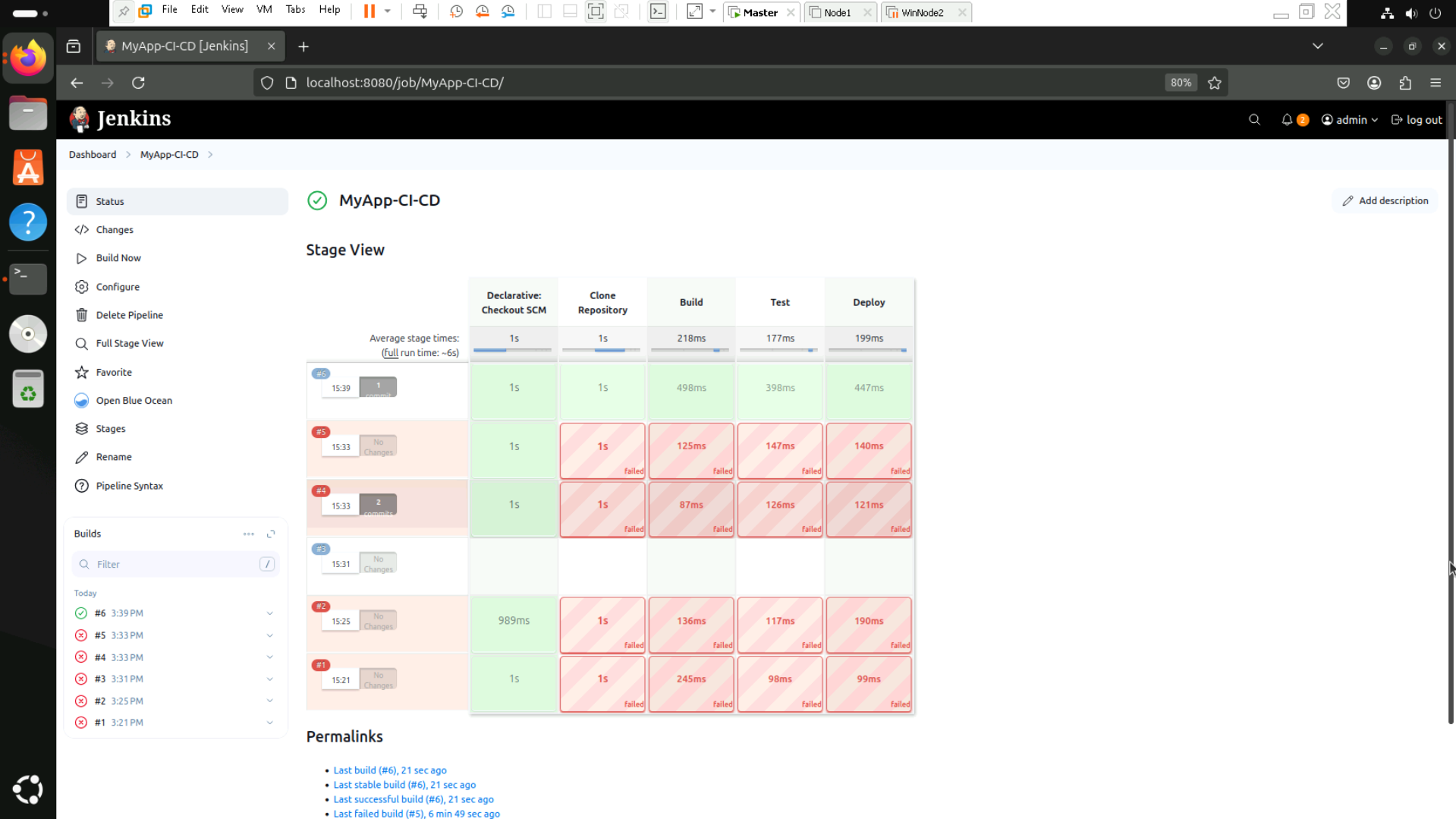
`Name: Barathkumar J K

Jenkins

**Project 1**

1. **Build and Deploy a Simple App using Jenkins**



**Project 2**

1. **Build and Deploy a Simple App using Jenkins & docker**

**Project Structure**

my-flask-app  
│── app.py  
│── test.py  
│── requirements.txt  
│── Dockerfile  
│── Jenkinsfile

**1️) Setup a Simple Flask App**

**app.py**

from flask import Flask

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

@app.route('/')

def hello():

return 'Hello, World!'

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

app.run(debug=True, host='0.0.0.0')

**2️) Create a Test File**

**test.py**

import unittest

from app import app

class FlaskAppTestCase(unittest.TestCase):

@classmethod

def setUpClass(cls):

cls.client = app.test\_client()

cls.client.testing = True

def test\_homepage(self):

response = self.client.get('/')

self.assertEqual(response.status\_code, 200)

self.assertIn(b'Hello, World!', response.data)

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

unittest.main()

**3️) Create a requirements.txt File**

**requirements.txt**

Flask==2.2.2

Werkzeug==2.2.2

**4️) Create a Dockerfile**

**Dockerfile**

FROM python:3.12-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 5000

ENV FLASK\_APP=app.py

ENV FLASK\_RUN\_HOST=0.0.0.0

CMD ["flask", "run"]

**5️) Build & Push Docker Image**

1. Build the Docker image:

docker build -t your-dockerhub-username/my-flask-app:latest .

1. Push the image to DockerHub:

docker login

docker push your-dockerhub-username/my-flask-app:latest

**6️) Connect Jenkins Pipeline to GitHub**

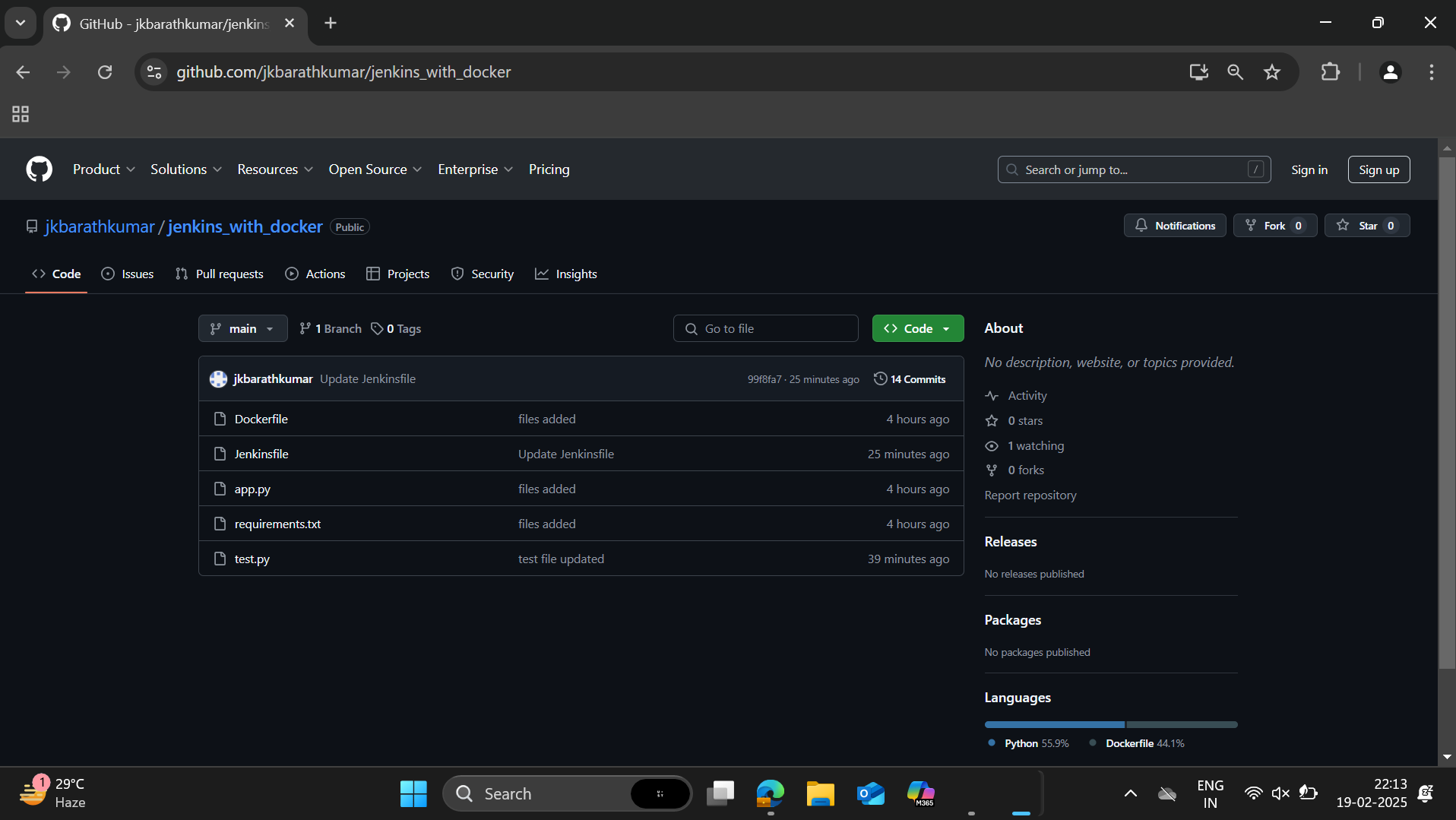
1. Open Jenkins Dashboard.
2. Click on **New Item** → Select **Pipeline** → Name it → Click **OK**.
3. In the **Pipeline** section:
   * Select **Pipeline script from SCM**.
   * Choose **Git** as SCM.
   * Enter your **GitHub repository URL**.
   * Set **Branch: main**.
   * Define the **Jenkinsfile path**.
4. Click **Save**.

**7️) Click "Build Now"**

1. Navigate to **Jenkins Dashboard** → Your Pipeline.
2. Click **Build Now**.

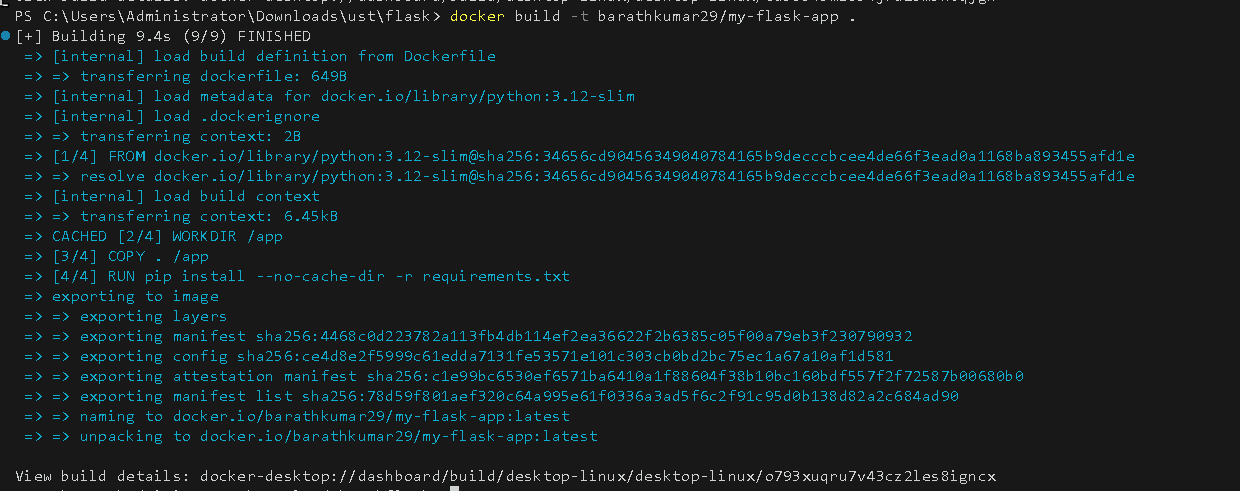
**Results:**

**GITHUB**

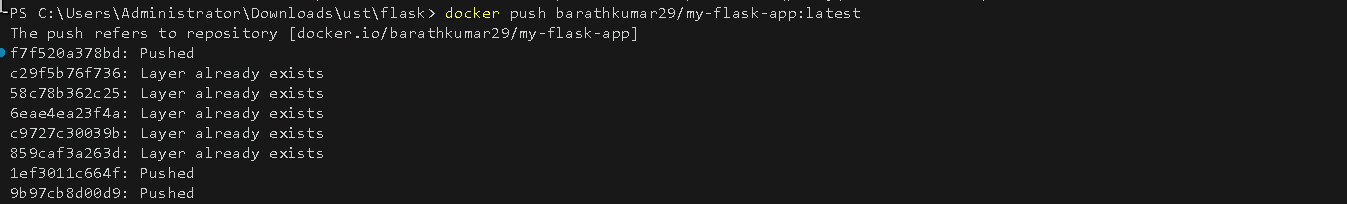
****

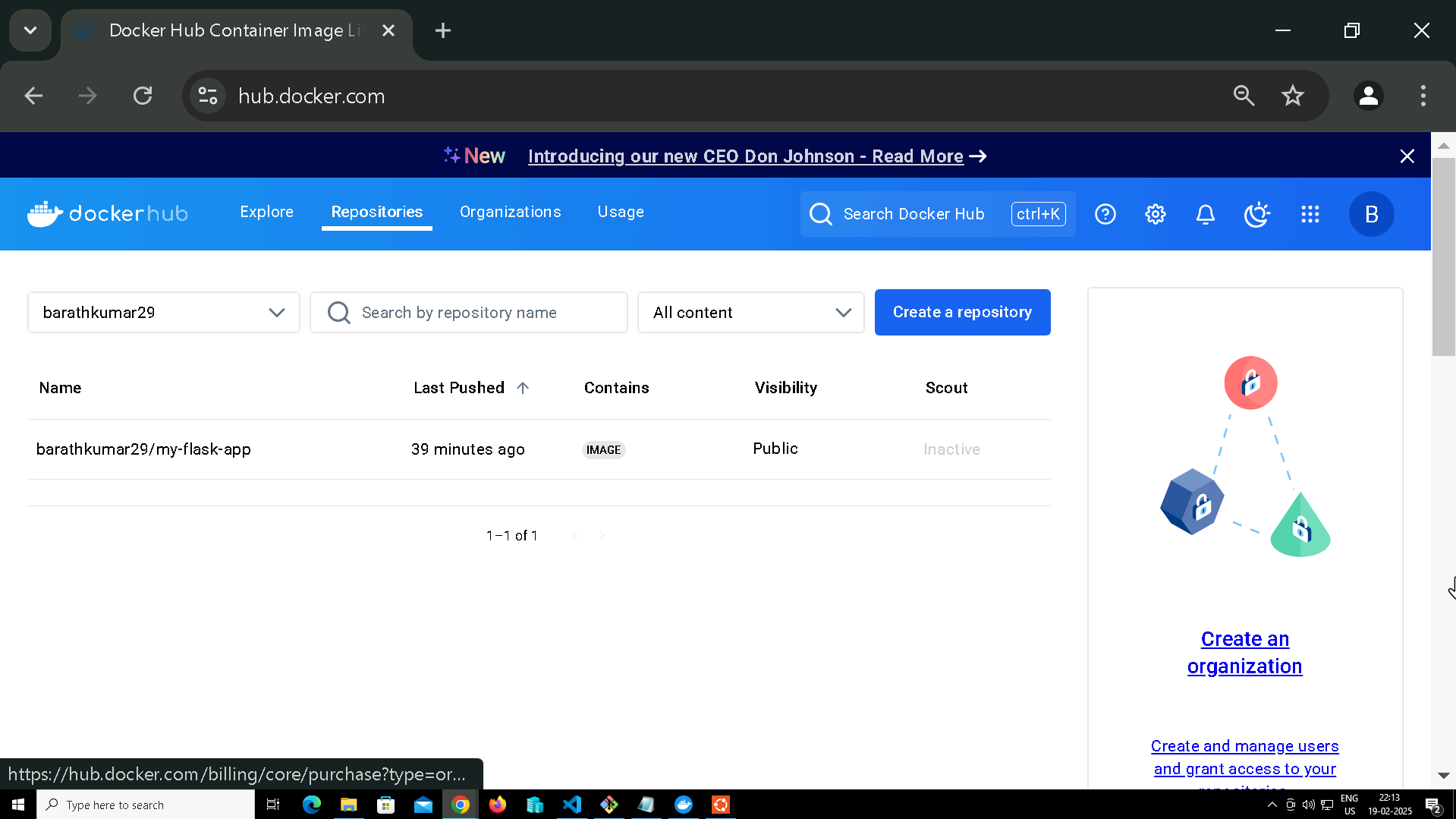
**Docker**

**Creating the docker image using dockerfile**

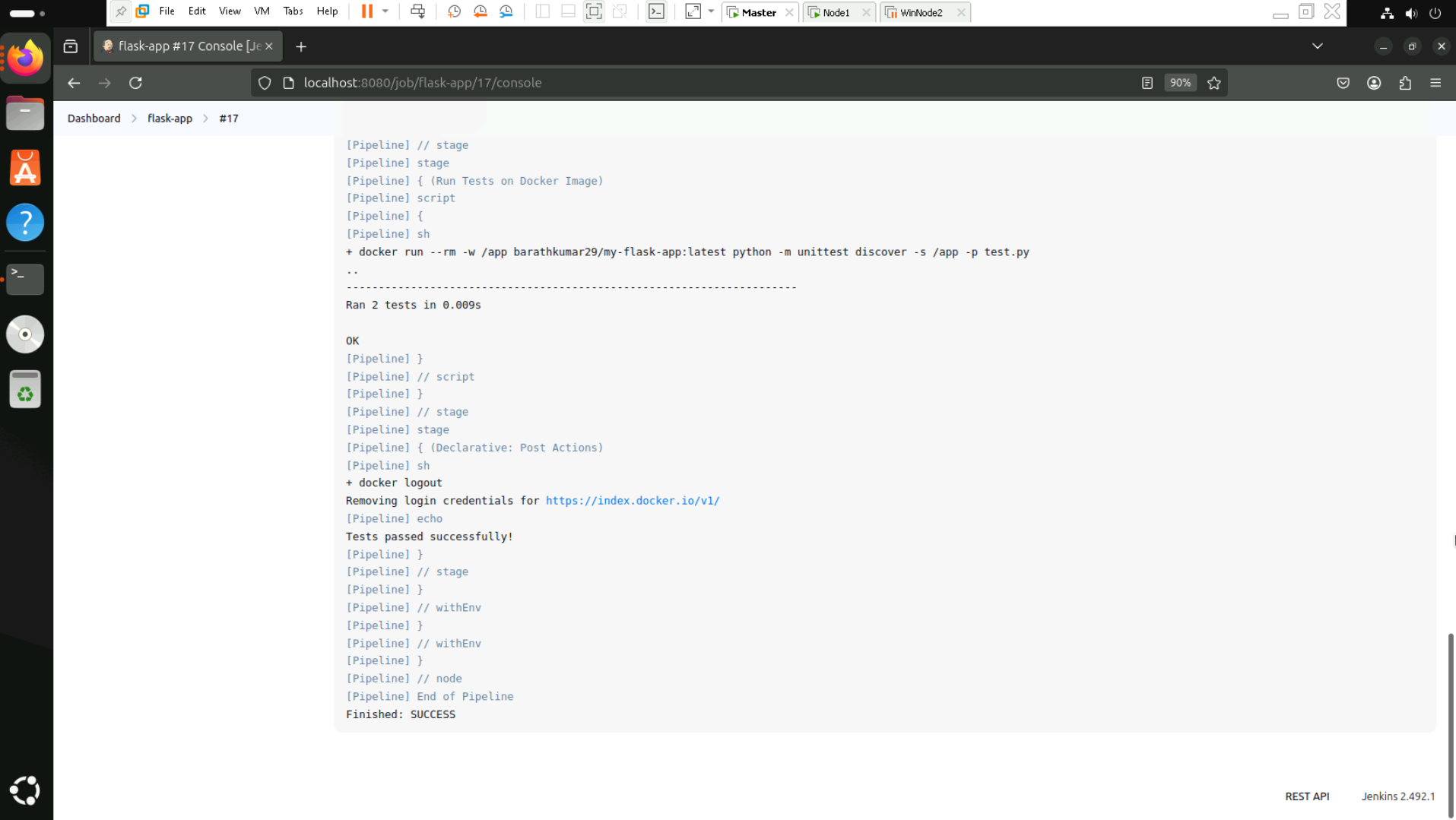
****

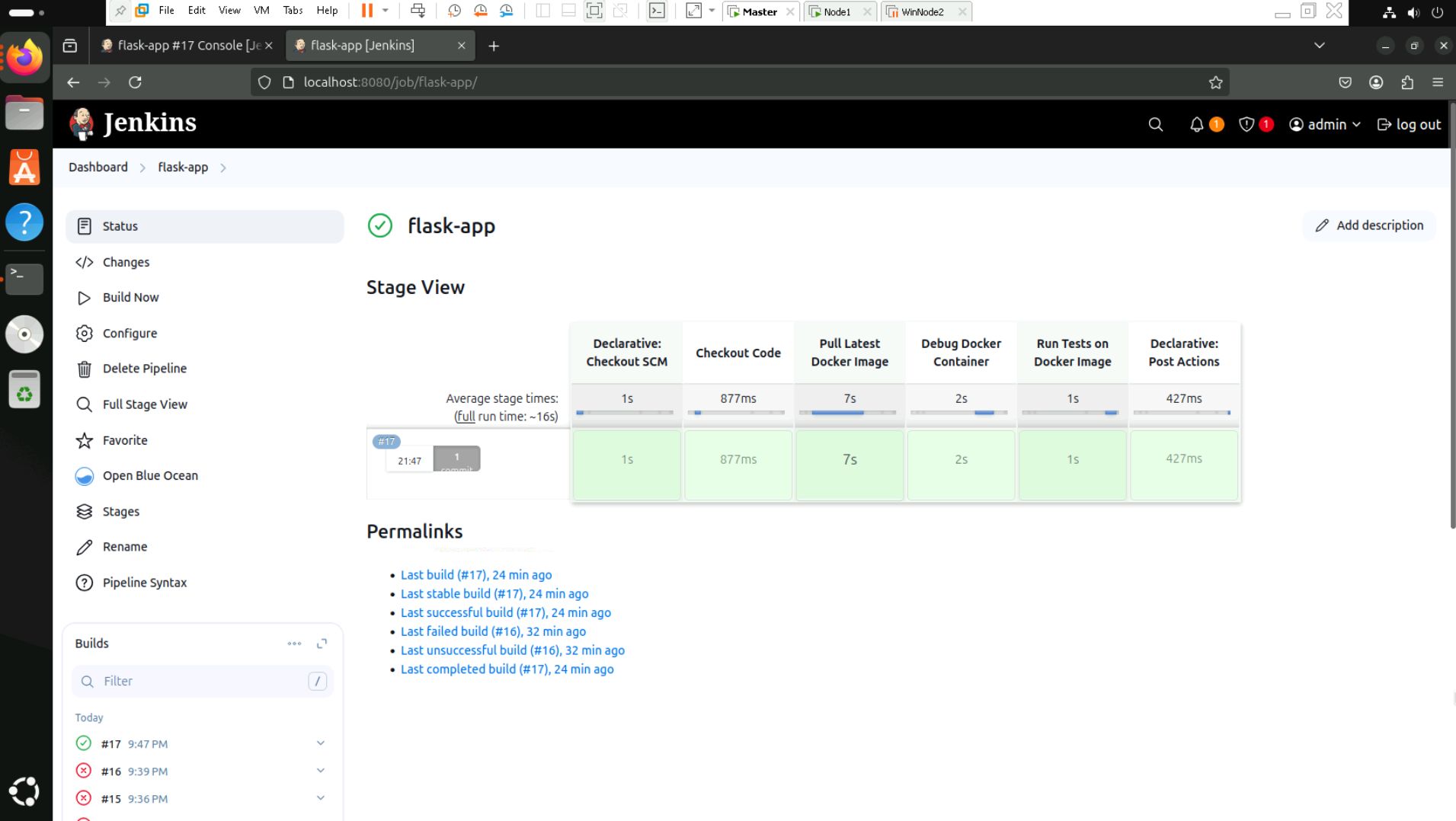
**Push the docker image to dockerhub:**

****

****

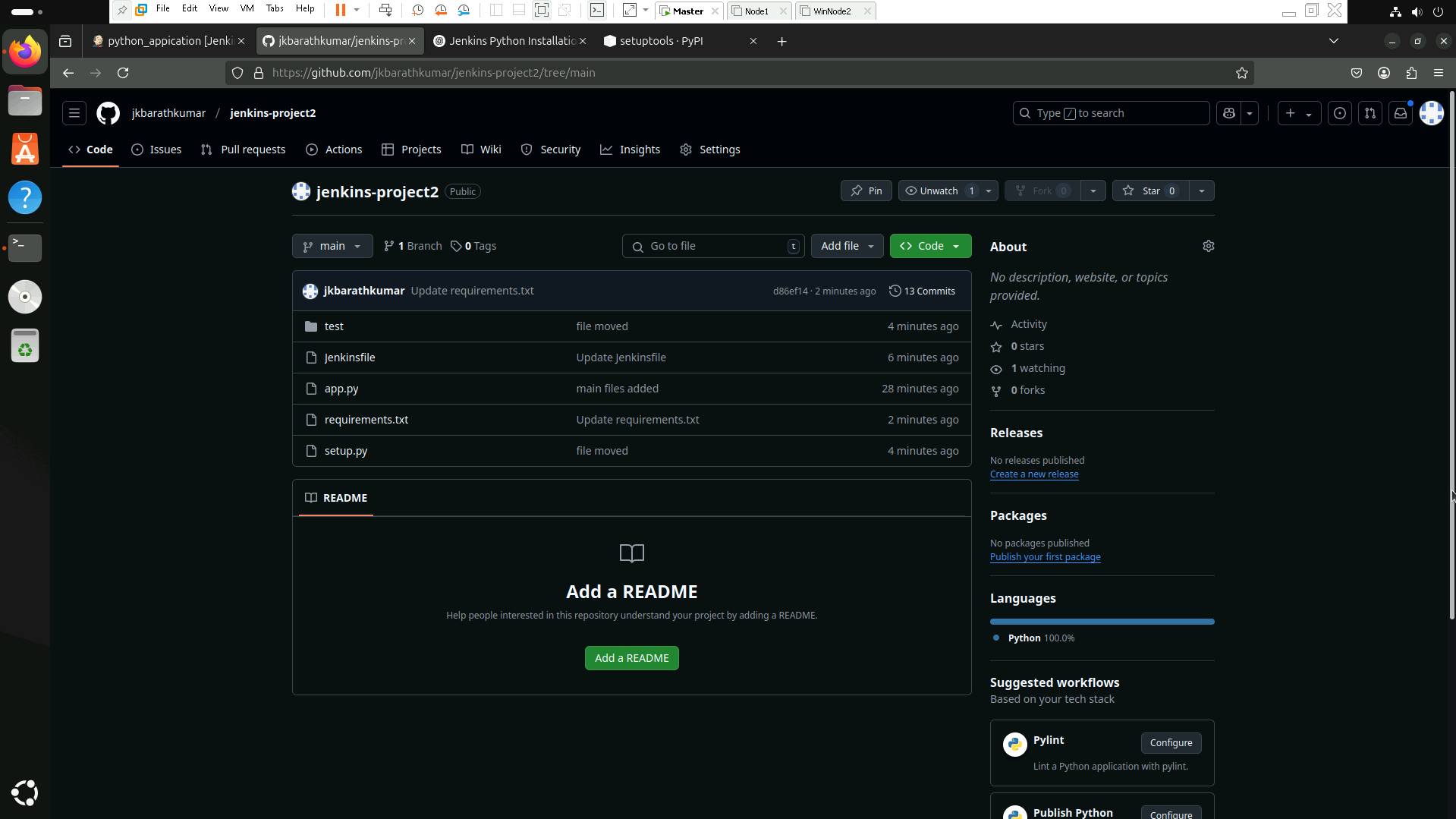
**Jenkins**

****

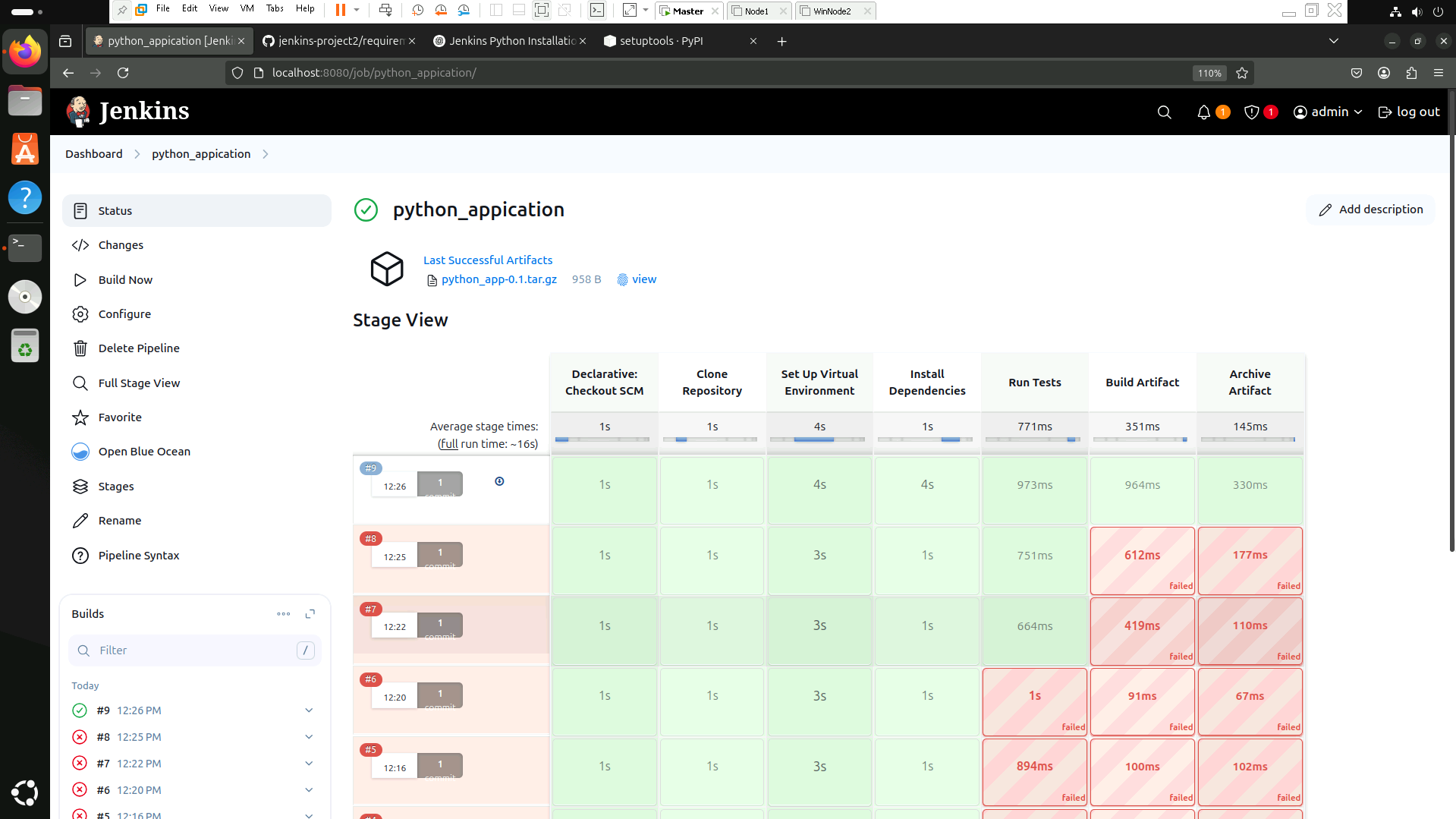
****

**Project 3:**

**Automating a Python Application Build using Jenkins**

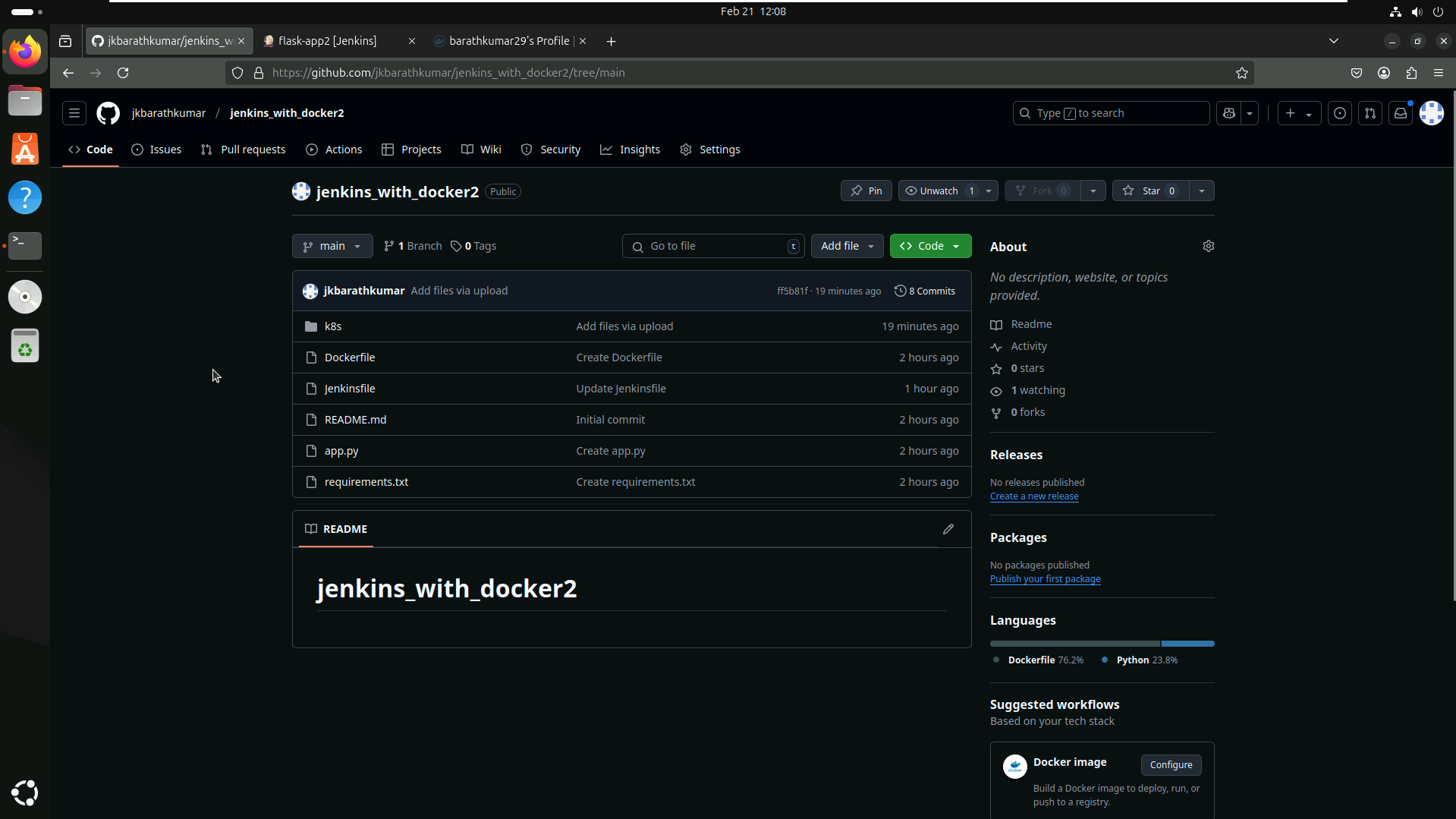




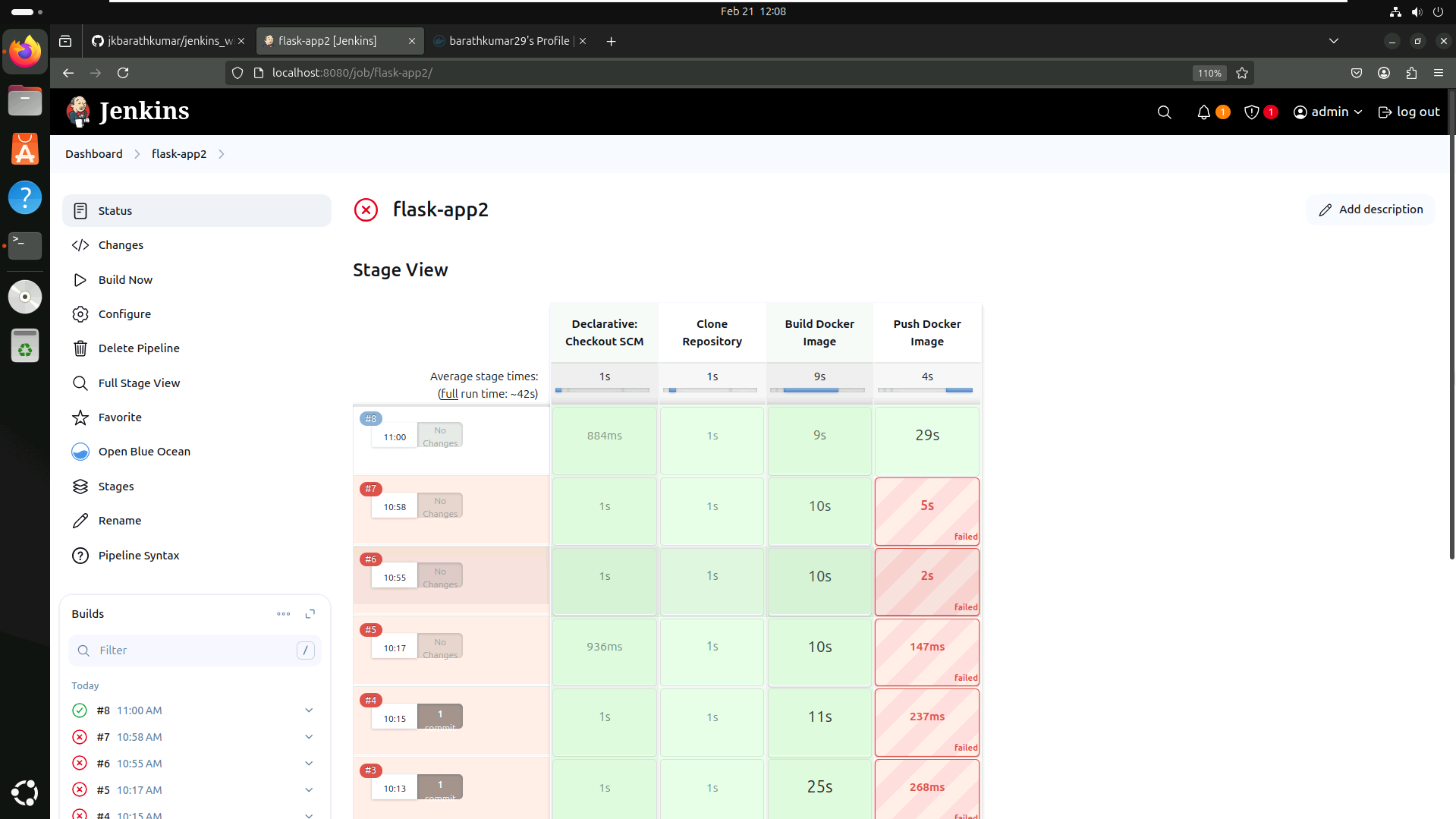


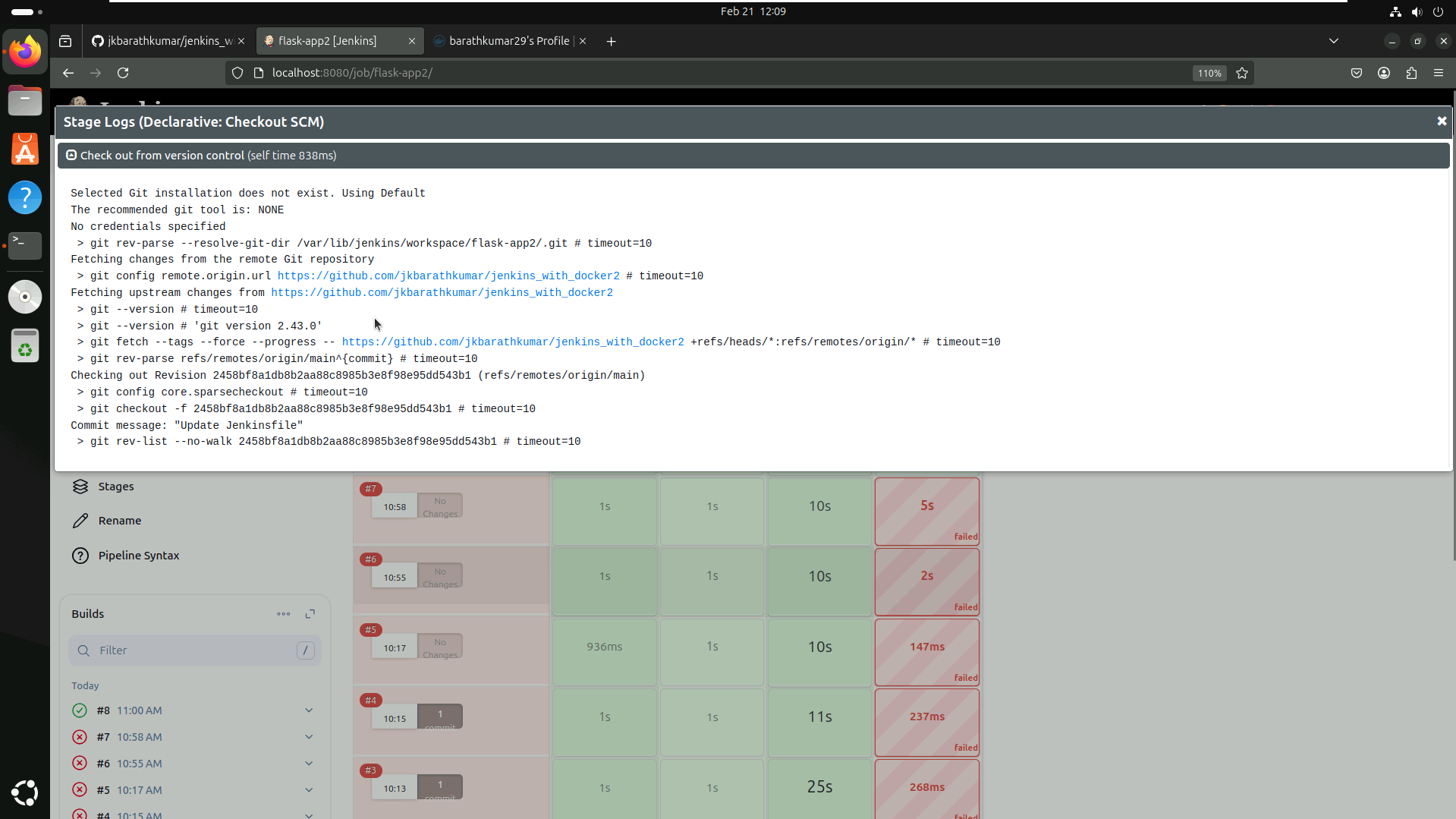
**Project 4:**

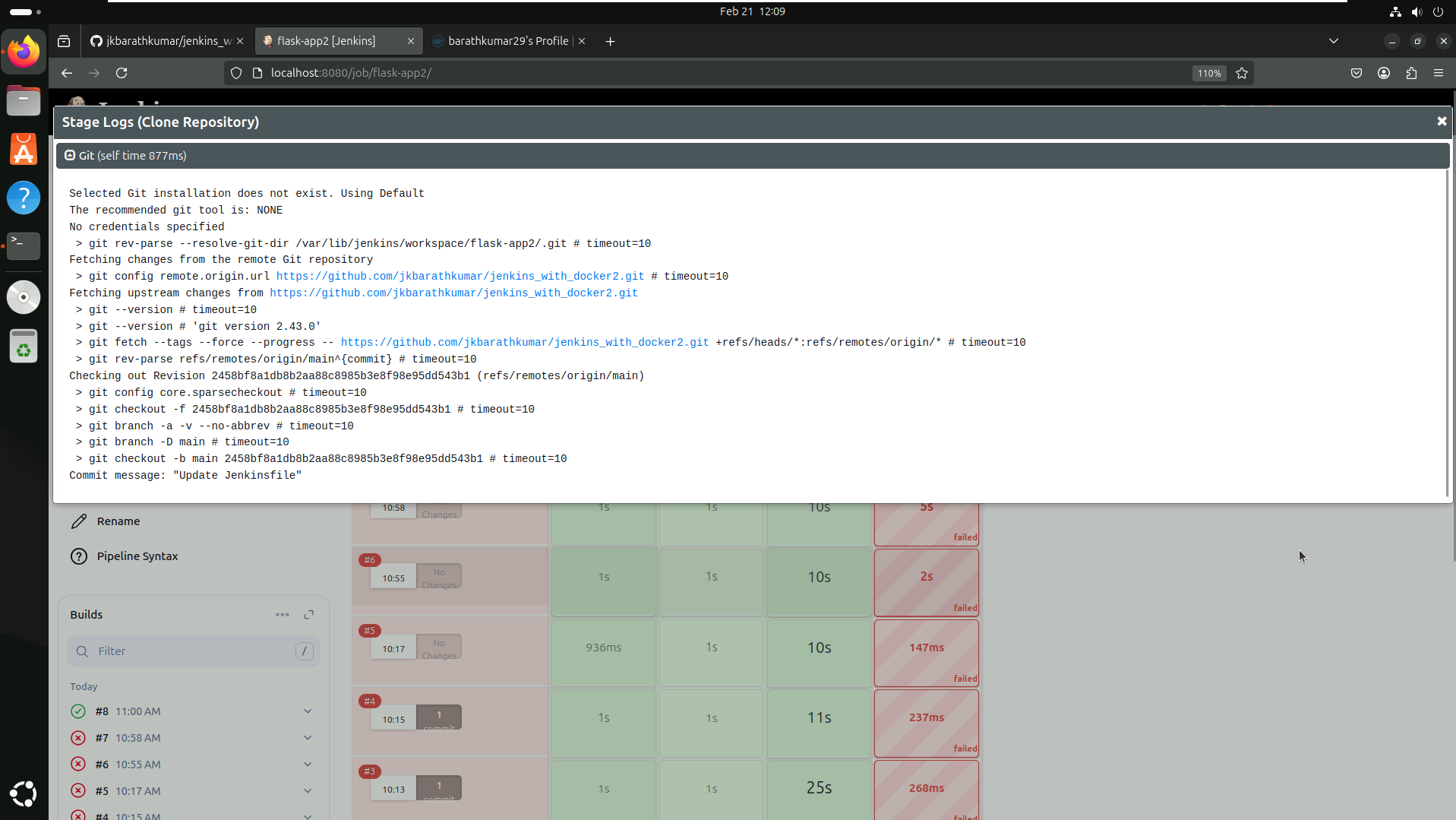
**Docker-based Jenkins CI/CD Pipeline**

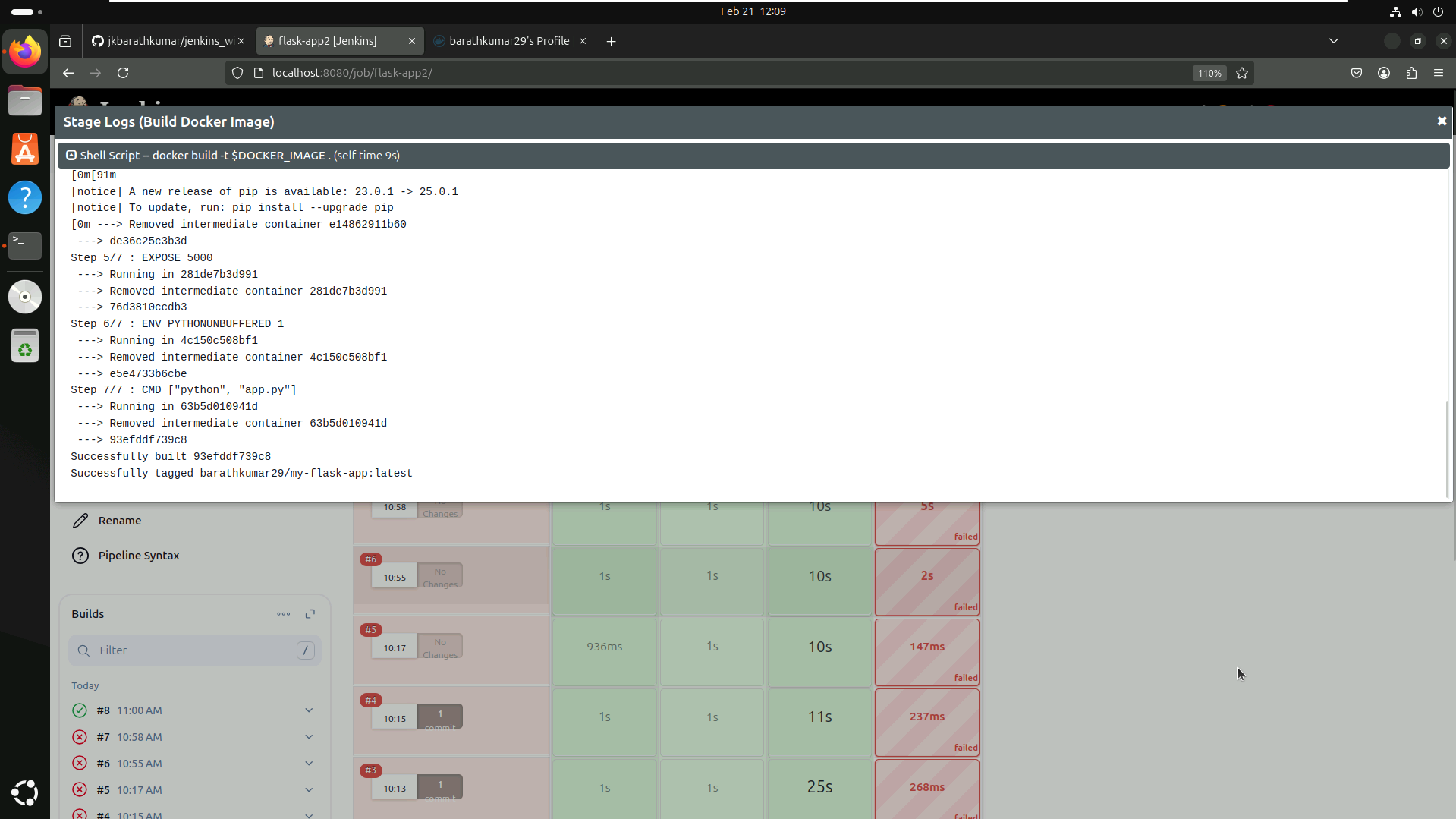
**Github**

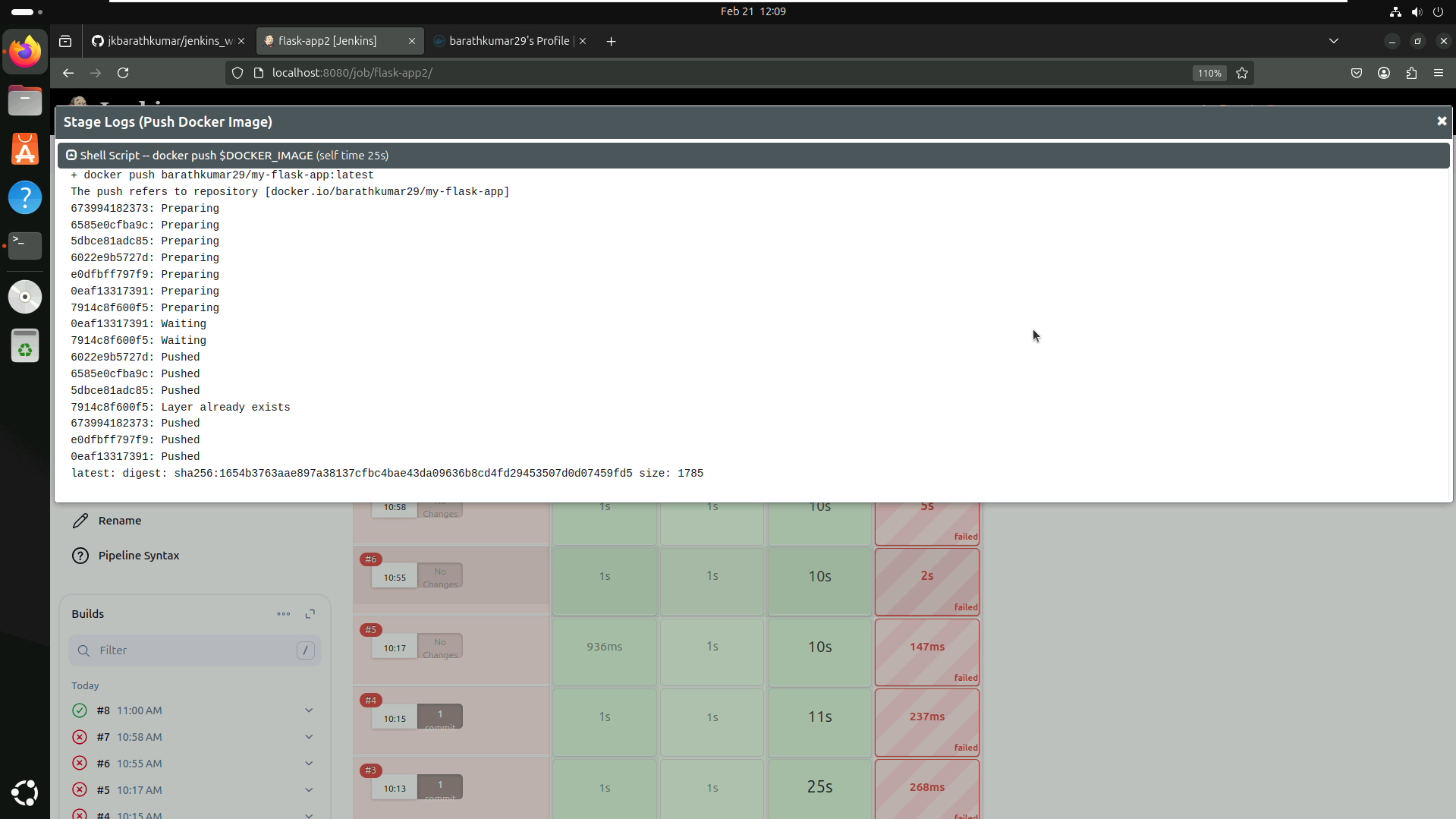
**Jenkins**

****

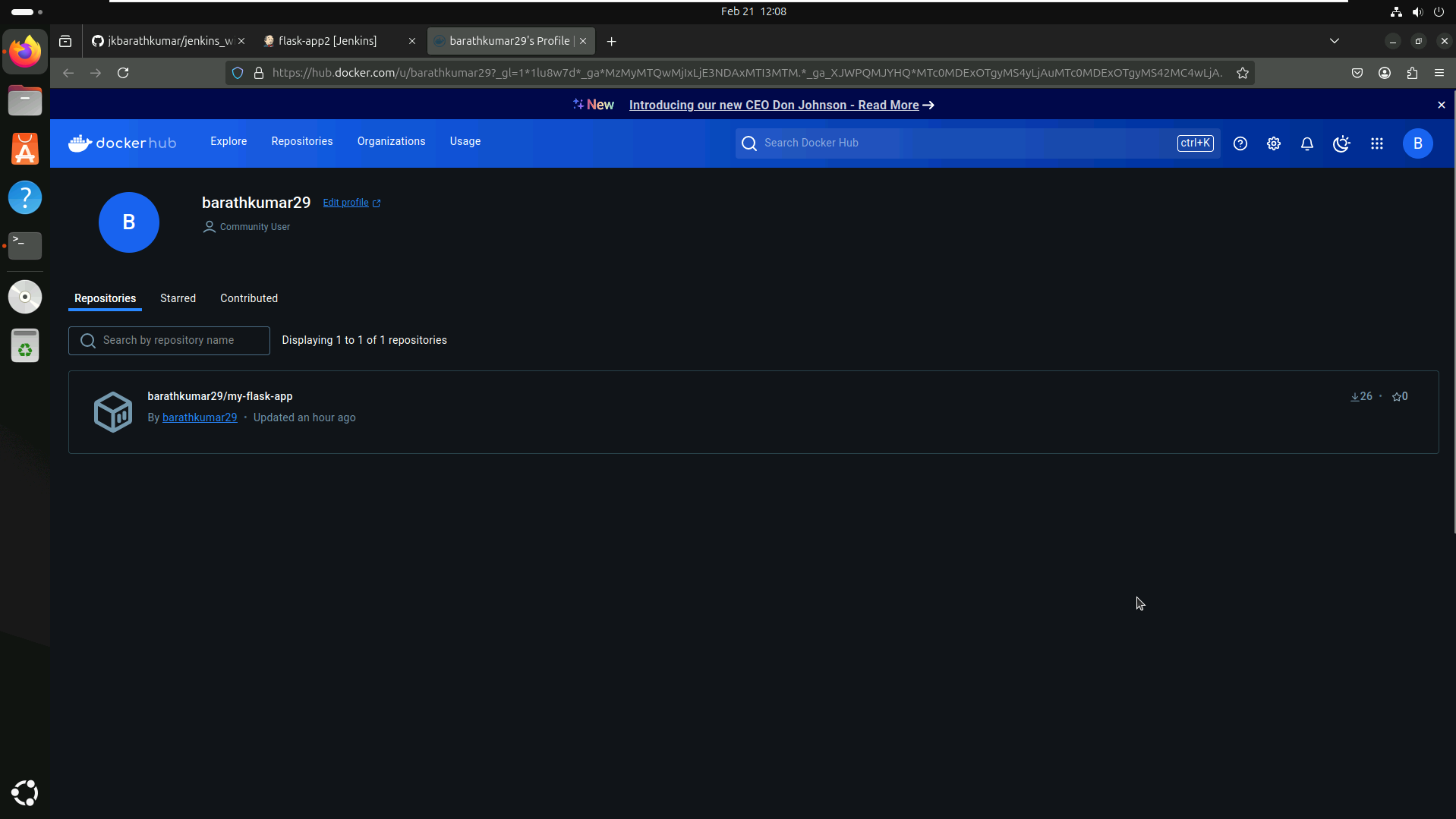
****

****

****

****

**Dockerhub**

****

**Project 5**

**Auto Trigger and Deploy with GitHub Pages:**

**Objective:**

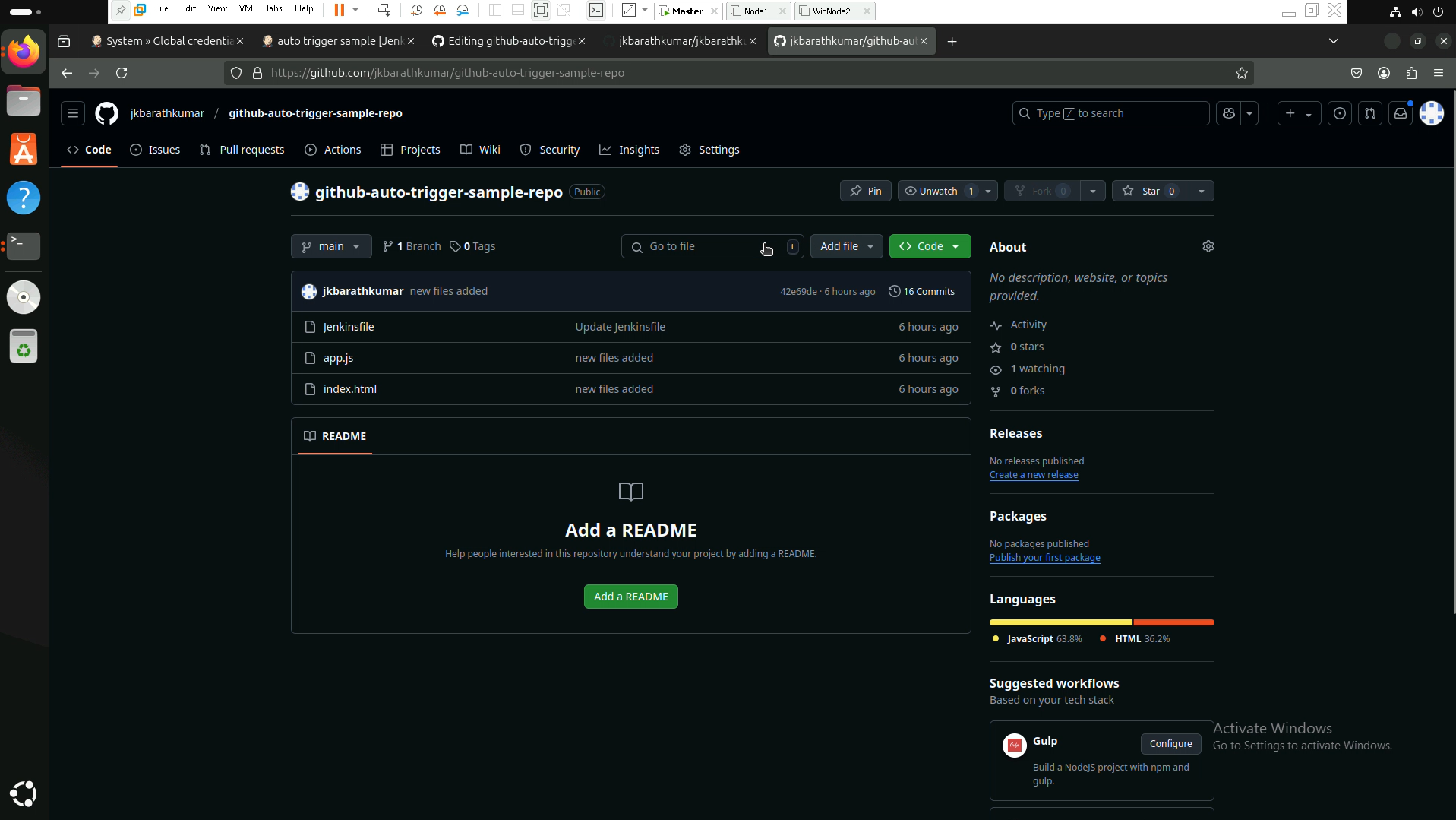
The goal of this project is to automatically trigger a deployment pipeline whenever a new code push is made to the GitHub repository. The pipeline will run and deploy the latest code changes to GitHub Pages, eliminating the need for manual deployment.

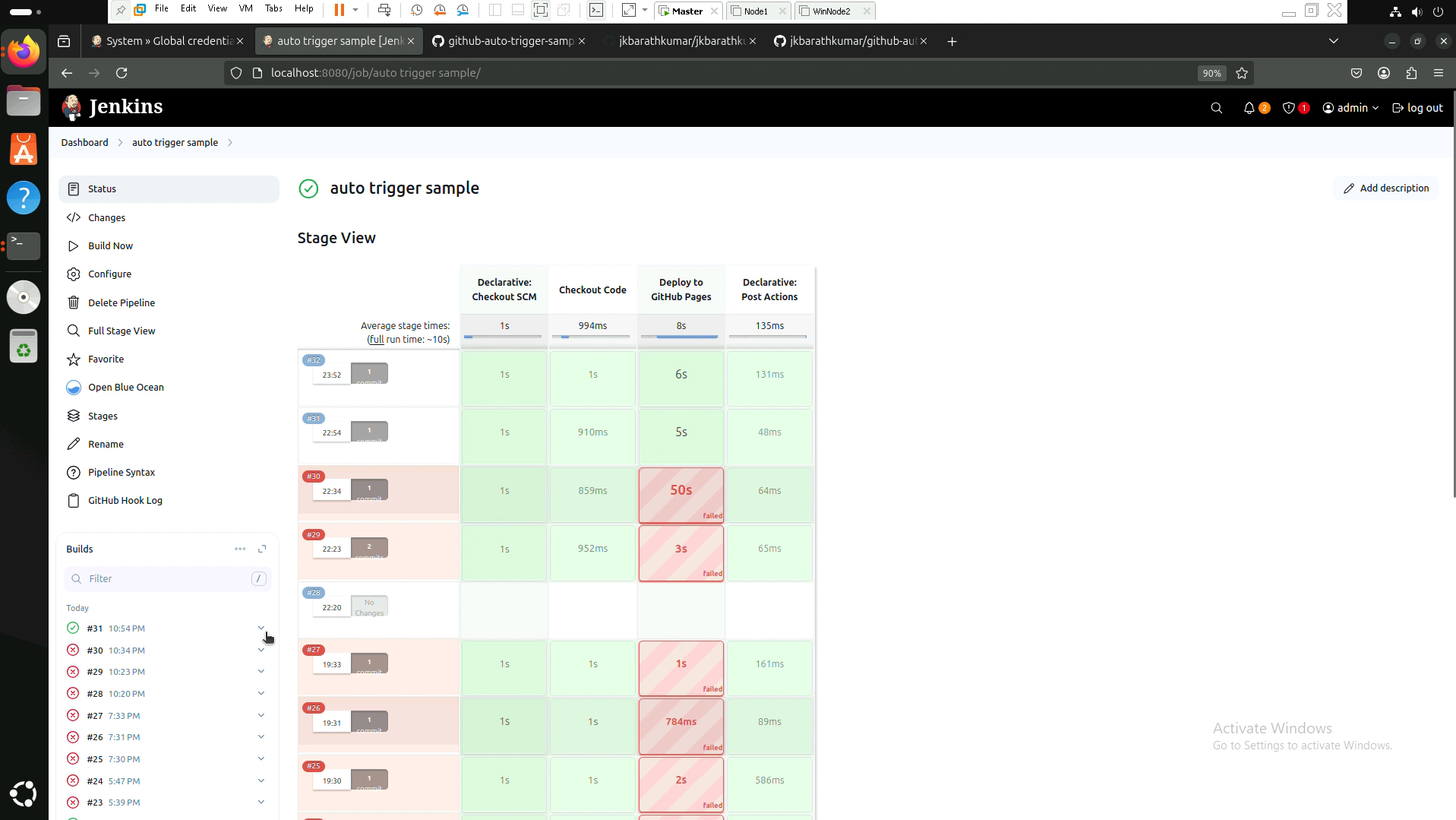
**Key Features:**

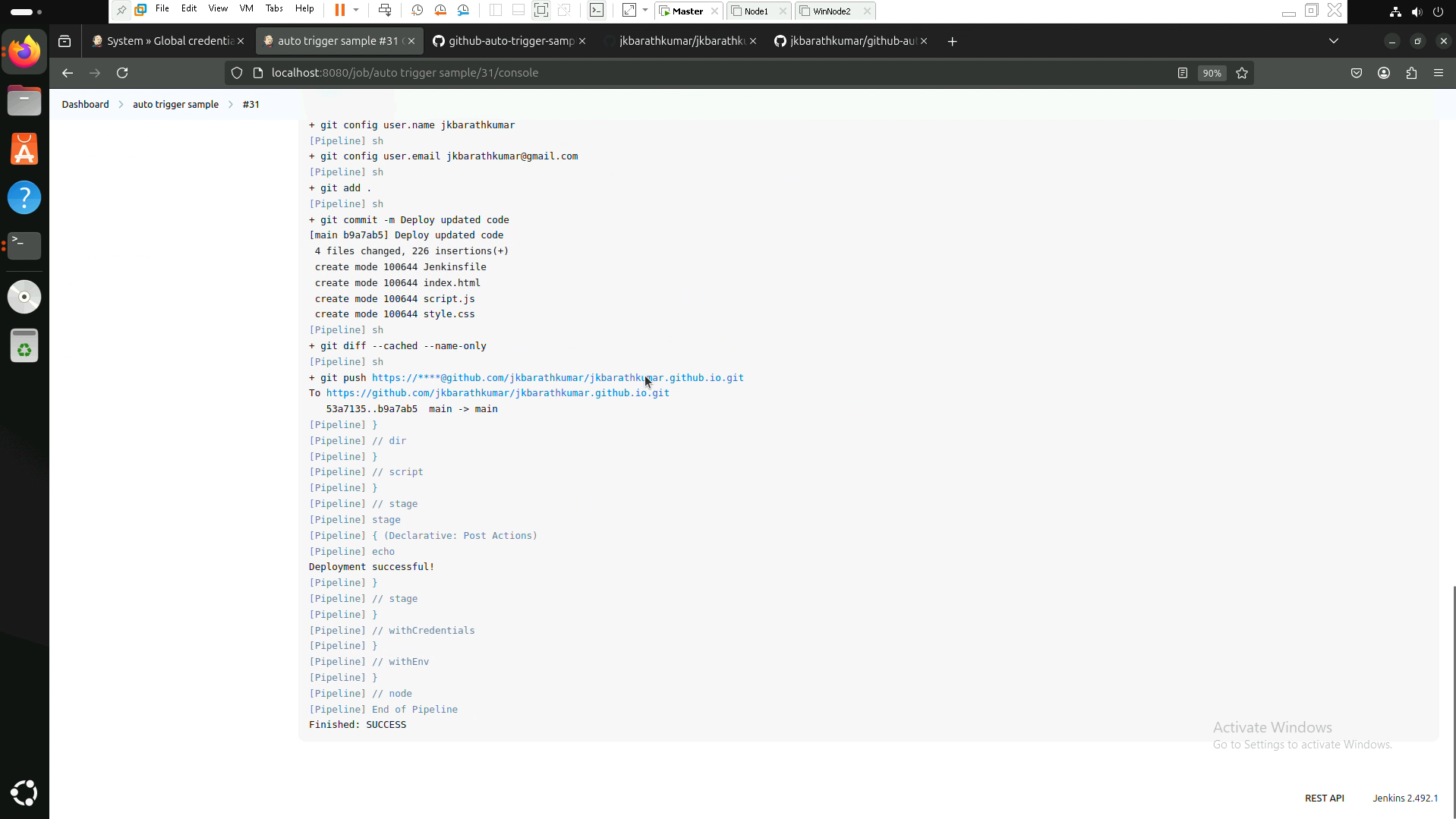
* Automatic Trigger: Every time a new commit is pushed to the repository, the pipeline will automatically trigger and begin the deployment process.
* Deployment to GitHub Pages: Once the code is built and tested, it will be deployed to GitHub Pages, which will serve the application without requiring any cloud infrastructure.

**Process Flow:**

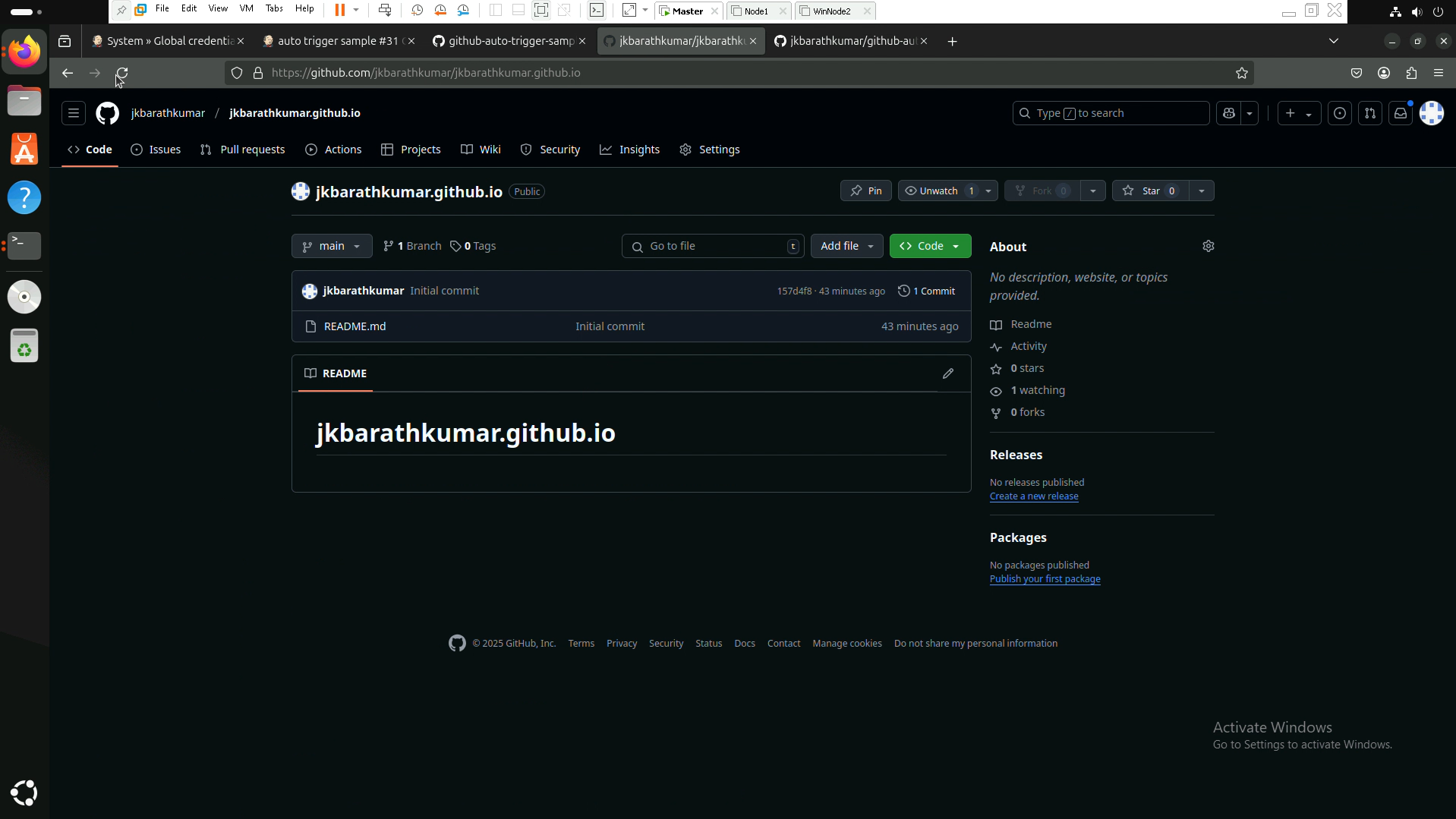
1. Code Push: When a developer pushes new code to the repository on GitHub.
2. Trigger Pipeline: A GitHub Actions pipeline will automatically be triggered by the push event.
3. Build and Test: The pipeline will build the code and run necessary tests.
4. Deploy to GitHub Pages: Upon successful build and test, the code will be deployed to GitHub Pages, where it will be publicly accessible.



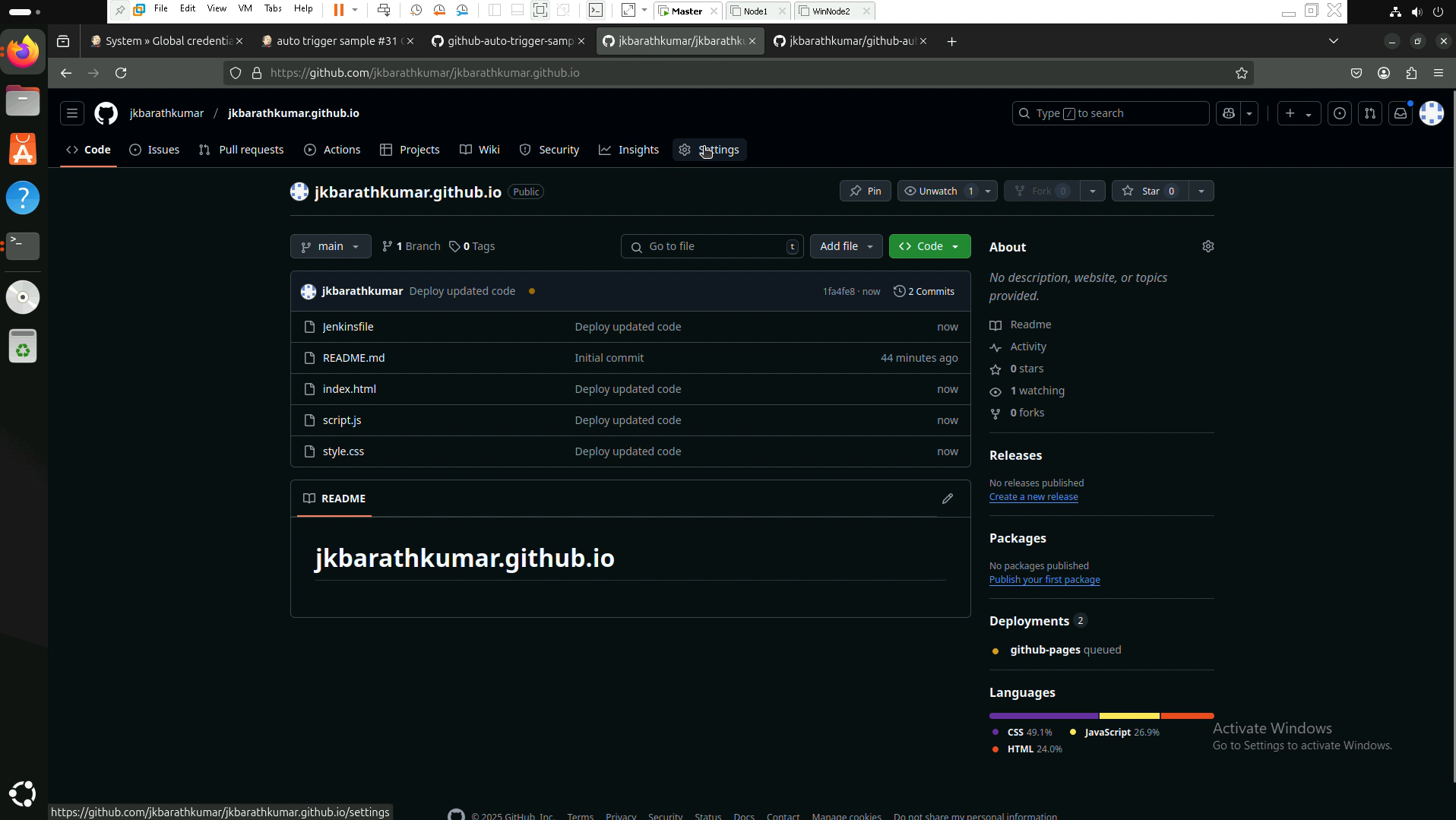




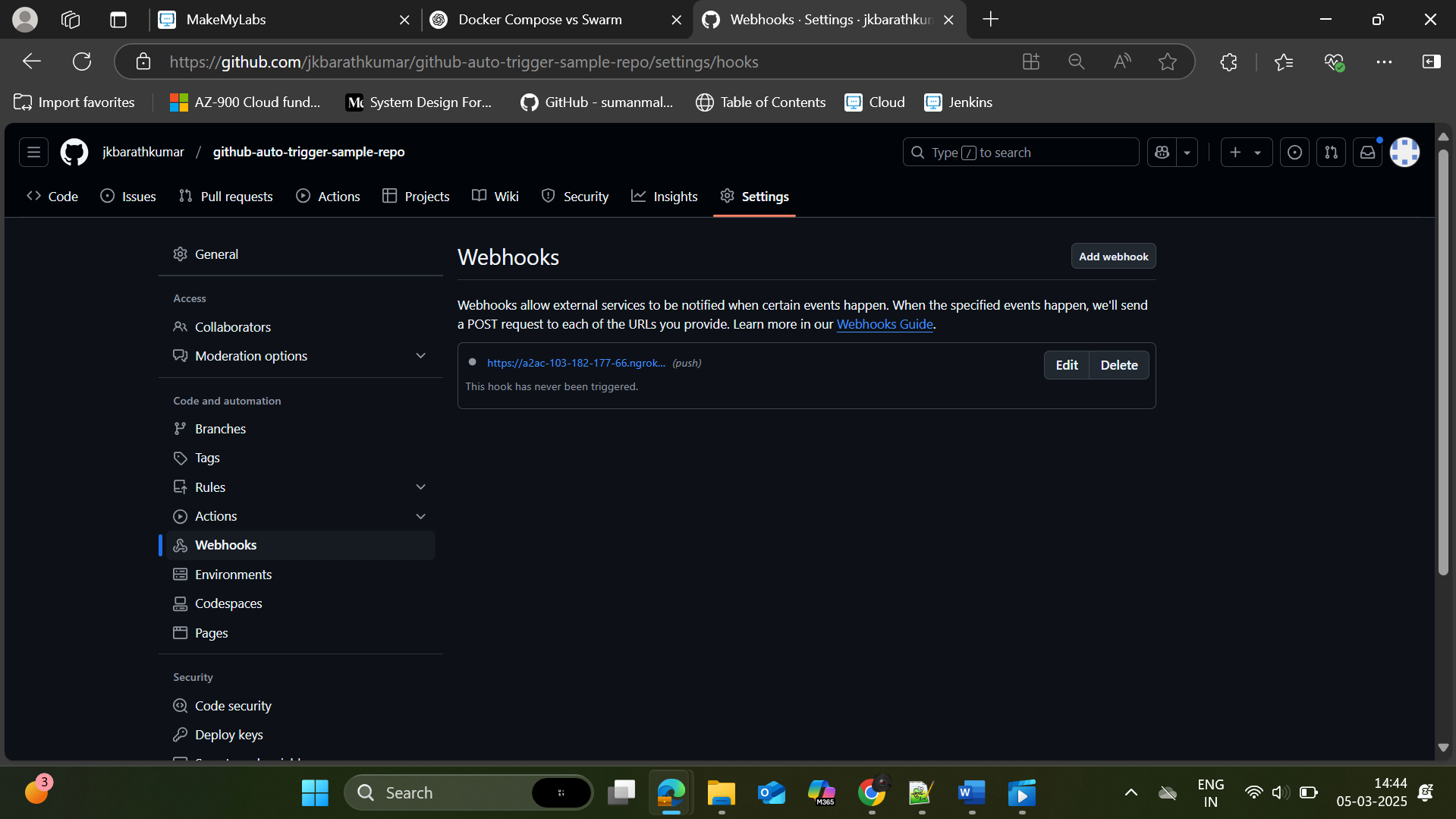
**Before deploying:**



**After deploying:**

****

**Github webhooks setup for auto trigger pipeline**

****

****