Automated CI/CD Pipeline for a Dockerized Weather Forecast App Using Jenkins

Report submitted by
Mugle Sruthi

- This title clearly conveys the purpose of the project—automating the CI/CD process for a Dockerized weather app using Jenkins. It highlights the core technologies involved and gives an idea of the project's technical scope.

- This project is a Dockerized Flask-based Weather Forecast Web Application that allows users to check current weather conditions for any city using a clean, responsive interface. The frontend is built with HTML and CSS, offering a visually appealing experience with dynamic weather information such as temperature, humidity, wind speed, and pressure. The backend is powered by Python and Flask, which fetches real-time weather data from the OpenWeatherMap API using an API key stored securely in a `.env` file. Users can enter a city name, and the app will display the corresponding weather details with icons, descriptions, and other relevant data.

- To ensure smooth deployment and scalability, the entire application is containerized using Docker. A `Dockerfile` defines the setup for building the image, which is then automatically built and pushed to Docker Hub using a Jenkins CI/CD pipeline. This pipeline pulls the latest code from GitHub, builds the Docker image, authenticates with Docker Hub using stored credentials, and deploys the application to a server by running it inside a Docker container. This setup not only simplifies deployment but also ensures consistency across different environments. Overall, the project demonstrates seamless integration of web development, API usage, containerization, and DevOps practices.

- **Docker** is a platform that enables developers to package applications and their dependencies into containers, ensuring that the application runs consistently across various environments. Containers are lightweight, portable, and isolated, which makes them ideal for ensuring that an application behaves the same on a developer's local machine as it does in production. Docker eliminates the "it works on my machine" problem, allowing applications to be easily moved between different systems and cloud environments. With Docker, developers can define the entire environment, including the operating system, libraries, and configuration files, using a simple `Dockerfile`, making the process of setting up applications more streamlined and reproducible.

- **Jenkins** is an open-source automation server that facilitates continuous integration (CI) and continuous delivery (CD) in the software development lifecycle. It automates the process of building, testing, and deploying applications, which helps improve development efficiency and product quality. Jenkins integrates with numerous plugins, enabling support for a wide variety of tools and technologies, including version control systems like Git, build tools like Maven or Gradle, and deployment platforms like Docker. In the context of CI/CD, Jenkins allows developers to automatically trigger builds when code **c**hanges are pushed to a repository, run automated tests to ensure the application works correctly, and deploy the application to production or staging environments seamlessly.

Entire explanation about project
work flow of project

```
weather-app/
├── __pycache__/
├── static/
├── templates/
├── .env
├── Dockerfile
├── Jenkinsfile
├── README.md
├── app.py
├── config.py
├── requirements.txt
```

Explanation:

- `__pycache__/`: Stores compiled Python files (usually auto-generated).
- `static/`: Typically holds CSS, JS, and image files for the frontend.
- `templates/`: Contains HTML files for rendering via Flask/Jinja.
- `.env`: Stores environment variables (API keys, secrets, etc.).
- `Dockerfile`: Used to containerize the application.
- `Jenkinsfile`: Defines Jenkins CI/CD pipeline steps.
- `README.md`: Project documentation.
- `app.py`: Main application script (likely Flask-based).
- `config.py`: Configuration settings.
- `requirements.txt`: Python dependencies.

Project code and its explanation

app.py

```
1.from flask import Flask, render_template, request
import requests
import os
from dotenv import load_dotenv

load_dotenv()
```

- **`Flask`: The main web framework you're using.**

- `render_template`: Renders your HTML (from `templates/index.html`).
- `request`: Allows access to form data (like the city name).
- `requests`: Makes HTTP requests to the weather API.
- `os`: Accesses environment variables (like the API key).
- `load_dotenv()`: Loads the `.env` file where your API key is stored.

```
2.app = Flask(__name__)
```

Initializes your Flask web app.
So that it works

- **Home Route `/`**

```
3.@app.route("/", methods=["GET", "POST"])
def index():
```

This function handles both GET (default) and POST requests to the root URL (`/`).

- **Handle Form Submission and Fetch Weather Data**

```
 4.   weather = {}
    error = None

    if request.method == "POST":
        city = request.form.get("city")
```

`weather`: Dictionary to store weather data.

- `error`: To store error messages (if any).
- `request.form.get("city")`: Gets the city input from the form.

- **Call the OpenWeatherMap API**

```
5.      if city:

          api_key = os.getenv("WEATHER_API_KEY")

          url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}
&units=metric"
```

Retrieves your API key from the `.env` file.

- Constructs the **URL** for fetching weather data (in metric units).

- **Send the Request and Process the Response**

```
6.          try:
             response = requests.get(url)
             response.raise_for_status()
            data = response.json()
```

Sends an HTTP GET request to the API.

- `raise_for_status()`: Raises an error if the API call failed.
- `data`: The JSON response from the weather API.

- **If Successful, Extract Weather Data**

```
7.          if data.get("main"):
             weather = {
                "city": data["name"],
                "temp": data["main"]["temp"],
                "feels_like": data["main"]["feels_like"],
                "humidity": data["main"]["humidity"],
                "description": data["weather"][0]["description"].capitalize(),
                "icon_url": f"http://openweathermap.org/img/wn/{data['weather'][0]['icon']}@2x.png",
                "wind_speed": data["wind"]["speed"],
                "pressure": data["main"]["pressure"]
             }
```

Checks if the response contains weather data (`main`).

- Extracts key details like:
    - Temperature, humidity, weather condition (description), wind speed, etc.
- `icon_url`: Constructs a URL for the weather icon.

- **If the City Wasn't Found or Request Fails**

```
8.          else:
             error = "City not found. Please try another location."
```

```
        except requests.exceptions.RequestException as e:
            error = "Unable to connect to weather service. Please try again later."
```

- ➢ **Render HTML Template with Weather Data or Error**

9. return render_template("index.html", weather=weather, error=error)

Passes the `weather` dictionary and any `error` message to the `index.html` template for display.

- ➢ **Run the App**

```
10.if __name__ == '__main__':
   app.run(host='0.0.0.0', port=5000, debug=True)
```

Starts the app on port `5000`.

- `host='0.0.0.0'`: Makes it accessible externally (e.g., in Docker).
- `debug=True`: Enables debug mode (for development only).

Index.html explanation

This is a Jinja2 template (used by Flask) that:

- Displays a **form** for entering a city name.
- Shows either an **error message** or **weather information**, depending on what Flask sends.

🔍 Code Breakdown

📌 `<head>` Section

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Weather Forecast</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css') }}">
</head>
```

- Sets up character encoding and responsive design.
- Sets the page title.
- Loads your **CSS file** from the `static/css/styles.css` path using `{{ url_for() }}` — this is how Flask dynamically generates URLs.

📦 Main Container

```
<body>
    <div class="weather-container">
        <h1>Weather Forecast</h1>
```

- Wraps everything in a `div` with a class `weather-container` — helpful for styling.

🧾 Weather Form

```
        <form method="post">
            <input type="text" name="city" placeholder="Enter city name" required>
            <button type="submit">Get Forecast</button>
        </form>
```

- Creates a form that submits via `POST` method (to the `/` route).
- User inputs the **city name** here.
- The `required` attribute ensures the input isn't empty.

❌ Error Message Handling

```
        {% if error %}
            <p class="error-message">{{ error }}</p>
        {% endif %}
```

- Checks if an `error` variable was sent from Flask.
- If yes, it shows the error in a paragraph with the class `error-message`.

🌦️ Weather Data Display

```
        {% if weather %}
            <div class="weather-info">{{ weather.temp }}°C</div>
```

- Checks if the `weather` dictionary was sent.
- Displays the **temperature** (e.g., `26°C`).

```
            <div class="weather-detail">
                <img class="weather-icon" src="{{ weather.icon_url }}" alt="Weather
Icon">
                <p>{{ weather.city }} - {{ weather.description }}</p>
                <p>Feels like: {{ weather.feels_like }}°C</p>
                <p>Humidity: {{ weather.humidity }}%</p>
            </div>
```

- Shows:
  - Weather icon (`icon_url`)
  - City and description (like “Hyderabad - Light rain”)
  - “Feels like” temperature
  - Humidity

```
            <div class="weather-footer">
                <div>Wind: {{ weather.wind_speed }} m/s</div>
                <div>Pressure: {{ weather.pressure }} hPa</div>
            </div>
        {% endif %}
```

- Displays:
  - Wind speed
  - Atmospheric pressure
- All values are pulled from the `weather` dictionary in your Flask route.

Style.css explanation

🧱 body Styling

```
body {
    margin: 0;
    padding: 0;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background-color: #f0f8ff;
    color: #333;
    background-image: url("../images/w_img.jpeg");
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    min-height: 100vh;
}
```

- **Font:** Uses modern, clean fonts.
- **Background:** A **full-screen image** (`w_img.jpeg`) with no repeats, centered, and covers entire screen.
- **Min height:** Ensures it fills the whole height of the viewport.

- **Color:** Default text color is dark gray `#333` for readability.

📦 `.weather-container`

```
.weather-container {
    max-width: 450px;
    margin: 50px auto;
    background: rgba(255, 255, 255, 0.9);
    padding: 25px;
    border-radius: 15px;
    text-align: center;
    box-shadow: 0 6px 15px rgba(0, 0, 0, 0.1);
}
```

- Centers the container with a **max-width** of 450px.
- **Translucent white background** makes the content pop against the background image.
- Rounded corners (`border-radius`) + **soft shadow** = elegant card-like feel.

🔠 Header Style

```
.weather-container h1 {
    font-size: 2.2rem;
    color: #2c3e50;
    margin-bottom: 25px;
}
```

- Slightly large and bold heading.
- **Dark blue color** gives it a professional tone.

✍️ Form Elements

Input:

```
input[type="text"] {
    padding: 10px;
    width: 70%;
    border: 1px solid #ddd;
    border-radius: 5px;
    font-size: 1rem;
}
```

- Smooth input with padding, rounded corners, and neutral border.

Button:

```
button {
    padding: 10px 20px;
    background-color: #3498db;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 1rem;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #2980b9;
}
```

- **Blue button** with hover effect.
- Looks clickable and pleasant for users.

⚠️ Error Message

```
.error-message {
    color: #e74c3c;
    margin: 15px 0;
}
```

- Red-colored message for errors like “City not found”.

🌡️ Weather Information

Temperature:

```
.weather-info {
    font-size: 3.2rem;
    font-weight: bold;
    color: #e67e22;
    margin: 20px 0;
}
```

- Displays temp like `28°C` in **big bold orange text**.

Detailed Info:

```
.weather-detail {
    font-size: 1.2rem;
    color: #34495e;
    background: rgba(236, 240, 241, 0.9);
    padding: 15px;
    border-radius: 10px;
    display: inline-block;
    margin: 15px 0;
    width: 80%;
}
```

- Inside a **semi-transparent gray box**, nicely rounded and centered.

Weather Icon:

```
.weather-detail img {
    width: 90px;
    margin: 10px 0;
}
```

- Icon looks clean and proportional.

💨 Footer Info

```
.weather-footer {
    display: flex;
    justify-content: space-around;
    font-size: 1.1rem;
    margin-top: 20px;
    color: #7f8c8d;
}
```

- Displays **wind speed and pressure** side-by-side using Flexbox.
- Uses a soft gray tone.

Config.py explanation :
This short code snippet is used to securely load environment variables, specifically your API key, from a `.env` file. Here's a breakdown of each part:

📜 Line-by-Line Explanation

🔹 `import os`

- Imports Python’s built-in `os` module.
- It allows your script to interact with the operating system — here, specifically to access **environment variables**.

🔹 `from dotenv import load_dotenv`

- Imports the `load_dotenv()` function from the `python-dotenv` package.
- This function reads the `.env` file and loads its variables into the environment.

🔹 `load_dotenv()`

- Searches for a file named `.env` in the current directory and loads the variables into the environment.
- Example `.env` file content:

  ```
  ini
  CopyEdit
  API_KEY=abc123xyz456
  ```

🔹 `API_KEY = os.getenv("API_KEY")`

- Retrieves the value of the `API_KEY` from environment variables.
- If it's in your `.env` file, it now becomes available in your code without hardcoding it.

🔒 Why Use `.env` and `dotenv`?

- ✅ Keeps your **API key secret** (not exposed in code).
- ✅ Helps you **avoid committing** secrets to GitHub or sharing with others.
- ✅ Makes it easier to switch between **dev and production environments**.

Requirements.txt
Flask
python-dotenv
requests

.env file

✅ `.env` file

```
WEATHER_API_KEY=b7b7737cef72f3057a5a95cfde0c9823
#open source key
```

- `WEATHER_API_KEY` is the variable name.
- `b7b7737cef72f3057a5a95cfde0c9823` is your actual OpenWeatherMap API key.
- The `#open source key` is just a comment (for your reference).

🧠 How it works in your code

In `app.py`, this line:

```
api_key = os.getenv("WEATHER_API_KEY")
```

pulls that value from your environment, allowing you to use the API securely without hardcoding the key.

🔒 Note:

Even though this key is "open source" (probably from a free tier), **avoid sharing it publicly** if you're pushing your project to GitHub. Use `.gitignore` to ignore your `.env` file:

```
# .gitignore
.env
```

DOCKER FILE
🔧 Dockerfile Explanation

```
# Use official python image
FROM python:3.12-slim
```

- ✅ **Base Image**: Uses a lightweight official Python 3.12 image.
- `slim` means it's minimal — saves disk space and speeds up build.

Dockerfile

```
# Set working directory
WORKDIR /app
```

- ✅ Creates and switches into `/app` inside the Docker container.
- All subsequent commands (e.g., `COPY`, `RUN`) will operate from here.

Dockerfile

```
# Copy requirements first to leverage Docker cache
COPY requirements.txt .
```

- ✅ Copies only `requirements.txt` into the container first.
- This allows Docker to **cache** the dependencies. If you don't change `requirements.txt`, Docker won't reinstall them.

Dockerfile

```
# Install python dependencies
RUN pip install --no-cache-dir -r requirements.txt
```

- ✅ Installs Python packages listed in `requirements.txt`.
- `--no-cache-dir` reduces image size by skipping pip's cache.

Dockerfile

```
# Copy the rest of the application
COPY . .
```

- ✅ Copies all files from your local project into the container’s `/app`.

Dockerfile

```
# Expose the port the app runs on
EXPOSE 5000
```

- ✅ Opens port **5000** so Docker knows the app runs on it.
- This doesn’t actually publish it; that’s done via `docker run -p`.

Dockerfile

```
# Set environment variables
ENV FLASK_APP=app.py
ENV FLASK_ENV=production
```

- ✅ Sets environment variables:
    - `FLASK_APP=app.py`: Tells Flask which file to run.
    - `FLASK_ENV=production`: Disables debug mode (you can change to `development` if needed).

Dockerfile

```
# Run the application
CMD ["flask", "run", "--host=0.0.0.0"]
```

- ✅ This is the **default command** the container runs.
- `--host=0.0.0.0` makes Flask listen on all network interfaces (needed inside Docker so it's accessible from outside).

🔁 Summary Workflow:

1. Pull base image ✅
2. Set working directory ✅
3. Install dependencies ✅
4. Copy app files ✅
5. Expose port ✅
6. Run app ✅

Jenkins file
This `Jenkinsfile` defines a **Declarative Jenkins Pipeline** that automates the process of **building**, **pushing**, and **deploying** your Dockerized Flask weather app.

✅ pipeline block

```
pipeline {
    agent any
```

- **`agent any`**: Run the pipeline on any available Jenkins agent (worker node).

✅ Environment Variable

```
    environment {
        DOCKER_IMAGE = 'sruthimugle19/weather-app'
    }
```

- Declares a reusable **environment variable** (`DOCKER_IMAGE`) for your Docker Hub image name (e.g., `sruthimugle19/weather-app`).

✅ Stage 1: Checkout Code

```
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/Mugle-Sruthi/jenkins-docker-weather-app.git'
            }
        }
```

- Pulls the latest code from your GitHub repository (`main` branch).

- This is your app's source code that contains the Flask app, Dockerfile, etc.

✅ Stage 2: Build & Push Docker Image

```
        stage('Build & Push Docker Image') {
            steps {
                script {
                    docker.build("${env.DOCKER_IMAGE}:latest")
                    docker.withRegistry('https://registry.hub.docker.com', 'docker-hub-creds') {
                        docker.image("${env.DOCKER_IMAGE}:latest").push()
                    }
                }
            }
        }
```

Here's what happens:

1. **`docker.build(...)`**: Builds your Docker image using the Dockerfile in your repo.
2. **`docker.withRegistry(...)`**:
    - Authenticates with Docker Hub.
    - `'docker-hub-creds'` is the Jenkins **credential ID** for your Docker Hub login.
3. **`.push()`**: Pushes the image to your Docker Hub repository.

✅ Stage 3: Deploy the App

```
        stage('Deploy') {
            steps {
                sh 'docker stop weather-app || true'
                sh 'docker rm weather-app || true'
                sh "docker run -d -p 5000:5000 --env-file .env --name weather-app ${env.DOCKER_IMAGE}:latest"
            }
        }
    }
}
```

Explanation:

- **`docker stop weather-app || true`**: Tries to stop a running container named `weather-app`. If it doesn't exist, continue anyway (`|| true`).
- **`docker rm weather-app || true`**: Removes the stopped container if it exists.
- **`docker run ...`**: Starts a new container:

- `-d`: Detached mode.
- `-p 5000:5000`: Maps container's port 5000 to host's port 5000.
- `--env-file .env`: Loads environment variables (like your `WEATHER_API_KEY`) from the `.env` file.
- `--name weather-app`: Names the container.
- `IMAGE_NAME:latest`: Runs the image just built and pushed.

✅ Recap of Pipeline Flow:

1. 📥 Clone your project from GitHub.
2. 🐳 Build and push the Docker image to Docker Hub.
3. 🚀 Deploy the new image by running a fresh Docker container.

Now
building docker image
docker build -t weather-app .
Running container
docker run -p 5000:5000 --env-file .env weather-app
Tagging image
docker tag weather-app sruthimugle19/weather-app:latest
Docker login
Pushing image to docker hub
docker push sruthimugle19/weather-app:latest

pushing code to github
git init
git add .
git commit -m "Initial commit: Dockerized weather app with Jenkins CI/CD"
git remote add origin https://github.com/Mugle-Sruthi/jenkins-docker-weather-app.git
git branch -M main
git push -u origin main

1. Write Code → 2. Build Image → 3. Run Container → 4. Test in Browser

Working of jenkins
Jenkins pulls the latest code from GitHub automatically when changes are pushed.

It builds a fresh Docker image using the Dockerfile in the repo.

Then it deploys the containerized app.

Finally, the weather app becomes available in the browser at port 5000.

To demonstrate, I would:

Log into Jenkins

Locate our 'weather-app-pipeline' job

Click 'Build Now'

Watch the pipeline execute through each stage in the console output

Once complete, open a browser to http://server-ip:5000 to show the live weather app

The entire process is automated - from code push to live deployment."

Key Technical Points to Mention:

The pipeline is defined in the Jenkinsfile in our repo

Uses Docker for consistent environments

Follows CI/CD best practices (build → test → deploy)

Can be triggered manually or automatically via webhooks

****

```
# Install Java (required)
sudo apt update && sudo apt install openjdk-17-jdk

# Add Jenkins repo
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

# Install Jenkins
sudo apt update && sudo apt install jenkins
sudo systemctl start jenkins
```

****

Correct Jenkins Installation for Fedora

1. Install Java (Required)

```
sudo dnf install java-17-openjdk-devel
```

2. Add Jenkins Repository

```
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

```
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
```

3. Install Jenkins

```
sudo dnf install jenkins
```

4. Start Jenkins

```
sudo systemctl enable jenkins
sudo systemctl start jenkins
```

5. Access Jenkins

Get the initial admin password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Open in browser:

http://localhost:8080

1. Configure Jenkins for Your Project

A. Access Jenkins

Open Jenkins in your browser (typically http://localhost:8080).

Log in with the admin password (found in /var/lib/jenkins/secrets/initialAdminPassword or set during installation).

B. Install Required Plugins

Go to Manage Jenkins → Plugins → Available Plugins and install:

Docker Pipeline (to build/push images)

GitHub Integration (for webhooks)

Blue Ocean (optional, for a better UI)

C. Add Credentials

Docker Hub (to push images):

Go to Manage Jenkins → Credentials → System → Global Credentials.

Add a Username with password type credential with your Docker Hub username and password (or access token).

GitHub (if your repo is private):

Add a Secret text credential with a GitHub personal access token (with repo permissions).

2. Create a Jenkins Pipeline

A. New Pipeline Job

Click New Item → Name: weather-app-pipeline → Select Pipeline.

Under Pipeline:

Definition: Pipeline script from SCM

SCM: Git

Repository URL: https://github.com/Mugle-Sruthi/jenkins-docker-weather-app.git

Credentials: Select your GitHub credentials (if repo is private).

Branch: main

Script Path: Jenkinsfile (ensure this file exists in your repo root).

B. Sample Jenkinsfile
Create this file in your GitHub repo:

3. Set Up GitHub Webhooks (Auto-Trigger)
Go to your GitHub repo → Settings → Webhooks → Add Webhook.

Configure:

Payload URL: http://<your-local-ip>:8080/github-webhook/
(Find your IP with hostname -I on Linux/macOS or ipconfig on Windows).

Content Type: application/json

Events: Just the push event

Active: ✔️

4. Run Your Pipeline
Manual Trigger
In Jenkins, go to your pipeline job.

Click Build Now.

Auto-Trigger (Test)
Make a small change in your code (e.g., update README.md).

Push to GitHub:

bash

```
git add . && git commit -m "Trigger Jenkins build" && git push
```

Jenkins will automatically:

Pull the latest code.

Rebuild the Docker image.

Push to Docker Hub.

Redeploy the container.

5. Verify Deployment

Check running containers:

```
docker ps
```

Access your app:

Open http://localhost:5000 in your browser.

Troubleshooting

"Permission denied" on Docker commands?
Add the Jenkins user to the docker group:

bash

```
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
```

Webhook not triggering?
Check Jenkins logs:

bash

```
sudo tail -f /var/log/jenkins/jenkins.log
```

Docker Hub push fails?
Verify credentials in Jenkins and ensure the image name matches your Docker Hub repo.

Next Steps

Add Tests: Include a stage('Test') in your Jenkinsfile (e.g., run pytest).

Secure Secrets: Use Jenkins' "Credentials Binding" for .env variables.

Monitor: Set up email notifications for build failures.

Conclusion

By completing this project, we have achieved several key goals that demonstrate proficiency in both web development and DevOps practices:

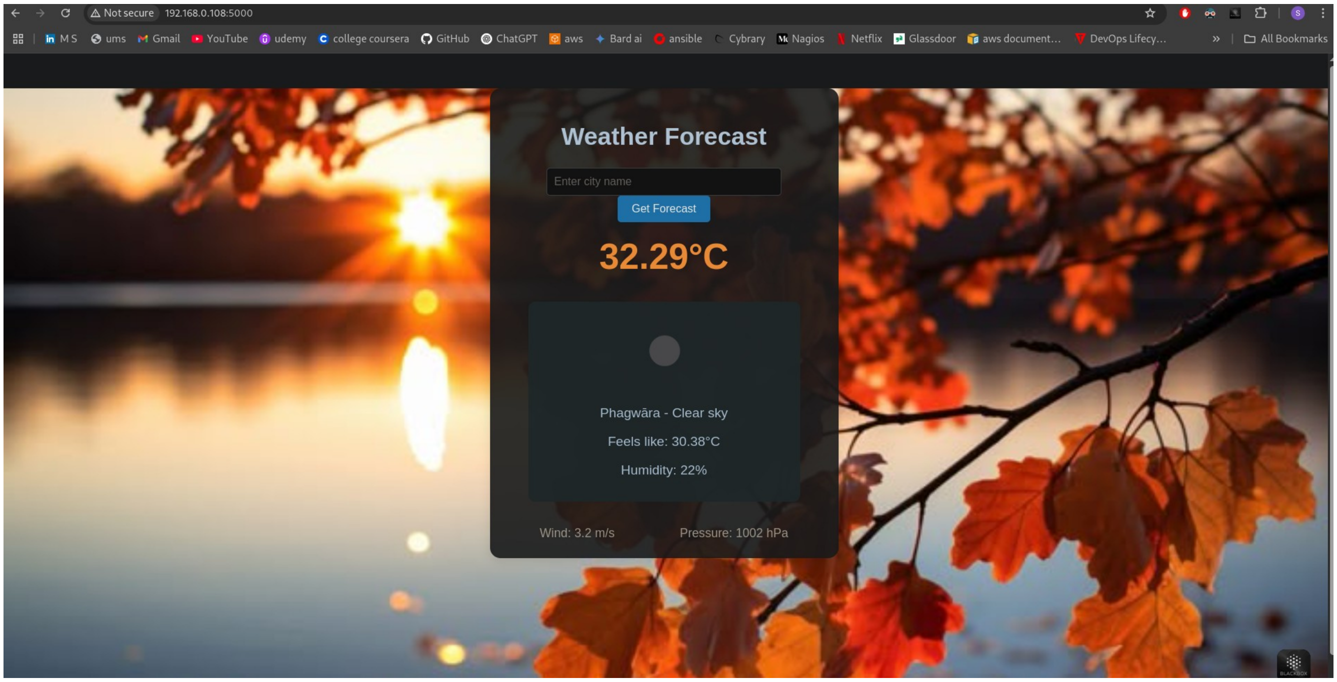
1. **Dockerization of a Web Application**: We successfully containerized a Flask-based weather forecast application using Docker. This ensures that the application runs consistently across different environments, eliminating configuration issues and simplifying deployment. By defining a `Dockerfile`, we packaged the application with all its dependencies into a single container, making it portable and easy to deploy anywhere.
2. **CI/CD Pipeline with Jenkins**: We set up an automated Continuous Integration and Continuous Deployment (CI/CD) pipeline using Jenkins. This pipeline automates the process of pulling

code from GitHub, building the Docker image, pushing it to Docker Hub, and deploying it to a server. This not only streamlines the development workflow but also ensures that new changes are tested, built, and deployed efficiently without manual intervention, significantly improving productivity and reducing the risk of errors.

3. **Real-Time Weather Data Integration**: The application fetches real-time weather data from the OpenWeatherMap API and presents it in an easy-to-read format. This integration ensures that users can access up-to-date weather information for any city, showcasing our ability to work with external APIs and dynamically render data.
4. **Scalable and Automated Deployment**: By leveraging Docker and Jenkins, we established an environment where the app can be easily scaled and deployed on different servers or cloud platforms. This not only reduces the complexity of deployment but also provides a reliable and automated process for maintaining and updating the application in production.

Overall, this project demonstrates the integration of modern web development, containerization, and DevOps best practices, enhancing both development efficiency and the ability to deliver reliable software at scale.

Images :



Weather Forecast

32.29°C

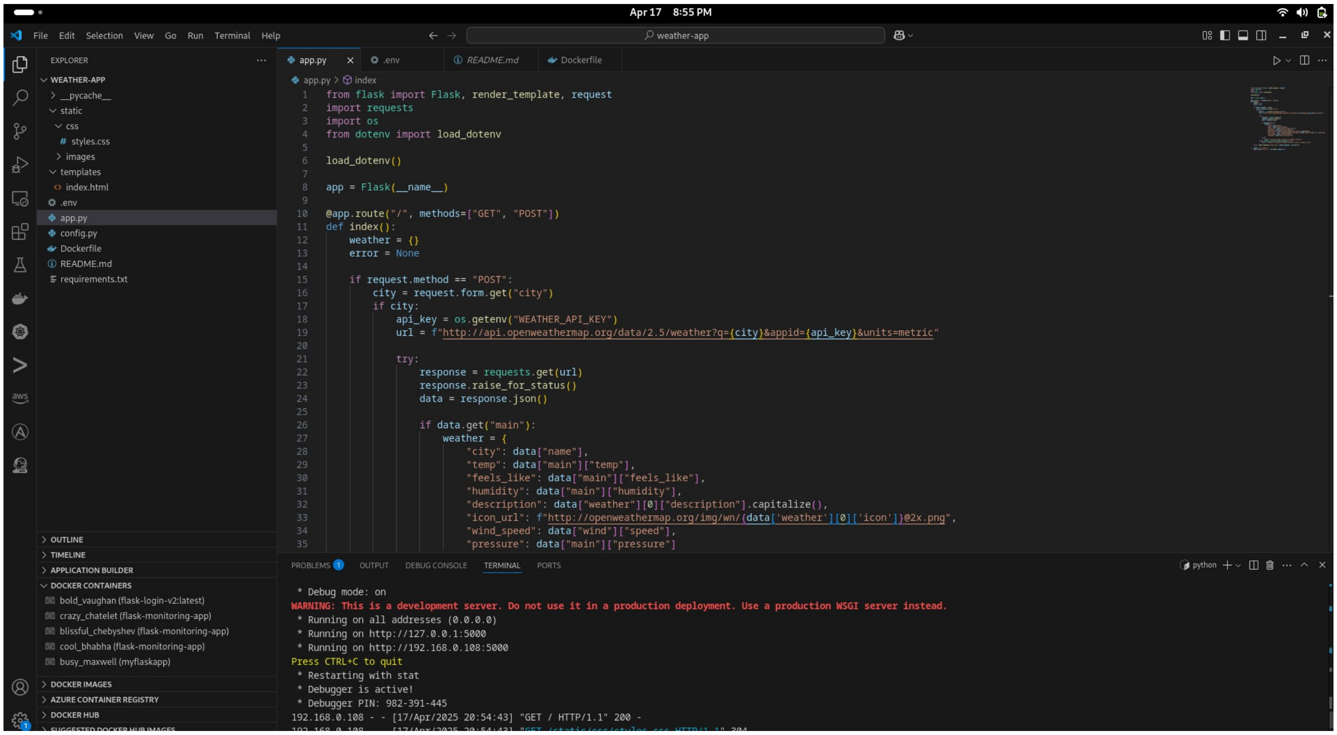
Phagwāra - Clear sky

Feels like: 30.38°C

Humidity: 22%

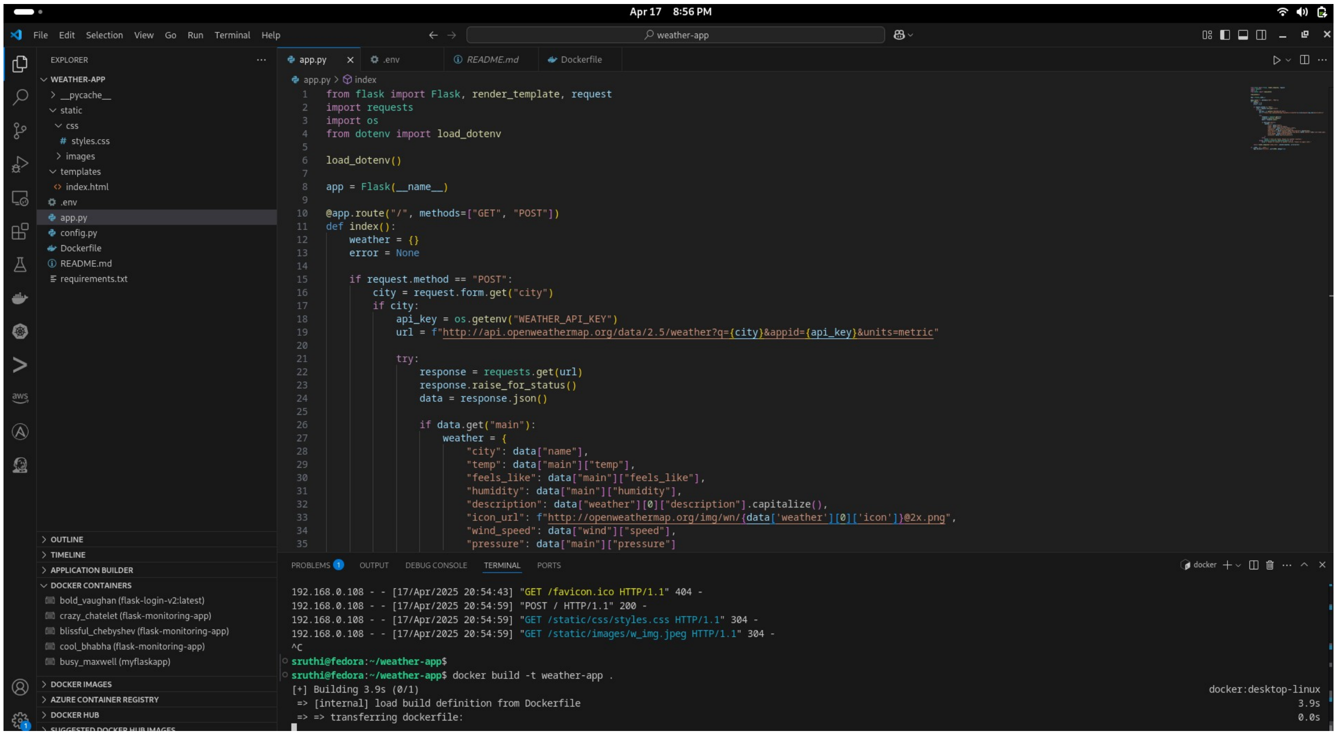
Wind: 3.2 m/s

Pressure: 1002 hPa



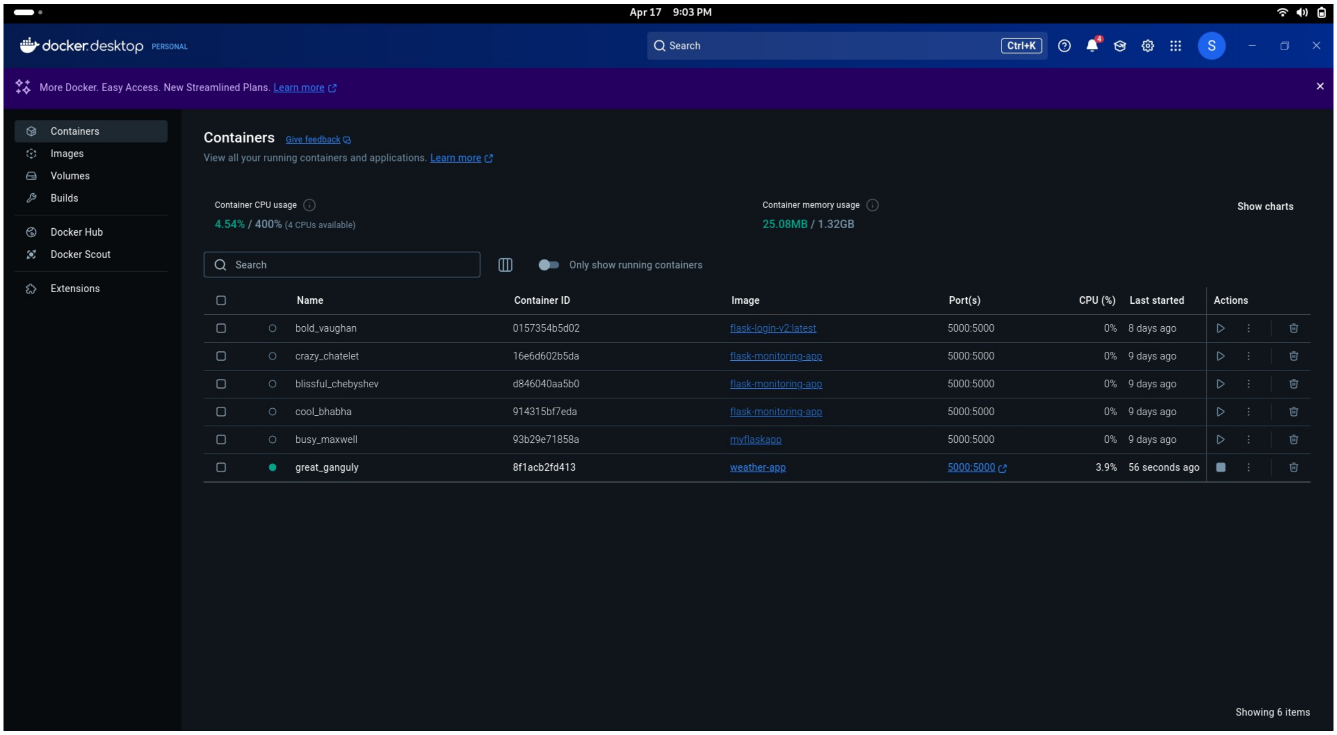
```python
from flask import Flask, render_template, request
import requests
import os
from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    weather = {}
    error = None

    if request.method == "POST":
        city = request.form.get("city")
        if city:
            api_key = os.getenv("WEATHER_API_KEY")
            url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"

            try:
                response = requests.get(url)
                response.raise_for_status()
                data = response.json()

                if data.get("main"):
                    weather = {
                        "city": data["name"],
                        "temp": data["main"]["temp"],
                        "feels_like": data["main"]["feels_like"],
                        "humidity": data["main"]["humidity"],
                        "description": data["weather"][0]["description"].capitalize(),
                        "icon_url": f"http://openweathermap.org/img/wn/{data['weather'][0]['icon']}@2x.png",
                        "wind_speed": data["wind"]["speed"],
                        "pressure": data["main"]["pressure"]
```

```
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.108:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 982-391-445
192.168.0.108 - - [17/Apr/2025 20:54:43] "GET / HTTP/1.1" 200 -
192.168.0.108 - - [17/Apr/2025 20:54:43] "GET /static/css/styles.css HTTP/1.1" 304
```



```python
from flask import Flask, render_template, request
import requests
import os
from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    weather = {}
    error = None

    if request.method == "POST":
        city = request.form.get("city")
        if city:
            api_key = os.getenv("WEATHER_API_KEY")
            url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"

            try:
                response = requests.get(url)
                response.raise_for_status()
                data = response.json()

                if data.get("main"):
                    weather = {
                        "city": data["name"],
                        "temp": data["main"]["temp"],
                        "feels_like": data["main"]["feels_like"],
                        "humidity": data["main"]["humidity"],
                        "description": data["weather"][0]["description"].capitalize(),
                        "icon_url": f"http://openweathermap.org/img/wn/{data['weather'][0]['icon']}@2x.png",
                        "wind_speed": data["wind"]["speed"],
                        "pressure": data["main"]["pressure"]
```

```
192.168.0.108 - - [17/Apr/2025 20:54:43] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.108 - - [17/Apr/2025 20:54:59] "POST / HTTP/1.1" 200 -
192.168.0.108 - - [17/Apr/2025 20:54:59] "GET /static/css/styles.css HTTP/1.1" 304 -
192.168.0.108 - - [17/Apr/2025 20:54:59] "GET /static/images/w_img.jpeg HTTP/1.1" 304 -
^C
sruthi@fedora:~/weather-app$
sruthi@fedora:~/weather-app$ docker build -t weather-app .
[+] Building 3.9s (0/1)                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                       3.9s
 => => transferring dockerfile:                                                            0.0s
```



Containers

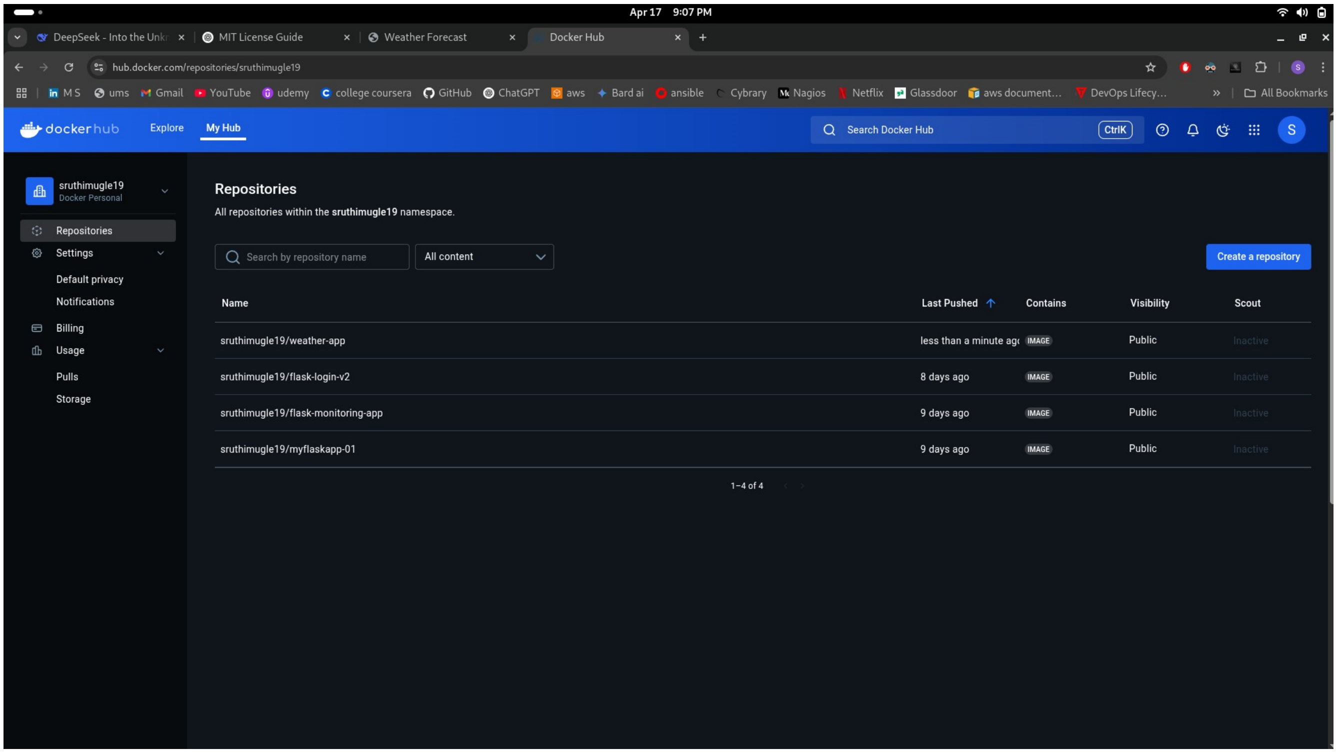
View all your running containers and applications. Learn more

Container CPU usage: 4.54% / 400% (4 CPUs available)

Container memory usage: 25.08MB / 1.32GB

☐		Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
☐	○	bold_vaughan	0157354b5d02	flask-login-v2:latest	5000:5000	0%	8 days ago	
☐	○	crazy_chatelet	16e6d602b5da	flask-monitoring-app	5000:5000	0%	9 days ago	
☐	○	blissful_chebyshev	d846040aa5b0	flask-monitoring-app	5000:5000	0%	9 days ago	
☐	○	cool_bhabha	914315bf7eda	flask-monitoring-app	5000:5000	0%	9 days ago	
☐	○	busy_maxwell	93b29e71858a	myflaskapp	5000:5000	0%	9 days ago	
☐	●	great_ganguly	8f1acb2fd413	weather-app	5000:5000	3.9%	56 seconds ago	

Showing 6 items



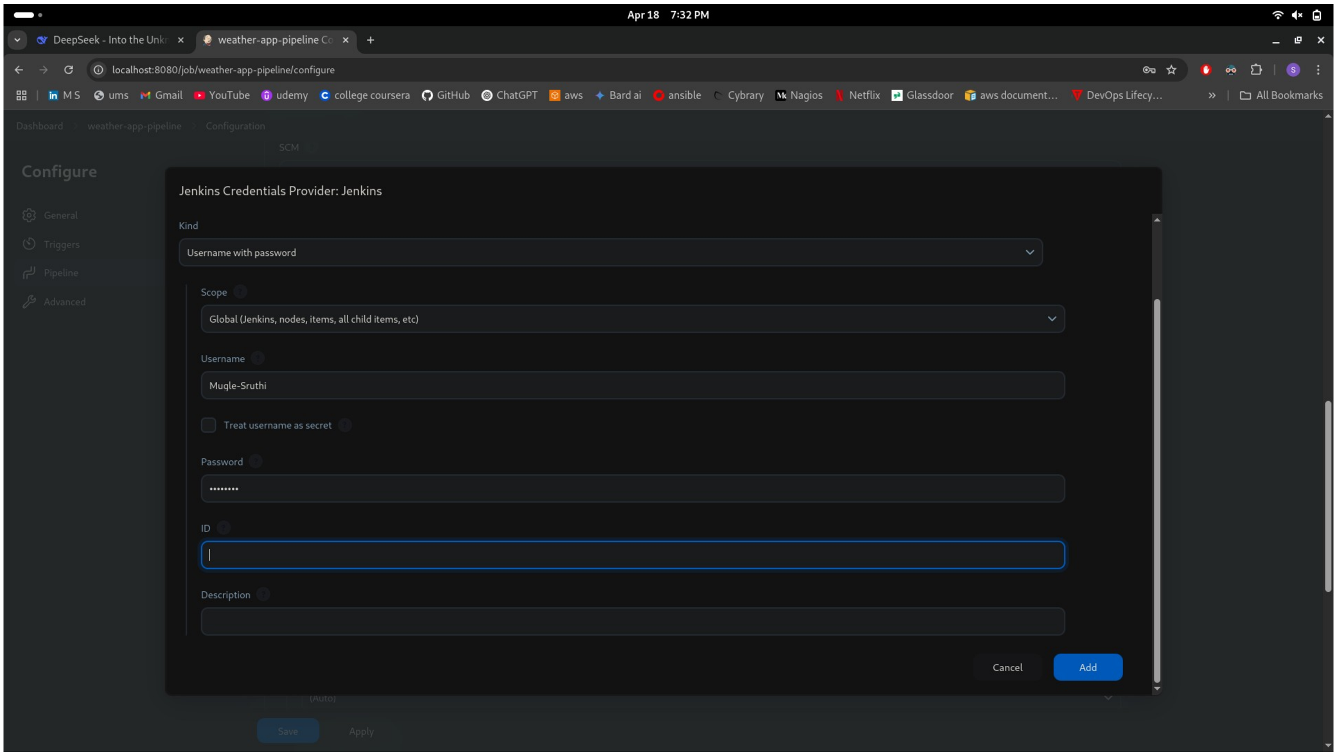
Repositories

All repositories within the **sruthimugle19** namespace.

Name	Last Pushed ↑	Contains	Visibility	Scout
sruthimugle19/weather-app	less than a minute ag	IMAGE	Public	Inactive
sruthimugle19/flask-login-v2	8 days ago	IMAGE	Public	Inactive
sruthimugle19/flask-monitoring-app	9 days ago	IMAGE	Public	Inactive
sruthimugle19/myflaskapp-01	9 days ago	IMAGE	Public	Inactive

1–4 of 4

Pipleline images



Jenkins Credentials Provider: Jenkins

Kind

Username with password

Scope

Global (Jenkins, nodes, items, all child items, etc)

Username

Muqle-Sruthi

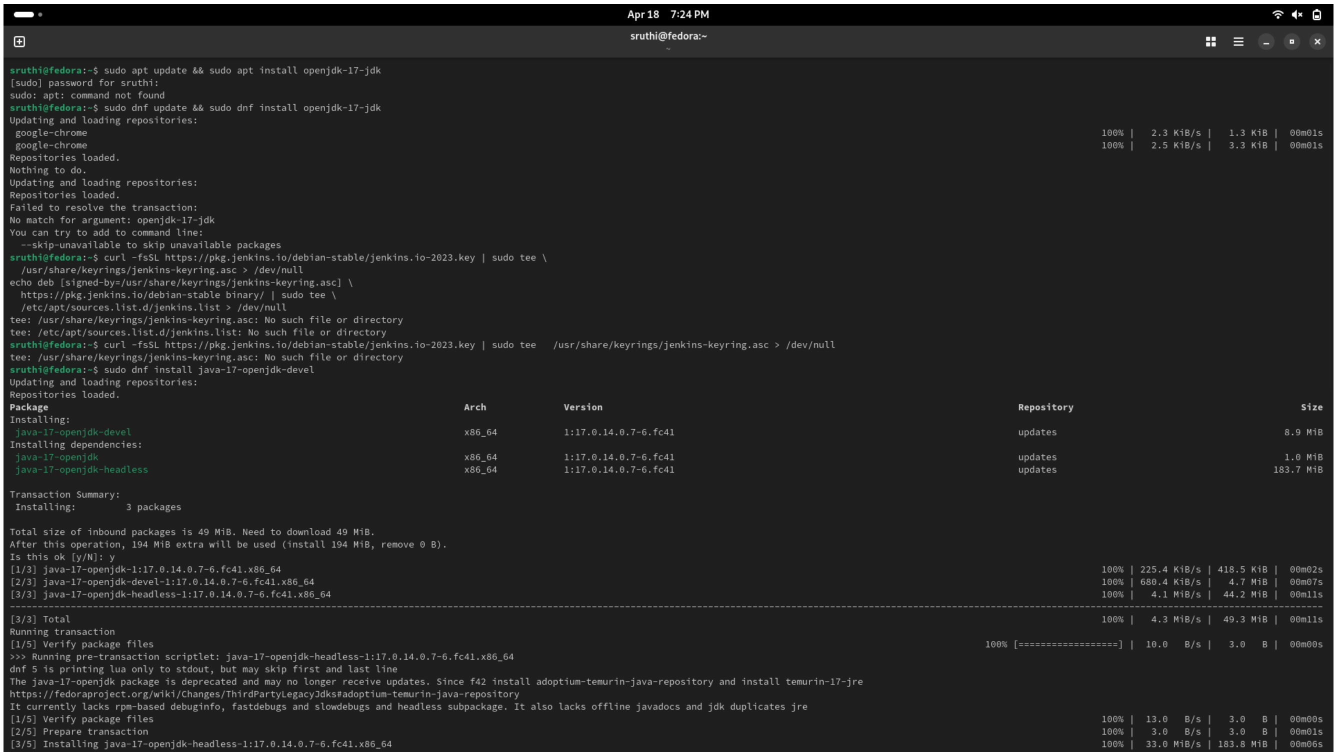
□ Treat username as secret

Password

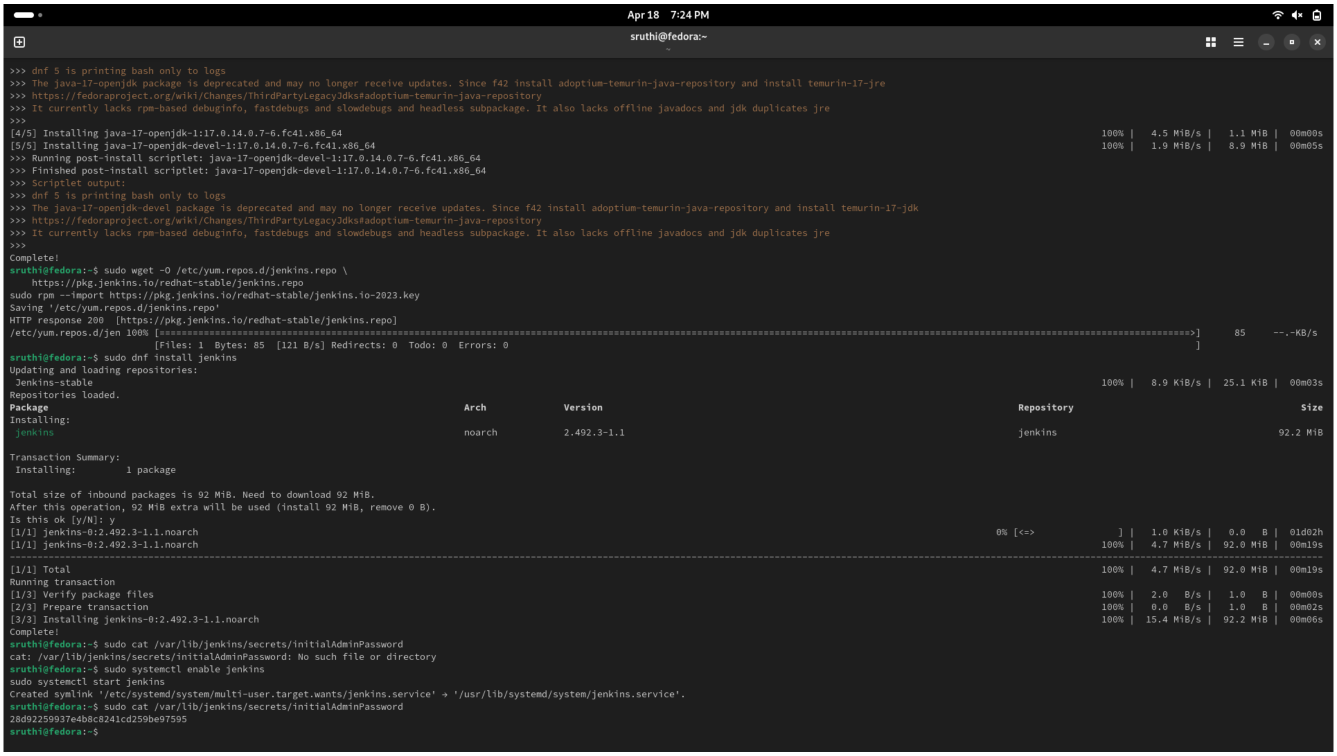
••••••••

ID

Description



```
sruthi@fedora:~$ sudo apt update && sudo apt install openjdk-17-jdk
[sudo] password for sruthi:
sudo: apt: command not found
sruthi@fedora:~$ sudo dnf update && sudo dnf install openjdk-17-jdk
Updating and loading repositories:
 google-chrome                                                   100% |   2.3 KiB/s |   1.3 KiB |  00m01s
 google-chrome                                                   100% |   2.5 KiB/s |   3.3 KiB |  00m01s
Repositories loaded.
Nothing to do.
Updating and loading repositories:
Repositories loaded.
Failed to resolve the transaction:
No match for argument: openjdk-17-jdk
You can try to add to command line:
  --skip-unavailable to skip unavailable packages
sruthi@fedora:~$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
tee: /usr/share/keyrings/jenkins-keyring.asc: No such file or directory
tee: /etc/apt/sources.list.d/jenkins.list: No such file or directory
sruthi@fedora:~$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee   /usr/share/keyrings/jenkins-keyring.asc > /dev/null
tee: /usr/share/keyrings/jenkins-keyring.asc: No such file or directory
sruthi@fedora:~$ sudo dnf install java-17-openjdk-devel
Updating and loading repositories:
Repositories loaded.
Package                              Arch         Version                      Repository          Size
Installing:
 java-17-openjdk-devel               x86_64       1:17.0.14.0.7-6.fc41         updates          8.9 MiB
Installing dependencies:
 java-17-openjdk                     x86_64       1:17.0.14.0.7-6.fc41         updates          1.0 MiB
 java-17-openjdk-headless            x86_64       1:17.0.14.0.7-6.fc41         updates        183.7 MiB

Transaction Summary:
 Installing:         3 packages

Total size of inbound packages is 49 MiB. Need to download 49 MiB.
After this operation, 194 MiB extra will be used (install 194 MiB, remove 0 B).
Is this ok [y/N]: y
[1/3] java-17-openjdk-1:17.0.14.0.7-6.fc41.x86_64                100% | 225.4 KiB/s | 418.5 KiB |  00m02s
[2/3] java-17-openjdk-devel-1:17.0.14.0.7-6.fc41.x86_64          100% | 680.4 KiB/s |   4.7 MiB |  00m07s
[3/3] java-17-openjdk-headless-1:17.0.14.0.7-6.fc41.x86_64       100% |   4.1 MiB/s |  44.2 MiB |  00m11s
--------------------------------------------------------------------------------------------------------
[3/3] Total                                                      100% |   4.3 MiB/s |  49.3 MiB |  00m11s
Running transaction
[1/5] Verify package files                  100% [=================] |  10.0   B/s |   3.0   B |  00m00s
>>> Running pre-transaction scriptlet: java-17-openjdk-headless-1:17.0.14.0.7-6.fc41.x86_64
dnf 5 is printing lua only to stdout, but may skip first and last line
The java-17-openjdk package is deprecated and may no longer receive updates. Since f42 install adoptium-temurin-java-repository and install temurin-17-jre
https://fedoraproject.org/wiki/Changes/ThirdPartyLegacyJdks#adoptium-temurin-java-repository
It currently lacks rpm-based debuginfo, fastdebugs and slowdebugs and headless subpackage. It also lacks offline javadocs and jdk duplicates jre
[1/5] Verify package files                                       100% |  13.0   B/s |   3.0   B |  00m00s
[2/5] Prepare transaction                                        100% |   3.0   B/s |   3.0   B |  00m01s
[3/5] Installing java-17-openjdk-headless-1:17.0.14.0.7-6.fc41.x86_64    100% |  33.0 MiB/s | 183.8 MiB |  00m06s
```



```
>>> dnf 5 is printing bash only to logs
>>> The java-17-openjdk package is deprecated and may no longer receive updates. Since f42 install adoptium-temurin-java-repository and install temurin-17-jre
>>> https://fedoraproject.org/wiki/Changes/ThirdPartyLegacyJdks#adoptium-temurin-java-repository
>>> It currently lacks rpm-based debuginfo, fastdebugs and slowdebugs and headless subpackage. It also lacks offline javadocs and jdk duplicates jre
>>>
[4/5] Installing java-17-openjdk-1:17.0.14.0.7-6.fc41.x86_64                                                                                                                 100% |   4.5 MiB/s |    1.1 MiB |  00m00s
[5/5] Installing java-17-openjdk-devel-1:17.0.14.0.7-6.fc41.x86_64                                                                                                           100% |   1.9 MiB/s |    8.9 MiB |  00m05s
>>> Running post-install scriptlet: java-17-openjdk-devel-1:17.0.14.0.7-6.fc41.x86_64
>>> Finished post-install scriptlet: java-17-openjdk-devel-1:17.0.14.0.7-6.fc41.x86_64
>>> Scriptlet output:
>>> dnf 5 is printing bash only to logs
>>> The java-17-openjdk-devel package is deprecated and may no longer receive updates. Since f42 install adoptium-temurin-java-repository and install temurin-17-jdk
>>> https://fedoraproject.org/wiki/Changes/ThirdPartyLegacyJdks#adoptium-temurin-java-repository
>>> It currently lacks rpm-based debuginfo, fastdebugs and slowdebugs and headless subpackage. It also lacks offline javadocs and jdk duplicates jre
>>>
Complete!
sruthi@fedora:~$ sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
Saving '/etc/yum.repos.d/jenkins.repo'
HTTP response 200  [https://pkg.jenkins.io/redhat-stable/jenkins.repo]
/etc/yum.repos.d/jen 100% [=================================================================================================================================================================================================>]      85     --.-KB/s
                          [Files: 1  Bytes: 85  [121 B/s] Redirects: 0  Todo: 0  Errors: 0                                                                                                                                 ]
sruthi@fedora:~$ sudo dnf install jenkins
Updating and loading repositories:
 Jenkins-stable                                                                                                                                                                                            100% |   8.9 KiB/s |   25.1 KiB |  00m03s
Repositories loaded.
Package                                                                              Arch               Version                                                                              Repository                                                     Size
Installing:
 jenkins                                                                             noarch             2.492.3-1.1                                                                          jenkins                                                    92.2 MiB

Transaction Summary:
 Installing:         1 package

Total size of inbound packages is 92 MiB. Need to download 92 MiB.
After this operation, 92 MiB extra will be used (install 92 MiB, remove 0 B).
Is this ok [y/N]: y
[1/1] jenkins-0:2.492.3-1.1.noarch                                                                                                                                                   0% [<=>            ] |   1.0 KiB/s |    0.0   B |  01d02h
[1/1] jenkins-0:2.492.3-1.1.noarch                                                                                                                                                         100% |   4.7 MiB/s |   92.0 MiB |  00m19s
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
[1/1] Total                                                                                                                                                                                100% |   4.7 MiB/s |   92.0 MiB |  00m19s
Running transaction
[1/3] Verify package files                                                                                                                                                                 100% |   2.0   B/s |    1.0   B |  00m00s
[2/3] Prepare transaction                                                                                                                                                                  100% |   0.0   B/s |    1.0   B |  00m02s
[3/3] Installing jenkins-0:2.492.3-1.1.noarch                                                                                                                                              100% |  15.4 MiB/s |   92.2 MiB |  00m06s
Complete!
sruthi@fedora:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
cat: /var/lib/jenkins/secrets/initialAdminPassword: No such file or directory
sruthi@fedora:~$ sudo systemctl enable jenkins
sudo systemctl start jenkins
Created symlink '/etc/systemd/system/multi-user.target.wants/jenkins.service' → '/usr/lib/systemd/system/jenkins.service'.
sruthi@fedora:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
28d92259937e4b8c8241cd259be97595
sruthi@fedora:~$
```

Thank you