Title

Build Jenkins Pipeline to auto deploy Flask app using Docker from GitHub

Objective

Build a CI/CD pipeline using Jenkins that:

- Clones a Flask app from GitHub
- Build a Docker Image of the app
- Deploy it locally via Docker container
- Is triggered by a GitHub Webhook (optional)

Procedure

This lab can be divided into 5 segments;

1. Jenkins Installation and Setup
2. GitHub Repo Creation
3. Jenkins Pipeline Creation
4. Code Deployment (Manual Trigger)
5. Auto Trigger using GitHub Webhook (optional)

1) Jenkins Installation and Setup

As we will be using docker daemon to build our code inside the Jenkins container; we need to have docker there (inside the Jenkins container). Therefore, we will first build a custom Jenkins image that would include the docker with it.

Save this in a `Dockerfile` and run the command;

```
docker build -t jenkins-with-docker .
```

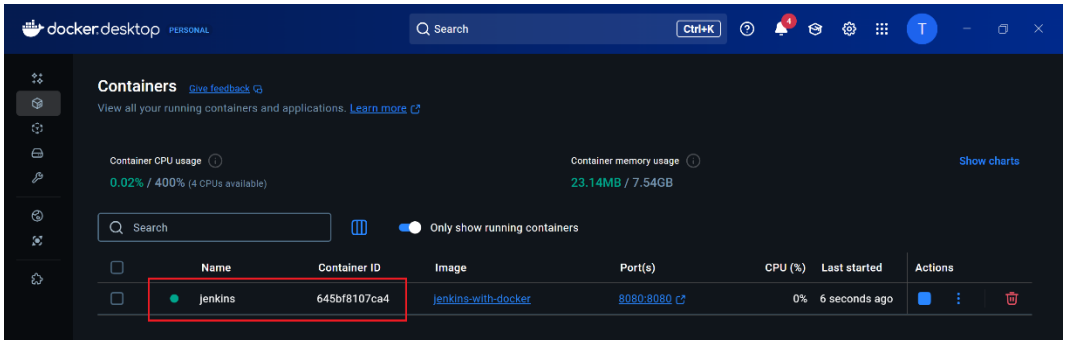
```
FROM jenkins/jenkins:lts
USER root
# Install Docker CLI inside the Jenkins Container
RUN apt-get update && \
    apt-get install -y docker.io && \
    apt-get clean
USER jenkins
```

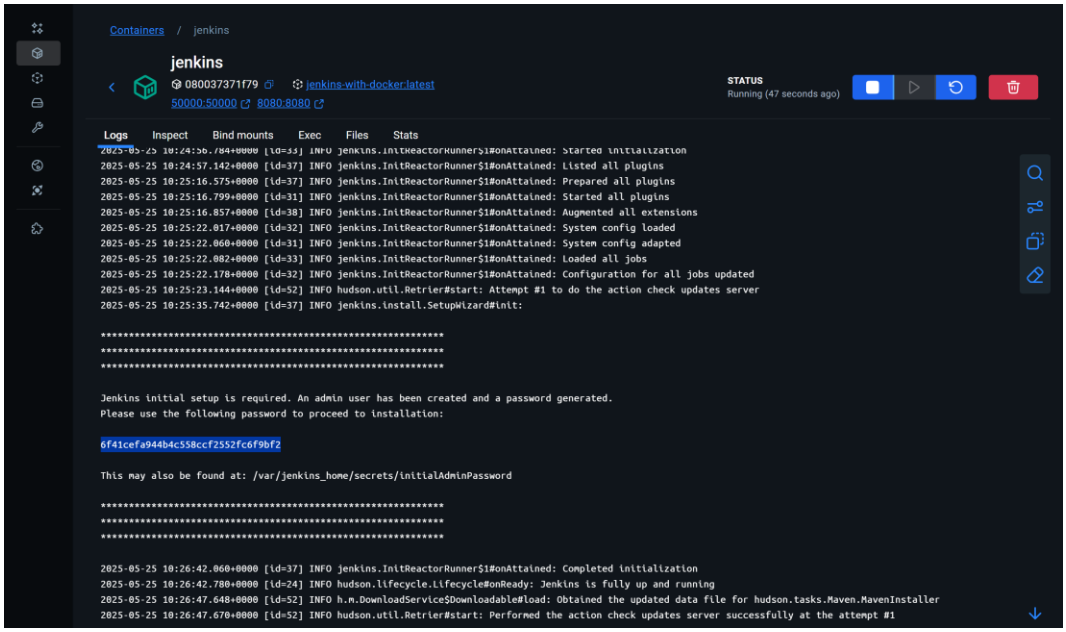
After the image has successfully build; run the following command to run the Jenkin container with custom Jenkins image.

`PowerShell / CommandPrompt:`

```
docker run -d --name jenkins -p 8080:8080 -v
/var/run/docker.sock:/var/run/docker.sock -v jenkins_data:/var/jenkins_home -u
root jenkins-with-docker
```

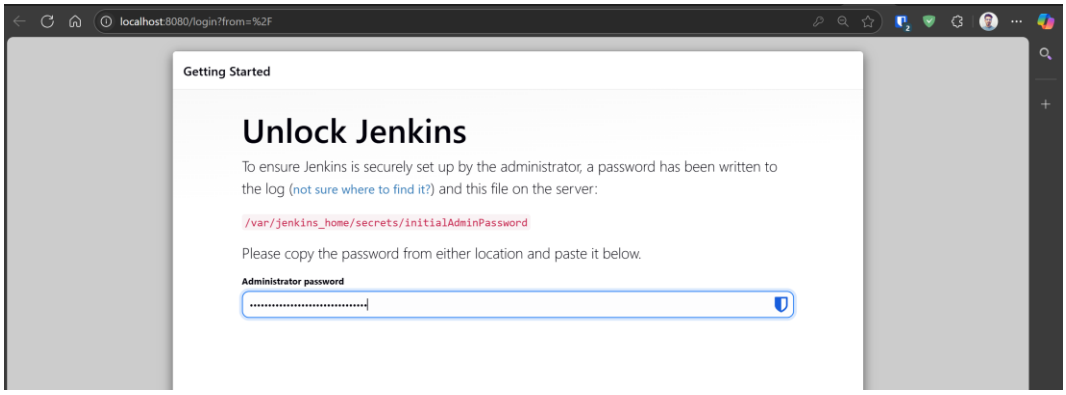
The Jenkins container will be started. Now go to Docker Desktop UI and click on the recently started container. It will open up the logs for that container. In the Jenkins logs, you will see the initial admin password. Copy that.

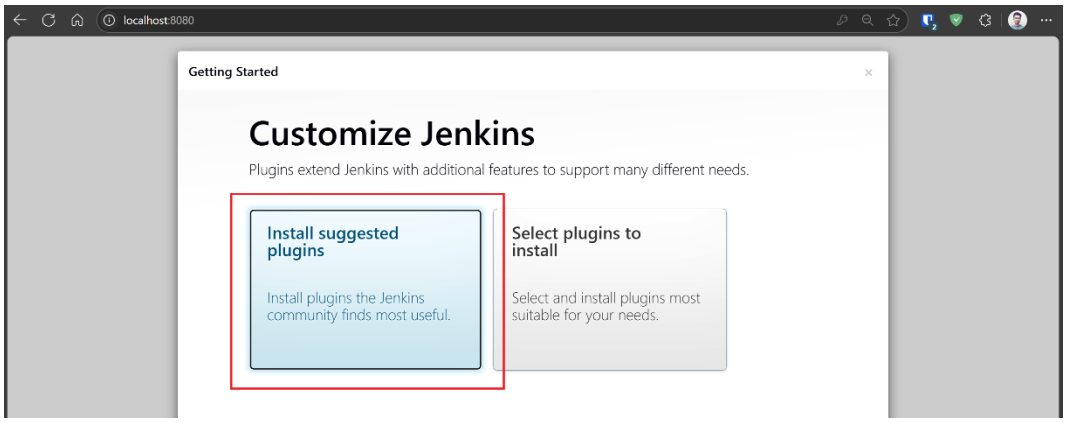


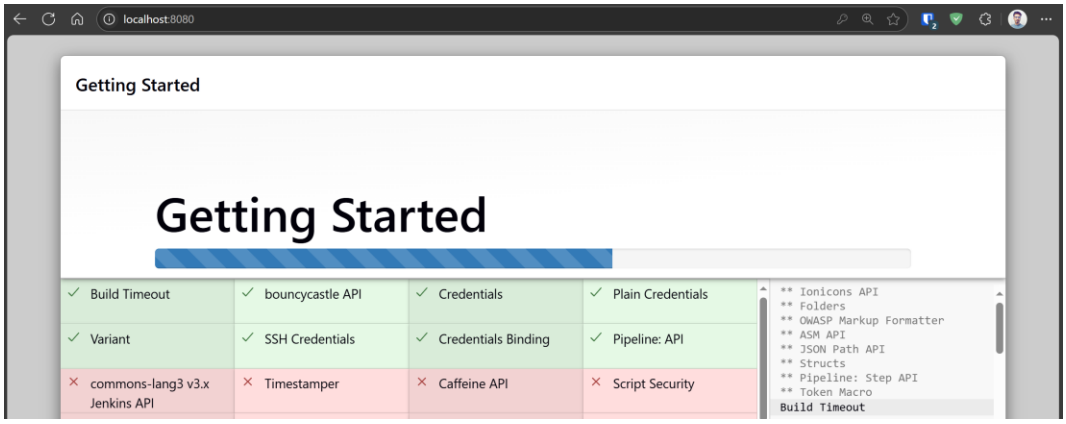


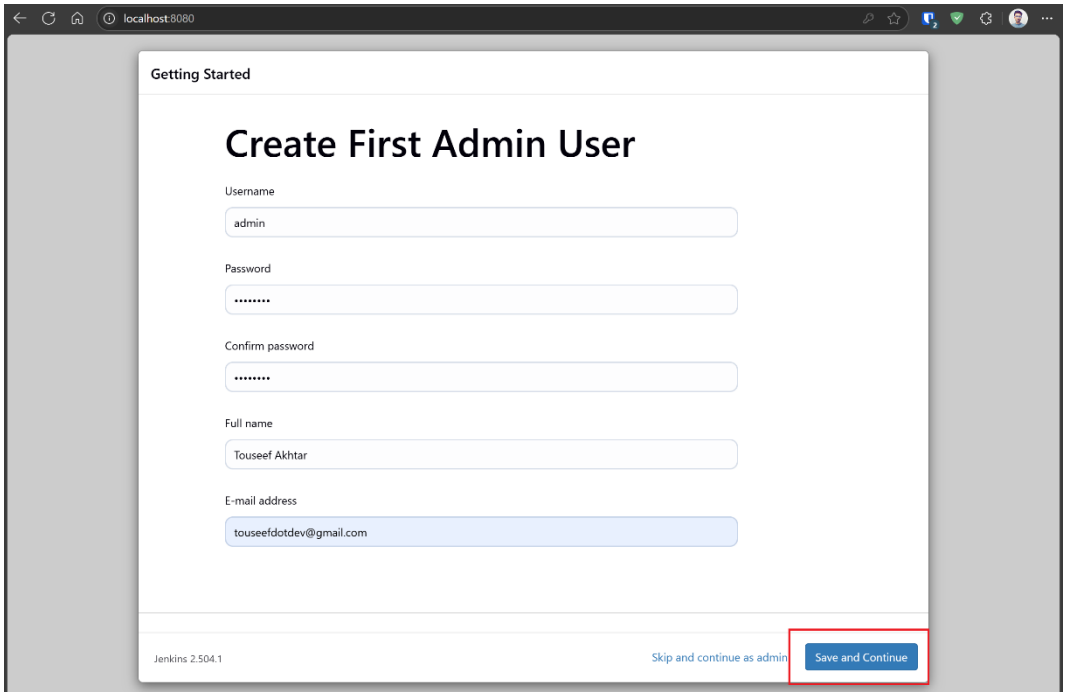
Now open up your browser and go to `localhost:8080`, you should be able to view the Jenkins interface.

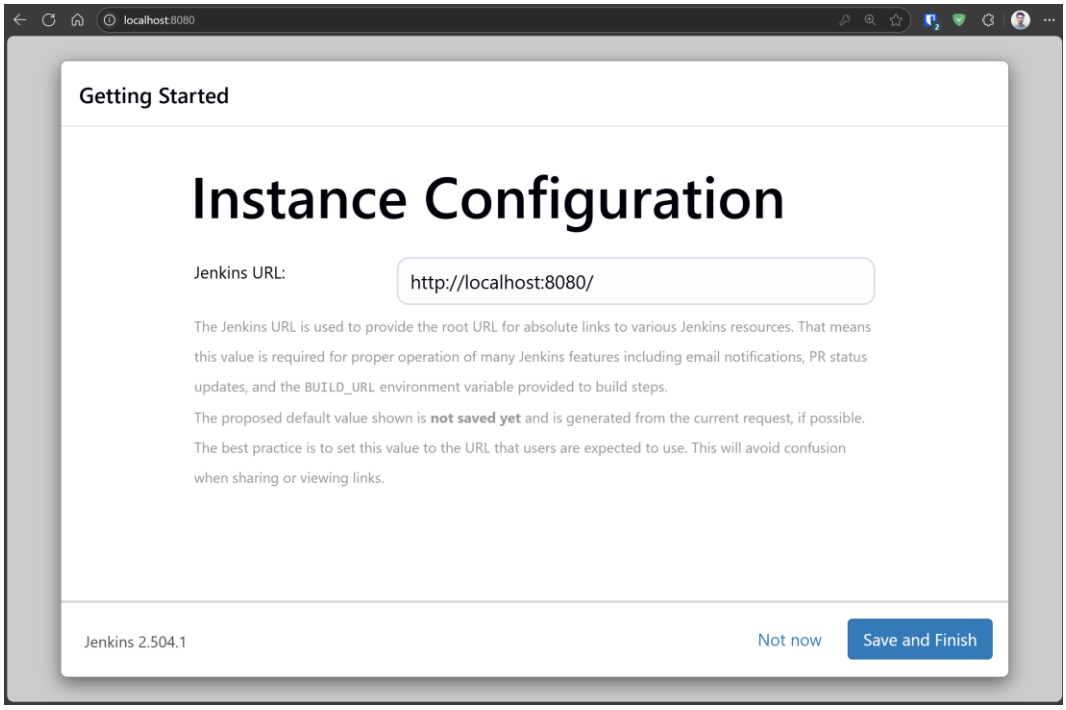
It will be asking for admin password, paste that initial admin password copied from logs here and proceed with the setup as demonstrated below with images.

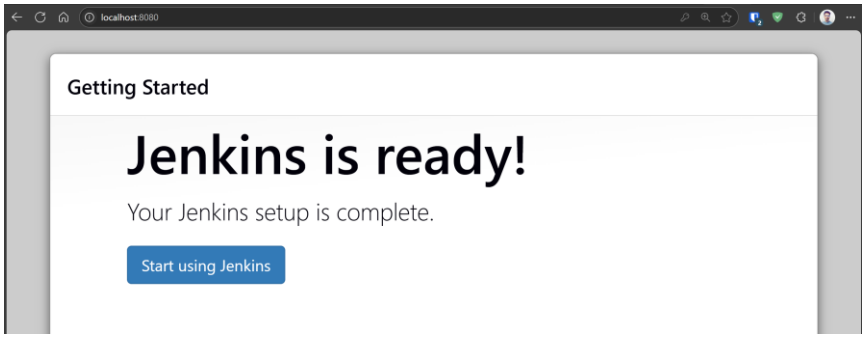

localhost:8080/login?from=%2F
Getting Started
Unlock Jenkins
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:
/var/jenkins_home/secrets/initialAdminPassword
Please copy the password from either location and paste it below.
Administrator password


localhost:8080
Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.


localhost:8080
Getting Started
Getting Started
Build Timeout
bouncycastle API
Credentials
Plain Credentials
Variant
SSH Credentials
Credentials Binding
Pipeline: API
commons-lang3 v3.x Jenkins API
Timestamper
Caffeine API
Script Security
** Ionicons API
** Folders
** OWASP Markup Formatter
** ASM API
** JSON Path API
** Structs
** Pipeline: Step API
** Token Macro
Build Timeout


localhost:8080
Getting Started
Create First Admin User
Username
admin
Password
Confirm password
Full name
Touseef Akhtar
E-mail address
touseefdotdev@gmail.com
Jenkins 2.504.1
Skip and continue as admin
Save and Continue


localhost:8080
Getting Started
Instance Configuration
Jenkins URL:
http://localhost:8080/
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.
The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.
Jenkins 2.504.1
Not now
Save and Finish



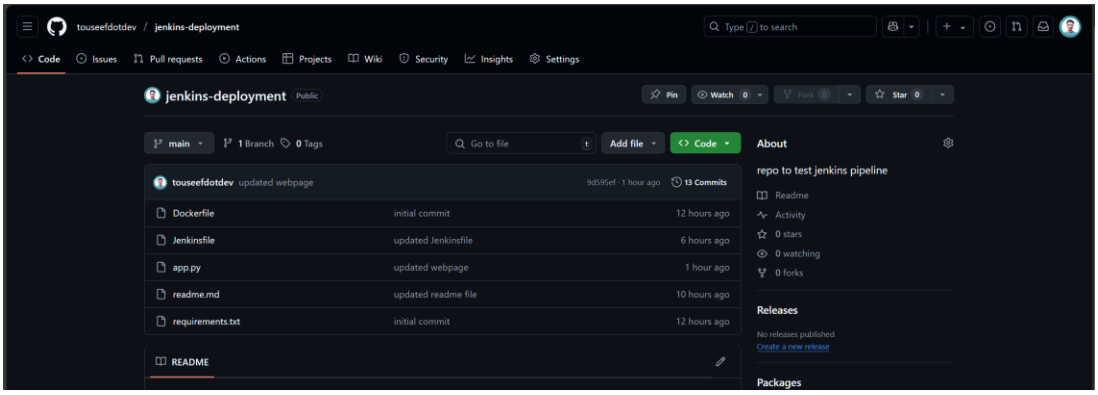
Now your Jenkins is ready to use. Let's proceed to the next step of the procedure.

2) GitHub Repo Creation

Create GitHub Repo

- Open your browser and login to your GitHub Account
- Create a new public repository on GitHub

Either upload the project files to the repo directly from browser or push them from Git CLI.



Find the file contents below.

Dockerfile

```
# Base image
FROM python:3.11-slim
# Set working directory
WORKDIR /app
# Copy files
COPY . .
# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt
# Expose port
EXPOSE 5000
# Run the Flask app
CMD ["python","app.py"]
```

Jenkinsfile

```
pipeline {
    agent any
    environment {
        IMAGE_NAME = 'my-flask-app'
        CONTAINER_NAME = 'flask-app-container'
    }
    stages {
        stage('Build Docker Image') {
            steps {
                sh "docker build -t ${IMAGE_NAME} ."
            }
        }
        stage('Run Docker Container') {
            steps {
                // Stop old container if running
                sh "docker rm -f ${CONTAINER_NAME} || true"
                // Run new container
                sh "docker run -d --name ${CONTAINER_NAME} -p 5000:5000 ${IMAGE_NAME}"
            }
        }
    }
}
```

app.py

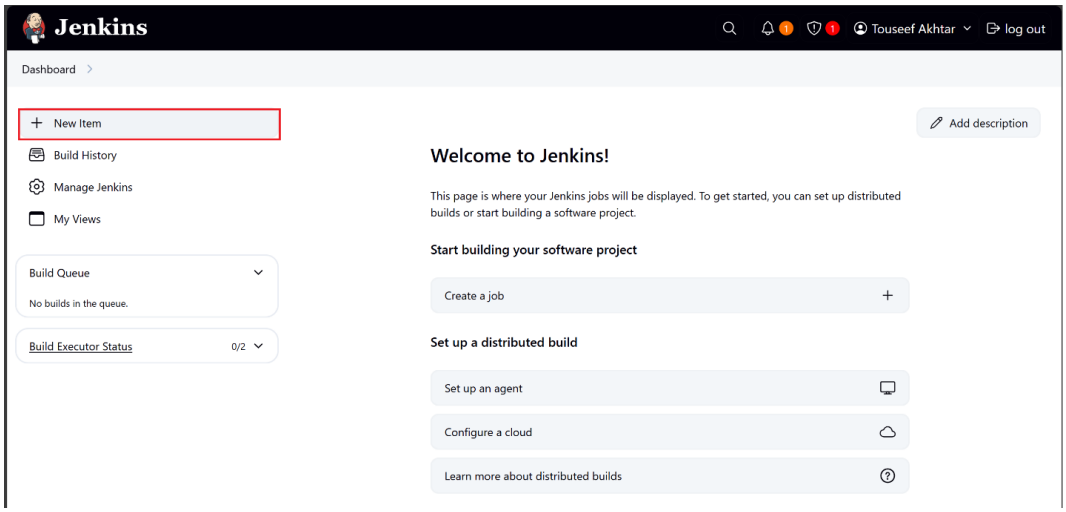
```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello from Flask deployed by Jenkins!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

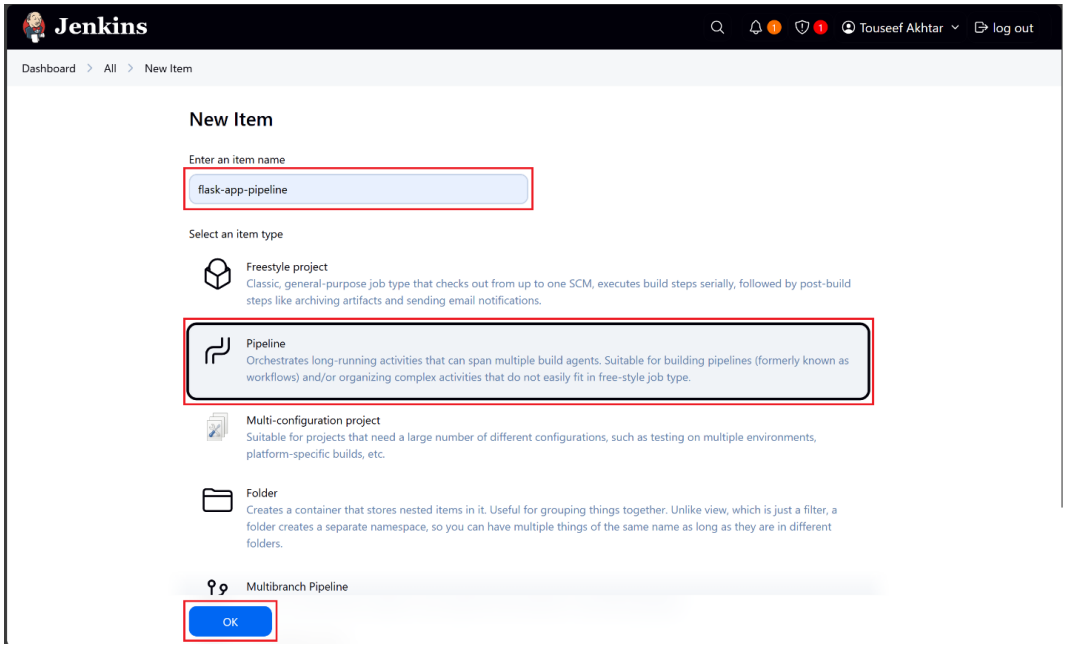
Requirements.txt

```
flask
```

3) Jenkins Pipeline Creation

After setting up our GitHub repo. Now let's create the Jenkins CI/CD pipeline. Follow the steps illustrated below.





Dashboard > flask-app-pipeline > Configuration

Configure

- General
- Triggers
- Pipeline
- Advanced

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/touseefdotdev/jenkins-deployment/

Credentials ?

- none -

+ Add

Advanced

Dashboard > flask-app-pipeline > Configuration

Configure

- General
- Triggers
- Pipeline
- Advanced

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

Script Path ?

Jenkinsfile

Lightweight checkout ?

Pipeline Syntax

Dashboard > flask-app-pipeline > Configuration

Configure

- General
- Triggers
- Pipeline
- Advanced

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

Script Path ?

Jenkinsfile

Lightweight checkout ?

Pipeline Syntax

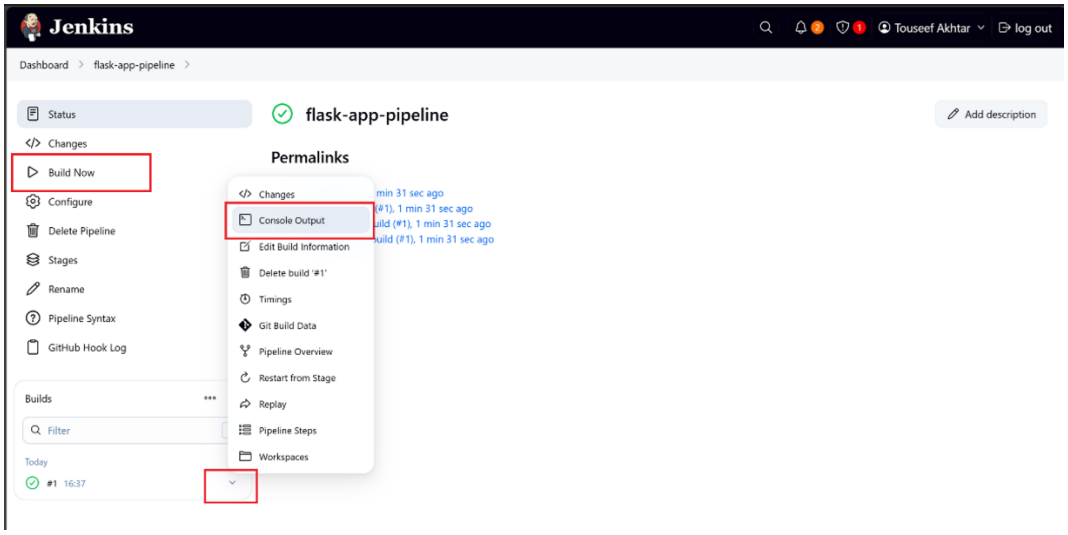
Advanced

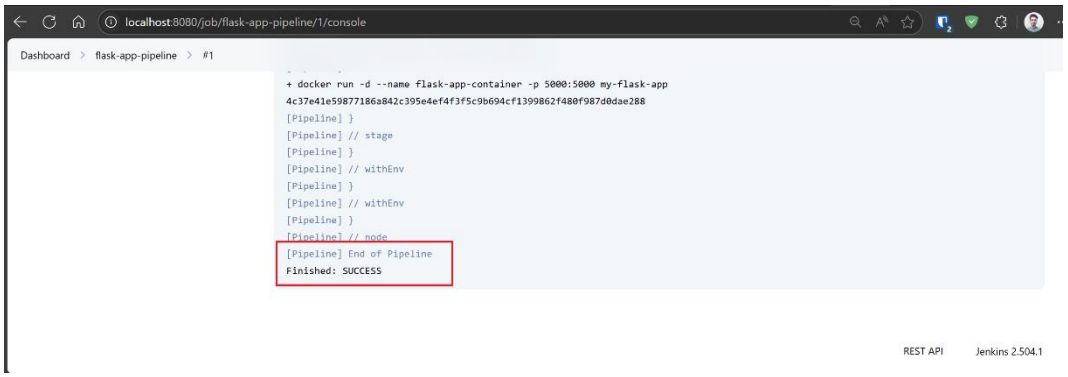
Advanced

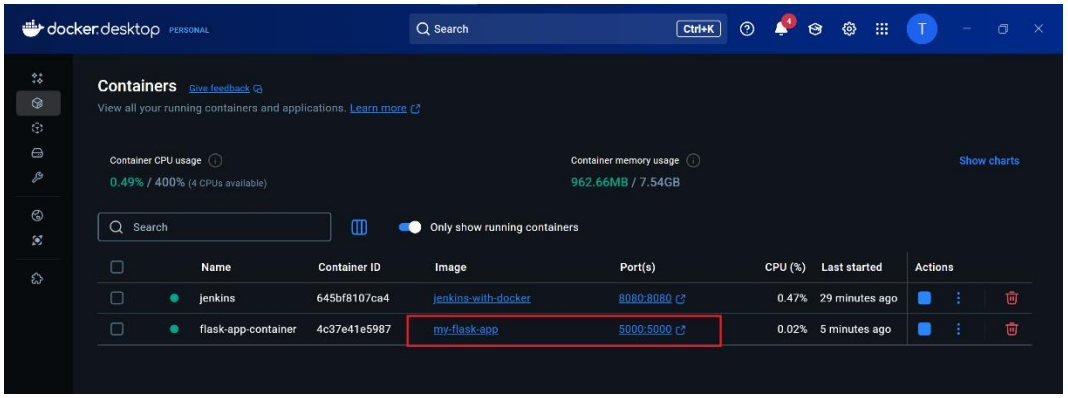
Save Apply

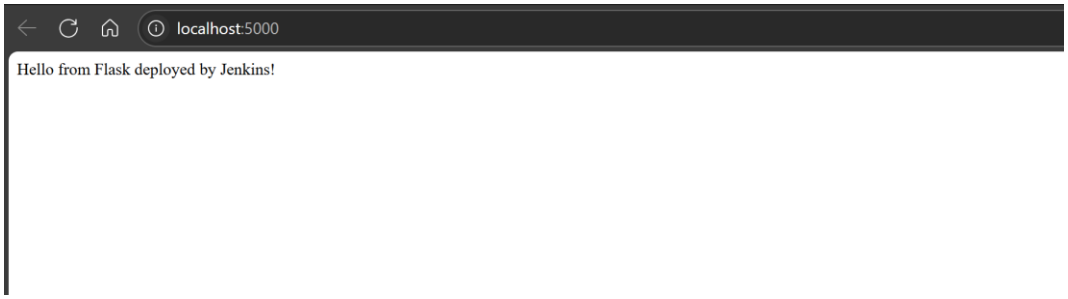
4) Code Deployment (Manual Trigger)

Let's build our pipeline manually and to View the logs of the pipeline build; do as follows;









Our code was successfully deployed by Jenkins pipeline.

5) Auto Trigger using GitHub Webhook (optional)

In order to implement Auto Trigger using Git Hub webhook; we need to configure following things;

a) Expose Local Running Jenkins to the Internet (so that GitHub can trigger it using Webhook)
b) Configure GitHub Webhook for our app code repo
c) Update our Jenkins pipeline to accept GitHub hook trigger

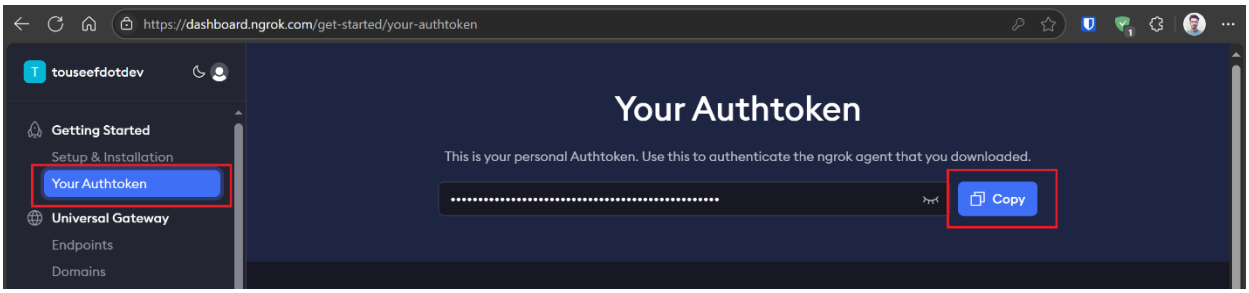
a) Expose Local Running Jenkins to the Internet (so that GitHub can trigger it using Webhook)

One way to achieve this is using a tool called `ngrok`.

To install ngrok; use this chocolatey command (for windows) in your command prompt;

```
choco install ngrok -y
```

Then go to https://dashboard.ngrok.com/signup and create a new account. After logging in to your account, copy your auth token;



Then run this command in command prompt;

```
ngrok authtoken <YOUR_AUTH_TOKEN>
```

```
ngrok http 8080
```

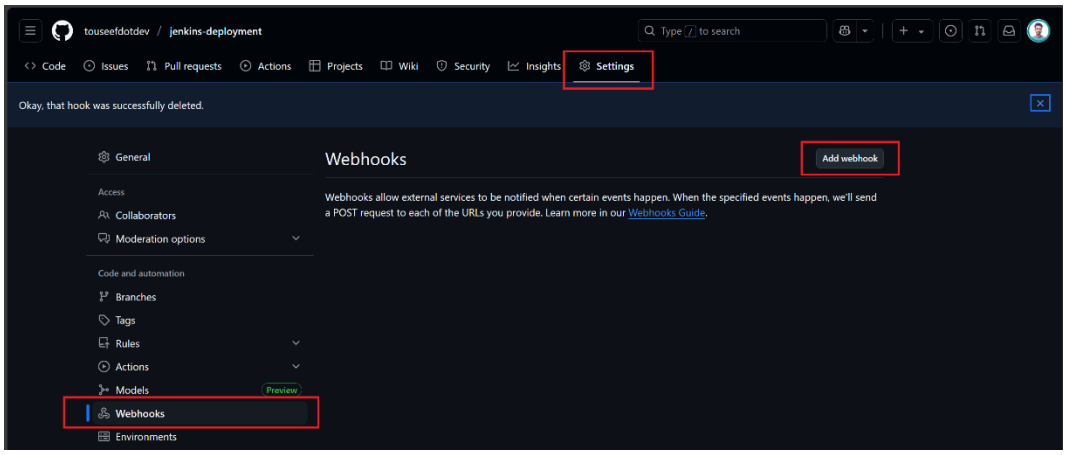
Then you are good to go. It would give you publicly available URL that is connected to your local Jenkins.

```
ngrok                                                                    (Ctrl+C to quit)

Load balance anything, anywhere with Endpoint Pools! https://ngrok.com/r/pools

Session Status                online
Account                       touseefdotdev (Plan: Free)
Version                       3.22.1
Region                        Europe (eu)
Latency                       111ms
Latency                       396ms
Latency                       170ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://87a0-39-61-57-10.ngrok-free.app -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              40      0       0.00    0.00    7.02    31.85

HTTP Requests
-------------

21:47:13.290 PKT POST /github-webhook/         200 OK
16:53:02.106 PKT POST /github-webhook/         200 OK
```

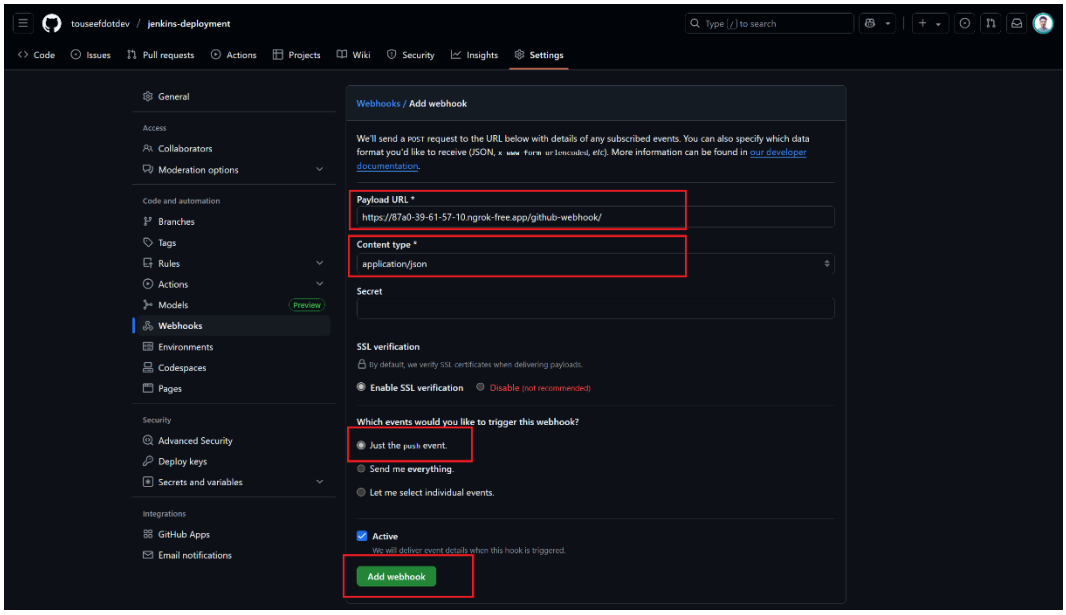
Copy that URL and use it to add webhook on the GitHub.

b) Configure GitHub Webhook for our app code repo

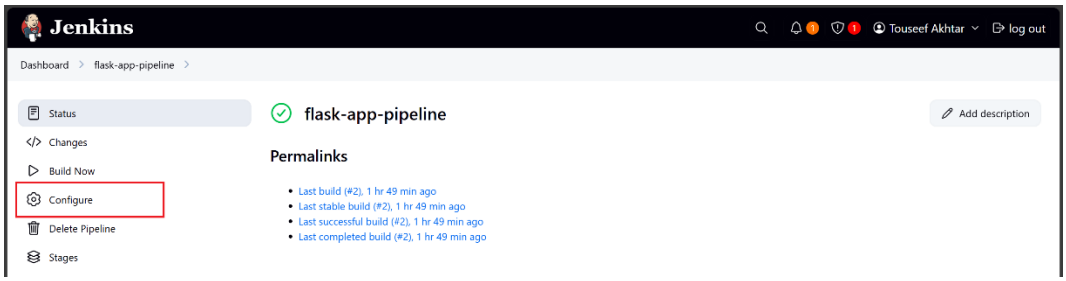
Go to your newly created repo > Settings > Webhooks > Add Webhook



GitHub Webhook Payload URL ➔ `<url-created-by-ngrok>/github-webhook/`



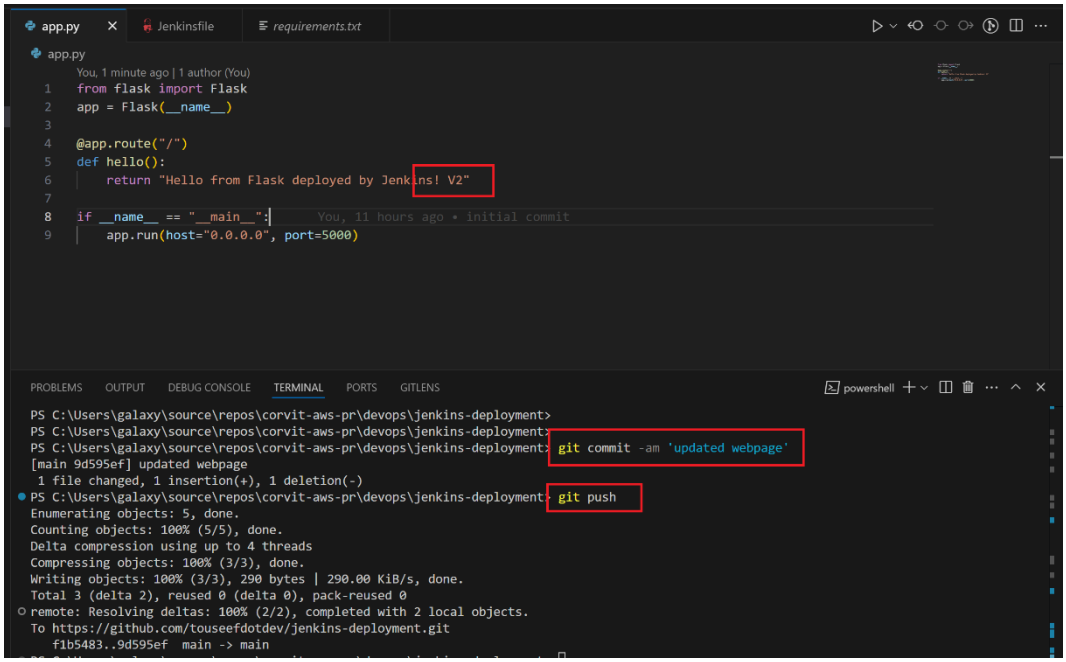
c) Update our Jenkins pipeline to accept GitHub hook trigger

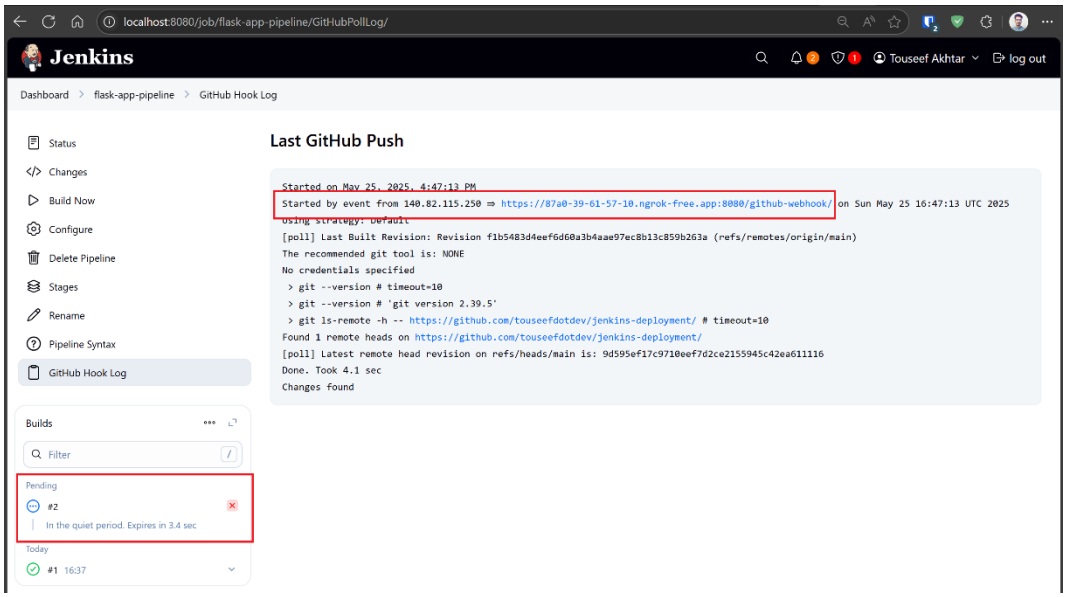


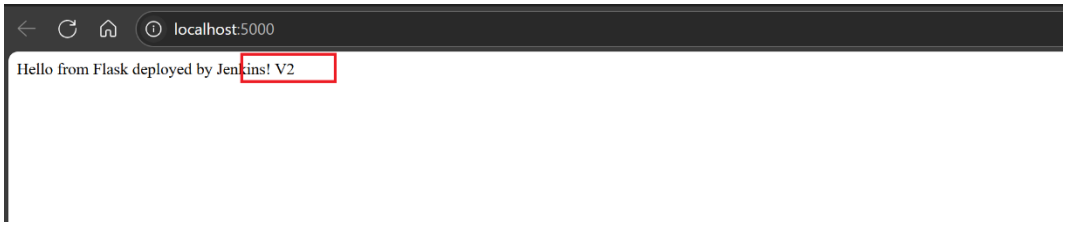
Apply and save!

Result:

After configuring the GitHub webhook trigger, if we update our code and push to GitHub main branch. We see that the Jenkins pipeline is triggered automatically, and our updated code is deployed.







Conclusion:

In this lab, we learned;

- How to install and setup the Jenkins
- How to build CI/CD pipelines on Jenkins using Docker
- How to setup GitHub Webhook and connect it with local Jenkins Instance