sample github jenkins docker example with a dockerfile

Below is a **simple, clean example** showing how to build and run a Dockerized app via **GitHub + Jenkins + Docker**, including:

- Directory structure
- `Dockerfile`
- Sample app (`app.py`)
- `Jenkinsfile` (pipeline that builds, tests, and pushes the image)

✅ Project Structure

```perl
my-jenkins-docker-example/
├── app.py
├── requirements.txt
├── Dockerfile
└── Jenkinsfile
```

🐳 Dockerfile (Python example)

```sql
FROM python:3.11-slim

WORKDIR /app

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

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

🐍 Simple Python App (`app.py`)

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

@app.route('/')
def home():
    return "Hello from Docker + Jenkins!"

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

📦 requirements.txt

```nginx
flask
```

🧪 Jenkinsfile (Declarative Pipeline)

This pipeline:

1. Pulls repo from GitHub
2. Builds Docker image
3. Runs container tests
4. Pushes image to Docker Hub (optional—enable if needed)

```typescript
pipeline {
    agent any

    environment {
        DOCKERHUB_USER = credentials('dockerhub-username')  // Jenkins credential ID
        IMAGE_NAME = "mydockeruser/jenkins-demo"
    }

    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-user/your-repo.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    dockerImage = docker.build("${IMAGE_NAME}:${BUILD_NUMBER}")
                }
            }
        }

        stage('Run Container for Tests') {
            steps {
                script {
                    dockerImage.run('-d -p 5000:5000')
                }
            }
        }

        stage('Push to Docker Hub') {
            when {
                branch 'main'
            }
            steps {
                script {
                    docker.withRegistry('', 'dockerhub-username') {
                        dockerImage.push("${BUILD_NUMBER}")
                        dockerImage.push("latest")
                    }
                }
            }
        }
    }

    post {
        always {
            sh 'docker ps -q --filter ancestor=$IMAGE_NAME | xargs -r docker stop'
        }
    }
}
```

🚀 How to Use

1. Push repo to GitHub
2. Configure Jenkins job with:
   - GitHub repo URL
   - Jenkins Docker plugin installed
   - DockerHub credentials stored as `dockerhub-username`
3. Run the Jenkins pipeline

🚀 Jenkins will build → test → push your Docker image automatically.

If you'd like, I can also generate:
✅ A GitHub Actions version
✅ Kubernetes deployment + Jenkins CI
✅ A multi-stage Dockerfile
Just tell me!

what is BUILD_NUMBER in this

`BUILD_NUMBER` is a **built-in Jenkins environment variable** that automatically increments with every pipeline run.

✔ What it represents

`BUILD_NUMBER` = the numeric ID of the current Jenkins job run.

Examples:

- First run → `BUILD_NUMBER = 1`
- Next run → `BUILD_NUMBER = 2`
- And so on…

✔ Why it's used in the Docker image name

In the Jenkinsfile:

```groovy
dockerImage = docker.build("${IMAGE_NAME}:${BUILD_NUMBER}")
```

This means:

- Each time Jenkins runs the pipeline, it produces a **unique Docker image tag**, like:
  - `mydockeruser/jenkins-demo:1`
  - `mydockeruser/jenkins-demo:2`
  - `mydockeruser/jenkins-demo:3`

This prevents overwriting previous images and makes builds traceable.

✔ Where it comes from

Jenkins automatically provides it—no configuration needed.

If you'd like, I can show you **all useful Jenkins default environment variables** or demonstrate how to create custom ones.