



Streamlining Application Deployment with Jenkins and Docker: CI/CD Pipeline Implementation

In today's fast-paced software development environment, organizations are increasingly relying on automated solutions to streamline the deployment process. DevOps practices, which emphasize collaboration and automation between development and operations teams, are essential for enabling continuous software delivery. This project focuses on the design and implementation of a Continuous Integration (CI) and Continuous Deployment (CD) pipeline using Jenkins and Docker to automate the software development lifecycle.

The CI/CD pipeline automates key processes such as code integration, testing, and deployment, allowing for faster feedback and more reliable deployments. By using Jenkins, an open-source automation tool, the project ensures that all code changes are automatically fetched, built, tested, and deployed. Docker plays a crucial role in containerizing the application, ensuring consistency and scalability across different environments, from development to production.

The goal of this project is to provide a fully automated, scalable, and efficient pipeline that eliminates manual intervention in the deployment process. The integration of Jenkins and Docker facilitates a smooth, error-free transition of code from development through testing and to production, ensuring that all code changes are continuously deployed in an optimized environment.

Throughout this project, I have designed the pipeline, configured Jenkins with necessary plugins, and set up Docker containers to facilitate smooth deployment workflows. This setup not only improves productivity by reducing manual steps but also enhances the overall quality of software releases by automating error detection and resolution.



Checking Node Setup and test file work or not?

```
PS E:\Youtube Projects\devopstest2\GitHub-Docker-and-Jenkins-CI-CD-Pipeline\nodeapp> npm start

> node-app@0.0.1 start
> node index.js

App listening on port 3000!
^C^CTerminate batch job (Y/N)? y
PS E:\Youtube Projects\devopstest2\GitHub-Docker-and-Jenkins-CI-CD-Pipeline\nodeapp> cd test
PS E:\Youtube Projects\devopstest2\GitHub-Docker-and-Jenkins-CI-CD-Pipeline\nodeapp\test> npm test

> node-app@0.0.1 test
> mocha ./test/test.js

App listening on port 3000!

GET /will
  ✓ respond with hello world

1 passing (40ms)
```

Create a docker image

```
> mocha ./test/test.js
PS E:\Youtube Projects\devopstest2\GitHub-Docker-and-Jenkins-CI-CD-Pipeline> docker build -t my-node-app2 .
[+] Building 157.9s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 609B
=> [internal] load metadata for docker.io/library/node:latest
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/node:latest@sha256:1745a99b66da41b5ccd6f7be3810f74ddab16eb4579de10de378adb50d2e6e6f
=> => resolve docker.io/library/node:latest@sha256:1745a99b66da41b5ccd6f7be3810f74ddab16eb4579de10de378adb50d2e6e6f
=> => sha256:1745a99b66da41b5ccd6f7be3810f74ddab16eb4579de10de378adb50d2e6e6f 6.41kB / 6.41kB
=> => sha256:b536d17a0f8a33c85b1ad262f50d3ffd82446c46c2240ba7657b9e66f3817d8d 6.39kB / 6.39kB
=> => sha256:54c7be425079efba0003054ee884bf72f1ffebca733bedd6f077d2809ee9aa6f 23.87MB / 23.87MB
=> => sha256:7aa8176e6d893aff4b57b2c22574ec2afadff4673b8e0954e275244bfd3d7bc1 64.39MB / 64.39MB
=> => sha256:8b2b79dc4164985acb914c82222c2a1e8277399a2b02c681109a04327baaf5b 2.49kB / 2.49kB
=> => sha256:0a96bdb8280554b560ffee0f2e5f9843dc7b625f28192021ee103ecbcc2d629b 48.50MB / 48.50MB
=> => sha256:1523f4b3f5602bf41caf8d8e649536ac0b3e56984c81a9f518fb20c6516ca075 211.31MB / 211.31MB
```



THISARA KANDAGE

UNDERGRADUATE - SLIIT

[E-mail](#) [LinkedIn](#) [GitHub](#) [Website](#)

The screenshot shows the Docker Desktop interface. On the left is a sidebar with navigation options: Containers, Images (selected), Volumes, Builds, Docker Scout, and Extensions. The main area is titled 'Images' and has tabs for 'Local' and 'Hub'. Below the tabs, it shows '942.72 MB / 3.46 GB in use' and '6 images'. A search bar is present. Below that is a table of local images. A red arrow points to the 'mysql' image in the table.

<input type="checkbox"/>	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	myapp	latest	1b674a4dc590	17 days ago	209.82 MB	▶ ⋮ 🗑
<input type="checkbox"/>	springboot-docker-app-optimized	latest	5baa779dbd78	27 days ago	351.38 MB	▶ ⋮ 🗑
<input type="checkbox"/>	thisarakandage/springboot-docker-app	latest	5baa779dbd78	27 days ago	351.38 MB	▶ ⋮ 🗑
<input type="checkbox"/>	docker-app	latest	1cbd2a1bff8e	27 days ago	351.38 MB	▶ ⋮ 🗑
<input type="checkbox"/>	thisarakandage/docker-app	latest	1cbd2a1bff8e	27 days ago	351.38 MB	▶ ⋮ 🗑
<input type="checkbox"/>	thisarakandage/my-node-app	latest	5f56979b32a2	1 month ago	1.14 GB	▶ ⋮ 🗑
<input type="checkbox"/>	mysql	8.0	6c55dbef969	3 months ago	591.33 MB	▶ ⋮ 🗑
<input type="checkbox"/>	my-node-app2	latest	7189e5a87614	2 minutes ago	1.14 GB	▶ ⋮ 🗑

Docker file

```
JS test.js x Dockerfile x
GitHub-Docker-and-Jenkins-CI-CD-Pipeline > Dockerfile > ...
1 # Use the latest version of the Node.js image as the base image
2 FROM node:latest
3
4 # Set the working directory inside the container to /usr/src/app
5 WORKDIR /usr/src/app
6
7 # Copy the contents of the local "nodeapp" directory to the root directory of the container
8 COPY nodeapp/* /
9
10 # Run the npm install command to install the dependencies specified in package.json
11 RUN npm install
12
13 # Expose port 3000 to allow incoming connections to the container
14 EXPOSE 3000
15
16 # Start the application by running the "npm start" command
17 CMD [ "npm", "start" ]
18
```



Create docker container using docker image

```
PS E:\Youtube Projects\devopstest2\GitHub-Docker-and-Jenkins-CI-CD-Pipeline> docker run -p 3000:3000 my-node-app2

> node-app@0.0.1 start
> node index.js

App listening on port 3000!
```

Docker Hub

The screenshot shows the Docker Desktop interface. On the left is a sidebar with navigation options: Containers, Images, Volumes, Builds, Docker Scout, and Extensions. The main area is titled 'Containers' and shows a table of running containers. Above the table, it displays 'Container CPU usage: 0.00% / 800% (8 CPUs available)' and 'Container memory usage: 37.09MB / 5.54GB'. A search bar and a toggle for 'Only show running containers' are also present. The table lists four containers, with 'compassionate_booth' highlighted by a red arrow.

	Name	Container ID ↑	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	reverent_lewin	0aa081185205	docker-app	8080:8080	0%	27 days ago	▶ ⋮ 🗑
<input checked="" type="checkbox"/>	compassionate_booth	d077926b4095	my-node-app2	3000:3000	0%	10 hours ago	▶ ⋮ 🗑
<input type="checkbox"/>	lucid_stonebraker	dd18733a0dd0	docker-app	8080:8080	0%	27 days ago	▶ ⋮ 🗑
<input type="checkbox"/>	docker	-	-	-	0%	27 days ago	▶ ⋮ 🗑

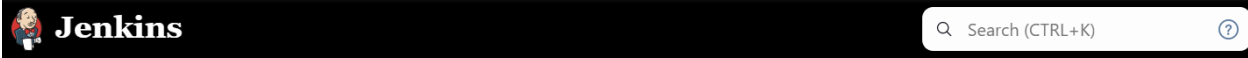


THISARA KANDAGE

UNDERGRADUATE - SLIIT

[E-mail](#) [LinkedIn](#) [GitHub](#) [Website](#)

Create Jenkins new item



[Dashboard](#) > [All](#) > [New Item](#)

New Item

Enter an item name

nodejs-test3

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Generate pipeline script

[Dashboard](#) > [nodejs-test3](#) > [Pipeline Syntax](#)

Secret text ?

Variable ?

test-dockerhubpass

Credentials ?

test-dockerhubpassword

+ Add

Add

Generate Pipeline Script

```
withCredentials([[string(credentialsId: 'test-dockerhubpassword', variable: 'test-dockerhubpass')]] {  
  // some block  
})
```





Jenkins file/Pipeline Script

```
pipeline {
    agent any

    stages {
        stage('SCM Checkout') {
            steps {
                retry(3) {
                    // Pull the latest code from the GitHub repository
                    git branch: 'main', url: 'https://github.com/Thisarak943/DockerJenkinsProject.git'
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                // List the directory contents to verify the Dockerfile is present
                bat 'dir'

                // Use the correct format for Windows environment variables
                bat '''
                    echo Building Docker image...

                    docker build -t thisarakandage/nodeapp2:%BUILD_NUMBER% .
                '''
            }
        }
    }
}
```



```
stage('Login to Docker Hub') {  
    steps {  
        withCredentials([string(credentialsId: 'test-dockerhubpassword', variable: 'test-  
dockerhubpass')]) {  
            // Login to Docker Hub using Jenkins credentials  
            bat 'docker login -u thisarakandage -p %test-dockerhubpass%'  
        }  
    }  
}  
  
stage('Push Image') {  
    steps {  
        // Push the image to Docker Hub with a unique tag based on build number  
        bat 'docker push thisarakandage/nodeapp2:%BUILD_NUMBER%'  
    }  
}  
  
post {  
    always {  
        // Logout from Docker Hub to ensure security  
        bat 'docker logout'  
    }  
}
```



Conclusion

1. Challenges Faced:

- Configuring Jenkins and Docker integration.
- Resolving misconfigurations in pipeline scripts.
- Handling failed authentication during Docker Hub push.

2. Solutions Implemented:

- Set up Jenkins and Docker integration, following best practices.
- Used retry mechanisms and corrected pipeline scripts.
- Configured secure authentication for Docker Hub login.

3. Key Learnings:

- Gained practical experience with Jenkins, Docker, and CI/CD.
- Improved troubleshooting skills in build and deployment stages.
- Learned containerization best practices for consistent deployments.

4. Impact on Knowledge:

- Enhanced proficiency in Jenkins and Docker.
- Strengthened problem-solving abilities in DevOps workflows.
- Acquired hands-on experience in automating deployment processes.