JENKINS: END-TO-END PIPELINE AS A CODE

In my organization, I received a project from a client. They need a webpage developed using HTML and CSS. I want to automate this using Jenkins.

We have 3 teams:

1. Developer Team
2. QA Team
3. Production Team

I'm planning to use Jenkins to automate the process. I'll set up the developers' systems as Jenkins slaves. When a developer writes new code, it will be pushed to GitHub. Jenkins will then download the code, create an environment, and launch it in a Docker container.

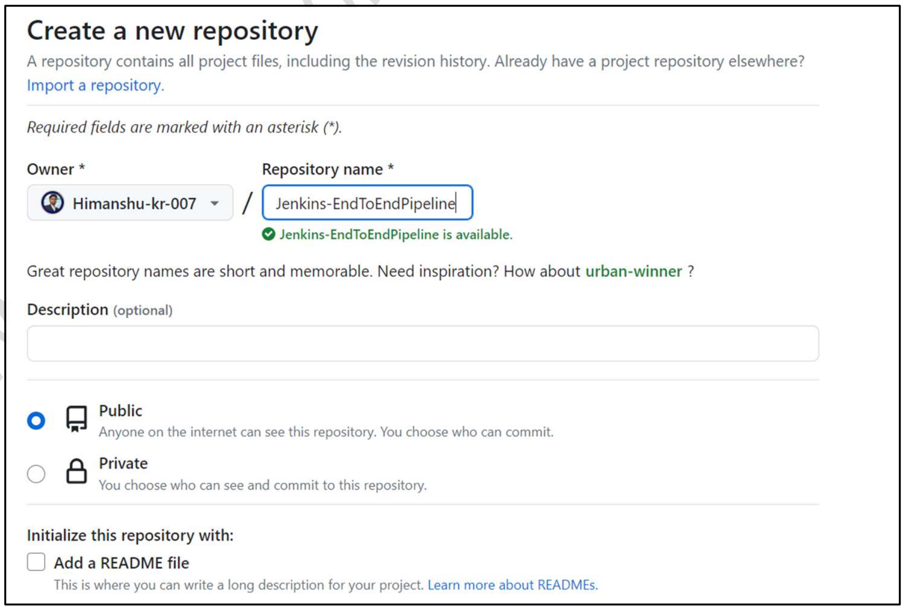
Here are the steps:

1. Download the webpage code locally.
2. Pull the "httpd" image from Docker Hub.
3. Copy the webpage files into the "Document Root" of the web server.
4. Expose port 80 for the web server and start it.
5. Push this code as a new Docker image to Docker Hub.

Next, the QA team will:

1. Download the image from Docker Hub.
2. Launch a container and test all the cases.
3. If tests pass, they can deploy it in the production environment.

I am going to create one repository for this in my GitHub where I am going to put the Dockerfile and the code for my webpages with the name of index.html.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * Himanshu-kr-007 / Repository name * Jenkins-EndToEndPipeline

Jenkins-EndToEndPipeline is available.

Great repository names are short and memorable. Need inspiration? How about urban-winner ?

Description (optional)

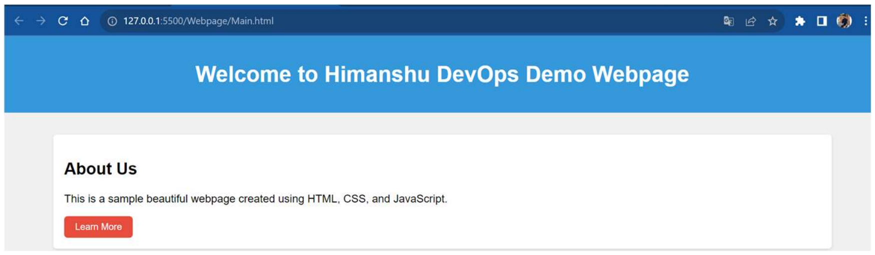
[✓ Public] — Anyone on the internet can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file — This is where you can write a long description for your project. Learn more about READMEs.

And I have the code for my webpages. The webpage is look like this.

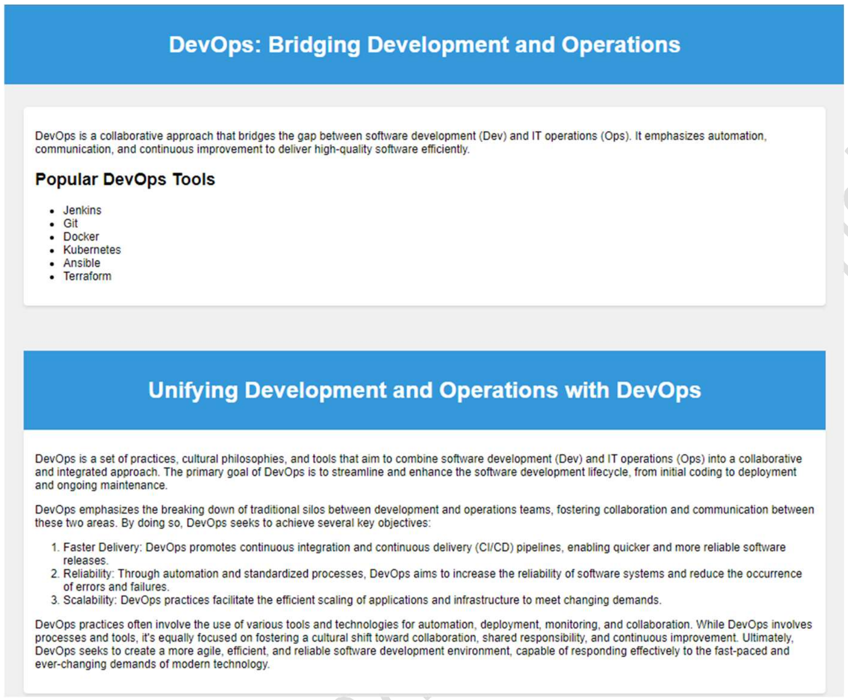


Welcome to Himanshu DevOps Demo Webpage

About Us

This is a sample beautiful webpage created using HTML, CSS, and JavaScript.

Version 1: Index.html



DevOps: Bridging Development and Operations

DevOps is a collaborative approach that bridges the gap between software development (Dev) and IT operations (Ops). It emphasizes automation, communication, and continuous improvement to deliver high-quality software efficiently.

Popular DevOps Tools

- Jenkins
- Git
- Docker
- Kubernetes
- Ansible
- Terraform

Unifying Development and Operations with DevOps

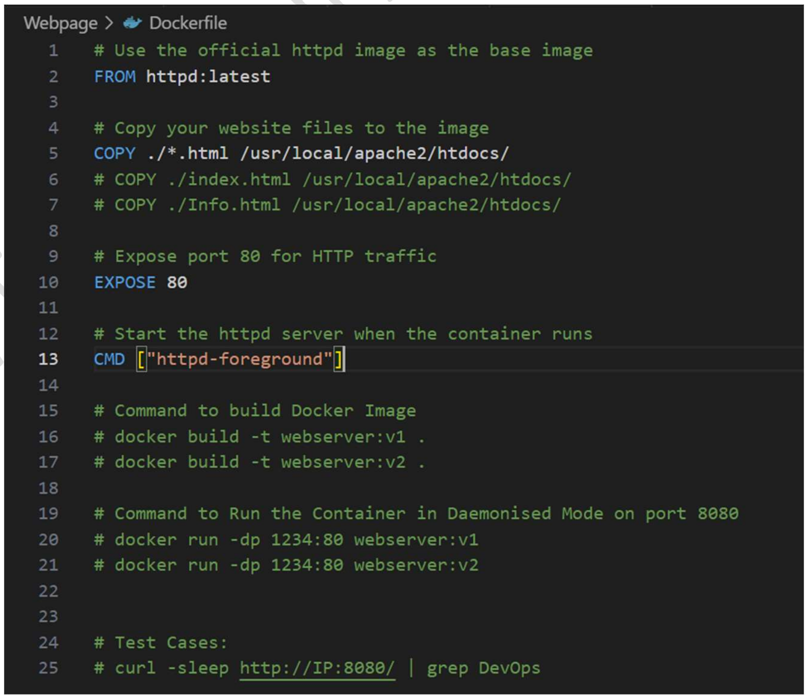
DevOps is a set of practices, cultural philosophies, and tools that aim to combine software development (Dev) and IT operations (Ops) into a collaborative and integrated approach. The primary goal of DevOps is to streamline and enhance the software development lifecycle, from initial coding to deployment and ongoing maintenance.

DevOps emphasizes the breaking down of traditional silos between development and operations teams, fostering collaboration and communication between these two areas. By doing so, DevOps seeks to achieve several key objectives:

1. Faster Delivery: DevOps promotes continuous integration and continuous delivery (CI/CD) pipelines, enabling quicker and more reliable software releases.
2. Reliability: Through automation and standardized processes, DevOps aims to increase the reliability of software systems and reduce the occurrence of errors and failures.
3. Scalability: DevOps practices facilitate the efficient scaling of applications and infrastructure to meet changing demands.

DevOps practices often involve the use of various tools and technologies for automation, deployment, monitoring, and collaboration. While DevOps involves processes and tools, it's equally focused on fostering a cultural shift toward collaboration, shared responsibility, and continuous improvement. Ultimately, DevOps seeks to create a more agile, efficient, and reliable software development environment, capable of responding effectively to the fast-paced and ever-changing demands of modern technology.

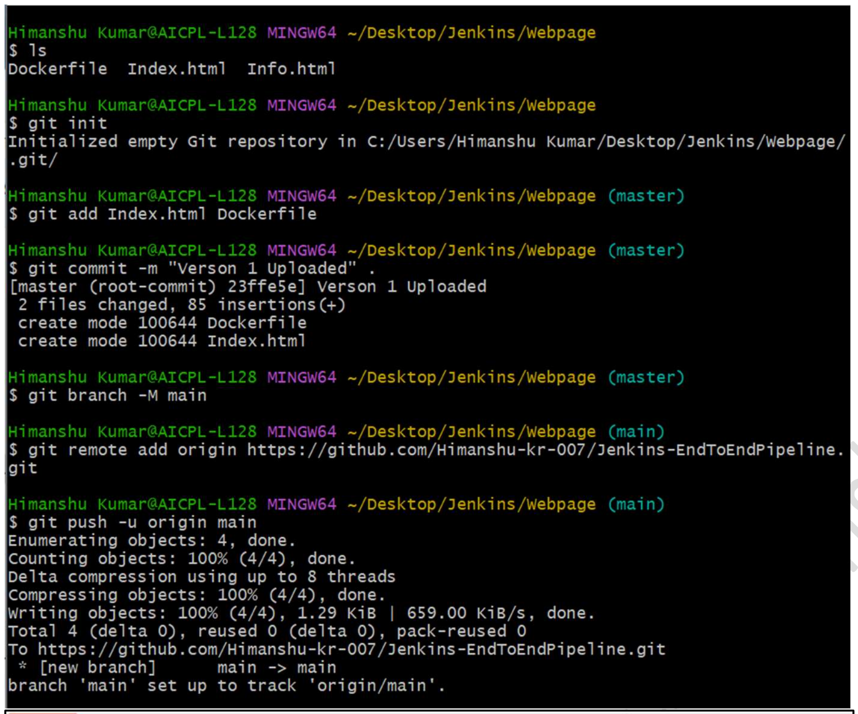
Version 2: Index.html



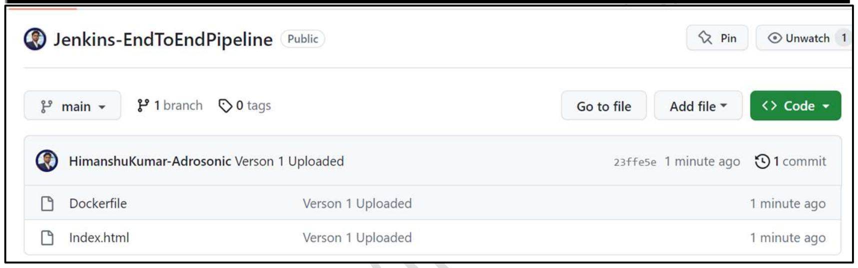
```dockerfile
# Use the official httpd image as the base image
FROM httpd:latest

# Copy your website files to the image
COPY ./*.html /usr/local/apache2/htdocs/
# COPY ./index.html /usr/local/apache2/htdocs/
# COPY ./Info.html /usr/local/apache2/htdocs/

# Expose port 80 for HTTP traffic
EXPOSE 80

# Start the httpd server when the container runs
CMD ["httpd-foreground"]

# Command to build Docker Image
# docker build -t webserver:v1 .
# docker build -t webserver:v2 .

# Command to Run the Container in Daemonised Mode on port 8080
# docker run -dp 1234:80 webserver:v1
# docker run -dp 1234:80 webserver:v2


# Test Cases:
# curl -sleep http://IP:8080/ | grep DevOps
```

Dockerfile

I am going to Upload the Code for Index.html: V1 and Dockerfile in my GitHub Repository.



```
Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/Webpage
$ ls
Dockerfile  Index.html  Info.html

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/Webpage
$ git init
Initialized empty Git repository in C:/Users/Himanshu Kumar/Desktop/Jenkins/Webpage/
.git/

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/Webpage (master)
$ git add Index.html Dockerfile

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/Webpage (master)
$ git commit -m "Verson 1 Uploaded" .
[master (root-commit) 23ffe5e] Verson 1 Uploaded
 2 files changed, 85 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 Index.html

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/Webpage (master)
$ git branch -M main

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/Webpage (main)
$ git remote add origin https://github.com/Himanshu-kr-007/Jenkins-EndToEndPipeline.
git

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/Webpage (main)
$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.29 KiB | 659.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Himanshu-kr-007/Jenkins-EndToEndPipeline.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```



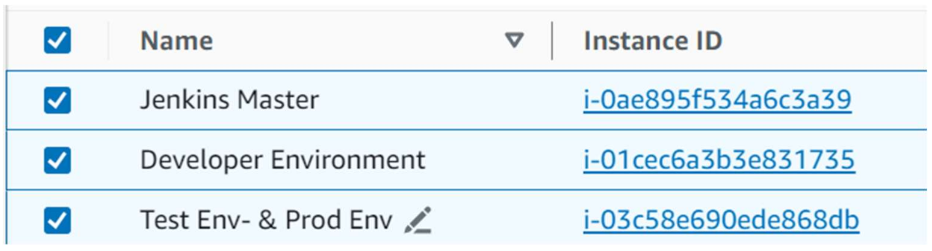
HimanshuKumar-Adrosonic Verson 1 Uploaded	23ffe5e 1 minute ago	1 commit
Dockerfile	Verson 1 Uploaded	1 minute ago
Index.html	Verson 1 Uploaded	1 minute ago

The code is Uploaded in the Centralised Repository. Now I am going to create the Pipeline by which I can perform Multiple Jobs.

Job A: Pull the Code from Git Hub.
Job B: Build the Dockerfile on top of Docker Engine.
Job C: Push the Image in the Docker Hub.
Job D: Deploy Webapp in Testing Environment.
Job E: Test the Code from Testing Environment.
Job F: If Testing Pass, Then Deploy Code in Production Environment.

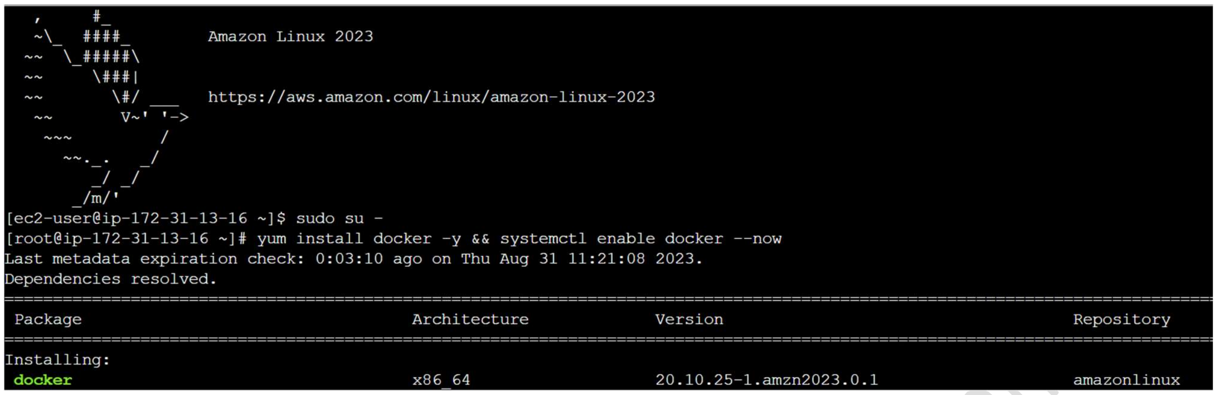
Let's Do this Thing Step by Step. For Creating the Pipeline, I am going to use the Pipeline Plugin and I am going to Use 3 OS.

- OS 1: Jenkins Master
- OS 2: Developer Environment (Slave Node)
- OS 3: Testing Environment & Production Environment



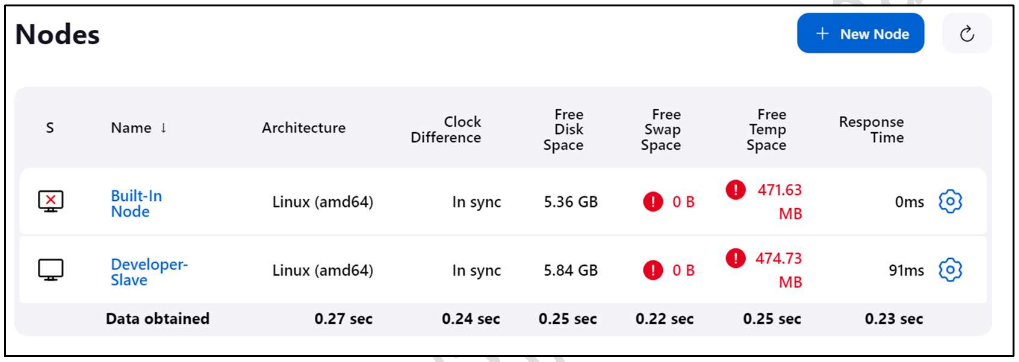
✓	Name	Instance ID
✓	Jenkins Master	i-0ae895f534a6c3a39
✓	Developer Environment	i-01cec6a3b3e831735
✓	Test Env- & Prod Env	i-03c58e690ede868db

NOTE: Install Docker and Enable Docker Engine in All System by using This Command.
Install Git & Java-11 in Master and Slave System.



```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-13-16 ~]$ sudo su -
[root@ip-172-31-13-16 ~]# yum install docker -y && systemctl enable docker --now
Last metadata expiration check: 0:03:10 ago on Thu Aug 31 11:21:08 2023.
Dependencies resolved.
================================================================================================================
 Package                                  Architecture             Version                           Repository
================================================================================================================
Installing:
 docker                                   x86_64                   20.10.25-1.amzn2023.0.1           amazonlinux
```

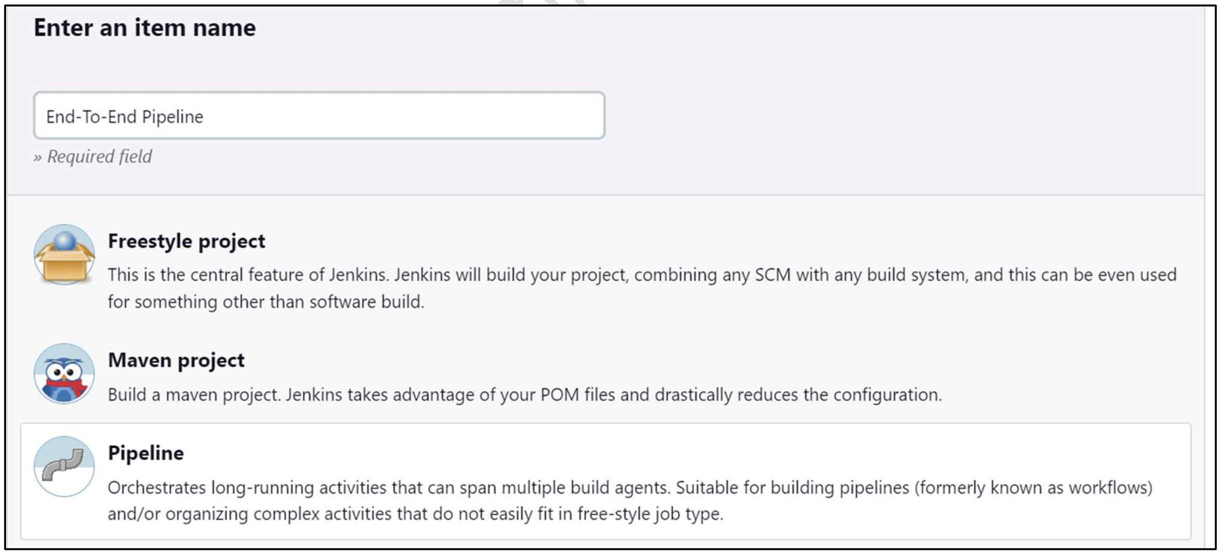
For setting Up Slave in Linux Refer to This Document: [Document Link]



Nodes

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	5.36 GB	0 B	471.63 MB	0ms
	Developer-Slave	Linux (amd64)	In sync	5.84 GB	0 B	474.73 MB	91ms
	Data obtained	0.27 sec	0.24 sec	0.25 sec	0.22 sec	0.25 sec	0.23 sec

I have Set the Slave System and Give the Name as Developer-Slave. After this I am going to create a new Job as a Pipeline.



Enter an item name

End-To-End Pipeline

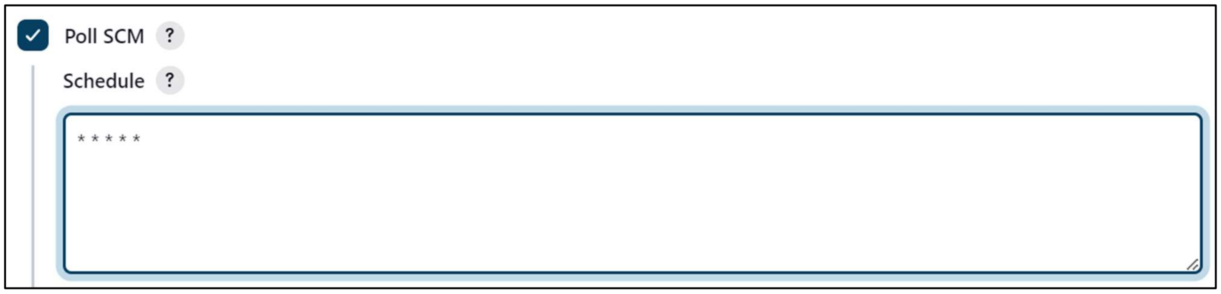
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

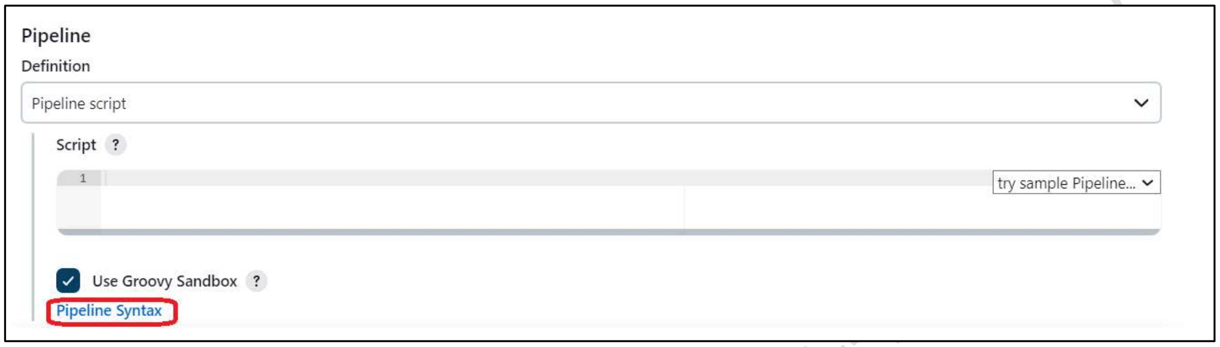
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Inside the Job, I want to pull this Project every time if any changes made. For this I enabled Poll SCM for every minute.

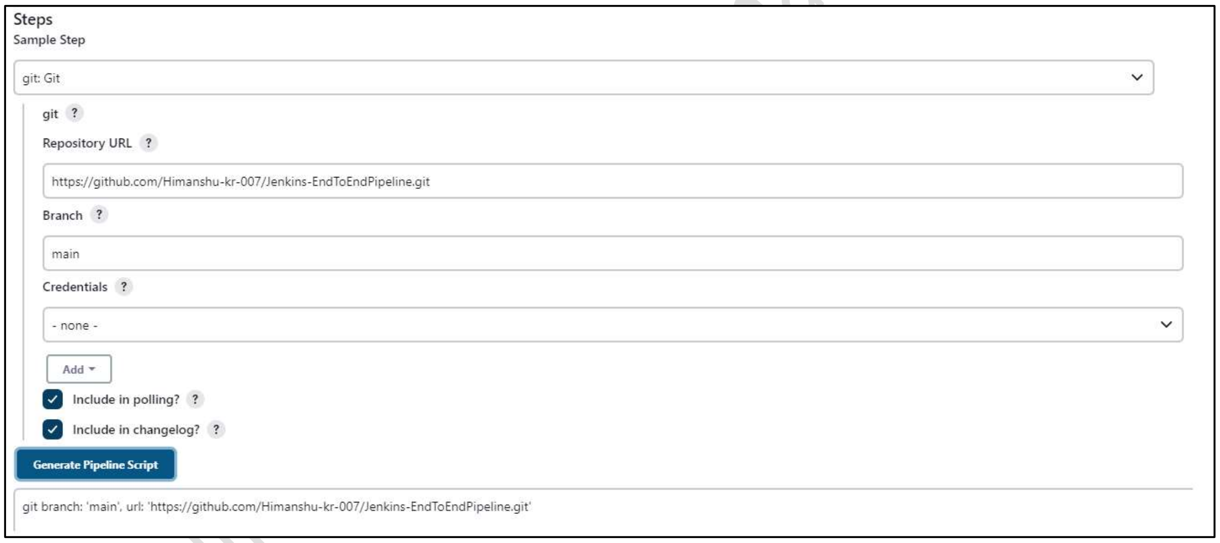


```
* * * * *
```

Go to Pipeline Section, there you find the link for **Pipeline Syntax** Click on that you will redirect to other page.



Based on the requirement select the Definition. Here, I want to get the Data from my GitHub from my main branch. After that, I have clicked on Generate Pipeline Script. It gives me the code for that.



git: Git

Repository URL: https://github.com/Himanshu-kr-007/Jenkins-EndToEndPipeline.git

Branch: main

Credentials: - none -

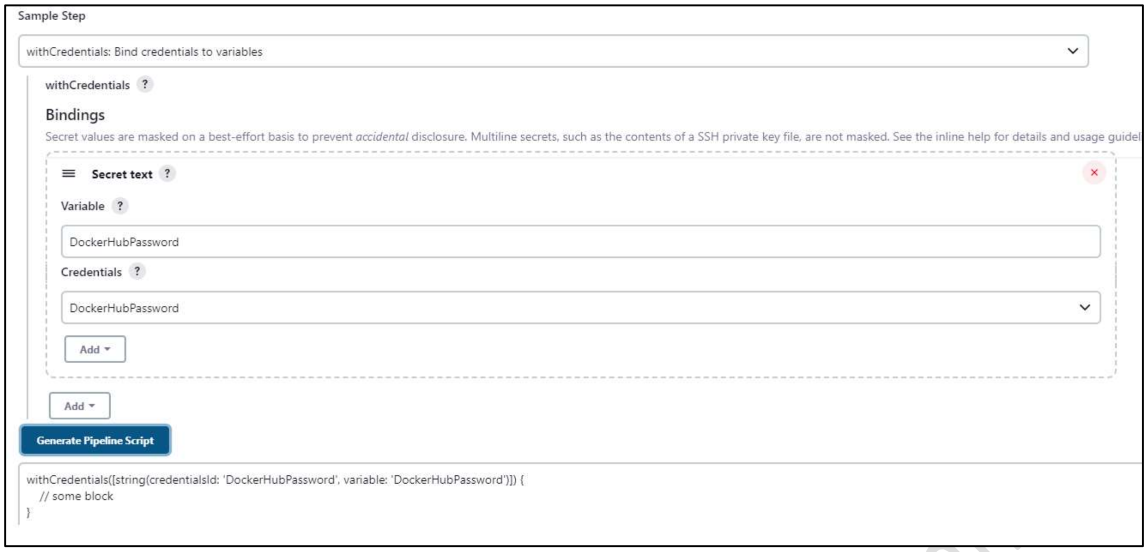
✓ Include in polling?

✓ Include in changelog?

```
git branch: 'main', url: 'https://github.com/Himanshu-kr-007/Jenkins-EndToEndPipeline.git'
```

I also want that when the Docker Image is build, then I want to Push the Image into my Docker Hub Account. For that I need to login inside My Docker Hub account. And I don't want to show my Docker Hub password for that. So, I am going to store my Docker Hub Password in a variable.

I select the Option: With Credentials: Bind Credentials to variables. Defining the Variable name and Binding the Credentials as **Secret Text.**


Sample Step

withCredentials: Bind credentials to variables

withCredentials

Bindings

Secret values are masked on a best-effort basis to prevent *accidental* disclosure. Multiline secrets, such as the contents of a SSH private key file, are not masked. See the inline help for details and usage guidel

Secret text

Variable: DockerHubPassword

Credentials: DockerHubPassword

```
withCredentials([string(credentialsId: 'DockerHubPassword', variable: 'DockerHubPassword')]) {
    // some block
}
```

Let’s see the Code:

```
pipeline {
    agent {
        label "DeveloperBuildNode"
    }
```

This specifies that the pipeline should run on a Jenkins agent labelled as "DeveloperBuildNode." Jenkins agents are the nodes that execute the tasks defined in the pipeline. In this case, it's specifying a specific agent to use.

The pipeline consists of several stages, each of which represents a distinct step in the overall process.

```
stages {
        stage('Pull Code From GitHub') {
            steps {
                git branch: 'main', url: 'https://github.com/Himanshu-kr-
007/Jenkins-EndToEndPipeline.git'
            }
        }
```

This stage is responsible for cloning the code from the specified GitHub repository. It checks out the 'main' branch from the provided URL.

```
        stage('Build Docker Image'){
            steps{
                sh 'sudo docker build -t himanshukr0612/webserver:${BUILD_TAG} .'
            }
        }
```

In this stage, a Docker image is built using the code retrieved from GitHub. The `docker build` command is used to create an image named "himanshukr0612/webserver" with a tag that includes the Jenkins BUILD_TAG variable.

```
        stage('Push Image Into Docker Hub'){
            steps{
                withCredentials([string(credentialsId: 'DockerHubPassword',
variable: 'DockerHubPassword')]) {
                    sh 'sudo docker login -u himanshukr0612 -p $DockerHubPassword'
                }
                sh 'sudo docker push himanshukr0612/webserver:${BUILD_TAG}'
            }
        }
```

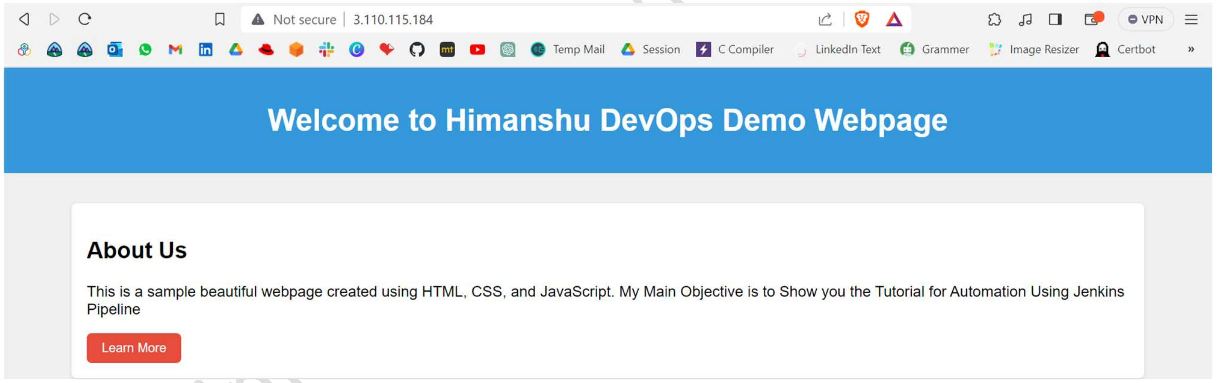
This stage is responsible for pushing the Docker image created in the previous stage to Docker Hub. It first logs in to Docker Hub using a stored credential (DockerHubPassword) and then uses the `docker push` command to upload the image.

```
        stage('Deploy Webapp In Dev Environment'){
            steps{
                sh 'sudo docker rm -f mywebserver'
                sh 'sudo docker run -d -p 80:80 --name mywebserver
himanshukr0612/webserver:${BUILD_TAG}'

            }
        }
```

In this stage, the previous Docker container with the name "mywebserver" is forcefully removed (`docker rm -f mywebserver`). Then, a new Docker container is launched in detached mode (`-d`) and mapped to port 80 on the host machine (`-p 80:80`). This container is named "mywebserver" and runs the Docker image built earlier.

Here jenkins, is login with the EC2-USER. And we can run the Docker command by EC2-USER using root power that's why I mentioned sudo.

Now the container is launched with the name of mywebserver and expose the Port 80. So, the real-world client can connect with the container.

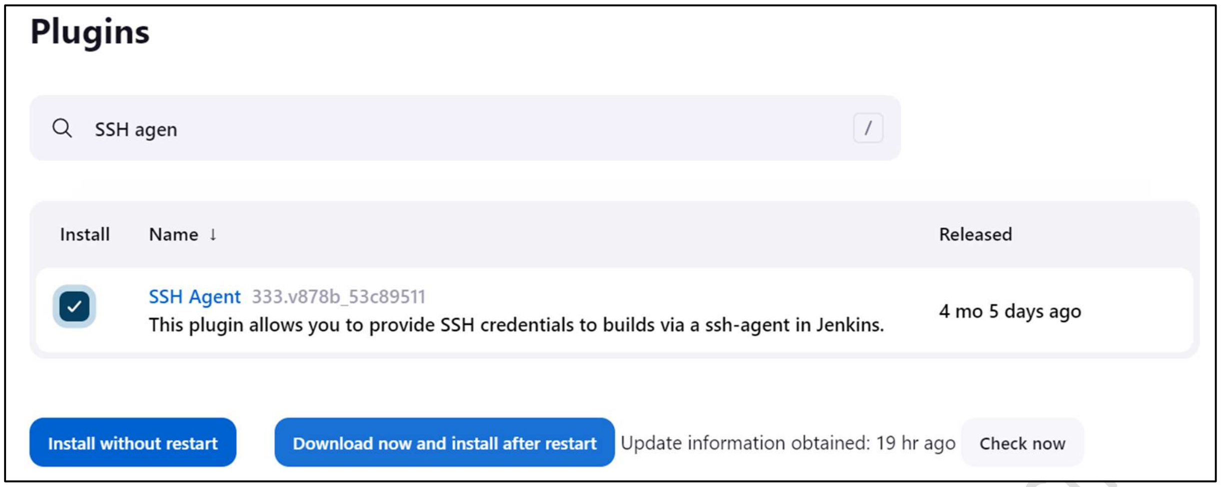


Welcome to Himanshu DevOps Demo Webpage

About Us

This is a sample beautiful webpage created using HTML, CSS, and JavaScript. My Main Objective is to Show you the Tutorial for Automation Using Jenkins Pipeline

After that I am going to run Some test case in this webpage in Testing Environment. And here I don't want to make Testing Environment as slave system for this I need to download a plugin called **SSH Agent**. Jenkins will login to Test system via SSH.



Plugins

SSH agen

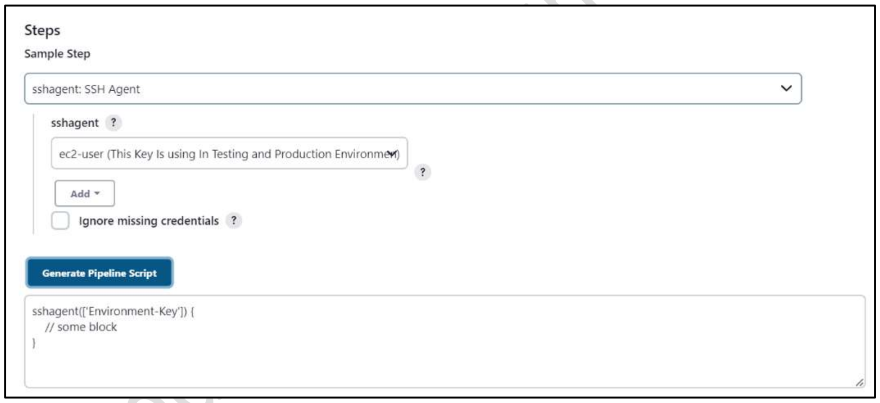
Install	Name ↓	Released
✓	SSH Agent 333.v878b_53c89511 This plugin allows you to provide SSH credentials to builds via a ssh-agent in Jenkins.	4 mo 5 days ago

Install without restart
Download now and install after restart
Update information obtained: 19 hr ago
Check now

After Installing this plugin, you need to restart the Jenkins. So that you can use this user. After that we need to create ec2-user Credentials for Testing and Production Environment. So, From the Steps → select **sshagent: SSH Agent**.

Here You are not getting any credentials for this so click on **Jenkins**. From Kind → Select **SSH USERNAME WITH PRIVATE KEY.**

Then mention ID, Username and paste the private key content in the text field and click on save button to generate the Credentials.



Steps

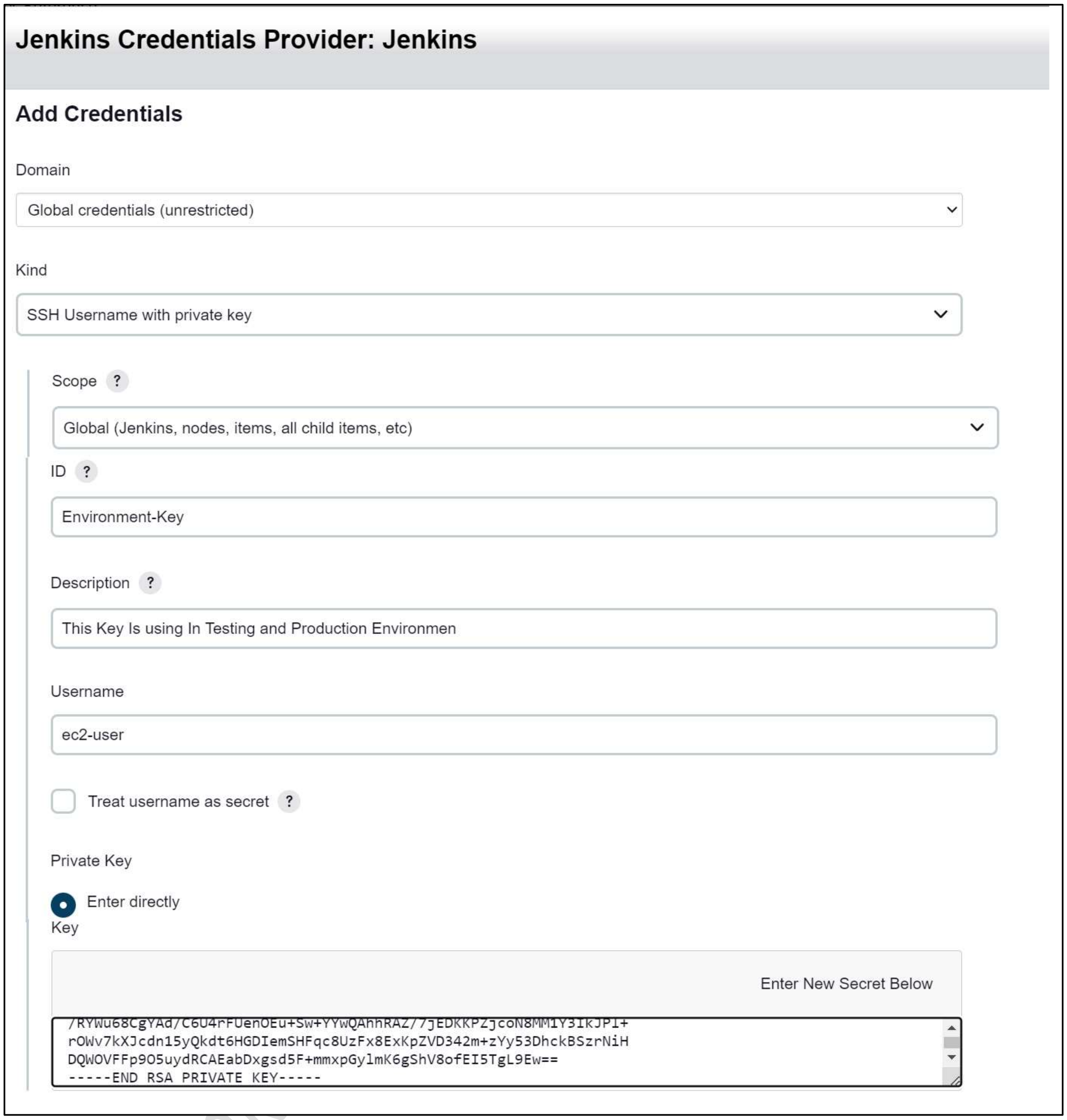
Sample Step: sshagent: SSH Agent

sshagent: ec2-user (This Key Is using In Testing and Production Environment)

Ignore missing credentials

```
sshagent(['Environment-Key']) {
    // some block
}
```

When we try to login via SSH using Jenkins, we need to create the SSH user which will go on behalf on jenkins to the Test Environment and Production Environment and launch the container after pulling the image form the docker hub. For this I have Create



Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: SSH Username with private key

Scope ?: Global (Jenkins, nodes, items, all child items, etc)

ID ?: Environment-Key

Description ?: This Key Is using In Testing and Production Environmen

Username: ec2-user

☐ Treat username as secret ?

Private Key

● Enter directly

Key

Enter New Secret Below

```
/RYWu68CgYAd/C6U4rFUenOEu+Sw+YYwQAhhRAZ/7jEDKKPZjcoN8MM1Y3IkJP1+
rOWv7kXJcdn15yQkdt6HGDIemSHFqc8UzFx8ExKpZVD342m+zYy53DhckBSzrNiH
DQWOVFFp9O5uydRCAEabDxgsd5F+mmxpGylmK6gShV8ofEI5TgL9Ew==
-----END RSA PRIVATE KEY-----
```

Now Let’s write the Job for the Login inside the Test Environment.

```
        stage('Launch Container In Testing Environment'){
            steps{
                sshagent(['Environment-Key']) {
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker rm -f mywebserver"
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker run -d -p 80:80 --name mywebserver himanshukr0612/webserver:${BUILD_TAG}"
                }
            }
        }
```

Jenkins will Go to the **Test Environment** as **ec2-user** using SSH agent and launch the container with the name of mywebserver after pulling the image from docker hub **himanshukr0612/webserver** with latest version. I have disabled the Host Key Checking so they don’t ask for yes and no from jenkins.

```
        stage('Testing Phase'){
            steps{
                sh 'curl -sleep http://13.126.224.70:80/ | grep DevOps'
                sh 'curl -sleep http://13.126.224.70:80/ | grep Jenkins'
            }
        }
```
Here I have mentioned 2 Simpe testing code for this website. If this webpage has the Content DevOps and Jenkins, then the test case will pass other wise test case will fail. And it leads to failure of pipeline.

```
        stage('Release To Production'){
            steps{
                input(message: "Release To Production?")
            }
        }
```
In simple terms, this stage is like a safety checkpoint before deploying to a production environment. It asks for manual confirmation from a user to ensure that the release to production is intentional and doesn't happen accidentally. Once the user provides the input, the pipeline will proceed accordingly based on the user's response (e.g., continue with the release or abort it).

```
        stage('Deploy In Prod'){
            steps{
                sshagent(['Environment-Key']) {
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker rm -f prod"
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker run -d -p 1234:80 --name prod himanshukr0612/webserver:${BUILD_TAG}"
                }
            }
        }
```
In simple terms, this Jenkins pipeline stage is responsible for deploying a Docker container with a web server to a production server. It ensures the deployment is secure by using SSH for authentication. The container is named "prod" and listens on port 1234 on the host machine, serving content from the Docker image specified by ${BUILD_TAG}. This stage helps in deploying new versions of the web server to the production environment.

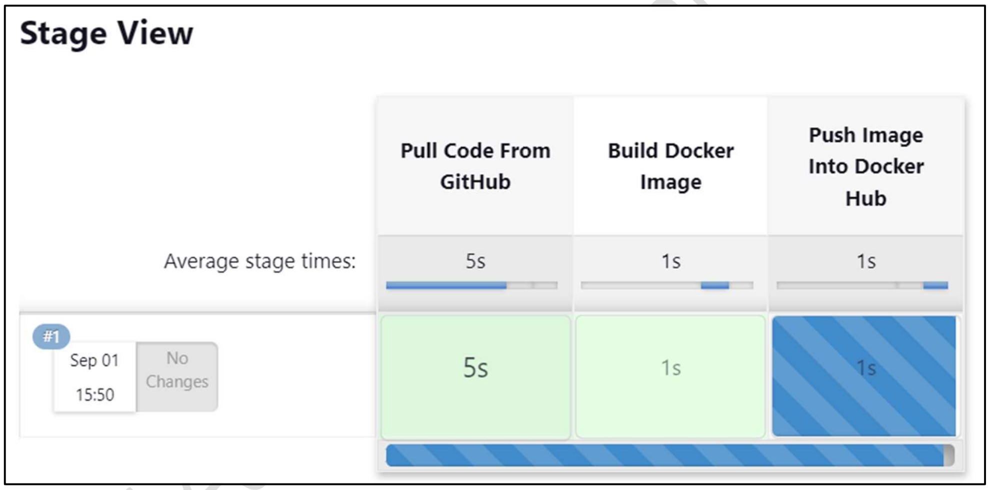
Environment Variable: **BUILD TAG →** This is used to manage the Version Control for Docker image which we push on docker hub. And if we don't use this then we need to manually change the Version in Image which causes the manual intervention of the user.

Now let's see the Entire Code

```
pipeline {
    agent {
        label "DeveloperBuildNode"
    }
    stages {
        stage('Pull Code From GitHub') {
            steps {
                git branch: 'main', url: 'https://github.com/Himanshu-kr-
007/Jenkins-EndToEndPipeline.git'
            }
        }
        stage('Build Docker Image'){
            steps{
                sh 'sudo docker build -t himanshukr0612/webserver:${BUILD_TAG} .'
            }
        }
        stage('Push Image Into Docker Hub'){
            steps{
                withCredentials([string(credentialsId: 'DockerHubPassword',
variable: 'DockerHubPassword')]) {
                    sh 'sudo docker login -u himanshukr0612 -p $DockerHubPassword'
                }
                sh 'sudo docker push himanshukr0612/webserver:${BUILD_TAG}'
            }
        }
        stage('Deploy Webapp In Dev Environment'){
            steps{
                sh 'sudo docker rm -f mywebserver'
                sh 'sudo docker run -d -p 80:80 --name mywebserver
himanshukr0612/webserver:${BUILD_TAG}'

            }
        }
        stage('Launch Container In Testing Environment'){
            steps{
                sshagent(['Environment-Key']) {
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker rm -f mywebserver"
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker run -d -p 80:80 --name mywebserver himanshukr0612/webserver:${BUILD_TAG}"
                }
            }
        }
        stage('Testing Phase'){
            steps{
                sh 'curl -sleep http://13.126.224.70:80/ | grep DevOps'
                sh 'curl -sleep http://13.126.224.70:80/ | grep Jenkins'
            }
        }
        stage('Release To Production'){
            steps{
                input(message: "Release To Production?")
            }
        }
        stage('Deploy In Prod'){
            steps{
                sshagent(['Environment-Key']) {
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker rm -f prod"
                    sh "ssh -o StrictHostKeyChecking=no ec2-user@13.126.224.70 sudo
docker run -d -p 1234:80 --name prod himanshukr0612/webserver:${BUILD_TAG}"
                }
            }
        }
    }
}
```

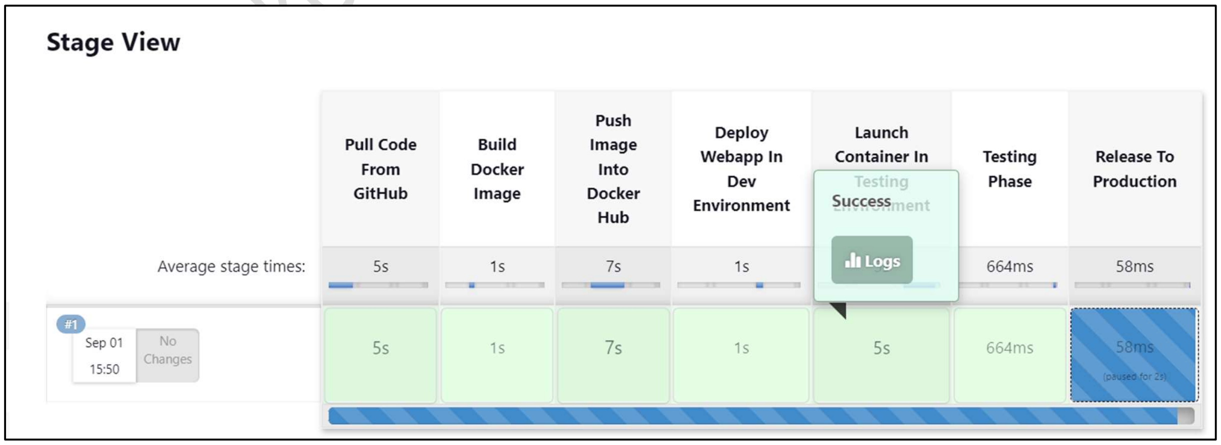
I have also Uploaded this code in my GitHub Link: Click Here

When I Save the code, automatically It will start building.



Stage View

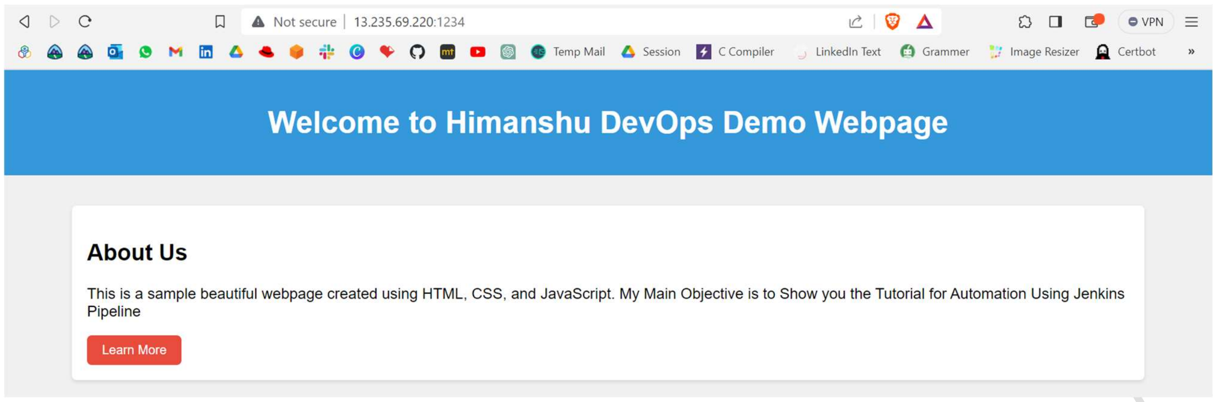
	Pull Code From GitHub	Build Docker Image	Push Image Into Docker Hub
Average stage times:	5s	1s	1s
#1 Sep 01 15:50 No Changes	5s	1s	1s



Stage View

	Pull Code From GitHub	Build Docker Image	Push Image Into Docker Hub	Deploy Webapp In Dev Environment	Launch Container In Testing Environment	Testing Phase	Release To Production
Average stage times:	5s	1s	7s	1s		664ms	58ms
#1 Sep 01 15:50 No Changes	5s	1s	7s	1s	5s	664ms	58ms (paused for 2s)

Here in last stage, It will waiting for manual approval. Sometimes it is very good to have manual approval. If we hover mouse their then it will ask for approval. If will click on approve then it will deploy the code in the production.


Not secure | 13.235.69.220:1234

Temp Mail Session C Compiler LinkedIn Text Grammer Image Resizer Certbot

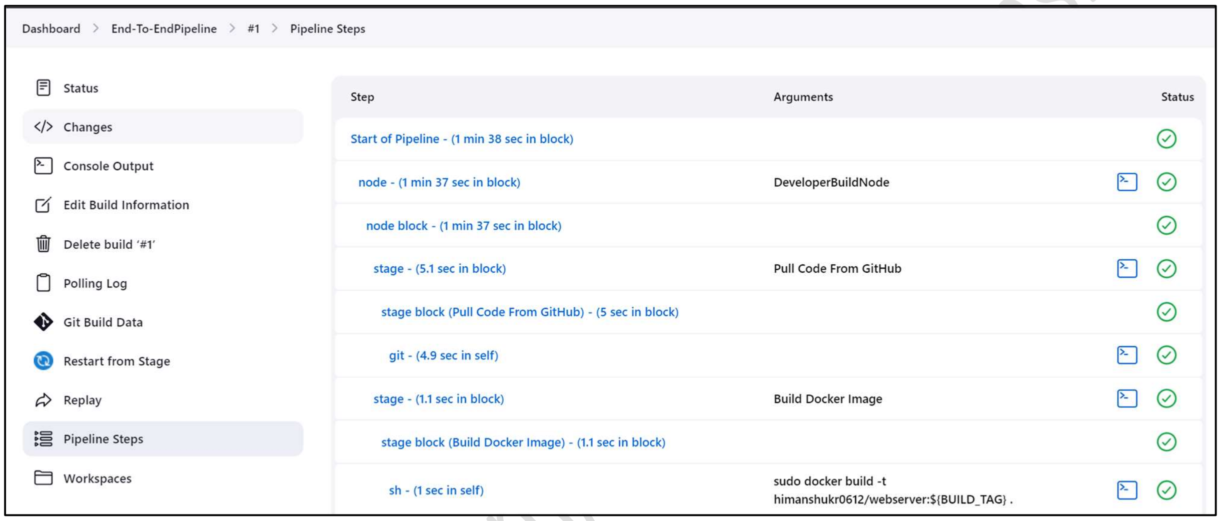
Welcome to Himanshu DevOps Demo Webpage

About Us

This is a sample beautiful webpage created using HTML, CSS, and JavaScript. My Main Objective is to Show you the Tutorial for Automation Using Jenkins Pipeline

Learn More

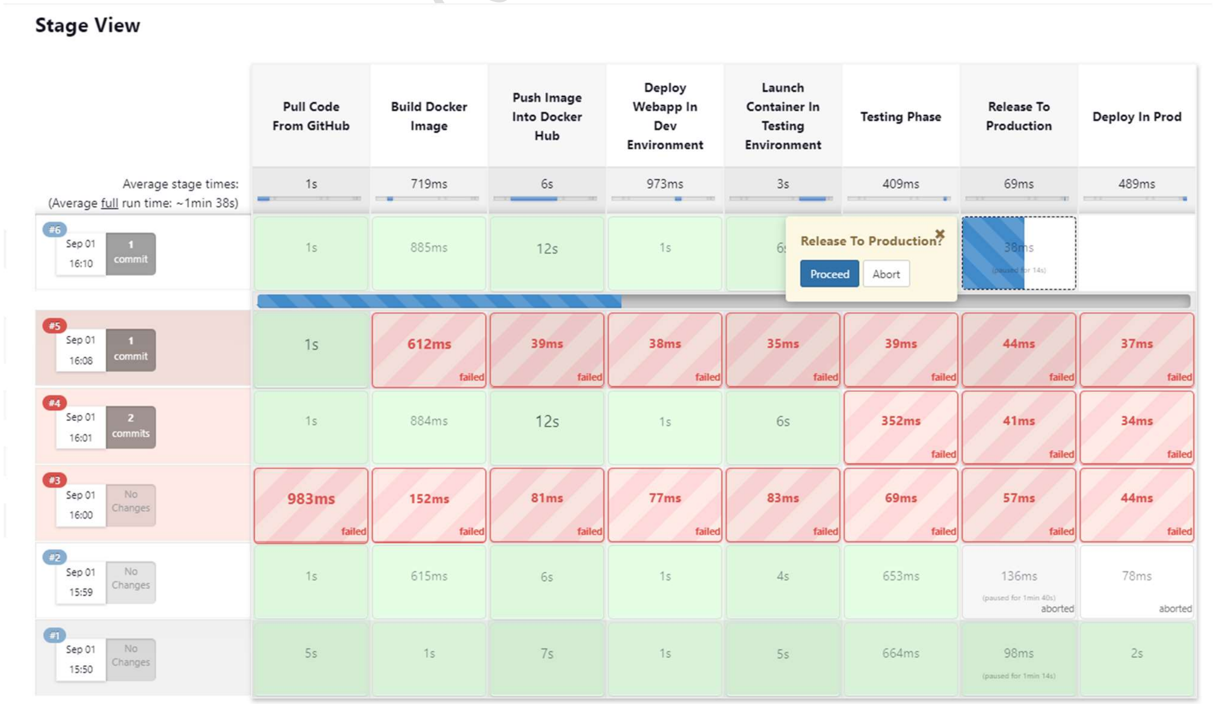
We can visualize all the Steps, after clicking inside the Build Number. It will give the brief information about the job.


Dashboard > End-To-EndPipeline > #1 > Pipeline Steps

- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#1'
- Polling Log
- Git Build Data
- Restart from Stage
- Replay
- Pipeline Steps
- Workspaces

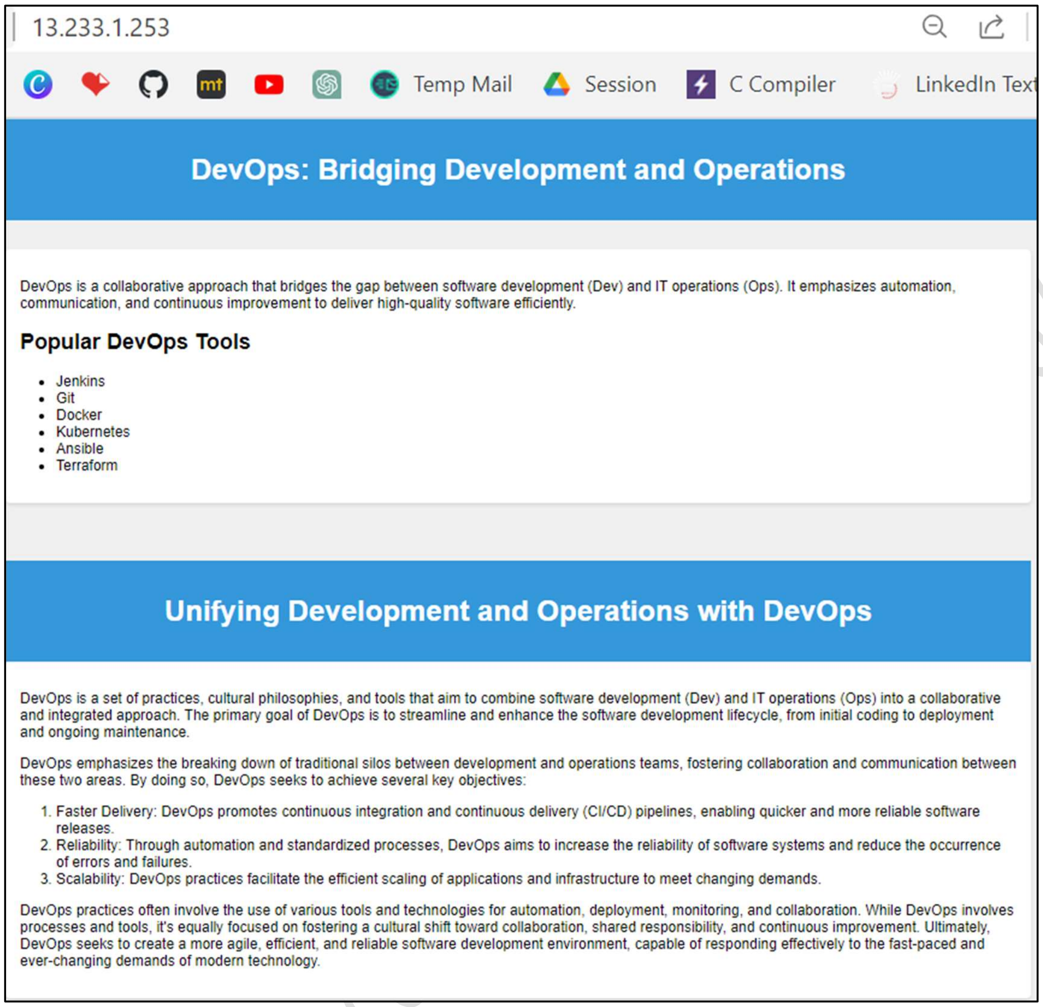
Step	Arguments	Status
Start of Pipeline - (1 min 38 sec in block)		✓
node - (1 min 37 sec in block)	DeveloperBuildNode	✓
node block - (1 min 37 sec in block)		✓
stage - (5.1 sec in block)	Pull Code From GitHub	✓
stage block (Pull Code From GitHub) - (5 sec in block)		✓
git - (4.9 sec in self)		✓
stage - (1.1 sec in block)	Build Docker Image	✓
stage block (Build Docker Image) - (1.1 sec in block)		✓
sh - (1 sec in self)	sudo docker build -t himanshukr0612/webserver:$(BUILD_TAG) .	✓

Now let's update the entire code. And push in the GitHub.


Stage View

	Pull Code From GitHub	Build Docker Image	Push Image Into Docker Hub	Deploy Webapp In Dev Environment	Launch Container In Testing Environment	Testing Phase	Release To Production	Deploy In Prod
Average stage times: (Average full run time: ~1min 38s)	1s	719ms	6s	973ms	3s	409ms	69ms	489ms
#6 Sep 01 16:10, 1 commit	1s	885ms	12s	1s	6s	Release To Production? Proceed Abort	38ms (paused for 14s)	
#5 Sep 01 16:08, 1 commit	1s	612ms failed	39ms failed	38ms failed	35ms failed	39ms failed	44ms failed	37ms failed
#4 Sep 01 16:01, 2 commits	1s	884ms	12s	1s	6s	352ms failed	41ms failed	34ms failed
#3 Sep 01 16:00, No Changes	983ms failed	152ms failed	81ms failed	77ms failed	83ms failed	69ms failed	57ms failed	44ms failed
#2 Sep 01 15:59, No Changes	1s	615ms	6s	1s	4s	653ms	136ms (paused for 1min 40s) aborted	78ms aborted
#1 Sep 01 15:50, No Changes	5s	1s	7s	1s	5s	664ms	98ms (paused for 1min 14s)	2s

After we approve the code in final step. The code will change in the Live Page. Here we are achieving Fully Continuous Integration and Continuous Delivery.



DevOps: Bridging Development and Operations

DevOps is a collaborative approach that bridges the gap between software development (Dev) and IT operations (Ops). It emphasizes automation, communication, and continuous improvement to deliver high-quality software efficiently.

Popular DevOps Tools

- Jenkins
- Git
- Docker
- Kubernetes
- Ansible
- Terraform

Unifying Development and Operations with DevOps

DevOps is a set of practices, cultural philosophies, and tools that aim to combine software development (Dev) and IT operations (Ops) into a collaborative and integrated approach. The primary goal of DevOps is to streamline and enhance the software development lifecycle, from initial coding to deployment and ongoing maintenance.

DevOps emphasizes the breaking down of traditional silos between development and operations teams, fostering collaboration and communication between these two areas. By doing so, DevOps seeks to achieve several key objectives:

1. Faster Delivery: DevOps promotes continuous integration and continuous delivery (CI/CD) pipelines, enabling quicker and more reliable software releases.
2. Reliability: Through automation and standardized processes, DevOps aims to increase the reliability of software systems and reduce the occurrence of errors and failures.
3. Scalability: DevOps practices facilitate the efficient scaling of applications and infrastructure to meet changing demands.

DevOps practices often involve the use of various tools and technologies for automation, deployment, monitoring, and collaboration. While DevOps involves processes and tools, it's equally focused on fostering a cultural shift toward collaboration, shared responsibility, and continuous improvement. Ultimately, DevOps seeks to create a more agile, efficient, and reliable software development environment, capable of responding effectively to the fast-paced and ever-changing demands of modern technology.

Continuous Integration and Continuous Deployment (CI/CD) is crucial in the software development process because it ensures that code changes are tested, integrated, and delivered quickly and reliably. Without CI/CD, software development can become slow, error-prone, and less efficient.

CI/CD automates the building, testing, and deployment of software, allowing developers to catch and fix bugs early in the development cycle. This leads to higher software quality, fewer production issues, and faster development cycles. It also enables teams to release new features and updates more frequently, improving the user experience and staying competitive in the market.

Furthermore, CI/CD promotes collaboration among developers, testers, and operations teams by providing a standardized and automated process for code changes. It reduces the manual effort required for deployment, leading to cost savings and increased productivity.

In summary, CI/CD is essential because it accelerates the software development process, enhances software quality, reduces errors, and fosters collaboration. It's a key practice for modern software development that enables organizations to deliver better software faster and more reliably.

<u>THANK YOU</u>