Abstract

The By automating the entire build, test, and deployment process, this CI/CD pipeline significantly improves the efficiency and reliability of software development. Developers can focus on writing code while the pipeline handles the rest, leading to faster development cycles and reduced time to market. The use of Docker ensures consistent and reproducible deployments across different environments, minimizing the risk of configuration issues and reducing downtime.

The pipeline begins with a GitHub webhook that triggers Jenkins whenever code is pushed to the repository. Jenkins then pulls the latest code, builds a Docker image using the provided Dockerfile, and runs any necessary tests. Once the tests pass, Ansible takes over, using its playbooks to provision the necessary infrastructure (e.g., virtual machines, network configurations) and deploy the Docker image into the target environment. This automated process streamlines the entire release process, making it faster, more reliable, and less prone to human error.

This CI/CD pipeline represents a significant step towards modernizing software development practices. By leveraging tools like Jenkins, Ansible, and Docker, the project demonstrates a robust and efficient approach to building, testing, and deploying applications. This automation not only accelerates the development process but also enhances the quality and consistency of software releases, ultimately leading to improved customer satisfaction and a competitive advantage.

Table of contents

Introduction

Configuring host key verification for Jenkins is crucial for secure communication between your Jenkins server and remote repositories. This guide will walk you through the process in a comprehensive and easy-to-follow manner.

First, access your Jenkins dashboard through your web browser. This is your central hub for managing Jenkins, where you can configure various settings and monitor your builds. Once you're on the dashboard, look for the 'Manage Jenkins' option in the left-hand menu. Click on it to open the Jenkins configuration page.

Next, navigate to the 'Configure Global Security' section. This section allows you to define security settings that apply to your entire Jenkins instance. Scroll down until you find the 'Git Host Key Verification Configuration' section. This is where you can specify how Jenkins should verify the SSH host keys of remote Git repositories.

At this point, you'll need to set up a known_hosts file, which Jenkins will use to verify the host keys. If you don't already have a known_hosts file, you'll need to create one. Start by opening a terminal on your Jenkins server. You'll use the SSH-keyscan command to retrieve the SSH host keys from your Git server and add them to the known_hosts file. For example, if you're using GitHub, you can run the following command:

```
ssh-keyscan github.com >> /var/lib/jenkins/.ssh/known_hosts
```

This command fetches the SSH host keys for GitHub and appends them to the known_hosts file located in the specified directory. Ensure that the Jenkins user has read access to this file. You can adjust the permissions using the chmod command if necessary.

After setting up the known_hosts file, return to the Jenkins configuration page. In the 'Git Host Key Verification Configuration' section, select the 'Known hosts file' strategy. You'll be prompted to provide the path to your known_hosts file. Enter the path where you saved the file (e.g., /var/lib/jenkins/.ssh/known_hosts).

Save your changes by clicking on the 'Apply' and 'Save' buttons at the bottom of the page. This will update Jenkins' configuration with your new host key verification settings. To ensure everything is working correctly, try running a build that interacts with your remote Git repository. If the build succeeds without any host key verification errors, you've successfully configured Jenkins to use your known_hosts file.

In addition to improving security, configuring host key verification can help prevent issues caused by man-in-the-middle attacks or incorrect host keys. By verifying the SSH host keys, Jenkins ensures that it is connecting to the correct remote server, reducing the risk of unauthorized access.

Overall, this setup process involves accessing Jenkins' security settings, creating and populating a known_hosts file, and configuring Jenkins to use this file for host key verification. With these steps, you can enhance the security of your Jenkins builds and ensure reliable communication with your remote repositories.

Tools and Technologies Used

Jenkins:

Jenkins is an open-source automation server that streamlines the software development process by automating tasks such as building, testing, and deploying applications. By supporting a wide range of plugins, Jenkins integrates seamlessly with various tools and technologies, enabling continuous integration and continuous delivery (CI/CD). Its user-friendly interface, coupled with features like distributed builds, pipeline as code, and extensive reporting capabilities, makes Jenkins a versatile and powerful tool for development teams. With Jenkins, teams can enhance their workflow efficiency, maintain high code quality, and accelerate the delivery of software projects.

Git: A distributed version control system used to track changes in source code during software development.

GitHub: A web-based platform for version control using Git, providing hosting for software development and version control using Git.

SSH (Secure Shell): A cryptographic network protocol for operating network services securely over an unsecured network.

SSH-keyscan:A utility for gathering the SSH public keys of servers.

Known_hosts File: A file that contains the SSH host keys for remote servers, used to verify the identity of the servers Jenkins connects to.

SSH Configuration: Securely configuring access and permissions for the Jenkins user to interact with the SSH host keys.

Jenkins Security Settings: Configurations within Jenkins that allow you to define how host key verification should be handled.

Unix/Linux Commands: Basic commands like chmod to adjust file permissions.

Docker:

An open-source platform that automates the deployment, scaling, and management of applications by using containerization technology to package applications and their dependencies into lightweight, portable containers.

Each of these tools and technologies plays a crucial role in ensuring a secure, efficient, and automated software development process. By using these tools in tandem, you can streamline the workflow from code development to deployment, ensuring consistency and reliability across different environments. Docker, in particular, enhances the portability and scalability of your applications, allowing for seamless integration with other components of your development pipeline.

Implementation

Set Up the Environment

- **Tools Required:**
  - **Jenkins:** Install Jenkins on a server or cloud instance (e.g., AWS, Azure).
  - **GitHub:** Repository to store source code.
  - **Docker:** For containerization.
  - **Ansible:** For configuration management and deployment automation.
- Ensure all tools are installed and configured on your server or virtual machine.

2. Step-by-Step Implementation

Step 1: Create and Push Code to GitHub

- Develop your application or microservice and push the code to a GitHub repository.
- Add a Jenkinsfile in the repository to define your CI/CD pipeline.

Step 2: Configure GitHub Webhooks

- In your GitHub repository:
  - Go to **Settings > Webhooks > Add Webhook**.
  - Set the **Payload URL** to your Jenkins server's GitHub webhook endpoint (e.g., `http://<JENKINS_SERVER>:8080/github-webhook/`).
  - Select `application/json` as the content type and ensure **Push events** are selected.

Step 3: Set Up Jenkins

- **Install Required Plugins:**
  - GitHub Integration Plugin.

    - Docker Plugin.
    - Ansible Plugin.
- **Create a New Job:**
    - Select **Pipeline** project.
    - Configure the GitHub repository URL and credentials in the job settings.
- **Pipeline Script:**
    - Use the Jenkinsfile in the repository. It should include steps to:
        1. Pull code from GitHub.
        2. Build a Docker image.
        3. Push the image to a Docker registry.
        4. Use Ansible to deploy the application.

Step 4: Build and Push Docker Image

- Define Docker steps in the pipeline:

```
groovy
CopyEdit
pipeline {
    agent any
    stages {
        stage('Clone Repository') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                sh 'docker build -t your-image-name .'
            }
        }
        stage('Push to Docker Registry') {
            steps {
                sh '''
                docker login -u <username> -p <password>
                docker push your-image-name
                '''
            }
        }
    }
}
```

Step 5: Trigger Deployment

- Update the pipeline to invoke the Ansible playbook after building and pushing the Docker image:

  ```
  groovy
  CopyEdit
  stage('Deploy with Ansible') {
      steps {
          sh 'ansible-playbook -i inventory playbook.yml'
      }
  }
  ```

3. Testing and Validation

- Push changes to GitHub and verify the entire pipeline:
  - Jenkins should automatically trigger a build.
  - Docker image is created and pushed.
  - Ansible deploys the container.

Result

1. Code Push to GitHub

- **Result:**
  - Your codebase is successfully pushed to the GitHub repository.
  - Webhook triggers Jenkins as configured.

2. Jenkins Pipeline Execution

Stage 1: Clone Repository

- **Result:**
  - Jenkins fetches the latest code from GitHub.
  - Build logs confirm successful code checkout:

```
CopyEdit
Cloning the repository...
Repository cloned successfully!
```

Stage 2: Build Docker Image

- **Result:**
  - Docker successfully builds an image of your application using the `Dockerfile` in the repository.
  - Build logs confirm:

```
vbnet
CopyEdit
Step 1/5: FROM base-image
Step 2/5: ADD application files
Step 3/5: EXPOSE 80
Image built successfully: your-docker-image-name
```

Stage 3: Push Docker Image

- **Result:**
  - The Docker image is pushed to a Docker registry (e.g., Docker Hub or private registry).
  - Push logs confirm:

    ```
    arduino
    CopyEdit
    Logging into Docker Hub...
    Pushing image: your-docker-image-name:latest
    Image pushed successfully!
    ```

Stage 4: Deploy Container

- **Result:**
  - Docker pulls the image to the deployment server.
  - Any existing container with the same name is stopped and removed.
  - A new container is deployed, mapped to the desired ports.
  - Deployment logs confirm:

    ```
    sql
    CopyEdit
    Pulling latest Docker image: your-docker-image-name
    Stopping old container (if exists)...
    Removing old container (if exists)...
    Running new container: your-container-name
    Application deployed and running at http://<server-ip>:80
    ```

3. Deployment Verification

- Access the deployed application via a browser using the deployment server's IP or domain (e.g., `http://<server-ip>`).
- Verify that the application is up and running.

4. Continuous Integration and Delivery

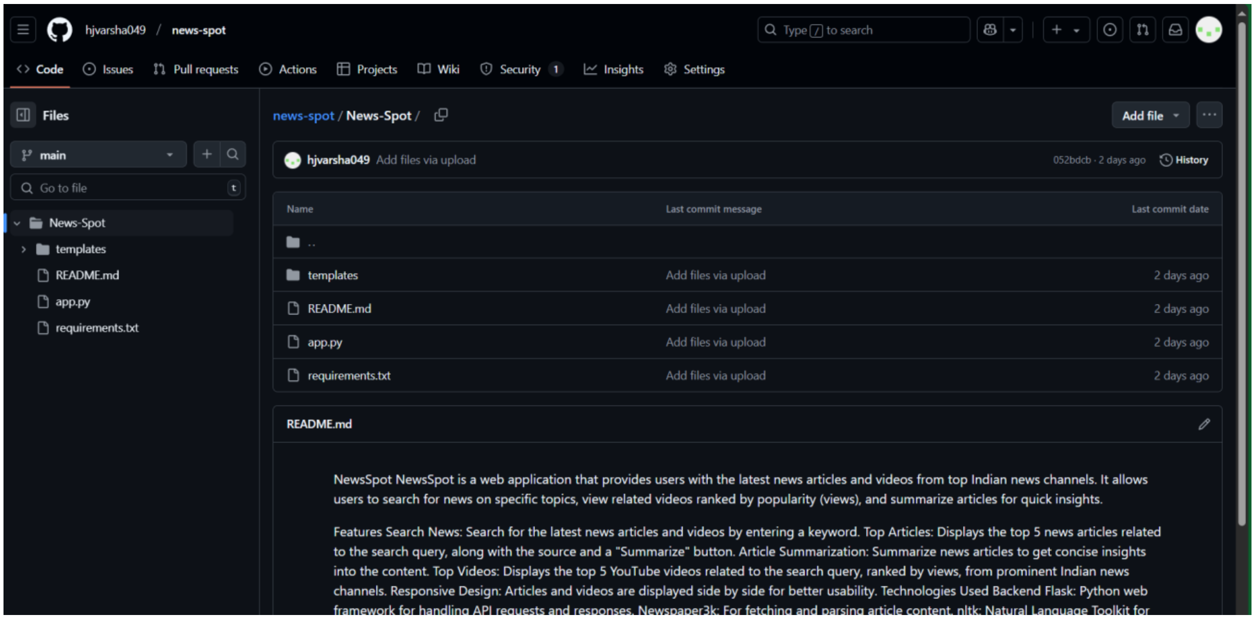
- Every subsequent code push to GitHub triggers the Jenkins pipeline automatically (due to webhooks).
- The pipeline repeats the above process:
  - Builds the new image.
  - Pushes it to the Docker registry.
  - Replaces the running container with the new version.

Expected Outcome

- **End-to-End Automation:** Code changes are automatically built, tested, pushed to a registry, and deployed without manual intervention.
- **Live Application:** The application is live and updated with each new push.
- **Pipeline Efficiency:** Logs for each stage provide visibility into the process, ensuring errors can be quickly addressed.

Screen shots

Github



hjvarsha049 Add files via upload 052bdcb · 2 days ago

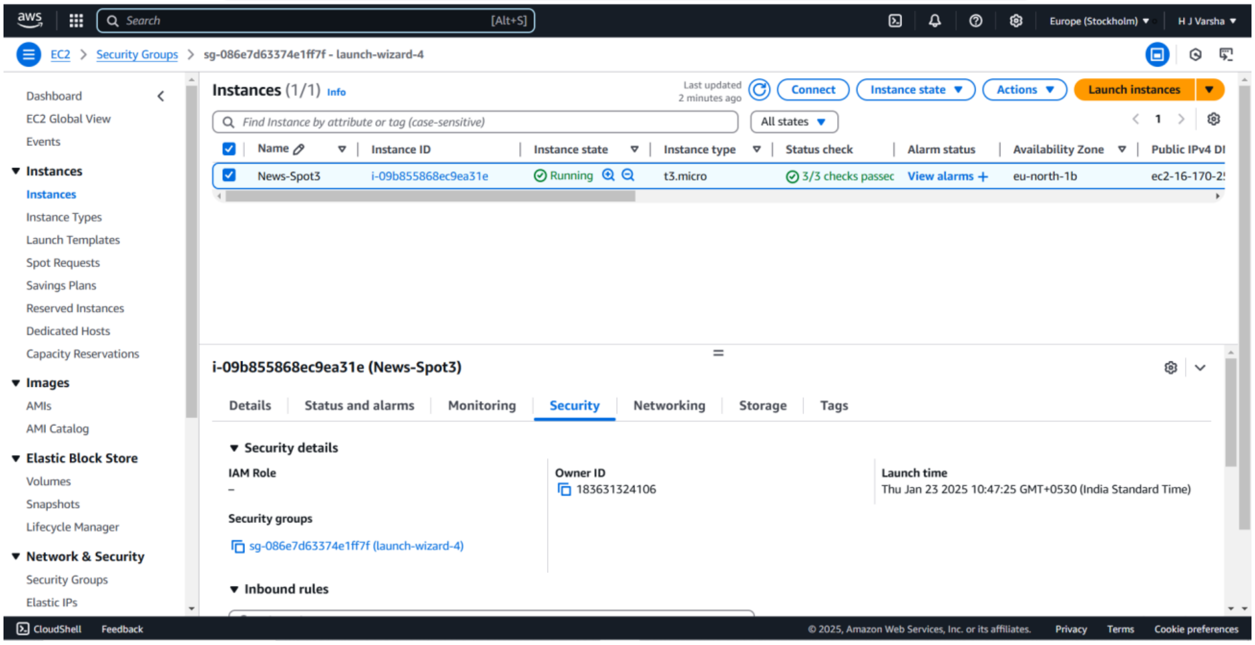
Name	Last commit message	Last commit date
..		
templates	Add files via upload	2 days ago
README.md	Add files via upload	2 days ago
app.py	Add files via upload	2 days ago
requirements.txt	Add files via upload	2 days ago

README.md

NewsSpot NewsSpot is a web application that provides users with the latest news articles and videos from top Indian news channels. It allows users to search for news on specific topics, view related videos ranked by popularity (views), and summarize articles for quick insights.

Features Search News: Search for the latest news articles and videos by entering a keyword. Top Articles: Displays the top 5 news articles related to the search query, along with the source and a "Summarize" button. Article Summarization: Summarize news articles to get concise insights into the content. Top Videos: Displays the top 5 YouTube videos related to the search query, ranked by views, from prominent Indian news channels. Responsive Design: Articles and videos are displayed side by side for better usability. Technologies Used Backend Flask: Python web framework for handling API requests and responses. Newspaper3k: For fetching and parsing article content. nltk: Natural Language Toolkit for

Launch instances



Instances (1/1) Info

☑	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DI
☑	News-Spot3	i-09b855868ec9ea31e	Running	t3.micro	3/3 checks passec	View alarms +	eu-north-1b	ec2-16-170-2!

i-09b855868ec9ea31e (News-Spot3)

Security details

IAM Role
–

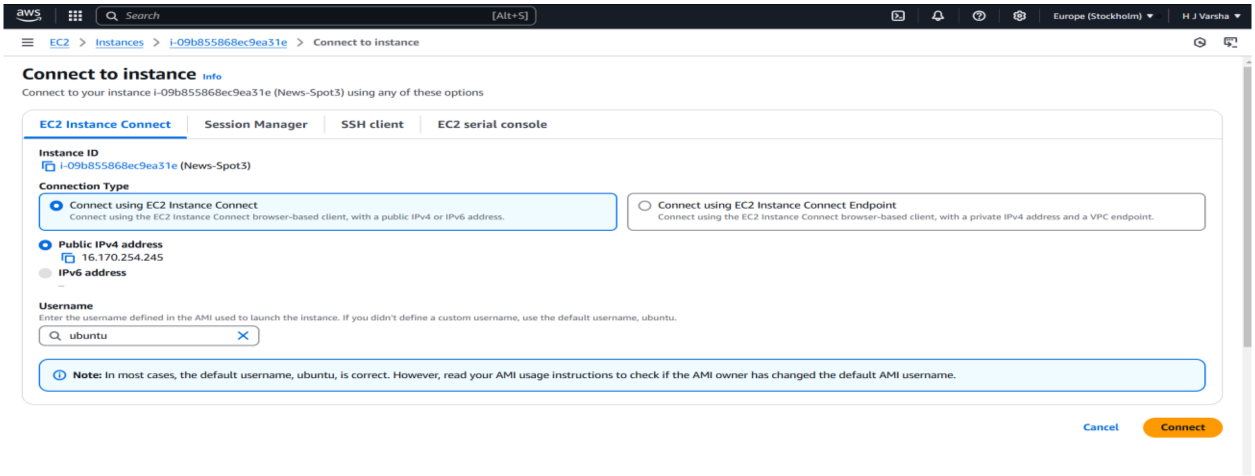
Owner ID
183631324106

Launch time
Thu Jan 23 2025 10:47:25 GMT+0530 (India Standard Time)

Security groups
sg-086e7d63374e1ff7f (launch-wizard-4)

Inbound rules

Connect EC2 instance connect browser-base client



Connect to instance Info

Connect to your instance i-09b855868ec9ea31e (News-Spot3) using any of these options

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID
i-09b855868ec9ea31e (News-Spot3)

Connection Type

● Connect using EC2 Instance Connect
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

○ Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

● Public IPv4 address
16.170.254.245

○ IPv6 address
–

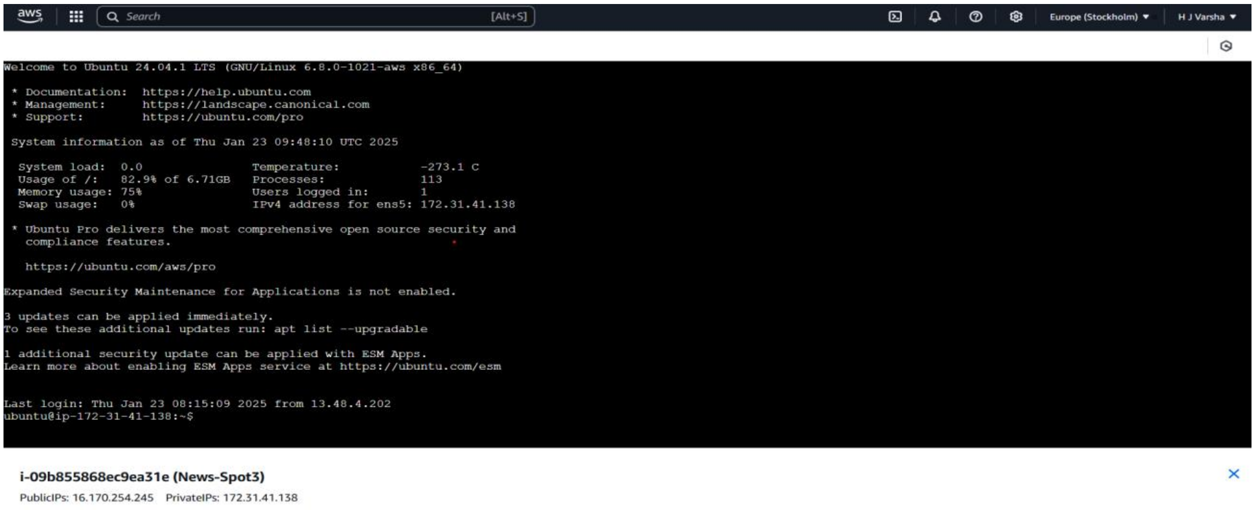
Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ubuntu.

ubuntu

Note: In most cases, the default username, ubuntu, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel Connect

terminal session of an AWS EC2 instance.

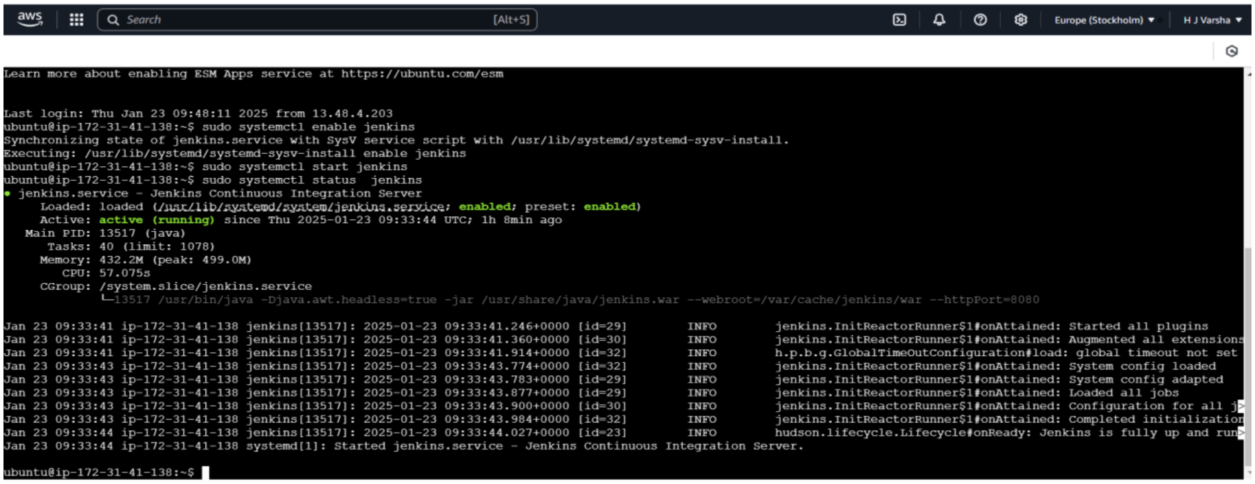


```
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1021-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Thu Jan 23 09:48:10 UTC 2025

  System load:  0.0               Temperature:           -273.1 C
  Usage of /:   82.9% of 6.71GB   Processes:             113
  Memory usage: 75%               Users logged in:       1
  Swap usage:   0%                IPv4 address for ens5: 172.31.41.138

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

1 additional security update can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm


Last login: Thu Jan 23 08:15:09 2025 from 13.48.4.202
ubuntu@ip-172-31-41-138:~$
```

i-09b855868ec9ea31e (News-Spot3)

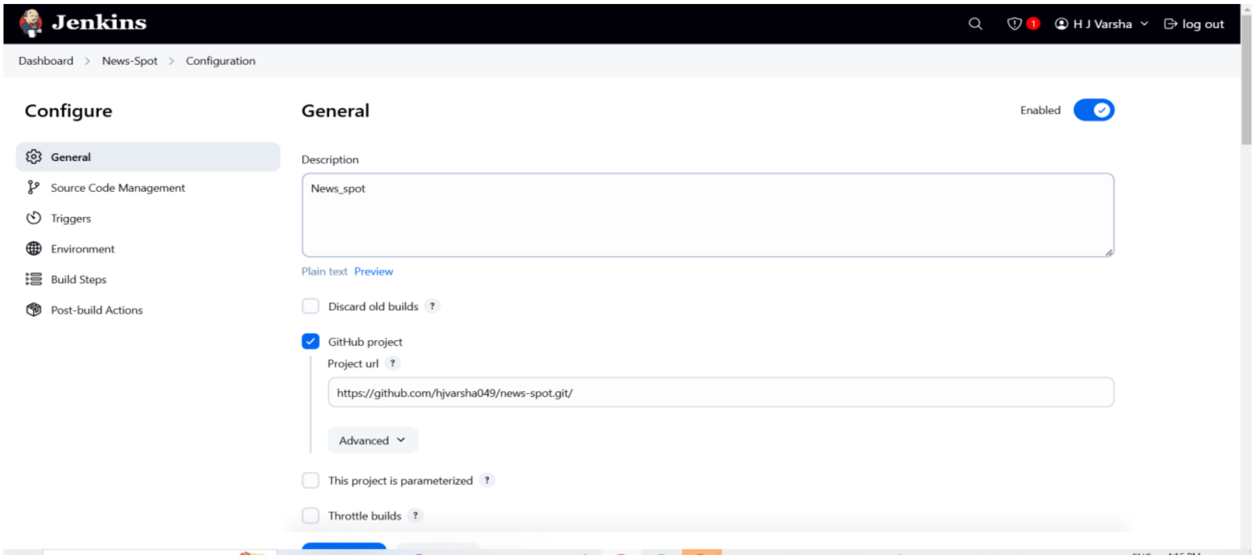
PublicIPs: 16.170.254.245   PrivateIPs: 172.31.41.138

Jenkins service running on an Ubuntu server



```
Learn more about enabling ESM Apps service at https://ubuntu.com/esm


Last login: Thu Jan 23 09:48:11 2025 from 13.48.4.203
ubuntu@ip-172-31-41-138:~$ sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-41-138:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-41-138:~$ sudo systemctl status  jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Thu 2025-01-23 09:33:44 UTC; 1h 8min ago
   Main PID: 13517 (java)
      Tasks: 40 (limit: 1078)
     Memory: 432.2M (peak: 499.0M)
        CPU: 57.075s
     CGroup: /system.slice/jenkins.service
             └─13517 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Jan 23 09:33:41 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:41.246+0000 [id=29]        INFO        jenkins.InitReactorRunner$1#onAttained: Started all plugins
Jan 23 09:33:41 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:41.360+0000 [id=30]        INFO        jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
Jan 23 09:33:41 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:41.914+0000 [id=32]        INFO        h.p.b.g.GlobalTimeOutConfiguration#load: global timeout not set
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.774+0000 [id=32]        INFO        jenkins.InitReactorRunner$1#onAttained: System config loaded
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.783+0000 [id=29]        INFO        jenkins.InitReactorRunner$1#onAttained: System config adapted
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.877+0000 [id=29]        INFO        jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.900+0000 [id=30]        INFO        jenkins.InitReactorRunner$1#onAttained: Configuration for all j>
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.984+0000 [id=32]        INFO        jenkins.InitReactorRunner$1#onAttained: Completed initialization
Jan 23 09:33:44 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:44.027+0000 [id=23]        INFO        hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and run>
Jan 23 09:33:44 ip-172-31-41-138 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.

ubuntu@ip-172-31-41-138:~$
```

Jenkins job configuration page



General

Enabled

Description: News_spot

Plain text Preview

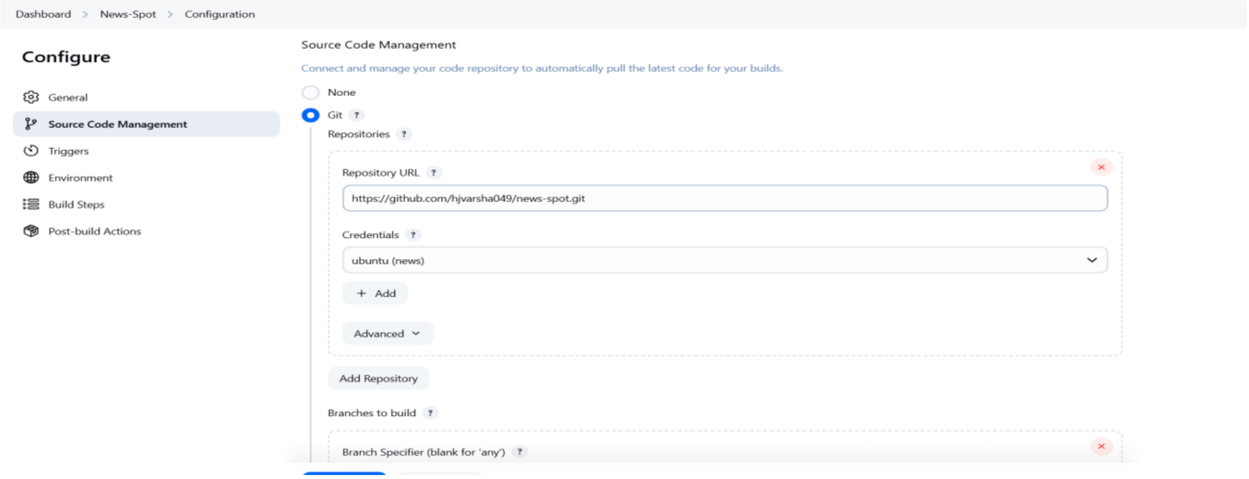
☐ Discard old builds ?

☑ GitHub project

Project url ? https://github.com/hjvarsha049/news-spot.git/

☐ This project is parameterized ?

☐ Throttle builds ?



Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

○ None

● Git ?

Repositories ?

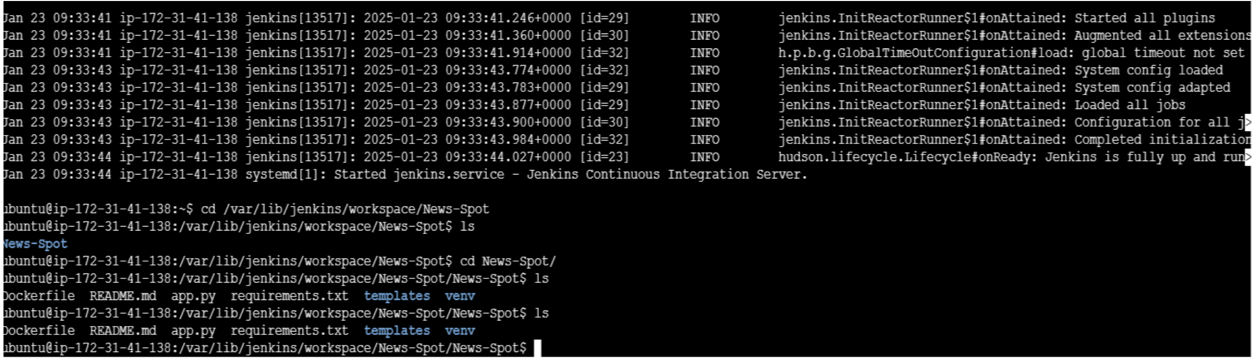
Repository URL ? https://github.com/hjvarsha049/news-spot.git

Credentials ? ubuntu (news)

Branches to build ?

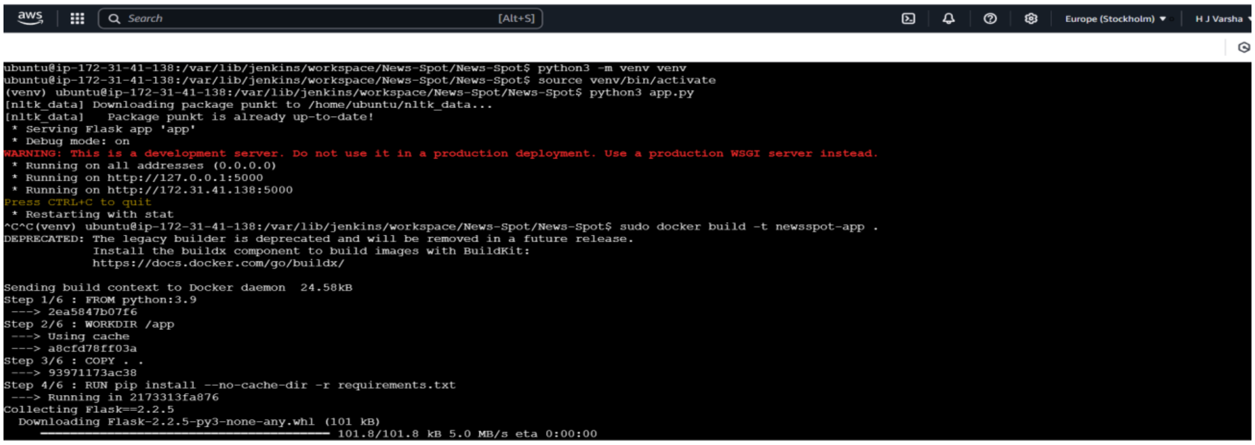
Branch Specifier (blank for 'any') ?

The path /var/lib/jenkins/workspace/News-Spot/ contains the files checked out by Jenkins from your GitHub repository.



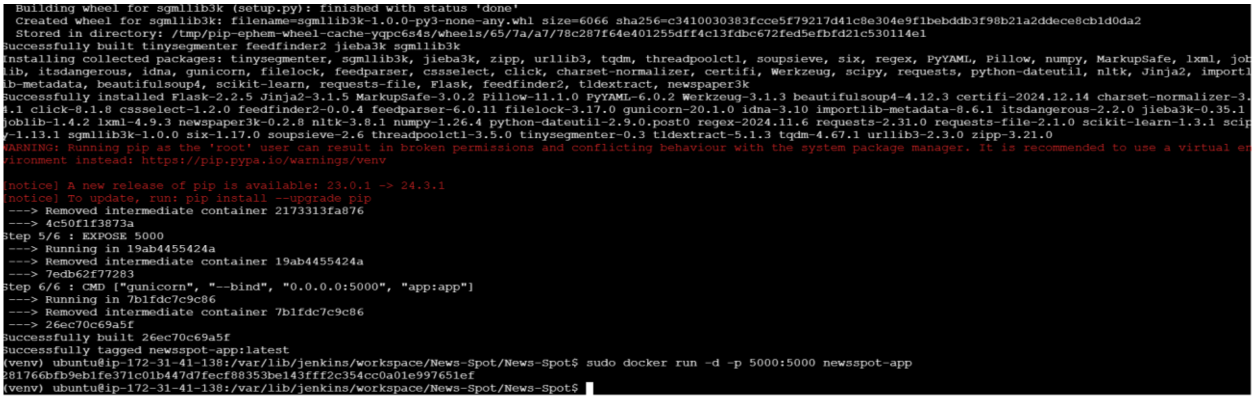
```
Jan 23 09:33:41 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:41.246+0000 [id=29]       INFO       jenkins.InitReactorRunner$1#onAttained: Started all plugins
Jan 23 09:33:41 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:41.360+0000 [id=30]       INFO       jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
Jan 23 09:33:41 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:41.914+0000 [id=32]       INFO       h.p.b.g.GlobalTimeOutConfiguration#load: global timeout not set
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.774+0000 [id=32]       INFO       jenkins.InitReactorRunner$1#onAttained: System config loaded
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.783+0000 [id=29]       INFO       jenkins.InitReactorRunner$1#onAttained: System config adapted
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.877+0000 [id=29]       INFO       jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.900+0000 [id=30]       INFO       jenkins.InitReactorRunner$1#onAttained: Configuration for all j>
Jan 23 09:33:43 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:43.984+0000 [id=32]       INFO       jenkins.InitReactorRunner$1#onAttained: Completed initialization
Jan 23 09:33:44 ip-172-31-41-138 jenkins[13517]: 2025-01-23 09:33:44.027+0000 [id=23]       INFO       hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and run>
Jan 23 09:33:44 ip-172-31-41-138 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.

ubuntu@ip-172-31-41-138:~$ cd /var/lib/jenkins/workspace/News-Spot
ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot$ ls
News-Spot
ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot$ cd News-Spot/
ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$ ls
Dockerfile  README.md  app.py  requirements.txt  templates  venv
ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$ ls
Dockerfile  README.md  app.py  requirements.txt  templates  venv
ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$
```

Flask App Setup and Docker Build

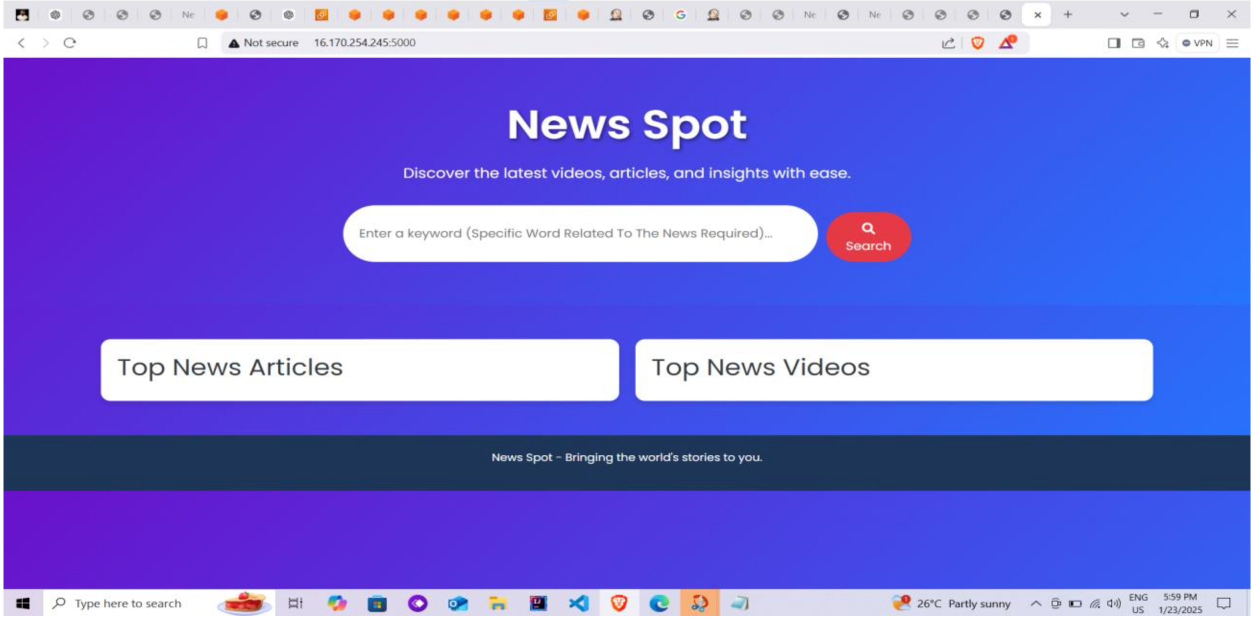


```
ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$ python3 -m venv venv
ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$ source venv/bin/activate
(venv) ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$ python3 app.py
[nltk_data] Downloading package punkt to /home/ubuntu/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
 * Serving Flask app 'app'
 * 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://172.31.41.138:5000
Press CTRL+C to quit
 * Restarting with stat
^C^C(venv) ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$ sudo docker build -t newsspot-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  24.58kB
Step 1/6 : FROM python:3.9
 ---> 2ea5847b07f6
Step 2/6 : WORKDIR /app
 ---> Using cache
 ---> a8cfd78ff03a
Step 3/6 : COPY . .
 ---> 93971173ac38
Step 4/6 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Running in 2173313fa876
Collecting Flask==2.2.5
  Downloading Flask-2.2.5-py3-none-any.whl (101 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.8/101.8 kB 5.0 MB/s eta 0:00:00
```

Create the Docker



```
  Building wheel for sgmllib3k (setup.py): finished with status 'done'
  Created wheel for sgmllib3k: filename=sgmllib3k-1.0.0-py3-none-any.whl size=6066 sha256=c3410030383fcce5f79217d41c8e304e9f1bebddb3f98b21a2ddece8cb1d0da2
  Stored in directory: /tmp/pip-ephem-wheel-cache-yqpc6s4s/wheels/65/7a/a7/78c287f64e401255dff4c13fdbc672fed5efbfd21c530114e1
Successfully built tinysegmenter feedfinder2 jieba3k sgmllib3k
Installing collected packages: tinysegmenter, sgmllib3k, jieba3k, zipp, urllib3, tqdm, threadpoolctl, soupsieve, six, regex, PyYAML, Pillow, numpy, MarkupSafe, lxml, job
lib, itsdangerous, idna, gunicorn, filelock, feedparser, cssselect, click, charset-normalizer, certifi, Werkzeug, scipy, requests, python-dateutil, nltk, Jinja2, importl
ib-metadata, beautifulsoup4, scikit-learn, requests-file, Flask, feedfinder2, tldextract, newspaper3k
Successfully installed Flask-2.2.5 Jinja2-3.1.5 MarkupSafe-3.0.2 Pillow-11.1.0 PyYAML-6.0.2 Werkzeug-3.1.3 beautifulsoup4-4.12.3 certifi-2024.12.14 charset-normalizer-3.
4.1 click-8.1.8 cssselect-1.2.0 feedfinder2-0.0.4 feedparser-6.0.11 filelock-3.17.0 gunicorn-20.1.0 idna-3.10 importlib-metadata-8.6.1 itsdangerous-2.2.0 jieba3k-0.35.1
joblib-1.4.2 lxml-4.9.3 newspaper3k-0.2.8 nltk-3.8.1 numpy-1.26.4 python-dateutil-2.9.0.post0 regex-2024.11.6 requests-2.31.0 requests-file-2.1.0 scikit-learn-1.3.1 scip
y-1.13.1 sgmllib3k-1.0.0 six-1.17.0 soupsieve-2.6 threadpoolctl-3.5.0 tinysegmenter-0.3 tldextract-5.1.3 tqdm-4.67.1 urllib3-2.3.0 zipp-3.21.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual en
vironment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
 ---> Removed intermediate container 2173313fa876
 ---> 4c50f1f3873a
Step 5/6 : EXPOSE 5000
 ---> Running in 19ab4455424a
 ---> Removed intermediate container 19ab4455424a
 ---> 7edb62f77283
Step 6/6 : CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
 ---> Running in 7b1fdc7c9c86
 ---> Removed intermediate container 7b1fdc7c9c86
 ---> 26ec70c69a5f
Successfully built 26ec70c69a5f
Successfully tagged newsspot-app:latest
(venv) ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$ sudo docker run -d -p 5000:5000 newsspot-app
281766bfb9eb1fe371c01b447d7fecf88353be143fff2c354cc0a01e997651ef
(venv) ubuntu@ip-172-31-41-138:/var/lib/jenkins/workspace/News-Spot/News-Spot$
```



News Spot

Discover the latest videos, articles, and insights with ease.

Enter a keyword (Specific Word Related To The News Required)...

Search

Top News Articles

Top News Videos

News Spot - Bringing the world's stories to you.

Conclusion

The image depicts a typical CI/CD pipeline utilizing popular tools like Jenkins, Ansible, and Docker. The process begins with developers pushing code changes to a GitHub repository. This triggers a webhook, notifying Jenkins of the update. Jenkins then initiates an automated build process, resulting in the creation of a Docker image containing the latest code.

Once the Docker image is built, Ansible takes over. Ansible, an automation platform, leverages its orchestration capabilities to deploy the newly created Docker image to the designated target environment. This could be a server, a cloud instance, or any other infrastructure where the application needs to run.

The entire pipeline is designed to be continuous, ensuring that code changes are rapidly tested, built, and deployed. This approach significantly reduces the time and effort required for software releases while minimizing the risk of errors and inconsistencies. By automating the deployment process, organizations can achieve faster development cycles and deliver software updates more frequently to their users.

References

General CI/CD Concepts:

- **Continuous Integration and Continuous Delivery (CI/CD):**
  - Wikipedia: https://en.wikipedia.org/wiki/Continuous_integration
  - YouTube
  - Chatgpt: https://chatgpt.com/share/67903643-a054-8004-b598-0b12aaa56367

Tools Used in the Pipeline:

- **Jenkins:**
  - Jenkins Website: https://www.jenkins.io/
- **Docker:**
  - Docker Website: https://www.docker.com/
- **GitHub:**
  - GitHub Website: https://github.com/