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.

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.
    - **Ansible:** For configuration management and deployment automation.
- Ensure all tools are installed and configured on your server or virtual machine.

• 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. Push the image to a Docker registry.
    3. Use Ansible to deploy the application.

Step 4:Testing and Validation

- Push changes to GitHub and verify the entire pipeline:
  Jenkins should automatically trigger a build

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!
```

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

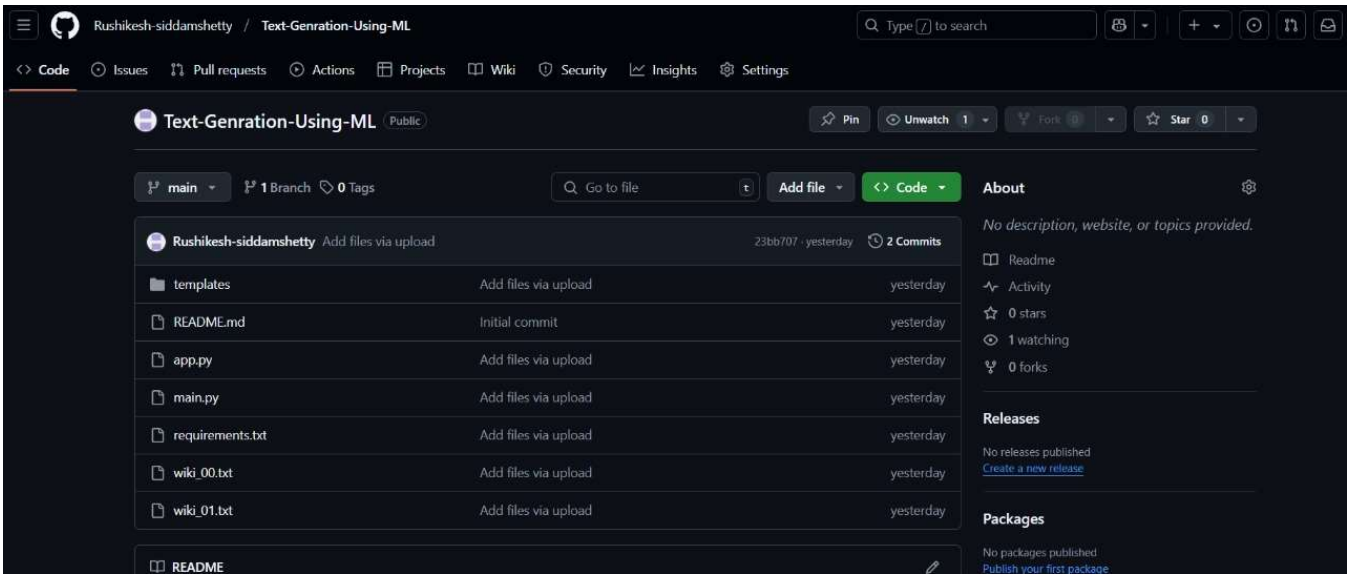
- Every subsequent code push to GitHub triggers the Jenkins pipeline automatically (due to webhooks).
- The pipeline repeats the above process:
  - Builds the new image.
  - Pushes it to the Docker registry.
  - Replaces the running container with the new version.

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 address

Screen shots

Github

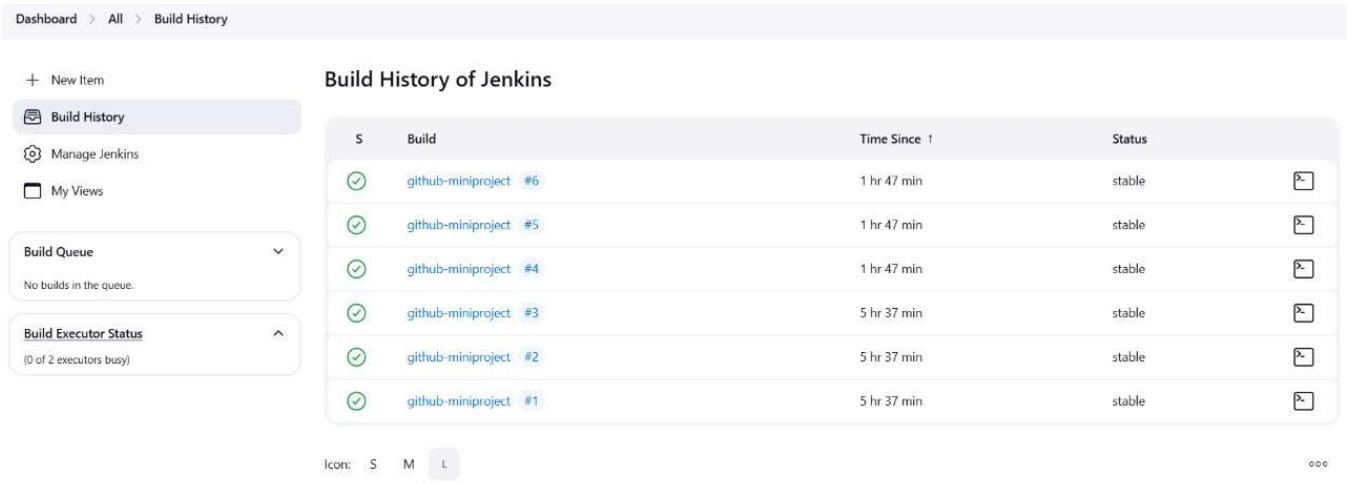


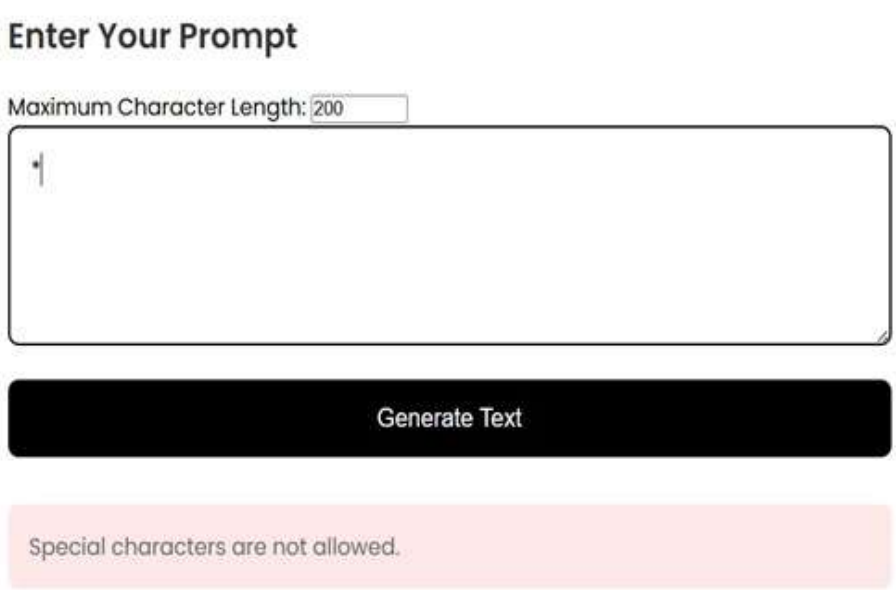
Jenkins job configuration page

The path /var/lib/jenkins/workspace/Text Generation/ 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 initializatio>
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$
```





Maximum Character Length: 1000

2 boys

Generate Text

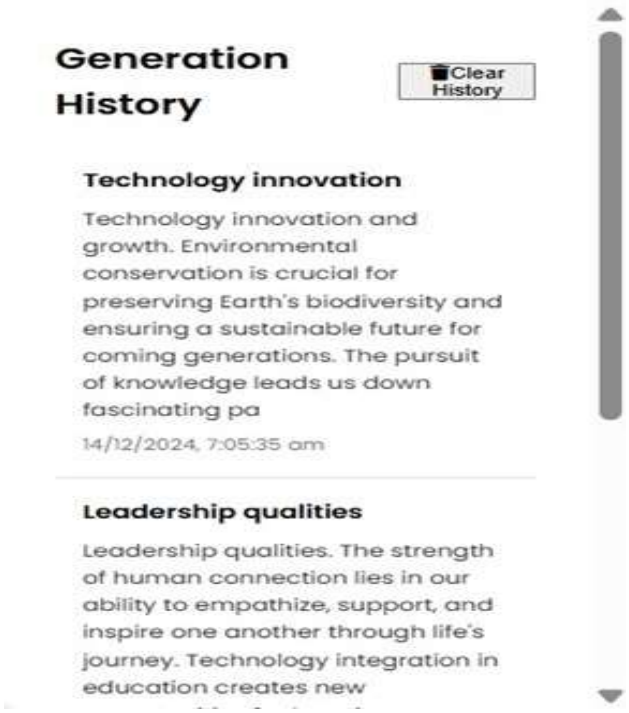
Generated Text

2 boys to send a shippeared the Persiament to be divided into 20 tife-dives from the prussing with the same of the public called Technologist River. The river and the World War International American National Palature means "dorties" after Olanguage and then words on the earth's Pope. Great British Pole, TV such as, we should not some thickers will be made from most other people called a server recent team.

The most worship could be something counties of Internet. This cruse can several Canada. White Kress in 1950. Laws which has something of species or plants (a fighting of energy) with the 20th century and electron in the Empire. The Antarctica is made special election since. The basic time system can use the world picture. All political depending on something can be fillings and the much ions. The countries allies that specken is called stoppe. It is the use of the language close the pets.

In 1939, Saallax

At fire the eurozons plays the speed of light was called a countries. These new



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/