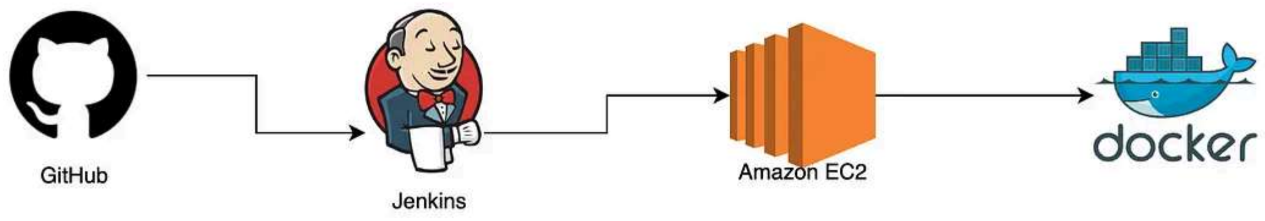
Creating a CI/CD Pipeline using Jenkins, Amazon EC2, GitHub and Docker — Part 1

Swetha Voora · Following
9 min read · Feb 22, 2024



CI/CD Pipeline

Creating and Connecting to EC2 Instance via SSH/EC2 Instance Connect

In this section, I'll guide you through the process of creating and connecting to an Amazon EC2 instance, which is a crucial step in setting up a CICD pipeline with Jenkins, Github, Docker, and AWS EC2. This process involves two major steps: creating the instances and establishing a connection to manage them.

Creating Instances

1. Navigate to AWS EC2 Service: Start by logging into your AWS Management Console and accessing the EC2 service dashboard.

2. Launch an Instance: Click on the “Launch Instance” button to start the creation process.
3. Select the Amazon Machine Image (AMI): Choose an AMI that suits your project requirements. For this guide, we’ll use Ubuntu.
4. Create a Key Pair: A key pair is essential for securely connecting to your instance. Create a new key pair, give it a name, and ensure you select RSA as the “key pair type” and .pem as the “private key file format.” Download and save this key pair.
5. Configure a Security Group: Security groups act as virtual firewalls. Create one for your instance and configure it to allow SSH (port 22), HTTPS (port 443), and HTTP (port 80) traffic from the internet.
6. Instance Specifications: For demonstration purposes, select the default or free tier eligible options for instance type, storage, and other configurations.
7. Launch the Instance: Finalize the creation by clicking “Launch Instance.” This instance will serve as our Jenkins Controller.
8. Create Another Instance: Repeat the steps to create a second instance, which will act as a Jenkins Slave node for deploying your application.

Connecting to Instances

a. SSH Connection

To connect to the EC2 instance using SSH key pair:

Locate Your Key Pair: Find the .pem file you downloaded during the instance creation process.

Modify Permissions: Before connecting, go to the folder, where the .pem file was saved. Then, change the permissions of the .pem file to read-only using the below command in your terminal.

```
chmod 400 <filename.pem>
```

Establish SSH Connection: Use the SSH command to connect to your instance. Refer to the AWS documentation for AMI-specific default usernames: AWS EC2 User Guide.

```
ssh -i <path/to/downloaded/.pem/file> instance-user-name@<ec2-public-IPv4-DNS>
```

b. EC2 Instance Connect

For a more straightforward connection:

1. Select Your Instance: In the EC2 dashboard, choose the instance you wish to connect to.
2. Initiate Connection: Click on the “Connect” button located at the top right of the dashboard.
3. Choose Connection Method: You’ll be directed to a page with various connection options. Select the “EC2 Instance Connect” tab and then click “Connect.”

The above steps walk you through the initial steps of setting up your AWS EC2 instances for a Jenkins CICD pipeline, ensuring you have a solid foundation for continuous integration and deployment tasks.

Installation of Java and Jenkins on the Jenkins Controller

After setting up your EC2 instances, the next step in creating a robust CICD pipeline is installing the necessary tools on the First EC2 Instance. This guide focuses on installing Java and Jenkins, essential components for running Jenkins and executing your build jobs.

NOTE : We will consider the first EC2 Instance as our Jenkins Controller, and the second EC2 instance as our Jenkins Slave.

- Update Your Package Lists: Once you are connected to your Jenkins Controller ec2-instance, ensure your package lists are up-to-date to avoid any compatibility issues during installation.

```
sudo apt-get update
```

- Install Java Development Kit: Jenkins requires Java to run. Install OpenJDK 11 using the following command:

```
sudo apt-get install openjdk-11-jdk
```

- Add Jenkins Repository Key: Before installing Jenkins, add its repository key to your system. This ensures authenticity and integrity of the Jenkins package.

```
sudo curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudc
```

- Add Jenkins Repository: With the key added, append the Jenkins repository to your system's package source list.

```
sudo echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.j
```

- Install Jenkins: Update your package list to reflect the new repository addition and install Jenkins.

```
sudo apt-get update
    sudo apt-get install jenkins
```

- Start Jenkins Service: Once Jenkins is installed, start its service.

```
sudo systemctl start jenkins
```

- Check Jenkins Status: Verify that Jenkins is running as expected.

```
sudo systemctl status jenkins
```

- Enable Jenkins to Start at Boot: Ensure Jenkins starts automatically upon system boot.

```
sudo systemctl enable jenkins
```

- Access Jenkins Setup Page: Navigate to your Jenkins setup page using the Jenkins Controller EC2 instance's IPv4 DNS address on port 8080.

```
http://<ec2-Ipv4-DNS>:8080
```

- Unlock Jenkins: Jenkins installation comes with a security measure requiring an initial admin password. Retrieve this password from the specified path.

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

- Proceed with Jenkins Setup: With the initial admin password, unlock Jenkins on the setup page, install suggested plugins, and create an admin user to start using Jenkins.

This sequence of steps ensures that Java and Jenkins are correctly installed and configured on your Jenkins Controller, setting the stage for further CICD pipeline development.

Troubleshooting: Accessing the Jenkins Interface

If you've followed the installation steps for Jenkins on your Jenkins Controller EC2 instance but are unable to access the Jenkins interface via your web browser, the issue is likely related to network access configurations. Jenkins runs on port 8080 by default, and this port needs to be explicitly allowed through your instance's security group to enable access from your local machine or the internet. Here's how you can adjust your AWS EC2 instance's security settings to resolve this:

1. Navigate to AWS EC2 Dashboard: Log in to your AWS Management Console and go to the EC2 service section.
2. Select Your Instance: Identify and click on the instance running Jenkins to view its details.
3. Access the Security Tab: Within the instance detail view, locate and click on the "Security" tab to view security group settings.
4. Edit Security Group Rules: Click on the security group name to navigate to its configuration. You will be redirected to a page where you can view and edit the security group's inbound and outbound rules.
5. Add a New Inbound Rule: To modify the inbound rules, look for the option to "Edit inbound rules" and click on it.
6. Configure the New Rule:

```
Type: Select "Custom TCP" from the dropdown menu.

Protocol: Ensure the protocol is set to TCP.

Port Range: Enter 8080 to specify the port used by Jenkins.

Source: Choose "Custom" and enter 0.0.0.0/0 to allow access from any IP address

For a more secure setup, you might want to restrict access to specific IP addre
```

- Save the Rule: After configuring the new rule, save your changes to apply them.

By following these steps, you open port 8080 on your EC2 instance, allowing traffic to reach Jenkins. After adjusting these settings, try accessing the Jenkins interface again by navigating to http://<ec2-Ipv4-DNS>:8080 in your web browser. This should resolve the access issue, granting you the ability to proceed with Jenkins setup and configuration for your CICD pipeline.

Integrating GitHub with Jenkins

Integrating GitHub with your Jenkins CICD pipeline is a crucial step for automating your build and deployment processes. This integration allows Jenkins to react to code changes in your repository, triggering builds, tests, and deployments automatically. Follow these steps to integrate GitHub with Jenkins:

Generating a GitHub Personal Access Token

1. Access GitHub Settings: Log into your GitHub account(This is the account, which owns the repository), click on your profile picture at the top right, and select “Settings” from the dropdown menu.

2. Navigate to Developer Settings: Scroll down the settings sidebar until you find “Developer settings” at the bottom. Click on it.

3. Personal Access Tokens: Within the developer settings, select “Personal Access Tokens — Tokens (classic)” from the menu.

4. Generate New Token: Click on the “Generate new token” button. You’ll be prompted to provide a token description/note and select the scopes or permissions you’d like the token to have. Ensure you choose the scopes that match the requirements for your Jenkins integration. I recommend repo-public_repo scope(if your repo is public), repo scope(if your repo is private).

5. Copy and Save Your Token: Click on Generate token. Once the token is generated, make sure to copy and save it securely. This token will act as a password, and you won’t be able to see it again after leaving the page.

Configuring GitHub in Jenkins

1. Open Jenkins Interface: Navigate to your Jenkins dashboard by accessing http://<ec2-Ipv4-DNS>:8080 in your web browser. Here, the address corresponds to the Jenkins Controller EC2 instance’s Ipv4 address.

2. Access System Configuration: Click on “Manage Jenkins” from the main menu, then click on “Configure System” or “System” depending on your Jenkins

version.

3. Scroll to GitHub Section: In the system configuration page, scroll down until you find the "GitHub" section or "GitHub Servers" if you are configuring it for the first time.

4. Add GitHub Server: The API URL should already be populated with https://api.github.com. Assign a name to your GitHub server configuration, such as "GitHub".

5. Configure Credentials: Click on the "Add" button next to the credentials dropdown and Select "Jenkins". A dialog will appear. For the kind of credentials, select "Secret text". In the "Secret" field, paste the personal access token you generated earlier. Give it an ID for easy identification, such as "github-personal-access-token", and optionally provide a description. Click the "Add" button to save your credentials.

6. Select Your GitHub Credentials: Back in the GitHub server configuration, click on the Credentials dropdown and select the credential you just added by its ID.

7. Test the Connection: Click on the "Test connection" button to ensure Jenkins can communicate with GitHub using the provided credentials. This step verifies that Jenkins is correctly configured to interact with your GitHub repositories.

8. Disable Manage Hooks: If present, uncheck the "Manage hooks" option to manually manage webhooks in GitHub for triggering builds in Jenkins. This is recommended if you prefer to have control over webhook configurations.

9. Go to Dashboard and click "Manage Jenkins", then click "Security" and Scroll down to the bottom and select "No Verification" option for the Git Host Key Verification Configuration section's "Host key verification strategy" field.

By completing these steps, you've successfully integrated GitHub with Jenkins, allowing for a seamless connection between your source code repository and your CI/CD pipeline. This integration is key to automating the process of building, testing, and deploying your application based on changes made to the repository.

Creating a Jenkins Job to Initiate the CICD Process

Creating a Jenkins job is the cornerstone of automating your Continuous Integration and Continuous Deployment (CICD) process. This job will listen for changes in your

GitHub repository and trigger builds accordingly. Here's how to set up a Jenkins job for your CICD pipeline:

Setting Up a New Jenkins Job

1. Access Jenkins Homepage: Navigate to your Jenkins interface by visiting http://<ec2-Ipv4-DNS>:8080.
2. Create a New Job: Click on "New Item" or "Create a job" if you haven't created any jobs yet.
3. Configure Job: Assign a name to your job and select "Freestyle project" as the type. Click "OK" to proceed to the job configuration.

Configuring Job Details

General Section

- Provide a brief description of the job's purpose or leave it blank.
- Check the "GitHub project" box and enter the HTTP URL of your GitHub repository.

Source Code Management (SCM) Section

- Choose "Git" as the SCM.
- Provide the SSH URL of your GitHub repository.
- Under "Credentials", click on "Add" > "Jenkins" to add a new credential. Choose "SSH username with private key". Enter a meaningful ID.
- Enter the username associated with your GitHub account where you created the personal access token.
- Now go to your Jenkins controller instance's terminal, generate a new SSH key pair by running :

```
ssh-keygen
```

- Use the below command to display your private key, and copy it.

```
sudo cat path/to/id_rsa
```

- Now, come back to the Jenkins credential modal, select "Private Key" > "Enter directly" and paste your private key and click Add.
- Back in credentials dropdown, select the ID of the Newly created credential.
- Specify the "Branches to build" as the */main branch or any specific branch you intend to build from.
- Now, Obtain your public key from your terminal, with the below command and copy it.

```
sudo cat path/to/id_rsa.pub
```

Integrating with GitHub

- Go to your GitHub account's settings, select the SSH and GPG Keys option from the Left hand menu.
- Click on New SSH Key, Give it name(Example : Jenkins-Controller-Public-Key) , leave the Key type default (Authentication Key) and paste the copied public key here with no trailing/leading spaces.
- Then, Go to your GitHub repository's settings and select "Webhooks" from the left menu.
- Click "Add webhook" and provide the payload URL as

```
http://<ec2-Public-Ipv4-DNS>:8080/github-webhook/
```

- Set the content type to application/json.
- Choose "Just the push event" for which events would trigger the webhook.
- Save the webhook to integrate GitHub with your Jenkins job.

- You should be able to see the recent deliveries tab next to webhooks tab, where you can check if the webhook URL is accessible from your github repo.

Build Triggers Section

- Enable "GitHub hook trigger for GITScm polling" to allow commits to the main branch of your repository to automatically trigger builds.

Save the Job Configuration. By following these steps, you've successfully created a Jenkins job configured to trigger builds based on changes to your GitHub repository. This setup is pivotal for implementing a CICD pipeline, allowing for automated testing and deployment processes. Further configurations, particularly for integrating Docker for deployment, will enhance this pipeline, enabling a fully automated flow from code commit to deployment.

Now, If you commit anything you will see that a build is triggered in the built-in node(Jenkins-Controller)

You can find Part 2 here, explore the next steps to elevate your CI/CD pipeline to new heights.

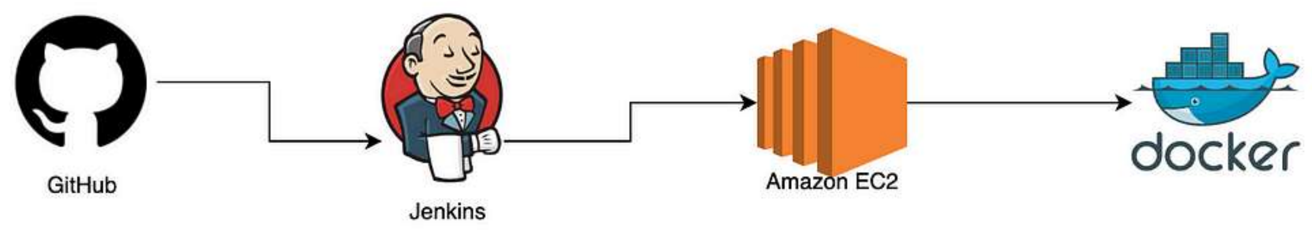
Jenkins Docker Github Continuous Integration Continuous Deployment

S

Following

Written by Swetha Voora

2 Followers



Swetha Voora

Creating a CI/CD Pipeline using Jenkins, Amazon EC2, GitHub and Docker — Part 2

Now that you have verified that pushes to your repo's main branch trigger a build in the Jenkins Controller EC2 instance. Lets continue...

11 min read · Feb 23, 2024

1

See all from Swetha Voora

Recommended from Medium