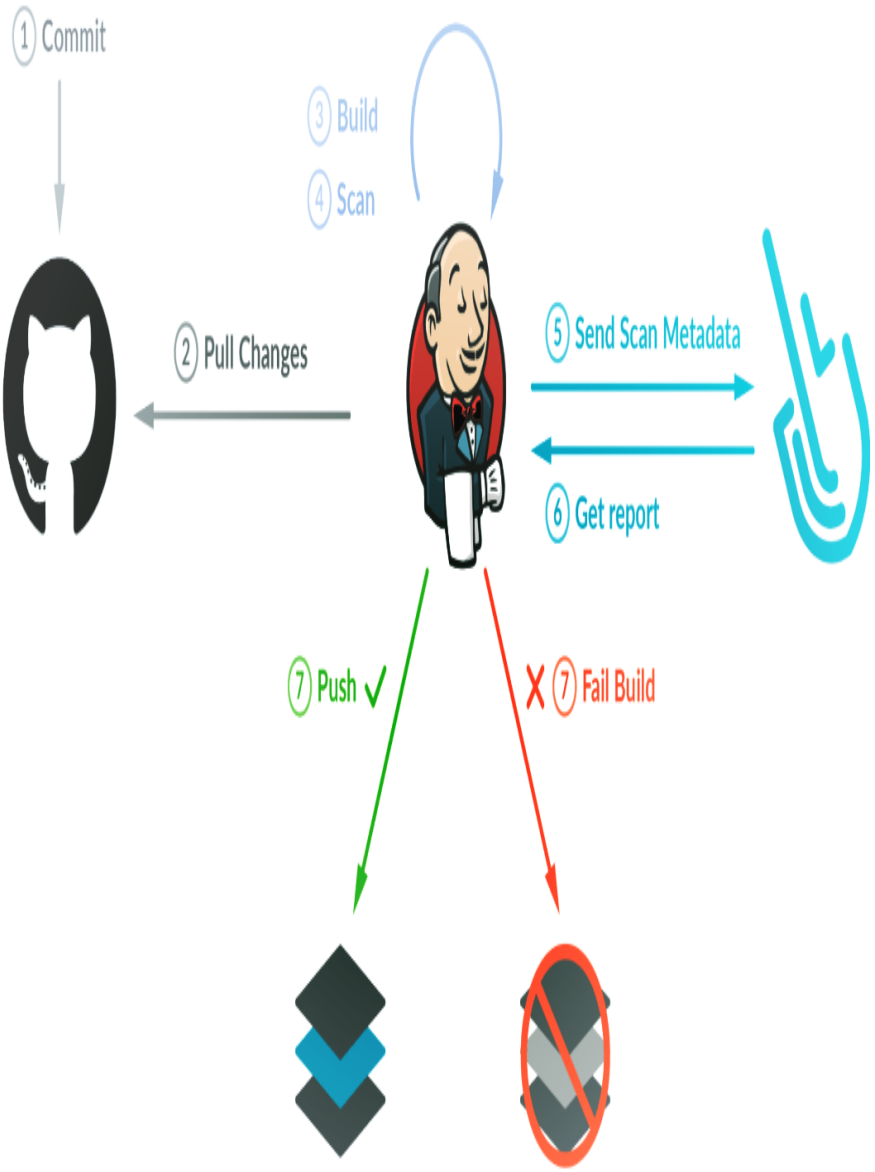
Project

CI Pipelining using Jenkins

Created By : Aditya Kumar Verma

Guided By : Ashish Mittal Sir



Introduction

Jenkins is one of the most widely used open-source CI/CD DevOps tools. It enables developers to implement CI/CD pipelines within the development environment in a comprehensive manner. Jenkins is written in Java and supports various version control tools including Git and Maven. A pipeline in Jenkins is a set of plugins that support continuous delivery (CD) pipelines into Jenkins. A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers.

In Jenkins, a pipeline is defined as a series of steps that are executed in a specific order. These steps can be used to build, test, and deploy applications. You can use Jenkins to create a pipeline for your application by defining the steps that you want to execute. You can also use Jenkins to automate the entire process of building, testing, and deploying your application.

Why Use Jenkins?

Jenkins is a popular choice for CI (Continuous Integration) pipelining due to its versatility, extensive plugin ecosystem, and robust features. Some reasons include:

1. Flexibility and Extensibility: Jenkins provides a highly flexible and extensible framework that allows you to build custom CI/CD pipelines tailored to your specific needs. It supports a wide range of plugins for integrating with various tools, technologies, and platforms, allowing you to create complex workflows and automate diverse tasks within your pipeline.

2. Easy Integration: Jenkins seamlessly integrates with version control systems like Git, enabling it to monitor repositories for changes and trigger builds automatically. It can also integrate with other development and deployment tools, such as issue trackers, testing frameworks, and container orchestration platforms, providing a unified view of the entire development and delivery process.

3. Scalability: Jenkins is designed to handle projects of all sizes, from small teams to enterprise-scale organizations. It supports distributed builds, enabling you to distribute workloads across multiple agents or nodes, improving performance and scalability. You can also set up master-slave configurations to manage multiple Jenkins instances and delegate tasks effectively.

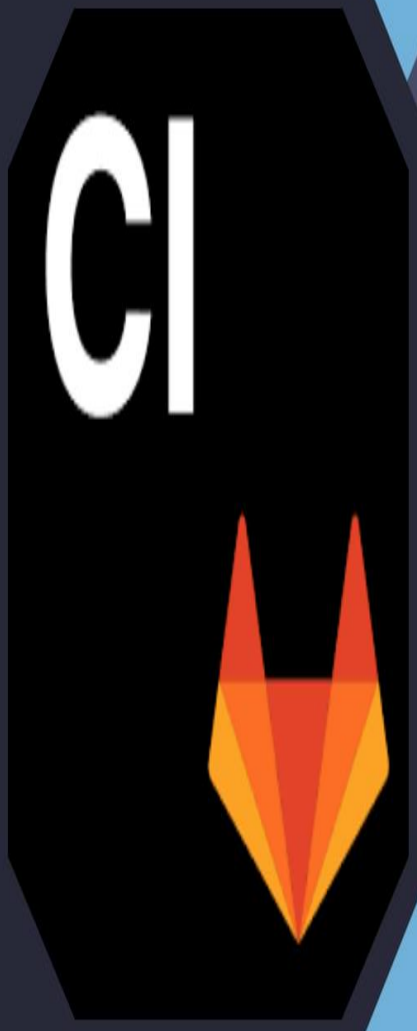
4. Pipeline as Code: Jenkins introduced the concept of "Pipeline as Code" with its Jenkins file, which allows you to define your entire CI/CD pipeline as a text file stored alongside your source code. This approach enables version control, code review, and collaboration on your pipeline code, promoting best practices and making it easier to manage and reproduce pipelines across environments.

5. Extensive Plugin Ecosystem: Jenkins offers a vast collection of plugins that extend its functionality and support integration with various tools and technologies. These plugins cover areas such as source code management, build automation, testing, static code analysis, artifact management, deployment, and more. The plugin ecosystem allows you to customize your pipeline to fit your specific requirements and leverage existing tools seamlessly.

Jenkins Pipelines

Jenkins pipeline is a powerful tool for automating the software delivery process. It provides a way to define and manage the entire build, test, and deployment workflow as a code-based pipeline. With Jenkins pipeline, developers can describe their entire software delivery process in a declarative or scripted manner, allowing for greater flexibility and control. The pipeline can be divided into stages, each representing a specific step in the process, such as building, testing, and deploying the application.

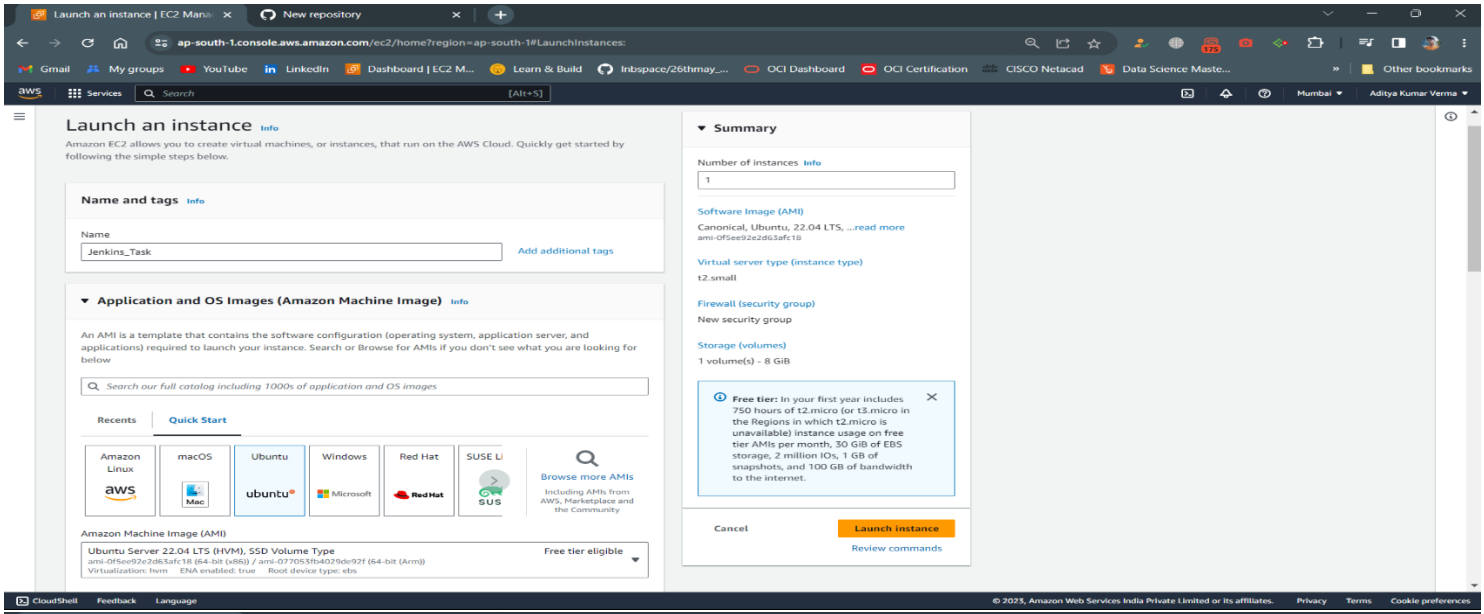
It integrates seamlessly with version control systems and allows for easy collaboration and continuous integration. Overall, Jenkins pipeline simplifies the software delivery process by providing a standardized and automated approach, ensuring faster, reliable, and consistent releases.

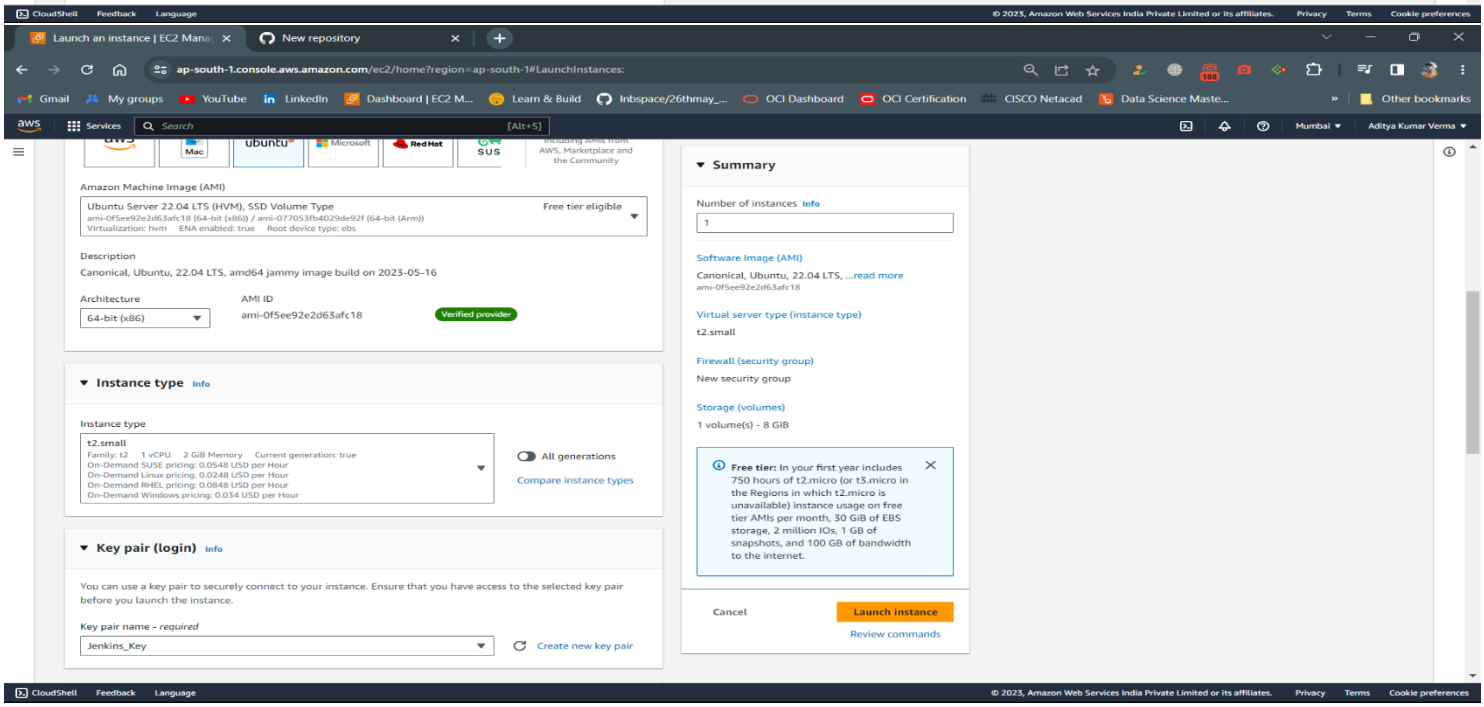


Launching an Instance

- Create a EC2 Instance on AWS and connect it to web server.

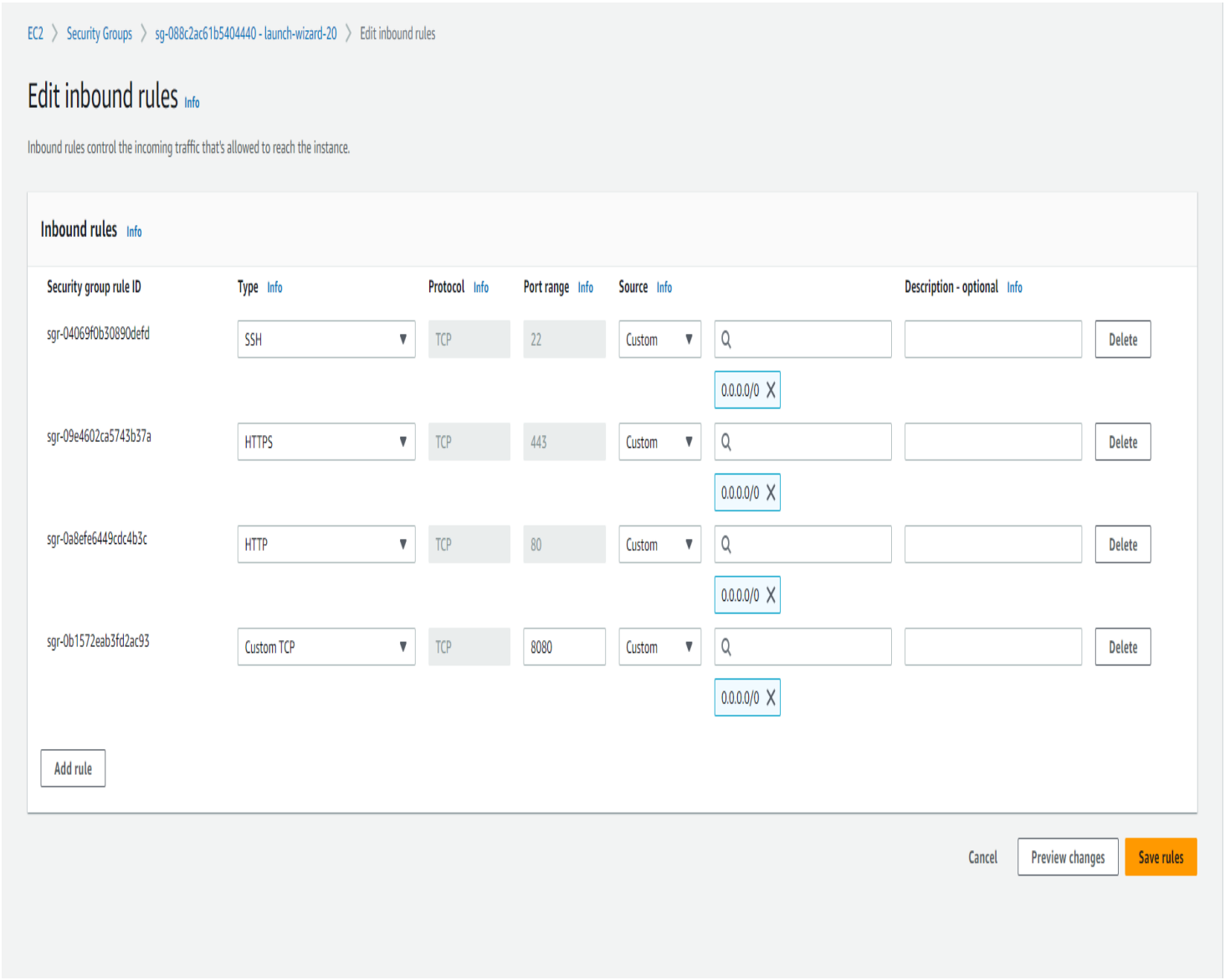
Note : select t2.small or higher in the Instance Type.





Checking the 8080 port

- After launching an EC2 instance, now review the inbound rules under the security groups & add a new inbound rule for 8080 port. This is the most important step after launching the instance.



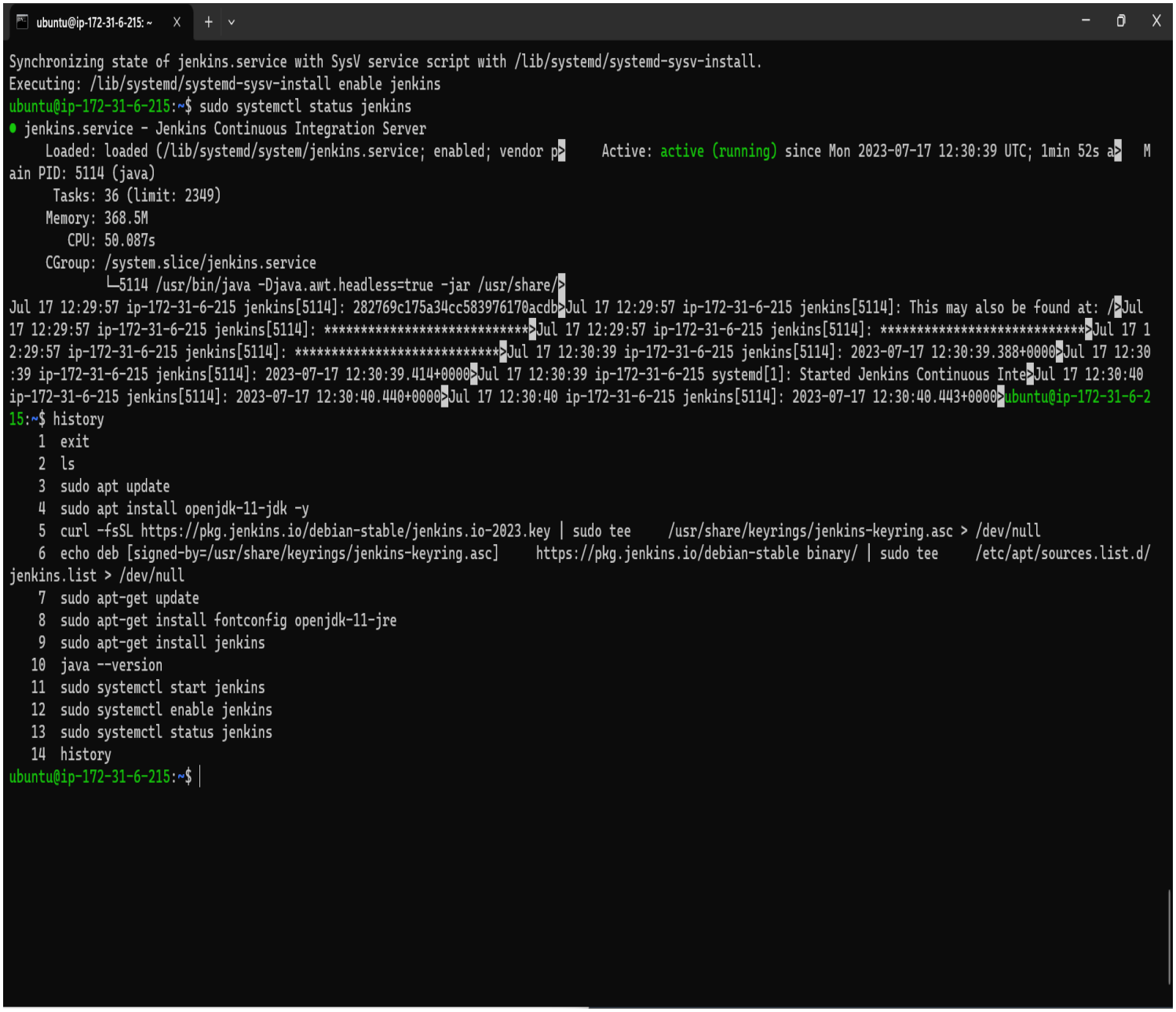
Installation of Java & Jenkins

- After this, connect the instance & using the below mentioned commands, install **Java & Jenkins** on your cmd.

```
15:~$ history
    1  exit
    2  ls
    3  sudo apt update
    4  sudo apt install openjdk-11-jdk -y
    5  curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee     /usr/share/keyrings/jenkins-keyring.asc > /dev/null
    6  echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]     https://pkg.jenkins.io/debian-stable binary/ | sudo tee     /etc/apt/sources.list.d/
jenkins.list > /dev/null
    7  sudo apt-get update
    8  sudo apt-get install fontconfig openjdk-11-jre
    9  sudo apt-get install jenkins
   10  java --version
```

Enabling Jenkins

- After successful installation of Java & Jenkins in the system, it is the time to enable Jenkins for use & checking its running status.

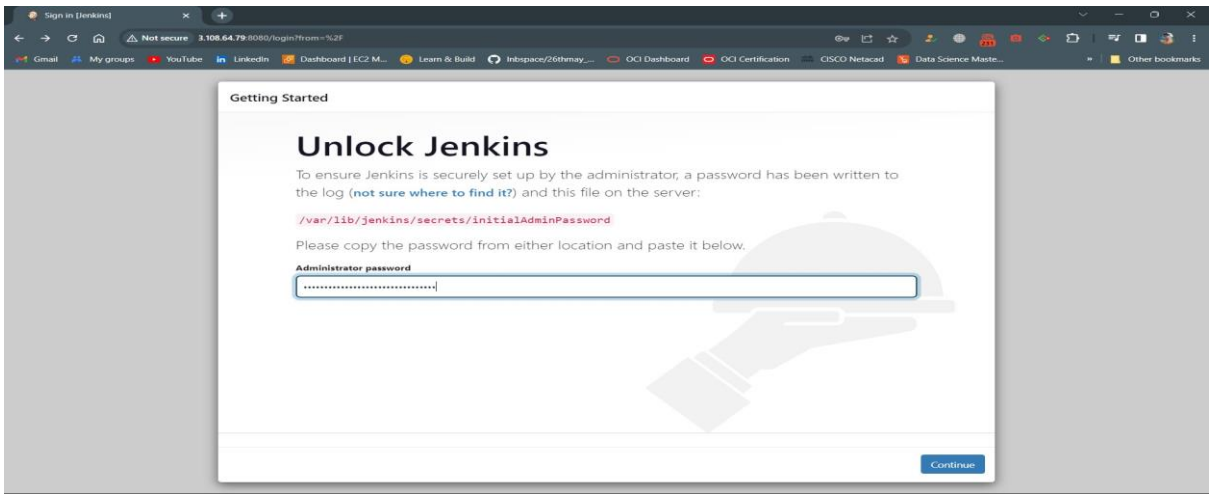


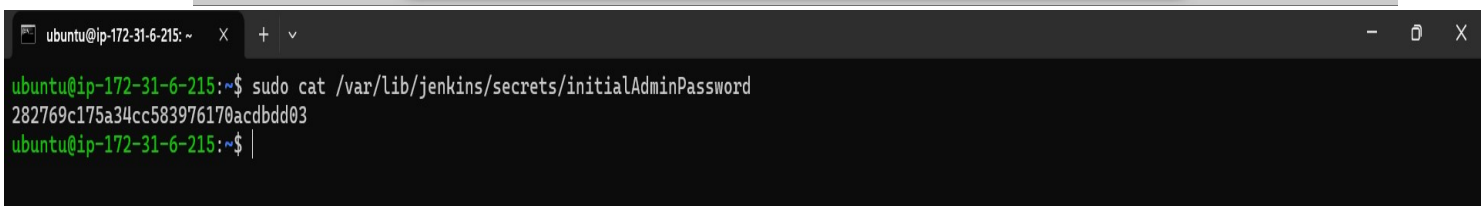
Setting Up Jenkins

- After enabling Jenkins, redirect to a new tab in a browser & search :- <public ip:8080> to open jenkins.

  Note : Copy the public IPV4 address from your built AWS EC2 Instance & use in the place of public ip.

- Run “sudocat var/lib/jenkins/secrets/initialAdminPassword “ in your instance and copy paste the password in the ‘Administrator password’ field and hit Continue. Select “Install Suggested plugIns ” box and wait for successful installation.



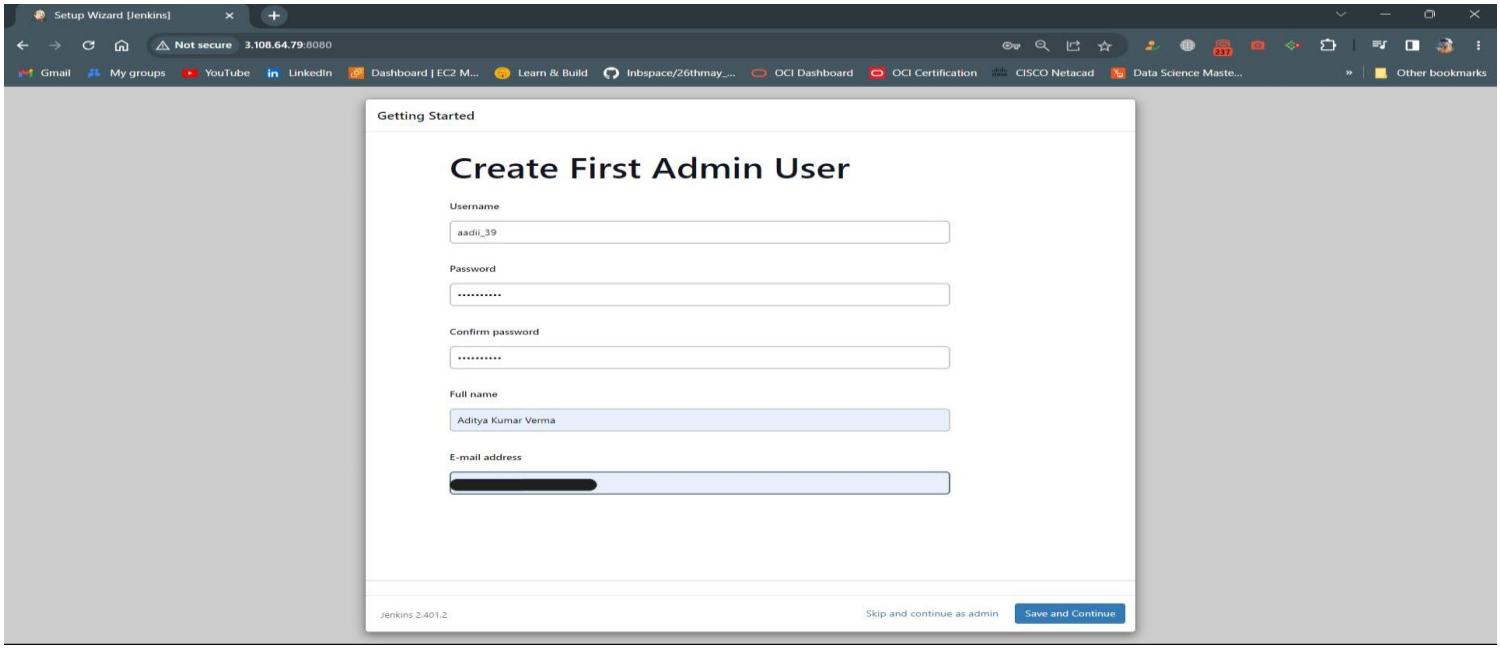


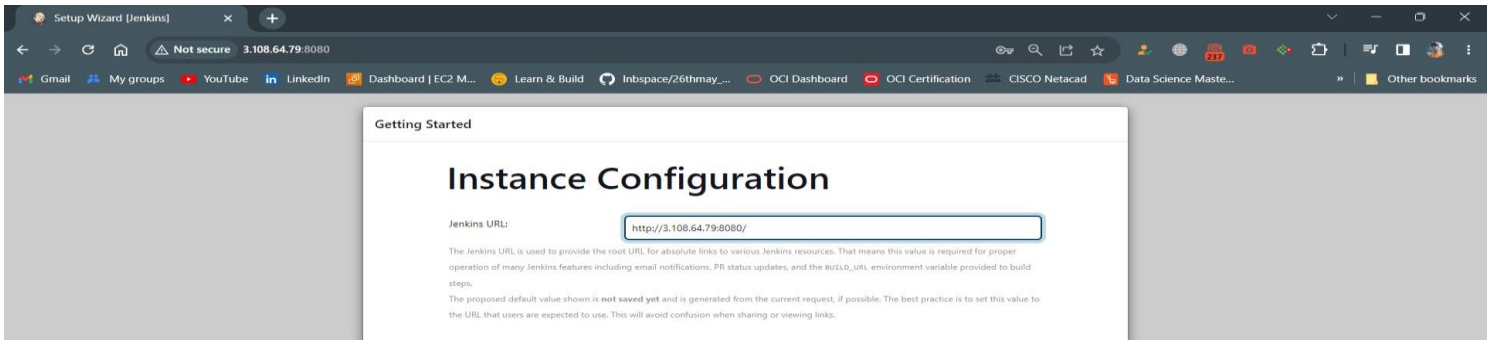
Getting Started with Jenkins

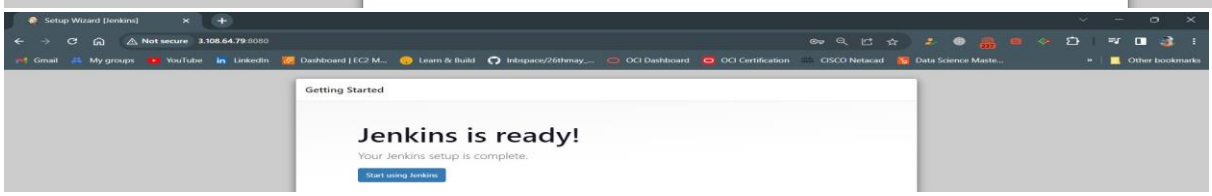
- Now fill up the necessary credentials to create first admin user on Jenkins.

  Note : Save the “Jenkins URL” showing on Instance Configuration page for future use.

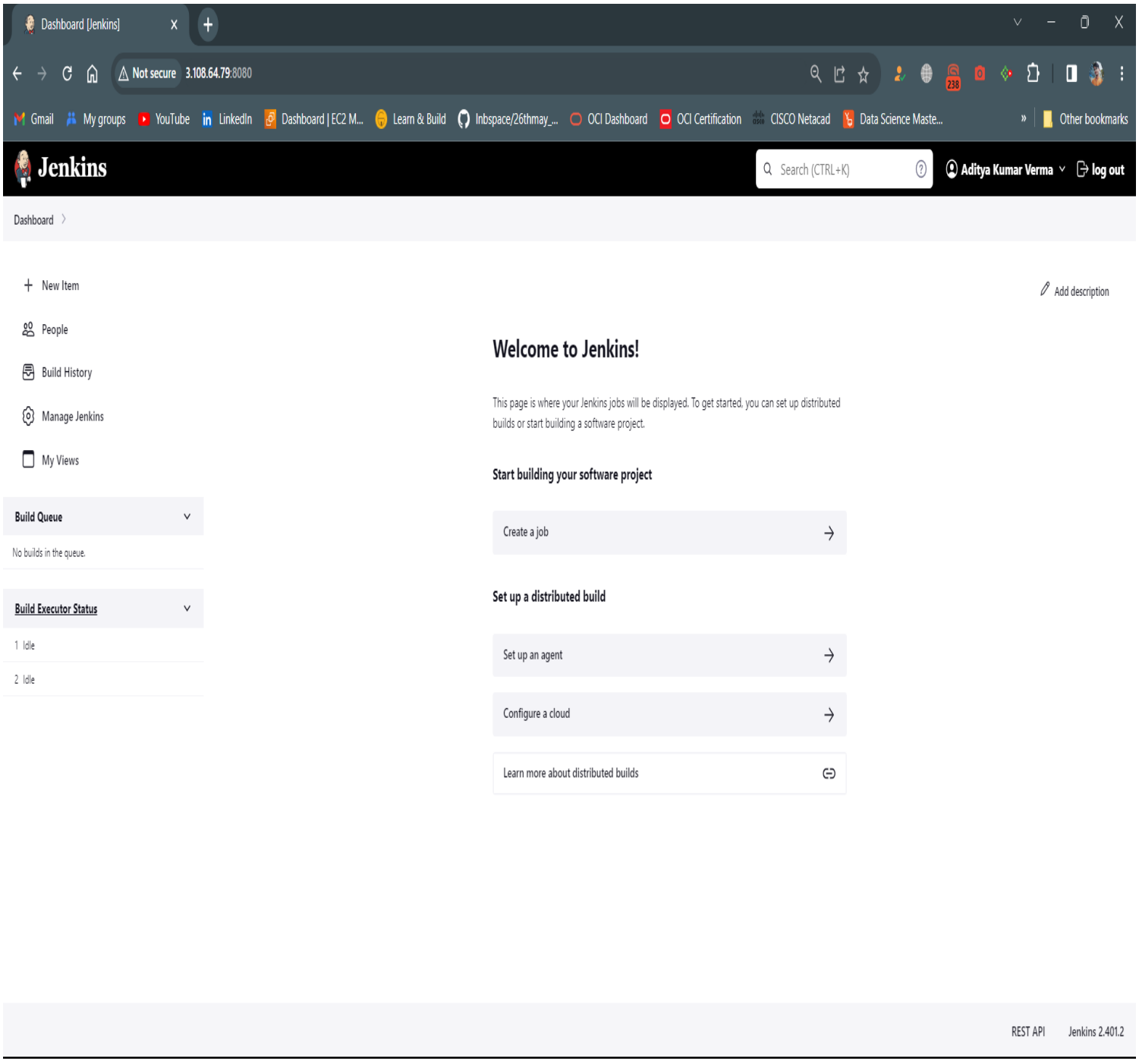
- Complete the Jenkins Setup.





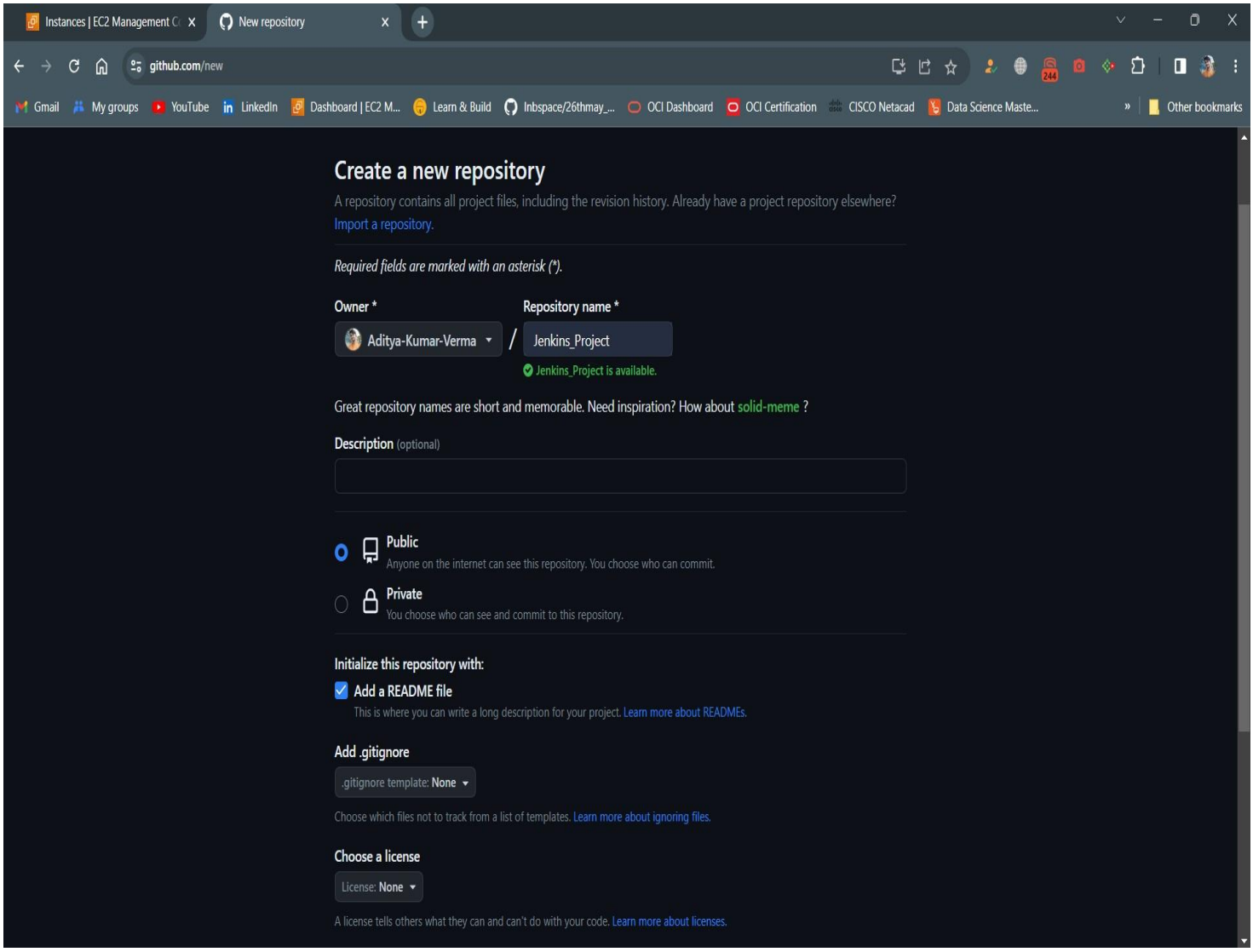


Jenkins Dashboard



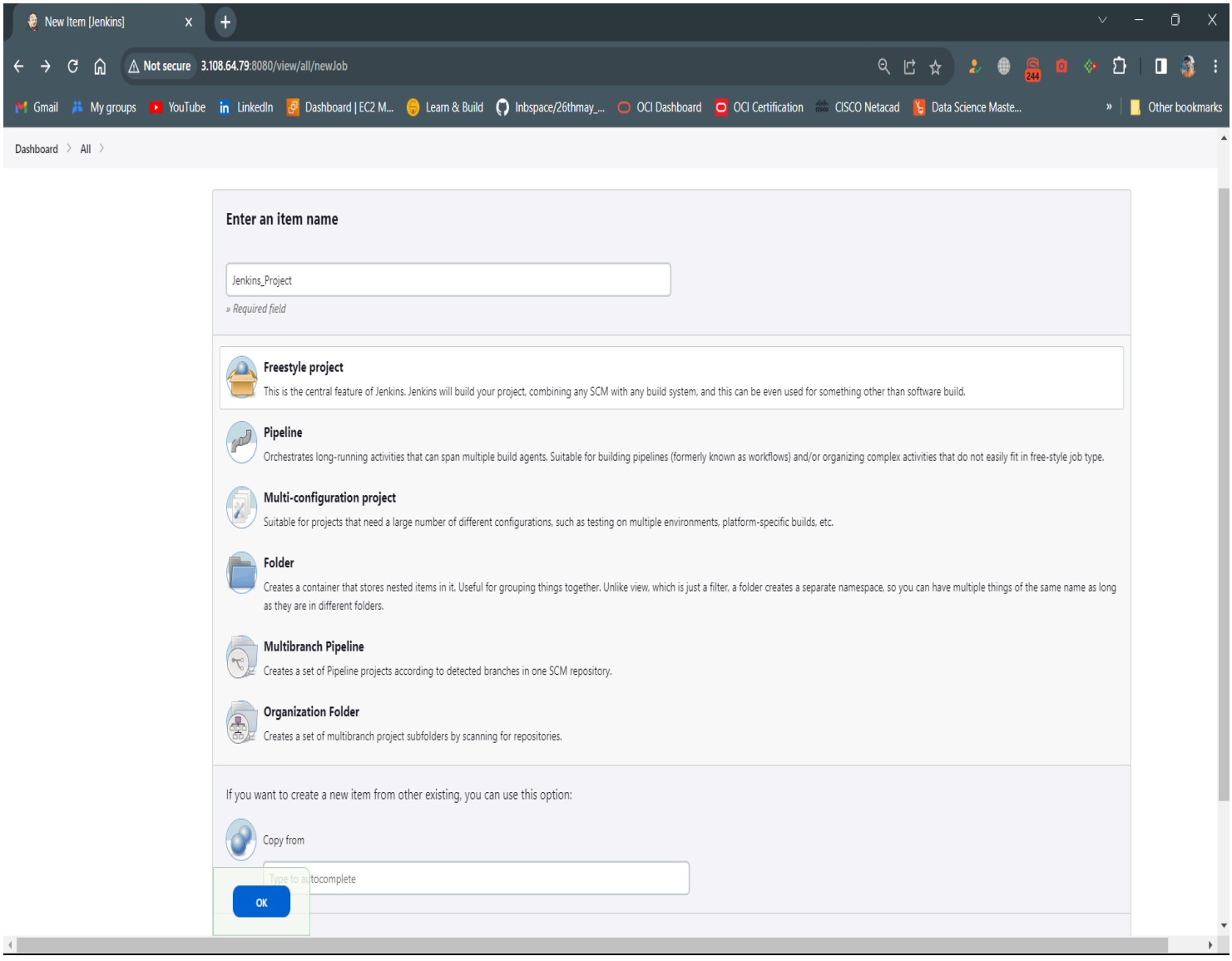
Creating a New Git Repository

- Simultaneously create a new Git repository for the project. And save the repository URL for further use.

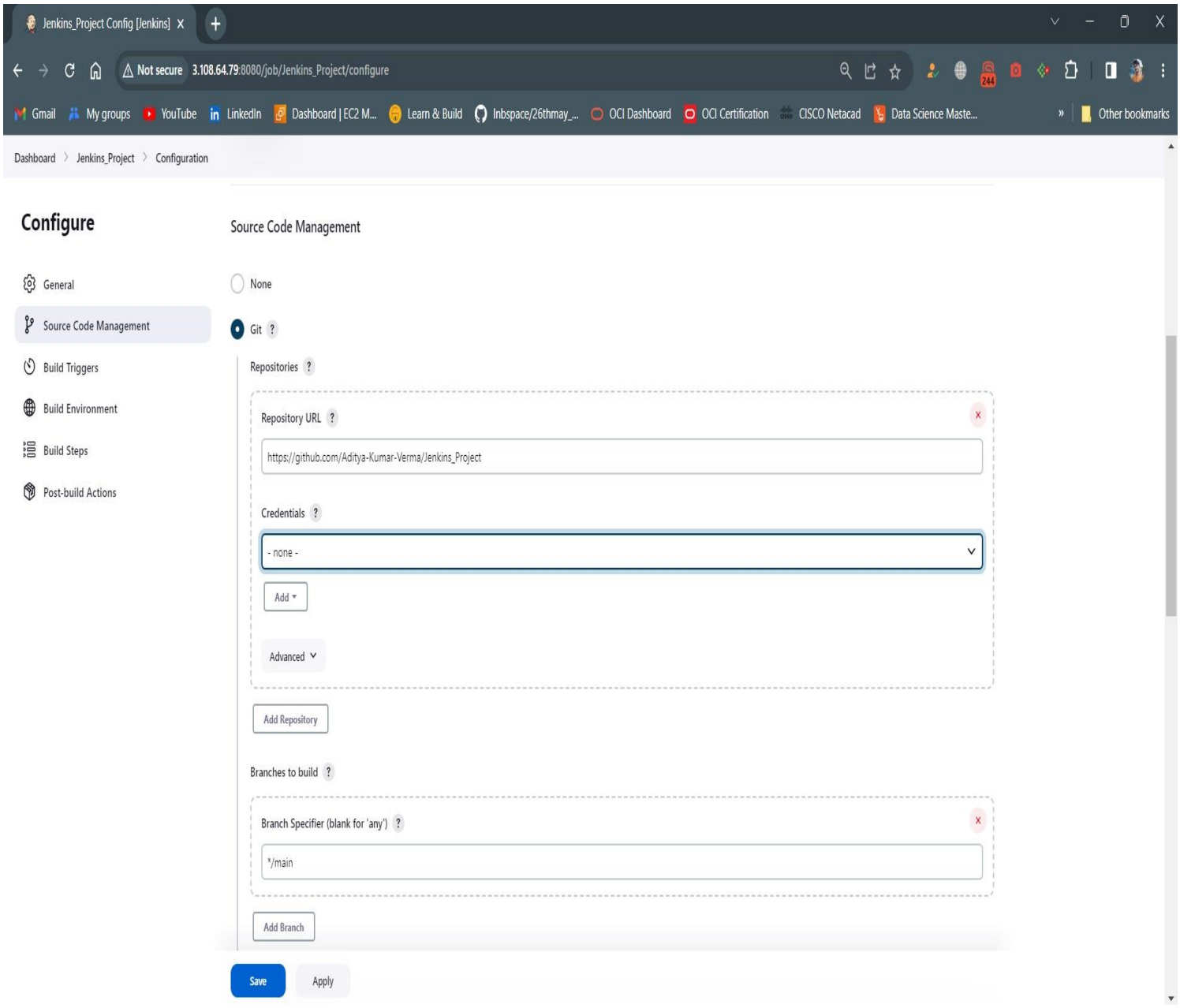


Creating a New project for CI Pipelining

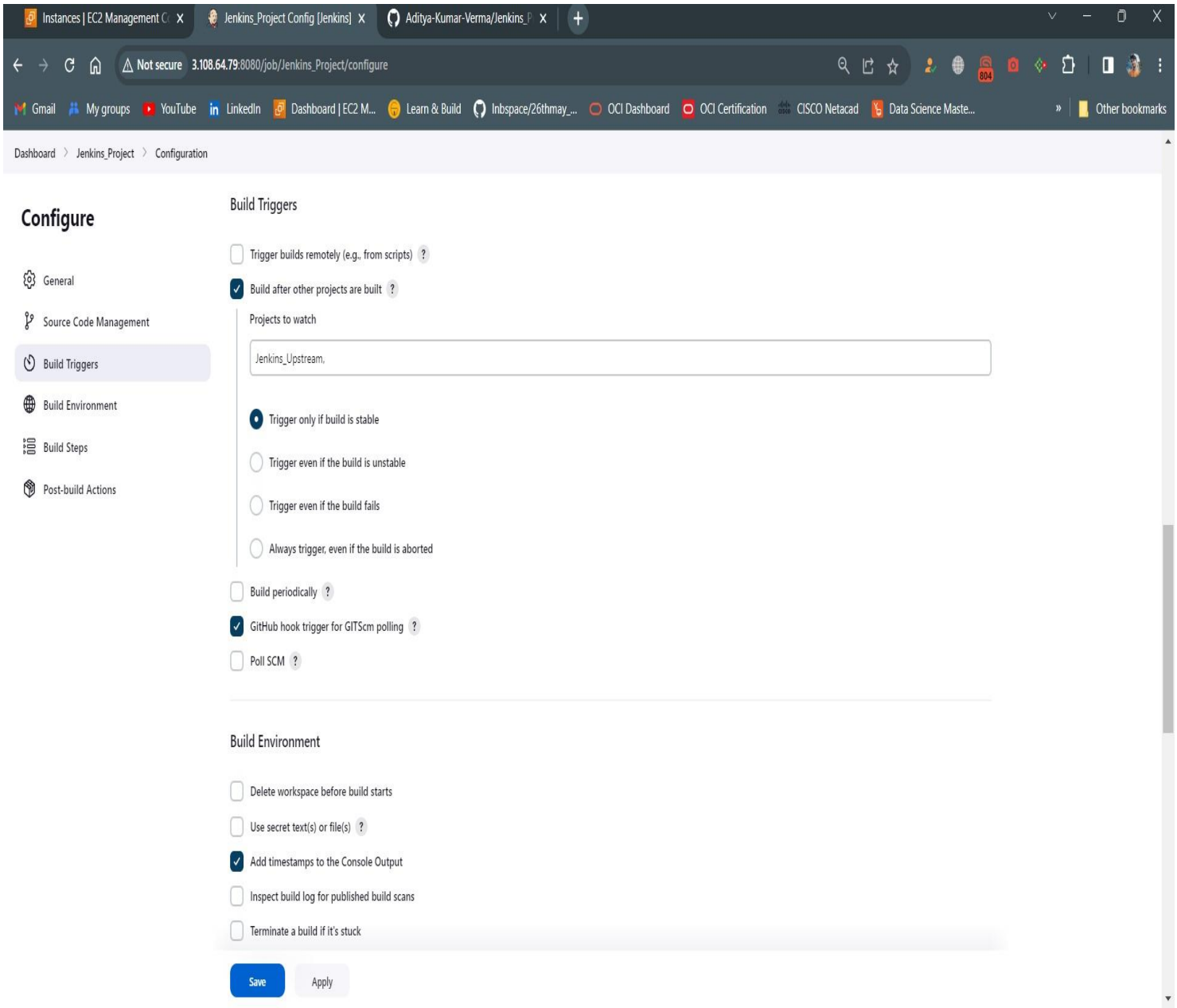
- **Step 1**: Select New item and make a new Build job (Jenkins_Project).



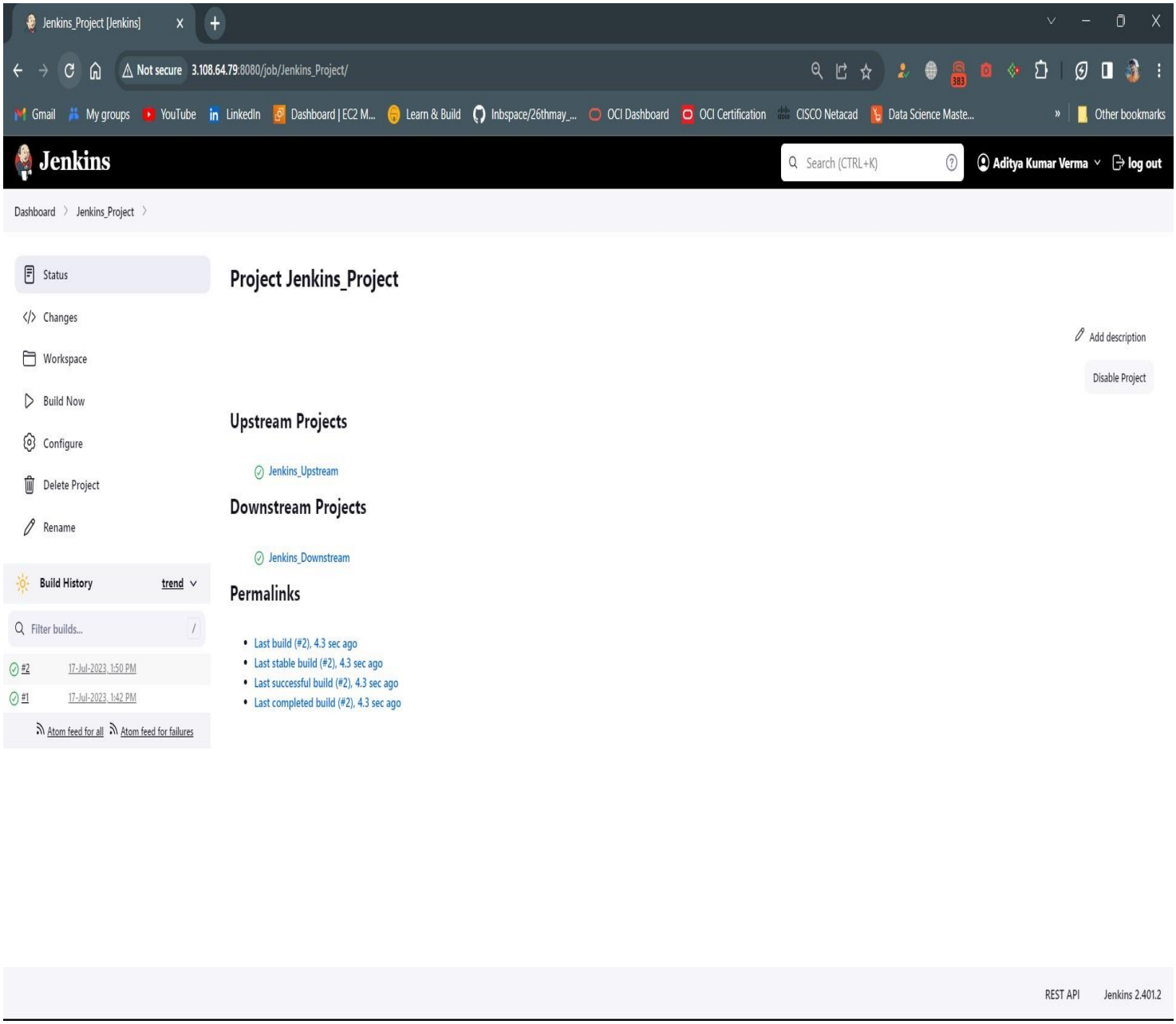
- **<u>Step 2</u>**: Select Git in Source Code Management section. Here, paste the Repository URL of the GitHub (mentioned earlier to save).



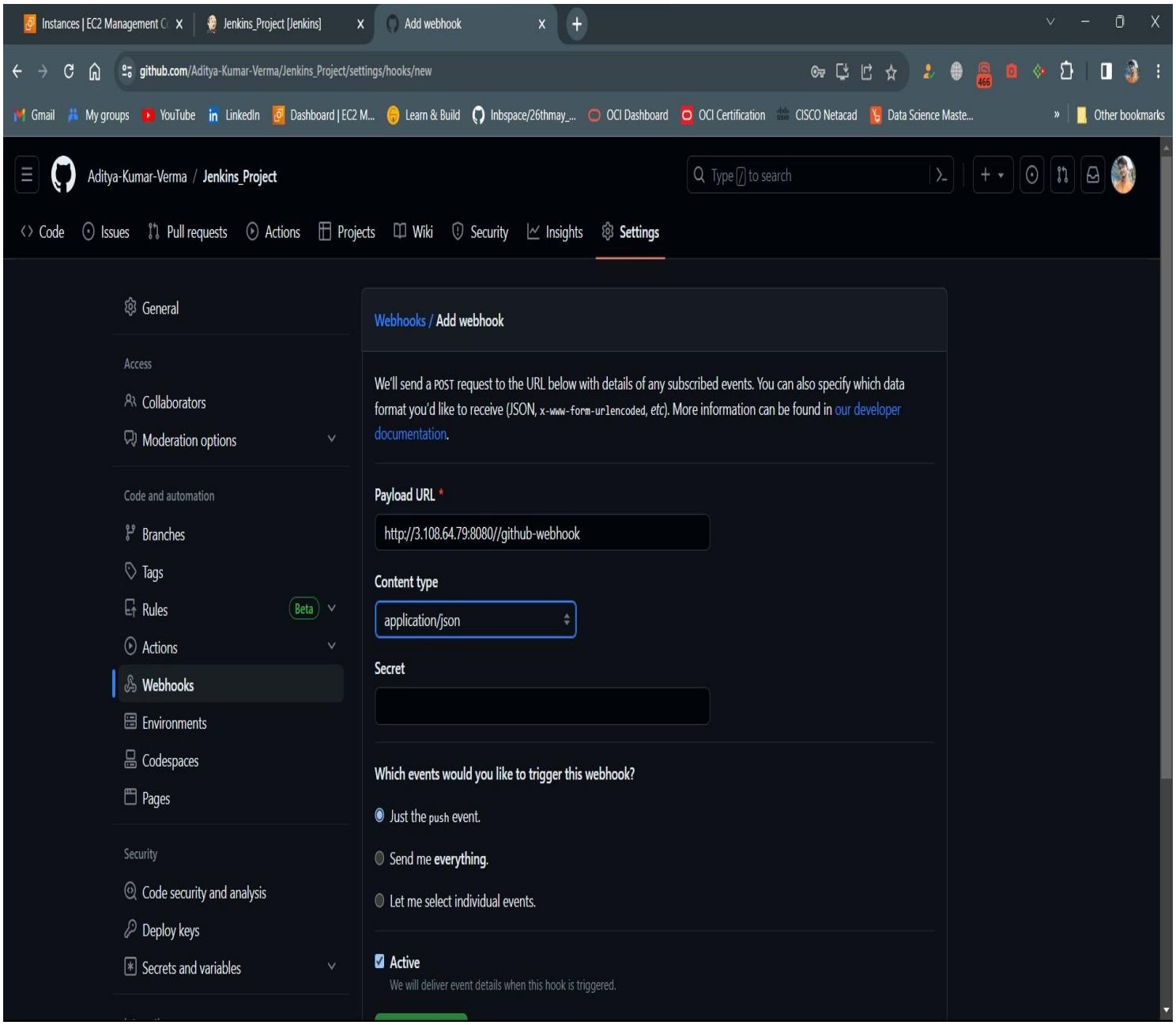
- **<u>Step 3</u>**: Adjust the Build Triggers section. And also mention the Upstream & Down-stream projects.



- **Step 4**: Click Save.



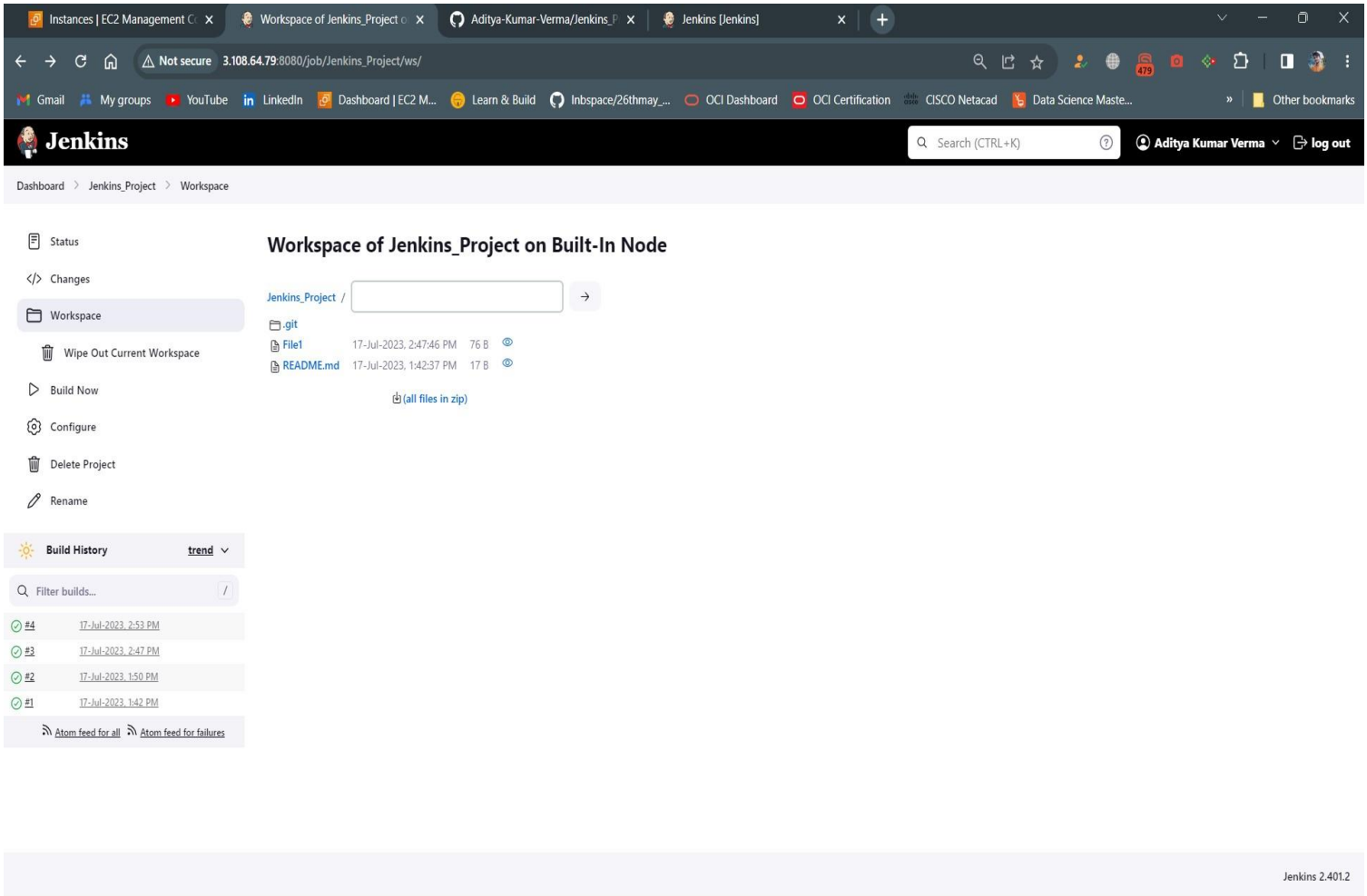
- **<u>Step 5</u>**: Create a new WebHook in GitHub. Paste the Jenkins URL (saved earlier in project) in the Payload URL field and add //github-webhook at the end of URL.



Check the working of Webhook

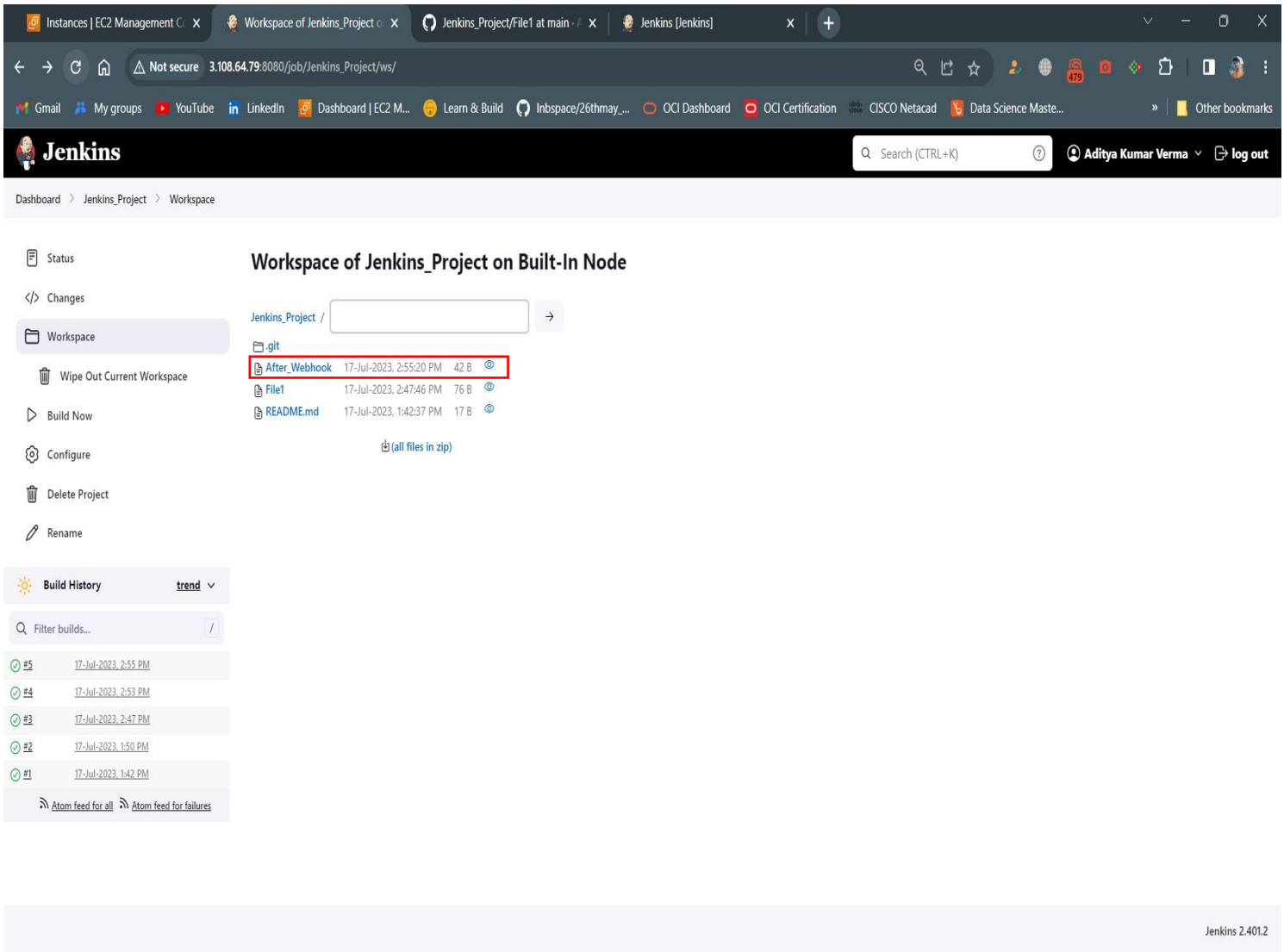
- To check the proper working of webhook, commit a file in github and check it in jenkins workspace.

  (BEFORE COMMITING)



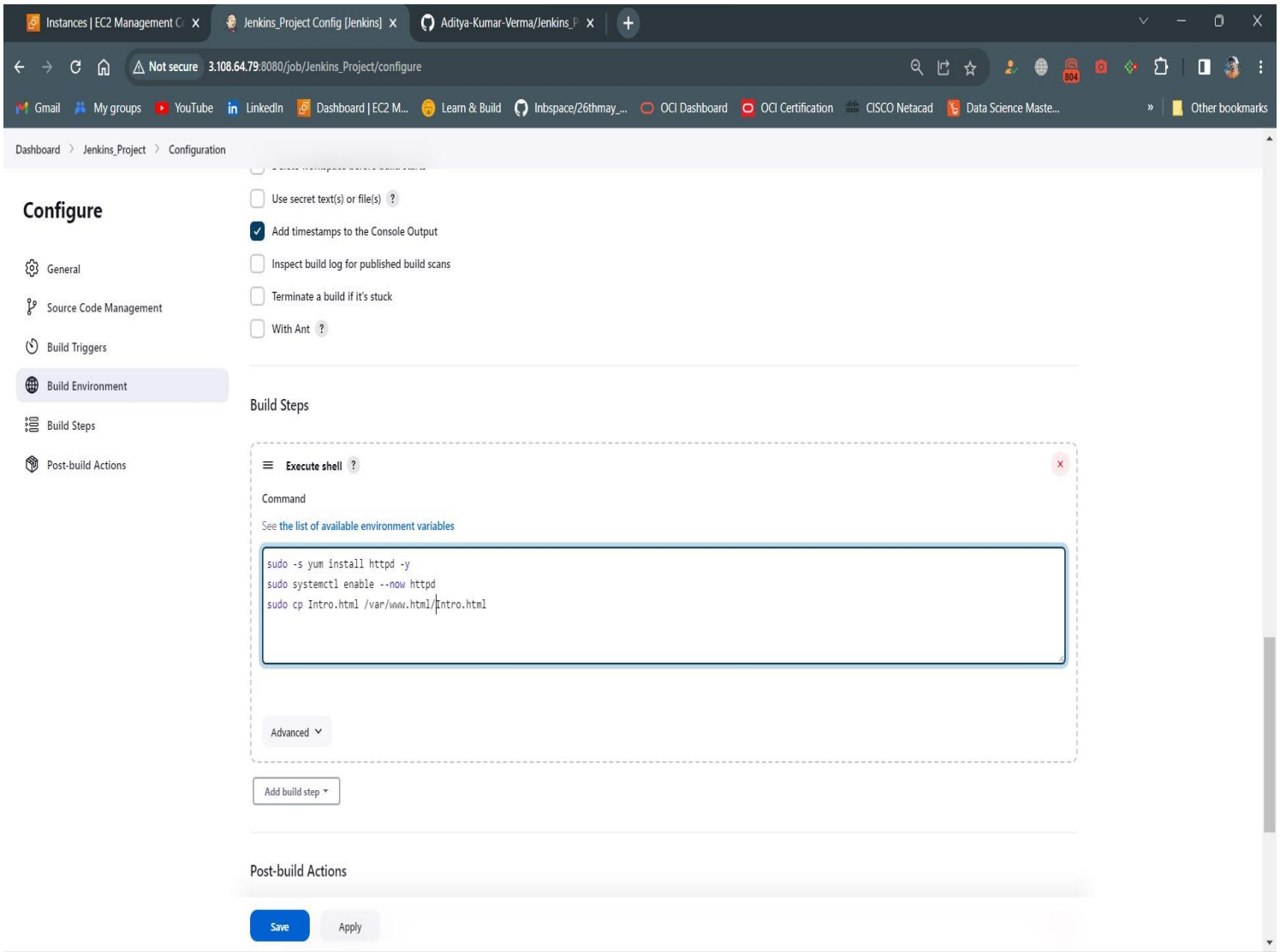
- Click build now in your project and come to workspace and refresh the tab.

  (AFTER COMMITING)



Configuring the Jenkins job

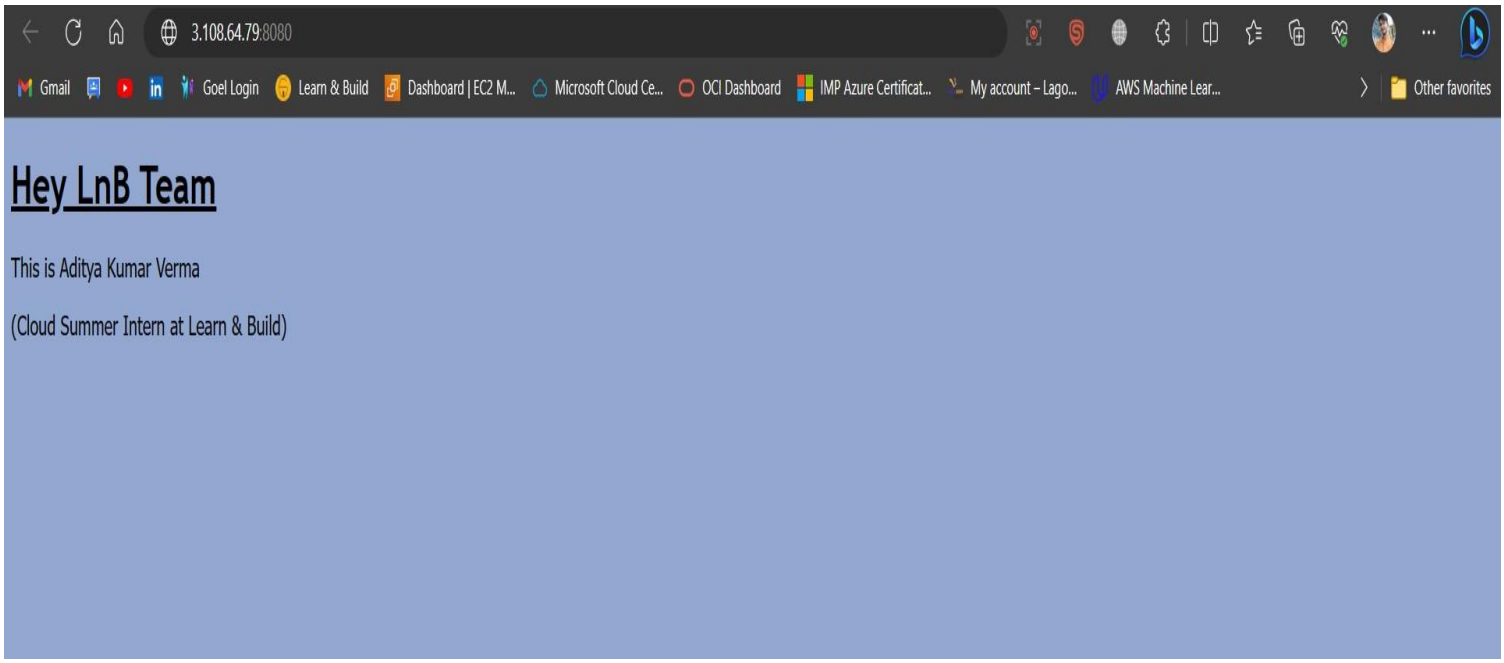
- Head back to the Configure section of the job; in the Execute Shell present in the Build Steps section, enter the following commands.



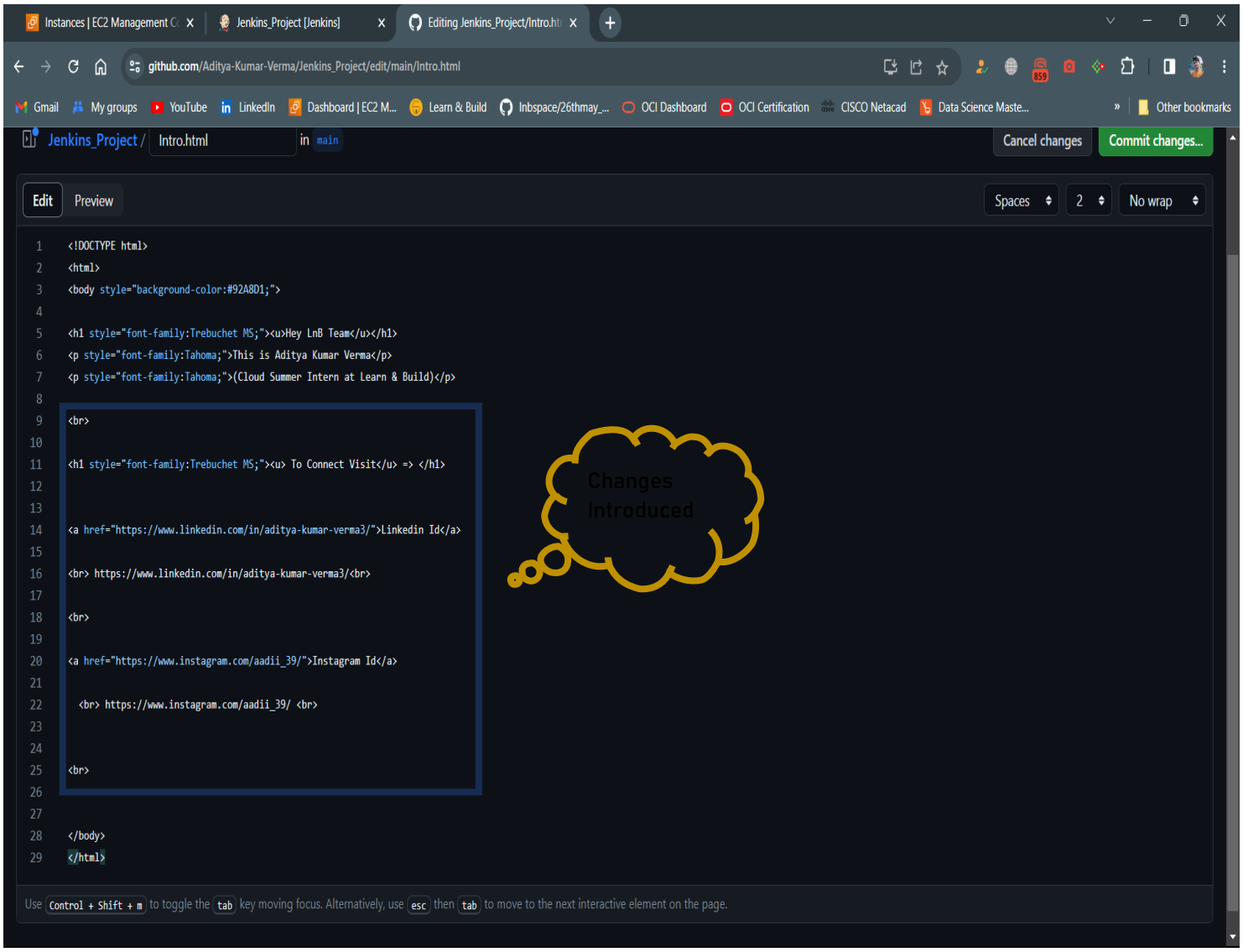
CI Pipelining Demonstration

- **Step 1**: Now open new tab and search for: <public ip:8080> to open your github project file.

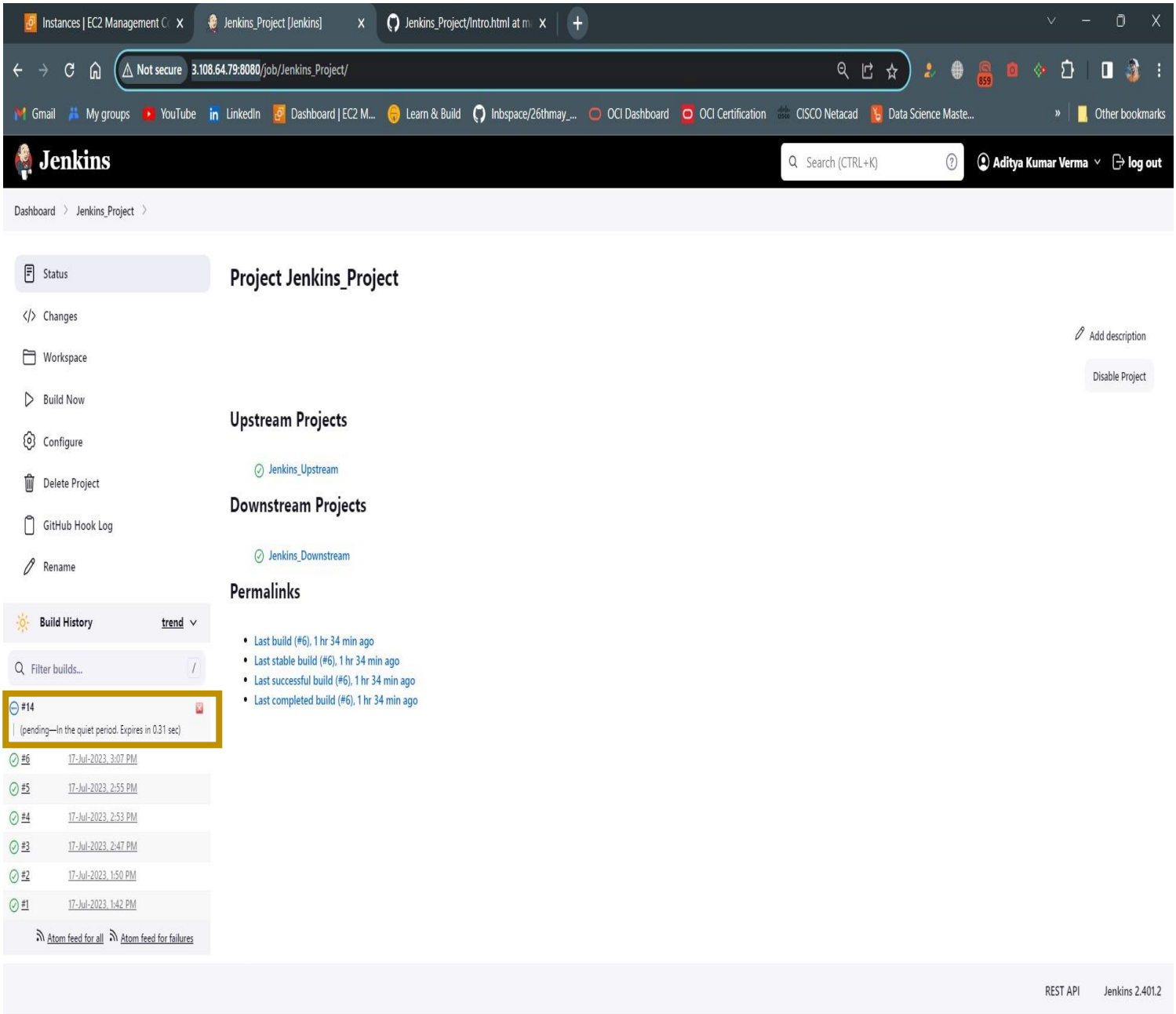
Note : Copy the public IPV4 address from your built AWS EC2 Instance & use in the place of public ip.



- **<u>Step 2</u>**: To check the successful functioning of CI pipelines, edit the github file (here Intro.html) and build the jenkins project again, by clicking on Build Now.

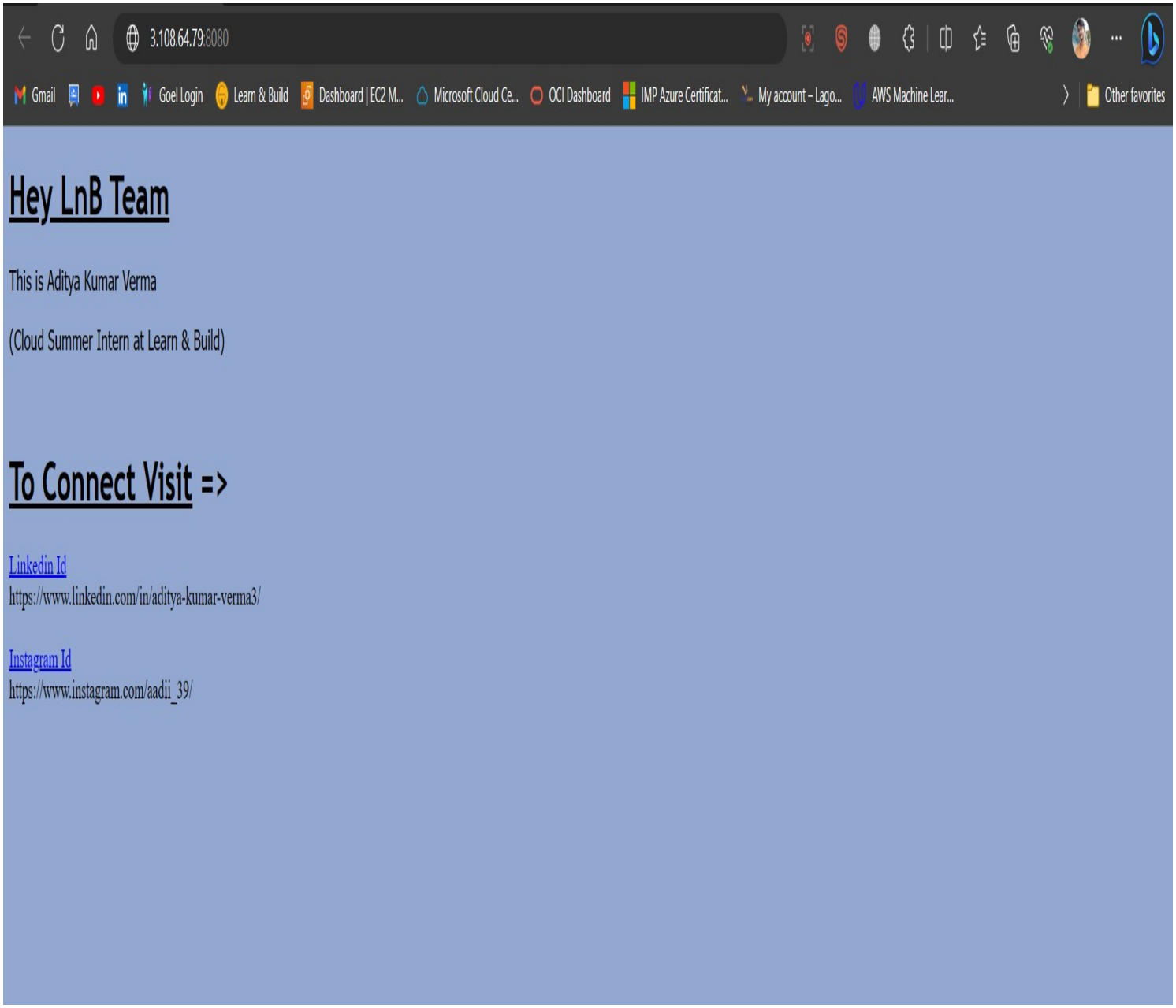


- **<u>Step 3</u>**: Check the Build history which shows the auto integration of the repository.



Step 3: Finally refresh the tab in which the project file has been opened.

IT UPDATES !!!



Conclusion

This shows the demonstrative approach to use Jenkins for CI Pipelining using Github