**CI-CD PIPELINE WITH**

**JENKINS**

**Introduction**

Developing and releasing a software can be a complicated process, especially as application, teams and deployment infrastructure grow in complexity themselves to overcome this complexity we can use continuous integration and continuous delivery (CI-CD) pipelines.

**Scope**

Jenkins is used to build and test your product continuously, so developers can continuously integrate changes into the build. Jenkins is the most popular open-source CI/CD tool on the market today and is used in support of DevOps, alongside other cloud native tools.

**Purpose**

User should not face any problem while accessing a web application designed by the developers. All the changes should be reflected in the application used by the end user for that CI-CD is the best way possible. It is fun to create and there is no need of rewriting the whole code or building the application over and over again each time the developer wants to make changes in the application. He can just use Jenkins for continuous integration and continuous delivery (CI-CD) and sit back and watch the application running with no effort. This is mostly used for application that requires a lot of change each and every day (Like Zomato, any shopping application where there are constant changes in the form of sales and discounts).

**Tools used**

* GitHub
* GitHub is a cloud-based service that hosts a version control system (VCS). It allows developers to collaborate and make changes to shared projects while keeping detailed track of their progress.
* Docker Hub
* Docker is a tool that makes creating, deploying, and running applications easier with the use of containers. The keywords of Docker are developed, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.
* Jenkins
* It was a tool to perform Continuous Integration (CI) by conducting automated tests on his code as changes were being made to it. Jenkins supports the complete development life cycle of software from building, testing, documenting the software, deploying, and other stages of the software development life cycle.
* Any open-source console (PuTTY)
* It is a popular tool for text-based communication and is also a popular utility for connecting servers from Microsoft operating system-based computers.

**Technologies used**

* Amazon Web Services (AWS)
* AWS services are delivered to customers via a network of AWS server farms located throughout the world. Fees are based on a combination of usage (known as a "Pay-as-you-go" model), hardware, operating system, software, or networking features chosen by the subscriber required availability, redundancy, security, and service options.

**Create the CI-CD flow for NodeJS web application using Freestyle project**

* Create an AWS account
* Create a docker hub account with user name and password
* Create a GitHub access token
* Navigate to GitHub settings, go to developer settings, in personal access token, by ticking the repo option generate a token.
* Note: The token should be copied since it is not possible to access it again.
* Create a node in Jenkins.

**What is Jenkins?**

Jenkins, originally called Hudson, is an open-source Continuous Integration tool written in Java. Boasting a dominant market share, Jenkins is used by teams of all sizes, for projects in a wide variety of languages and technologies, including .NET, Ruby, Groovy, Grails, PHP and more, as well as Java.

Firstly, Jenkins is easy to use. The user interface is simple, intuitive, and visually appealing, and Jenkins as a whole has a very low learning curve. As we will see in the next chapter, you can get started with Jenkins in a matter of minutes.

However Jenkins does not sacrifice power or extensibility: it is also extremely flexible and easy to adapt to your own purposes. Hundreds of open source plugins are available, with more coming out every week. These plugins cover everything from version control systems, build tools, code quality metrics, build notifiers, integration with external systems, UI customization, games, and much more. And installing them is quick and easy.

Last, but certainly not least, much of Jenkins’s popularity comes from the size and vibrancy of its community. The Jenkins community is a large, dynamic, reactive and welcoming bunch, with passionate champions, active mailing lists, IRC channels and a very vocal blog and twitter account. The development pace is fast, with releases coming out weekly with the latest new features, bug fixes, and plugin updates.

**What is Node and how it is used in Jenkins?**

A machine which is part of the Jenkins environment and capable of executing Pipelines or jobs. Both the Controller and Agents are considered to be Nodes.

Nodes are the "machines" on which build agents run. Jenkins monitors each attached node for disk space, free temp space, free swap, clock time/sync, and response time. A node is taken offline if any of these values go outside the configured threshold. Jenkins supports two types of nodes:

* agents
* Agents manage the task execution on behalf of the Jenkins controller by using executors. An agent is a small (170KB single jar) Java client process that connects to a Jenkins controller and is assumed to be unreliable. An agent can use any operating system that supports Java. Any tools required for building and testing get installed on the node where the agent runs. Because these tools are a part of the node, they can be installed directly or in a container, such as Docker or Kubernetes. Each agent is effectively a process with its own Process Identifier (PID) on the host machine. In practice, nodes and agents are essentially the same but it is good to remember that they are conceptually distinct.
* built-in node.
* The built-in node is a node that exists within the controller process. It is possible to use agents and the build-in node to run tasks. However, running tasks on the built-in node is discouraged for security, performance, and scalability reasons. The number of executors configured for the node determines the node’s ability to run tasks. Set the number of executors to 0 to disable running tasks on the built-in node

**Plugin management (ensure that the plugins you installed are safe)**

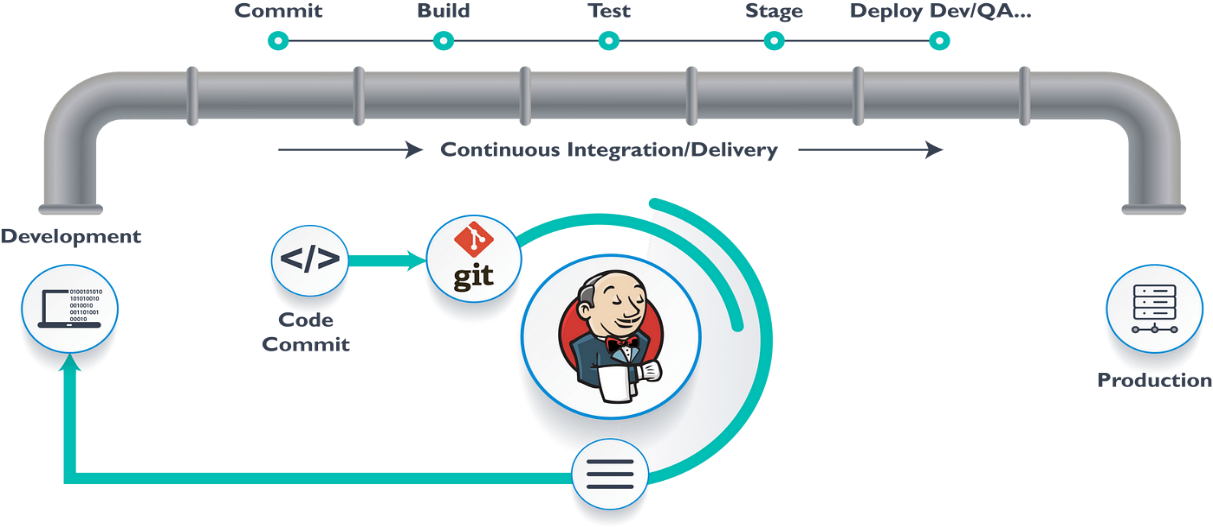
* Node related plugins
* Git related plugin
* Docker related plugins

**Overview of CI-CD Pipeline**

 A CI/CD pipeline “**automates your software delivery process”**. The pipeline builds code, runs tests (CI), and safely deploys a new version of the application (CD). Automated pipelines remove manual errors, provide standardized feedback loops to developers, and enable fast product iterations.

**Continuous Integration:** Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control.

**Continuous Delivery:** CD is the process of automating build, test, configuration and deployment from a build to a production environment. Continuous Delivery automates the entire software release process**.**



**Fig**: CI-CD Pipeline Flow

**CI-CD Benefits:**

* Superior Code Quality
* Reduced changes and review time
* Accelerated Release Cycles
* Fault Detection and Isolation
* Enhanced test Reliability
* Reduced Backlog
* Improved Mean Time to Resolution (MTTR)
* Cost Deduction
* Enhanced Transparency and Accountability
* Boosted Customer Satisfaction
* Streamlined Communication
* Frequent updates and maintenance
* Enhanced Performance metrics

**Steps to follow to create a CI-CD Pipelines using Jenkins:**

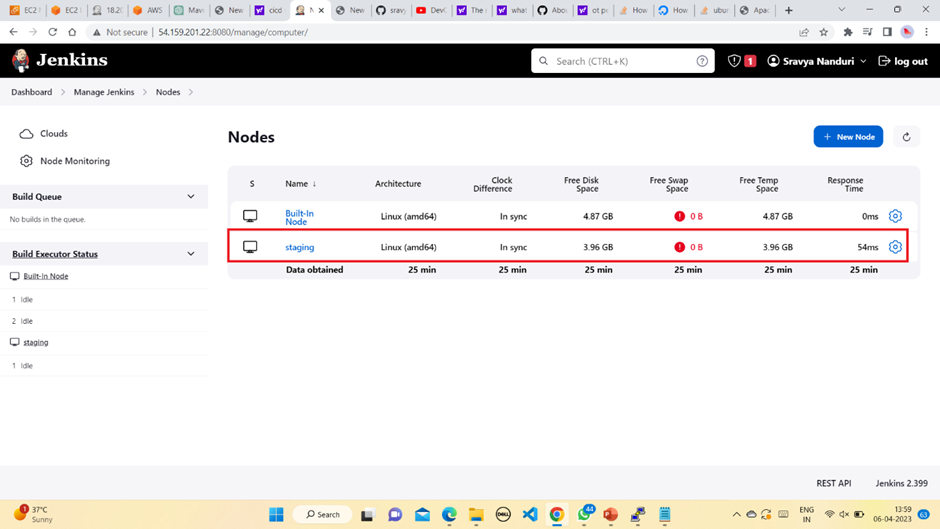
* Create 2 ubuntu instances in AWS.
* Let them be Development and Staging.
* Development Environment is where we do all the building i.e., writing the code in your local machine.
* Staging environment is where we deploy or push our code into the server. This is the IP of our web application.
* Connect to Development instance using PuTTY (If you have downloaded the .pem key, you can convert it into .ppk format through PuTTYgen.
* In edit inbound rules add another rule with port number 8080.
* In this instance, we have to install Jenkins by using the following commands.
* sudo apt update
* sudo apt install openjdk-11-jre
* curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \

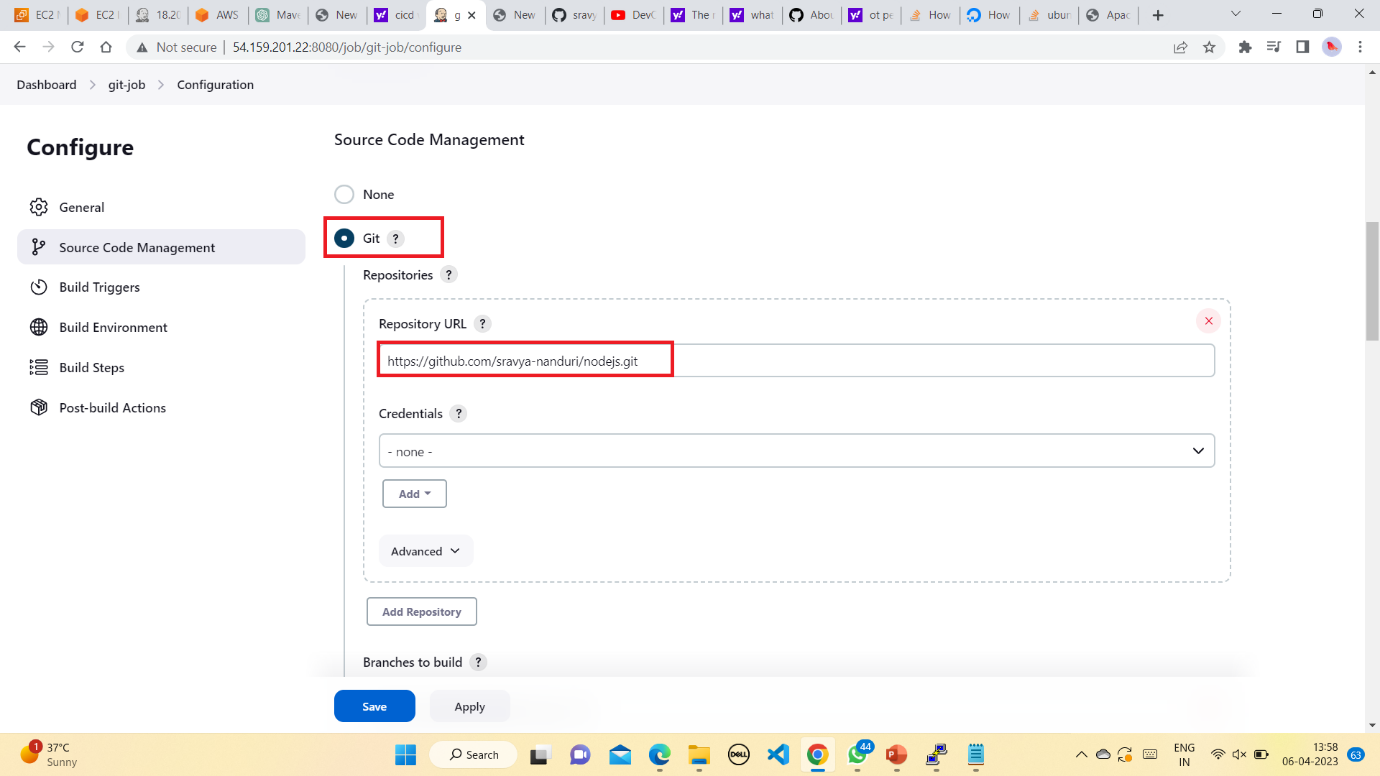
/usr/share/keyrings/jenkins-keyring.asc > /dev/null

* echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

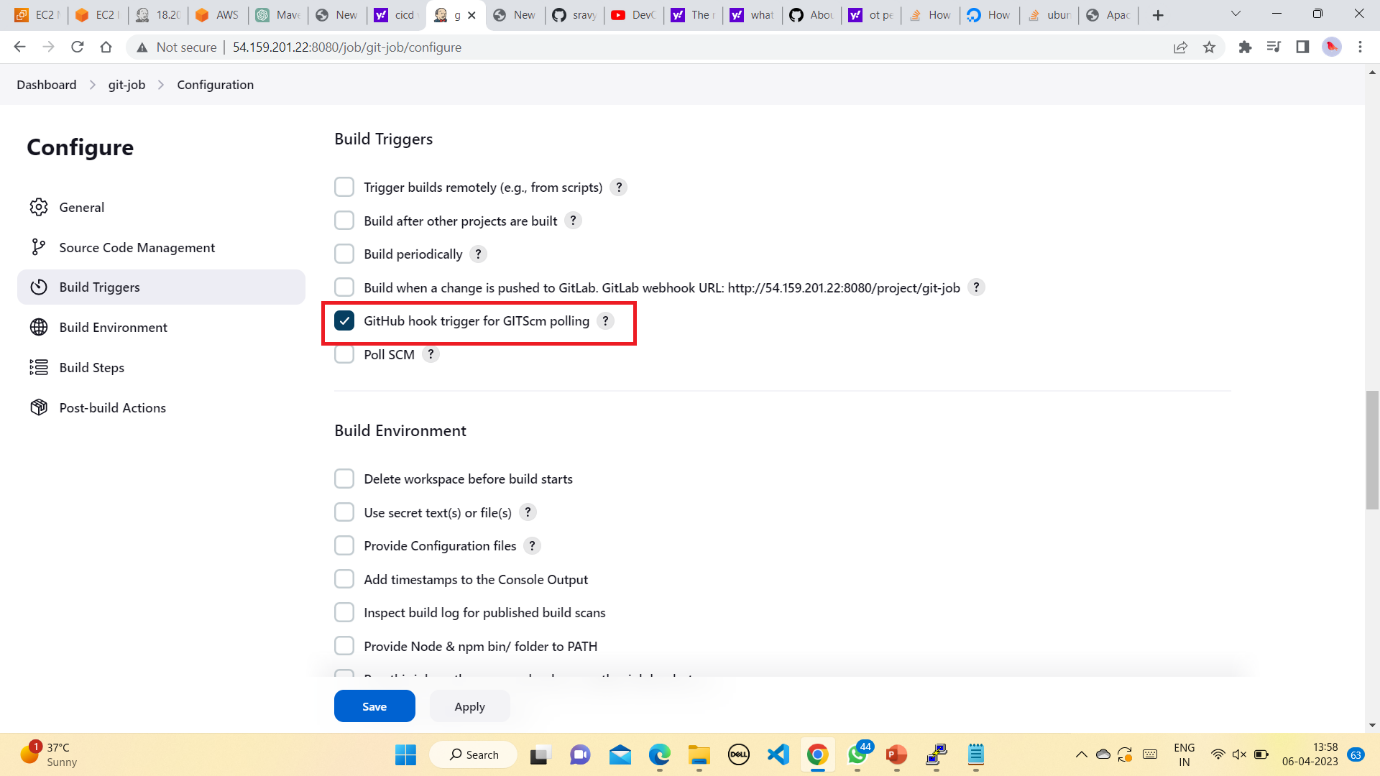
https://pkg.jenkins.io/debian binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

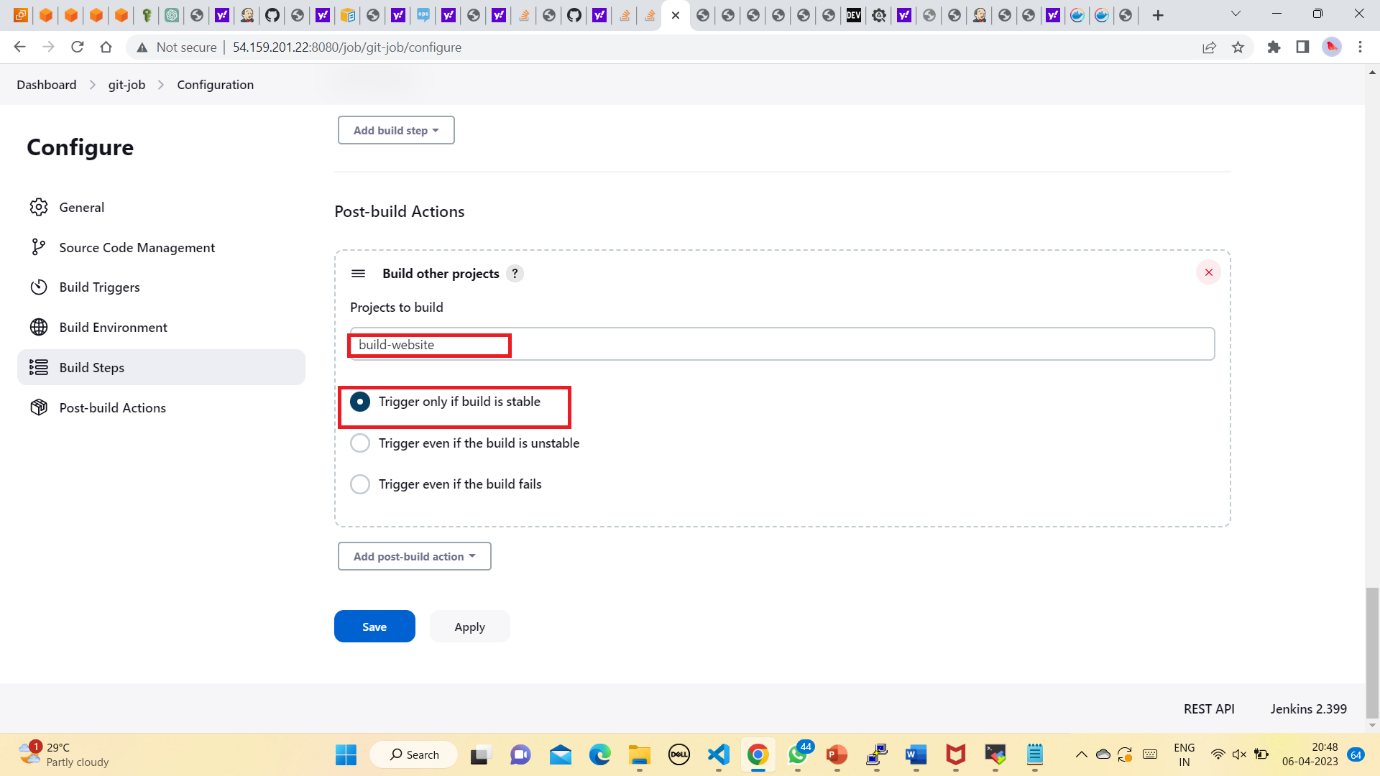
* sudo apt update
* sudo apt install jenkins
* sudo apt install git
* sudo apt install apache2
* With the public IP of your instance and colon(: ) 8080 check whether the jenkins is loading or not.
* Install all the necessary plugins required.
* Login to the Jenkins tool using user details.
* Connect the staging instance and install Openjdk.
* Navigate to the Jenkins tool, in configure security set agents to random (this is very important since we are connecting the servers through a JNPL connection).
* Now in Manage Nodes and Clouds, create a new item by specifying the remote directory.
* Now paste the commands displayed in the node in the Staging instance (to download agent .jar, slave-agent files, this is very important to establish the connection between the two servers).
* By doing the above step the node will come into sync.
* Now we have successfully created a node
* Now take a sample git repository of a web application and git clone it into the Jenkins instance
* This will ask for authentication. Then we have to specify the username of our github.com account, but the password is the token generated earlier.
* Now we have to git push this sample web application into a new repository.
* After which we should go back to the Jenkins tool and create an item(git-job), by selecting a Freestyle project, with the following configuration.
* **Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project combining any SCM and any build system. A Free-Style project is a project that can incorporate almost any type of build. The Free-Style project is the more "generic" form of a project. You can execute shell/dos scripts, invoke ant, and a lot more. Majority of the plugins are written to use the free-style project.
* **Maven project**: A maven project is a project that will analyze the pom.xml file in greater detail and produce a project that's geared towards the targets that are invoked. The maven project is smart enough to incorporate build targets like the javadoc or test targets and automatically setup the reports for those targets.
* **Multi-configuration project**: The “multiconfiguration project” (also referred to as a “matrix project”) lets you run the same build job in many different configurations. This powerful feature can be useful for testing an application in many different environments, with different databases, or even on different build machines. We will be looking at how to configure multiconfiguration build jobs later on in the book.
* **Monitor an external job**: The “Monitor an external job” build job lets you keep an eye on noninteractive processes, such as cron jobs.



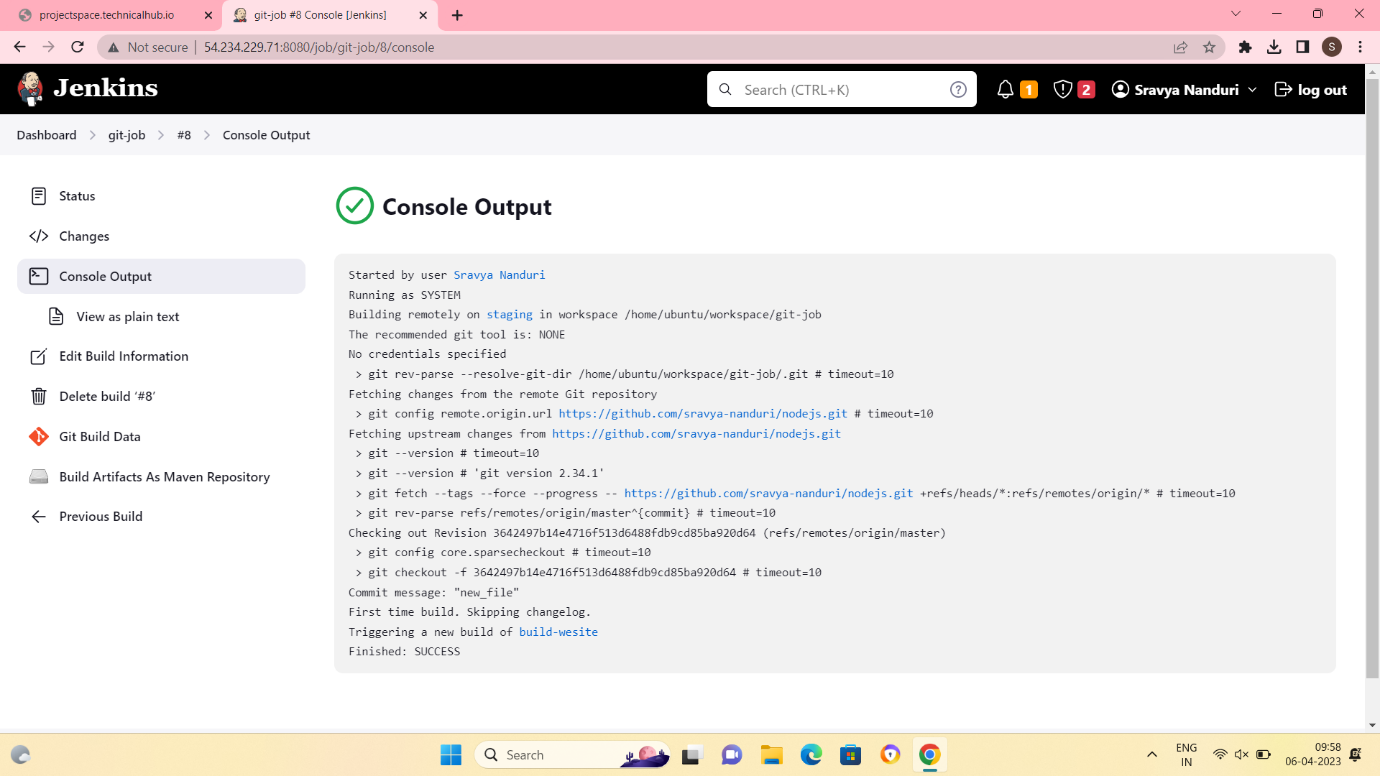
* We are specifying the git repository into which we have pushed our web application



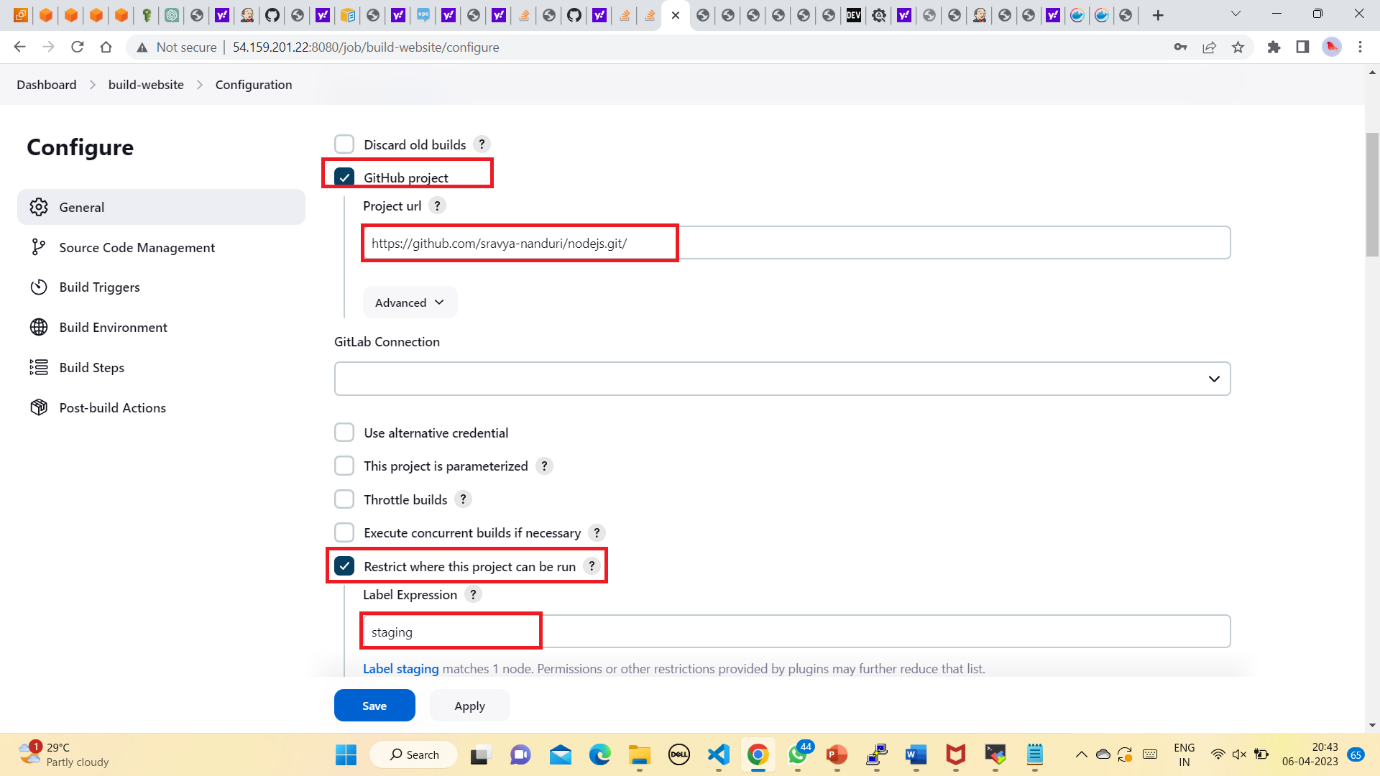
* Here we are selecting the GitHub hook trigger so that it will trigger every time we make changes in our application since we are adding a git webhook later



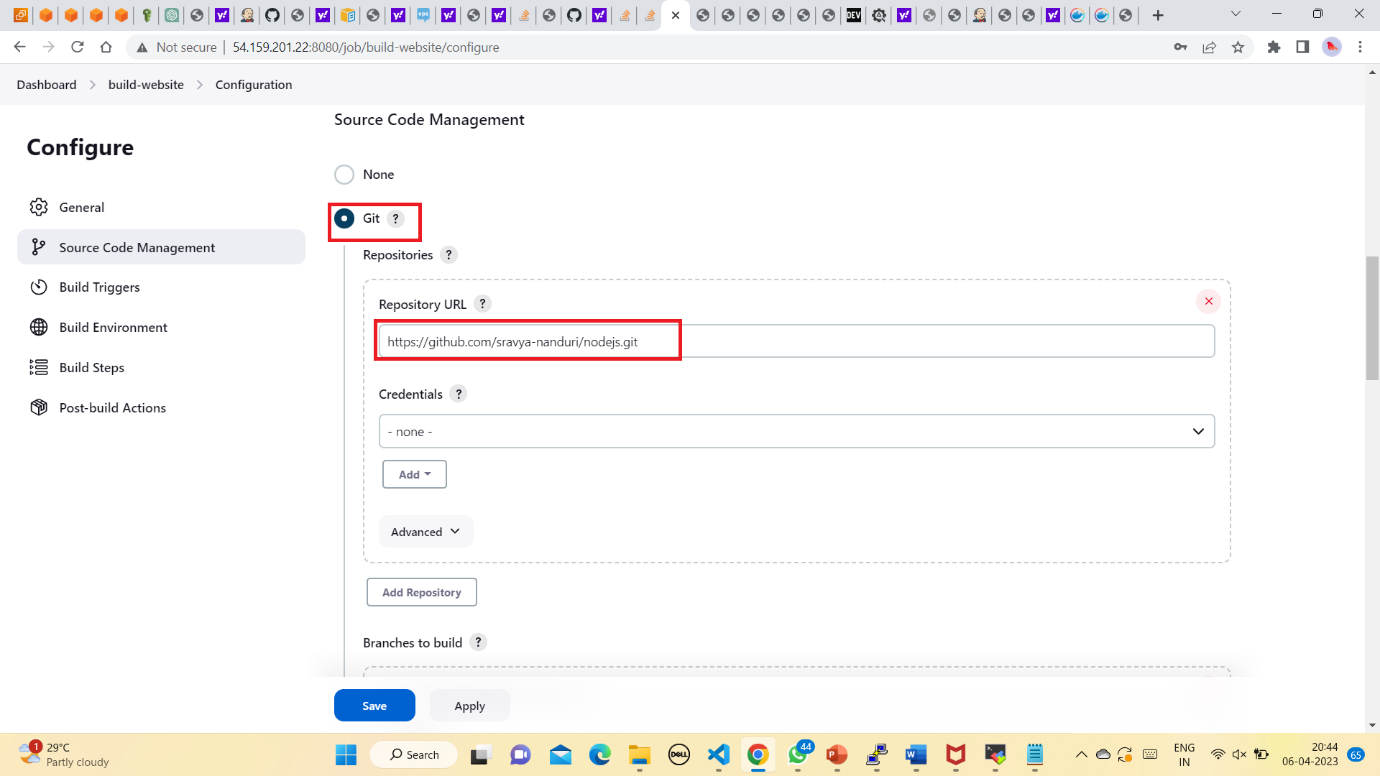
* Here, in the post build actions we are giving the build-website to triiger that job after this job is built successfully
* We should check whether the item or the job is building properly or not.



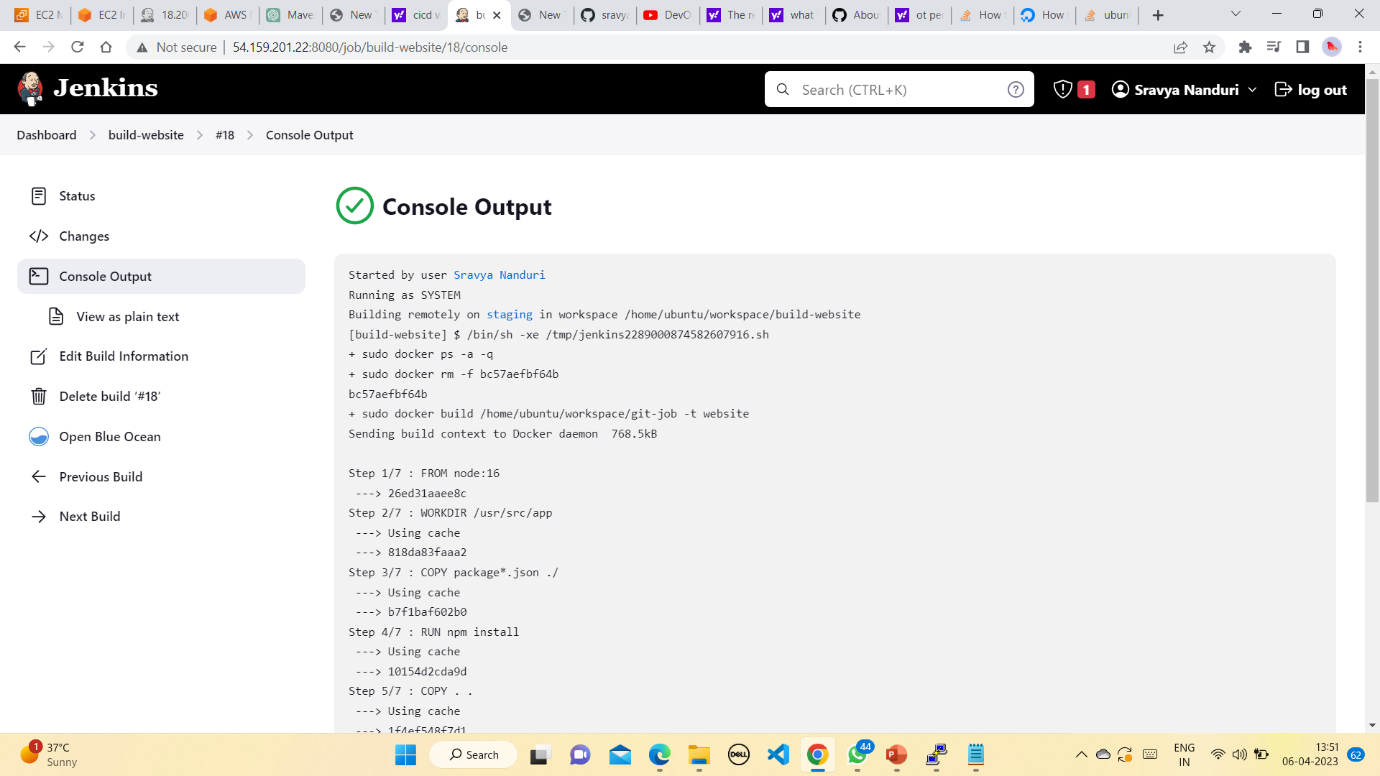
* After that we have to create another item, with the following configuration.

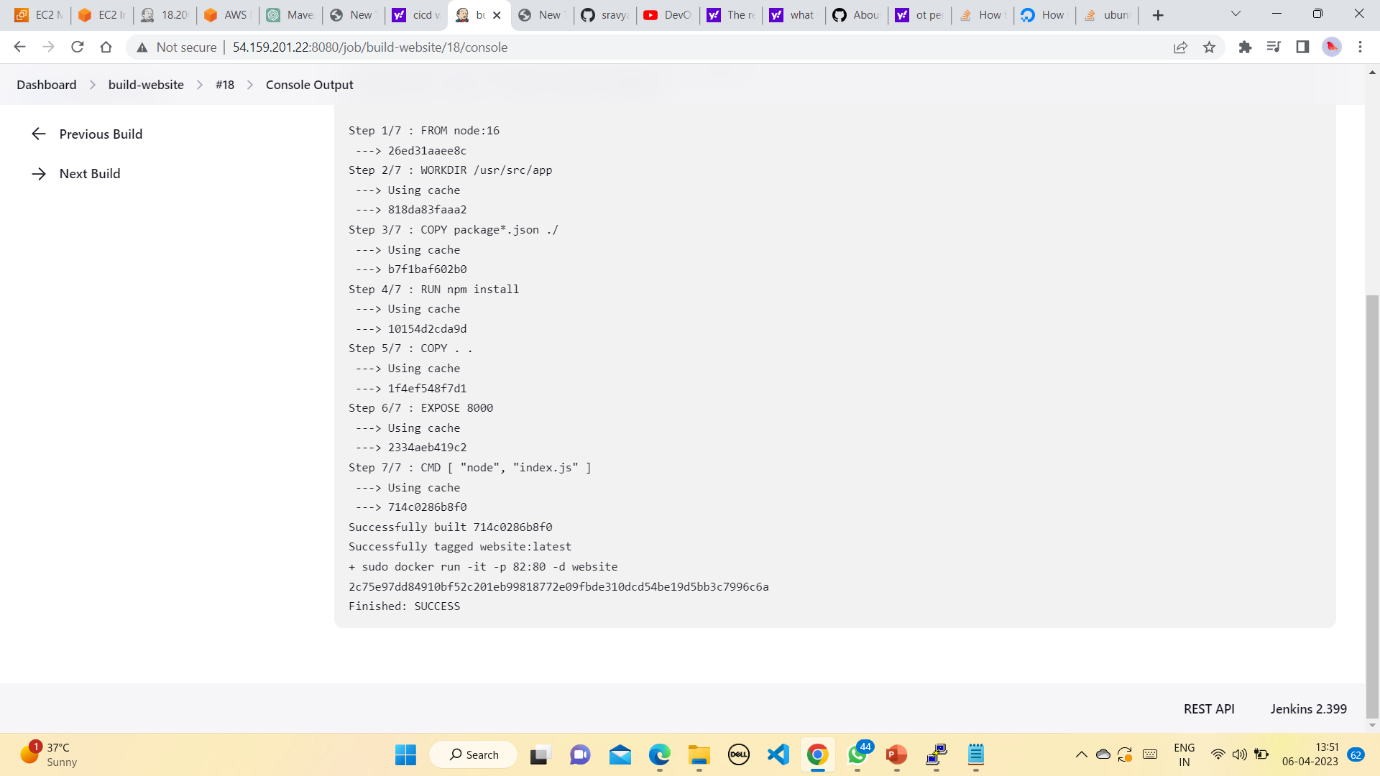


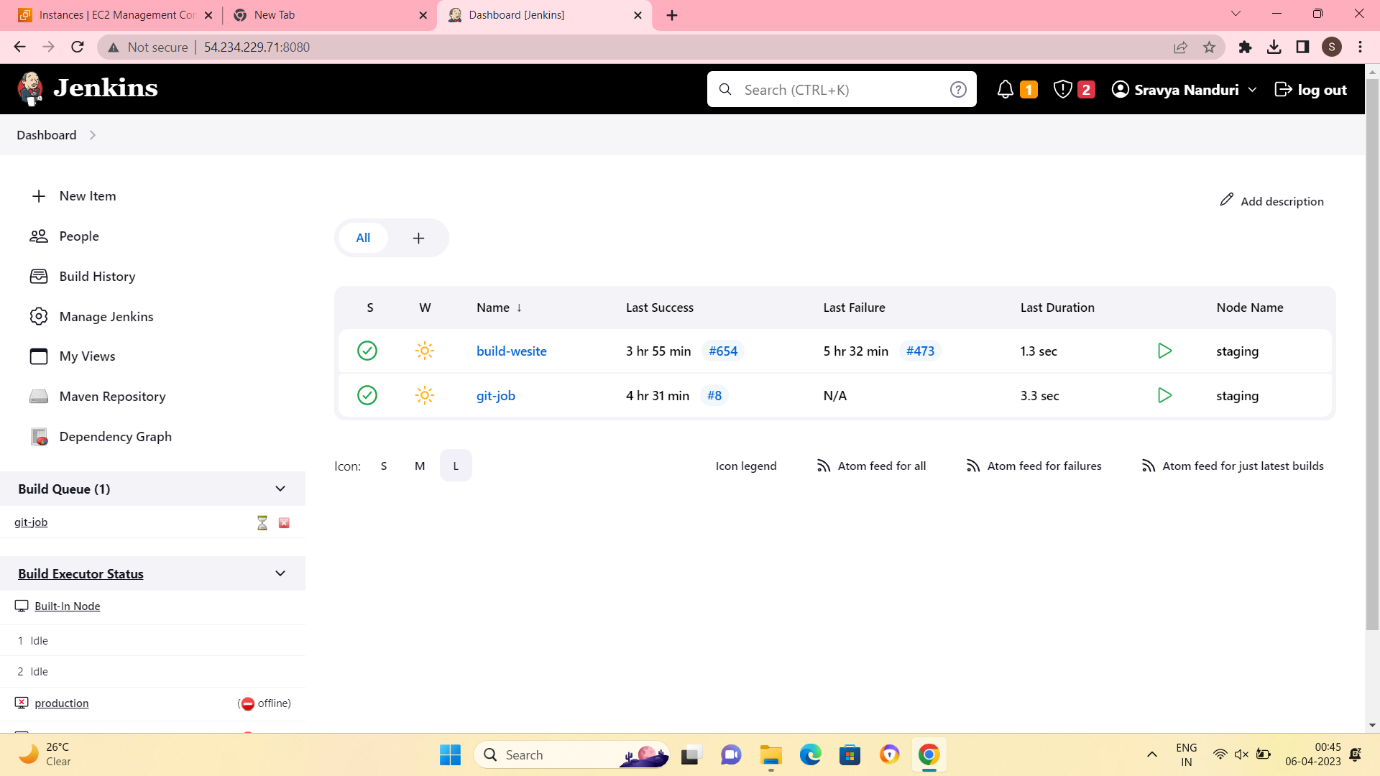
* We have to mention the staging node in the restrict where the project can run. So that this can have a isolated environment.



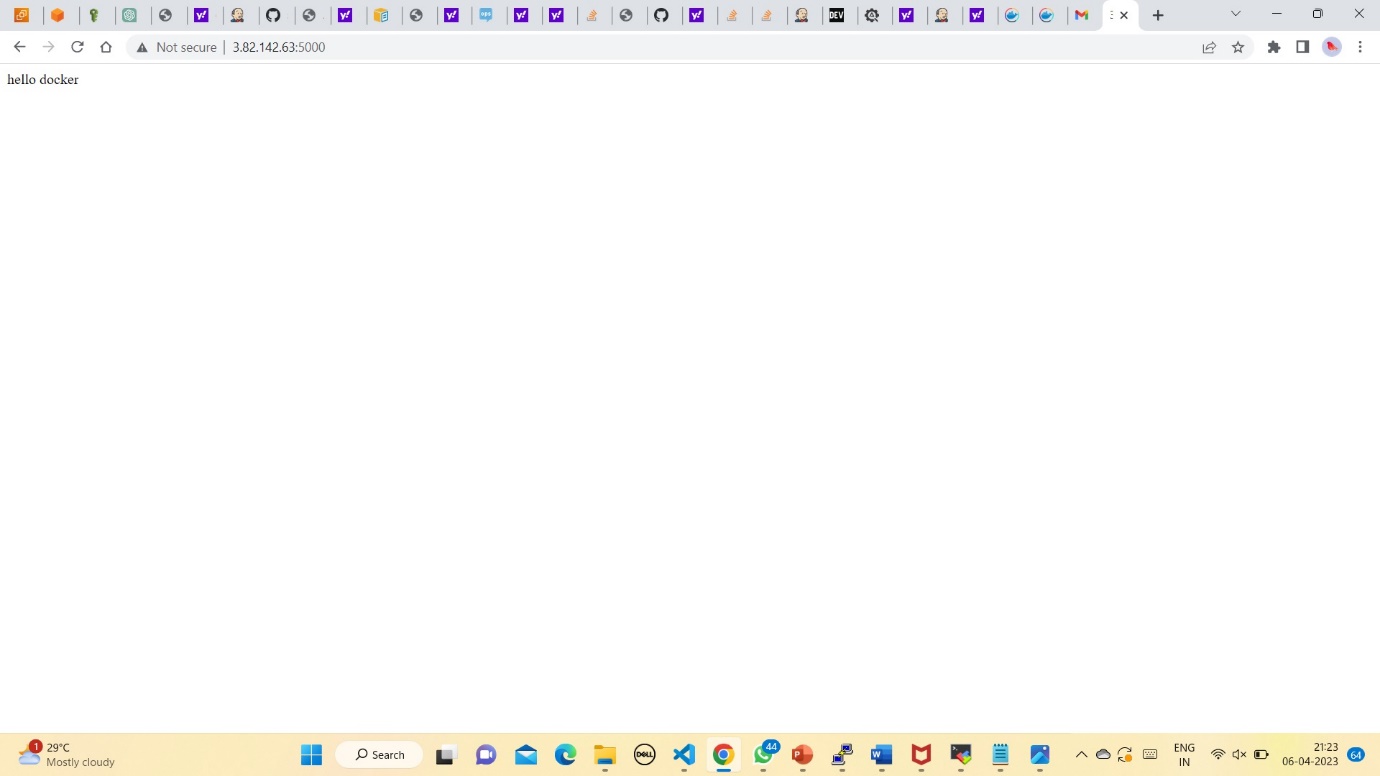
* In build steps in execute shell we have to give some commands in order to run the docker container those are
* sudo docker rm -f $(sudo docker ps -a -q) (all the running containers will be removed)
* sudo docker build <path in staging> -t website (name of the container)
* sudo docker run -it -p 82:80(any port) -d website (name of image)
* We have to check whether the second item (build-website) is building successfully or not.



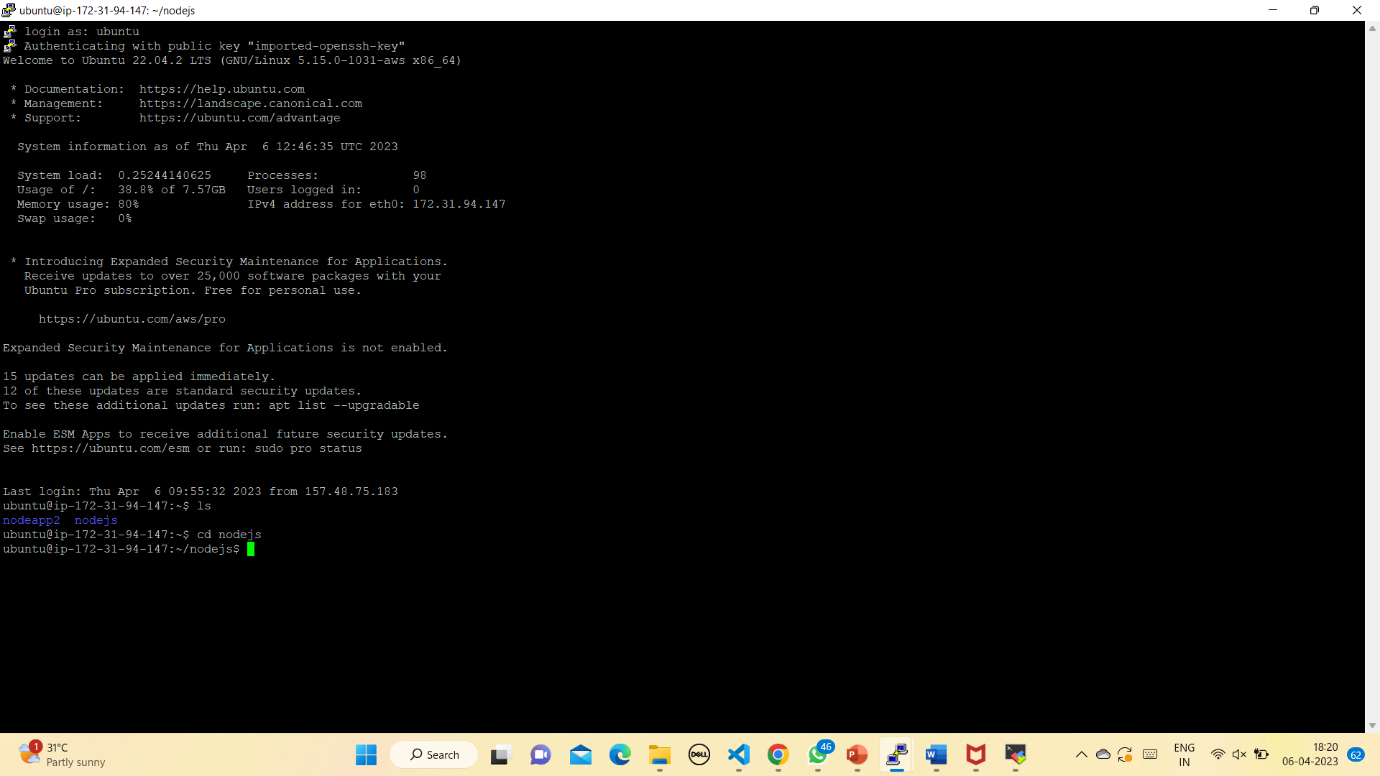


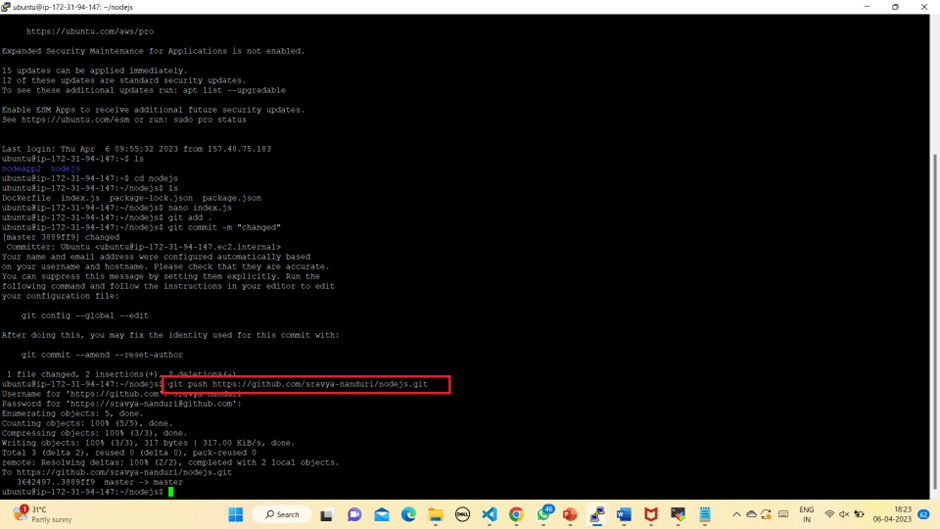


* Now we have to create a webhook for the git repository. For that we have to navigate to our git repository and we have to go to settings and add a webhook.
* After building both the jobs successfully we have to check whether the application is displayed by giving the respective port and the IP address.

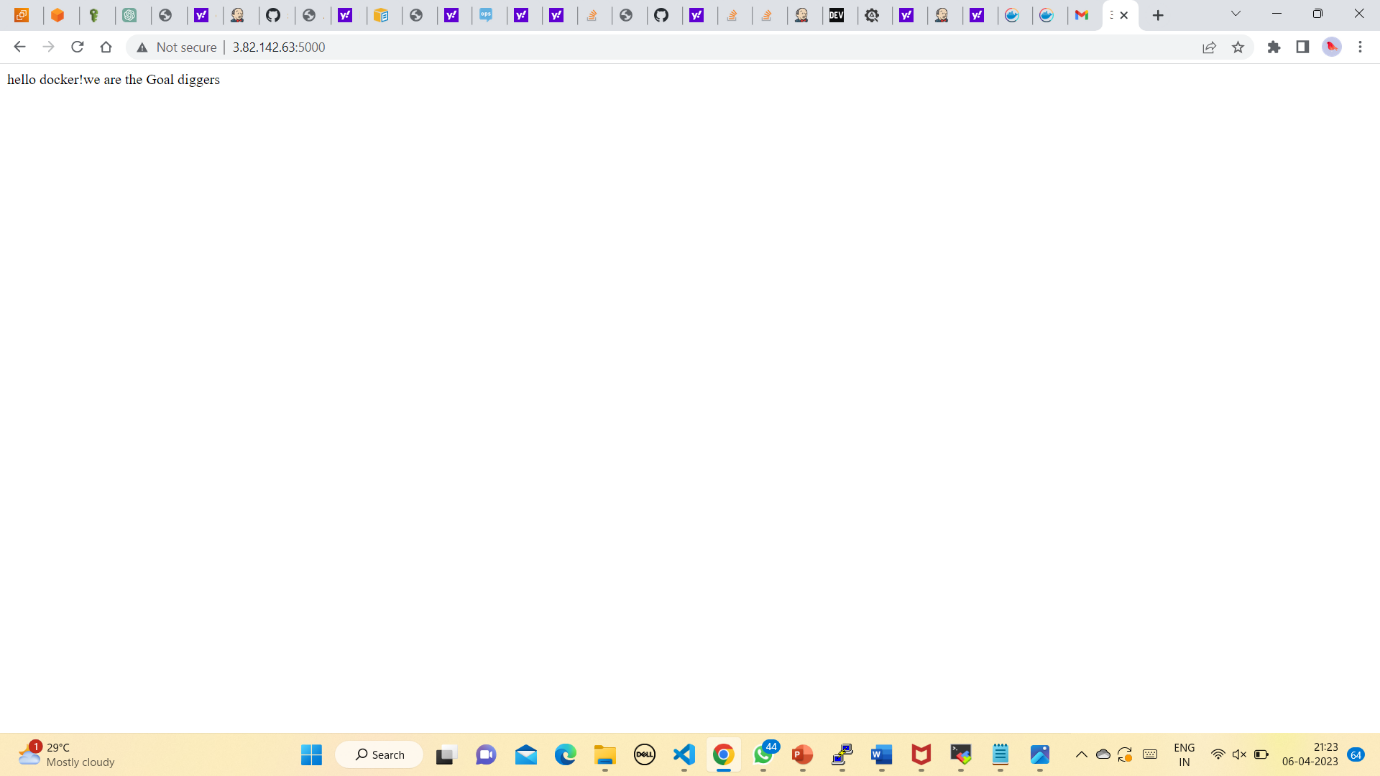


* The content in the indes.js file of web application is displayed successfully.
* Then we can see the content written in the index.js file in the NodeJS application.
* Now to check whether the CI-CD pipeline is created successfully or not we have to make some changes in the index.js file in our application through the Development server.





* Pushing the changes into the git repository
* We have to commit the changes again and we have to git push it to the repository. After doing this, we have to refresh the web page our application with the changes made. This means you have successfully created a CI-CD pipeline.



**Presented by:**

**Harsha Srivalli**

**Email: harshasrivalli313@gmail.com**

**Venkata Madhuri**

**Email: kamisettikavya@gmail.com**

**Kalpana Chawla**

**Email: kalpanachawlamaddipatla@gmail.com**

**Sravya**

**Email: sravya.nanduri03@gmail.com**