Jenkins

Jenkins is an **automation tool** used to create CI/CD pipelines that build, test and deploy software. It is not limited to that it can automate anything, you can run your python, bash scripts as well as your Ansible playbooks.

Additionally, it will have various plugins that can be used that work well with the testing tools like Junit and Selenium. It can be installed on operating system like windows, Linux, MacOS.

It has a web interface, it can create a job and create a functionality you want to like build triggers, provide RBAC and encryption.

Jenkins Infrastructure

Master server which controls the pipelines and schedules the builds to the agent. Then the **Agents/Minions** will run the build on the workspace.

Example workflow:

1. A developer commits a code to the Git Repository, Master becomes aware of this commit and triggers the appropriate pipeline and then distributes the builds to one of the agents to run, selects the appropriate agent based on the labels.
2. Agents run the builds, which are basically Linux commands to build, test and deploy the code.

Jenkins Agents

Permanent Agent: This are just dedicated, everyday standalone Linux, windows servers set up to run Jenkins job. You need to have JAVA installed on the servers and set up a SSH as master makes the communication using SSH. Necessary tools must also be installed on the servers.

Cloud Agents: These are more popular choices in the real world, Ephemeral/Dynamic agents spun up on the demand based on the agents template you configure.

Example: Docker, Kubernetes and AWS fleet manager.

Build Types

Freestyle builds Project: Simplest method to create a build. You can use UI and plugin to manage the build. Mostly they will be set up to execute a shell script.

Pipeline:

Pipeline used groovy syntax to specify what happens in the build. Pipelines are mostly broken down into stages.

Common stages of pipeline are:

Clone: Usually clone the local repository and set up environment variables on the agent.

Build: It will take up the code and builds it, means generating local artifacts like a JAR file, executable or a container image.

Test: Test against the newly built code.

Package: It gets packaged ready for the deployment**.**

Deploy: Sent out artifacts to our registry.

Install Jenkins

You can directly install Jenkins on the OS or deploy it in a container like docker or Kubernetes, suggested method is to use in the docker.

Install **blue ocean** is popular addon for the Jenkins which provides a new user experience based on personalized, modern design that allows users to graphically create, visualize and diagnose CI/CD pipelines.

I'm using AWS EC2 instance to launch Jenkins via a docker container, please follow the below steps:

Prerequisites

- A system with Docker installed. You can download Docker from the official website.
- An active internet connection.

Step 1: Connect to your EC2 instance and you can follow Jenkins installation as per your OS form the official Jenkins page: https://www.jenkins.io/doc/book/installing/docker/ or clone my Git repository link: https://github.com/manasi09-pd/DOCKER.git

Step2: I have ubuntu OS, use the build command to build the docker Image, and use docker file.

Add the ubuntu user to the docker group, it should have Sudo permission.

- docker build -t myjenkins-blueocean:2.414.2 .
- `#IF you are having problems building the image yourself`

- docker pull devopsjourney1/jenkins-blueocean:2.332.3-1 && docker tag devopsjourney1/jenkins-blueocean:2.332.3-1 myjenkins-blueocean:2.332.3-1

```
ubuntu@ip-172-31-39-42:~$ ls
DOCKER
ubuntu@ip-172-31-39-42:~$ cd DOCKER/
ubuntu@ip-172-31-39-42:~/DOCKER$ ls
README.md  first-docker  flask-docker  jenkins-docker  kubernetes-docker
```

```
FROM jenkins/jenkins:2.414.2-jdk11
USER root
RUN apt-get update && apt-get install -y lsb-release python3-pip
RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
  https://download.docker.com/linux/debian/gpg
RUN echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
  https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list
RUN apt-get update && apt-get install -y docker-ce-cli
USER jenkins
RUN jenkins-plugin-cli --plugins "blueocean:1.25.3 docker-workflow:1.28"
```

```
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker$ ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker$ docker build -t myjenkins-blueocean:2.414.2 .
ERROR: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Head "http://%2Fvar%2Frun%2Fdocker.sock/_ping": dial unix /v
ar/run/docker.sock: connect: permission denied
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker$ sudo docker build -t myjenkins-blueocean:2.414.2 .
[+] Building 61.5s (8/9)                                                                                                                    docker:default
 => => sha256:5e0093fb8d5f36e86b93870a544e50bf21f3968e872f91ad0a2bc9abdc672472 8.68MB / 8.68MB                                                        0.7s
 => => sha256:b705323eaf70a7da4c1eed8b816f33dff2d5c8c3671170a2c17cf77aa4f15432 3.12kB / 3.12kB                                                        0.0s
 => => sha256:3d0d48f61941fc7cb5253b07efd932d06ee6e335445f6f535ac3d1d49e385a7e 12.62kB / 12.62kB                                                      0.0s
 => => sha256:f6154b0007a4a7dcee550de8b63ddbdf177961a972a33c54b2c4390dc4a873bf 61.32MB / 61.32MB                                                      1.9s
 => => sha256:34d3735955786e60f8bb8e7bd6fa2dfd7739dc047ba49c637c18a5d54b99d23c 1.23kB / 1.23kB                                                        1.0s
 => => sha256:2471c7529ac37af126ea6c1afad134f08afa5c684f7e10816e314d286ce7dc38 89.33MB / 89.33MB                                                      3.1s
 => => extracting sha256:012c0b3e998c1a0c0bedcf712eaaafb188580529dd026a04aa1ce13fdb39e42b                                                             4.3s
 => => sha256:10cad8b7dbf876c2afb6402c97060b77d723bd6838daef8c2fadab362864f768 183B / 183B                                                            1.2s
 => => sha256:655eafc427231f280d11bcd10dfa0887a3165d2eb9345ab44ed75063bed17ae7 189B / 189B                                                            1.4s
 => => sha256:3145851d1a67db7eb3dd0d7f6734a26340476b9d1f6db58668088d72b4833847 6.09MB / 6.09MB                                                        1.7s
 => => sha256:614d8a8c6a95f7a9ab30388cf16a55f34eb54e0bcc52d4c3ef5033ac46200cd8 77.15MB / 77.15MB                                                      3.5s
 => => sha256:95f5dcd4e08b1d468bac93b830b9aa4d9c0f302d5be66c478dc70fb48fb6bf62 1.93kB / 1.93kB                                                        2.2s
 => => sha256:7c8f869ed080beb2a95df25c497c699a86746c448fbb3a6f29bfd42313ec2a49 1.16kB / 1.16kB                                                        2.4s
```

Step3: Create the network for Jenkins by using the command below. This isolation can be useful when you want to group containers together logically and control their communication.

```
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker$ sudo docker network create jenkins
680072ee301b5ff73b2e20ec5b71a51812a3e7310b3a6a905566ac8bc082edf5
```

```
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker$ sudo docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
efa26d92a8d6   bridge    bridge    local
2310246b1da9   host      host      local
680072ee301b   jenkins   bridge    local
74929e22421c   none      null      local
```

Step4: Create the docker container. Specifying the network Jenkins and environment variables specifying the location of certificates and volumes.

```
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker$ sudo docker run \
  --name jenkins-blueocean \
  --restart=on-failure \
  --detach \
  --network jenkins \
  --env DOCKER_HOST=tcp://docker:2376 \
  --env DOCKER_CERT_PATH=/certs/client \
  --env DOCKER_TLS_VERIFY=1 \
  --publish 8080:8080 \
  --publish 50000:50000 \
  --volume jenkins-data:/var/jenkins_home \
  --volume jenkins-docker-certs:/certs/client:ro \
  myjenkins-blueocean:2.492.1-1
9715ad345ef94f260af75a6cf61d241c2e737b567e19fdf9c45e9865afc69e10
```

```
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker$ docker ps
CONTAINER ID   IMAGE                           COMMAND                  CREATED              STATUS              PORTS
                                   NAMES
9715ad345ef9   myjenkins-blueocean:2.492.1-1   "/usr/bin/tini -- /u…"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp, 0.0.0.0:500
00->50000/tcp, [::]:50000->50000/tcp   jenkins-blueocean
```

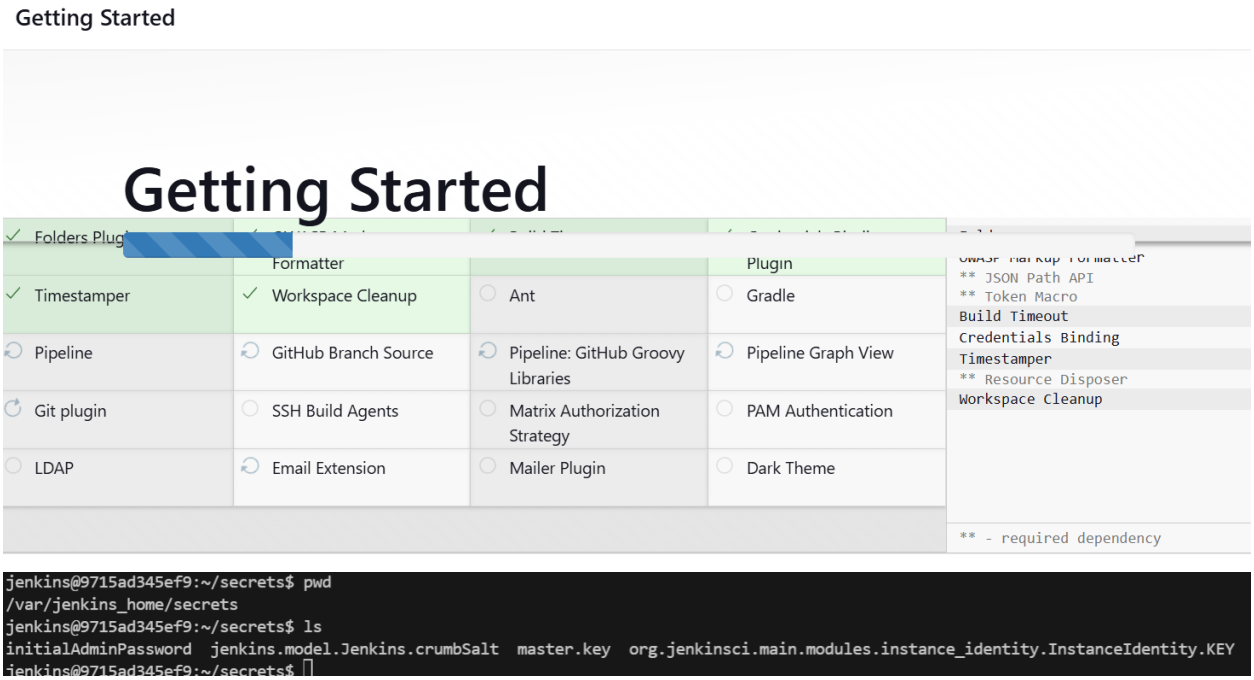
Step5: Access your Jenkins WEB page via your private IP address and if its not opening add custom TCP port 8080, edit the inbound traffic rules.

–	sgr-0dd9b4f1a9c904cb5	8080	TCP	0.0.0.0/0

Step6: To find the password, docker exec command can be used to check the admin password.

```
ubuntu@ip-172-31-39-42:~$ docker exec -it 9715ad345ef9 /bin/bash
```

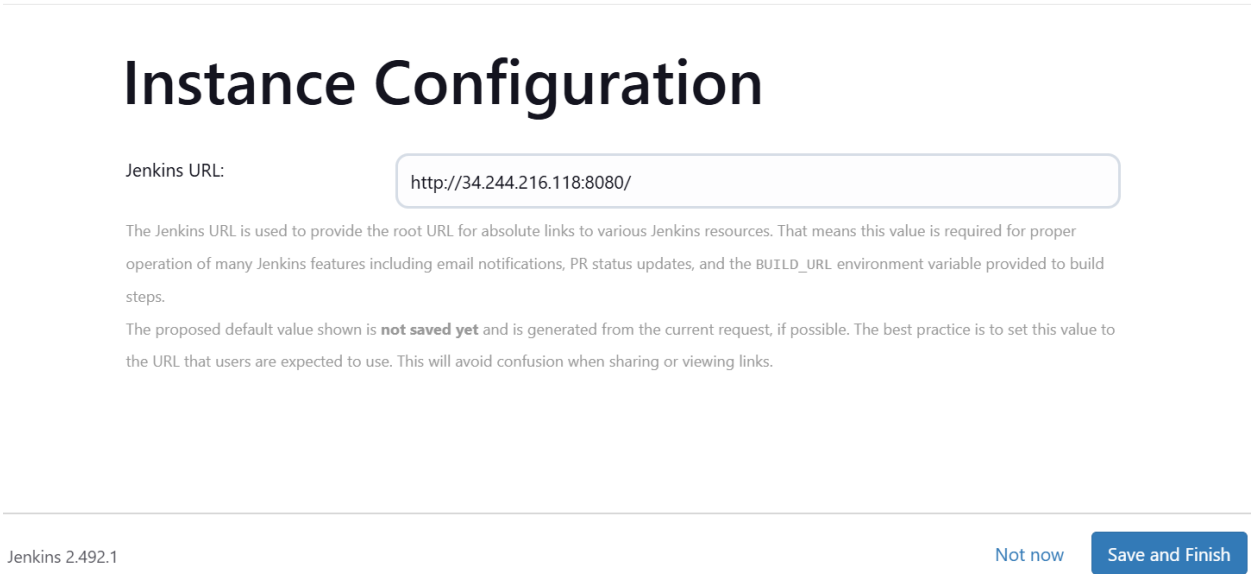
Step7: Installed the suggested plugins for Jenkins.




```
jenkins@9715ad345ef9:~/secrets$ pwd
/var/jenkins_home/secrets
jenkins@9715ad345ef9:~/secrets$ ls
initialAdminPassword  jenkins.model.Jenkins.crumbSalt  master.key  org.jenkinsci.main.modules.instance_identity.InstanceIdentity.KEY
jenkins@9715ad345ef9:~/secrets$
```

Step8: Create the Administrator account.

Step9: Default configuration can be used or provide a different hostname or IP.



Step10: Jenkins WEB GUI page is now ready to use.

Dashboard: It's a home page, easy to navigate through Jenkins.

New Item: creating projects and pipelines.

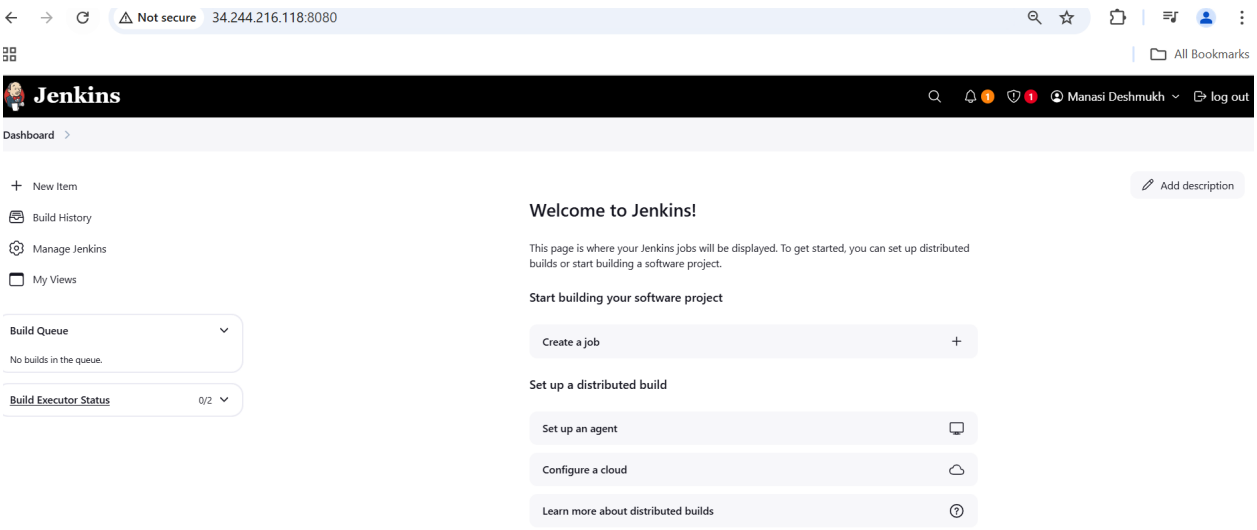
People: You administrate people and user accounts.

Build History: Gives history for all your jobs that have been run.

Manage Jenkins; This has all the management stuff like install plugins, agents.

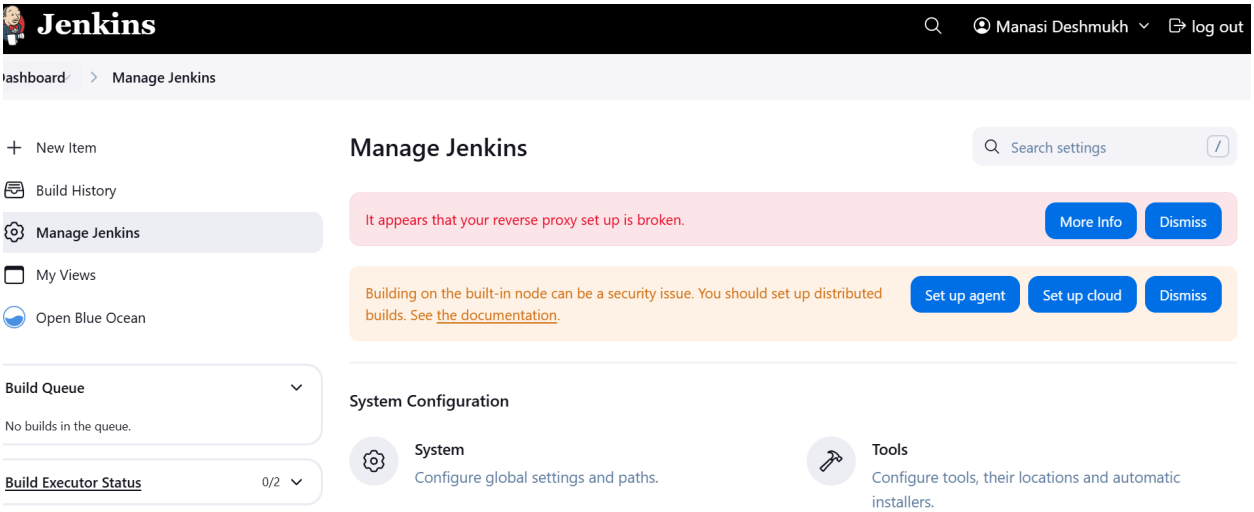
My view: Just a way to organize your jobs.

Blue ocean: Plugin that enhances the CI/CD pipelines.



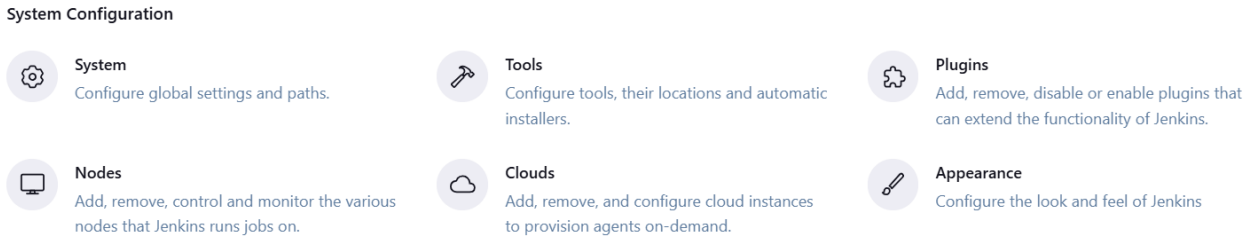
On the **Manage Jenkins page**, you would see some errors like reverse proxy this issue occurs if you have used the public IP address for your ec2 instance, wherever the ec2 instance goes down the Public IP address changes so for that you can attach a elastic IP address to the EC2 instance.

Agent needs to be set up so that it can run the pipeline.



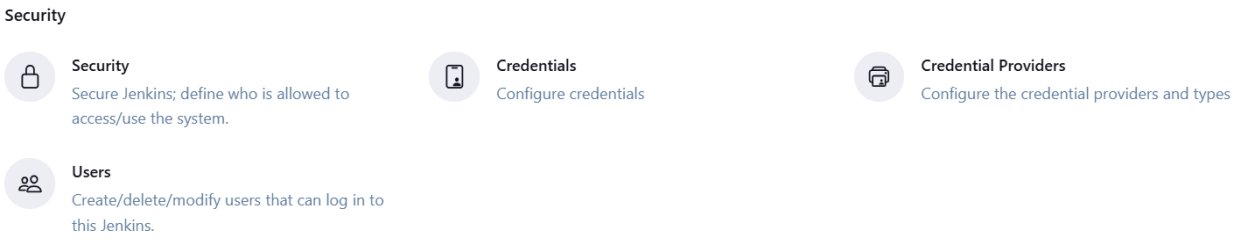
System Configuration:

Information of a Jenkins server, set global parameters, manage plugins Jenkins administrator spend lot of time here, manage node an clouds we can set up the agents, docker or Kubernetes.

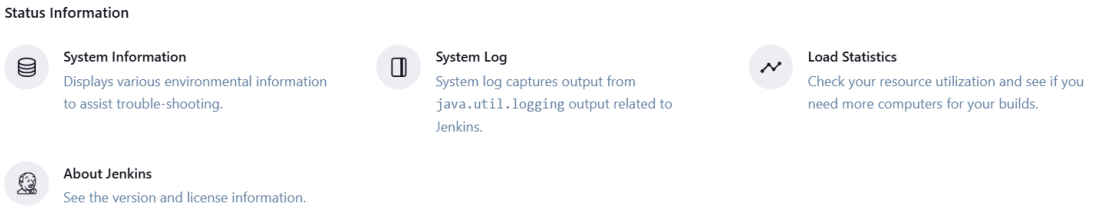


Security:

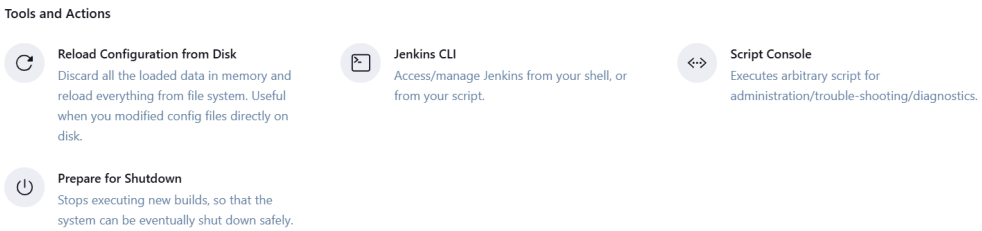
Configure global security, manage the credentials and the users, store SSH keys, API keys. You can also use AWS secrets manager to store the keys, or Jenkins has a built-in credentials storage manager.



System information, logs and statistics.

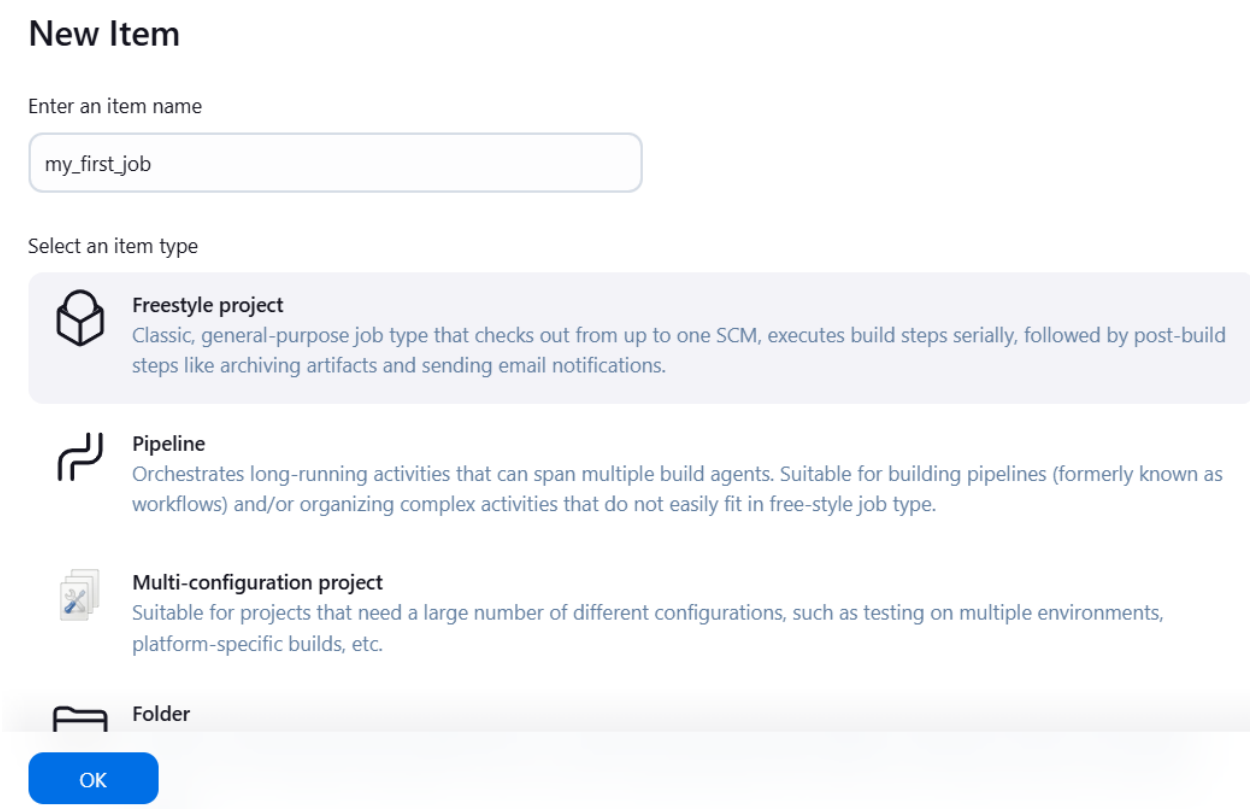


Other tools like preparing for shutdown, use this on the days where you need to shut down the Jenkins server for maintenance, if you reboot it the current jobs will stop, so prepare for shutdown Jenkins will complete the jobs that are running and will not take up any new jobs.

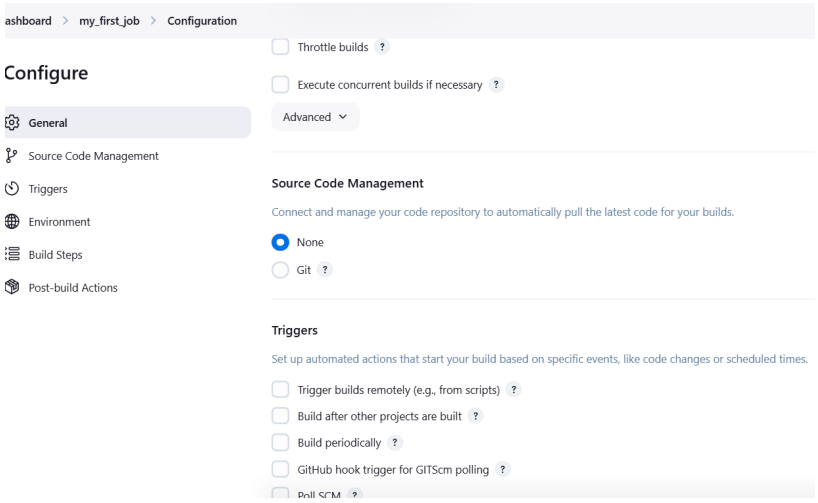


Let's start with the freestyle project:

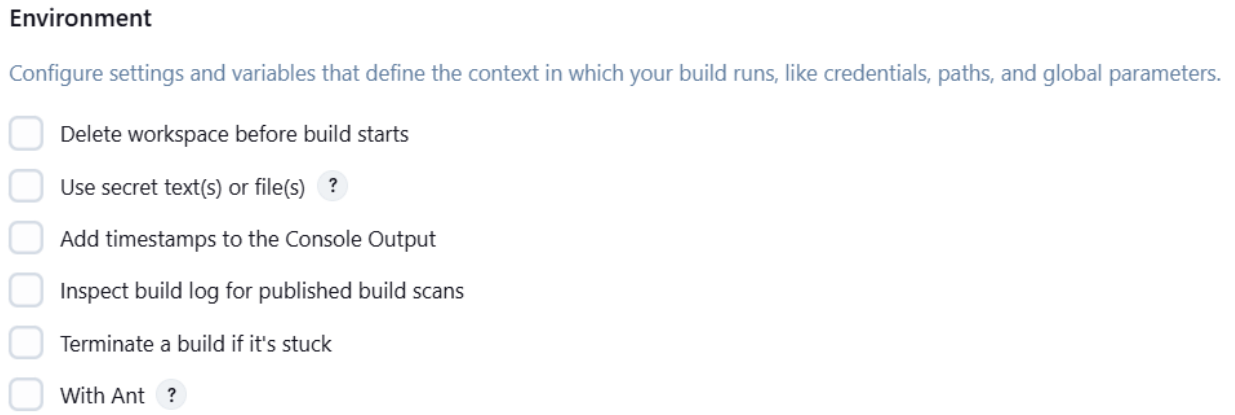
Don't provide space while entering a new item name, as at the back end the Jenkins job will create a folder. It's recommended to use _ or – instead.



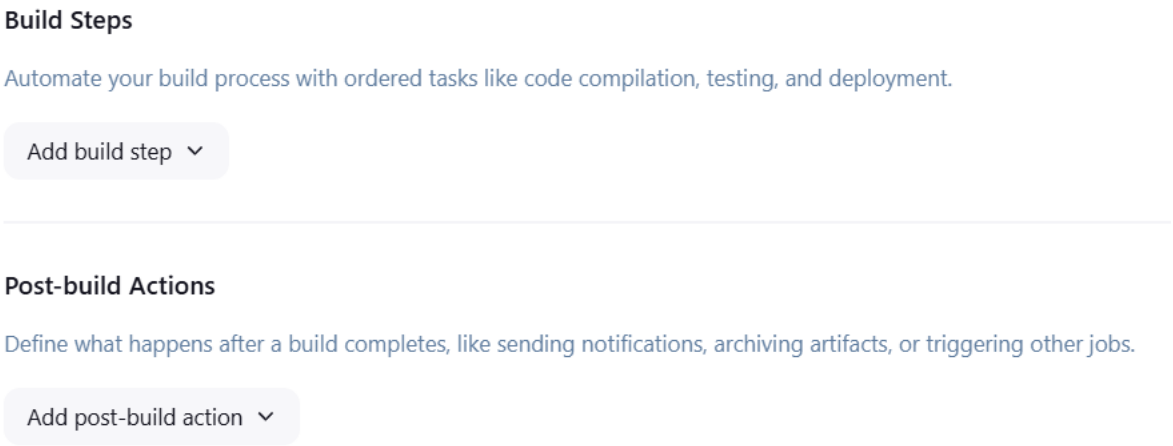
Then with the project settings, you can set the git repository hosted on this project and build triggers you can set the webhook triggers. If you have a firewall, then you need to open the ports or use a proxy server. Another way is to use SCM Jenkins will reach the GitHub and check regularly for any changes.



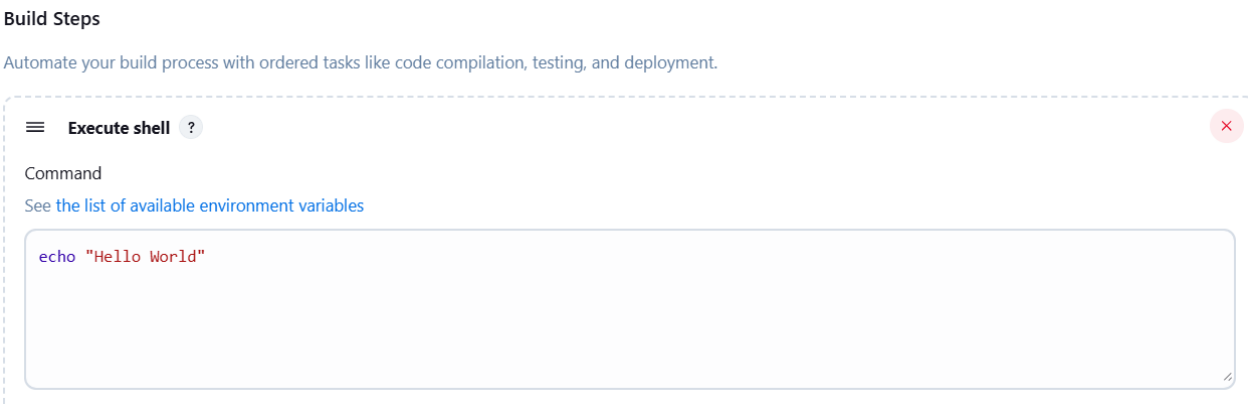
Another option is built periodically like a corn job, For build environment delete the workspace before build starts make sure has clean environment and there are no artifacts by the pervious build on Jenkins.

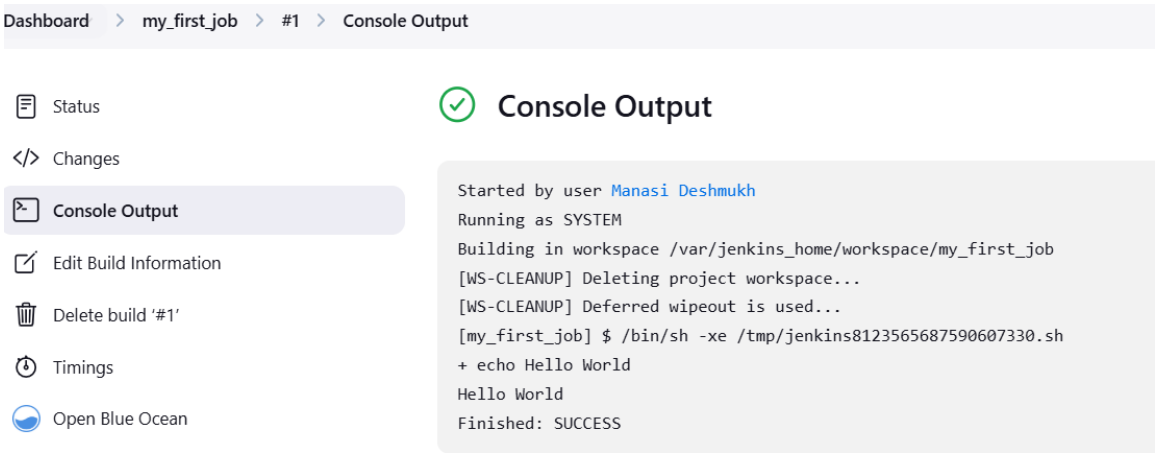


Build steps: what steps work while building a project, post steps after the build like email notification or slack notification by plugin.

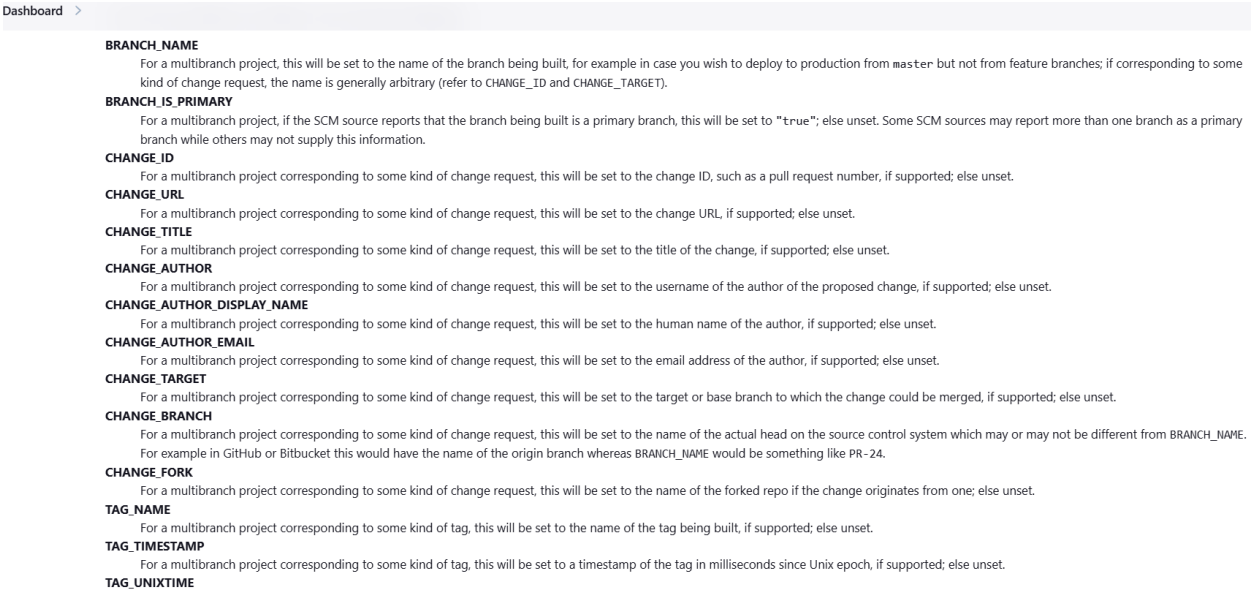


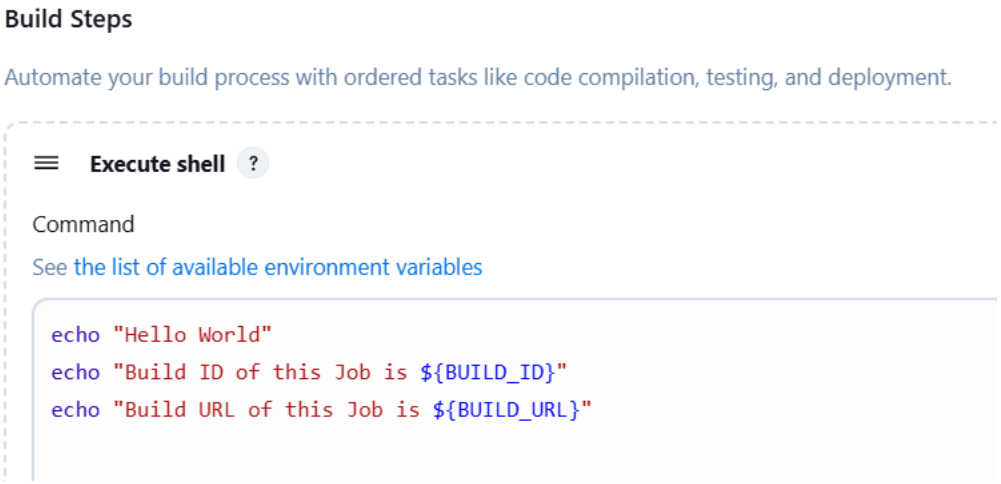
Added first shell command “Hello World” and run the build its successful.

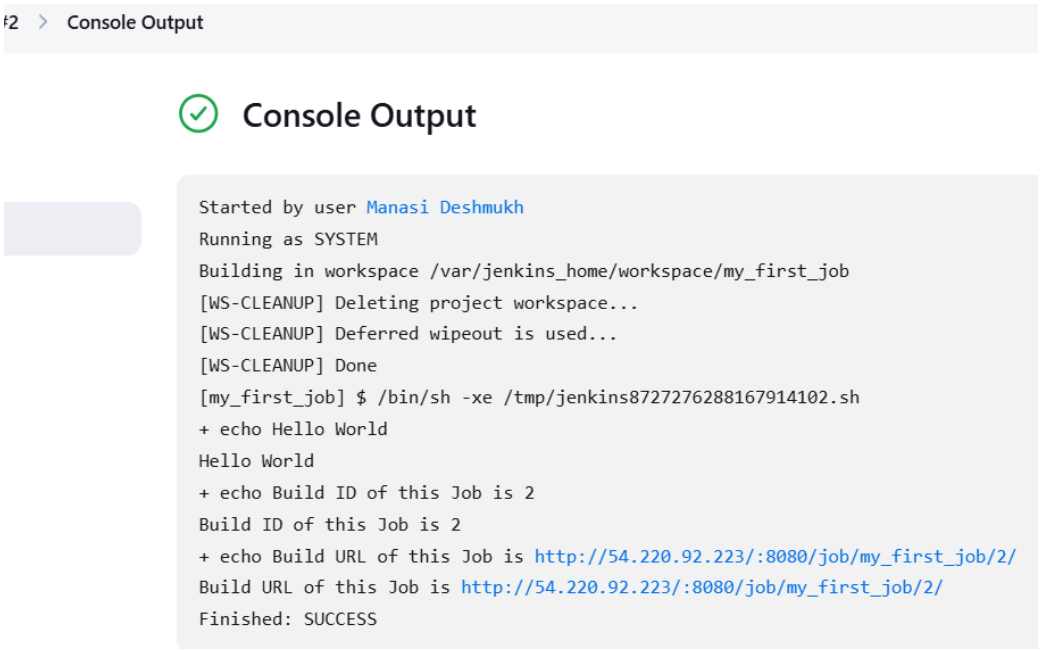




Environment variables: set in Jenkins to view your build, most popular on BUILD_ID will give you current build ID. BUILD_URL gives the URL for the build.

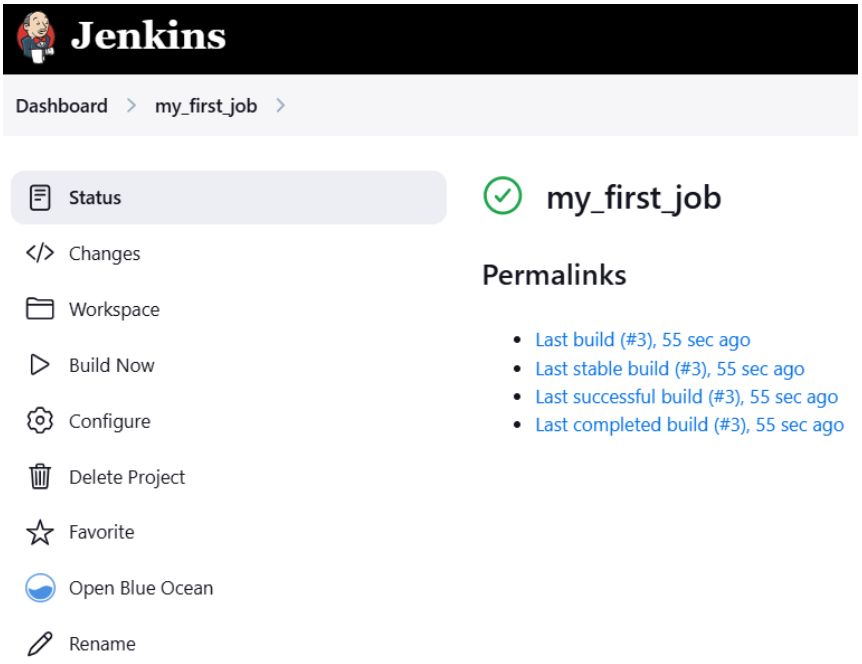


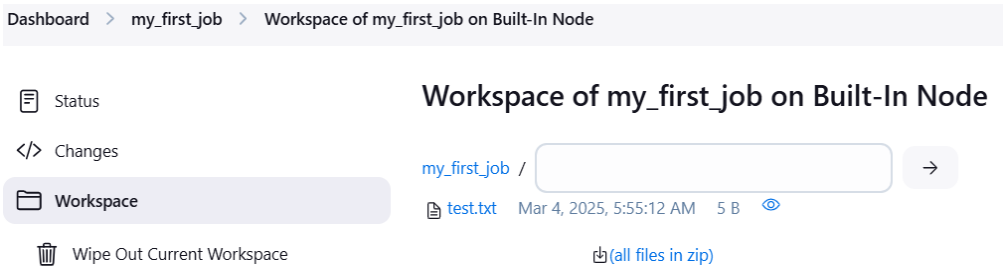




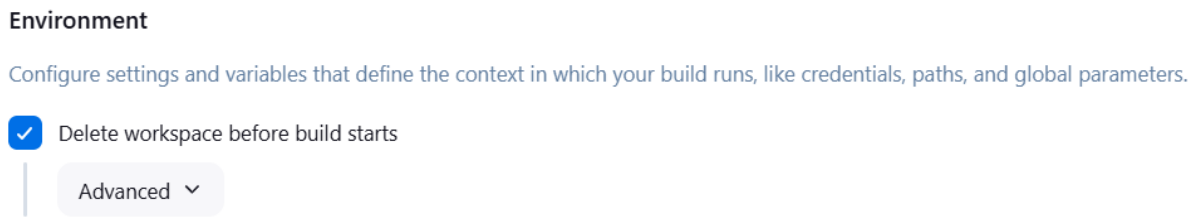
- Standard input is with + command and standard output is the command that has run.

In the workspace we will have the files that have been created and stored during the build.

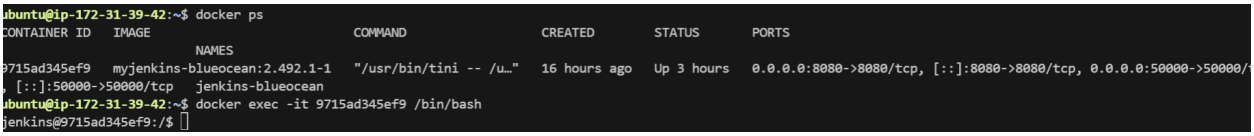


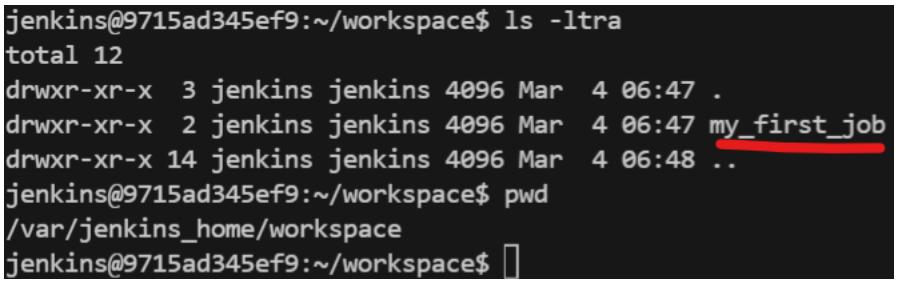


If the Job is run again the file is new file is stored along with older file, you can enable the option of delete the workspace for more cleaner builds.



To Understand the actual **file system of Jenkins**, you can go to the Jenkins master container.



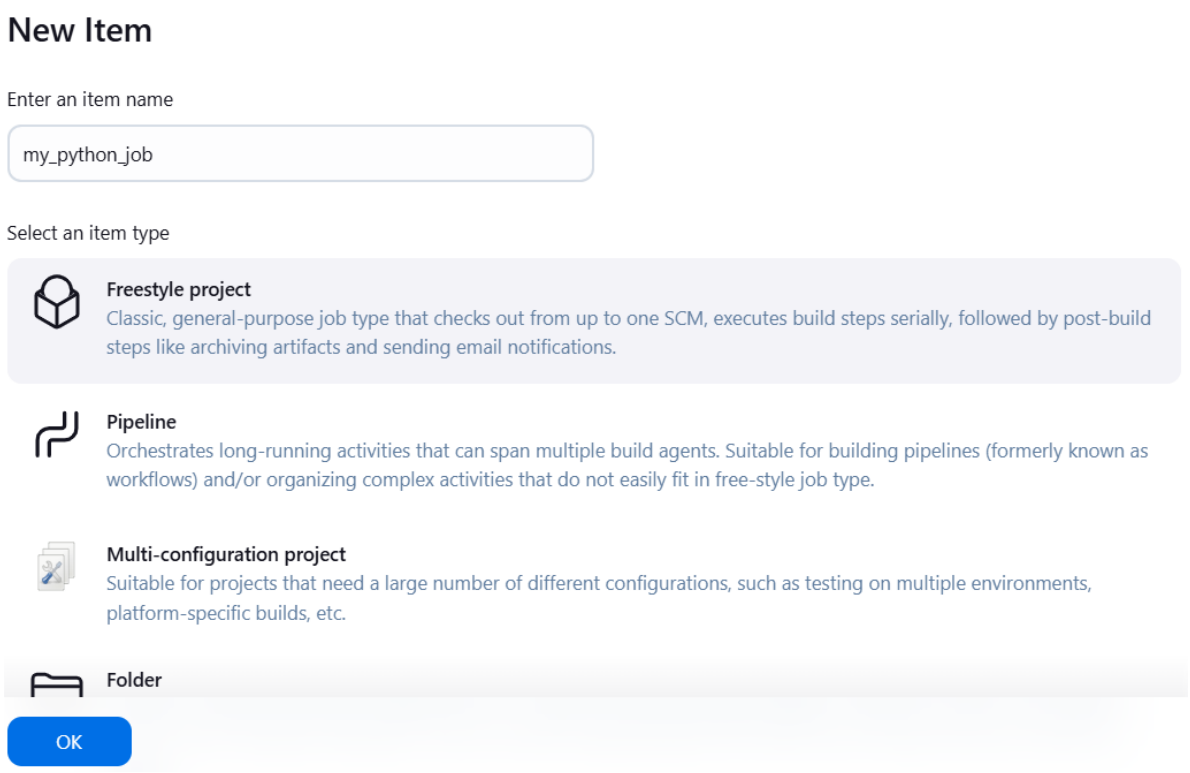


Whenever you create a build, it will have its own workspace, with a folder of the job name.

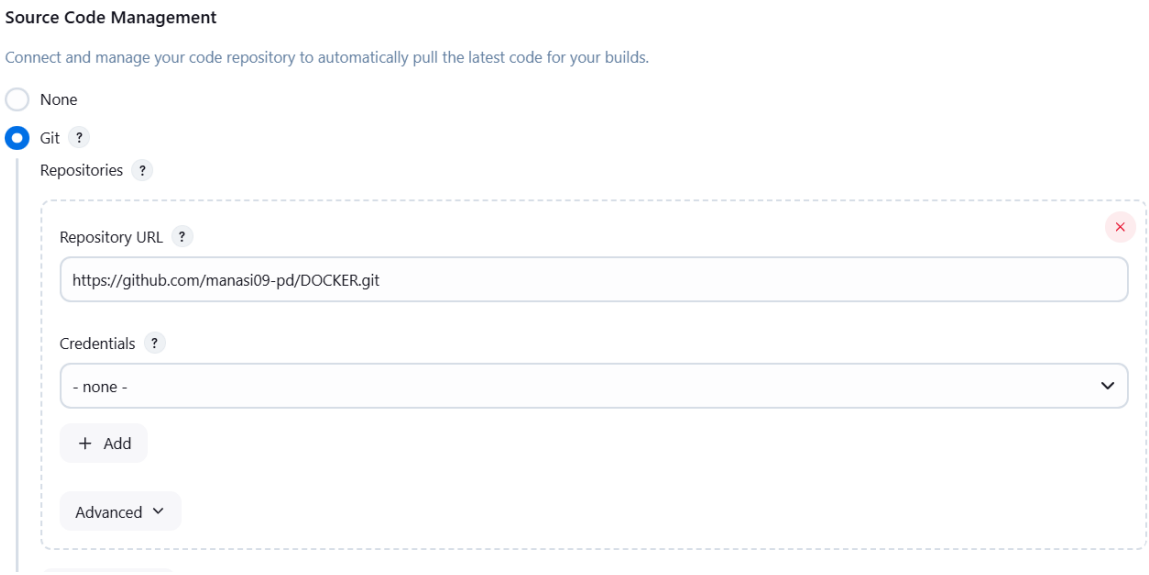
In the home folder you have some other files that you can troubleshoot like plugins, secrets etc.,

```
jenkins@9715ad345ef9:~/workspace/my_first_job$ cd ~
jenkins@9715ad345ef9:~$ pwd
/var/jenkins_home
jenkins@9715ad345ef9:~$ ls -ltra
total 256
drwxr-xr-x   1 root    root     4096 Feb  5 13:33 ..
drwxr-xr-x  10 jenkins jenkins  4096 Mar  3 15:07 war
drwxr-xr-x   4 jenkins jenkins  4096 Mar  3 15:07 .cache
drwxr-xr-x   3 jenkins jenkins  4096 Mar  3 15:07 .java
-rw-r--r--   1 jenkins jenkins    64 Mar  3 15:07 secret.key
-rw-r--r--   1 jenkins jenkins     0 Mar  3 15:07 secret.key.not-so-secret
-rw-r--r--   1 jenkins jenkins   171 Mar  3 15:07 jenkins.telemetry.Correlator.xml
drwxr-xr-x   3 jenkins jenkins  4096 Mar  3 15:07 .groovy
-rw-------   1 jenkins jenkins  1680 Mar  3 15:07 identity.key.enc
-rw-r--r--   1 jenkins jenkins   370 Mar  3 15:07 hudson.plugins.git.GitTool.xml
drwxr-xr-x   2 jenkins jenkins  4096 Mar  3 15:07 userContent
drwxr-xr-x 119 jenkins jenkins 28672 Mar  3 15:19 plugins
drwxr-xr-x   2 jenkins jenkins  4096 Mar  3 15:19 updates
```

Let’s start with a new Job:



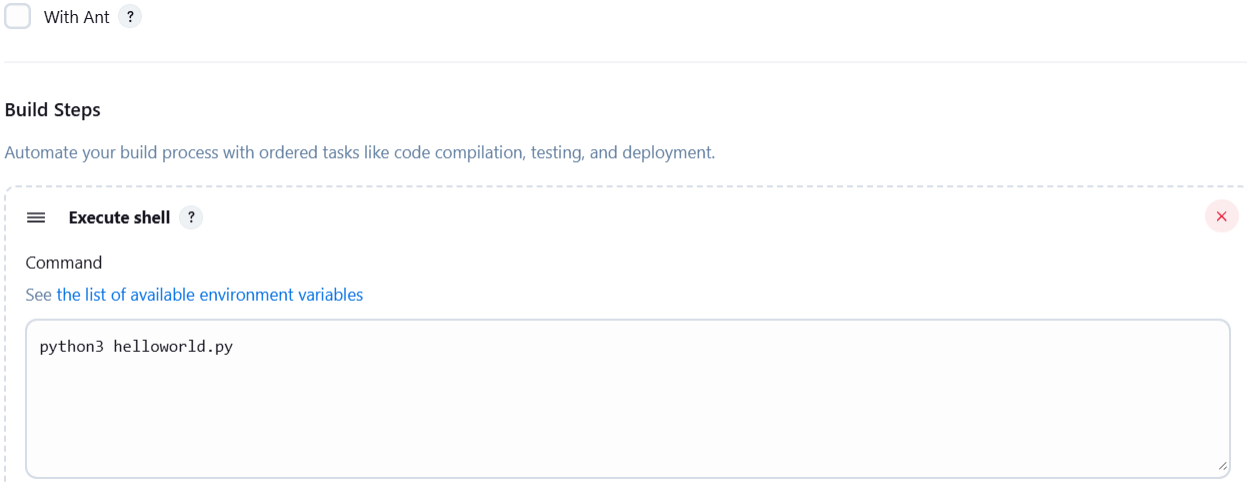
Add the Git repository, for public repository no need for credentials, if your using private repository add the credentials.

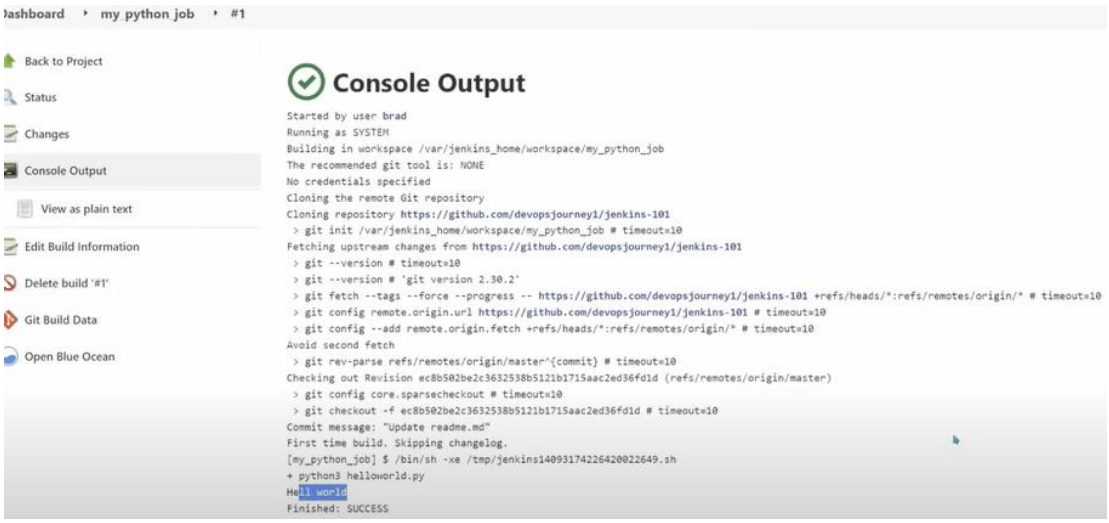


If the Jenkins docker container does not have python3 installed on it, you can install it while switching to root user and execute the docker container and run the commands.

- docker exec -u root -it 9715ad345ef9 /bin/bash
- apt update
- apt install -y python3

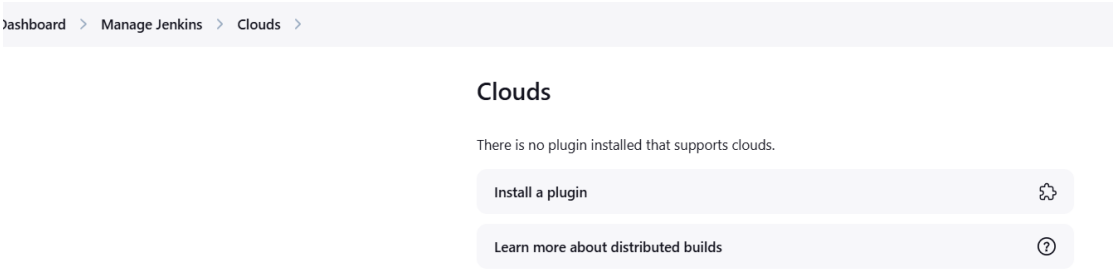
Write the shell command to execute the script and mention the script name.

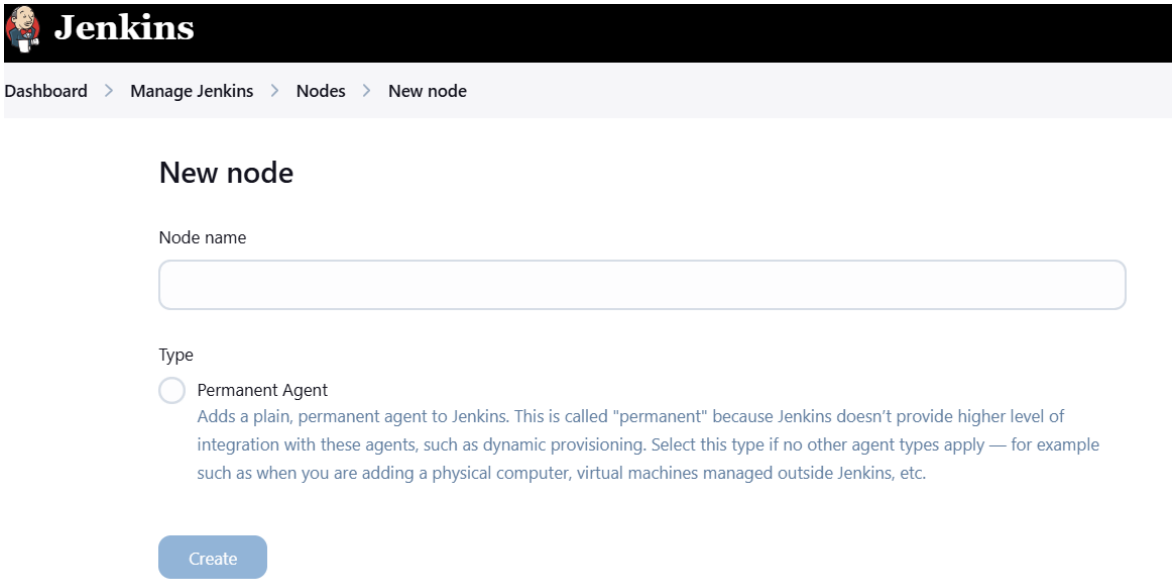




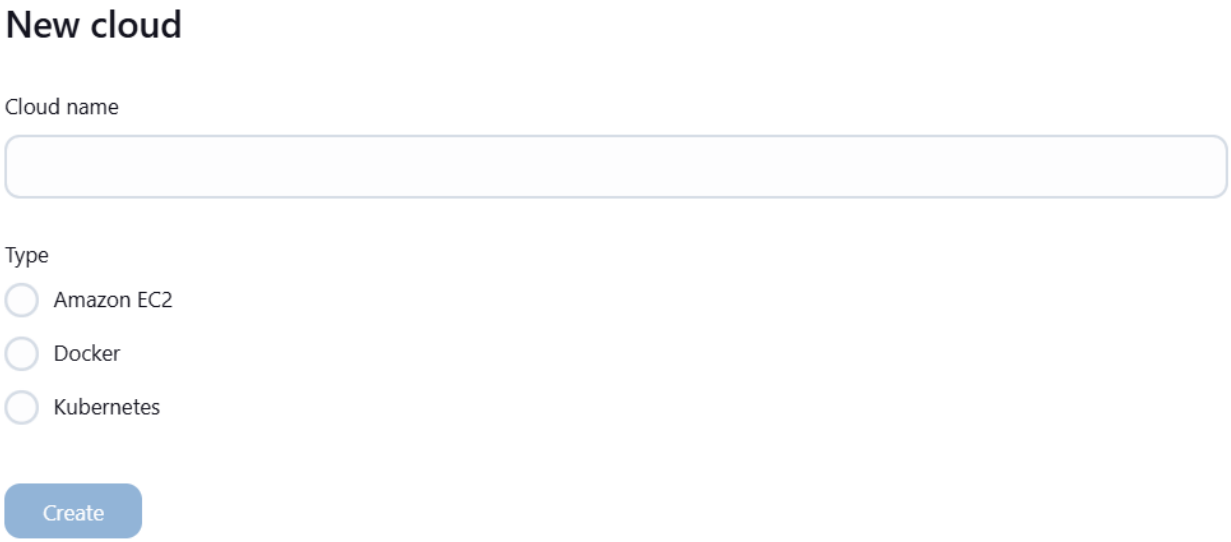
Setting up a cloud Agent:

The new node is basically a new Linux server or windows sort of always available as a permanent agent via SSH, it's a older method new way to set a cloud agent like Docker, Kubernetes and Amazon EC2. Install the plugin Docker.



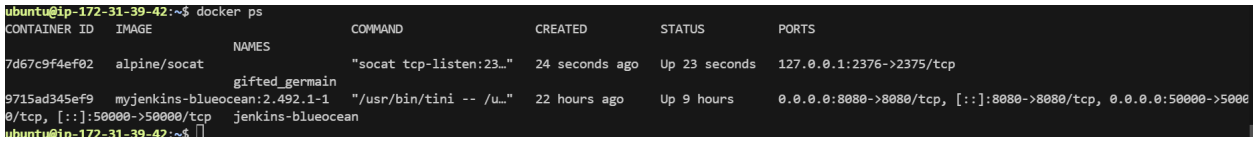


Let's set up the docker cloud, First thing you need to set uri of the host that is running docker, For Jenkins 127.0.0.1:2376 and port for docker, or add the remote server IP that has the docker installed on it.



Create a **new alpine container** on Jenkins Master with the command below.

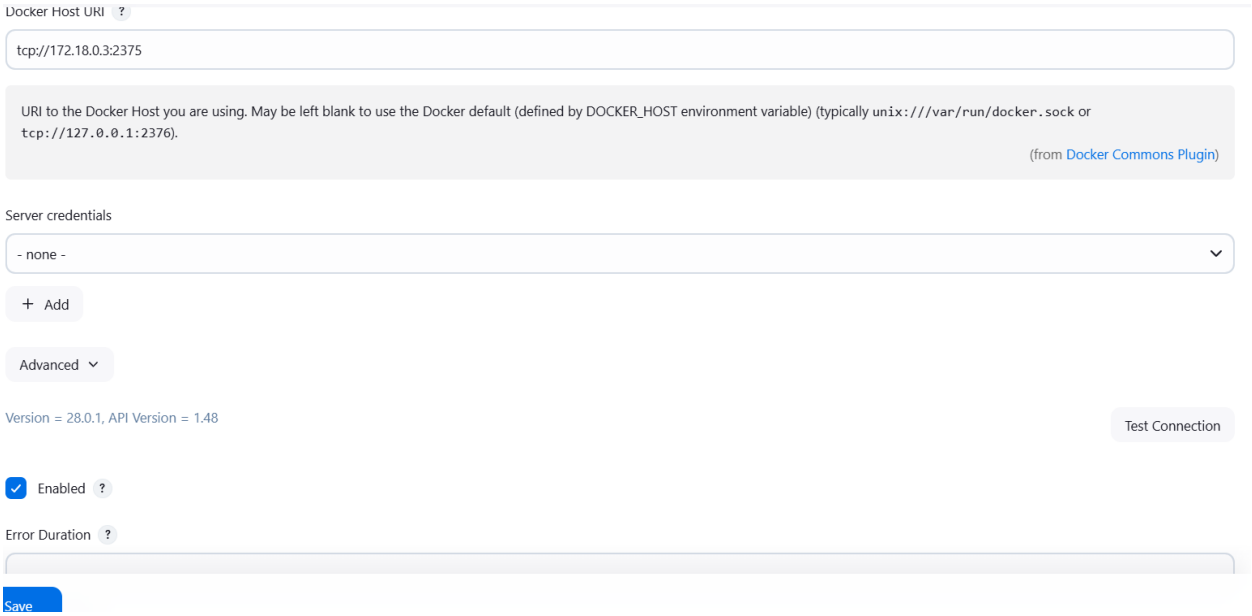
- docker run -d --restart=always -p 127.0.0.1:2376:2375 --network jenkins -v /var/run/docker.sock:/var/run/docker.sock alpine/socat tcp-listen:2375,fork,reuseaddr unix-connect:/var/run/docker.sock



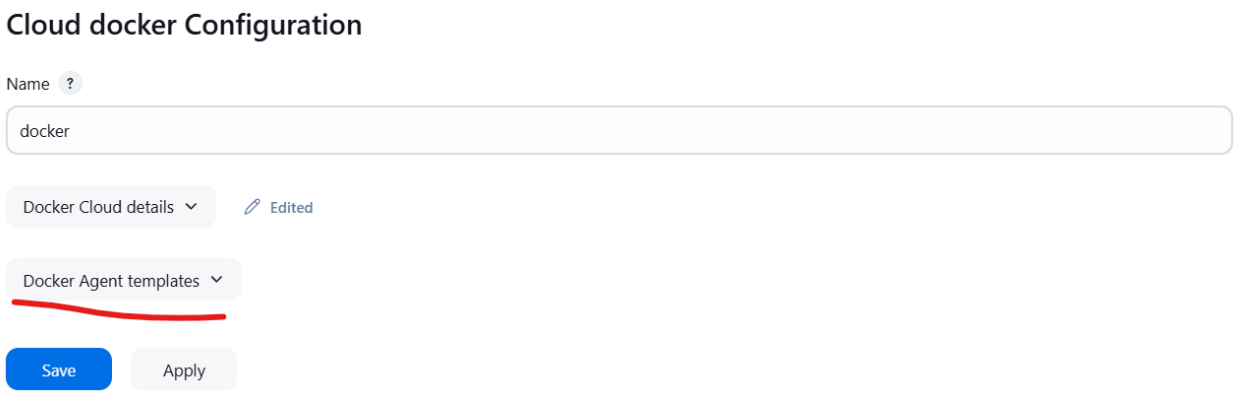
Perform the docker inspect on the docker container. To get the IP address of the container.

```
ubuntu@ip-172-31-39-42:~$ docker inspect 7d67c9f4ef02
[
    {
        "Id": "7d67c9f4ef022399d68cda3551713d7052190521d969aa7ef7c0c78a61a22e15",
        "Created": "2025-03-04T12:41:22.842938289Z",
        "Path": "socat",
        "Args": [
            "tcp-listen:2375,fork,reuseaddr",
            "unix-connect:/var/run/docker.sock"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
```

Add the IP address to cloud agent configuration. And test the connection.

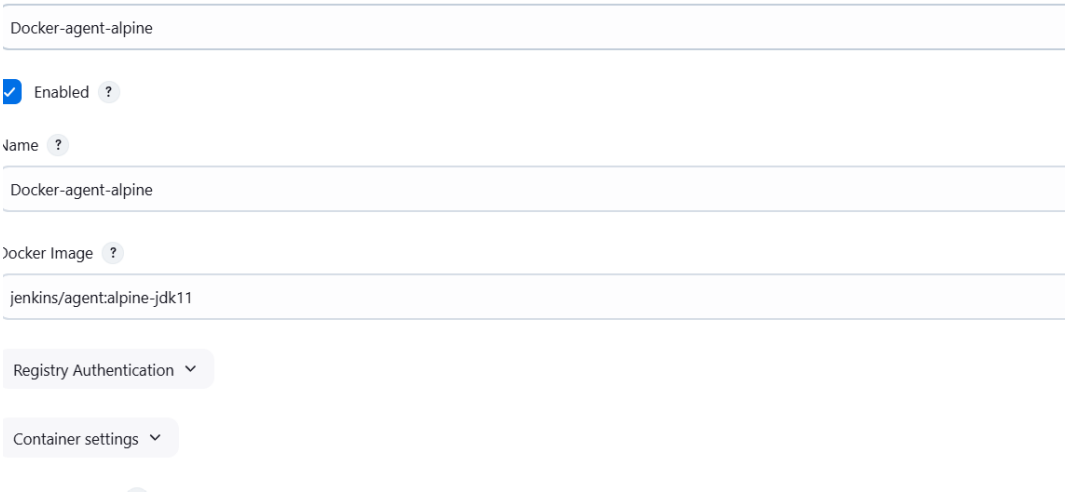


Set up the docker agent template.

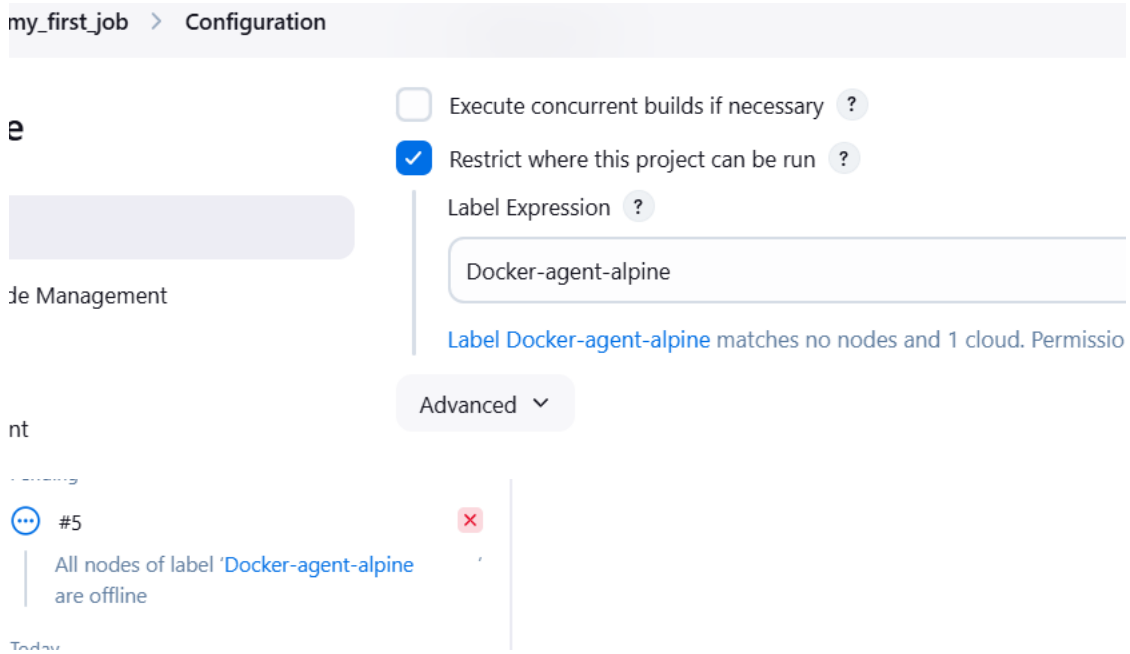


Labels are added so that the master decides which agent should be selected for a particular build. I'm using the image from Docker Hub. If you want to use your own image you can point it to the docker hub or set up a private repository.

Instance capacity is how many agents you want to spin up. **Remote File System Root** is where your root folder gets created.



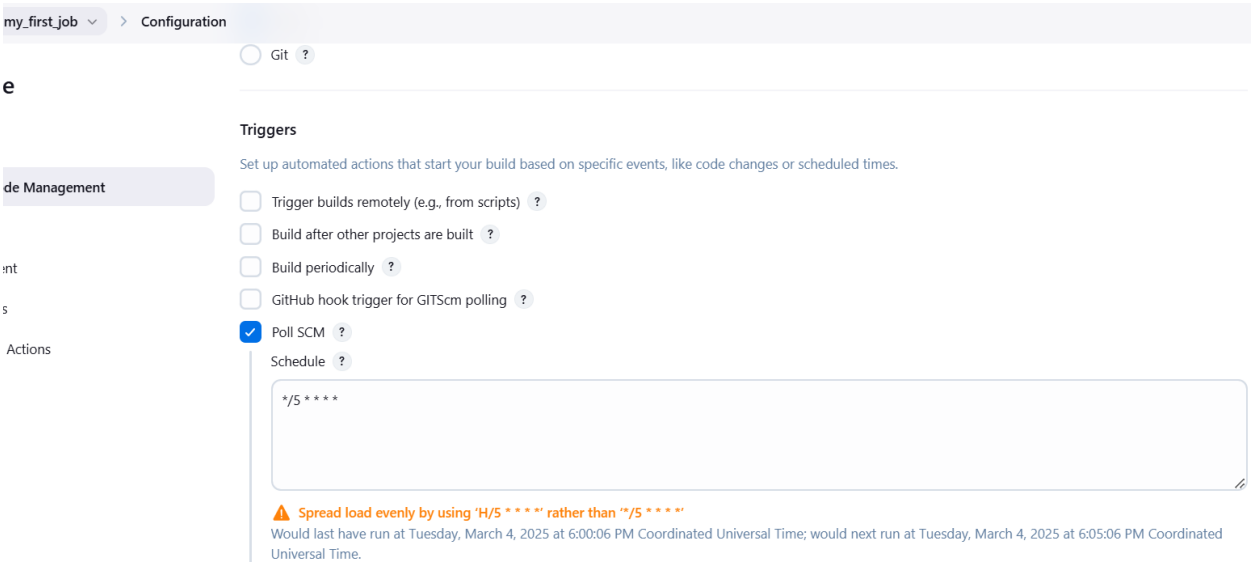
Configuring a job with the agent we just created and trigger new the build.



The agent is not started so build is stuck, let's troubleshoot it by going to the nodes and triggering the launch agent.

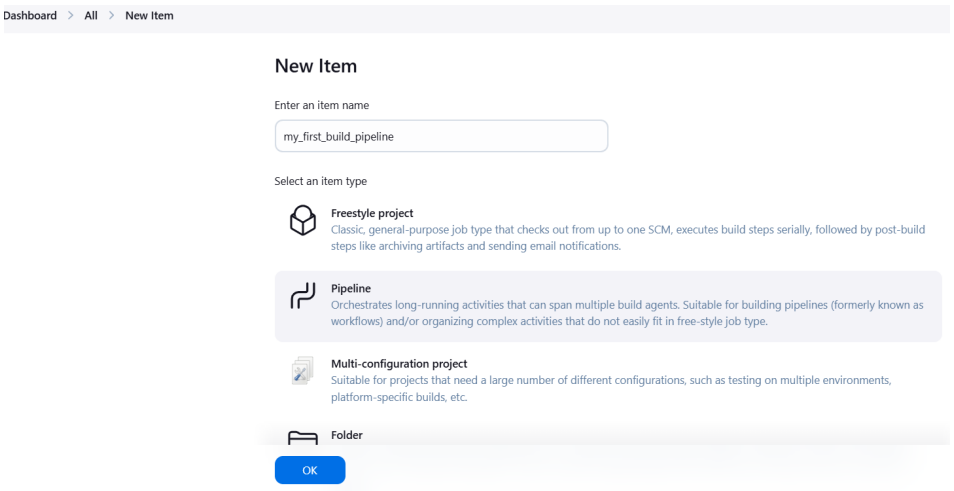
Launched the agent and the build used the agent docker-agent-alpine.

Setting the automatic triggers in the builds: **Poll SCM** is the method to monitor your GitHub repository check for any changes. It's better to use this method if your Jenkins is set up against a firewall like Cron Job it would run when a change is made.



Setting CI/CD pipeline in a Declarative pipeline:

In the configuration you don't have a lot of options, you can define your pipeline script in the Jenkins UI page, or you can use Jenkins file on your source repository. Point the pipeline over to the Jenkins file and it will take it there.



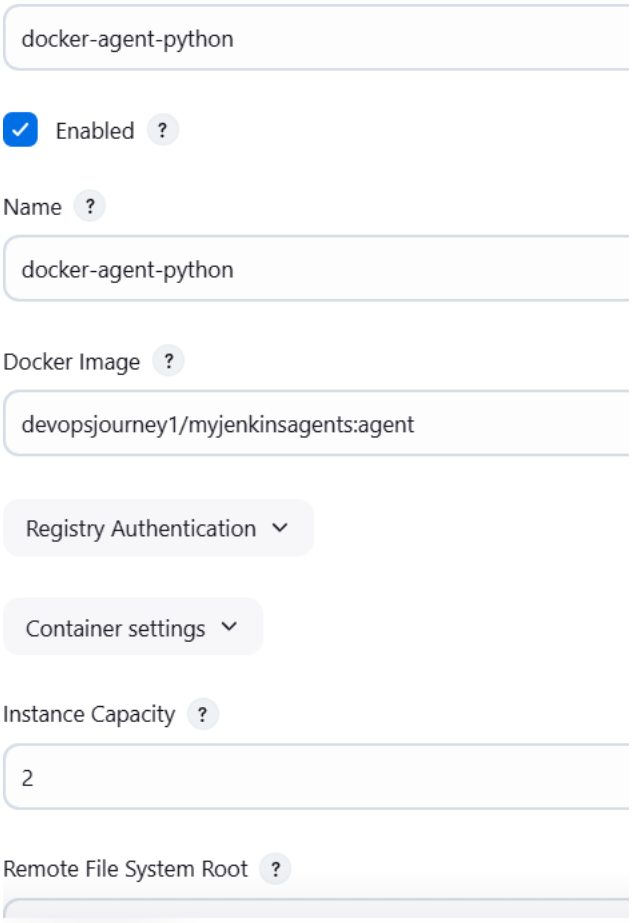
I have written a declarative pipeline in groovy language, which is easy to collaborate and write.

Setting up a new docker agent for python I have pulled the image of python from docker hub to Jenkins server and built the image into the container using the docker file for python.

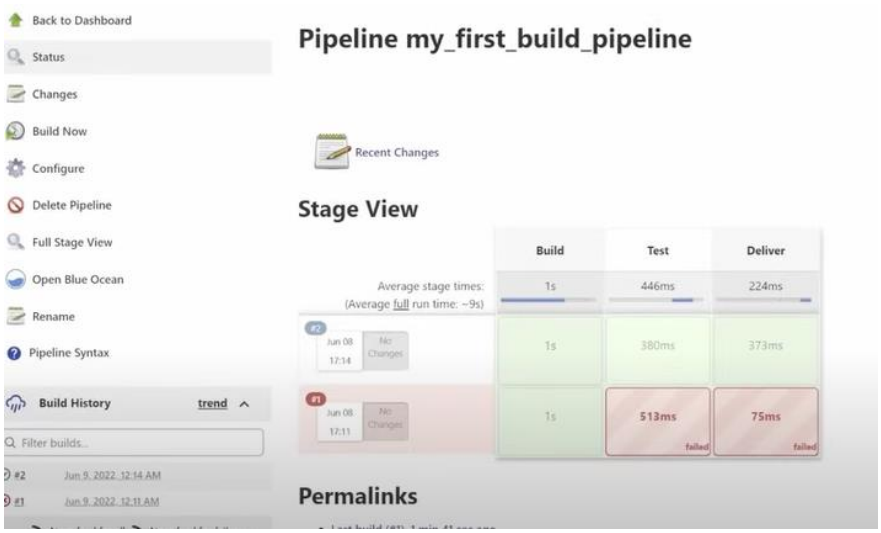
```
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker/python$ docker images
REPOSITORY                          TAG             IMAGE ID       CREATED          SIZE
devopsjourney1/myjenkinsagents      python          3093cc90c354   54 seconds ago   198MB
myjenkins-blueocean                 2.492.1-1       ecbd36856e47   2 days ago       811MB
alpine/socat                        latest          5c51ae7b8113   4 days ago       9.1MB
jenkins/agent                       alpine-jdk11    eb03a60b3dba   5 months ago     134MB
devopsjourney1/myjenkinsagents      <none>          fe573725a2e4   2 years ago      196MB
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker/python$ docker pull devopsjourney1/myjenkinsagents:python
python: Pulling from devopsjourney1/myjenkinsagents
Digest: sha256:b8aa6f53acd4ee8a304fcc93f6a667d5acda02df977647f2cd5e15e3854d7941
Status: Downloaded newer image for devopsjourney1/myjenkinsagents:python
docker.io/devopsjourney1/myjenkinsagents:python
```

```
ubuntu@ip-172-31-39-42:~/DOCKER/jenkins-docker/python$ docker build -t devopsjourney1/myjenkinsagents:python .
[+] Building 9.8s (7/7) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 134B
 => [internal] load metadata for docker.io/jenkins/agent:alpine-jdk11
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/jenkins/agent:alpine-jdk11
 => [2/3] RUN apk add python3
 => [3/3] RUN apk add py3-pip
 => exporting to image
 => => exporting layers
 => => writing image sha256:3093cc90c354fbe3df41b05dc2fa3b07c0c5b764e151128cfebe4ed6029ebb83
```

Added one more docker agent on the docker cloud.

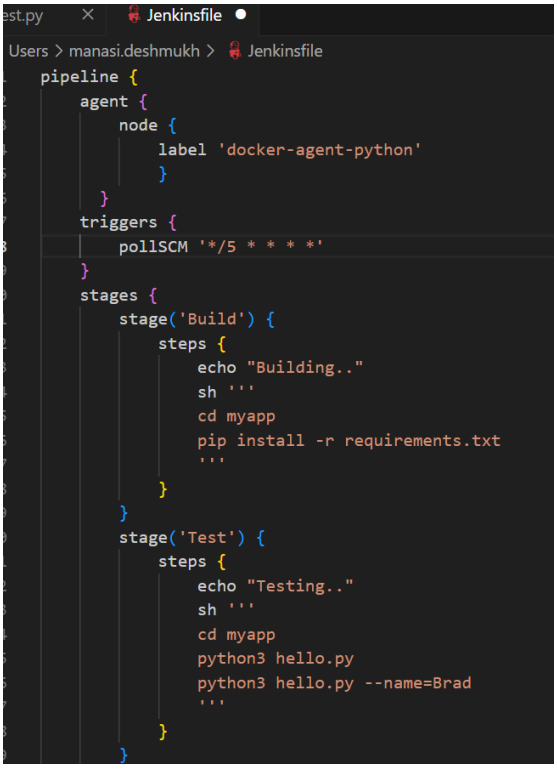


After setting up the docker agent, run the build pipeline. The pipeline looks quite different, like it's separated into three different stages, like **BUILD, TEST** and **DELVIER**. Test and deliver stage failed you can check the logs. If the Test stage failed it's an upstream stage so the deliver stage failed, once this is fixed all the stages are successfully built.



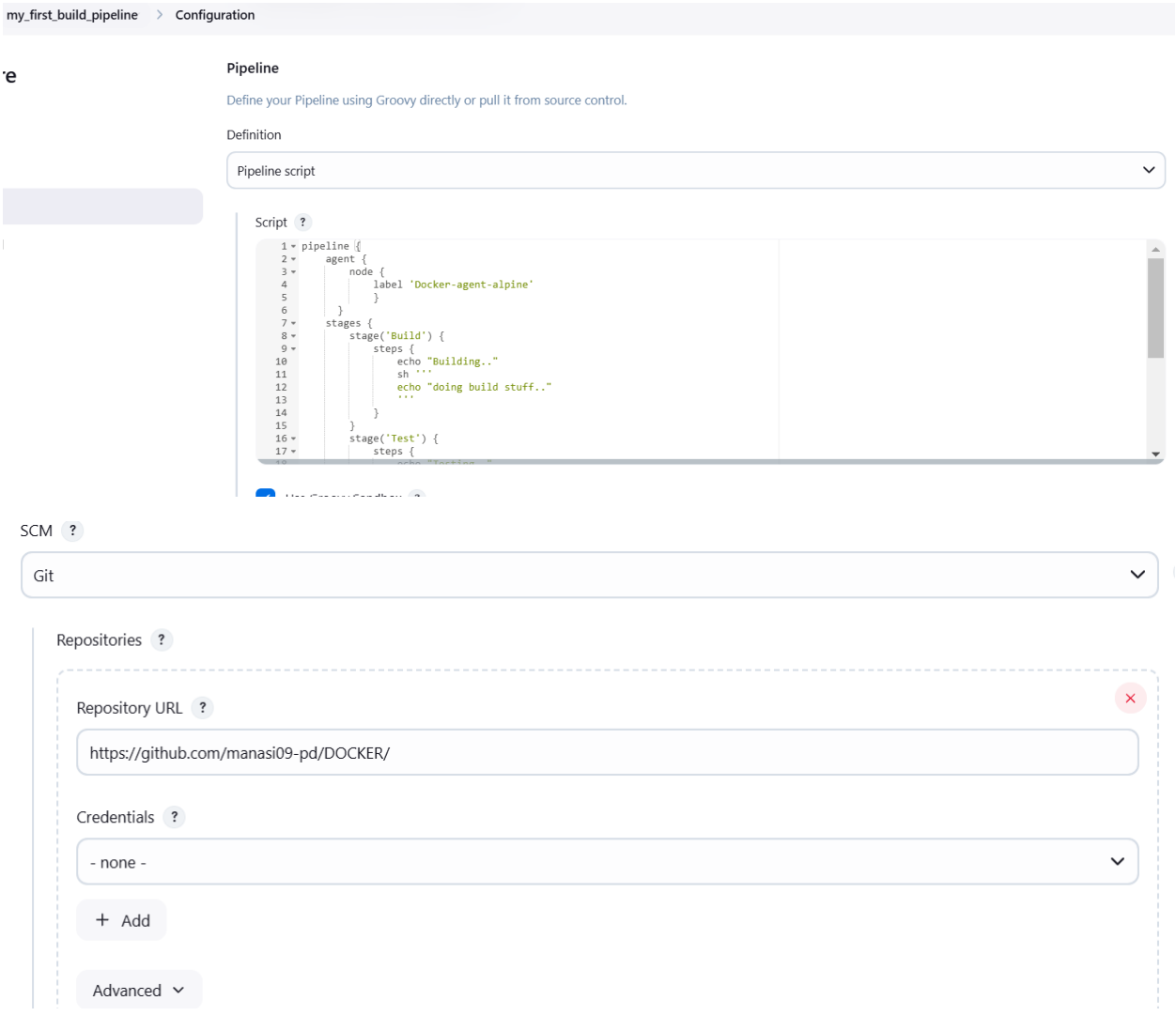
Let's use a Jenkins file to run a pipeline job:

First let's build a Jenkins file with different stages like mentioning the Agent, Poll how frequently the job will be triggered, then stages like build, test and deliver.

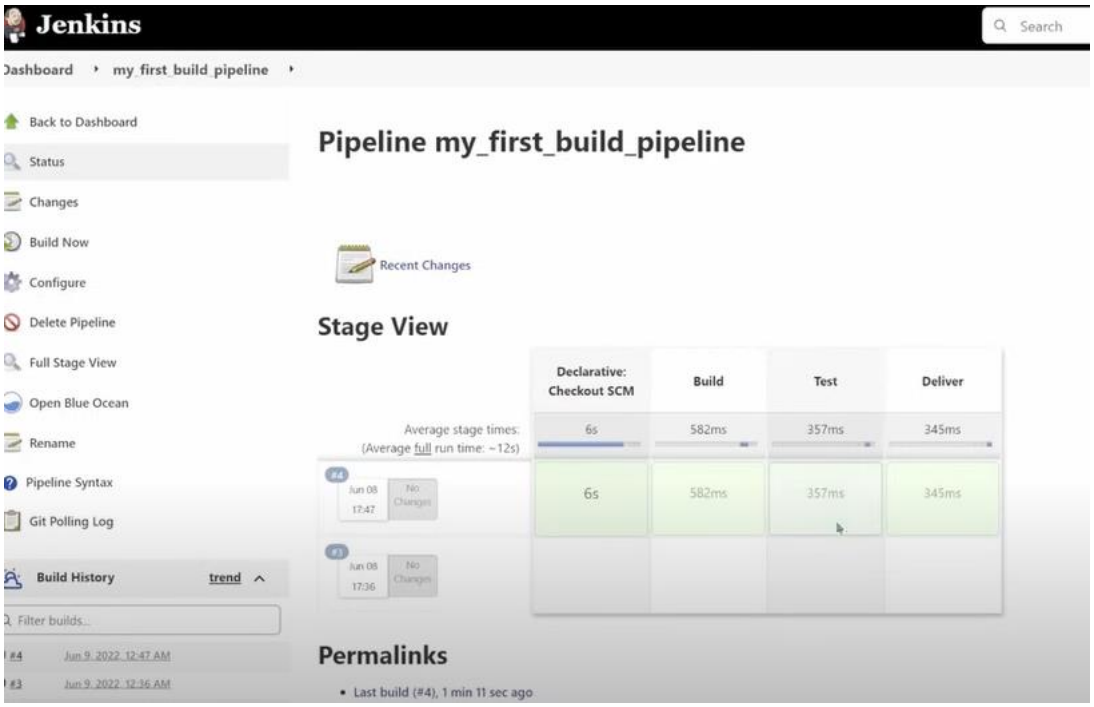


```
    }
    stage('Test') {
        steps {
            echo "Testing.."
            sh '''
            cd myapp
            python3 hello.py
            python3 hello.py --name=Brad
            '''
        }
    }
    stage('Deliver') {
        steps {
            echo 'Deliver....'
            sh '''
            echo "doing delivery stuff.."
            '''
        }
    }
}
```

Changing my first_build_pipeline pipeline script to SCM.



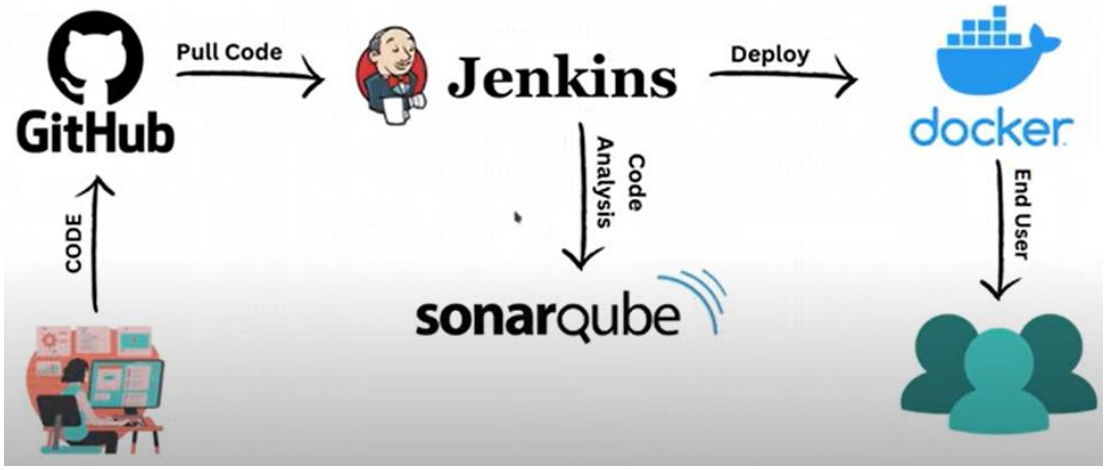
Every 5 minutes the job will run, it added one more stage that is git checkout since it will take the pipeline script from SCM, will add the git checkout stage.



PROJECT

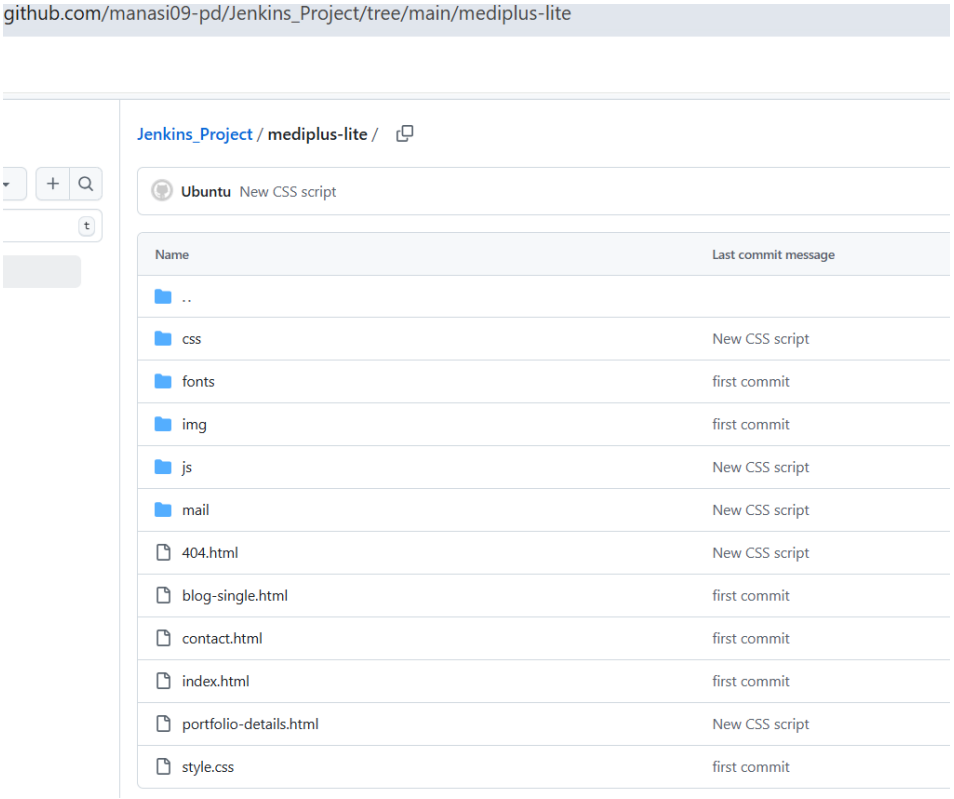
In the below CI/CD pipeline Jenkins will pull the code from the GitHub and we will test the code on SonarQube which is a static code analysis tool where you can scan the vulnerabilities of the code also, we can add the rules for scanning.

After the code is tested it will be deployed on the docker. For this you will need three EC2 instances for Jenkins, SonarQube and Docker. After deploying it on the docker we will verify by accessing it through browser.



I have downloaded basic html CSS template from the internet for a website. I have extracted and pushed the code to my GitHub repository, please use the link below to access the scripts for the web page.

Link: https://github.com/manasi09-pd/Jenkins_Project.git

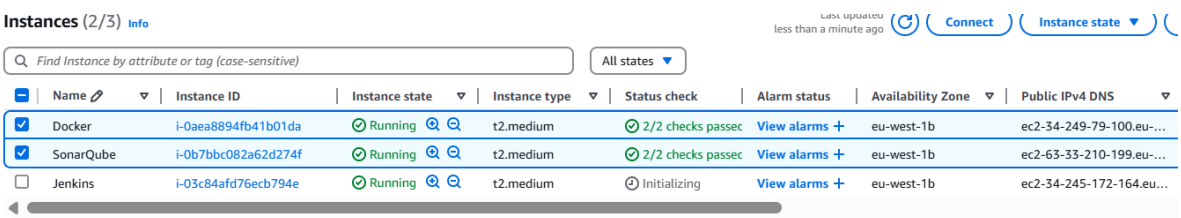


Create Three EC2 instances for Jenkins, SonarQube and Docker.

Jenkins EC2 instance is created, with Ubuntu OS and t2.medium storage with 2VCPU and 4 GB memory and created a new SSH key pair.

SonarQube instance is created, with Ubuntu OS, it will consume lot of memory at least have 2GB memory, select the same key.

Docker instance is created, with Ubuntu OS and t2.medium storage with 2VCPU and 4 GB memory and created a new SSH key pair.

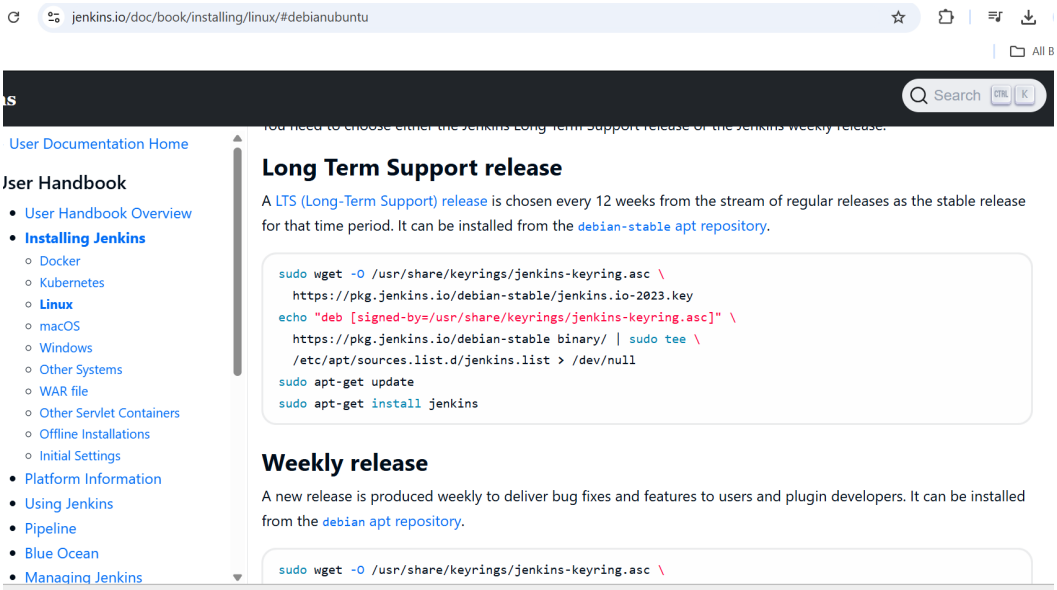


Login to your Jenkins instance with PEM key file and update the packages using the below commands.

```
>ssh -i "7-3-2025.pem" ubuntu@34.245.172.164
```

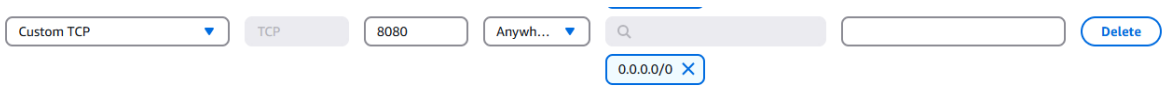
- sudo apt-get update
- sudp spt-get upgrade
- sudo apt-get install openjdk-11-jre (Java installation is necessary for Jenkins).

Follow Jenkins.io page to install Jenkins on your preferred operating system.




```
ubuntu@ip-172-31-40-239:~$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
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
sudo apt-get update
sudo apt-get install jenkins
--2025-03-07 18:23:45--  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
Resolving pkg.jenkins.io (pkg.jenkins.io)... 199.232.26.133, 2a04:4e42:43::645
Connecting to pkg.jenkins.io (pkg.jenkins.io)|199.232.26.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3175 (3.1K) [application/pgp-keys]
Saving to: ‘/usr/share/keyrings/jenkins-keyring.asc’

/usr/share/keyrings/jenkins-keyring.asc    100%[=============================================
```

Allow default port 8080 to access Jenkins web page.



Or run Jenkins on an EC2 instance with Docker, follow the commands given below.

Install the Docker packages.

- sudo apt-get upgrade
- sudo apt-get update
- sudo apt-get install docker.io -y
- sudo systemctl start docker
- sudo usermod -a -G docker $(whoami)

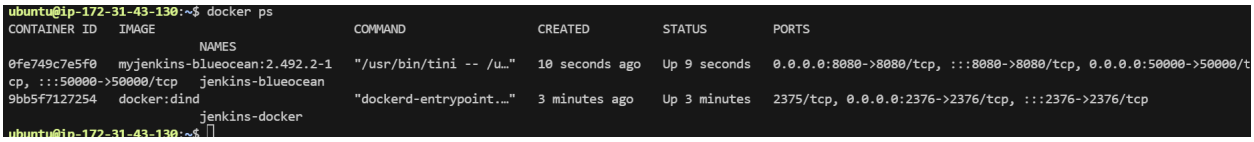
Next, let's run Jenkins using Docker:

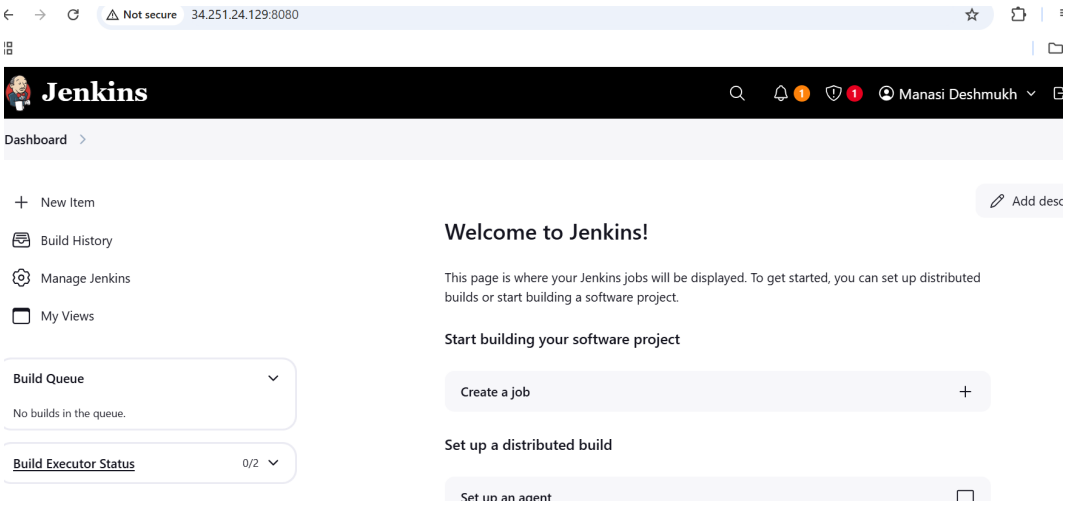
- docker network create Jenkins

- docker run --name jenkins-docker --rm --detach \
- --privileged --network jenkins --network-alias docker \
- --env DOCKER_TLS_CERTDIR=/certs \
- --volume jenkins-docker-certs:/certs/client \
- --volume jenkins-data:/var/jenkins_home \
- --publish 2376:2376 \
- docker:dind --storage-driver overlay2

Create a Docker file and build the docker images and run the docker container, Follow the Jenkins page.

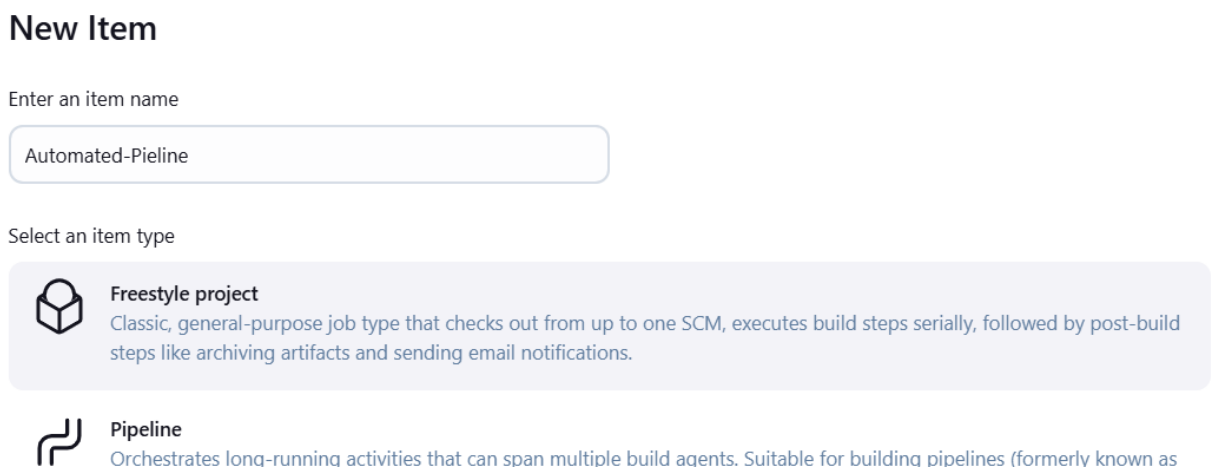
Link: https://www.jenkins.io/doc/book/installing/docker/



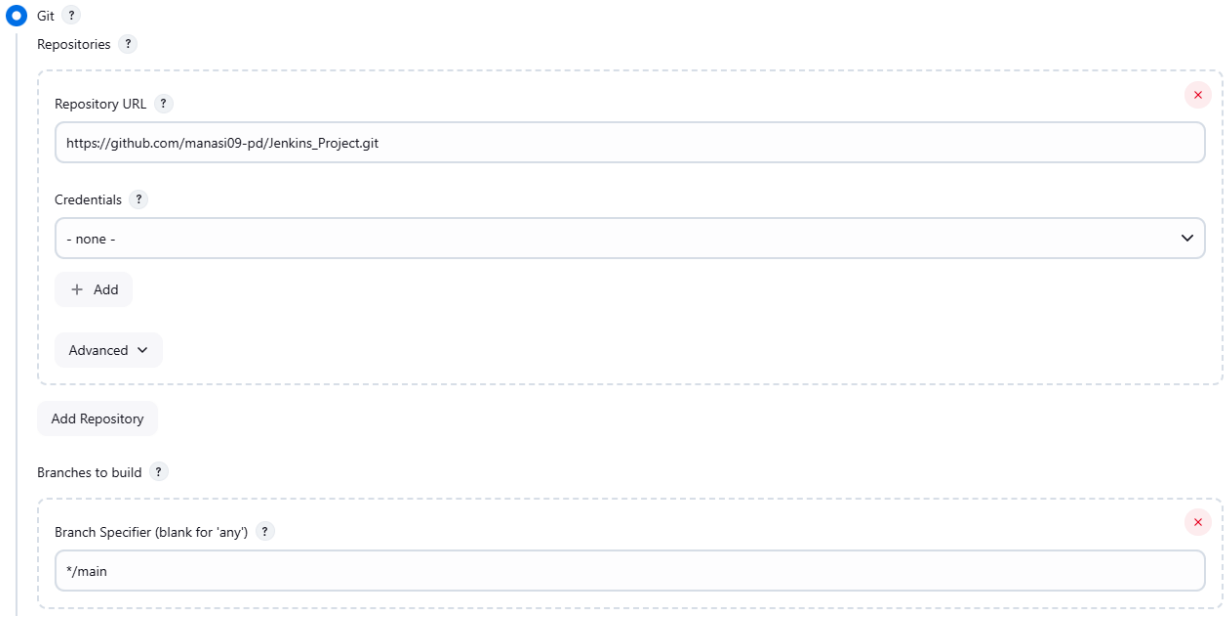


Next, we will pull the code from Jenkins and test it on SonarQube instance if it is fine we will deploy it on the docker instance.

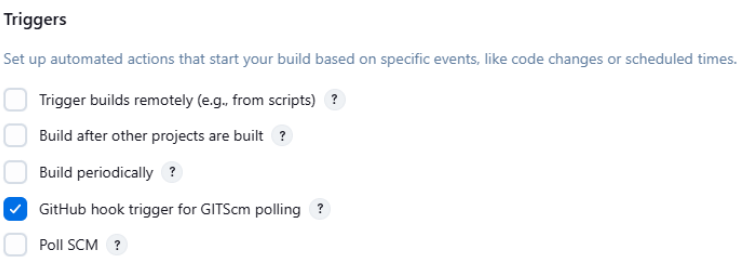
Create pipeline in Jenkins.



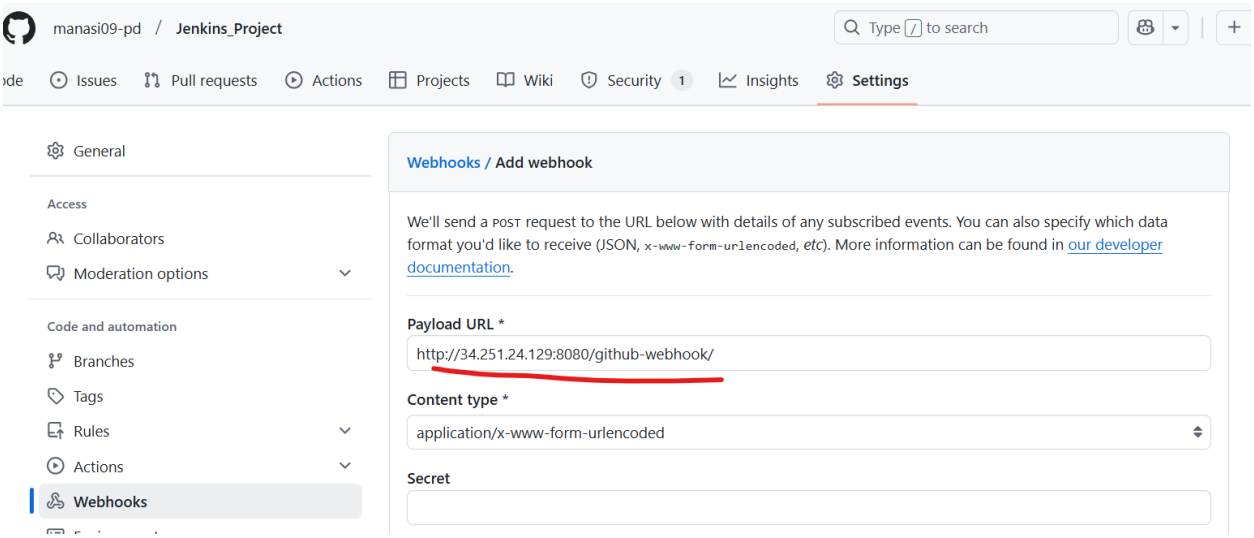
Added my git repository in the source code management of Jenkins.



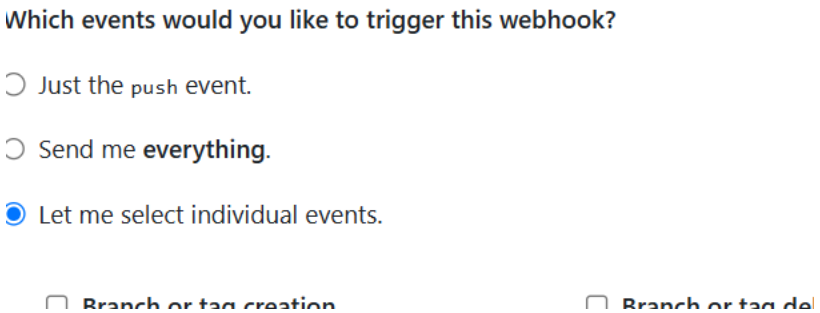
Also select the trigger, whenever there is a change in the repository the build gets triggered.



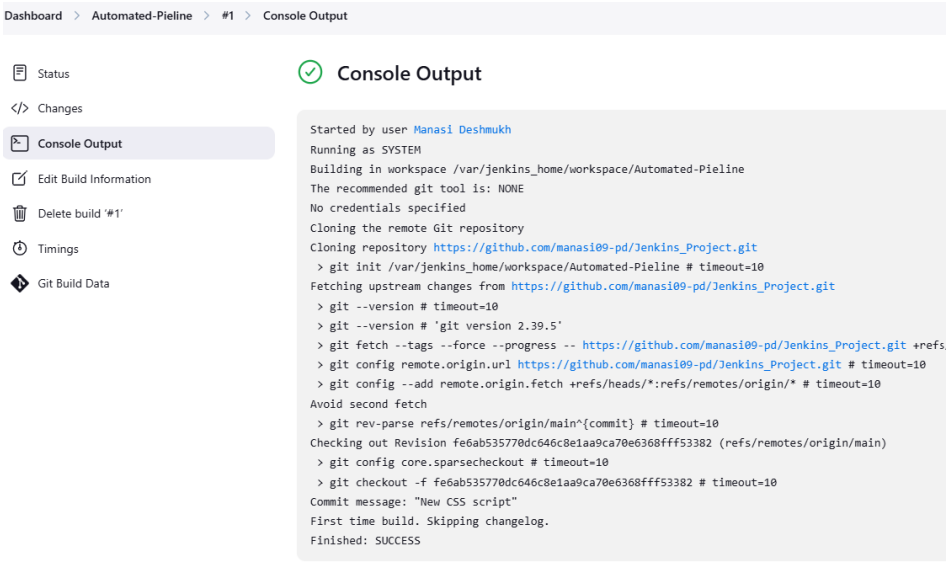
Add the webhook from the GitHub account, add the Jenkins URL.



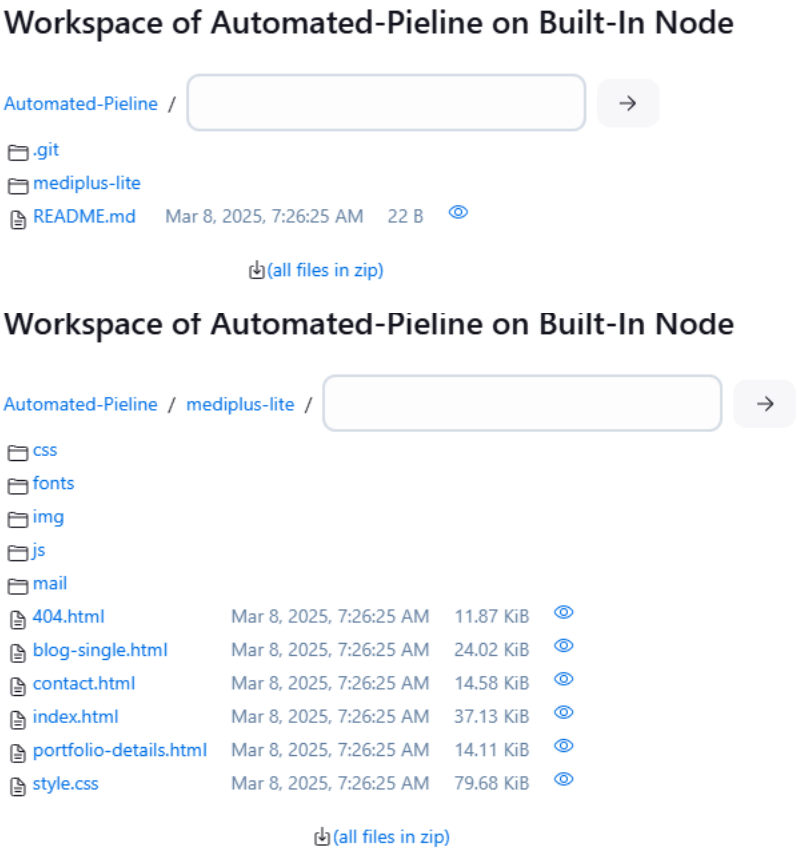
Select webhook triggers like pushes and pull requests.



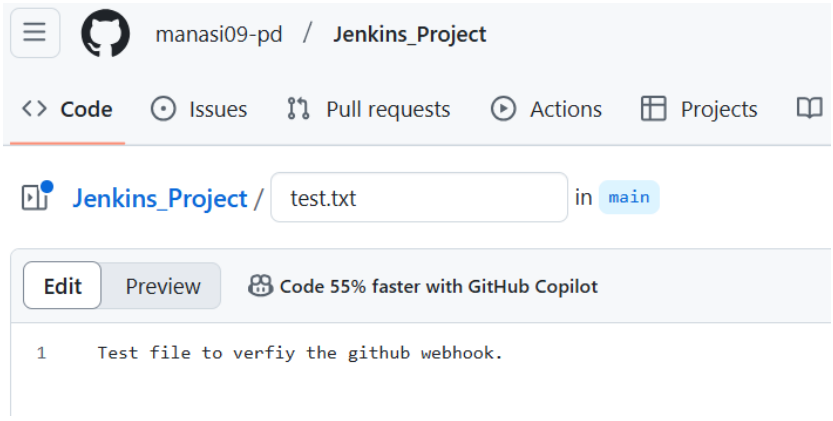
Manually triggered the build.



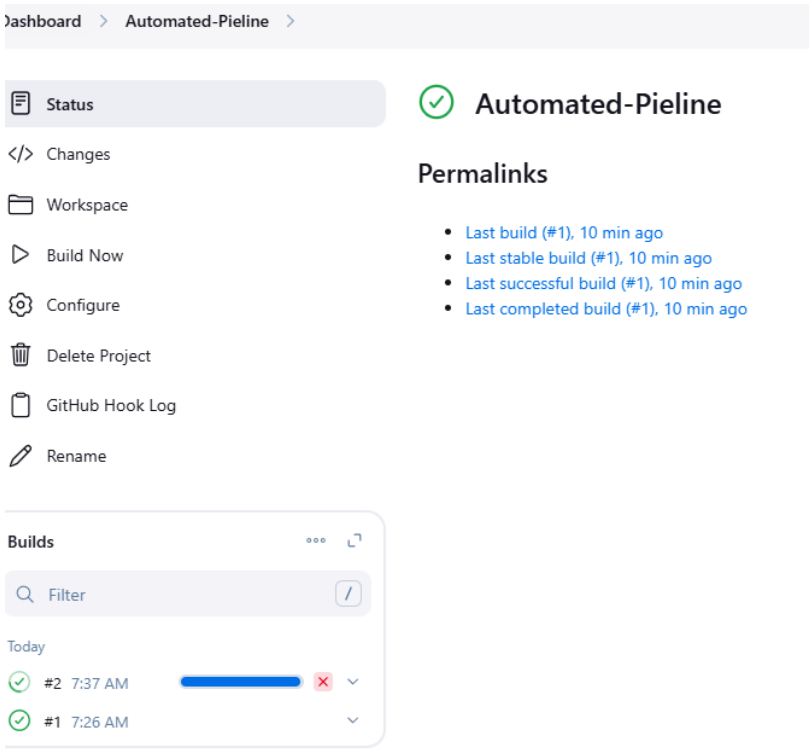
I have two folders, one is. git folder and other one is CSS folder, in that I have multiple files and folders.



I have created a test file in GitHub to check if the build gets triggered for new push.



As you can see, the pipeline is triggered. So whenever the developer changes the code, adds a new file or any changes pushed the Jenkins build is automatically triggered from webhook.



Login to SonarQube and Jenkins for instance and change the hostname. Update the repository and for SonarQube to be installed on the instance we need java 11 with the below command.

- sudo apt-get install openjdk-17-jre

```
ubuntu@ip-172-31-34-61:~$ sudo hostnamectl set-hostname sonarqube
ubuntu@ip-172-31-34-61:~$ /bin/bash
ubuntu@sonarqube:~$
```

- Install PostgreSQL

```
curl https://www.postgresql.org/media/keys/ACCC4CF8.asc | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/apt.postgresql.org.gpg >/dev/null
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt (lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
sudo apt update
sudo apt install postgresql postgresql-contrib
sudo systemctl status postgresql
sudo -u postgres psql
postgres=# CREATE ROLE sonaruser WITH LOGIN ENCRYPTED PASSWORD 'your_password';
postgres=# CREATE DATABASE sonarqube;
postgres=# \q
```

- exit

Install sonarqube

- wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.6.1.59531.zip
- wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.6.1.59531.zip

```
ubuntu@sonarqube:~$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.6.1.59531.zip
--2025-03-08 10:47:22--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.6.1.59531.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 18.66.171.117, 18.66.171.63, 18.66.171.104, ..
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|18.66.171.117|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 290033351 (277M) [binary/octet-stream]
Saving to: ‘sonarqube-9.6.1.59531.zip’

sonarqube-9.6.1.59531.zip                 100%[=============================================================

2025-03-08 10:47:31 (31.8 MB/s) - ‘sonarqube-9.6.1.59531.zip’ saved [290033351/290033351]

ubuntu@sonarqube:~$ ls
sonarqube-9.6.1.59531.zip
```

- unzip -q sonarqube-9.6.1.59531.zip

Move the files to the /opt/sonarqube directory.

- sudo mv sonarqube-9.6.1.59531 /opt/sonarqube

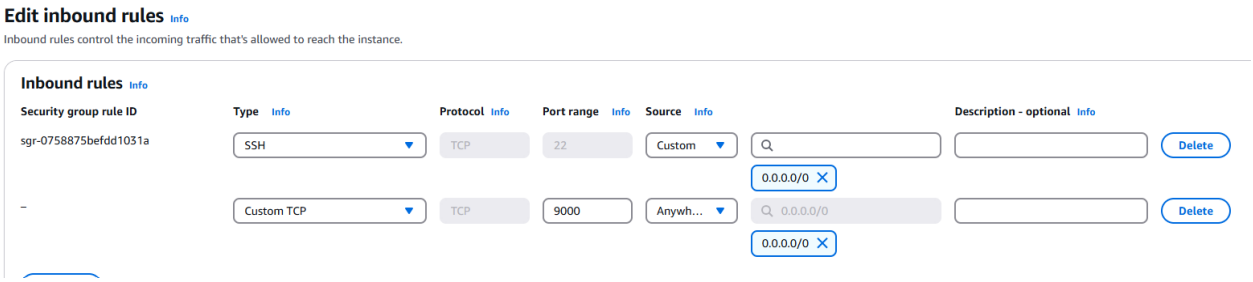
```
ubuntu@sonarqube:~$ ls
sonarqube-9.6.1.59531  sonarqube-9.6.1.59531.zip
ubuntu@sonarqube:~$ sudo mv sonarqube-9.6.1.59531 /opt/sonarqube
```

IN the /opt/sonarqube path there is bin folder which has options for Linux, windows and MacOS, since I have Linux instance we execute the which provide various options like start, stop, reset.
I have used the command console to check the logs

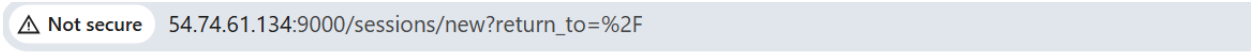
- ./sonar.sh console

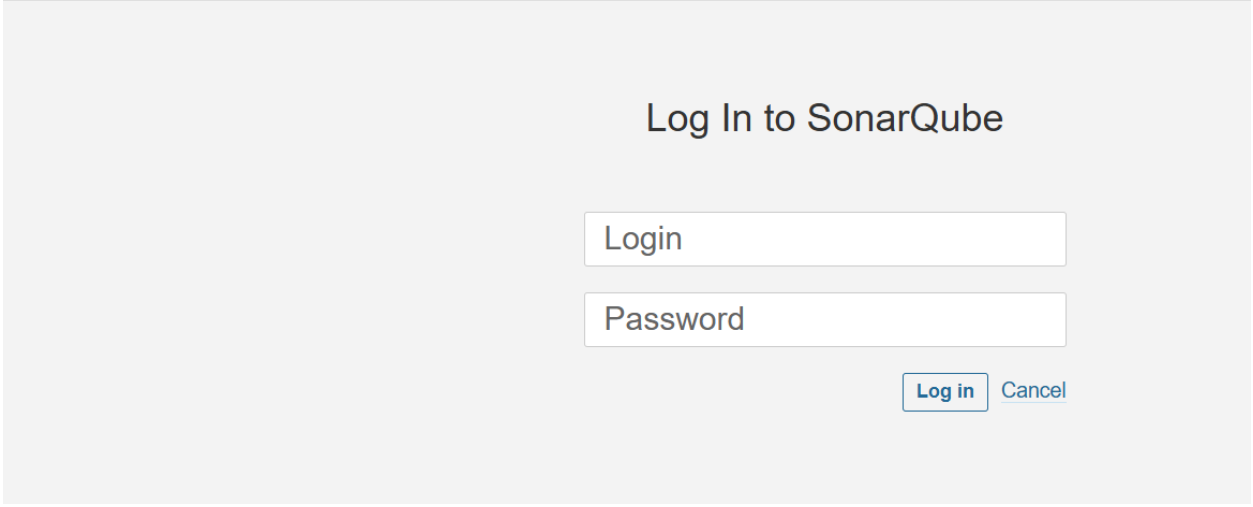
```
ubuntu@sonarqube:/opt/sonarqube/bin/linux-x86-64$ ./sonar.sh
/usr/bin/java
Usage: ./sonar.sh { console | start | stop | force-stop | restart | status | dump }
```

Edit the security groups by adding the port 9000 for the SonarQube with 0.0.0.0/0 which will accept all the IP addresses.

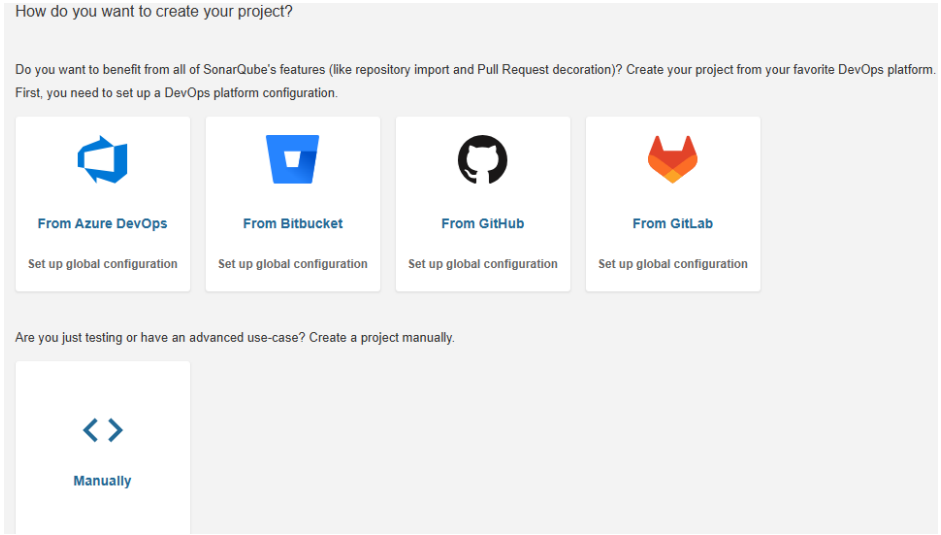


Try to access the SonarQube web page with our public IP address. Default login is admin/admin then set the new password.

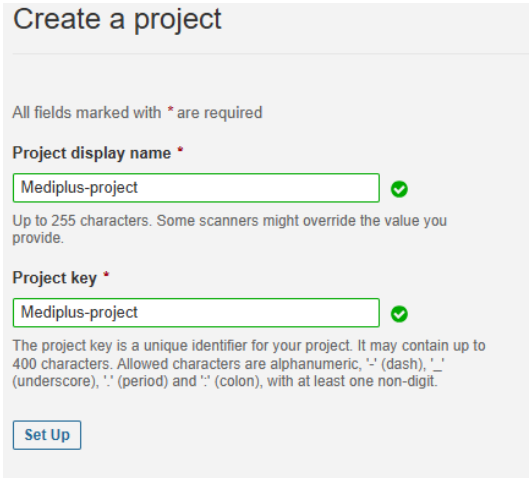




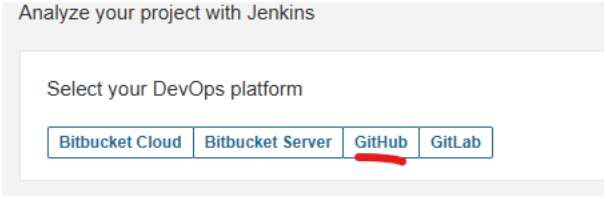
Logged into SonarQube page, selected the manual project.



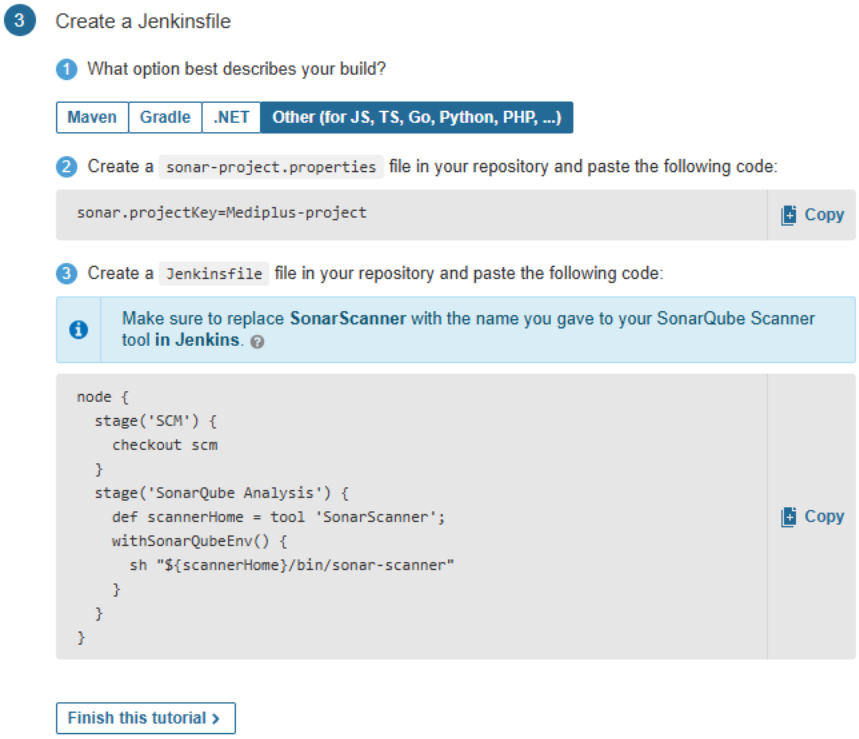
Given a name for the project, same as well as for the key.



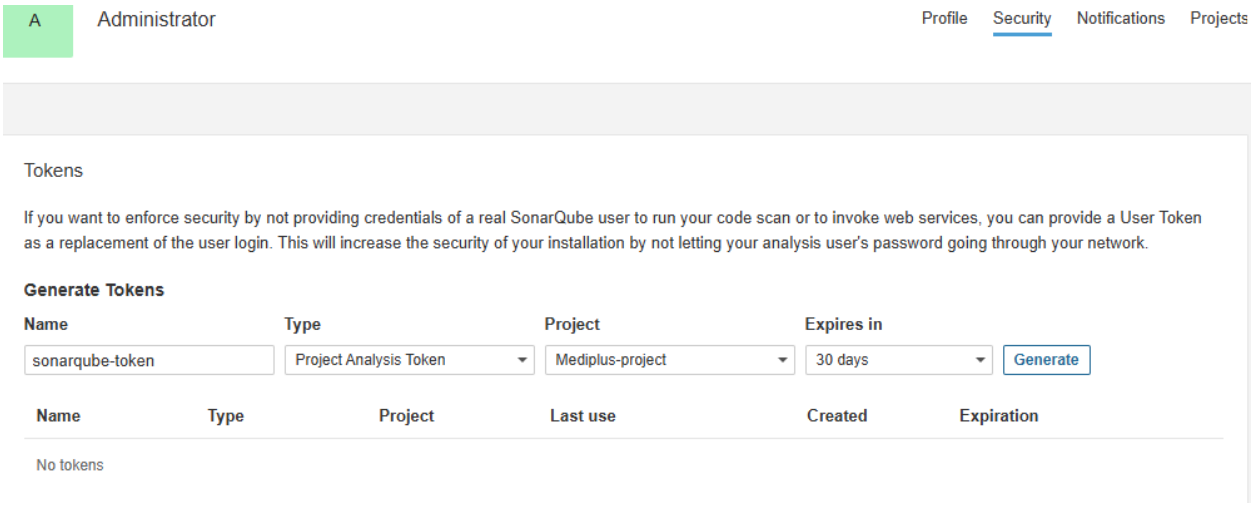
To analyze the repository, select the Jenkins. Select the GitHub option.



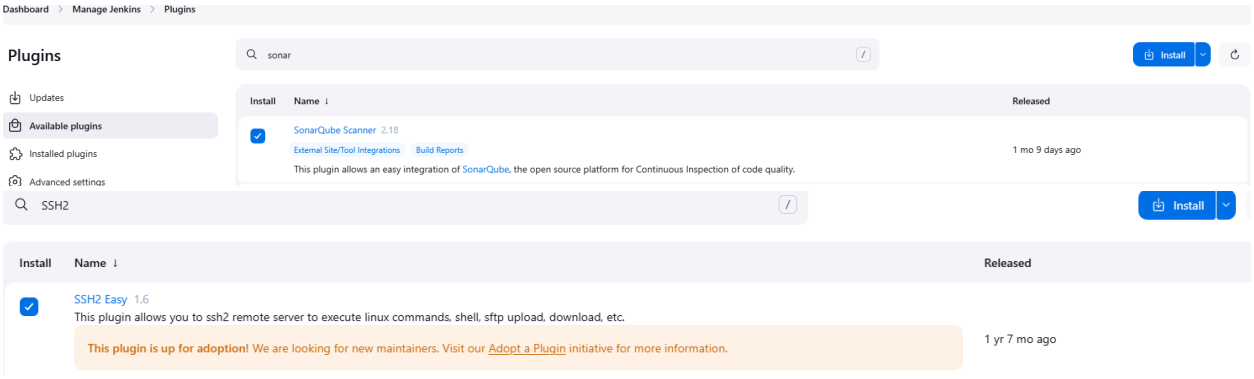
Configure the analysis, click on the continuation option for the pipeline. For creating the Jenkins file choose the other option and save the project key onto some text folder and at last finish the tutorial.



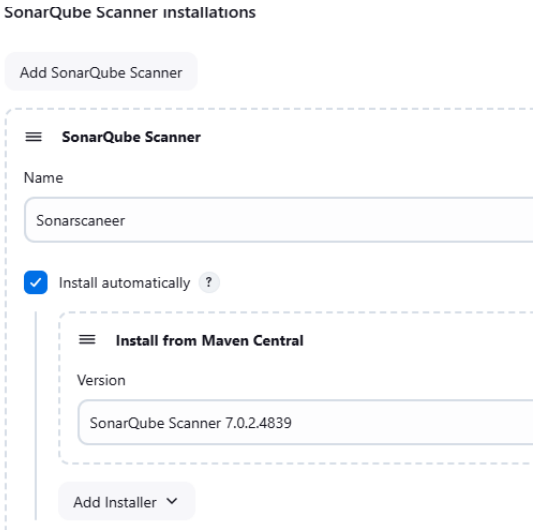
To create a token, go to Administration > My Account > Security, If you select the project analysis token it will be used for specific project, the token will be authenticated for a particular project. The Global Analysis token will be used for all the projects on your SonarQube. Generate the token and save it.



Go back to your Jenkins page, install two plugins in manage Jenkins section.



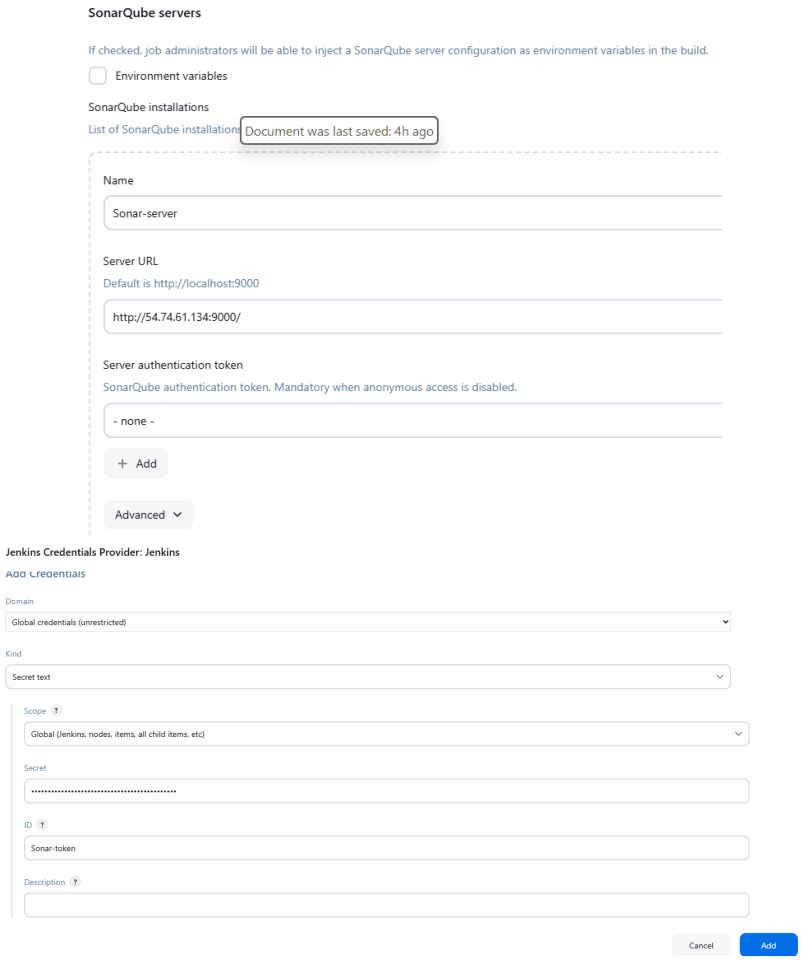
Next go to manage jenkins and tools configuration. There you will see the SonarQube scanner. Add it and save it.



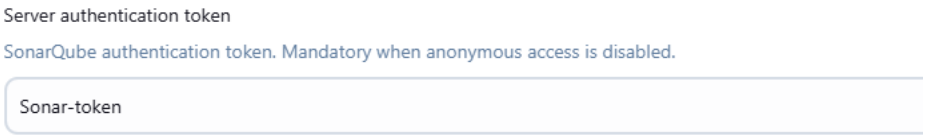
Go to the configure system, you will be able to see SonarQube servers. Add the server's name and URL of your SonarQube server.

In the pipeline select the configuration, add the build steps. Select the "**Execute SonarQube Scanner"** option and add the **project key.**

Then go to manage Jenkins to add the SonarQube token, click on add in SonarQube servers. Select the kind as Secret Text and add the token generated in SonarQube.



Select the token generated and save.



Go to the pipeline and build the pipeline, if it's working or not. The build is running where the SonarQube is scanning the code we have received the status as success. Let's refresh the page of SonarQube and go to the project section you will find that it has scanned our code, and the code passed with no vulnerabilities, some bugs. Once it is passed, we will deploy it on the docker server.

4.251.24.129:8080/job/Automated-Pieline/3/console

ne > #3 > Console Output

```
11:24:07.067 INFO  1/1 source file has been analyzed
11:24:07.071 INFO  No PHPUnit tests reports provided (see 'sonar.php.tests.reportPath' property)
11:24:07.072 INFO  No PHPUnit coverage reports provided (see 'sonar.php.coverage.reportPaths' property)
11:24:07.072 INFO  Sensor PHP sensor [php] (done) | time=199ms
11:24:07.072 INFO  Sensor Analyzer for "php.ini" files [php]
11:24:07.075 INFO  Sensor Analyzer for "php.ini" files [php] (done) | time=3ms
11:24:07.076 INFO  Sensor Text Sensor [text]
11:24:07.079 INFO  20 source files to be analyzed
11:24:07.160 INFO  20/20 source files have been analyzed
11:24:07.160 INFO  Sensor Text Sensor [text] (done) | time=84ms
11:24:07.160 INFO  Sensor VB.NET Project Type Information [vbnet]
11:24:07.161 INFO  Sensor VB.NET Project Type Information [vbnet] (done) | time=1ms
11:24:07.162 INFO  Sensor VB.NET Analysis Log [vbnet]
11:24:07.173 INFO  Sensor VB.NET Analysis Log [vbnet] (done) | time=11ms
11:24:07.173 INFO  Sensor VB.NET Properties [vbnet]
11:24:07.173 INFO  Sensor VB.NET Properties [vbnet] (done) | time=0ms
11:24:07.178 INFO  ------------ Run sensors on project
11:24:07.241 INFO  Sensor Analysis Warnings import [csharp]
11:24:07.242 INFO  Sensor Analysis Warnings import [csharp] (done) | time=1ms
11:24:07.242 INFO  Sensor Zero Coverage Sensor
11:24:07.252 INFO  Sensor Zero Coverage Sensor (done) | time=10ms
11:24:07.254 INFO  SCM Publisher SCM provider for this project is: git
11:24:07.256 INFO  SCM Publisher 20 source files to be analyzed
11:24:07.735 INFO  SCM Publisher 20/20 source files have been analyzed (done) | time=478ms
11:24:07.751 INFO  CPD Executor Calculating CPD for 6 files
11:24:07.815 INFO  CPD Executor CPD calculation finished (done) | time=63ms
11:24:07.893 INFO  Analysis report generated in 76ms, dir size=792.0 kB
11:24:07.945 INFO  Analysis report compressed in 52ms, zip size=198.6 kB
11:24:08.034 INFO  Analysis report uploaded in 88ms
11:24:08.036 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://54.74.61.134:9000/dashboard?id=Mediplus-project
11:24:08.036 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
11:24:08.037 INFO  More about the report processing at http://54.74.61.134:9000/api/ce/task?id=AZV6peenavNaJVyU_aA1
11:24:08.729 INFO  Analysis total time: 10.084 s
```

Dashboard > Automated-Pieline >

- Status
- Changes
- Workspace
- Build Now
- Configure
- Delete Project
- GitHub Hook Log
- Favorite
- SonarQube
- Open Blue Ocean
- Rename

Builds

Filter /

Today

#3 11:23 AM

Automated-Pieline

SonarQube

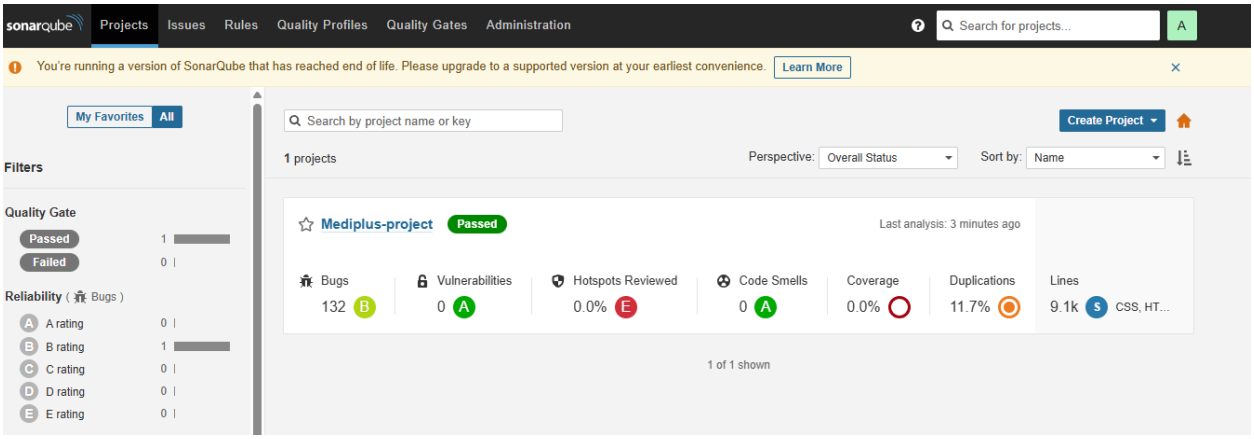
SonarQube Quality Gate

Mediplus-project Passed

server-side processing: Success

Permalinks

- Last build (#3), 4 min 12 sec ago
- Last stable build (#3), 4 min 12 sec ago
- Last successful build (#3), 4 min 12 sec ago
- Last completed build (#3), 4 min 12 sec ago



Connect to our three EC2 instances where we will be installing docker.

Install docker of the EC2 instance of ubuntu by following the official docker page.

Link: https://docs.docker.com/engine/install/ubuntu/

```
ubuntu@docker:~$ docker --version
Docker version 28.0.1, build 068a01e
```

Login to Jenkins server and execute the docker instance with Jenkins user and try to access the docker instance from Jenkins docker instance, where I faced permission denied error.

```
ubuntu@jenkins:~$ docker exec -it 0fe749c7e5f0 /bin/bash
jenkins@0fe749c7e5f0:/$ ssh ubuntu@34.240.100.90
ubuntu@34.240.100.90: Permission denied (publickey).
jenkins@0fe749c7e5f0:/$
```

To solve the problem, go to docker instance and edit the sshd_config file.

```
ubuntu@docker:~$ sudo su
root@docker:/home/ubuntu# vi /etc/ssh/sshd_config
```

Uncomment on the below line

```
#MaxSessions 10

PubkeyAuthentication yes
```

```
# To disable tunneled clear text passwords, ch
PasswordAuthentication yes
#PermitEmptyPasswords no
```

Restart the ssh services

```
root@docker:/home/ubuntu# systemctl restart ssh
Warning: The unit file, source configuration file or drop-in
root@docker:/home/ubuntu# systemctl daemon-reload
```

I am now asking for ubuntu user password.

```
jenkins@0fe749c7e5f0:/$ ssh ubuntu@34.240.100.90
(ubuntu@34.240.100.90) Password:
```

Changed the ubuntu password

```
root@docker:/home/ubuntu# passwd ubuntu
New password:
Retype new password:
passwd: password updated successfully
root@docker:/home/ubuntu#
```

Successfully logged into the docker machine from Jenkins machine.

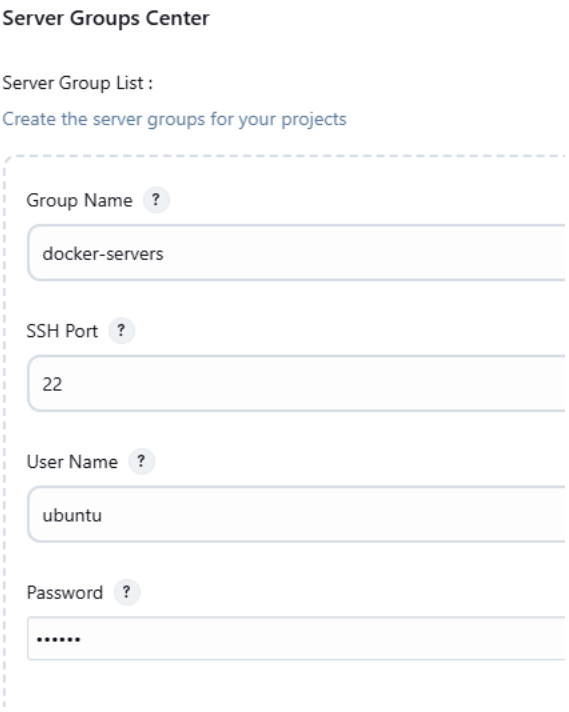
```
ubuntu@jenkins:~$ docker exec -it 0fe749c7e5f0 /bin/bash
jenkins@0fe749c7e5f0:/$ ssh ubuntu@34.240.100.90
(ubuntu@34.240.100.90) Password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1021-aws x86_64)

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

 System information as of Sun Mar  9 15:01:32 UTC 2025

  System load:  0.0               Processes:             135
  Usage of /:   39.9% of 6.71GB   Users logged in:       1
  Memory usage: 8%                IPv4 address for enX0: 172.31.46
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enabled.

137 updates can be applied immediately.
44 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Sun Mar  9 14:56:59 2025 from 157.45.74.194
ubuntu@docker:~$
```

Generate the ssh key in Jenkins instance and copy the key to the docker instance.

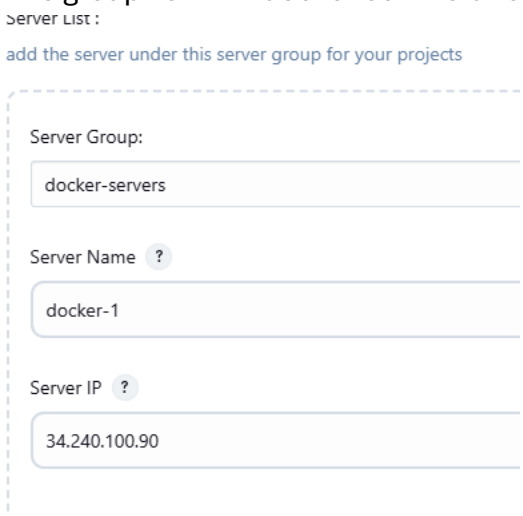
```
ubuntu@jenkins:~$ docker exec -it 0fe749c7e5f0 /bin/bash
jenkins@0fe749c7e5f0:/$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/jenkins_home/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/jenkins_home/.ssh/id_rsa
Your public key has been saved in /var/jenkins_home/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:47mSclGIyvaou5BBnbMwODea2+DYMSrvcxAPoRUEhls jenkins@0fe749c7e5f0
The key's randomart image is:
```

```
jenkins@0fe749c7e5f0:/$ ssh-copy-id ubuntu@34.240.100.90
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/var/jenkins_home/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
(ubuntu@34.240.100.90) Password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'ubuntu@34.240.100.90'"
and check to make sure that only the key(s) you wanted were added.

jenkins@0fe749c7e5f0:/$
```

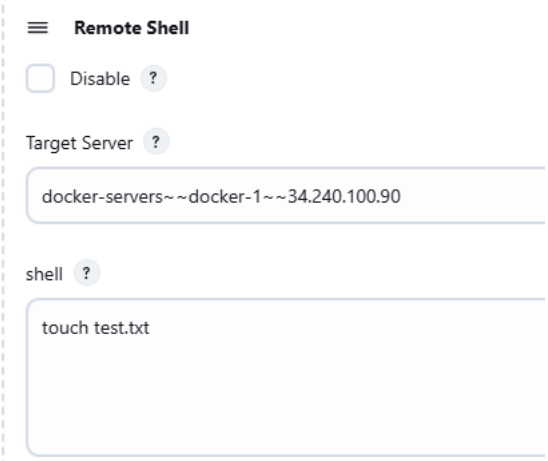
Next steps go to Jenkins web page and add the docker server in the manage Jenkins page.



Add the docker server to the group list with docker ec2 instance IP.

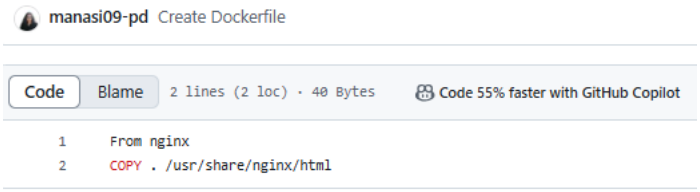


In the automation pipeline, go to configuration and at the end add the build step as Remote shell. To check if it's working added a line like touch command to create a file and triggered the build to check.

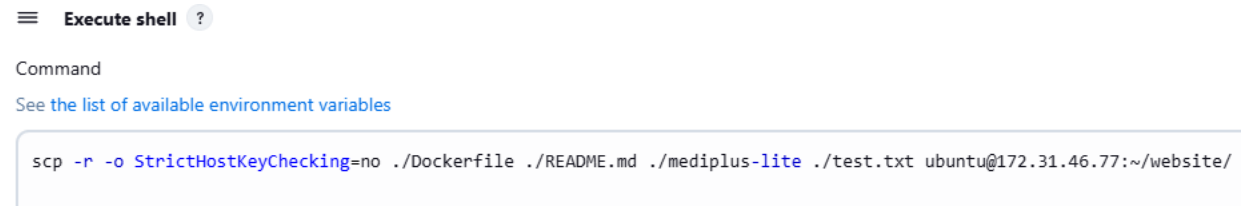


The build has run the remote command on the server.

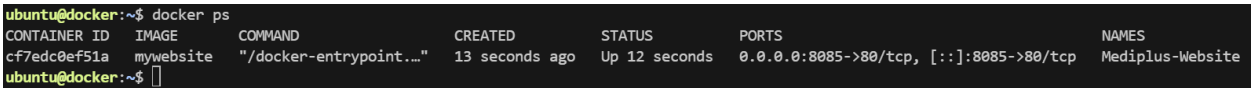
Go to the repository in GitHub and add a Docker file. After the commit the pipeline will be automatically triggered.



Go to the pipeline and configuration, remove the remote shell and add a build step, copy the contents of the current server to docker.



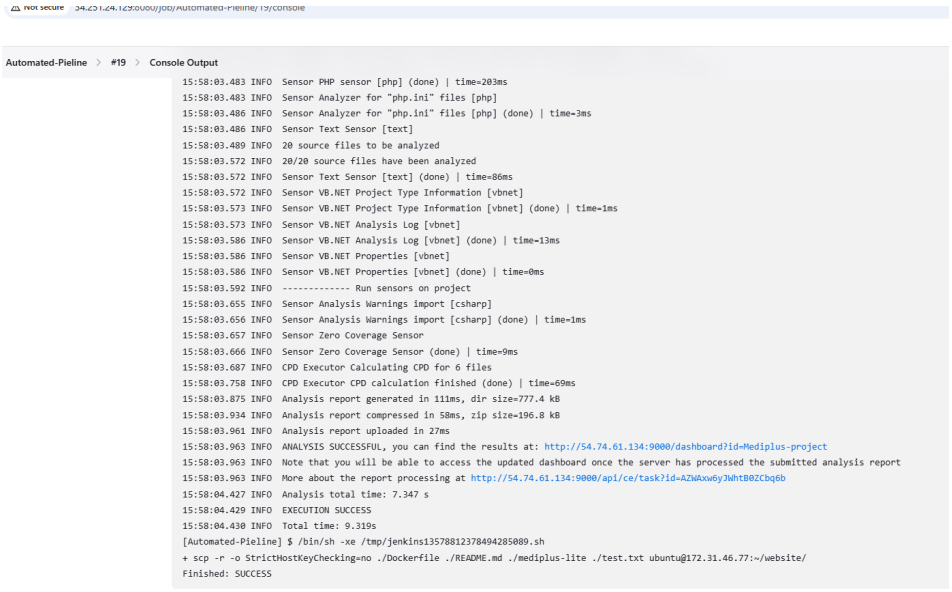
Build is successful, lets verify it.



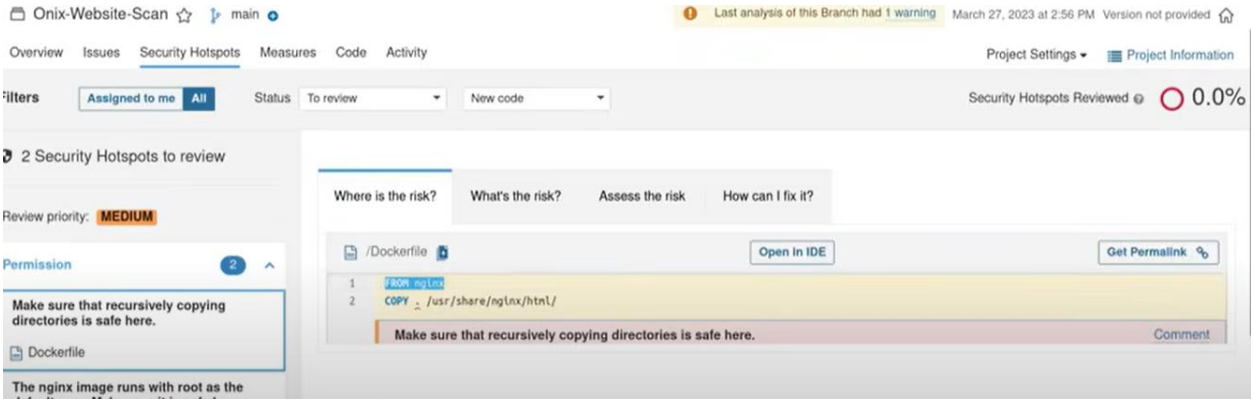
```
ubuntu@docker:~$ pwd
/home/ubuntu
ubuntu@docker:~$ mkdir website
ubuntu@docker:~$ ls
website
```

Create a website directory on docker server.

Build is successful.



Vulnerability scan is failing due to the addition of Docker file.



After the build is successful, check if the contents are copied to docker server.

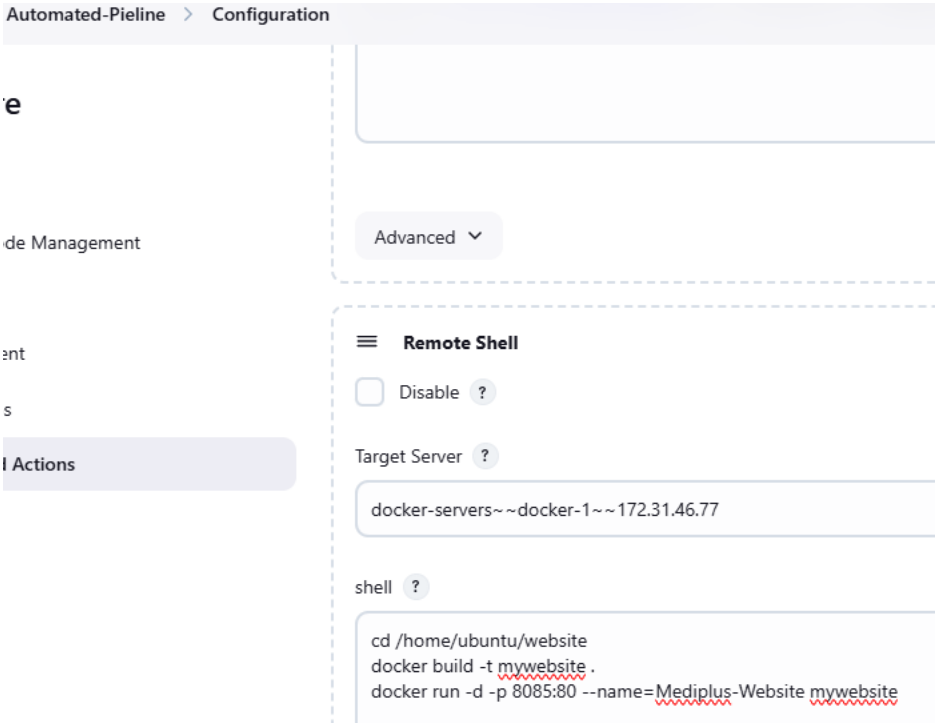
```
ubuntu@docker:~/website$ ls
Dockerfile  README.md  mediplus-lite  test.txt
ubuntu@docker:~/website$
```

Next go to the pipeline and select configure and add a build step as remote shell, go to the /home/ubuntu/website path and run the docker build.

First let's check if ubuntu user can perform the docker commands, otherwise add the ubuntu to the docker group with the below command.

- Sudo usermod -aG docker $user
- Newgrp docker

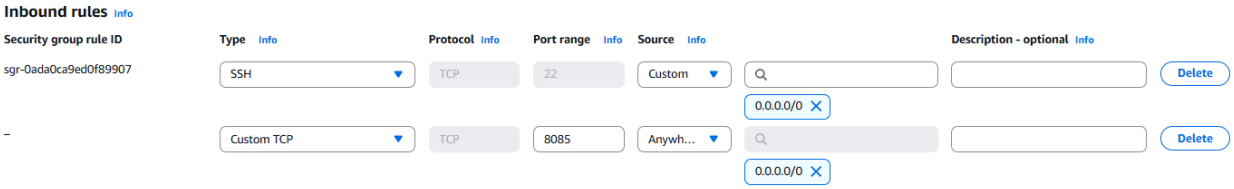
```
ubuntu@docker:~/website$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@docker:~/website$
```



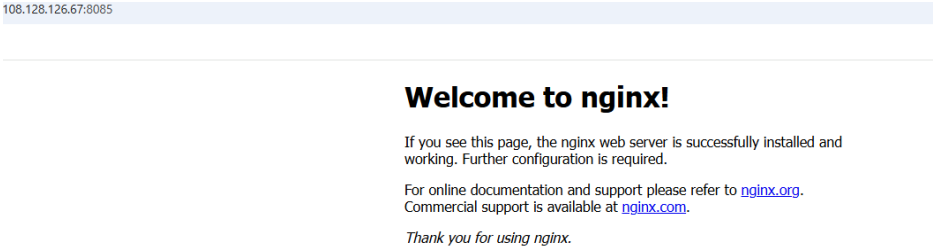
The remote shell will go to the path and perform docker build command to build my website and then run the container in detached mode, with port 80 that is container and 8085 port of the system.

```
ubuntu@docker:~$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                       NAMES
cf7edc0ef51a   mywebsite   "/docker-entrypoint.…"   13 seconds ago   Up 12 seconds   0.0.0.0:8085->80/tcp, [::]:8085->80/tcp   Mediplus-Website
ubuntu@docker:~$
```

Allow port 8085 in the inbound rules of the docker instance.



The website is working but the CSS is not working.



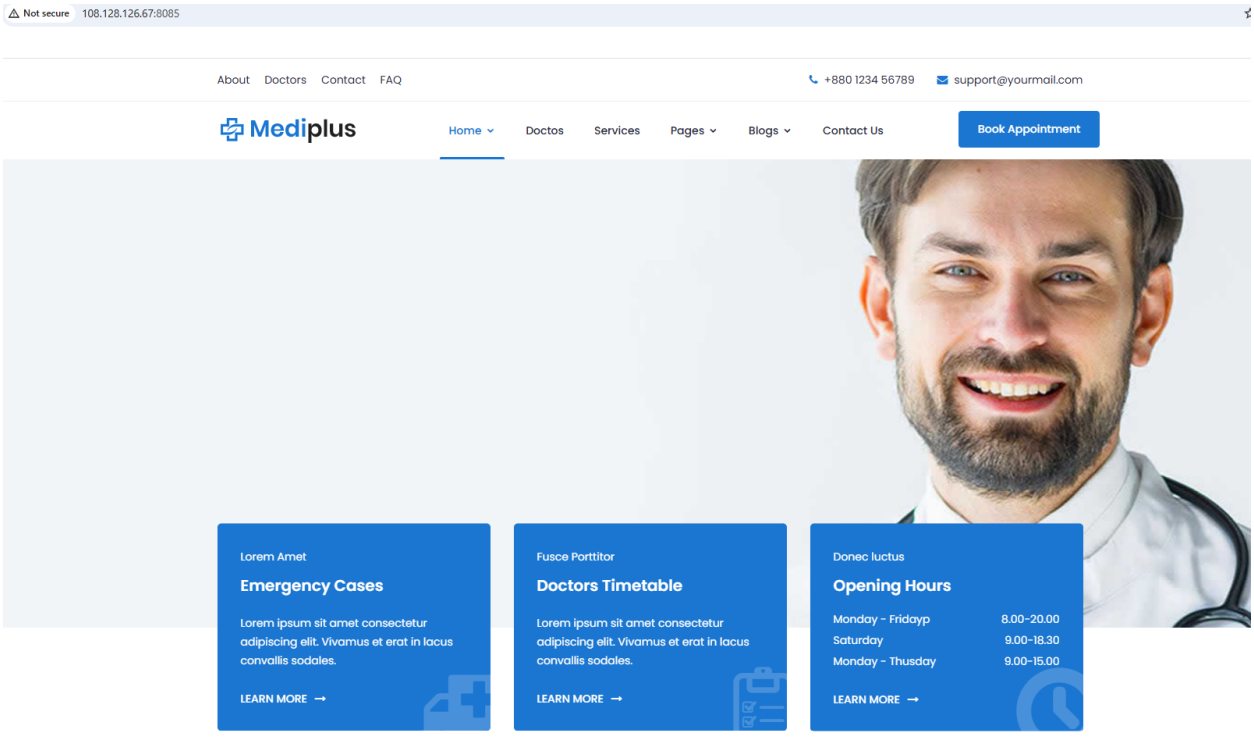
I have made some changes to the Docker file and pushed the code to GitHub and run the build same is copied on the docker instance.

```
# Use the offical Nginx image as a base
From nginx:latest

# Copy the website files into the appropriate Nginx directory
COPY ./mediplus-lite /usr/share/nginx/html

# Expose port 80 so that the website can be accessed
EXPOSE 80
~
~
```

Used the command in the Jenkins remote script.

- docker build -t mediplus-lite-web .
- docker run -d -p 8085:80 --name mediplus-container mediplus-lite-web

Now the website is hosted on the docker container.



Blog Single

Blog Categories

- Men's Apparel
- Women's Apparel
- Bags Collection
- Accessories
- Sun Glasses