International Institute of Information Technology
Bangalore

Software Production Engineering
CSE 816

Project Report

Report by:

Abhinav Kumar (IMT2022079)

October 2, 2025

SPE mini Project Report

Introduction

DevOps is a modern approach to software development and IT operations that emphasizes collaboration, automation, and continuous improvement across the entire application lifecycle. It bridges the gap between development (Dev) and operations (Ops) teams, ensuring faster delivery of reliable, scalable, and secure software. Organizations adopt DevOps to reduce time to market, improve product quality, enhance customer satisfaction, and increase agility in responding to business needs. By breaking down silos and fostering a culture of shared responsibility, DevOps enables teams to deliver innovation at speed while maintaining stability and efficiency.

Tools used

- github
- jenkins
- ansible
- docker
- ngrok
- maven

steps of workflow

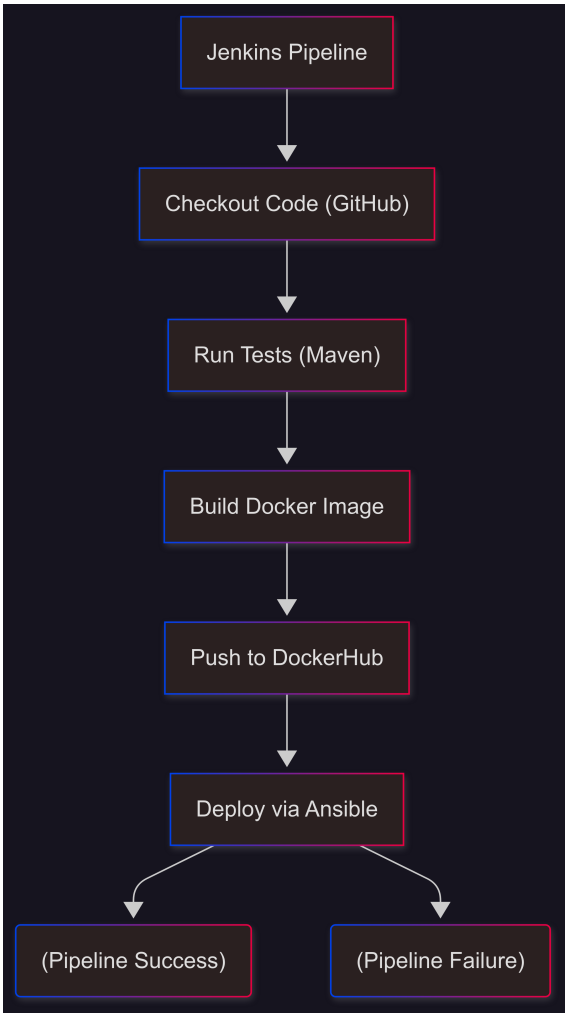


Figure 1: steps of pipeline

1. **Ngrok:** Expose Jenkins locally to GitHub by creating a public URL for webhook callbacks.
2. **Jenkins Pipeline Trigger:** GitHub webhook triggers the Jenkins pipeline.
3. **Checkout Stage:** Jenkins clones the repository from GitHub.
4. **Test Stage:** Run unit tests using Maven (‘./mvnw clean test‘).
5. **Build Docker Image:** Build a Docker image using the specified Dockerfile (‘Dockerfile.native‘).
6. **Push Docker Image:** Push the built Docker image to DockerHub.
7. **Deploy Stage:** Use Ansible playbooks to deploy the application on target servers.
8. **Post Actions:** Pipeline outputs success or failure messages based on the execution results.

Ngrok

Ngrok is used to expose local Jenkins server to the internet through a fixed public URL, allowing GitHub webhooks to reach Jenkins reliably. This enables the pipeline to be automatically triggered whenever changes are pushed to the repository or a pull request is created.

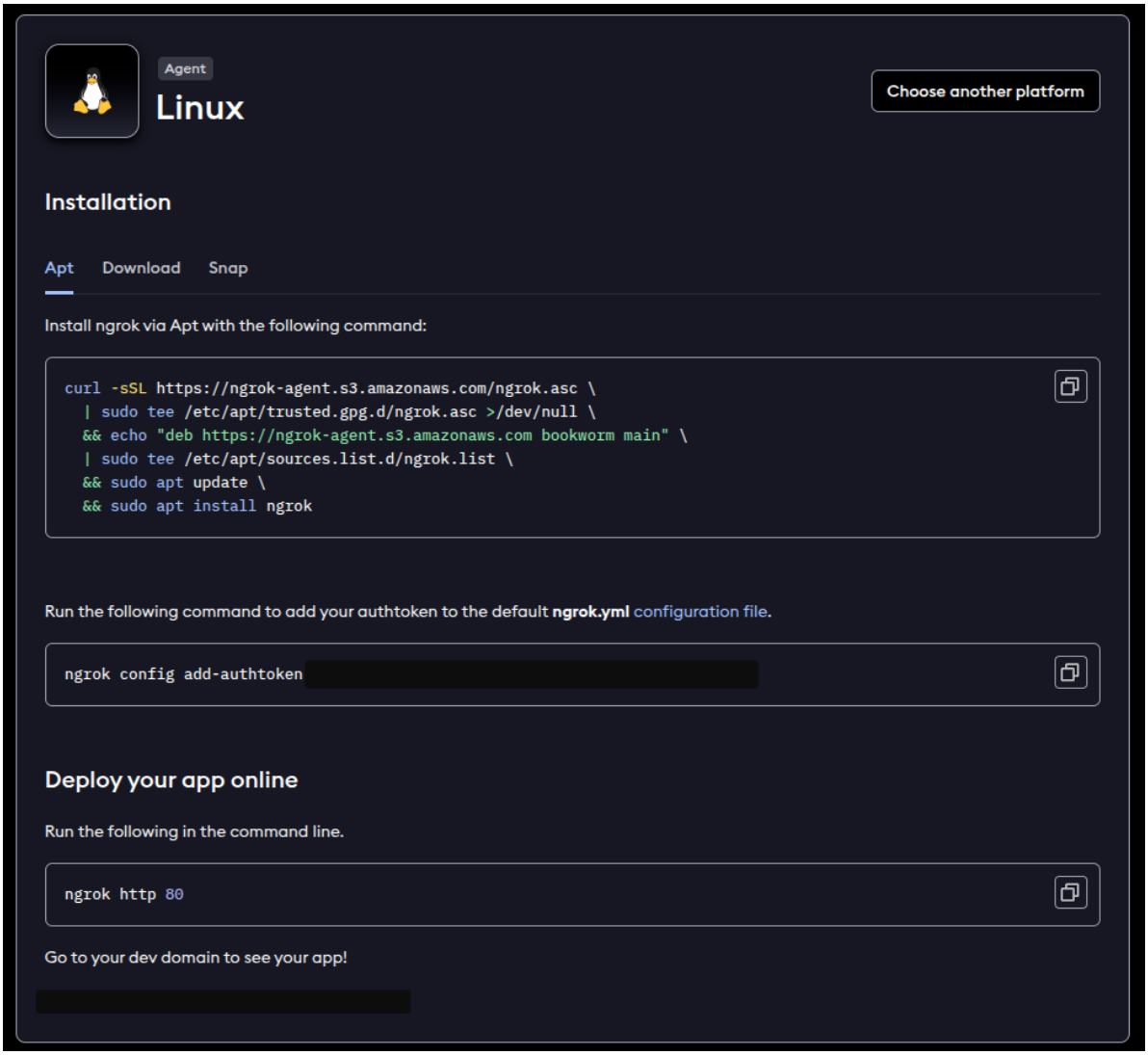


Figure 2: ngrok installation process

```
curl -sSL https://ngrok-agent.s3.amazonaws.com/ngrok.asc \
  | sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null \
  && echo "deb https://ngrok-agent.s3.amazonaws.com bookworm main" \
  | sudo tee /etc/apt/sources.list.d/ngrok.list \
  && sudo apt update \
  && sudo apt install ngrok
```

```
ngrok config add-authtoken <your-auth-token>
```

these above commands install and configure ngrok in an ubuntu/debian system
now we need to find our permanent url in ngrok dashboard

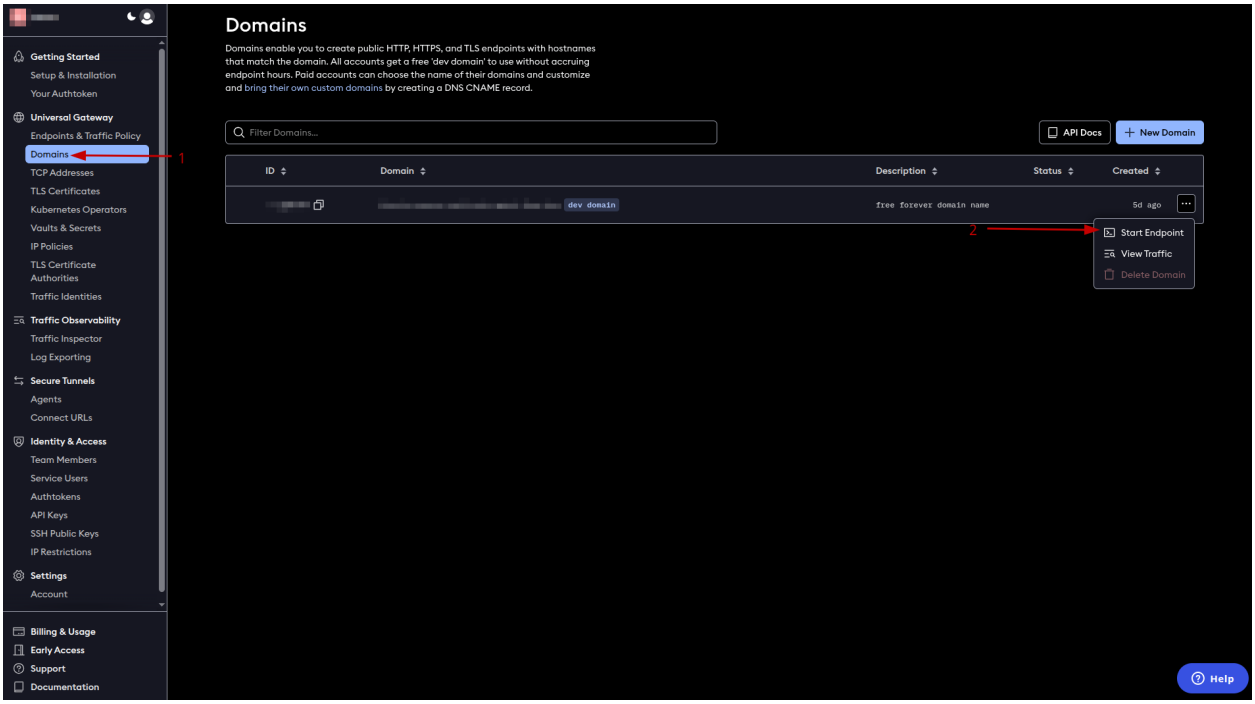


Figure 3: ngrok dashboard

```
ngrok http --url=<your-permanent-ngrok-url> 8080
```

this above command starts the ngrok tunnel on port 8080 which is the port jenkis use

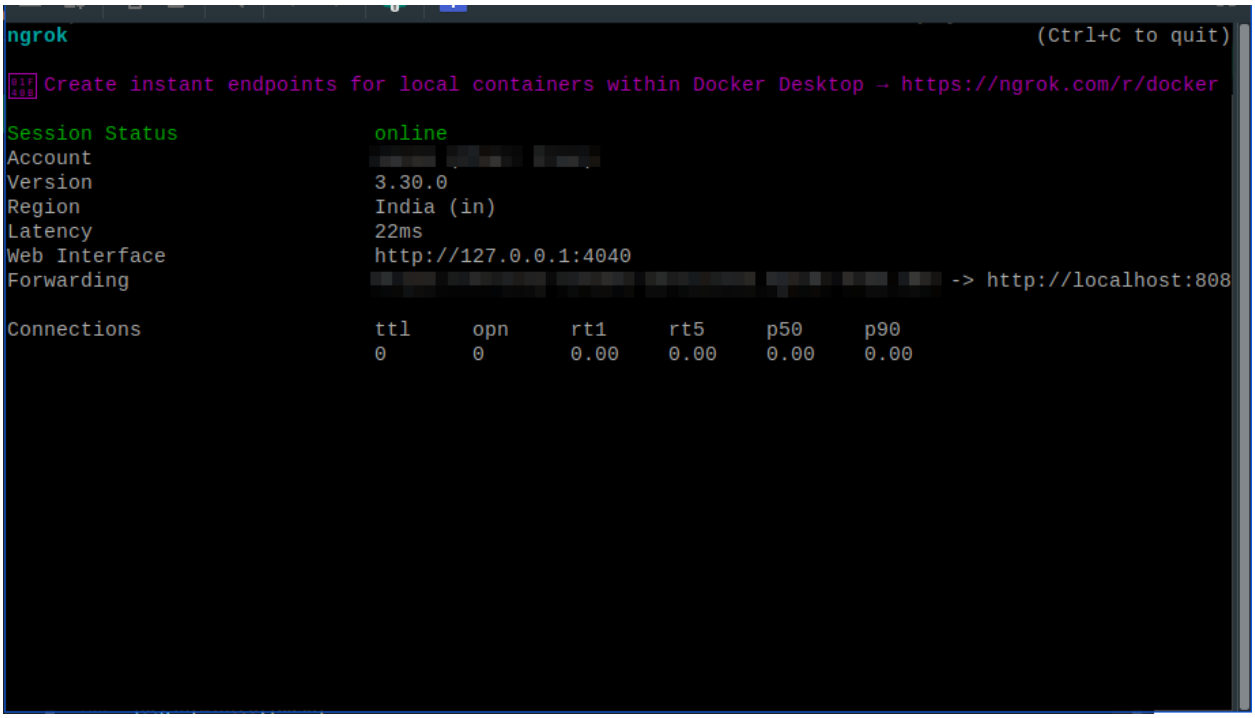


Figure 4: ngrok after running command

Jenkins

Jenkins automates the workflow by pulling code from Git and running tests, building and pushing Docker images, and deploying the application using Ansible. This reduces manual effort and speeds up software delivery.
for installing jenkins we need to have java 17 or later which can be installed using following commands:

```
sudo apt update
sudo apt install fontconfig openjdk-21-jdk
```

once we have install java we can install jenkins using following commands:

```
sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update
sudo apt install jenkins
```

once we have installed jenkins we can go to localhost:8080 in browser and set up jenkins by following on screen instructions and installing suggested plugins
Now we need to install docker pipeline plugin by following steps in below screenshots

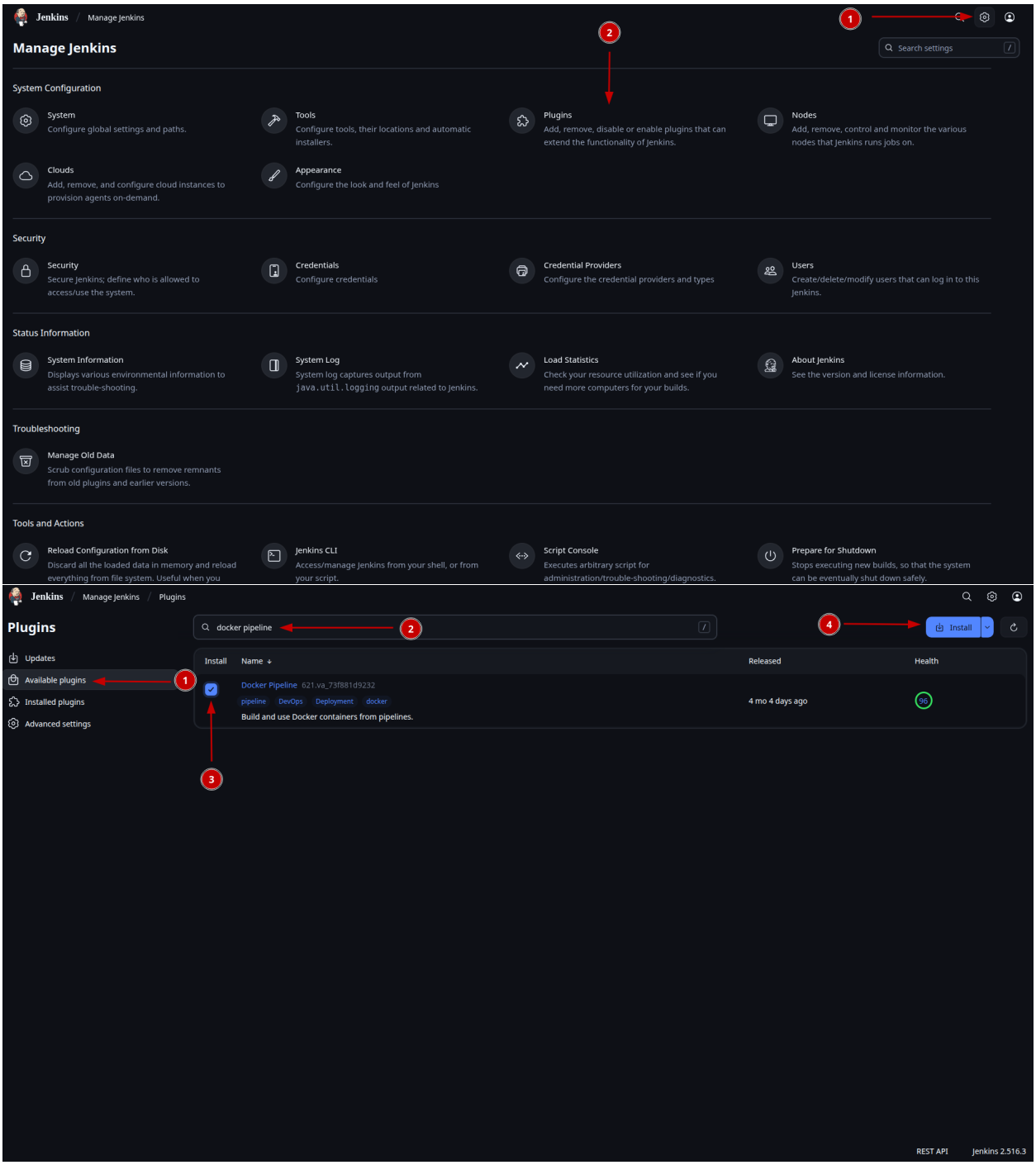
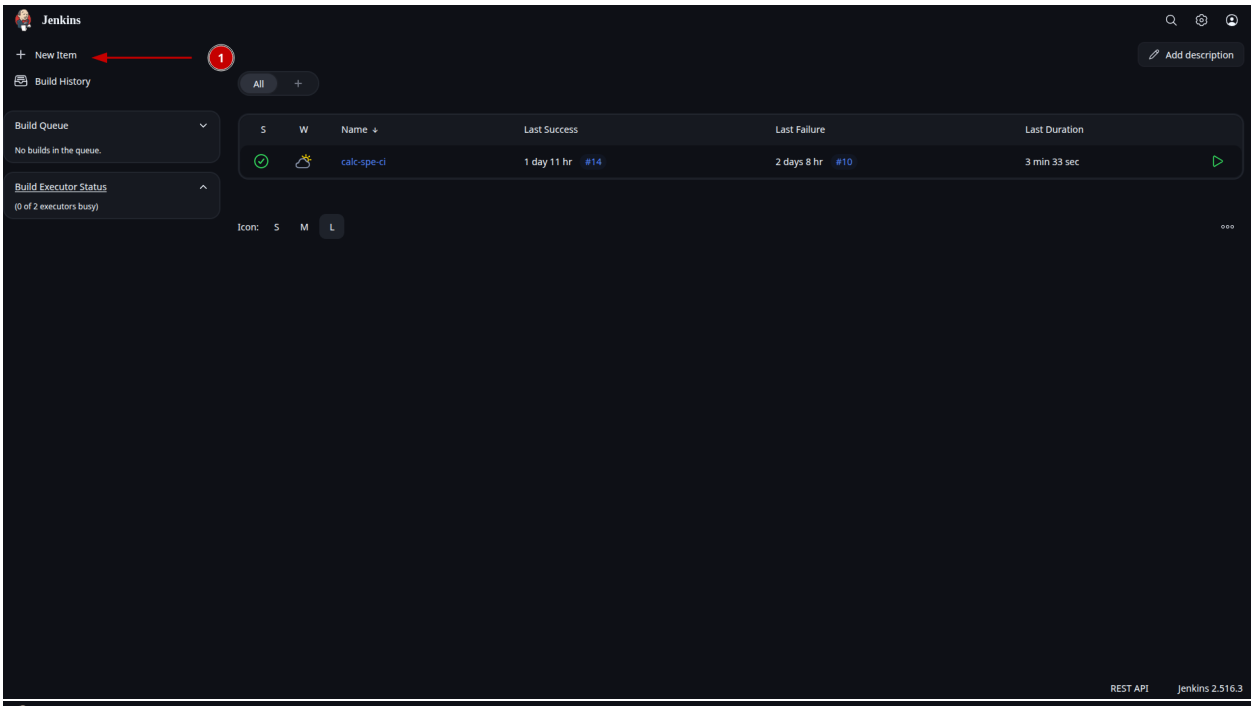
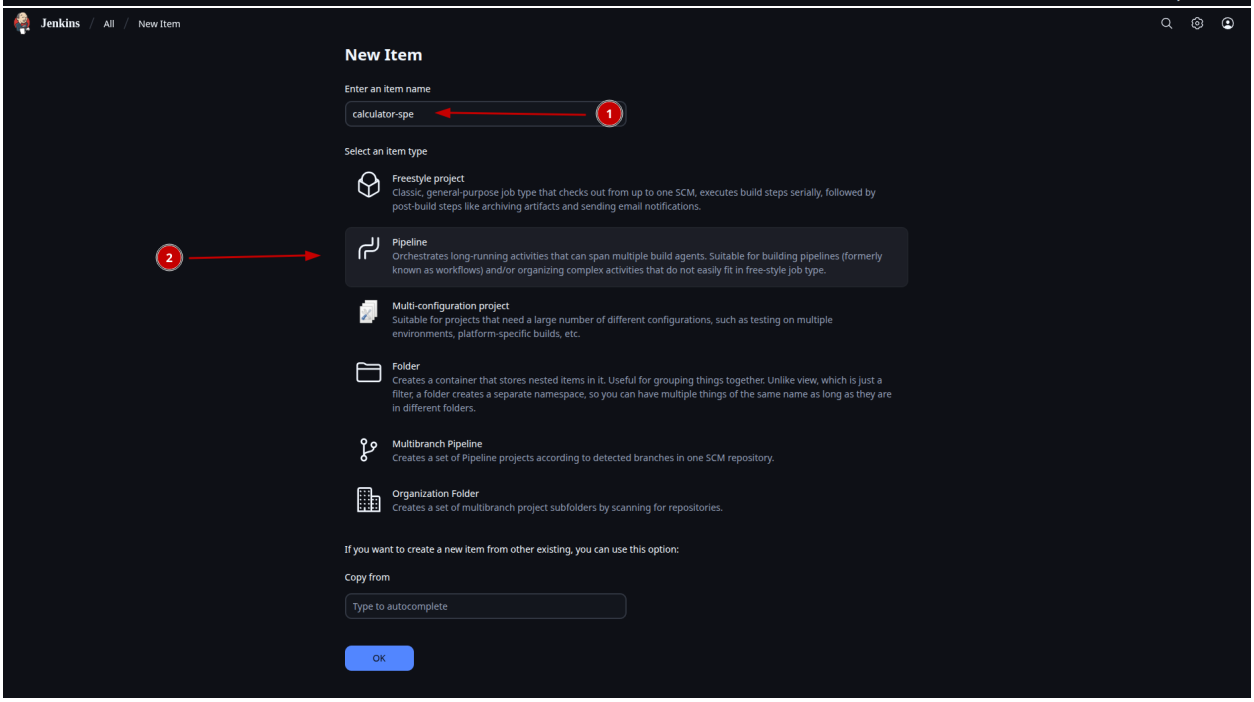


Figure 5: installing docker pipeline plugin

The steps to setup project in jenkins can be done by following steps


Jenkins
New Item
1
Build History
Add description
All
Build Queue
No builds in the queue.
Build Executor Status
(0 of 2 executors busy)
S
W
Name
Last Success
Last Failure
Last Duration
calc-spe-ci
1 day 11 hr #14
2 days 8 hr #10
3 min 33 sec
Icon: S M L
REST API
Jenkins 2.516.3


Jenkins / All / New Item
New Item
Enter an item name
calculator-spe
1
Select an item type
Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
2
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.
Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.
If you want to create a new item from other existing, you can use this option:
Copy from
Type to autocomplete
OK

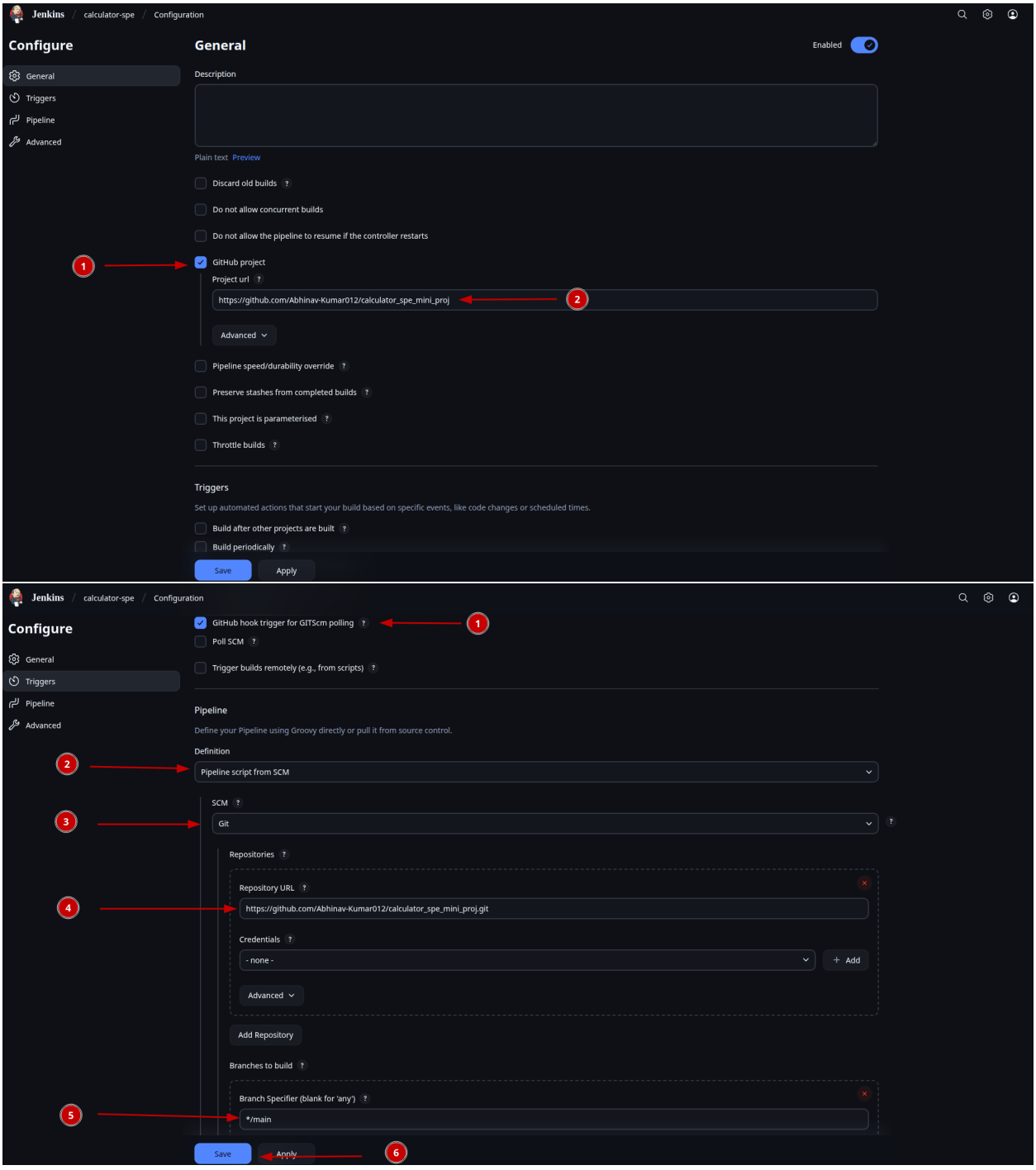


Figure 6: steps to configure jenkins

by following and above steps we can set up our pipeline for the project the jenkinsfile is as follows:

```
pipeline{
    agent any
    environment {
```

```
    DOCKERHUB_USER = 'vanos007'
    IMAGE_NAME = "calc-spe"
    IMAGE_TAG = "latest"
    DOCKER_IMAGE = "${DOCKERHUB_USER}/${IMAGE_NAME}:${IMAGE_TAG}"
    ANSIBLE_HOME = "/var/lib/jenkins/.local/bin"
    DOCKERFILE = "Dockerfile.native"
}
stages{
    stage('checkout'){
        steps{
            git branch: 'main', url:
            ↪  'https://github.com/Abhinav-Kumar012/calculator_spe_mini_proj.git'
        }
    }
    stage('test'){
        steps{
            dir('calc'){
                sh './mvnw clean test'
            }
        }
    }
    stage('Build Docker Image'){
        steps {
            script {
                docker.build("${DOCKER_IMAGE}", "-f ${DOCKERFILE} .")
            }
        }
    }
    stage('Push to DockerHub'){
        steps {
            script {
                docker.withRegistry('https://index.docker.io/v1/',
                ↪  'dockerhub-creds') {
                    docker.image("${DOCKER_IMAGE}").push()
                }
            }
        }
    }
    stage('Use ansible to deploy'){
        steps{
            dir('ansible'){
                withEnv(["PATH=${ANSIBLE_HOME}:${env.PATH}"]) {
                    sh 'ansible-playbook -i inventory.ini deploy.yml'
                }
            }
        }
    }
}
```

```
    post{
        success{
            echo "successfully executed the pipeline"
        }
        failure{
            echo "failed to execute the pipeline"
        }
    }
}
```

This Jenkins pipeline configuration automates the complete CI/CD process for the calculator project. It begins with the checkout stage, where Jenkins pulls the latest code from the GitHub repository's main branch, ensuring that the pipeline always works with updated code. Next, in the test stage, Maven is used to clean and run the unit tests inside the calc directory, verifying the correctness of the code before proceeding further. Once the tests pass, Jenkins moves to the Build Docker Image stage, where a Docker image is built using the specified Dockerfile.native. The image is tagged with the defined repository name and version (latest) and stored locally. After building, the Push to DockerHub stage uploads the Docker image to DockerHub using stored credentials, making it available for deployment across environments. Finally, in the Use Ansible to deploy stage, Jenkins executes an Ansible playbook (deploy.yml) with the given inventory to automate deployment on the target servers. The configuration also defines environment variables for Docker and Ansible paths to ensure smooth execution. At the end, the post section handles pipeline outcomes: displaying a success message if all stages run correctly, or a failure message if something goes wrong. Overall, this pipeline ensures a fully automated workflow from fetching code, testing it, packaging into containers, pushing to DockerHub, and finally deploying via Ansible streamlining continuous integration and delivery.

Github

GitHub is a cloud-based platform for hosting and managing Git repositories. In this process, it stores the project code, tracks changes through version control, and integrates with Jenkins to trigger automated builds and deployments whenever code is updated.
below are following steps to create a repo

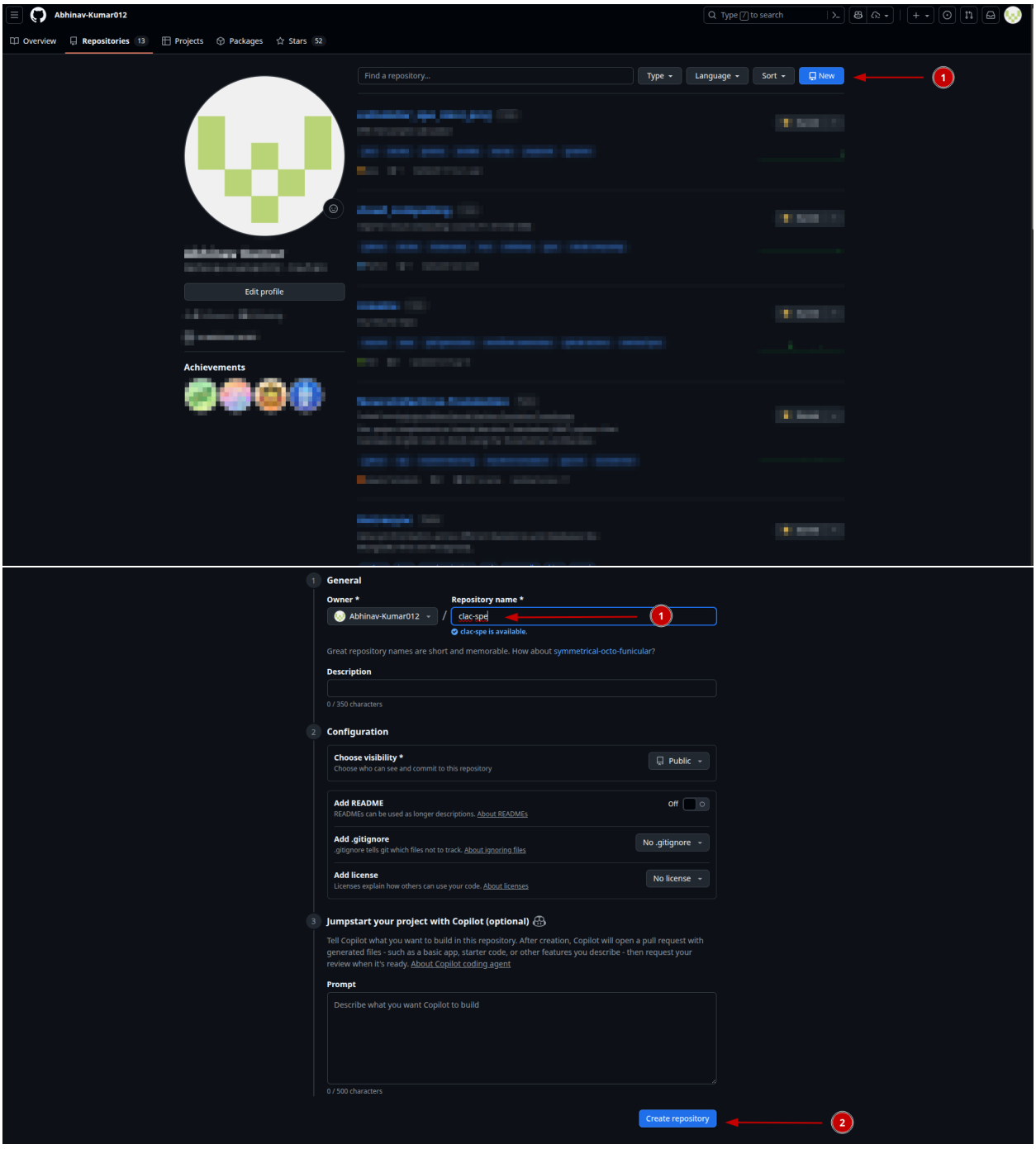
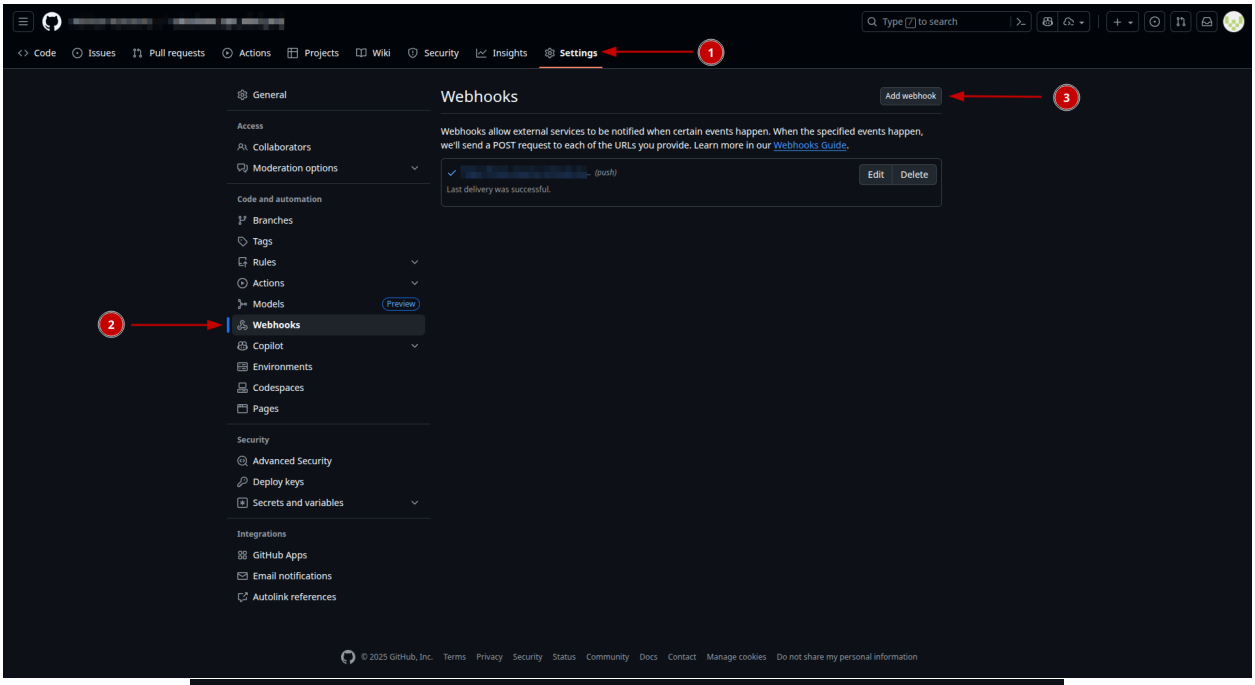


Figure 7: steps to create github repo

Now we need to configure github webook to trigger jenins on every push as follows:



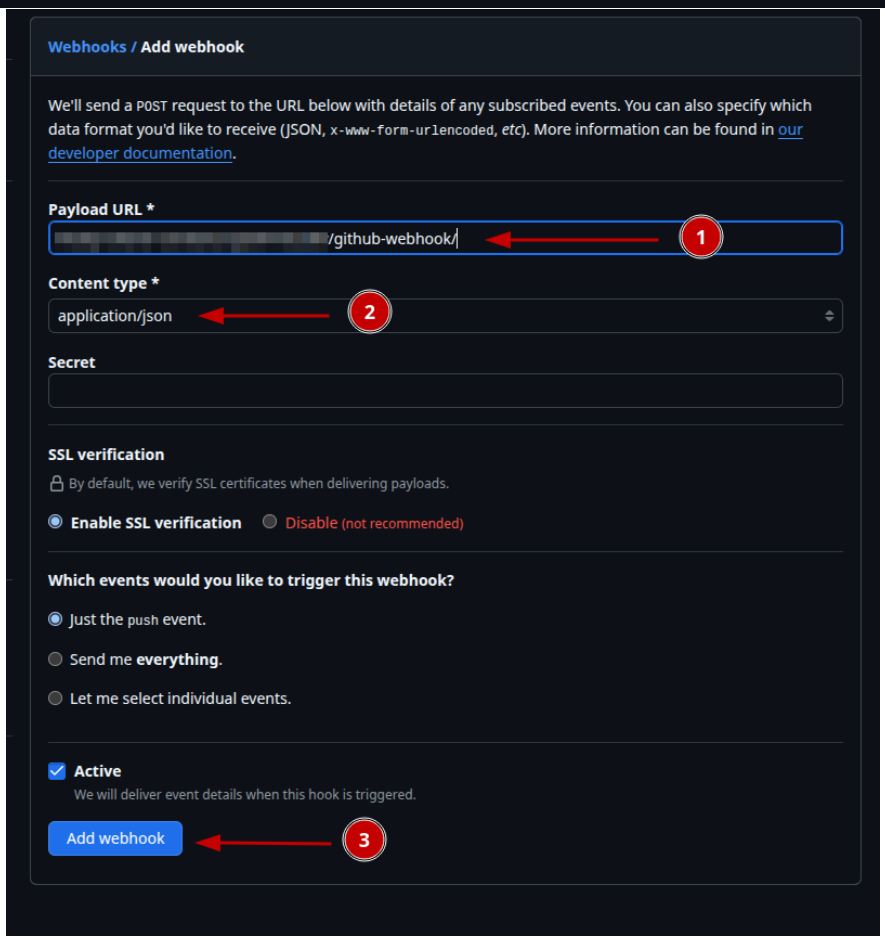


Figure 8: steps to add webhook to repo

here we fill the payload url[8] with the url obtained from ngrok website[3] in following format : `https://<your-ngrok-url>/github-webhook/`

```
#common git command
git init
git add .
git commit -m "<your-commit-message>"
git add remote <remote-name> <remote-url>
git push -u <remote-name> <branch-name>
git pull <remote-name> <branch-name>
git clone <repo-url>
```

these commands are used to set up git repo and remote server and push or pull the code from remote server

Docker

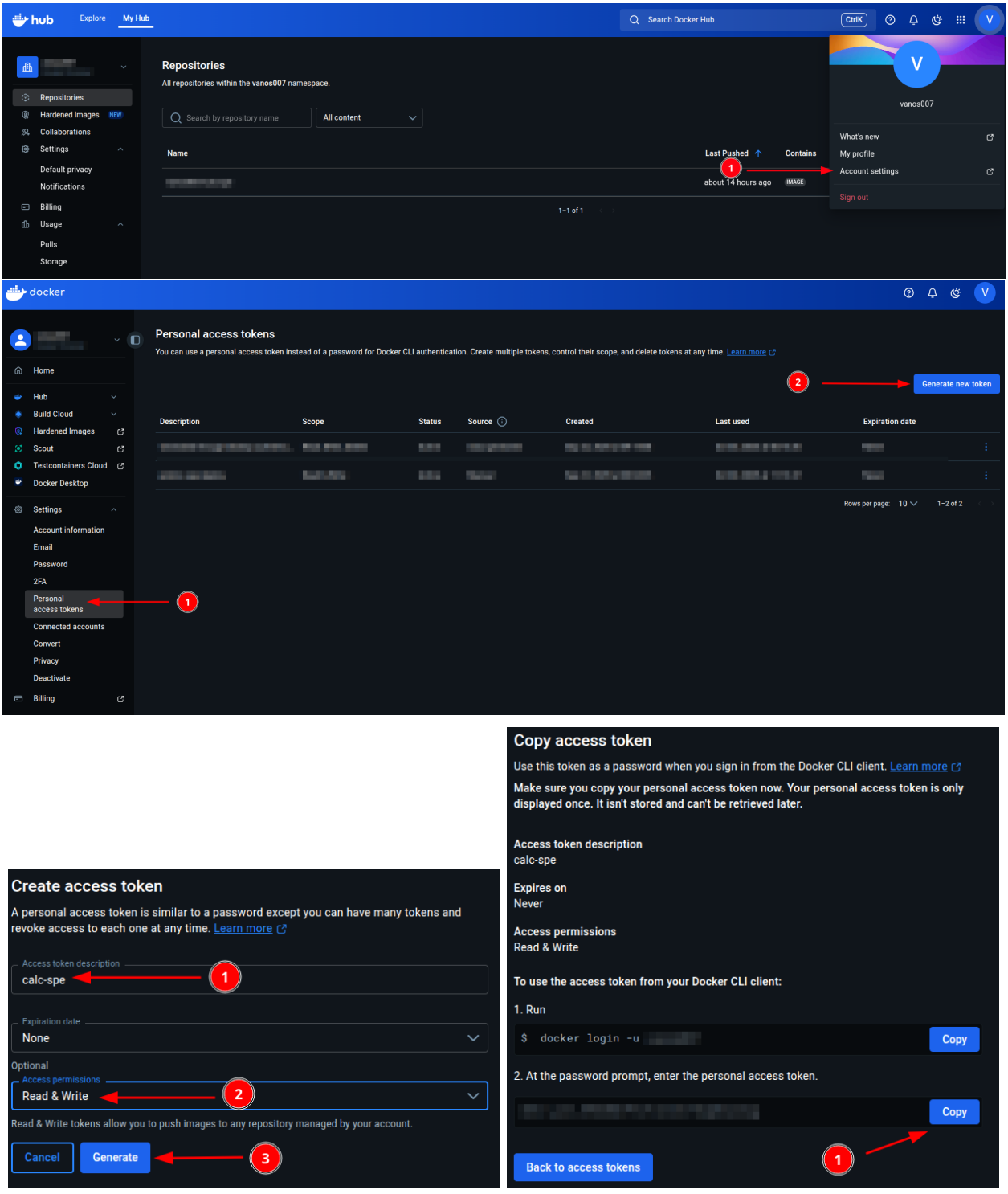
Docker is a platform that allows applications to run inside lightweight, portable containers. It helps ensure consistency across different environments and simplifies building, shipping, and deploying applications.
To install docker in a debian/ubuntu system we use the following commands:

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o
↪  /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
  ↪  https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
↪  docker-compose-plugin

# add jenkins to docker group
sudo usermod -aG docker jenkins
# build a docker image
docker build -t <image-name> -f <dockerfile-name> .
# docker create a continer
docker create -ti --name <container-name> <image-name>
# docker start and attach container
docker start -ai <container-name>
```



(a) (b)

Figure 9: obtain personal token from dockerhub

the above steps are used to obtain personal access token from dockerhub by navigating to https://hub.docker.com/ and logging in your account

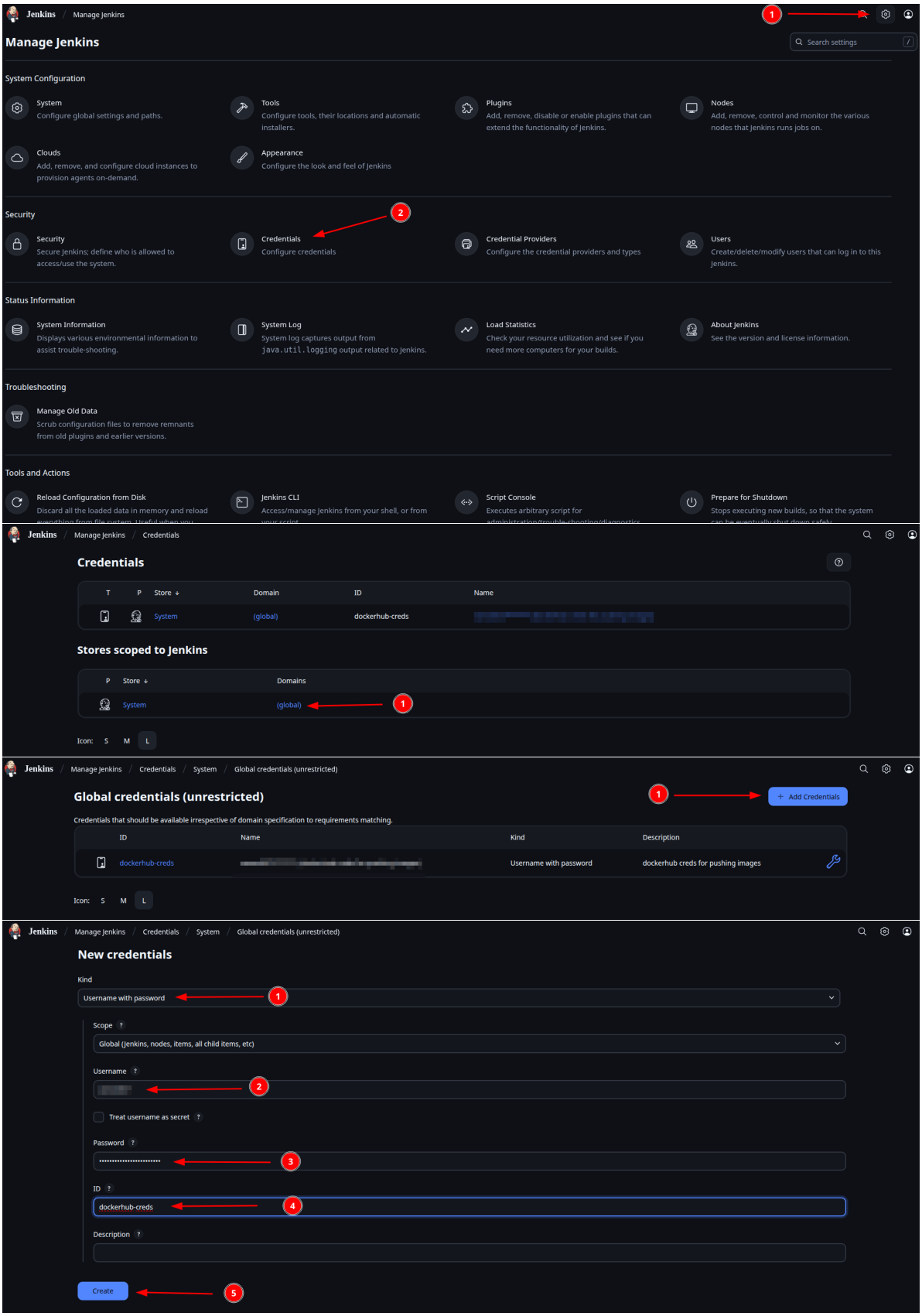


Figure 10: jenkins credential setup

following the above steps we can set up aur docker hub credentials in jenkins. We have to fill our docker hub username in username section and access token[9] in password section[10]
here is the dockerfile used to make the image

```
FROM ghcr.io/graalvm/native-image-community:21-muslib AS build

WORKDIR /app

COPY calc/ .

RUN ./mvnw -B clean package

RUN native-image --static --libc=musl -cp target/calc-1.0-SNAPSHOT.jar
↪  com.spe.calc.Main calc

FROM scratch

COPY --from=build /app/calc /calc

ENTRYPOINT ["/calc"]
```

This Dockerfile creates a lightweight container image for a Java application compiled into a native executable using GraalVM Native Image. It first uses the `ghcr.io/graalvm/native-image-community:21-muslib` base image to build the project, sets the working directory to '/app', copies the application source ('calc/'), and builds it with Maven ('./mvnw clean package'). Then, it generates a statically linked native binary ('calc') from the JAR using GraalVM's 'native-image' tool with the musl C library, ensuring portability and minimal runtime dependencies. Finally, a 'scratch' base image (an empty image) is used to copy only the compiled binary, making the final image extremely small and efficient. The container runs the native executable directly via the 'ENTRYPOINT'.

Maven

Maven is a build automation and project management tool for Java applications. It handles dependencies, compiles code, runs tests, and packages applications efficiently.

```
# make the jar file for execution
mvn clean install
# test the code
mvn clean test
```

Ansible

Ansible is an automation tool used for configuration management, application deployment, and task orchestration. It uses simple, human-readable YAML playbooks to automate repetitive IT tasks across multiple servers.
To install jenkins in ubuntu/debian sytems we must first install python and pipx as follows:

```
sudo apt install python3-full python3-pip pipx -y
sudo -u jenkins pipx install --include-deps ansible
```

To run a ansible playbook use the following command

```
ansible-playbook -i <machines>.ini <playbook>.yml
```

configration of ansible :
inventory.ini :

```
[myhosts]
localhost ansible_connection=local
```

deploy.yml:

```
- name: deploy calculator app
  hosts: myhosts
  vars:
    calc_image_name: "vanos007/calc-spe"
    calc_image_tag: "latest"
    calc_container: "calc_spe_container"

  tasks:
    - name: remove a container if necessary
      community.docker.docker_container:
        name: "{{ calc_container }}"
        state: absent

    - name: Pull latest Docker image from dockerhub
      community.docker.docker_image:
        name: "{{ calc_image_name }}"
        tag: "{{ calc_image_tag }}"
        source: pull

    - name: create container from image pulled
      community.docker.docker_container:
        name: "{{ calc_container }}"
        image: "{{ calc_image_name }}:{{ calc_image_tag }}"
        state: present
        tty: true
        interactive: true
        detach: true
        restart_policy: "no"
```

This Ansible configuration automates the deployment of the calculator application using Docker. It targets the ‘myhosts‘ group, which in this case points to the local machine. The playbook first ensures that any existing container named ‘calc_spe_container‘ is removed to avoid conflicts. It then pulls the latest Docker image (‘vanos007/calc-spe:latest‘) from DockerHub and creates a new container from that image with interactive and detached settings. The configuration ensures that the application runs consistently and can be redeployed easily, providing a simple, repeatable, and automated deployment process.

calcuator screenshots

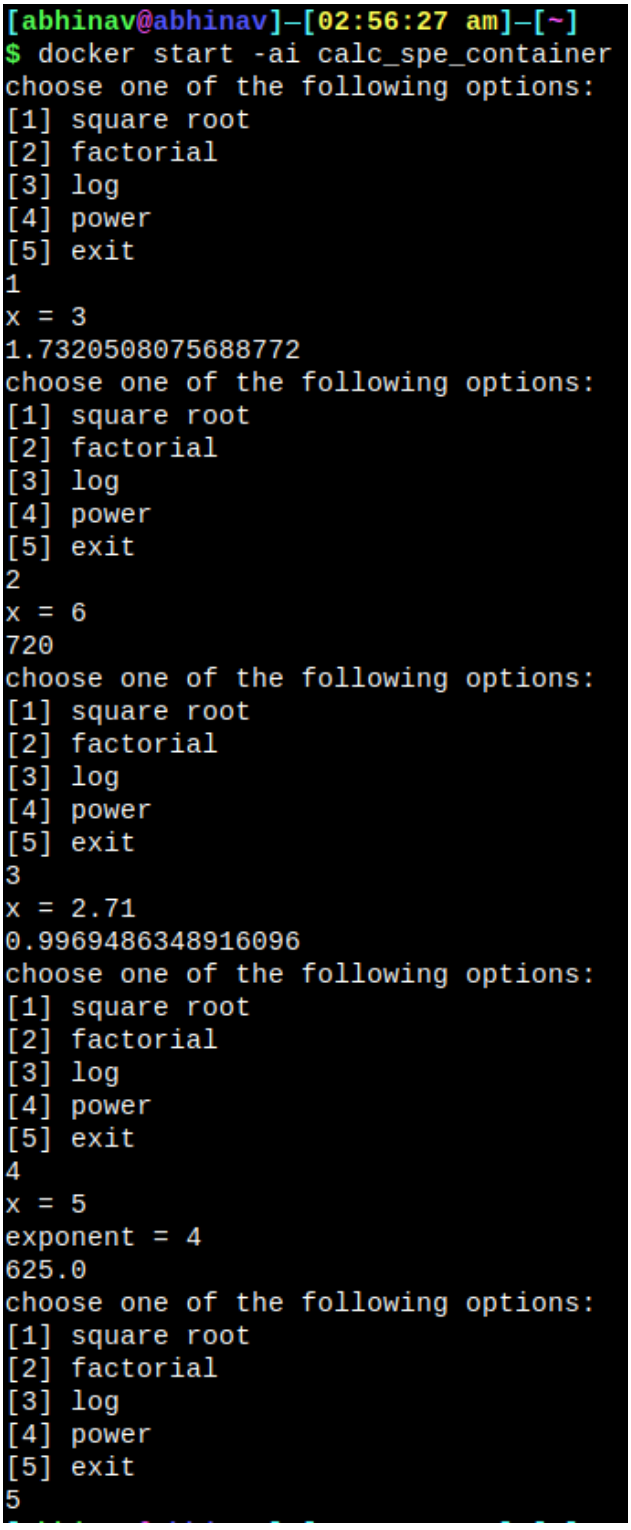


Figure 11: calcuator in action

Links to github repo and dockerhub

- github - https://github.com/Abhinav-Kumar012/calculator_spe_mini_proj
- docker hub - https://hub.docker.com/r/vanos007/calc-spe