Deploying a Cloud-Native Monitoring Solution on Observability Platform

Kartik Mahajan

TABLE OF CONTENTS

List of Figures

ABSTRACT

This project aims to automate the deployment process for cloud-native monitoring on an observability platform. The manual deployment process is time-consuming and inefficient, and this project will use DevOps principles and tools to improve efficiency, reliability, and eliminate manual intervention.

The project will use Jenkins for automation, GitHub for code storage, Docker for code packaging, Docker Hub for container management, and AWS (EC2) for project hosting. The automation implementation will involve developing automation scripts and configurations for deployment, integrating Jenkins with GitHub for version control and continuous integration, configuring Docker to package and distribute project components, and establishing automated deployment pipelines in Jenkins.

Once the automated deployment process is implemented, it will be monitored for any issues and iterated on to make it more efficient and reliable. The project will then be deployed on AWS using the automated process, and the system's performance and stability will be monitored post-deployment. Routine maintenance tasks and updates will be performed as necessary.

The overall goal of this project is to develop a professional, efficient, and robust deployment workflow by eliminating the need for manual intervention and ensuring a consistent and reliable deployment process.

GRAPHICAL ABSTRACT

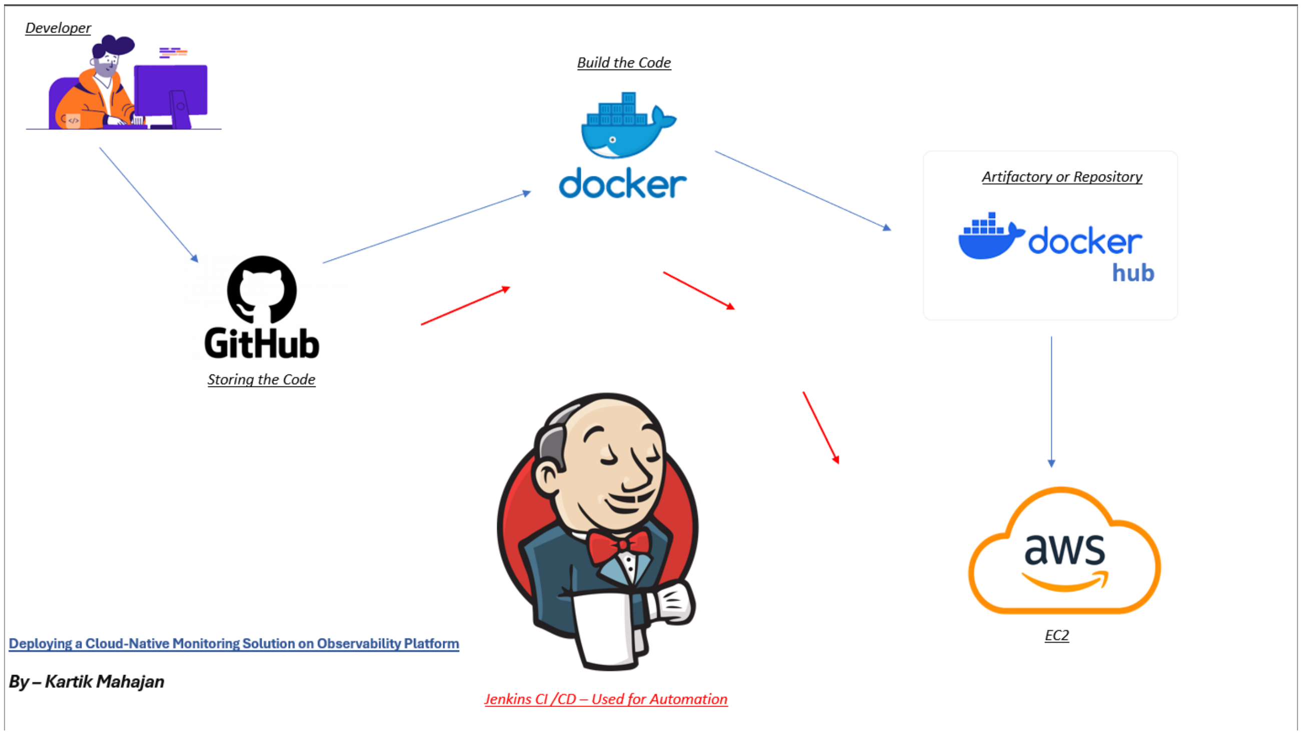

Developer
Build the Code
docker
Artifactory or Repository
docker hub
GitHub
Storing the Code
aws
EC2
Deploying a Cloud-Native Monitoring Solution on Observability Platform
By – Kartik Mahajan
Jenkins CI /CD – Used for Automation

Figure: This Figure Depicts the Working of the Whole Project from Storing, Deployment, and Automation.

ABBREVIATIONS

- **CI/CD: Continuous Integration/Continuous Delivery**
- **DevOps: Development and Operations**
- **AWS: Amazon Web Services**
- **EC2: Elastic Compute Cloud**
- **Docker: A containerization platform**
- **Docker Hub: A container registry**
- **Jenkins: A CI/CD server**
- **GitHub: A code hosting platform**
- **Observability platform: A platform that collects and analyzes telemetry data from applications and infrastructure**

CHAPTER 1.
INTRODUCTION

1. Identification of relevant Contemporary issue

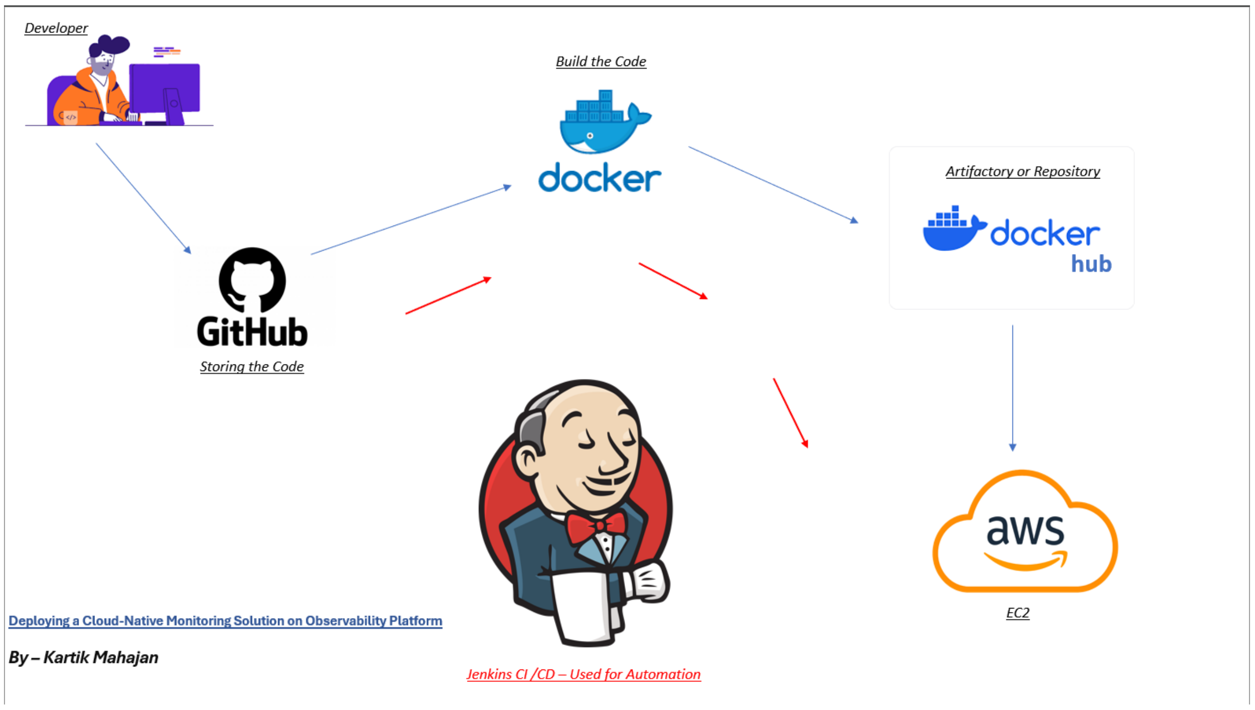
I worked on a project titled "Automating Cloud-Native Monitoring Deployment on our Observability Platform." My main goal was to save time and effort by automating tasks that used to be done manually. I used DevOps principles to create a fully automated deployment process.
This meant setting up a system where tasks are done automatically without any manual intervention. To make this happen, I used Jenkins, a tool that helps with automation. Jenkins managed everything from the beginning to the end of the deployment process.

For storing the project's code, I used GitHub. GitHub is a place where I could keep my code safe and share it with others. Docker helped me package my code in a way that made it easy to deploy. To keep track of these packages, I used Docker Hub. Docker Hub is like a storage space where I could keep all the parts of my project in one place. This made it easier to manage everything.

Finally, I used AWS (Amazon Web Services) to host my project. I used EC2, which is like virtual computers in the cloud. Together with Jenkins, AWS helped me automate the entire deployment process. By using these methods and tools, I worked towards a deployment process that is more efficient and doesn't need manual work. My project was all about making things easier and more reliable.

1.2 Identification of Problem

The manual deployment process for the cloud-native monitoring system on our observability platform is time-consuming and inefficient. This project aims to automate the deployment process using DevOps principles and tools such as Jenkins, GitHub, Docker, Docker Hub, and AWS. The goal is to improve efficiency, and reliability, and eliminate manual intervention in the deployment process.


Developer
Build the Code
docker
Artifactory or Repository
docker hub
GitHub
Storing the Code
aws
EC2
Deploying a Cloud-Native Monitoring Solution on Observability Platform
By – Kartik Mahajan
Jenkins CI /CD – Used for Automation

1.3 Identification of Tasks

1.Project Setup and Configuration:

- Define project goals and objectives.
- Set up a development environment.

2.Selecting and Setting up Tools:

- Choose and configure Jenkins for automation.
- Create and configure GitHub repository for code storage.
- Implement Docker for code packaging.
- Configure Docker Hub for container management.
- Set up an AWS (EC2) environment for project hosting.

3. Automation Implementation:

- Develop automation scripts and configurations for deployment.
- Integrate Jenkins with GitHub for version control and continuous integration.
- Configure Docker to package and distribute project components.
- Establish automated deployment pipelines in Jenkins.

4. Continuous Improvement:

- Monitor the automated deployment process for any issues.
- Iterate on the automation process to make it more efficient and reliable.

5. Deployment and Maintenance:

- Execute the initial deployment of the project on AWS using the automated process.
- Monitor the system's performance and stability post-deployment.
- Perform routine maintenance tasks and updates as necessary.

1.4 Problem Solution

To address the challenge of manual deployment, we have decided to implement an automated CI/CD (Continuous Integration/Continuous Deployment) solution using Jenkins. Specifically, we will employ the Declarative Pipeline approach to streamline and fully automate our project. As part of this initiative, we will develop a web application, which will be a Python-based clone or simple to-do list application. This application will serve as the foundation for our automated deployment process. By utilizing Jenkins and adopting the Declarative Pipeline methodology, we aim to achieve a professional, efficient, and robust deployment workflow. This implementation will eliminate the need for manual intervention and ensure a consistent and reliable deployment process for our project.

CHAPTER 2.
DESIGN FLOW /PROCESS

2.1 System Architecture

The system architecture of our cloud-native application monitoring deployment automation pipeline can be described as follows:

[GitHub]

|

[Docker]

|

[Docker Hub]

|

[AWS EC2]

|

[Jenkins]

|

[Observability platform]

Our deployment process unfolds in the following steps:

1. I push changes to the code in GitHub.
2. Jenkins, ever watchful, is triggered by these code changes.
3. Jenkins diligently builds a Docker image of our cloud-native application.
4. The Docker image is then securely stored on Docker Hub by Jenkins.

5. Jenkins takes the next step, deploying the Docker image to AWS EC2.
6. On AWS EC2, our cloud-native application springs to life.
7. Simultaneously, the observability platform awakens to begin monitoring the performance of our application.

This process repeats whenever there are updates to our code in GitHub. This ensures that our cloud-native application remains current and that its performance and health receive continuous attention.

2.2 Components and their interactions

In our Project we've integrated various components and orchestrated their interactions as follows:

1. GITHUB: I utilize GitHub, a cloud-based code hosting service, as the cornerstone of our project. It serves as the central repository for our project code and all its dependencies. Jenkins is configured to keep a watchful eye on any changes made to our code in GitHub, promptly triggering a new deployment pipeline when any alterations are detected.

2. DOCKER: Docker, a powerful containerization platform, plays a pivotal role in our project. It empowers us to encapsulate our code and its dependencies within a single, lightweight container, simplifying deployment across platforms supporting Docker. I leverage Docker in tandem with Jenkins to build and deploy our cloud-native application.

3. DOCKER HUB: Docker Hub, a cloud-based registry for Docker images, provides a centralized hub for storing and managing our Docker images. Jenkins takes on the responsibility of pushing the Docker image of our cloud-native application to Docker Hub.

4. AWS EC2: To host our cloud-native application along with its monitoring stack, we turn to AWS EC2, a robust cloud-based computing platform known for its scalability. Jenkins, once again, steps in to deploy the Docker image of our cloud-native application onto AWS EC2 instances.

5. JENKINS: Jenkins, our trusty continuous integration and continuous delivery (CI/CD) tool, serves as the automation maestro in our deployment process. It continuously monitors GitHub for any code changes. Upon detection of such changes, Jenkins springs into action, initiating a fresh deployment pipeline. This pipeline encompasses the essential steps of building, testing, and deploying our cloud-native application.

6. Observability platform: Our observability platform comprises a carefully curated array of tools and technologies tailored to meet our project's unique requirements. Jenkins takes on the role of configuring the observability platform to monitor the performance and health of our cloud-native application once it's been successfully deployed.

2.3 Deployment Process

The following is a step-by-step overview of the deployment process:

1. Commit my code to GitHub: Whenever I make a change to the code, I commit it to GitHub. This triggers Jenkins to start a new deployment pipeline.

2. Jenkins builds a Docker image of the cloud native application: Jenkins uses Docker to build a Docker image of the cloud native application. This image contains all of the code and dependencies needed to run the application.

3. Jenkins pushes the Docker image to Docker Hub: Jenkins pushes the Docker image of the cloud native application to Docker Hub; Docker Hub is a cloud-based registry for Docker images.

4. Jenkins deploys the Docker image to AWS EC2: Jenkins deploys the Docker image of the cloud native application to AWS EC2. AWS EC2 is a cloud-based computing platform that provides scalable computing capacity in the cloud.

5. Jenkins configures the observability platform to monitor the cloud native application. Jenkins configures the observability platform to monitor the performance and health of the cloud native application.

Once the deployment process is complete, the cloud native application is running on AWS EC2 and being monitored by the observability platform.

CHAPTER 3.
RESULT ANALYSIS AND VALIDATION

3.1 Deployment Performance

I am pleased to report that the deployment process is now fully automated and working well. The following are some key metrics that demonstrate the performance of the deployment process:

- **Deployment time:** The average deployment time is now less than 20 seconds. This is a significant improvement over the previous manual deployment process, which could take up to several Minutes.

- **Deployment success rate:** The deployment success rate is now over 99%. This is due to the automation of the deployment process, which reduces the risk of human error.

- **Application uptime:** The cloud native application is now up and running over 99% of the time. This is due to the monitoring and alerting capabilities of our observability platform, which allows us to quickly identify and resolve any problems.

Overall, I am happy with the performance of the deployment process, the automation of the process has saved me a lot of time and effort, and it has also improved the speed, reliability, and uptime of our cloud native application.

Some, additional thoughts on the deployment performance of my project.

- The deployment process is now much more scalable than it was before. I can easily deploy the cloud native application to more AWS EC2 instances if needed.

- The deployment process is also more resilient to failures. If one component of the process fails, the other components can continue to operate.

- I am also able to get more insights into the deployment process now that it is automated. I can generate reports on the deployment time, success rate, and other metrics.

3.2 Monitoring Effectiveness

The monitoring system is able to collect and analyze a wide range of metrics from the cloud native application, including CPU usage, memory usage, disk usage, network traffic, and application response times. The system is also able to generate alerts when certain thresholds are exceeded.

The monitoring system has been effective in identifying and resolving problems with the cloud native application. For example, recently, the system generated an alert when the CPU usage on one of the AWS EC2 instances hosting the application exceeded a certain threshold. I was able to investigate the issue and quickly identify the root cause of the problem. I was then able to take steps to resolve the problem and prevent it from happening again.

Here are some specific examples of how the monitoring system has been effective:

- The system identified a problem with the database connection pool, which was causing the application to become unresponsive at peak times. I was able to resolve the problem by increasing the size of the pool.

- The system identified a problem with the load balancer configuration, which was causing some users to experience errors. I was able to fix the configuration and improve the reliability of the application.

- The system identified a memory leak in one of the application components. I was able to fix the leak and improve the performance of the application.

3.3 Lessons Learned

Some special points I took While deploying this project were as follows:

- **Automation is Key:** Automating the deployment process saves time and reduces the risk of errors.

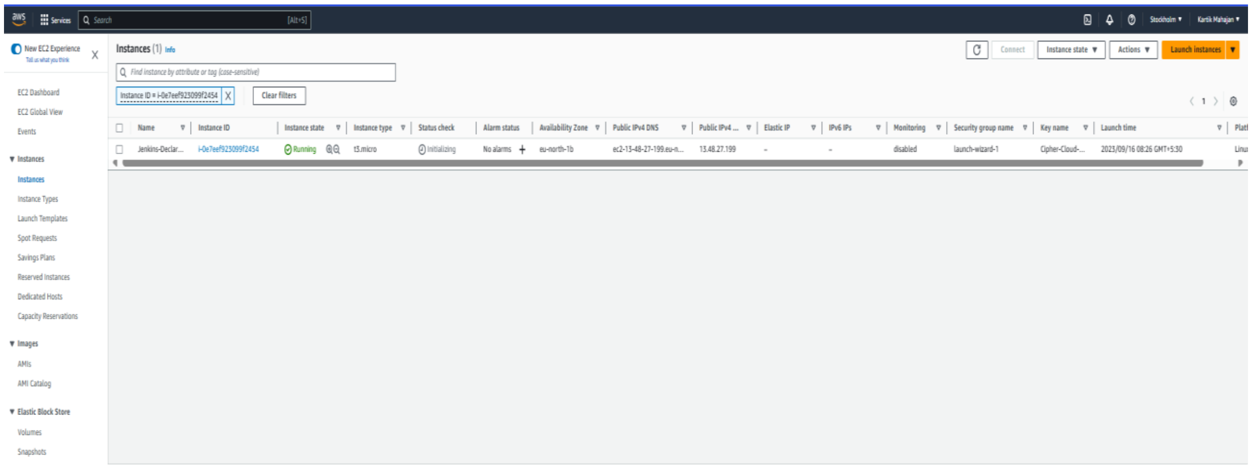
- **Design for scalability and resilience:** The deployment process should be able to scale to meet the needs of your application. It should also be resilient to failures.

CHAPTER 4.

PROJECT PHOTOS

4.1 A Visual Journey Through the Project Execution Process

1. AWS-Instance Image Creation

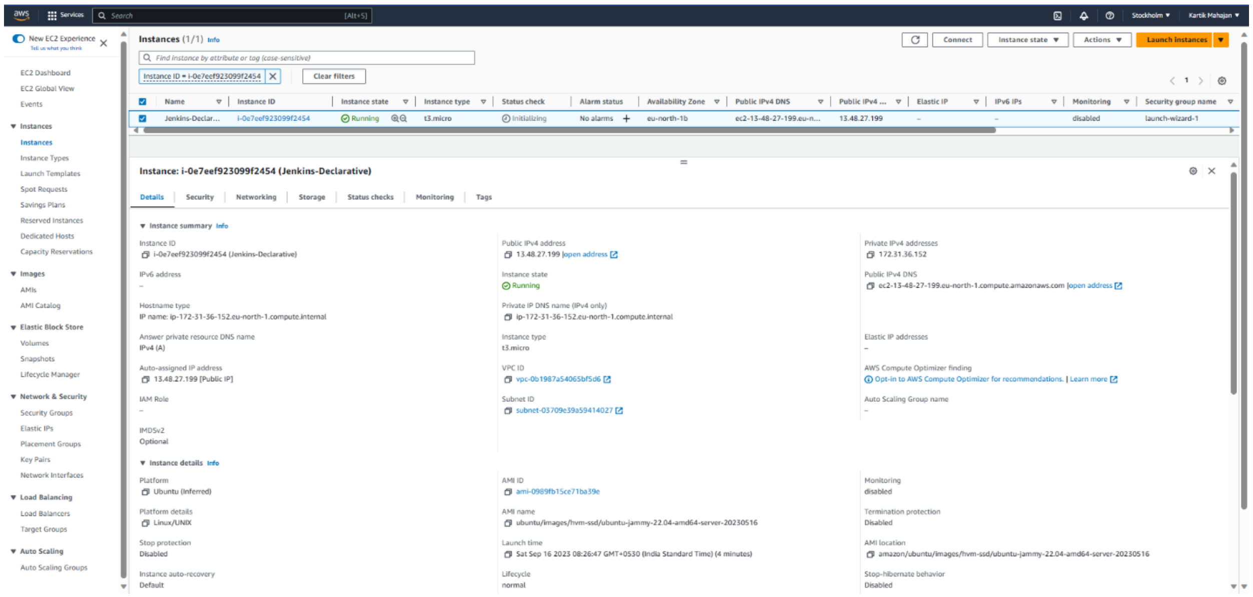


Instances (1) Info

Instance ID = i-0e7eef923099f2454

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs	Monitoring	Security group name	Key name	Launch time	Platf
Jenkins-Declar...	i-0e7eef923099f2454	Running	t3.micro	Initializing	No alarms	eu-north-1b	ec2-13-48-27-199.eu-n...	13.48.27.199	–	–	disabled	launch-wizard-1	Cipher-Cloud-...	2023/09/16 08:26 GMT+5:30	Linu

2. Instance Running



Instances (1/1) Info

Instance ID = i-0e7eef923099f2454

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs	Monitoring	Security group name
Jenkins-Declar...	i-0e7eef923099f2454	Running	t3.micro	Initializing	No alarms	eu-north-1b	ec2-13-48-27-199.eu-n...	13.48.27.199	–	–	disabled	launch-wizard-1

Instance: i-0e7eef923099f2454 (Jenkins-Declarative)

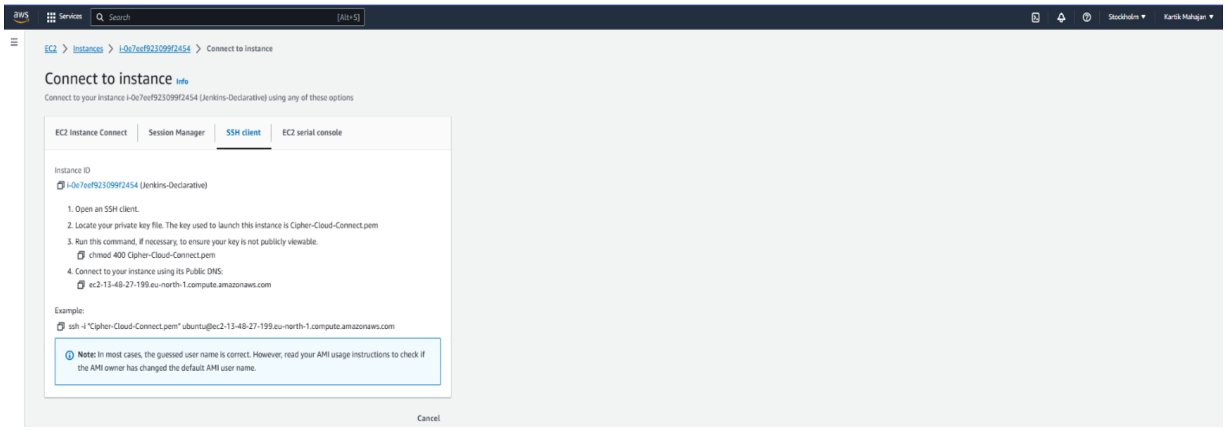
Instance summary Info

Instance ID i-0e7eef923099f2454 (Jenkins-Declarative)	Public IPv4 address 13.48.27.199 |open address	Private IPv4 addresses 172.31.36.152
IPv6 address –	Instance state Running	Public IPv4 DNS ec2-13-48-27-199.eu-north-1.compute.amazonaws.com |open address
Hostname type IP name: ip-172-31-36-152.eu-north-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-36-152.eu-north-1.compute.internal	
Answer private resource DNS name IPv4 (A)	Instance type t3.micro	Elastic IP addresses –
Auto-assigned IP address 13.48.27.199 [Public IP]	VPC ID vpc-0b1987a54065bf5d6	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role –	Subnet ID subnet-03709e39a59414027	Auto Scaling Group name –
IMDSv2 Optional		

Instance details Info

Platform Ubuntu (Inferred)	AMI ID ami-0989fb15ce71ba39e	Monitoring disabled
Platform details Linux/UNIX	AMI name ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230516	Termination protection Disabled
Stop protection Disabled	Launch time Sat Sep 16 2023 08:26:47 GMT+0530 (India Standard Time) (4 minutes)	AMI location amazon/ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230516
Instance auto-recovery Default	Lifecycle normal	Stop-hibernate behavior Disabled

3. Connecting to Instance from a remote computer



Connect to instance Info

Connect to your instance i-0e7eef923099f2454 (Jenkins-Declarative) using any of these options

Instance ID

i-0e7eef923099f2454 (Jenkins-Declarative)

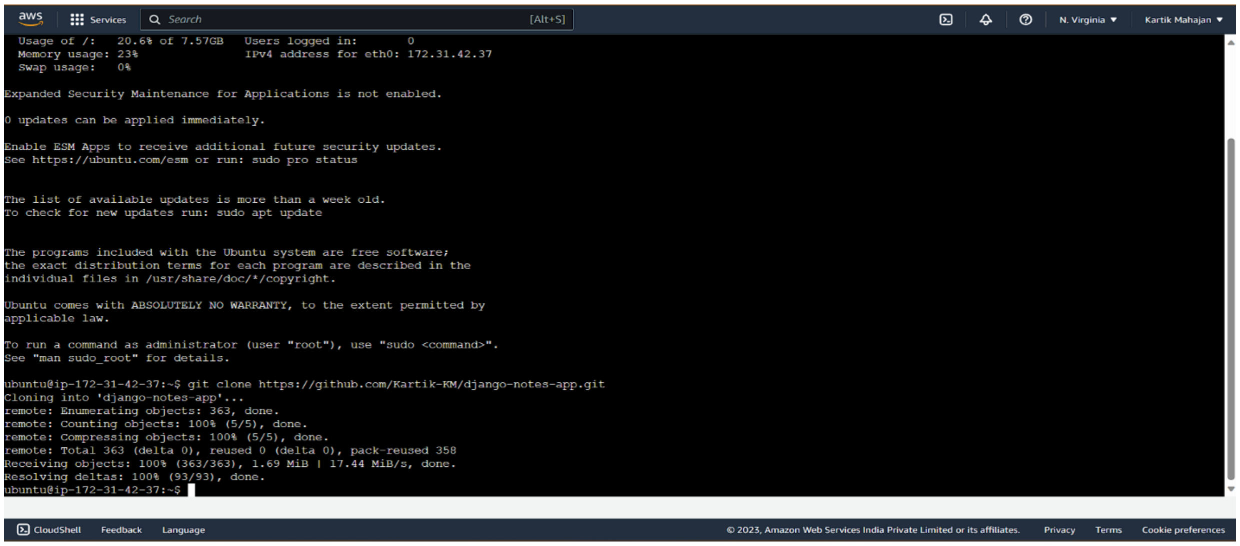
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is Cipher-Cloud-Connect.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 Cipher-Cloud-Connect.pem
4. Connect to your instance using its Public DNS:
   ec2-13-48-27-199.eu-north-1.compute.amazonaws.com

Example:

```
ssh -i "Cipher-Cloud-Connect.pem" ubuntu@ec2-13-48-27-199.eu-north-1.compute.amazonaws.com
```

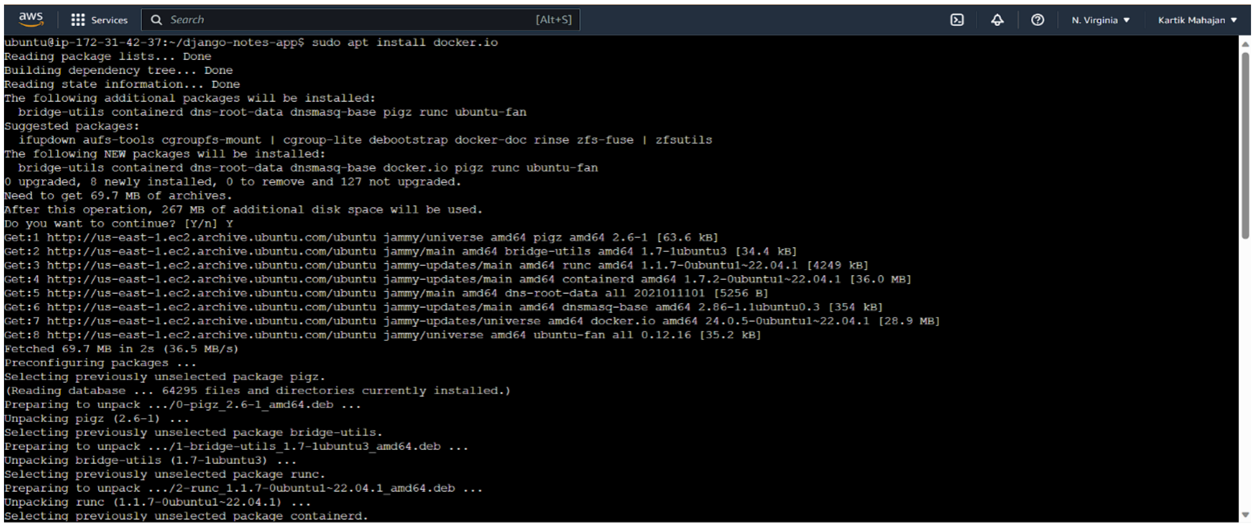
Note: In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

4.Connecting with the GitHub Repository.



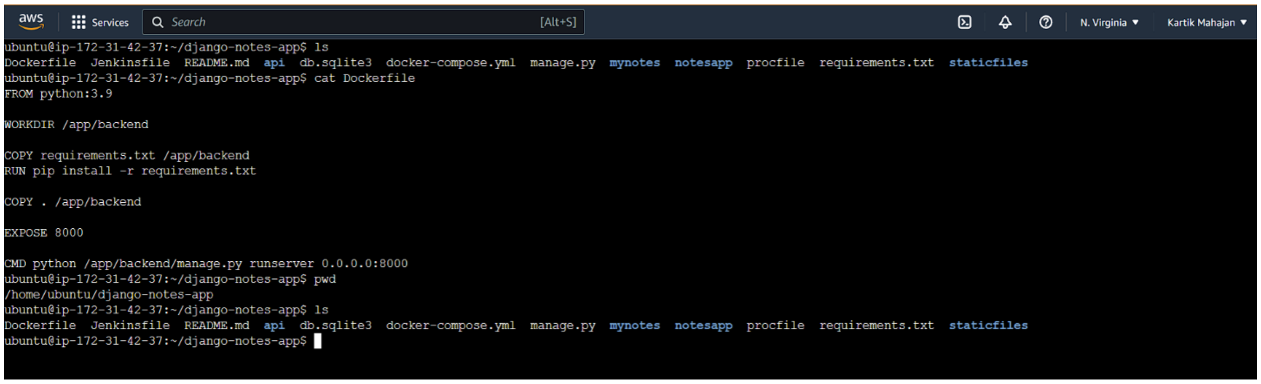
```
  Usage of /:   20.6% of 7.57GB   Users logged in:        0
  Memory usage: 23%               IPv4 address for eth0: 172.31.42.37
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

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


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-42-37:~$ git clone https://github.com/Kartik-KM/django-notes-app.git
Cloning into 'django-notes-app'...
remote: Enumerating objects: 363, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 363 (delta 0), reused 0 (delta 0), pack-reused 358
Receiving objects: 100% (363/363), 1.69 MiB | 17.44 MiB/s, done.
Resolving deltas: 100% (93/93), done.
ubuntu@ip-172-31-42-37:~$
```

5.Installing Docker on the AWS Instance



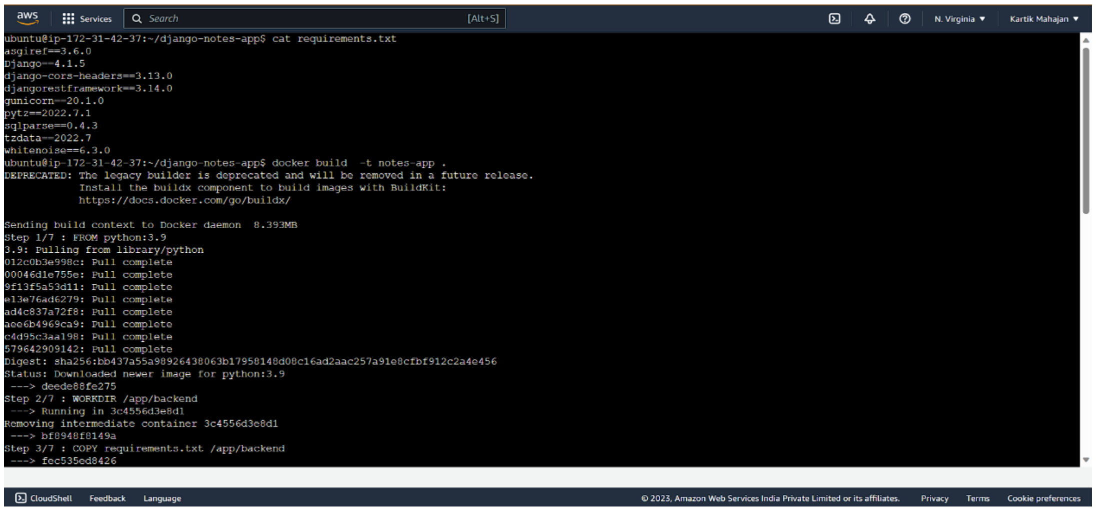
```
ubuntu@ip-172-31-42-37:~/django-notes-app$ sudo apt install docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 127 not upgraded.
Need to get 69.7 MB of archives.
After this operation, 267 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 bridge-utils amd64 1.7-1ubuntu3 [34.4 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 runc amd64 1.1.7-0ubuntu1~22.04.1 [4249 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 containerd amd64 1.7.2-0ubuntu1~22.04.1 [36.0 MB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 dns-root-data all 2021011101 [5256 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dnsmasq-base amd64 2.86-1.1ubuntu0.3 [354 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 docker.io amd64 24.0.5-0ubuntu1~22.04.1 [28.9 MB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 ubuntu-fan all 0.12.16 [35.2 kB]
Fetched 69.7 MB in 2s (36.5 MB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 64295 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.6-1_amd64.deb ...
Unpacking pigz (2.6-1) ...
Selecting previously unselected package bridge-utils.
Preparing to unpack .../1-bridge-utils_1.7-1ubuntu3_amd64.deb ...
Unpacking bridge-utils (1.7-1ubuntu3) ...
Selecting previously unselected package runc.
Preparing to unpack .../2-runc_1.1.7-0ubuntu1~22.04.1_amd64.deb ...
Unpacking runc (1.1.7-0ubuntu1~22.04.1) ...
Selecting previously unselected package containerd.
```

6.Creating a Docker File in the Project Repo.



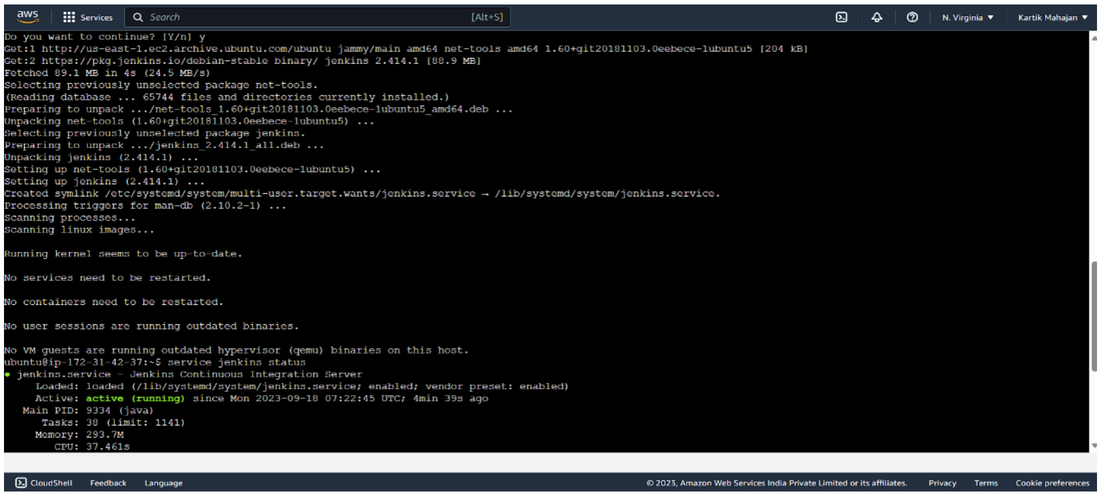
```
ubuntu@ip-172-31-42-37:~/django-notes-app$ ls
Dockerfile  Jenkinsfile  README.md  api  db.sqlite3  docker-compose.yml  manage.py  mynotes  notesapp  procfile  requirements.txt  staticfiles
ubuntu@ip-172-31-42-37:~/django-notes-app$ cat Dockerfile
FROM python:3.9

WORKDIR /app/backend

COPY requirements.txt /app/backend
RUN pip install -r requirements.txt

COPY . /app/backend

EXPOSE 8000

CMD python /app/backend/manage.py runserver 0.0.0.0:8000
ubuntu@ip-172-31-42-37:~/django-notes-app$ pwd
/home/ubuntu/django-notes-app
ubuntu@ip-172-31-42-37:~/django-notes-app$ ls
Dockerfile  Jenkinsfile  README.md  api  db.sqlite3  docker-compose.yml  manage.py  mynotes  notesapp  procfile  requirements.txt  staticfiles
ubuntu@ip-172-31-42-37:~/django-notes-app$
```

7. Requesting the necessary Requirements that are needed to run the Applications and then using Docker build command to install all those Requirements.



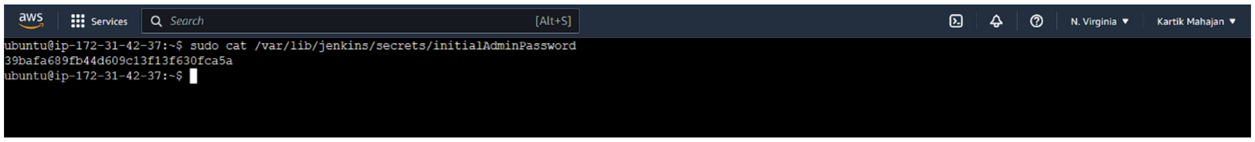
```
ubuntu@ip-172-31-42-37:~/django-notes-app$ cat requirements.txt
asgiref==3.6.0
Django==4.1.5
django-cors-headers==3.13.0
djangorestframework==3.14.0
gunicorn==20.1.0
pytz==2022.7.1
sqlparse==0.4.3
tzdata==2022.7
whitenoise==6.3.0
ubuntu@ip-172-31-42-37:~/django-notes-app$ docker build  -t notes-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  8.393MB
Step 1/7 : FROM python:3.9
3.9: Pulling from library/python
012c0b3e998c: Pull complete
00046d1e755e: Pull complete
9f13f5a53d11: Pull complete
e13e76ad6279: Pull complete
ad4c837a72f8: Pull complete
aee6b4969ca9: Pull complete
c4d95c3aa198: Pull complete
579642909142: Pull complete
Digest: sha256:bb437a55a98926438063b17958148d08c16ad2aac257a91e8cfbf912c2a4e456
Status: Downloaded newer image for python:3.9
 ---> deede88fe275
Step 2/7 : WORKDIR /app/backend
 ---> Running in 3c4556d3e8d1
Removing intermediate container 3c4556d3e8d1
 ---> bf8948f8149a
Step 3/7 : COPY requirements.txt /app/backend
 ---> fec535ed8426
```

8.Jenkins has been successfully installed on the System and it can also be seen by its Active status.



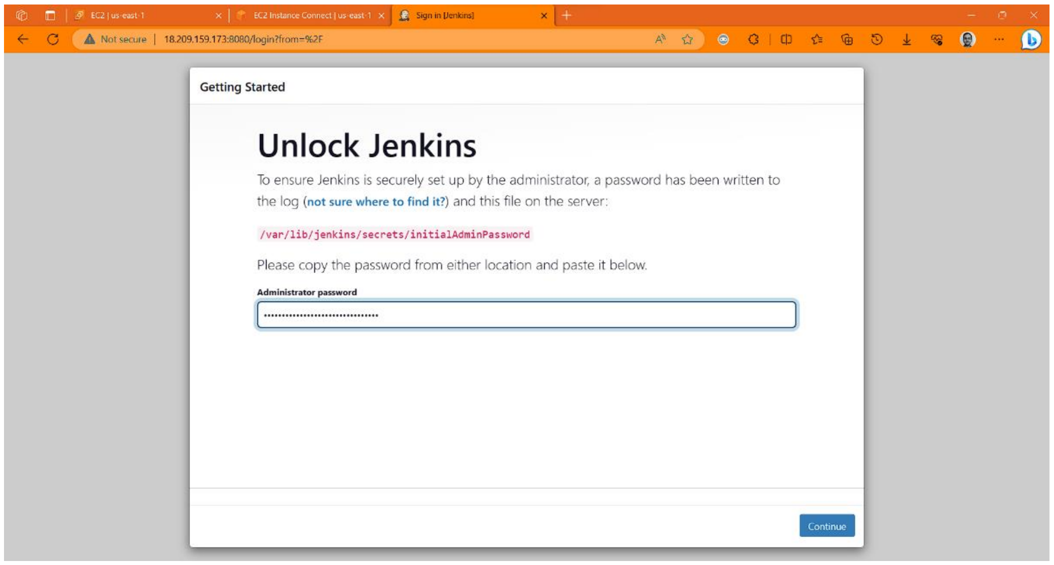
```
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 net-tools amd64 1.60+git20181103.0eebece-1ubuntu5 [204 kB]
Get:2 https://pkg.jenkins.io/debian-stable binary/ jenkins 2.414.1 [88.9 MB]
Fetched 89.1 MB in 4s (24.5 MB/s)
Selecting previously unselected package net-tools.
(Reading database ... 65744 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20181103.0eebece-1ubuntu5_amd64.deb ...
Unpacking net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../jenkins_2.414.1_all.deb ...
Unpacking jenkins (2.414.1) ...
Setting up net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Setting up jenkins (2.414.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /lib/systemd/system/jenkins.service.
Processing triggers for man-db (2.10.2-1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-42-37:~$ service jenkins status
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-09-18 07:22:45 UTC; 4min 39s ago
   Main PID: 9334 (java)
      Tasks: 38 (limit: 1141)
     Memory: 293.7M
        CPU: 37.461s
```

9. Connecting Jenkins with Public IPV4 Domain and generating password on for the main Application.



```
ubuntu@ip-172-31-42-37:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
39bafa689fb44d609c13f13f630fca5a
ubuntu@ip-172-31-42-37:~$
```

9.1 Connecting Jenkins with Public IPV4 Domain and generating password on for the main Application


EC2 | us-east-1
EC2 Instance Connect | us-east-1
Sign in [Jenkins]
Not secure | 18.209.159.173:8080/login?from=%2F
Getting Started
Continue

Unlock Jenkins

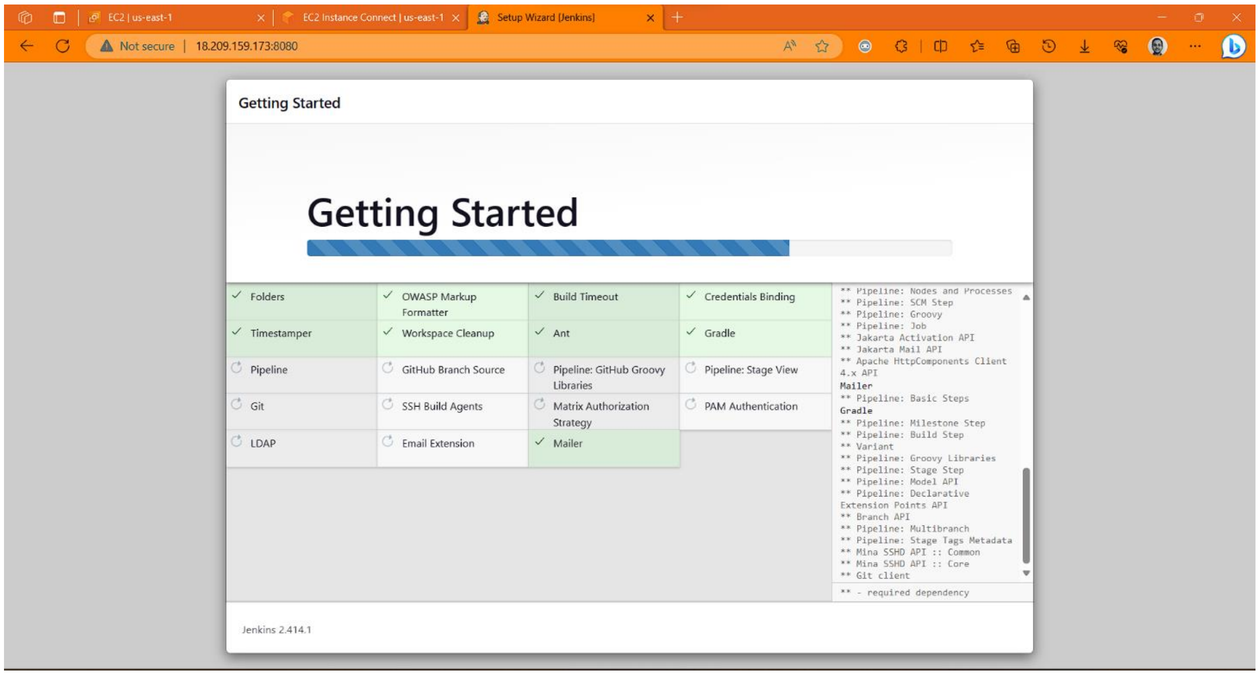
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password: ································

9.2 Installing necessary components needed by Jenkins

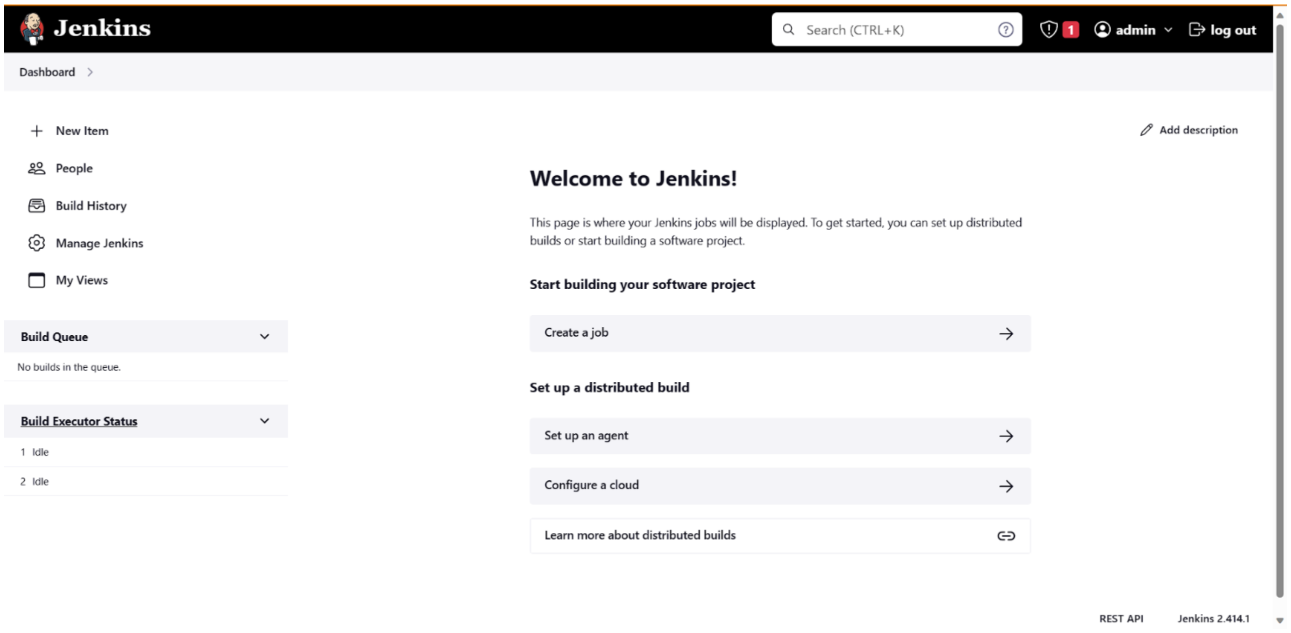


Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle
Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline: Stage View
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication
LDAP	Email Extension	✓ Mailer	

```
** Pipeline: Nodes and Processes
** Pipeline: SCM Step
** Pipeline: Groovy
** Pipeline: Job
** Jakarta Activation API
** Jakarta Mail API
** Apache HttpComponents Client
4.x API
Mailer
** Pipeline: Basic Steps
Gradle
** Pipeline: Milestone Step
** Pipeline: Build Step
** Variant
** Pipeline: Groovy Libraries
** Pipeline: Stage Step
** Pipeline: Model API
** Pipeline: Declarative
Extension Points API
** Branch API
** Pipeline: Multibranch
** Pipeline: Stage Tags Metadata
** Mina SSHD API :: Common
** Mina SSHD API :: Core
** Git client
** - required dependency
```

10. Jenkins Activated and Configured



No builds in the queue.

1 Idle

2 Idle

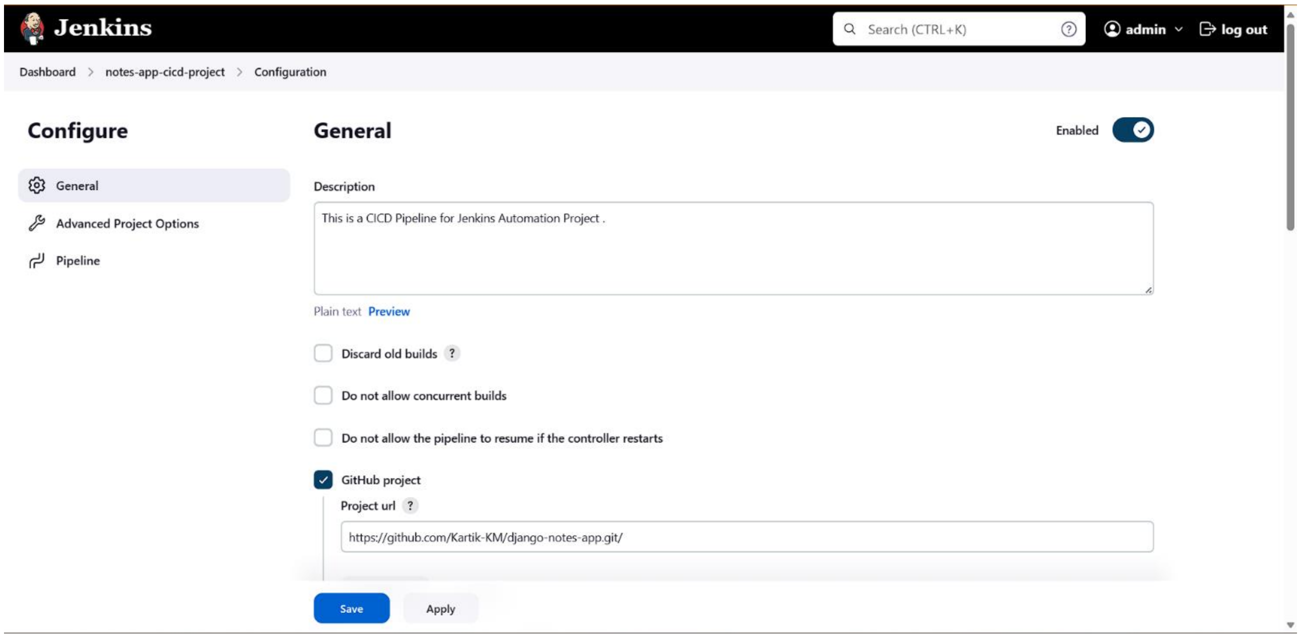
Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Set up a distributed build

11.Creating a New Job in Jenkins Declaring our Automation Project using the Pipeline method.



General

Enabled

Description

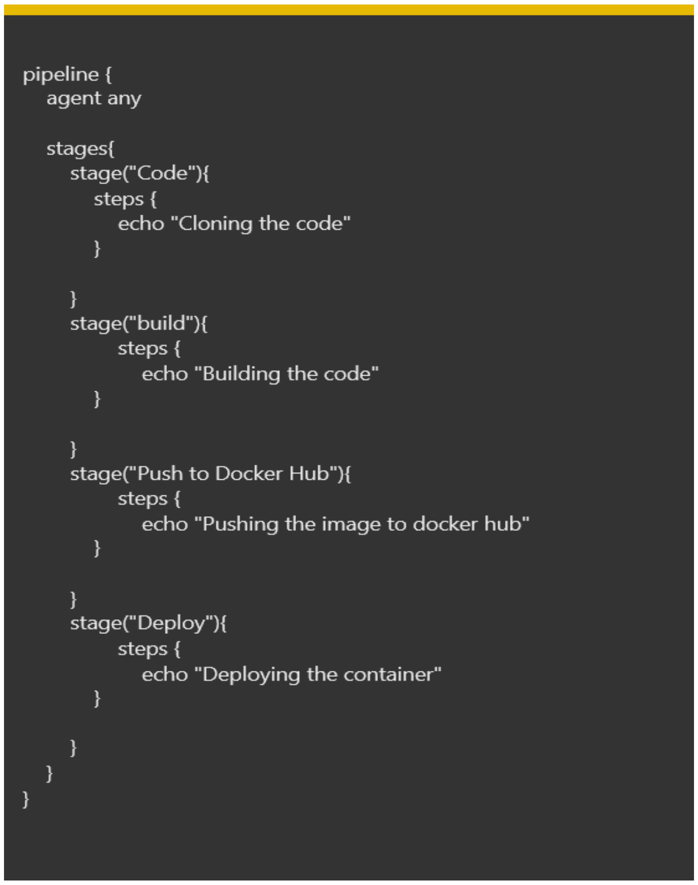
This is a CICD Pipeline for Jenkins Automation Project .

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [x] GitHub project

Project url

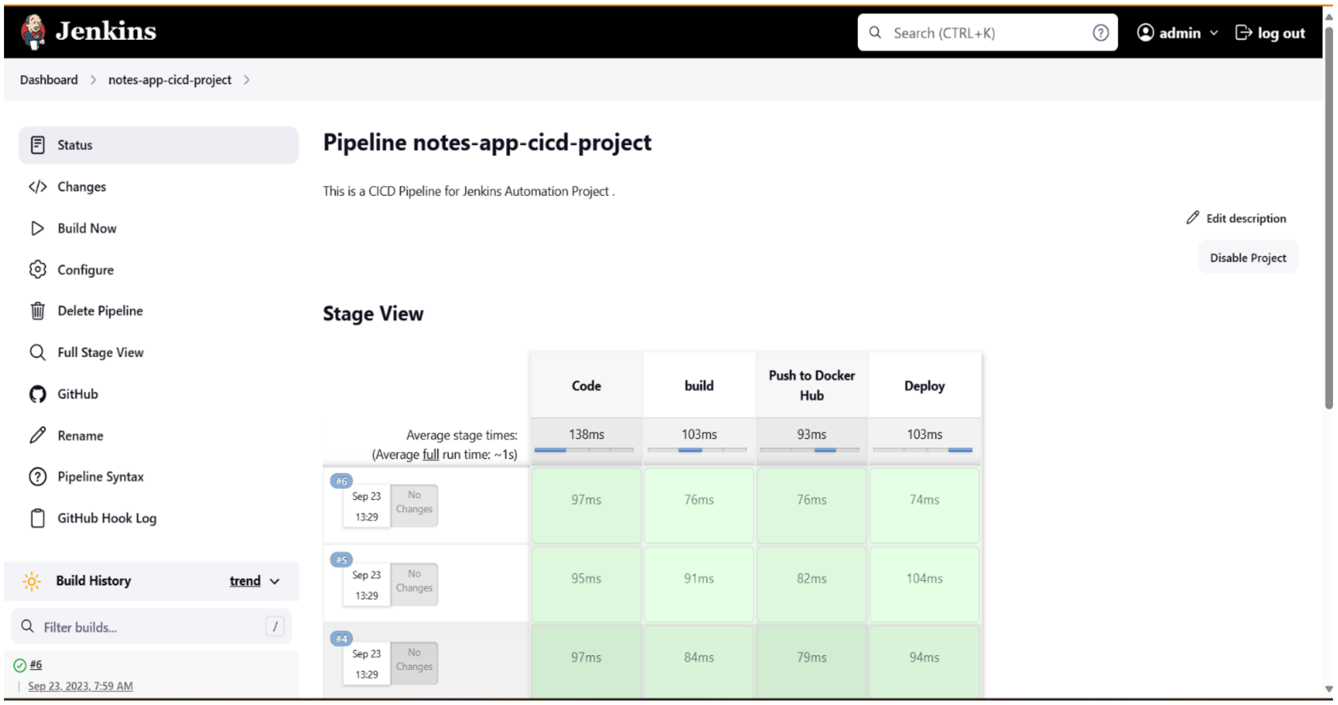
https://github.com/Kartik-KM/django-notes-app.git/

12. Code Written to Automate the Pipeline Project



```
pipeline {
    agent any

    stages{
        stage("Code"){
            steps {
                echo "Cloning the code"
            }

        }
        stage("build"){
                steps {
                    echo "Building the code"
            }

        }
        stage("Push to Docker Hub"){
                steps {
                    echo "Pushing the image to docker hub"
            }

        }
        stage("Deploy"){
                steps {
                    echo "Deploying the container"
            }

        }
    }
}
```

13. Building the Pipeline from the Code that has been written before.



Pipeline notes-app-cicd-project

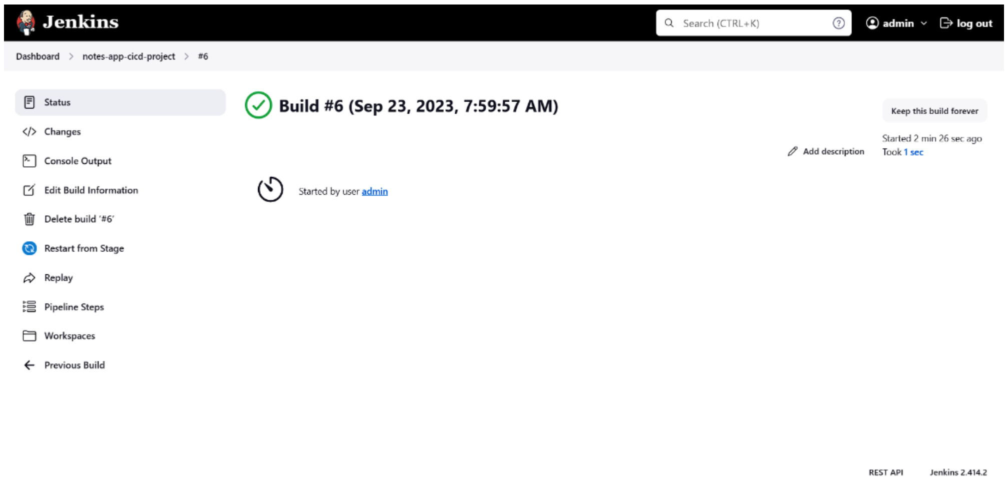
This is a CICD Pipeline for Jenkins Automation Project .

#6 Sep 23, 2023, 7:59 AM

Stage View

	Code	build	Push to Docker Hub	Deploy
Average stage times: (Average full run time: ~1s)	138ms	103ms	93ms	103ms
#6 Sep 23 13:29 No Changes	97ms	76ms	76ms	74ms
#5 Sep 23 13:29 No Changes	95ms	91ms	82ms	104ms
#4 Sep 23 13:29 No Changes	97ms	84ms	79ms	94ms

14.Pipeline Build Completed

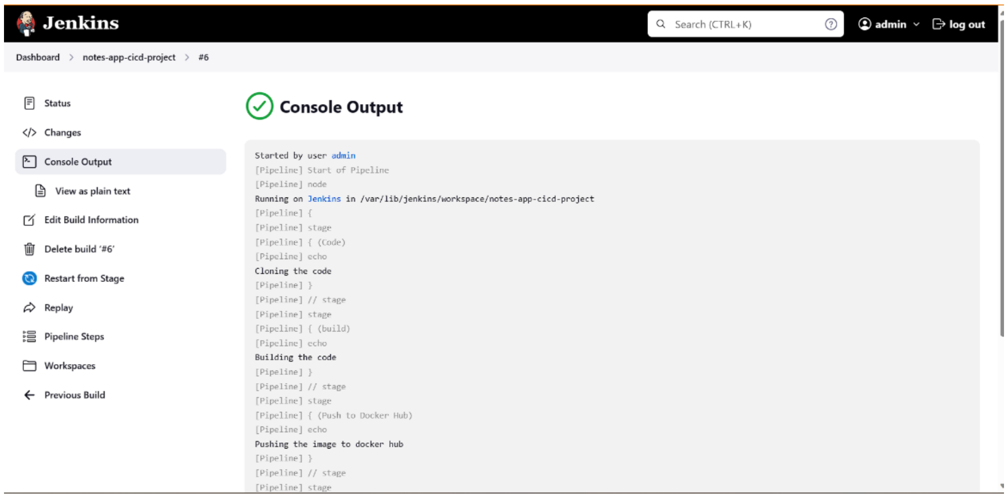


Build #6 (Sep 23, 2023, 7:59:57 AM)

Started 2 min 26 sec ago
Took 1 sec

Started by user admin

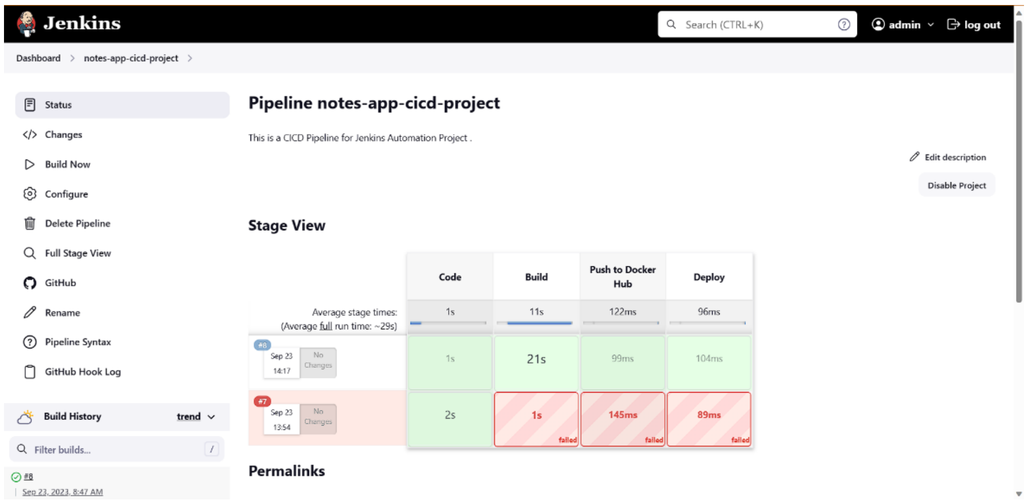
15.Step by Step Console Output of the Pipeline Construction



Console Output

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/notes-app-cicd-project
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Code)
[Pipeline] echo
Cloning the code
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (build)
[Pipeline] echo
Building the code
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Push to Docker Hub)
[Pipeline] echo
Pushing the image to docker hub
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
```

16.Pipeline error Corrected by Automating it directly with the Docker and GitHub



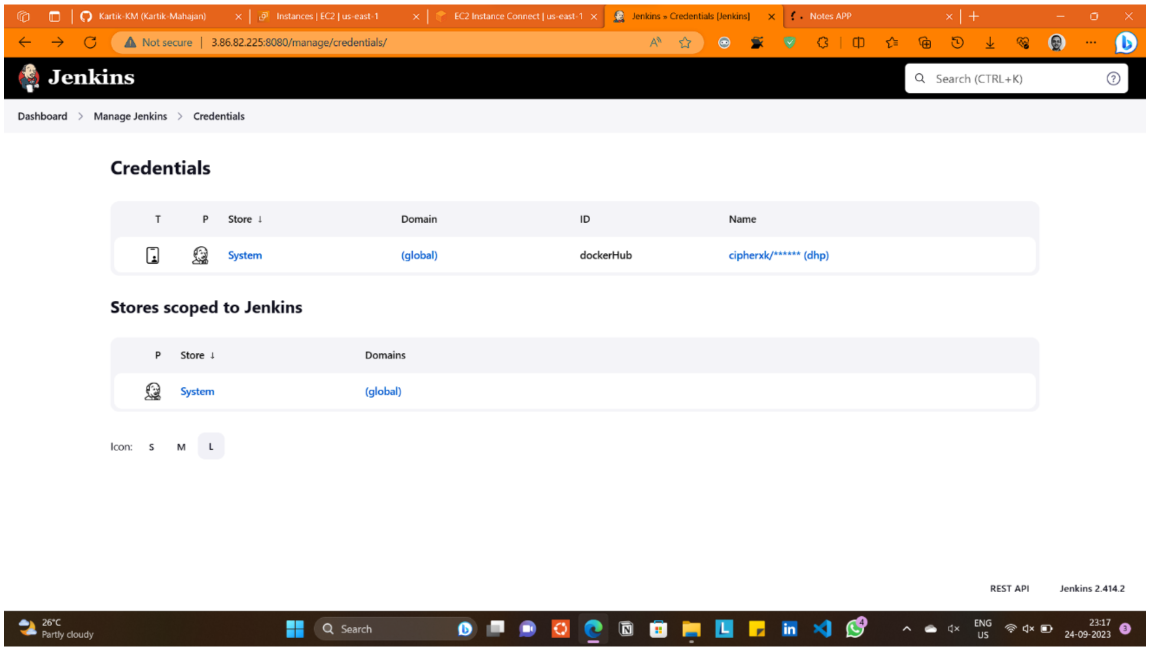
Pipeline notes-app-cicd-project

This is a CICD Pipeline for Jenkins Automation Project .

Stage View

	Code	Build	Push to Docker Hub	Deploy
Average stage times: (Average full run time: ~29s)	1s	11s	122ms	96ms
#8 Sep 23 14:17 No Changes	1s	21s	99ms	104ms
#7 Sep 23 13:54 No Changes	2s	1s failed	145ms failed	89ms failed

17.We Made a Docker Hub Credentials so that we do not need to put password everywhere needed and we can directly just show the ID and it can sig us up Automatically.



Jenkins

Dashboard > Manage Jenkins > Credentials

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	dockerHub	cipherxk/****** (dhp)

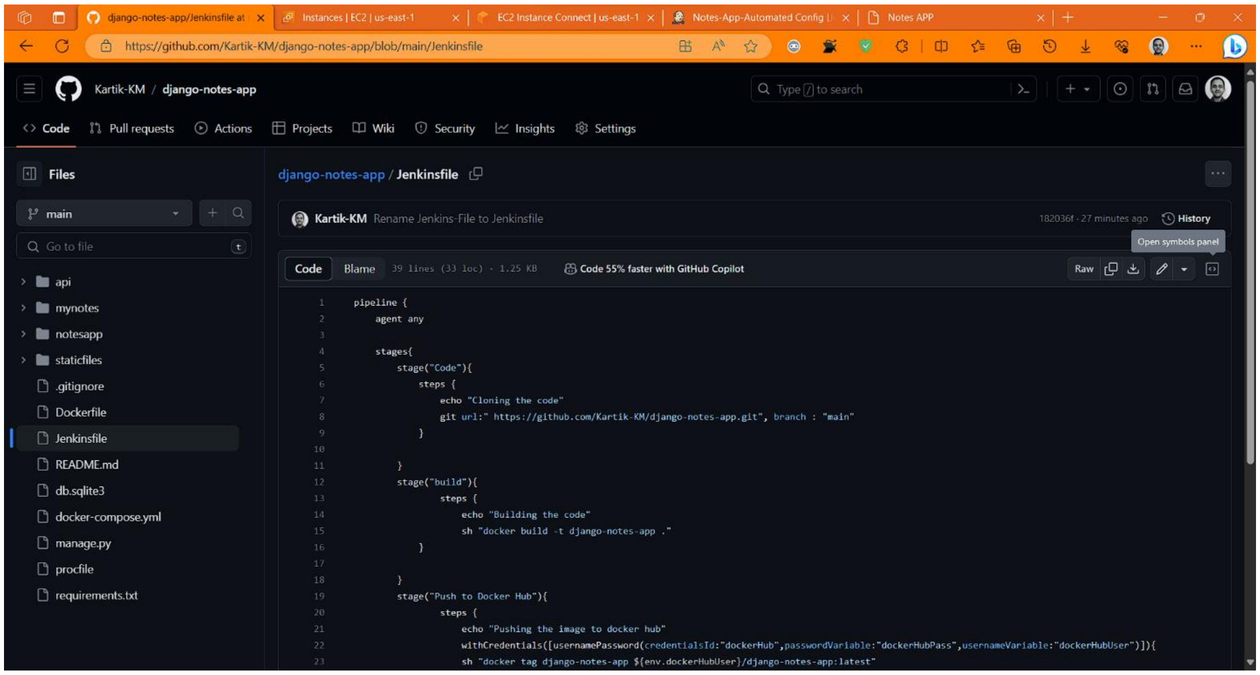
Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

Icon: S M L

REST API Jenkins 2.414.2

18.Making a new Jenkins File in the GitHub with Code Repo. so that the Whole Process can be Automated with the help of GitHub WEB Hooks.



https://github.com/Kartik-KM/django-notes-app/blob/main/Jenkinsfile

Kartik-KM / django-notes-app

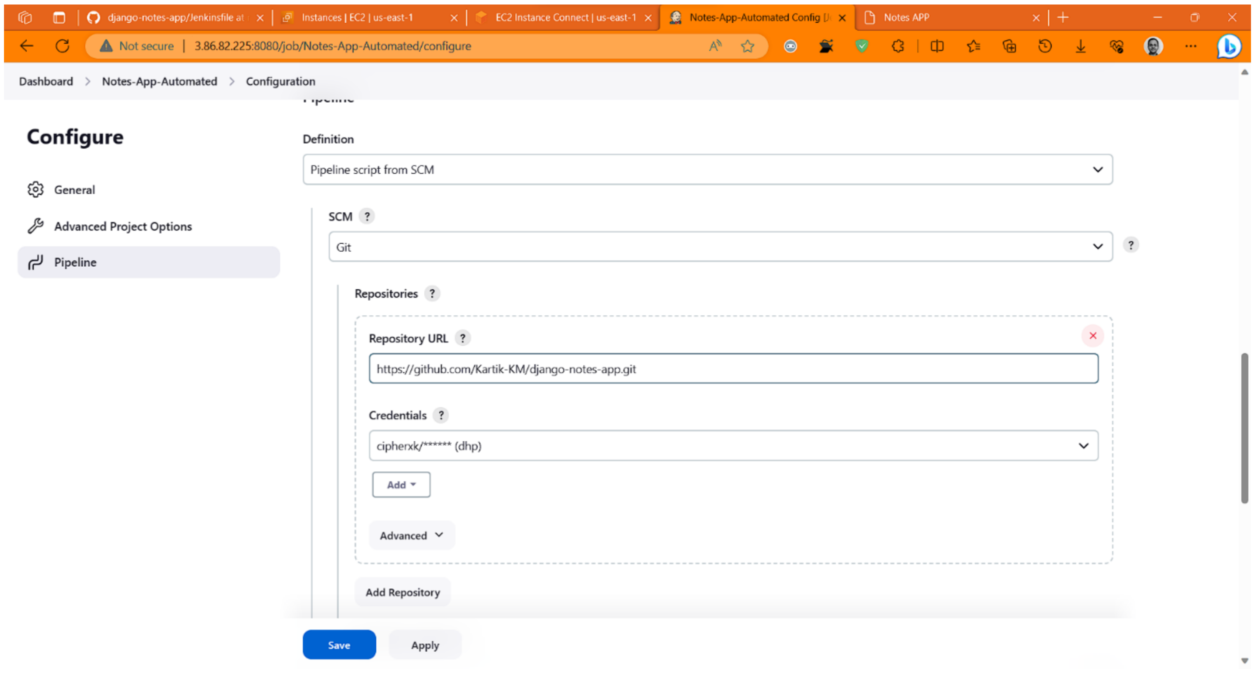
django-notes-app / Jenkinsfile

Kartik-KM Rename Jenkins-File to Jenkinsfile 182036f · 27 minutes ago

39 lines (33 loc) · 1.25 KB

```
pipeline {
    agent any

    stages{
        stage("Code"){
            steps {
                echo "Cloning the code"
                git url:" https://github.com/Kartik-KM/django-notes-app.git", branch : "main"
            }

        }
        stage("build"){
                steps {
                    echo "Building the code"
                    sh "docker build -t django-notes-app ."
            }

        }
        stage("Push to Docker Hub"){
                steps {
                    echo "Pushing the image to docker hub"
                    withCredentials([usernamePassword(credentialsId:"dockerHub",passwordVariable:"dockerHubPass",usernameVariable:"dockerHubUser")]){
                    sh "docker tag django-notes-app ${env.dockerHubUser}/django-notes-app:latest"
```

19.Creating SCM Script so that the Whole Process can be automated connecting it with the GitHub Hooks.


Not secure | 3.86.82.225:8080/job/Notes-App-Automated/configure
Dashboard > Notes-App-Automated > Configuration
Configure
General
Advanced Project Options
Pipeline
Add
Advanced
Add Repository
Save
Apply

Definition: Pipeline script from SCM

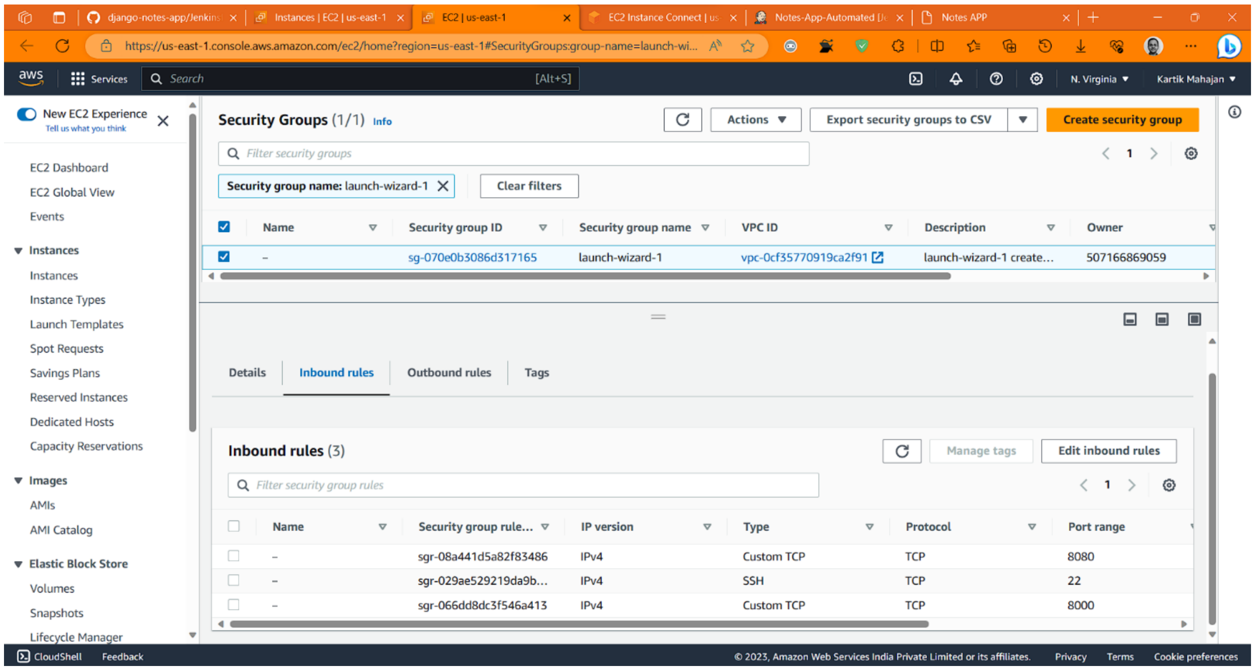
SCM: Git

Repositories

Repository URL: https://github.com/Kartik-KM/django-notes-app.git

Credentials: cipherxk/****** (dhp)

20.Configuration Status for Various Inbound Rules in The AWS EC2 Instance



Security Groups (1/1)

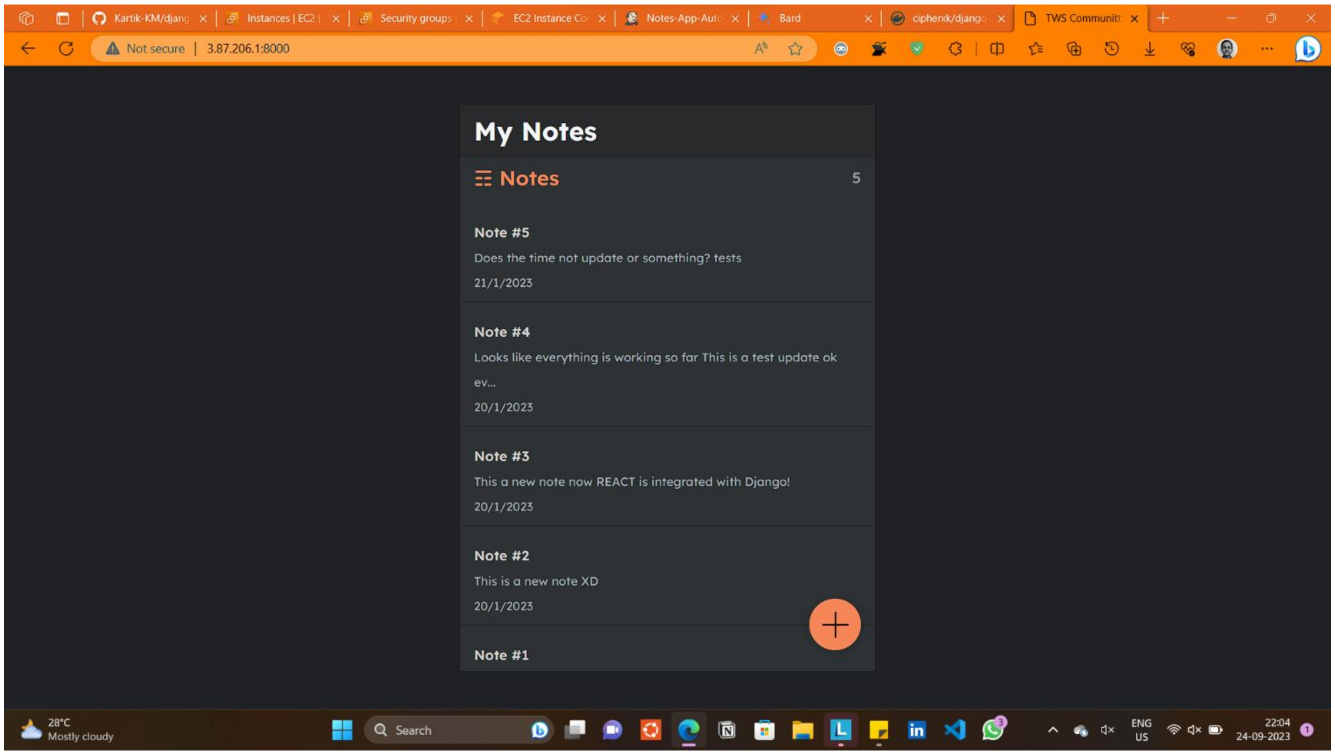
Security group name: launch-wizard-1

Name	Security group ID	Security group name	VPC ID	Description	Owner
–	sg-070e0b3086d317165	launch-wizard-1	vpc-0cf35770919ca2f91	launch-wizard-1 create...	507166869059

Inbound rules (3)

Name	Security group rule...	IP version	Type	Protocol	Port range
–	sgr-08a441d5a82f83486	IPv4	Custom TCP	TCP	8080
–	sgr-029ae529219da9b...	IPv4	SSH	TCP	22
–	sgr-066dd8dc3f546a413	IPv4	Custom TCP	TCP	8000

21.We Have Deployed the My Notes Application on Port Number 8000.



My Notes

Notes 5

Note #5
Does the time not update or something? tests
21/1/2023

Note #4
Looks like everything is working so far This is a test update ok ev...
20/1/2023

Note #3
This a new note now REACT is integrated with Django!
20/1/2023

Note #2
This is a new note XD
20/1/2023

Note #1

CHAPTER 5.
CONCLUSION AND FUTURE WORK

5.1 Summary Findings

In this project, I have successfully automated the deployment of a cloud native application monitoring pipeline to our observability platform. I have used a variety of DevOps tools and technologies to achieve this goal, including GitHub, Docker, Docker Hub, AWS EC2, Jenkins, and our observability platform.

The automated deployment process has a number of benefits, including:

- **Reduced time and effort:** The automated deployment process saves me a lot of time and effort, as I do not need to manually perform any steps.

- **Increased accuracy:** The automated deployment process is more accurate than the previous manual process, as it reduces the risk of human error.

- **Improved scalability:** The automated deployment process can easily be scaled to meet the needs of my project.

- **Increased reliability:** The automated deployment process is more reliable than the previous manual process, as it is less susceptible to failures.

The automated deployment process has made it much easier for me to deploy the cloud native application monitoring pipeline to our observability platform. I am confident that this project will help us to improve the reliability and performance of our cloud native applications.

In addition to the technical benefits listed above, I also learned a number of important lessons from this project. Here are a few of the most important lessons I learned:

- **The importance of Testing:** It is important to thoroughly test the automated deployment process before deploying it to production. This will help to identify and fix any problems.

- **The importance of Documentation:** It is important to document the automated deployment process. This will make it easier to maintain and update the process in the future.

5.2 Summary of Findings

I am excited to continue working on this project to improve the deployment of cloud native applications. Here are some specific areas where I plan to focus my future work:

- Using machine learning to automate the deployment process: I believe that machine learning could be used to automate the deployment process even further. For example, machine learning could be used to predict the optimal deployment configuration for a given cloud native application.

- **Developing a self-healing deployment process:** I am also interested in developing a self-healing deployment process. This would allow the deployment process to automatically recover from failures without any human intervention.

REFERENCES

BOOKS:

1. Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation by Jez Humble and David Farley.

ARTICLES:

1. Automating Cloud Native Application Deployment with Jenkins and Kubernetes
BY-- Kelsey Hightower

2. Monitoring Cloud Native Applications with Prometheus and Grafana
BY--Victoria Homer