ASI Insurance: Modernizing Operations with Microservices and CI/CD

by
Anjali Lalwani

Problem Statement

ASI Insurance is facing challenges in improving the SLA to its customers due to its organizational growth and existing monolithic application architecture. It requires transformation of the existing architecture to a microservice application architecture, while also implementing DevOps pipeline and automations.
The successful completion of the project will enable ASI Insurance to improve its overall application deployment process, enhance system scalability, and deliver better products and services to its customers.

Tasks

1. Create the Dockerfile, Jenkinsfile, Ansible playbook, and the source file of the static website
2. Upload all the created files to GitHub
3. Go to the terminal and install NodeJS 16
4. Open the browser and access the Jenkins application
5. Create Jenkins pipeline to perform CI/CD for a Docker container
6. Create Docker Hub Credentials and other necessary pre-requisites before running build
7. Set up Docker remote host on AWS and configure deploy stage in pipeline
8. Execute Jenkins Build
9. Access deployed application on Docker container

Goal

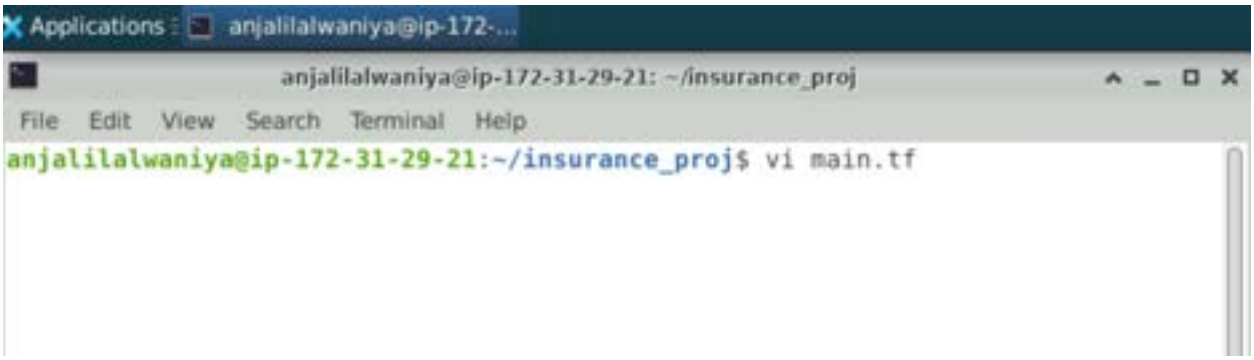
The goal of the project is to facilitate the transformation of ASI Insurance's existing monolithic application architecture into a more scalable and efficient microservices-based architecture. This transformation will be achieved by implementing a comprehensive DevOps pipeline, leveraging automation tools like Jenkins, GitHub, Docker Hub, and AWS. The ultimate aim is to enhance the company's ability to deliver products and services to its customers by improving SLAs, enhancing system scalability, and streamlining the application deployment process.

The subsequent sections in this document highlight the steps required to achieve this objective.

.STEPS:

Step 1: Provision an EC2 Instance using Terraform.

- Create main.tf and copy the following code into it.




```
provider "aws" {
  access_key = "XXXXXXX" ## replace with your access key
  secret_key = "XXXXXXX" ## replace with your secret key
  token = “XXXXXXX" ## replace with your token
  region     = "us-east-1"
}

resource "aws_security_group" "p1_sg" {
  name        = "p1_sg"
  description = "Project1 security group"

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Allow HTTP access from anywhere
  ingress {
    from_port   = 80
```

```
  to_port     = 80
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

# Allow HTTPS access from anywhere
ingress {
  from_port   = 443
  to_port     = 443
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

ingress {
  from_port   = 3000
  to_port     = 3000
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

# Allow SSH access from anywhere
ingress {
  from_port   = 22
  to_port     = 22
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

ingress {
  from_port   = 81
  to_port     = 81
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

 # Allow all outbound traffic
egress {
  from_port   = 0
```

```
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }


}

# Define the key pair for SSH access
resource "aws_key_pair" "key_pair" {
  key_name   = "p1-key-pair"
  public_key = file("~/.ssh/id_rsa.pub")
}

resource "aws_instance" "project1_instance" {
  ami                    = "ami-06aa3f7caf3a30282" # Canonical, Ubuntu, 20.04
LTS, amd64 focal image build on 2023-10-25
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.p1_sg.id]

  tags = {
    Name = "project1"
  }

  }

  output "public_ip" {
    value = aws_instance.project1_instance.public_ip
  }
```

Replace XXXXXXX in the code above with your AWS credentials.

- Execute the Terraform script using the following commands:

  terraform init

```
anjalilalwaniya@ip-172-31-29-21:~/insurance_proj$ vi main.tf
anjalilalwaniya@ip-172-31-29-21:~/insurance_proj$ terraform init

Initializing the backend...
```

terraform plan

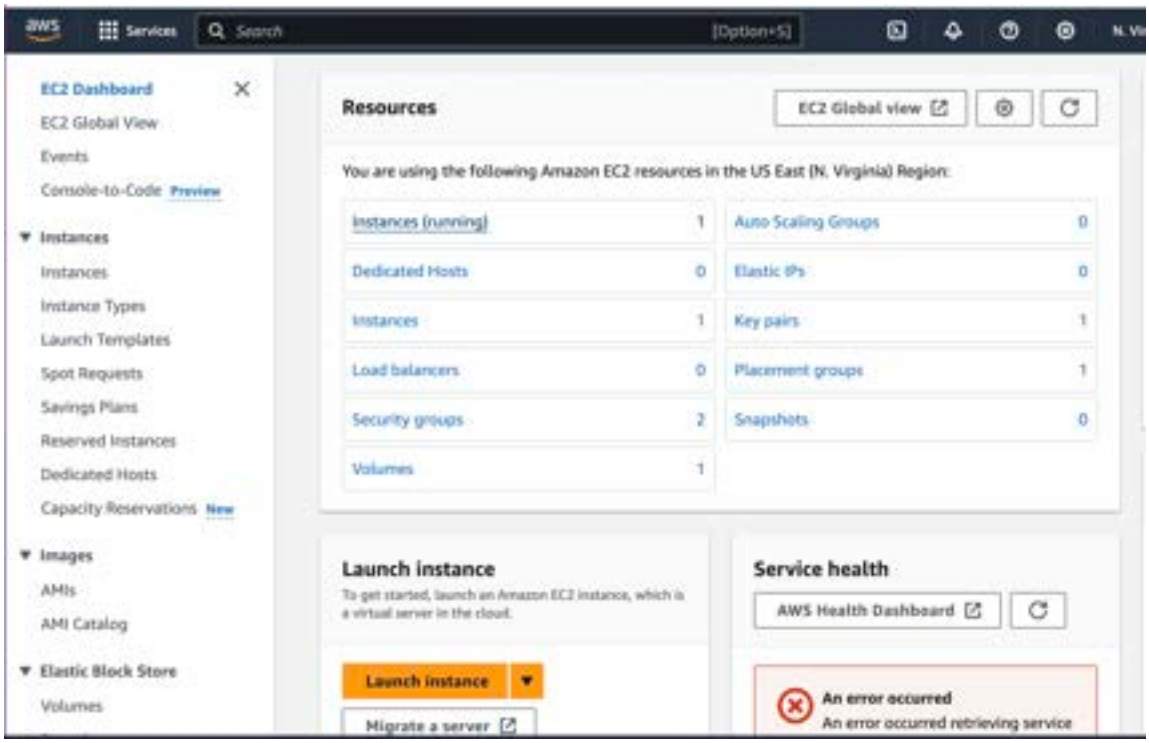
```
anjalilalwaniya@ip-172-31-29-21:~/insurance_proj$ terraform plan
aws_key_pair.key_pair: Refreshing state... [id=p1-key-pair]
aws_security_group.p1_sg: Refreshing state... [id=sg-0635ccec4759cdbd4]
aws_instance.project1_instance: Refreshing state... [id=i-02a82a6421ec03baa]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the
last "terraform apply":

  # aws_instance.project1_instance has been deleted
  - resource "aws_instance" "project1_instance" {
      - ami                                  = "ami-06aa3f7caf3a30282" -> null
      - arn                                  = "arn:aws:ec2:us-east-1:6763725894
07:instance/i-02a82a6421ec03baa" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "us-east-1a" -> null
      - cpu_core_count                       = 1 -> null
      - cpu_threads_per_core                 = 1 -> null
      - disable_api_stop                     = false -> null
      - disable_api_termination              = false -> null
      - ebs_optimized                        = false -> null
      - get_password_data                    = false -> null
      - hibernation                          = false -> null
```

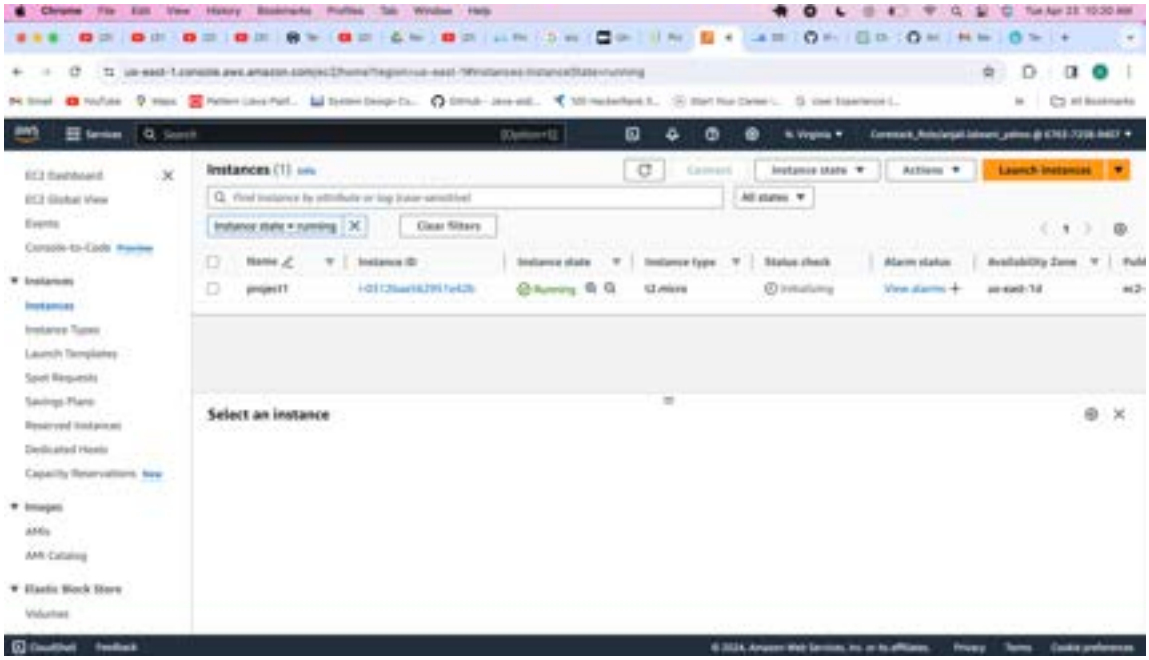
terraform apply

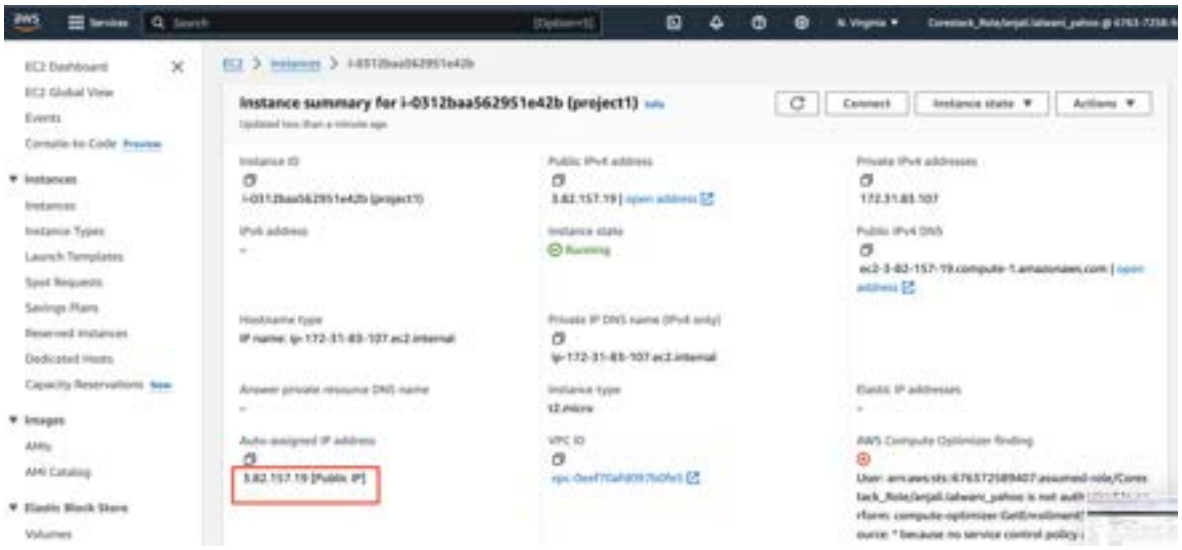
```
Applications : anjalilalwaniya@ip-172-...
anjalilalwaniya@ip-172-31-29-21: ~/insurance_proj
File  Edit  View  Search  Terminal  Help

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_key_pair.key_pair: Creating...
aws_security_group.p1_sg: Creating...
aws_key_pair.key_pair: Creation complete after 0s [id=p1-key-pair]
aws_security_group.p1_sg: Creation complete after 2s [id=sg-0cb4f989ffae9c476]
aws_instance.project1_instance: Creating...
aws_instance.project1_instance: Still creating... [10s elapsed]
aws_instance.project1_instance: Still creating... [20s elapsed]
aws_instance.project1_instance: Still creating... [30s elapsed]
aws_instance.project1_instance: Creation complete after 32s [id=i-0312baa562951e
42b]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

public_ip = "3.82.157.19"
anjalilalwaniya@ip-172-31-29-21:~/insurance_proj$
```

Note the public ip of the ec2 instance created. This will be required in some of the steps below.

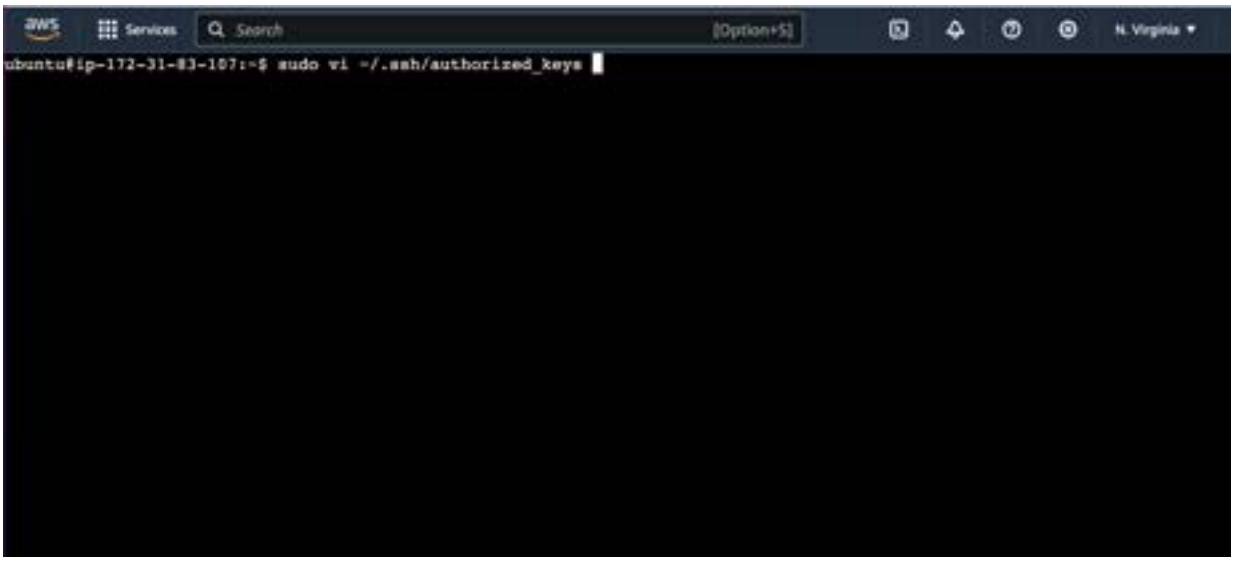
- Login to the AWS Web console. You will see that an ec2 instance with the above ip is now created.


Services
Search
[Option+S]
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Resources
EC2 Global view
You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:
Instances (running) 1
Dedicated Hosts 0
Instances 1
Load balancers 0
Security groups 2
Volumes 1
Auto Scaling Groups 0
Elastic IPs 0
Key pairs 1
Placement groups 1
Snapshots 0
Launch instance
To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.
Launch instance
Migrate a server
Service health
AWS Health Dashboard
An error occurred
An error occurred retrieving service


Instances (1)
Instance state
Actions
Launch instances
All states
Clear filters
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Select an instance



- Now from the AWS dashboard, go to your EC2 instance and click 'Connect'.Once connected, copy your public key from `~/.ssh/id_rsa` and add it to the `authorized_keys` file on the EC2 instance.



Step 2: Install Ansible

- Use the following commands to install ansible on your machine.

  sudo apt-get update
  sudo apt-get install -y ansible

```
Applications : anjalilalwaniya@ip-172-...
anjalilalwaniya@ip-172-31-29-21: ~/insurance_proj
File  Edit  View  Search  Terminal  Help
anjalilalwaniya@ip-172-31-29-21:~/insurance_proj$ sudo apt-get update
```

```
anjalilalwaniya@ip-172-31-29-21: ~/insurance_proj
File  Edit  View  Search  Terminal  Help
Get:16 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages
[956 kB]
Fetched 8520 kB in 1s (5708 kB/s)
Reading package lists... Done
W: An error occurred during the signature verification. The repository is not up
dated and the previous index files will be used. GPG error: https://dl.google.co
m/linux/chrome/deb stable InRelease: The following signatures couldn't be verifi
ed because the public key is not available: NO_PUBKEY E88979FB9B30ACF2
W: Failed to fetch https://dl.google.com/linux/chrome/deb/dists/stable/InRelease
  The following signatures couldn't be verified because the public key is not av
ailable: NO_PUBKEY E88979FB9B30ACF2
W: Some index files failed to download. They have been ignored, or old ones used
 instead.
anjalilalwaniya@ip-172-31-29-21:~/insurance_proj$ sudo apt-get install -y ansibl
e
Reading package lists... Done
Building dependency tree
Reading state information... Done
ansible is already the newest version (5.10.0-1ppa~focal).
The following package was automatically installed and is no longer required:
  daemon
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 295 not upgraded.
anjalilalwaniya@ip-172-31-29-21:~/insurance_proj$
```

Step 3: Setup Jenkins

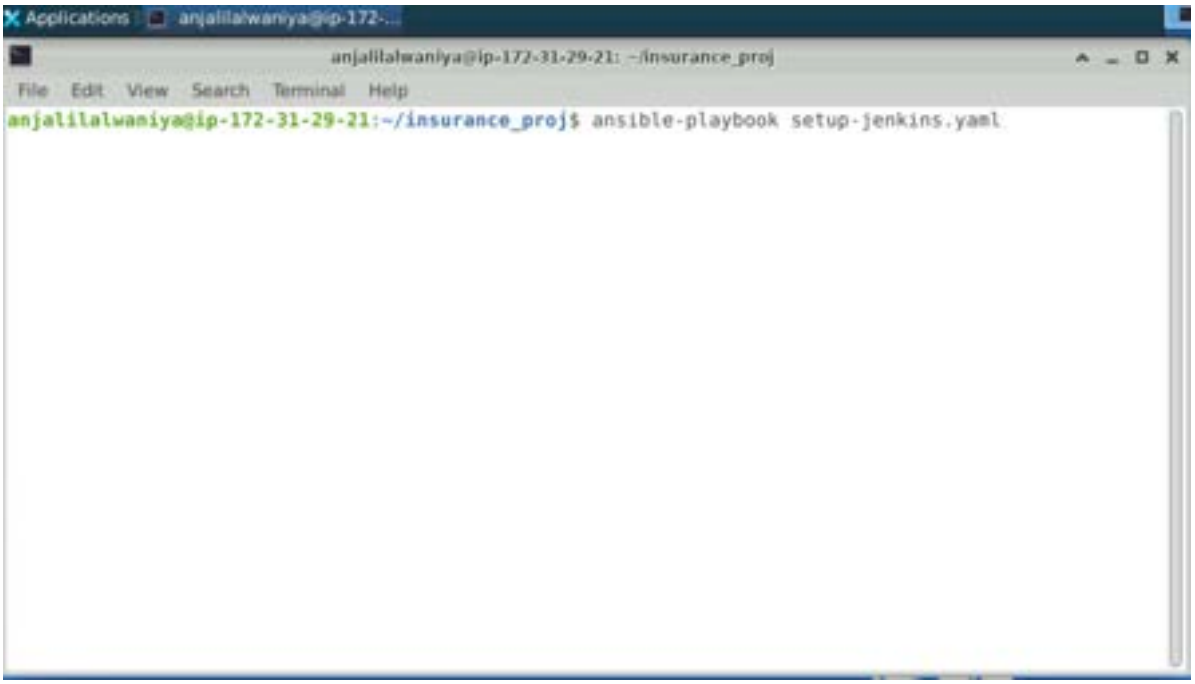
- Create setup-jenkins.yaml and copy the following into it.

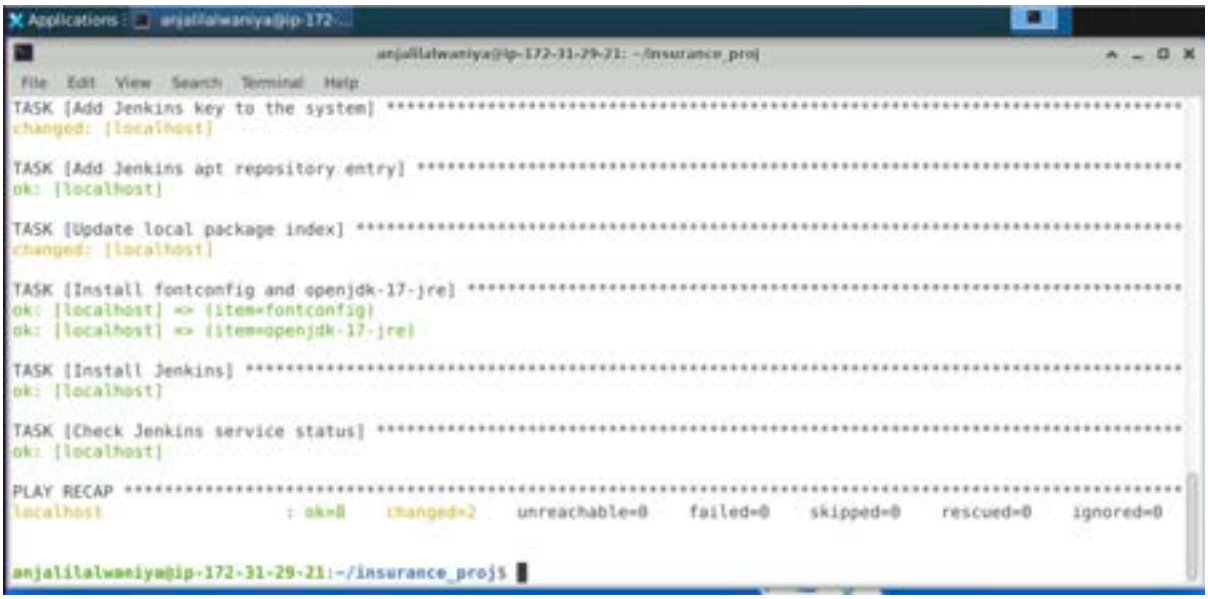
```
---
- name: Install Jenkins and OpenJDK
  hosts: localhost
  become: yes  # To run tasks with sudo

  tasks:
    - name: Install default-jre
      apt:
        name: default-jre
        state: present

    - name: Add Jenkins key to the system
      shell: |
        sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
          https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

    - name: Add Jenkins apt repository entry
      lineinfile:
        path: /etc/apt/sources.list.d/jenkins.list
        line: 'deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://
pkg.jenkins.io/debian-stable binary/'
        create: yes

    - name: Update local package index
      apt:
        update_cache: yes

    - name: Install fontconfig and openjdk-17-jre
      apt:
        name: "{{ item }}"
        state: present
      loop:
        - fontconfig
        - openjdk-17-jre
```

```
- name: Install Jenkins
  apt:
    name: jenkins
    state: present

- name: Check Jenkins service status
  service:
    name: jenkins
    state: started
    enabled: yes
```

- Run the following command to install Jenkins via ansible.

ansible-playbook setup-jenkins.yaml





- In your browser, open the following link -

  http://localhost:8080

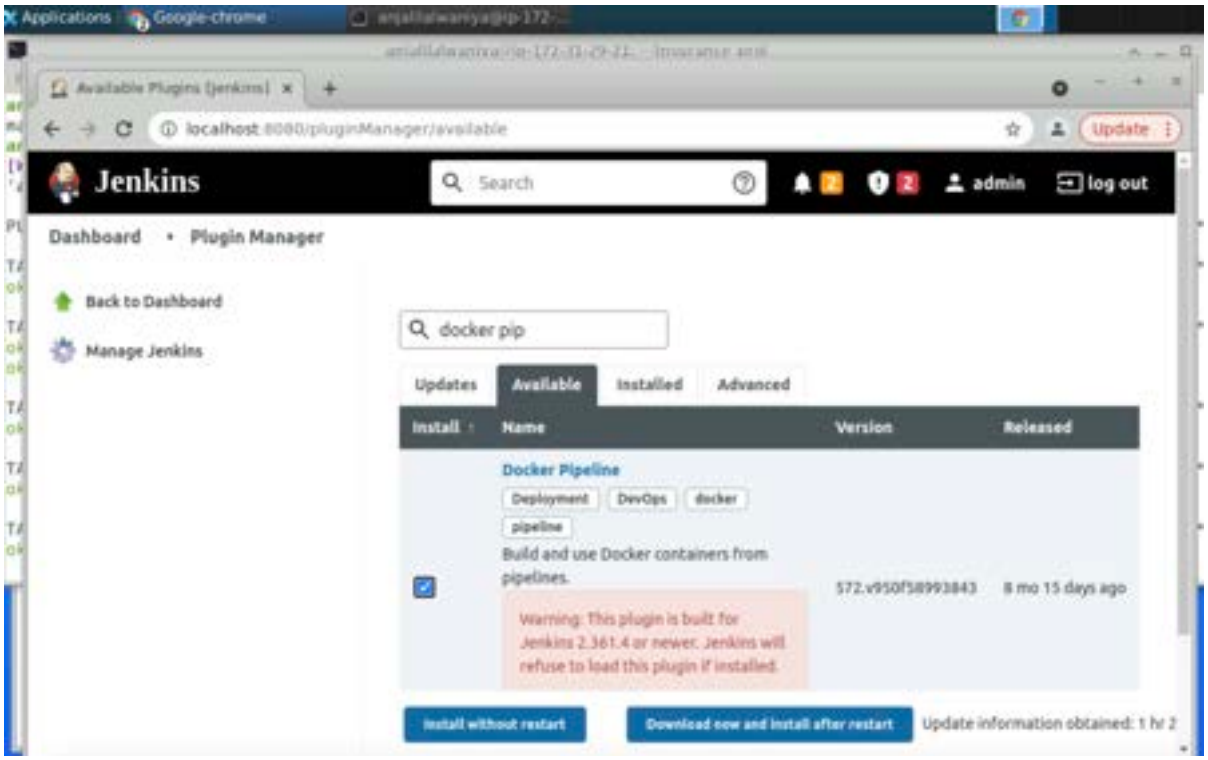
After you install Jenkins and login to the Jenkins for the first time , you will get an unlock Jenkins screen. To unlock this screen, you need an initial admin password.

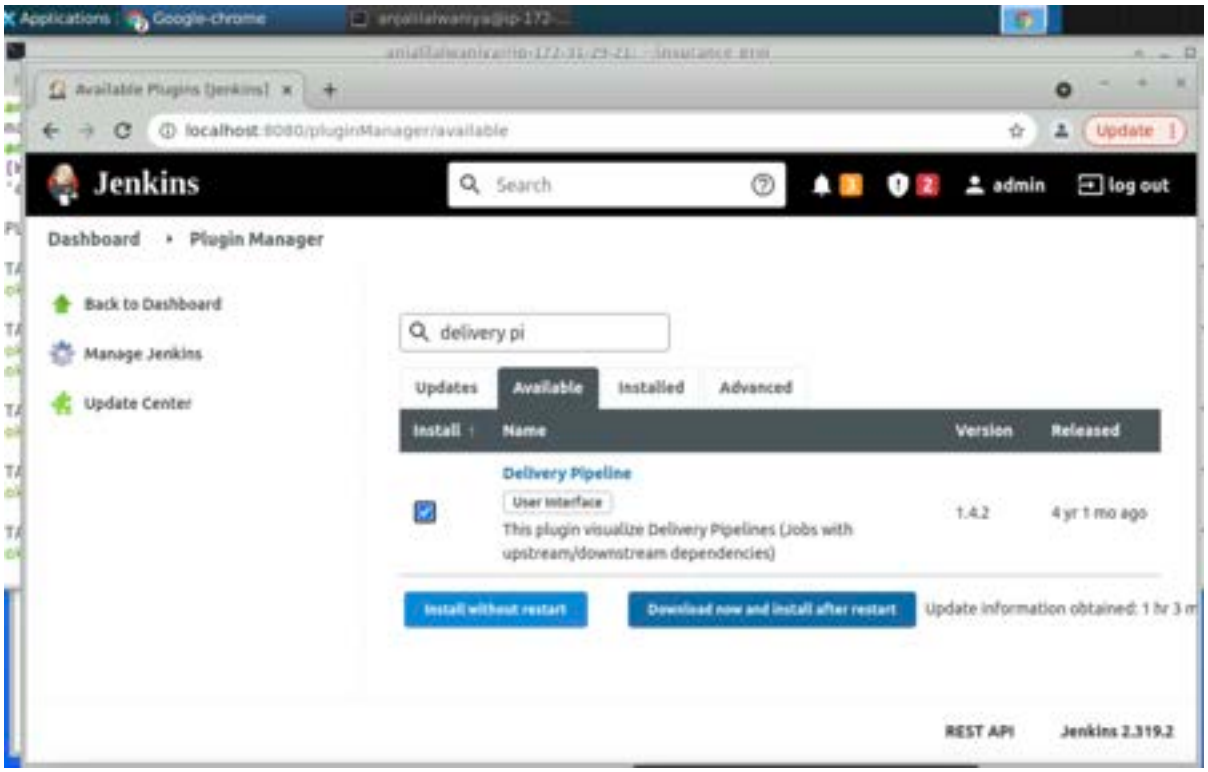
$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

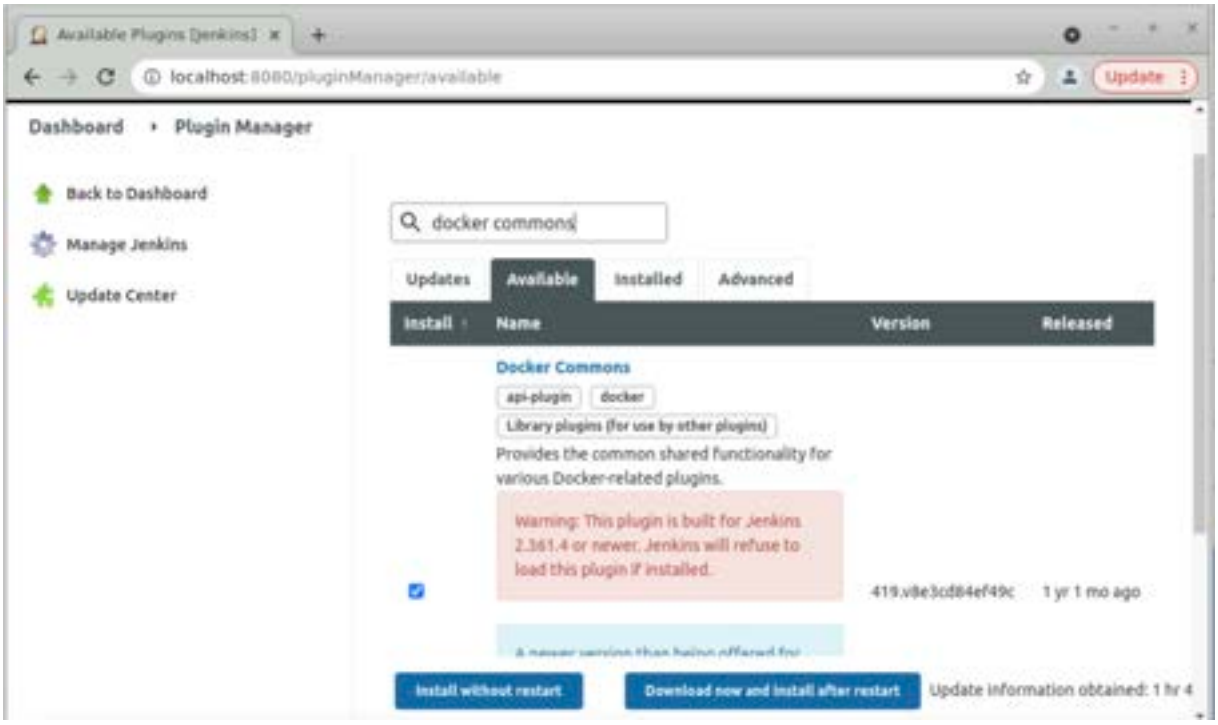
Create your first admin user and login.

- From the Jenkins dashboard, go to Manage Jenkins->Plugins Manager and install the following plugins:

  Docker Pipeline plugin
  Delivery Pipeline plugin
  Docker Commons plugin
  SSH plugin.

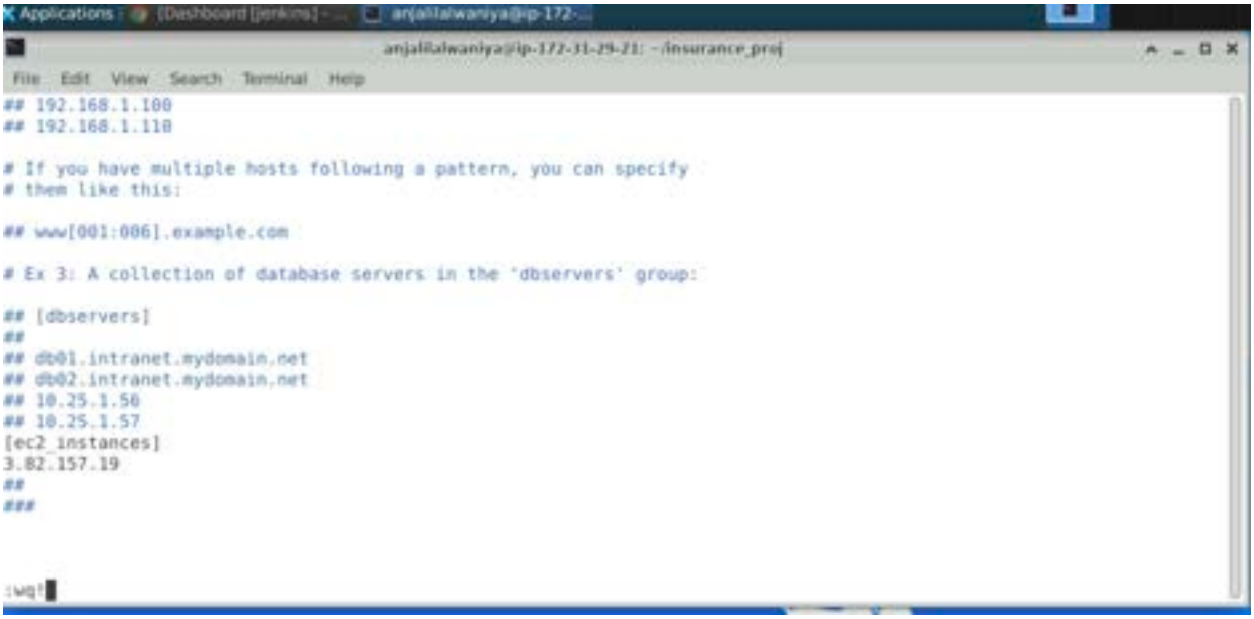

Applications
Google-chrome
Available Plugins [Jenkins]
localhost:8080/pluginManager/available
Update
Jenkins
Search
admin
log out
Dashboard
Plugin Manager
Back to Dashboard
Manage Jenkins
docker pip
Updates
Available
Installed
Advanced
Install
Name
Version
Released
Docker Pipeline
Deployment
DevOps
docker
pipeline
Build and use Docker containers from pipelines.
572.v950f58993843
8 mo 15 days ago
Warning: This plugin is built for Jenkins 2.361.4 or newer. Jenkins will refuse to load this plugin if installed.
Install without restart
Download now and install after restart
Update information obtained: 1 hr 2


Applications
Google-chrome
Available Plugins [Jenkins]
localhost:8080/pluginManager/available
Update
Jenkins
Search
admin
log out
Dashboard
Plugin Manager
Back to Dashboard
Manage Jenkins
Update Center
delivery pi
Updates
Available
Installed
Advanced
Install
Name
Version
Released
Delivery Pipeline
User Interface
This plugin visualize Delivery Pipelines (Jobs with upstream/downstream dependencies)
1.4.2
4 yr 1 mo ago
Install without restart
Download now and install after restart
Update information obtained: 1 hr 3 m
REST API
Jenkins 2.319.2



Step 4: Install Docker on EC2 Using Ansible Playbook

- Add the public ip of the EC2 instance noted in step 1 above to the /etc/ansible/hosts file and save it.



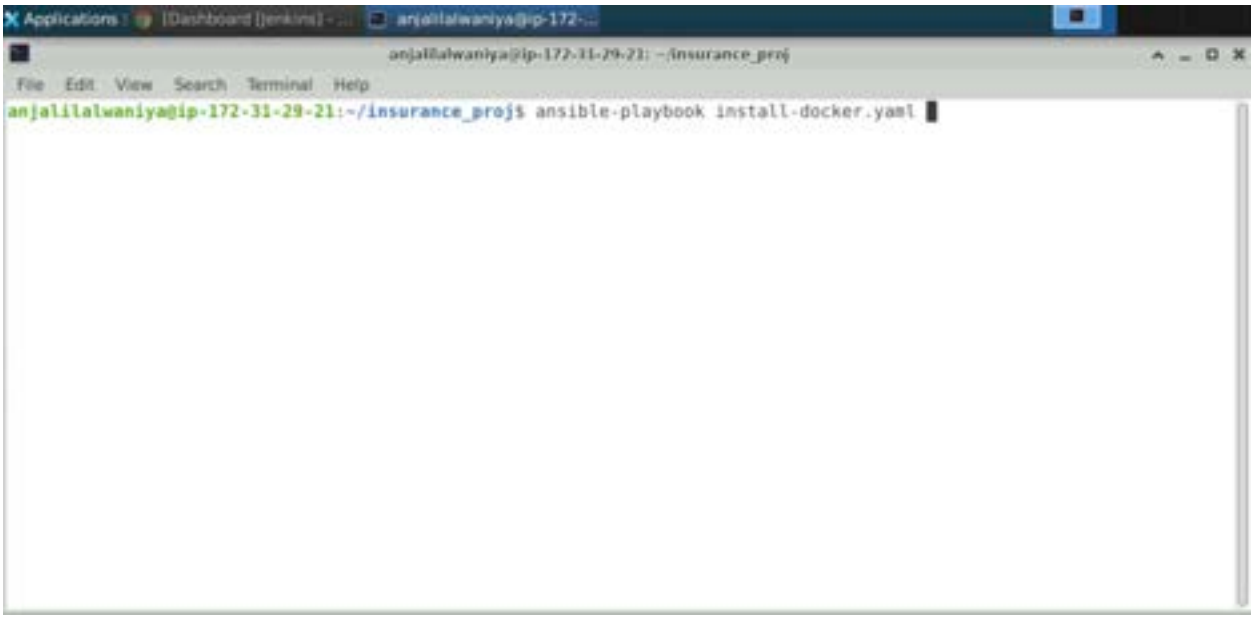
- Create install-docker.yaml and save the following code into it.

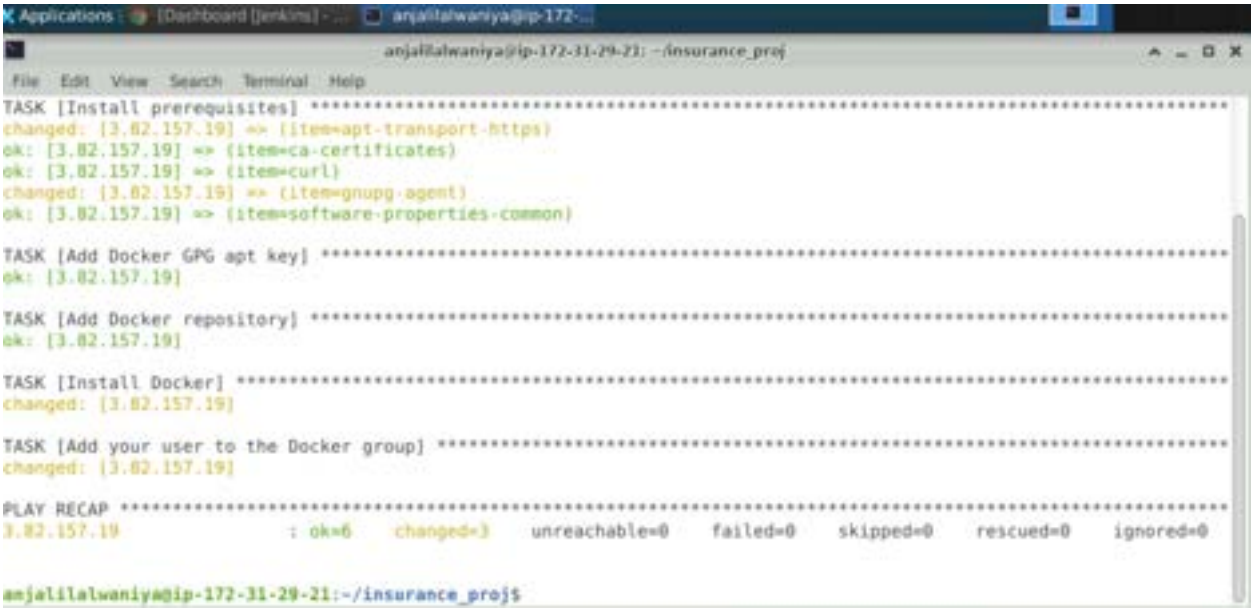
```
---
- name: Install Docker on EC2 instance
  hosts: ec2_instances
  remote_user: ubuntu
  become: true

  tasks:
    - name: Install prerequisites
      apt:
        name: "{{ item }}"
        state: present
      with_items:
        - apt-transport-https
        - ca-certificates
        - curl
        - gnupg-agent
        - software-properties-common

    - name: Add Docker GPG apt key
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present

    - name: Add Docker repository
      apt_repository:
        repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu
focal stable
        state: present

    - name: Install Docker
      apt:
        name: docker-ce
        state: present

    - name: Add your user to the Docker group
      user:
```

```
name: "{{ ansible_user }}"
groups: docker
append: yes
```

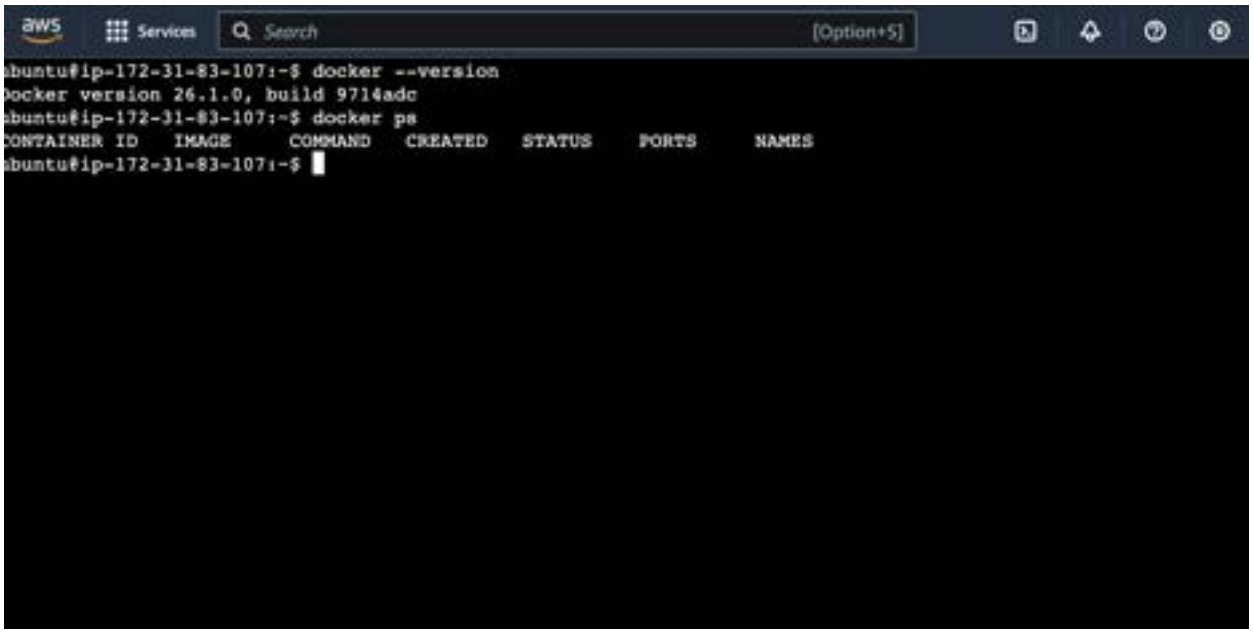
- Run the ansible playbook using the following command:

ansible-playbook install-docker.yaml



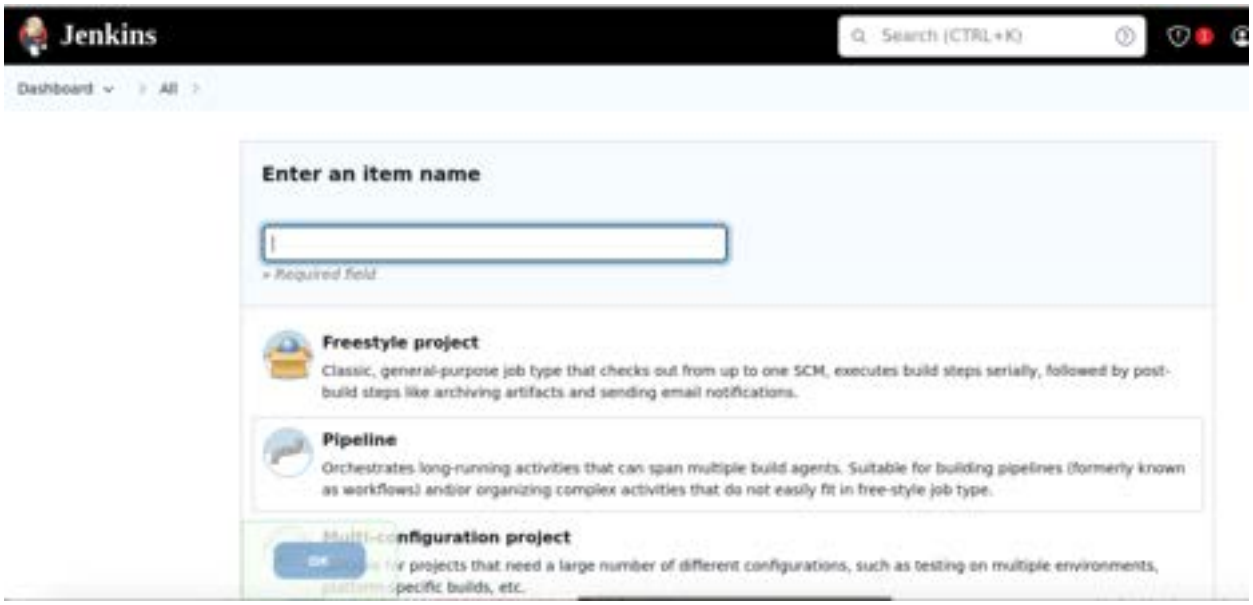


- Verify docker installation by connecting to the EC2 instance using the AWS web console and running the commands below:

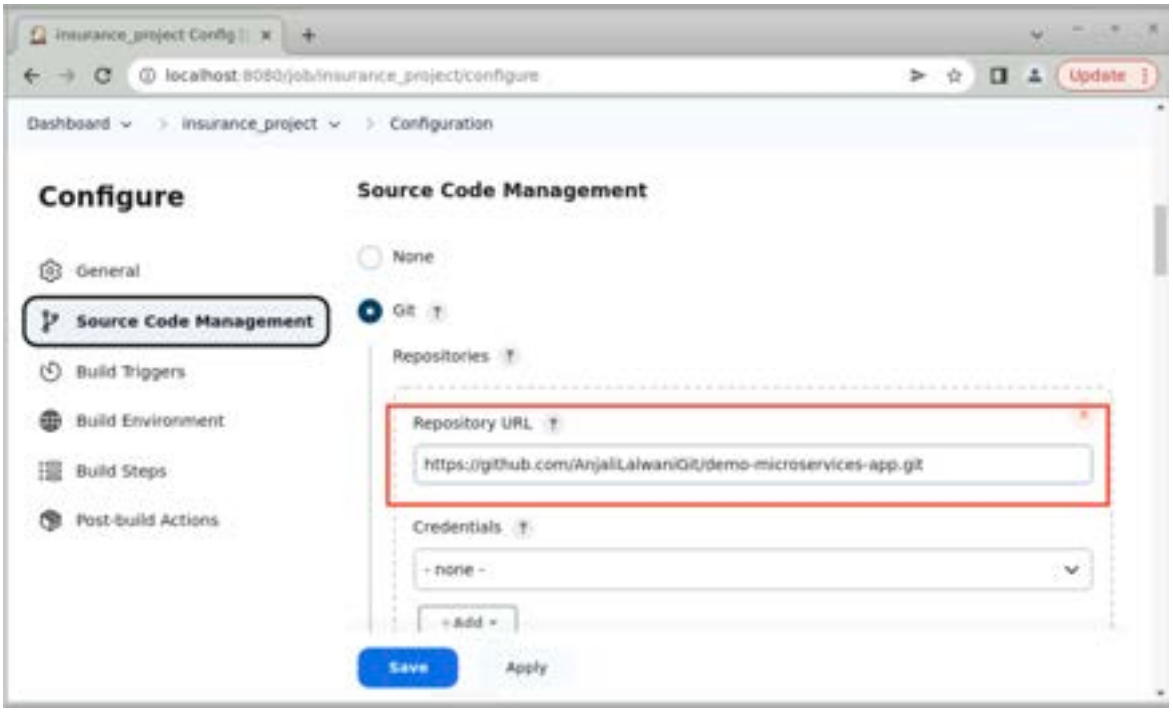


Step 5: Create Jenkins Pipeline to Deploy the Application on EC2 Using Dockerfile.

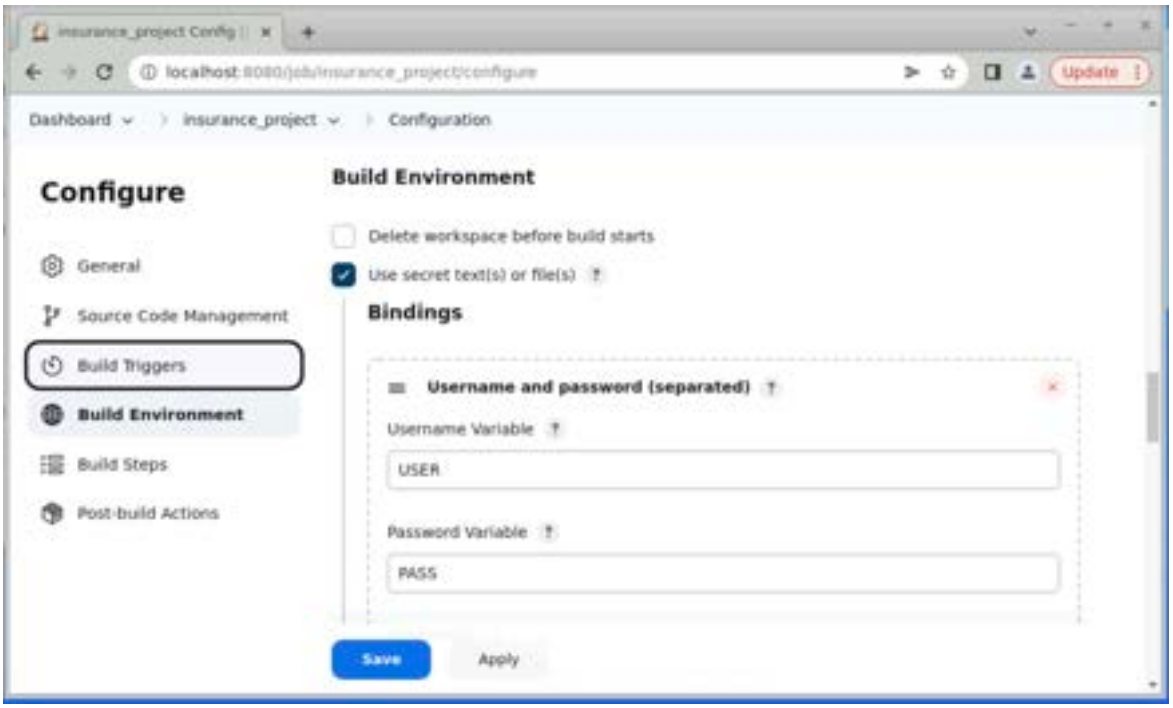
- Open your browser, and login into your Jenkins account.Create a freestyle project.



- Under the Source Code Management section, select Git and enter the Github url to your micro-services application.

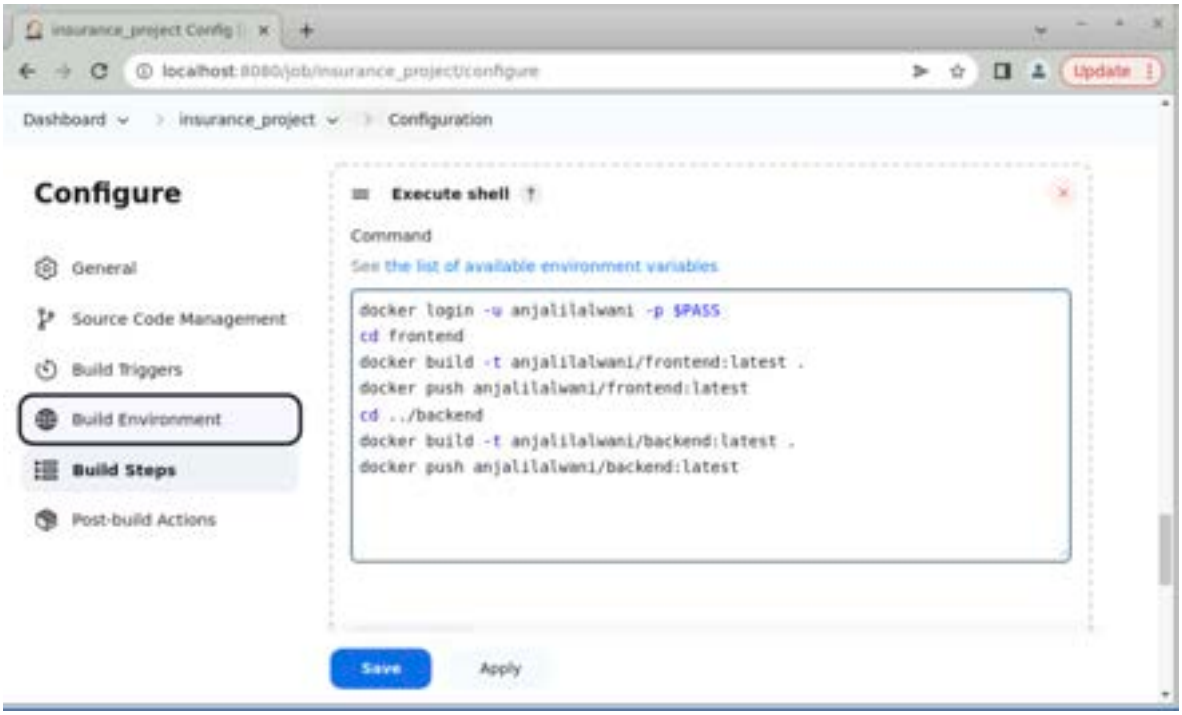


- Go to the Build Environment section, select the ‘use secret texts or files’ checkbox. Create a binding vairable for your username and password for logging into your GitHub Account.



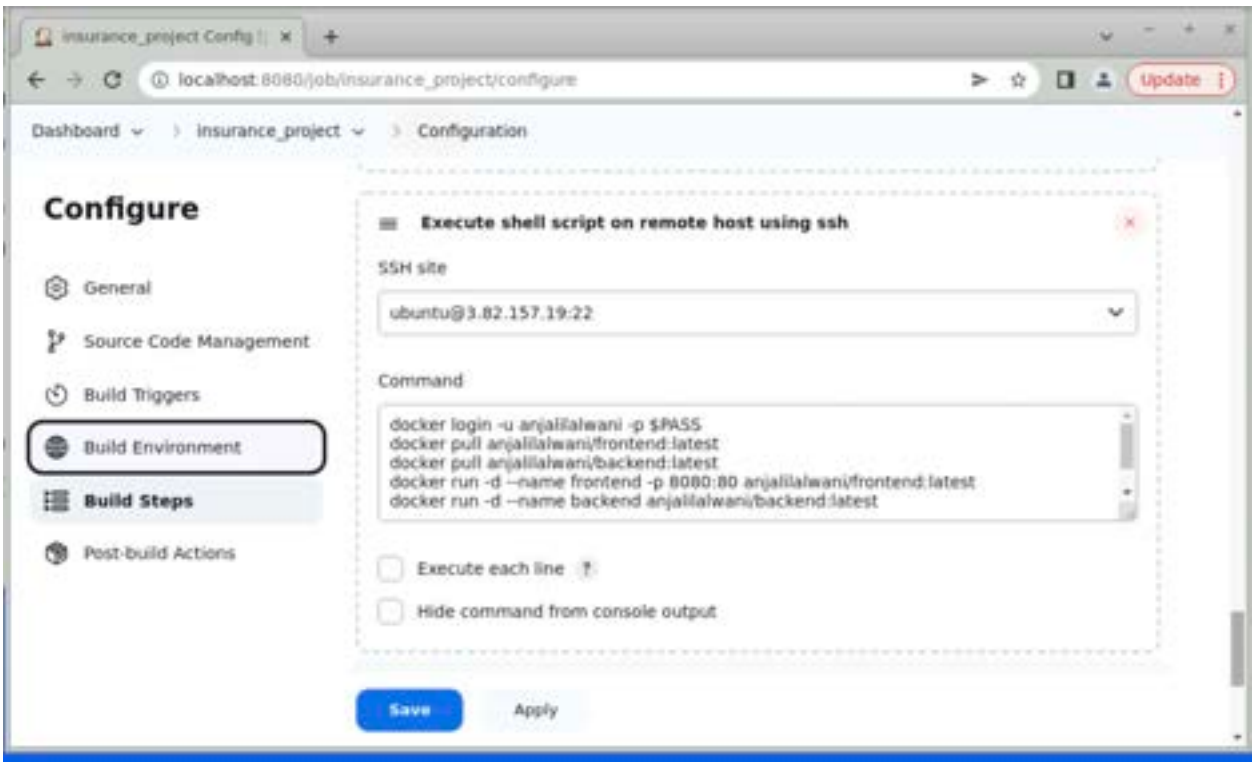
- Now add a build step to your pipeline and select the ‘Execute shell’ option. Add the following shell script -

```
docker login -u anjalilalwani -p $PASS
cd frontend
docker build -t anjalilalwani/frontend:latest .
docker push anjalilalwani/frontend:latest
cd ../backend
docker build -t anjalilalwani/backend:latest .
docker push anjalilalwani/backend:latest
```



- Now another build step to your pipeline and select the ‘Execute shell’ option. Add the following shell script -

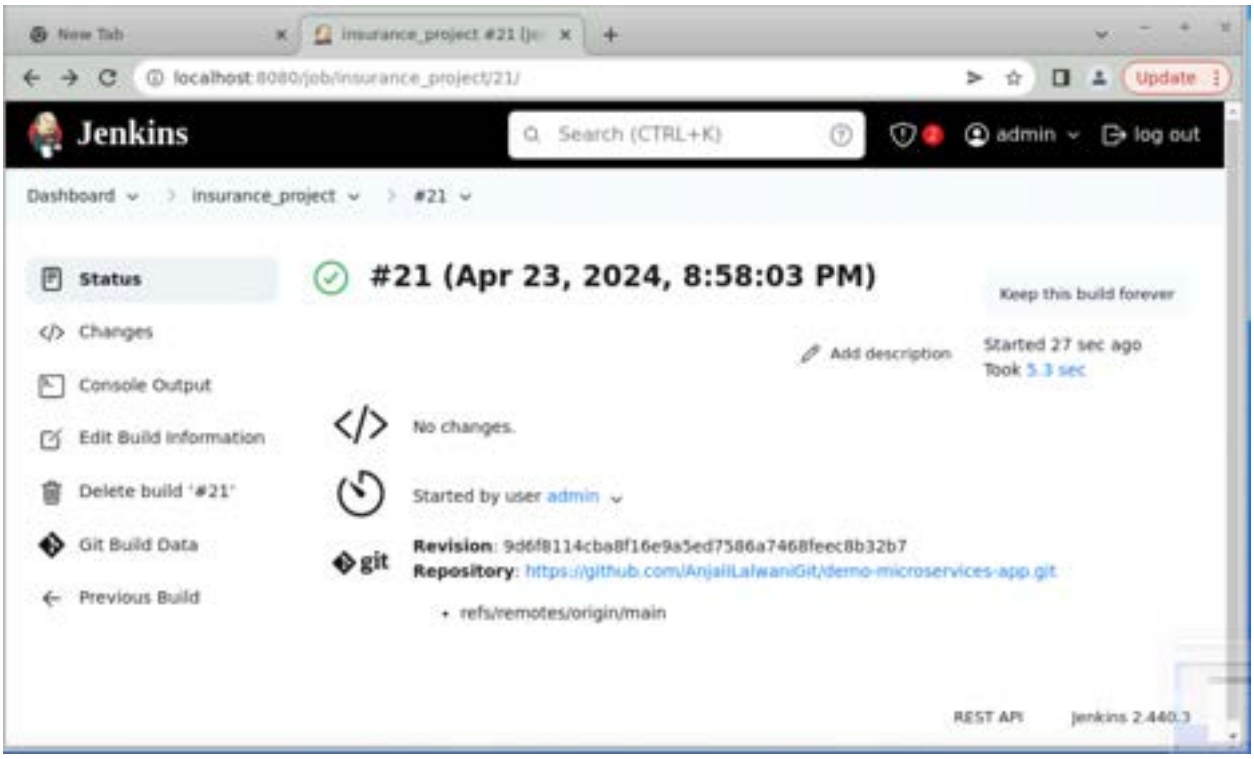
docker login -u anjalilalwani -p $PASS
docker pull anjalilalwani/frontend:latest
docker pull anjalilalwani/backend:latest
docker run -d --name frontend -p 8080:80 anjalilalwani/frontend:latest
docker run -d --name backend anjalilalwani/backend:latest

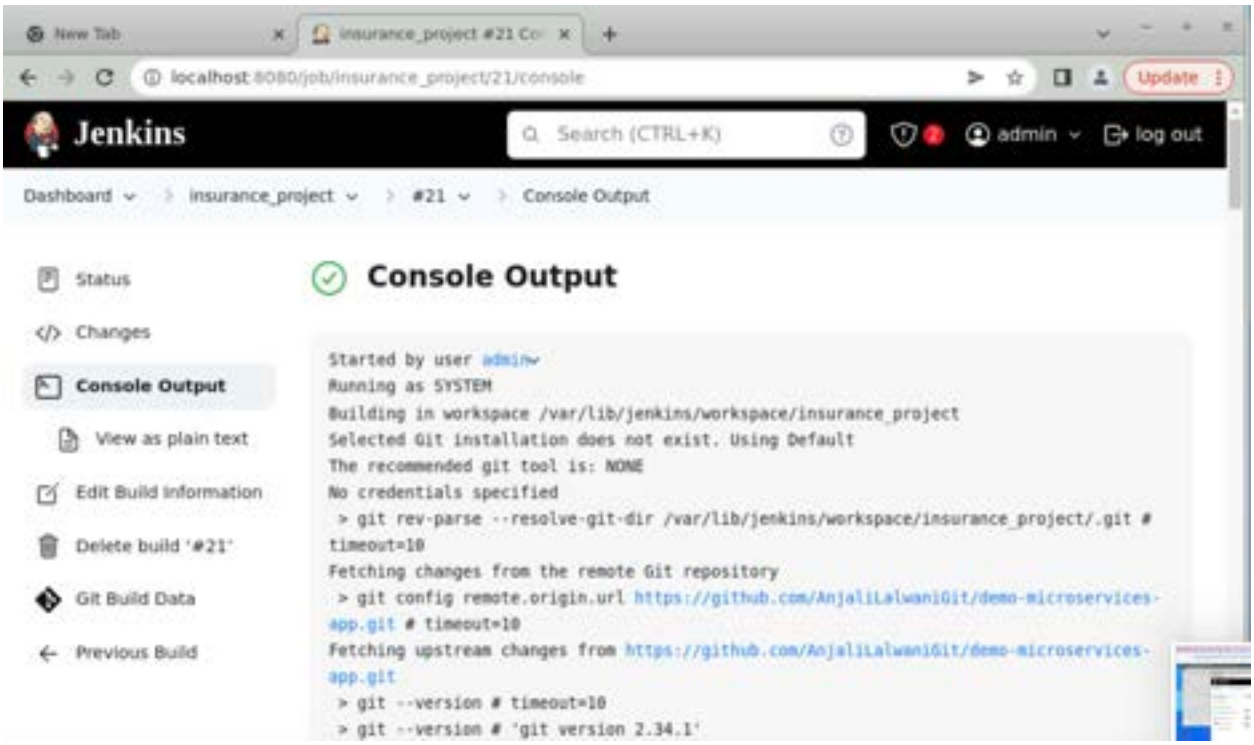


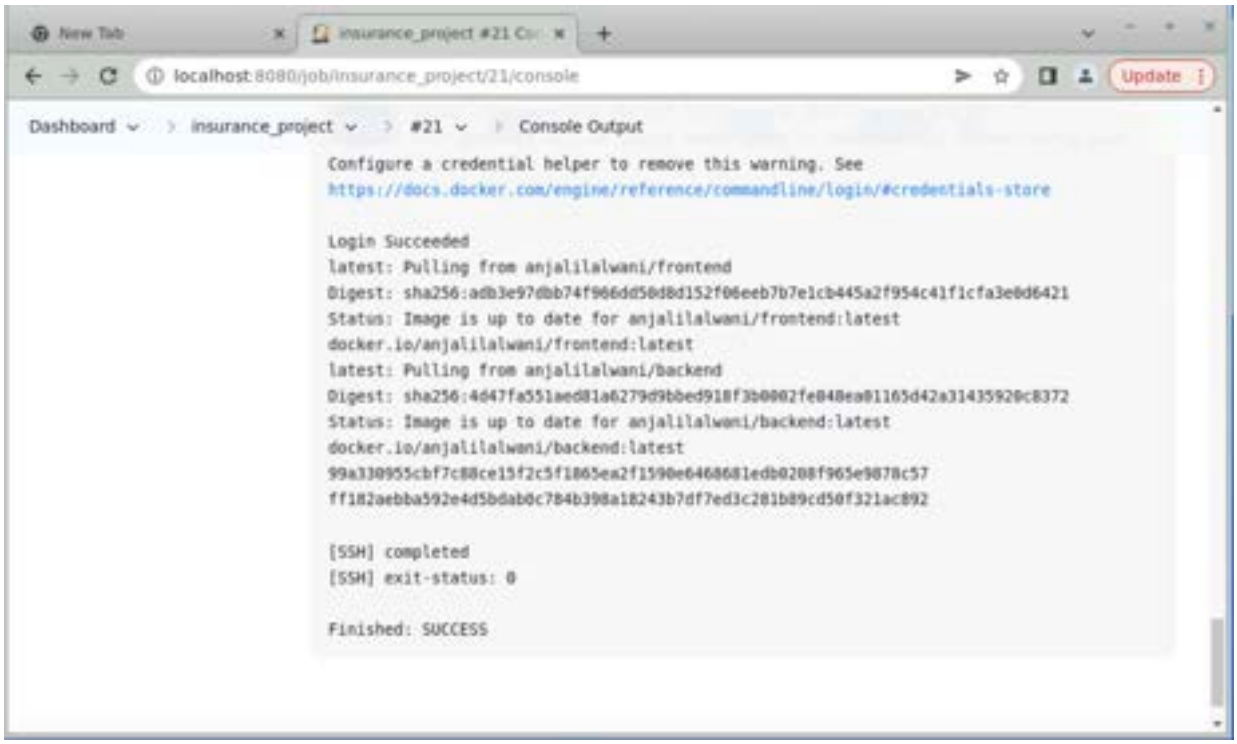
- Save the configuration changes made.

Step 6: Build and test the Jenkins pipeline.

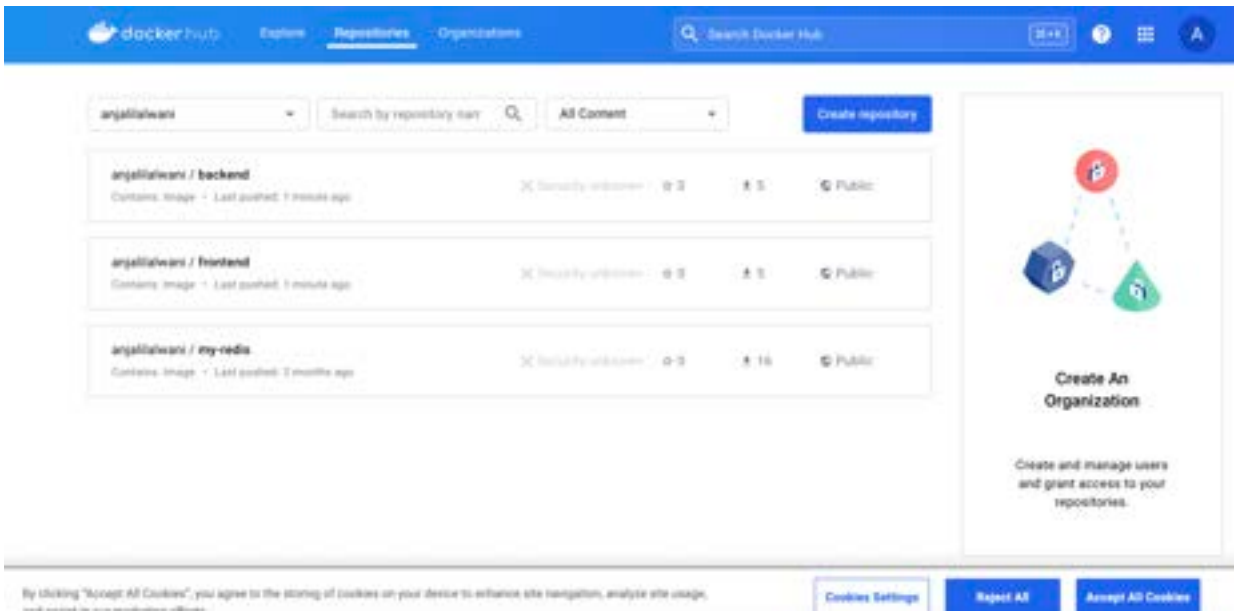
- From Jenkins Dashboard, go to your project and select ‘Build Now’.



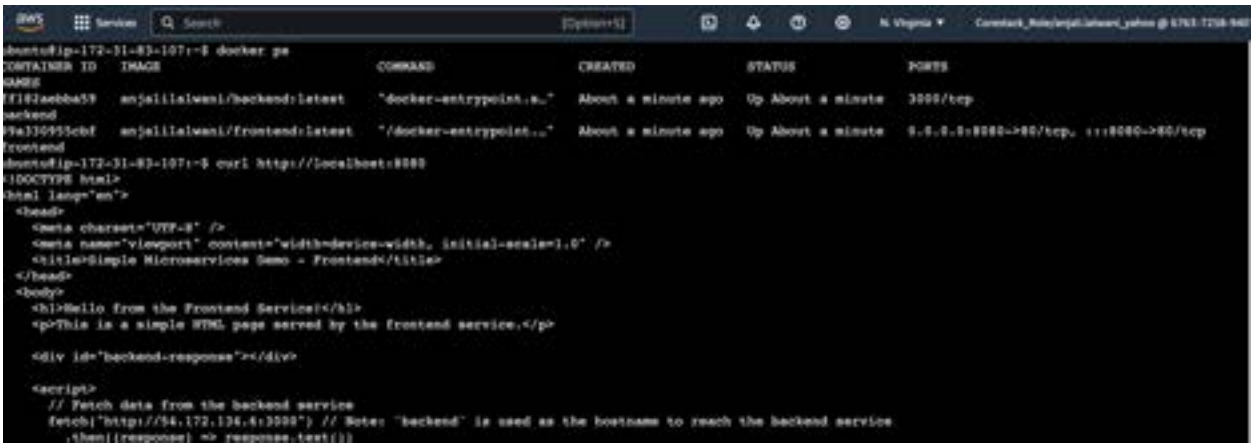


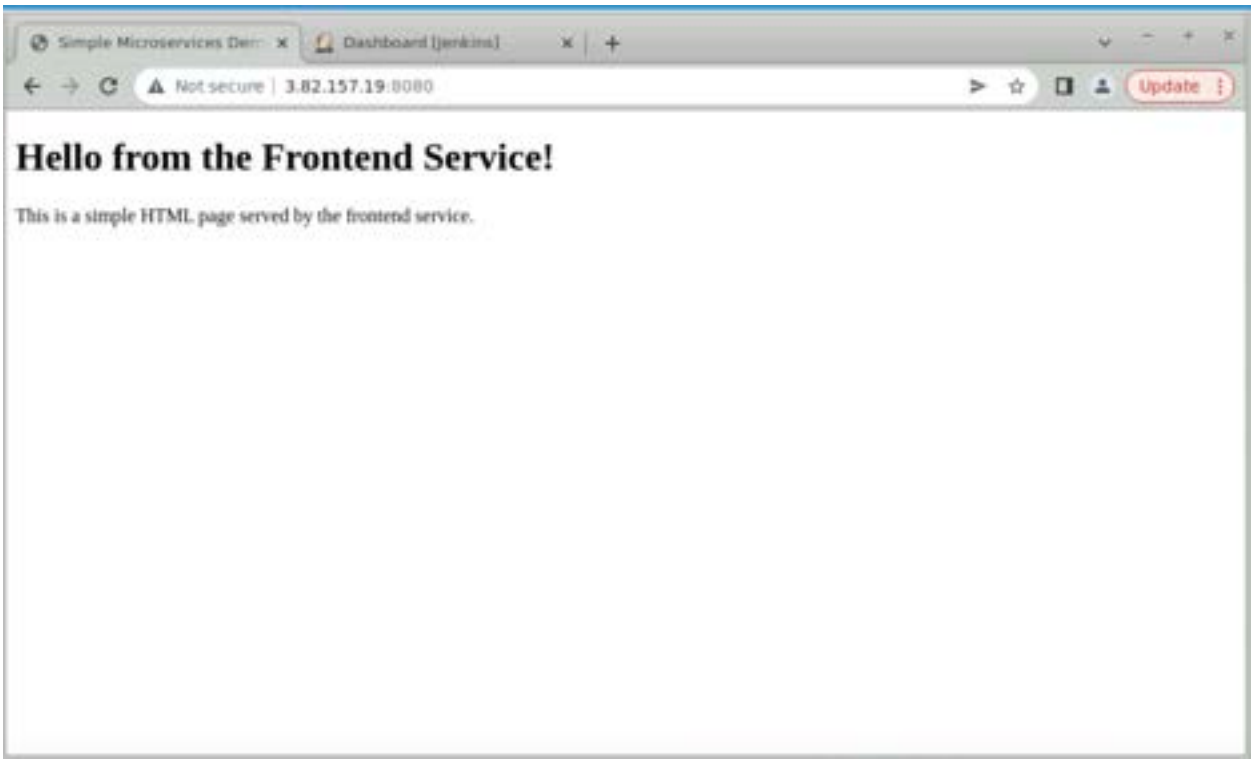


- Login into your Docker Hub account and verfiy that the images are successfully pushed.



- Now verify that your application is running successfully by connecting to the frontend application from your EC2 instance or using the public ip of the EC2 instance.





CONCLUSION:

In conclusion, the ASi Insurance project successfully leveraged infrastructure as code (IaC) tools like Terraform and Ansible to automate the provisioning of AWS resources and the installation of necessary software components such as Jenkins and Docker. By implementing these automation practices, the project achieved improved efficiency, scalability, and consistency in managing its infrastructure and deployment processes, laying a robust foundation for future development and operations.