



Task : 1

👉 Launch Public_EC2 in Public Subnet

1. Go to EC2 → Launch Instance
2. Choose:
 - ❖ AMI: Ubuntu
 - ❖ Type: t2.small (Free tier)
 - ❖ Network: Your VPC
 - ❖ Subnet: Public subnet
 - ❖ Auto-assign Public IP: Enable
3. Attach:
 - ❖ Key Pair: Choose the one you created
 - ❖ Security Group: The SSH security group created above



Launch the instance

The screenshot shows the AWS EC2 'Launch an instance' wizard. The first step, 'Name and tags', has 'Public_EC2' entered in the 'Name' field. The second step, 'Application and OS Images (Amazon Machine Image)', displays a grid of quick-start AMIs: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. Below this, a detailed view of the 'Amazon Linux 2023 kernel-6.1 AMI' is shown, including its AMI ID (ami-0150ccaf51ab55a51), Publish Date (2025-07-08), and a 'Verified provider' badge. The third step, 'Instance type', shows the selected 't2.micro' instance type, which is free tier eligible.

The screenshot shows the AWS EC2 Instances page with the instance details for i-0e1d0f886a08e4c0c. Key information displayed includes:

- Public IPv4 address:** 3.85.226.93
- Private IP address (IPv4 only):** 172.31.18.150
- Instance state:** Running
- VPC ID:** vpc-078bd6f00fcfc44
- Subnet ID:** subnet-06f7c678f5346c00e
- Instance ARN:** arn:aws:ec2:us-east-1:980104576357:instance/i-0e1d0f886a08e4c0c

👉 Steps to create s3 bucket and use it as terraform Backend

❖ Create an S3 Bucket for Terraform Backend:

1) Create S3 Bucket :

- ❖ Go to S3 Console → <https://s3.console.aws.amazon.com/s3/>
- ❖ Click "Create bucket"
- ❖ Enter:
 - Bucket name: **sagar-project-terraform-state-bucket**
 - Region: **us-east-1**
- ❖ Keep Block all public access checked
- ❖ (Optional) Enable Versioning
- ❖ Click "Create bucket"

2) Create Folder (Prefix)

- ❖ Inside the bucket, click "Create folder"
- ❖ Name it: **project/** (used as key prefix in backend)

3) Add Terraform Backend Configuration In your Terraform project:

- ❖ **create: backend.tf**

```
terraform {
  backend "s3" {
    bucket = "sagar-project-terraform-state-bucket"
    key    = "project/state.tfstate"
    region = "us-east-1"
  }
}
```

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with navigation links like 'Amazon S3', 'General purpose buckets', 'Directory buckets', 'Table buckets', 'Vector buckets', 'Access Grants', 'Access Points for general purpose buckets', 'Access Points for directory buckets', 'Object Lambda Access Points', 'Multi-Region Access Points', 'Batch Operations', and 'IAM Access Analyzer for S3'. The main area is titled 'General purpose buckets' and shows a table with one item: 'sagar-project-terraform-state-bucket'. The table includes columns for 'Name', 'AWS Region', and 'Creation date'. A 'Create bucket' button is visible at the top right of the table. To the right of the table, there are two boxes: 'Account snapshot' and 'External access summary - new'.

👉 Step-by-Step Terraform Installation on Ubuntu/Debian

- `wget -O https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg`
- `echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list`
- `sudo apt update && sudo apt install terraform`

👉 <https://www.cherryservers.com/blog/install-terraform-ubuntu>

```
ubuntu@ip-172-31-18-150:~$ wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
--2025-07-16 16:29:23-- https://apt.releases.hashicorp.com/gpg
Resolving apt.releases.hashicorp.com (apt.releases.hashicorp.com)... 18.160.10.45, 18.160.10.71, 18.160.10.69,
...
Connecting to apt.releases.hashicorp.com (apt.releases.hashicorp.com)|18.160.10.45|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3980 (3.9K) [binary/octet-stream]
Saving to: 'STDOUT'

[Progress Bar] 100%[=====] 3.89K --.
-KB/s   in 0s

2025-07-16 16:29:23 (1.07 GB/s) - written to stdout [3980/3980]

ubuntu@ip-172-31-18-150:~$ echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com noble main
ubuntu@ip-172-31-18-150:~$ sudo apt update && sudo apt install terraform
```

👉 Create IAM Role with Admin

- 1) Go to IAM > Roles > Create role.
- 2) Select Trusted Entity Type → "**AWS service**".
- 3) Choose Use case → **EC2**.
- 4) Click Next.
- 5) Attach policy:
 - ❖ For full admin: **AdministratorAccess**
- 6) Name the role : **terraformadmin**
- 7) Click Create Role.

IAM > Roles > Create role

Select trusted entity

Step 1 Select trusted entity **Step 2** Add permissions **Step 3** Name, review, and create

Trusted entity type

- AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity Allow users federated by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy Create a custom trust policy to enable others to perform actions in this account.

Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

EC2

Choose a use case for the specified service.

Use case

- EC2 Allows EC2 instances to call AWS services on your behalf.
- EC2 Role for AWS Systems Manager Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.
- EC2 Spot Fleet Role Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
- EC2 - Spot Fleet Auto Scaling Allows Auto Scaling to access and update EC2 spot fleets on your behalf.
- EC2 - Spot Fleet Tagging Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.
- EC2 - Spot Instances Allows EC2 Spot Instances to launch and manage spot instances on your behalf.

👉 Create Infrastructure using terraform

github.com/sagardpatil0055/upgrad-project/tree/main/terraform

sagardpatil0055 / upgrad-project

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Files

main

terraform

- alb-listeners.tf
- alb-targets.tf
- alb.tf
- backend.tf
- index.html
- instances.tf
- internet-gateway.tf
- myip.tf
- nat-gateway.tf
- provider.tf
- route-tables.tf
- security-groups.tf
- subnets.tf
- vpc.tf

upgrad-project / terraform / sagardpatil0055 Update myip.tf

Name	Last commit message	Last commit date
..		
alb-listeners.tf	Create alb-listeners.tf	1 hour ago
alb-targets.tf	Create alb-targets.tf	1 hour ago
alb.tf	Create alb.tf	1 hour ago
backend.tf	Add files via upload	1 hour ago
index.html	Create index.html	1 hour ago
instances.tf	Update instances.tf	1 hour ago
internet-gateway.tf	Add files via upload	1 hour ago
myip.tf	Update myip.tf	now
nat-gateway.tf	Add files via upload	1 hour ago
provider.tf	Add files via upload	Activate Windows Go to Settings to activate Win10 1 hour ago
route-tables.tf	Add files via upload	Update security-groups.tf 1 minute ago
security-groups.tf		

Type here to search

```
git clone https://github.com/sagardpatil0055/upgrad-project.git
```

```
cd upgrad-project/terraform
```

```
terraform init
terraform plan
terraform apply
```

The screenshot shows the AWS CloudShell interface in the us-east-1 region. The terminal window displays the following command sequence:

```
ubuntu@ip-172-31-18-150:~$ clear
ubuntu@ip-172-31-18-150:~$ git clone https://github.com/sagardpatil0055/upgrad-project.git
Cloning into 'upgrad-project'...
remote: Enumerating objects: 103, done.
remote: Counting objects: 100% (103/103), done.
remote: Compressing objects: 100% (77/77), done.
remote: Total 103 (delta 31), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (103/103), 30.07 KiB | 2.51 MiB/s, done.
Resolving deltas: 100% (31/31), done.
ubuntu@ip-172-31-18-150:~$ cd upgrad-project/terraform/
ubuntu@ip-172-31-18-150:~/upgrad-project/terraform$ terraform init
Initializing the backend...
```

Output from the `terraform init` command:

```
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing provider plugins...
- Finding hashicorp/http versions matching "> 3.0"...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/http v3.5.0...
- Installed hashicorp/http v3.5.0 (signed by HashiCorp)
- Installing hashicorp/aws v6.4.0...
- Installed hashicorp/aws v6.4.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.
```

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

```
ubuntu@ip-172-31-18-150:~/upgrad-project/terraform$
```

The screenshot shows the AWS CloudShell interface in the us-east-1 region. The terminal window displays the following command sequence:

```
ubuntu@ip-172-31-18-150:~/upgrad-project/terraform$ clear
ubuntu@ip-172-31-18-150:~/upgrad-project/terraform$ terraform plan
reread this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Output from the `terraform plan` command:

```
data.http.my_ip: Reading...
data.http.my_ip: Read complete after 0s [id=https://api.ipify.org]
aws_vpc.sagar: Refreshing state... [id=vpc-0eebe2d08b63887bd]
aws_security_group.alb_sg: Refreshing state... [id=sg-03b1f4989c23ef8b8]
aws_internet_gateway.igw: Refreshing state... [id=igw-0224967fc12e05579]
aws_subnet.public[0]: Refreshing state... [id=subnet-0afbf12da8040d912]
aws_subnet.public[1]: Refreshing state... [id=subnet-0f4fc779afe18662]
```

The screenshot shows the AWS CloudShell interface in the us-east-1 region. The terminal window displays the following command sequence:

```
ubuntu@ip-172-31-18-150:~/upgrad-project/terraform$
```



Task : 2

- 👉 Install Ansible on Bastion Machin & create inventory.yaml and playbook.yaml

◇ SSH → bastion machine → run Below Commands

```
git clone https://github.com/sagardpatil0055/upgrad-project.git  
cd upgrad-project/ansible  
sh install_ansible.sh
```

```
ubuntu@ip-10-0-1-247:~$ git clone https://github.com/sagardpatil0055/upgrad-project.git  
Cloning into 'upgrad-project'...  
remote: Enumerating objects: 171, done.  
remote: Counting objects: 100% (171/171), done.  
remote: Compressing objects: 100% (144/144), done.  
remote: Total 171 (delta 52), reused 0 (delta 0), pack-reused 0 (from 0)  
Receiving objects: 100% (171/171), 49.77 KiB | 3.83 MiB/s, done.  
Resolving deltas: 100% (52/52), done.  
ubuntu@ip-10-0-1-247:~$ cd upgrad-project/ansible  
ubuntu@ip-10-0-1-247:~/upgrad-project/ansible$ sh install_ansible.sh
```

```
ubuntu@ip-10-0-4-115:~$ docker ps  
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS     NAMES  
ubuntu@ip-10-0-4-115:~$
```

```
ubuntu@ip-10-0-3-146:~$ docker ps  
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS     NAMES  
ubuntu@ip-10-0-3-146:~$
```

- 👉 Install Jenkins on jenkins machine

◇ SSH = bastion → SSH = jenkins → run the below commands

```
sudo apt update  
sudo apt install openjdk-21-jdk -y  
java -version
```

```
ubuntu@ip-10-0-4-115:~$ ubuntu@ip-10-0-4-115:~$ java -version
java version "21.0.7" 2025-04-15
OpenJDK Runtime Environment (build 21.0.7+6-Ubuntu-0ubuntu124.04)
OpenJDK 64-Bit Server VM (build 21.0.7+6-Ubuntu-0ubuntu124.04, mixed mode, sharing)
ubuntu@ip-10-0-4-115:~$
```

❖ Install → Jenkins

```
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 update

sudo apt install jenkins -y
```

```
ubuntu@ip-10-0-4-115:~$ /jenkins.list > /dev/null
tee: '/etc/apt/sources.list.d/jenkins.list': No such file or directory
ubuntu@ip-10-0-4-115:~$ 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
ubuntu@ip-10-0-4-115:~$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
--2025-07-19 19:13:44-- https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
Resolving pkg.jenkins.io (pkg.jenkins.io)... 146.75.34.133, 2a04:4e42:79::645
Connecting to pkg.jenkins.io (pkg.jenkins.io)|146.75.34.133|: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-k 100%[=====] 3.10K --.KB/s in 0s
2025-07-19 19:13:44 (41.8 MB/s) - '/usr/share/keyrings/jenkins-keyring.asc' saved [3175/3175]

ubuntu@ip-10-0-4-115:~$ 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
ubuntu@ip-10-0-4-115:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Ign:4 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:5 https://pkg.jenkins.io/debian-stable binary/ Release [2044 B]
Get:6 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [933 B]
Hit:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:8 https://pkg.jenkins.io/debian-stable binary/ Packages [29.4 kB]
Fetched 32.3 kB in 1s (42.1 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
84 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-10-0-4-115:~$ sudo apt install jenkins -y
```



◊ Adding → App node

A screenshot of the Jenkins 'Nodes' configuration page. A new node named 'app' is being created. The configuration includes: Name: app, Description: (empty), Number of executors: 2, Remote root directory: /home/ubuntu, Labels: (empty). A 'Save' button is at the bottom. A watermark for 'Activate Windows' is visible on the right.

👉 Create ECR Repository in Console

- 1) Go to: [Amazon ECR Console](#)
- 2) Click “Create repository”
- 3) Set:
 - ◊ Repository name: **test-nodeapp-repo**
- 4) Visibility settings: **Private**

- 5) Tag immutability: **Mutable (default is fine)**
 - 6) Scan on push: **Enabled**
 - 7) Click “**Create repository**”
 - 8) Note the Repository URI :
- 980104576357.dkr.ecr.us-east-1.amazonaws.com/test-nodeapp-repo

The screenshot shows the 'Create private repository' page in the AWS ECR console. The URL in the address bar is `us-east-1.console.aws.amazon.com/ecr/private-registry/repositories/create?region=us-east-1`. The page has a breadcrumb navigation: Amazon ECR > Private registry > Repositories > Create private repository. The main section is titled 'Create private repository' and contains two main settings sections: 'General settings' and 'Encryption settings'. In the 'General settings' section, the 'Repository name' field is filled with `980104576357.dkr.ecr.us-east-1.amazonaws.com/test-nodeapp-repo`. Below it, there's a note about image tag mutability with two options: 'Mutable' (selected) and 'Immutable'. In the 'Encryption settings' section, there's a note that encryption settings can't be changed once the repository is created. The 'Encryption configuration' dropdown is set to 'AES-256'. A 'Activate Windows' watermark is visible in the bottom right corner.



Task 3

- 👉 Create a test-nodeapp project on Github

◇ <https://github.com/sagardpatil0055/test-nodeapp.git>

◇ Write a dockerfile to **test-nodeapp**

The screenshot shows a GitHub repository named 'test-nodeapp'. The 'Files' tab is selected, displaying the contents of the repository. A file named 'dockerfile' is highlighted. The code editor shows the following Dockerfile:

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
EXPOSE 8881
CMD ["npm", "start"]
```

- 👉 STEP 1: Testing Docker build & running it on bastion .

- ◇ Install docker on bastion
◇ Check for docker installation on bastion

```
ubuntu@ip-10-0-1-133:~$ docker ps
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS          NAMES
ubuntu@ip-10-0-1-133:~$
```

```
git clone https://github.com/sagardpatil0055/test-nodeapp.git
```

```
cd test-nodeapp
```

```
docker build -t test-nodeapp . # build project
```

```
ubuntu@ip-10-0-1-133:~/test-nodeapp$ docker build -t test-nodeapp .
[+] Building 0.2s (10/10) FINISHED
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 156B
=> [internal] load metadata for docker.io/library/node:18-alpine
=> [internal] load .dockerignore
=> => transferring context: 62B
=> [1/5] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
=> [internal] load build context
=> => transferring context: 163B
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY package*.json .
=> CACHED [4/5] RUN npm install
=> CACHED [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:028fdfd46f5084ce542daed0c5aa0ac58b5e650e7557b6db6a430f4d899b9d0
=> => naming to docker.io/library/test-nodeapp
ubuntu@ip-10-0-1-133:~/test-nodeapp$
```

Run the container

```
docker run -d -p 8081:8081 --name test-nodeapp test-nodeapp
```

```
ubuntu@ip-10-0-1-133:~/test-nodeapp$ docker run -d -p 8081:8081 --name test-nodeapp test-nodeapp
2fa566c124277e1e3c9d4842b5ae3b3e0ec75732f6346a4e9e8356d46a75e126
ubuntu@ip-10-0-1-133:~/test-nodeapp$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
2fa566c12427 test-nodeapp "docker-entrypoint.s..." 8 seconds ago Up 7 seconds 0.0.0.0:8081->8081/tcp, [::]:8081->8081/tcp test-nodeapp
ubuntu@ip-10-0-1-133:~/test-nodeapp$
```

👉 Output :



- 👉 Push the docker file of the node application in the same GitHub repository.

```
git init
git remote add origin https://github.com/sagardpatil0055/test-nodeapp.git
git add .
git commit -m "Add Dockerfile to Dockerize Node.js app"
git branch -M main    # Rename to 'main' to match GitHub default
git push -u origin main          # Push to 'main' branch
```

```

[ubuntu@ip-10-0-1-133:~/test-nodeapp]
ubuntu@ip-10-0-1-133:~/test-nodeapp$ nano server.js
ubuntu@ip-10-0-1-133:~/test-nodeapp$ git init
Reinitialized existing Git repository in /home/ubuntu/test-nodeapp/.git/
ubuntu@ip-10-0-1-133:~/test-nodeapp$ git remote add origin https://github.com/sagardpatil0055/test-nodeapp.git
error: remote origin already exists.
ubuntu@ip-10-0-1-133:~/test-nodeapp$ git add .
ubuntu@ip-10-0-1-133:~/test-nodeapp$ git commit -m "Changes"
[main c09aba0] Changes
Committer: Ubuntu <ubuntu@ip-10-0-1-133.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 1 insertion(+), 1 deletion(-)
ubuntu@ip-10-0-1-133:~/test-nodeapp$ git push -u origin main
Username for 'https://github.com': sagardpatil0055
Password for 'https://sagardpatil0055@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/sagardpatil0055/test-nodeapp.git
    7b500bf..c09aba0  main -> main
branch 'main' set up to track 'origin/main'.
ubuntu@ip-10-0-1-133:~/test-nodeapp$
```

👉 STEP 2: Create IAM Role for EC2 ECR Access

Create IAM Role

- 1) Go to: IAM Console – Roles
- 2) Click “Create role”

diamond Trusted entity type: AWS service

Use case: EC2

[Click Next](#)

👉 Attach ECR Access Policy

- 3) Search and select the following managed policy:
 - **AmazonEC2ContainerRegistryFullAccess**
- 4) Click **Next**
- 5) Role name: **ecr-instance-role**
- 6) Click **Create role**

👉 STEP 3: Attach IAM Role to Jenkins & App EC2

- 1) Go to: **EC2 Console**
- 2) Find your Jenkins and App instances

For each instance:

- ❖ Select instance → Click **Actions** → **Security** → **Modify IAM Role**
- ❖ Select **ecr-instance-role** and click **Update IAM role**

- 3) Authenticate EC2
 - ❖ Done using ansible **playbook** .
- 4) Use **amazon-ecr-credential-helper**
 - ❖ Install the **ECR Credential Helper** on all agent Done it via Ansible

```
sudo apt-get update
```

```
sudo apt-get install amazon-ecr-credential-helper
```

👉 Step 4 : Configure Docker to Use Credential Helper Done it via Ansible

Edit or create the file: → `~/docker/config.json`

Add the following content:

```
{  
  "credHelpers": {  
    "980104576357.dkr.ecr.us-east-1.amazonaws.com": "ecr-login"  
  }  
}
```

👉 Step 5 : Add Public Key to App EC2

- ❖ Done it using **Ansible** copied **project.pem** to app machine .

👉 Step 6: Configured Jenkins CICD for test-nodeapp using Declarative SCM & Jenkinsfile



Add node in Jenkins .

New node

Node name

app

Type

Permanent Agent
Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

Create

Name

app

Description

app

Plain text Preview

Number of executors

2

Remote root directory

/home/ubuntu

Labels

app

Usage

Use this node as much as possible

Launch method

Launch agents via SSH

◇ Creating test-nodeapp job

<https://github.com/sagardpatil0055/test-nodeapp.git>

The screenshot shows the Jenkins Pipeline configuration screen for a job named 'test-nodeapp'. The 'Pipeline' tab is selected in the sidebar. Under the 'Definition' section, 'Pipeline script from SCM' is chosen. The 'SCM' dropdown is set to 'Git'. In the 'Repositories' section, the 'Repository URL' is set to 'https://github.com/sagardpatil0055/test-nodeapp.git'. The 'Credentials' dropdown is set to '- none -'. There is a '+ Add' button for adding credentials. An 'Advanced' dropdown is also present.

◇ Jenkinsfile location

The screenshot shows the GitHub repository page for 'sagardpatil0055/test-nodeapp'. The 'Code' tab is selected. The repository name 'test-nodeapp' is displayed. Below it, there is a list of files and their commit history:

File / Commit	Description	Time
sagardpatil0055 Update server.js	Update server.js	66d2319 · 19 hours ago
.dockerignore	Add files via upload	2 days ago
Jenkinsfile	Update Jenkinsfile	19 hours ago
docker-compose.yaml	Create docker-compose.yaml	2 days ago
dockerfile	Update dockerfile	2 days ago
package.json	Add files via upload	2 days ago
server.js	Update server.js	19 hours ago

At the bottom, there is a 'README' section with a 'Add a README' button.

👉 Webhook url configuration for automation

The screenshot shows the GitHub settings page for a repository named 'test-nodeapp'. In the left sidebar, under 'Code and automation', the 'Webhooks' option is selected. A single webhook is listed with the URL 'http://sagar-alb-393990404.us-east-1.elb.amazonaws.com:8080/job/test-nodeapp/'. The status indicates 'Last delivery was successful.' There are 'Edit' and 'Delete' buttons next to the webhook entry.

👉 Running CI/CD for test-nodeapp with webhook

The screenshot shows the Jenkins pipeline status for the 'test-nodeapp' pipeline. The pipeline has two stages: '#17' and '#10'. Stage #17 is labeled 'Success' and has a 'Logs' button. Stage #10 is also labeled 'Success'. The pipeline summary indicates an average stage time of 125ms. The Jenkins interface includes a sidebar with options like Status, Changes, Build with Parameters, Configure, Delete Pipeline, Full Stage View, GitHub, Stages, Rename, Pipeline Syntax, and GitHub Hook Log. The top navigation bar shows the Jenkins logo and the current user 'admin'.

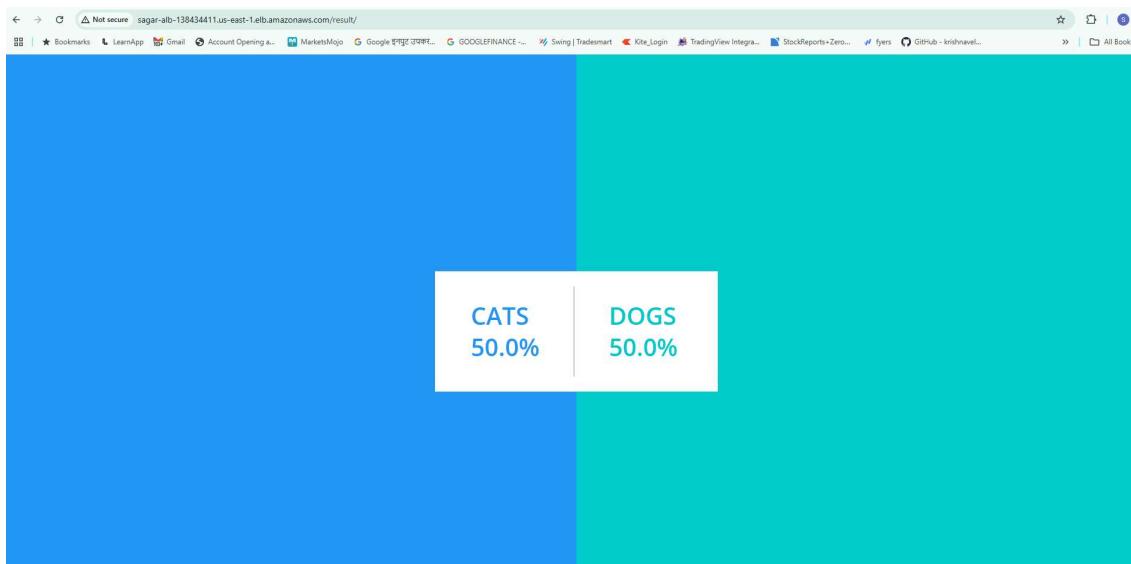
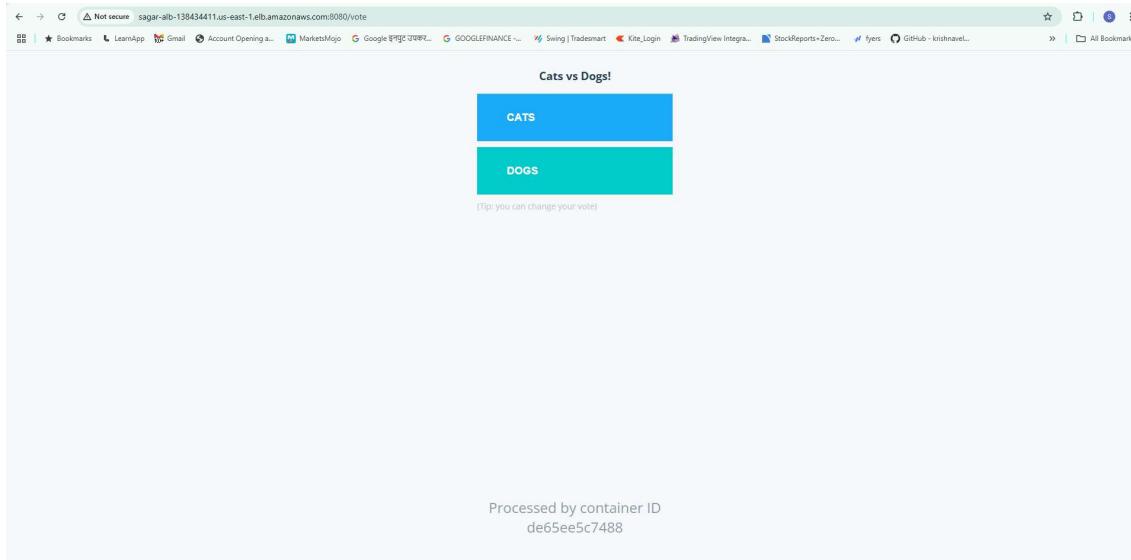
👉 Vote App installation using Jenkins CI/CD

◇ <https://github.com/sagardpatil0055/example-voting-app.git>

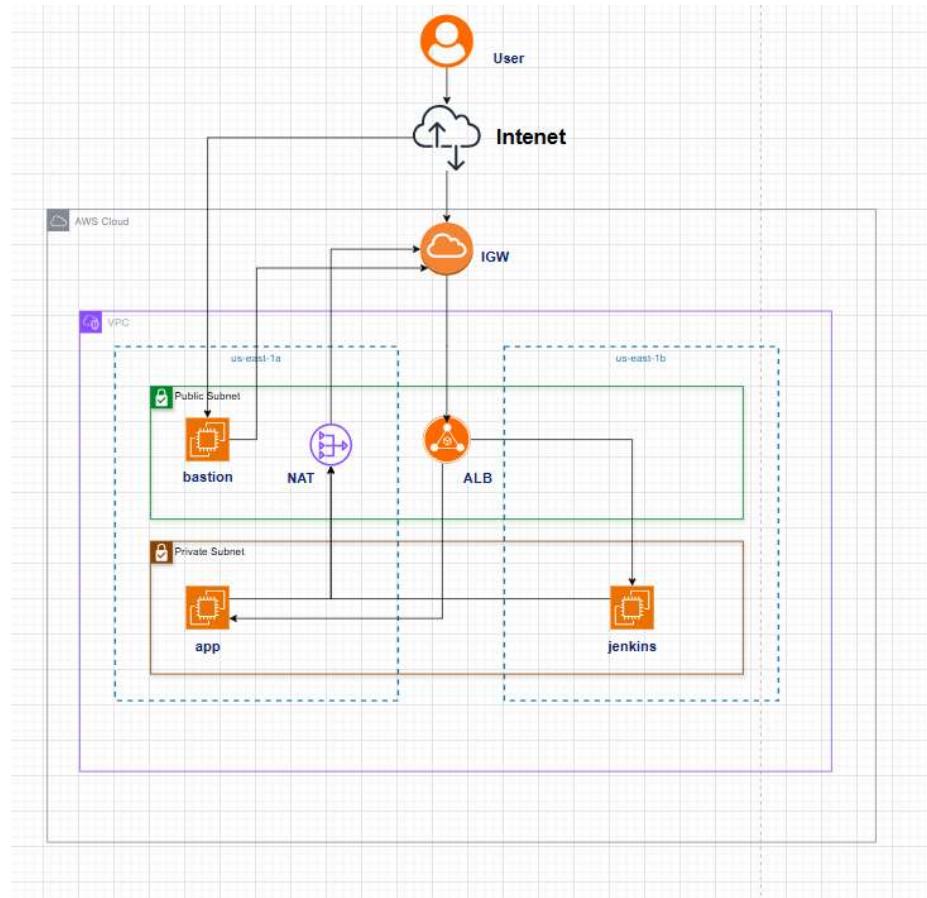
The screenshot shows the Jenkins job configuration page for 'vote-app'. The 'General' section is selected. The 'GitHub project' field contains the URL <https://github.com/sagardpatil0055/example-voting-app.git>. A 'String Parameter' named 'BRANCH' is defined with a default value of 'master'. The 'Save' button is visible at the bottom.

The screenshot shows the Jenkins Stage View for the 'vote-app' pipeline. The pipeline consists of several stages: Declarative: Checkout SCM, Clone Repository, Build & Push Vote Image, Build & Push Result Image, Fetch App Private IPs, Deploy Using Docker Compose, and Declarative: Post Actions. The 'Builds' table lists multiple runs, each with its status, duration, and specific steps. For example, run #10 took 313ms for the 'Build & Push Vote Image' step. The 'Builds' table also includes columns for 'Filter' and 'Today'.

Vote app Output



👉 Three-tier AWS Architecture Diagram .



👉 Full Monthly Cost Estimate :

Resource	Quantity	Unit Price (us-east-1)	Monthly Estimate
EC2 Instances	3 × t2.small	\$0.023/hr × 730 hrs	\$50.37
NAT Gateway	1	\$0.045/hr × 730 hrs	\$32.85
Elastic IP	1 (attached)	Free while attached	\$0.00
Amazon S3 (backend)	~1 object	~\$0.03/GB for backend	\$0.01
Amazon ECR	~0.9 GB	\$0.10/GB	\$0.09
Security Groups, Subnets, IGW, VPC, Route Tables	Included in AWS	Free	\$0.00

Total Monthly Estimate:
≈ \$83.32/Month

Total Hourly Estimate: ≈ \$0.116/hr