Task 14

Deploying Node.js Application On EC2 Using Terraform and Jenkins

Umar Satti

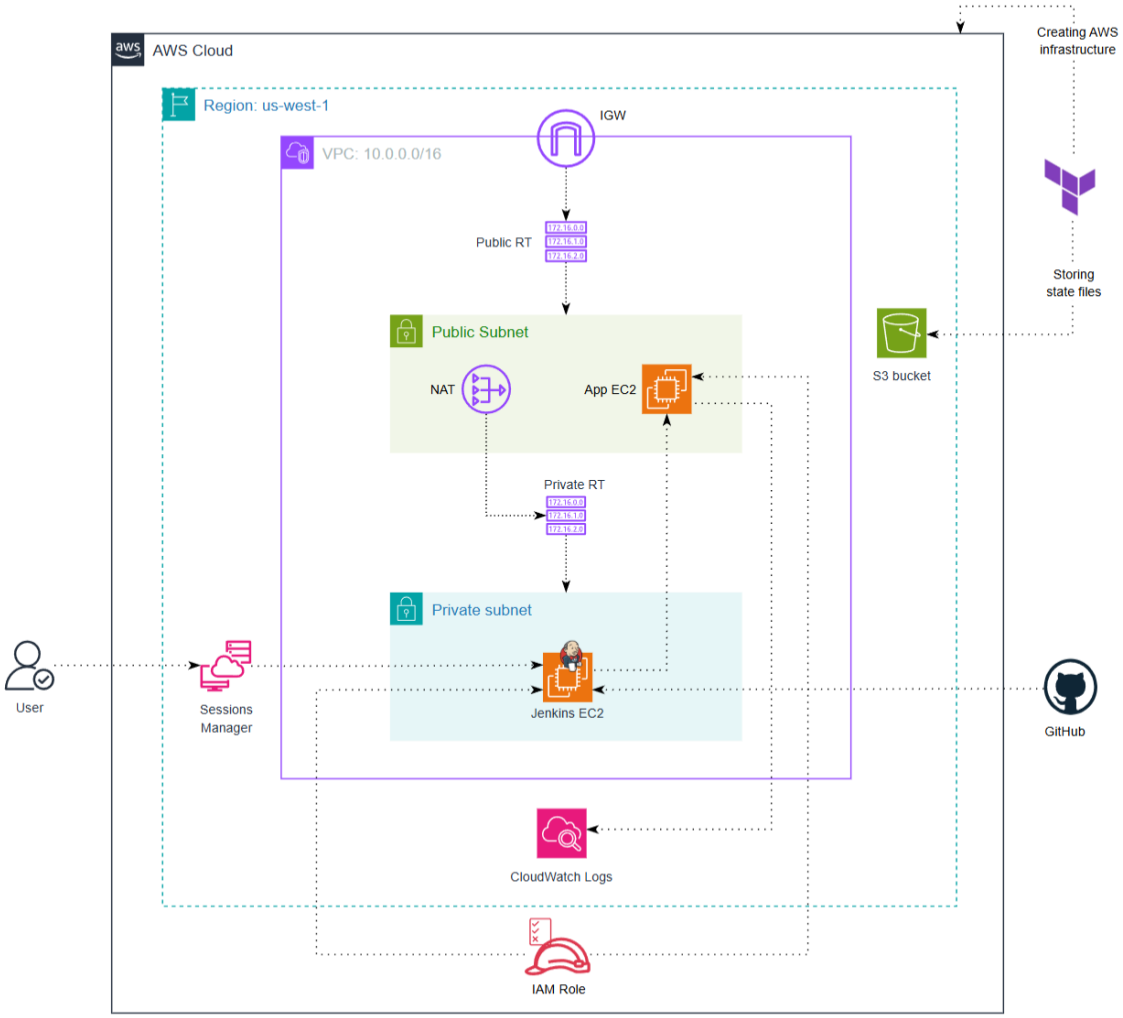
Table of Contents

1. Task Description

This project implements a CI/CD pipeline using Jenkins, Terraform, and Amazon EC2 Instances. All infrastructure including VPC networking, IAM roles, Application EC2, Jenkins EC2, and CloudWatch logs are provisioned using Terraform.

Jenkins is set up on a private EC2 instance without public IPs and is accessed securely via AWS Systems Manager (SSM) Session Manager. The Node.js application runs using PM2 process manager and Nginx. The application code is built, pushed, and deployed to Public-facing EC2 instance using Jenkins pipeline script. The pipeline also implements git pulls, rollbacks and health checks.

2. Architecture Diagram



3. Project Structure

This project is organized into application source files and Terraform infrastructure code. The structure separates application logic, CI/CD configuration, and infrastructure provisioning for clarity and maintainability.

3.1 Application Files

These files define the Node.js application that is deployed to Public EC2 instance.

3.1.1 app.js

- Main application entry point.
- Uses Express to serve static files from the public directory.
- Listens on port 3000 (or environment variable PORT).
- Includes a /health endpoint for basic health checks.

3.1.2 package.json

- Defines application metadata and dependencies.
- Specifies Node.js runtime version.
- Includes scripts for starting the application.
- Lists express as the primary dependency.

3.1.3 public/index.html

- Static HTML file served by the Node.js application.
- Displays a simple UI confirming successful deployment.
- Used to verify application availability after EC2 deployment.

3.2 Terraform Infrastructure Code

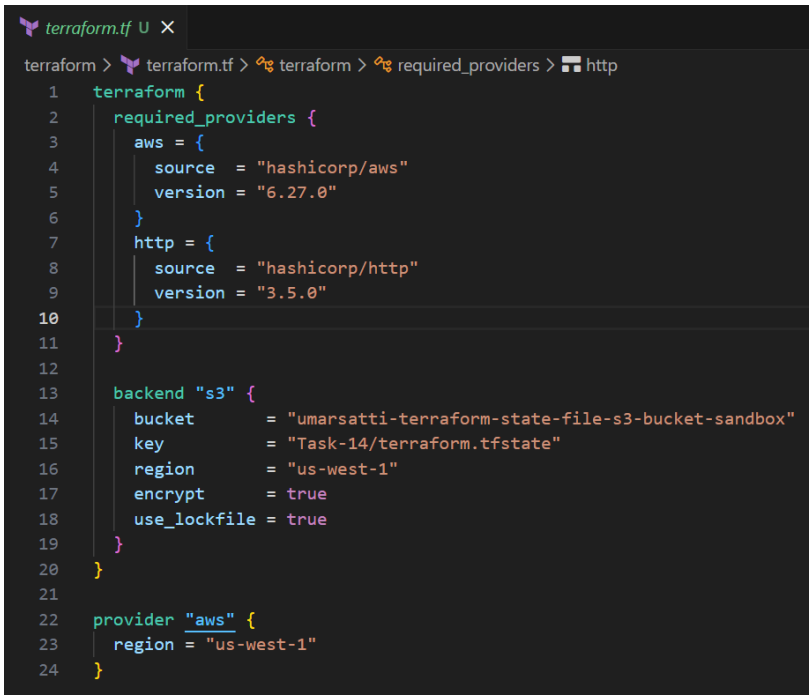
This directory contains Terraform configuration used to provision and manage all AWS infrastructure required for Jenkins, EC2, VPC, IAM, and CloudWatch. The setup follows a modular design to improve reusability and consistency.

3.2.1 Root Terraform Files

These files act as the entry point for Terraform and orchestrate all infrastructure modules.

1. terraform.tf

- Defines Terraform and AWS provider requirements.
- Locks the AWS provider version to 6.27.0 for consistency.
- Uses HTTP provider version 3.5.0 for fetching IP.
- Configures **remote state storage** using an S3 backend.
- Terraform state is stored in:
    - **S3 Bucket:** umarsatti-terraform-state-file-s3-bucket-sandbox
    - **State File Path:** Task-14/terraform.tfstate
    - **Region:** us-west-1
- Enables state encryption and locking to prevent concurrent modifications.
- Configures the AWS provider region as **us-west-1**.



2. main.tf

- Serves as the central orchestration file.
- Calls all Terraform modules required for the solution:
    - **VPC** – Networking i.e. subnets, IGW, NAT, route tables etc.
    - **IAM** – Roles and policies for EC2 instances.
    - **EC2** – Application and Jenkins instances along with log groups.
- Passes shared outputs (such as subnets, security groups, and IAM roles) between modules.
- Ensures proper dependency flow across infrastructure components.

3. variables.tf

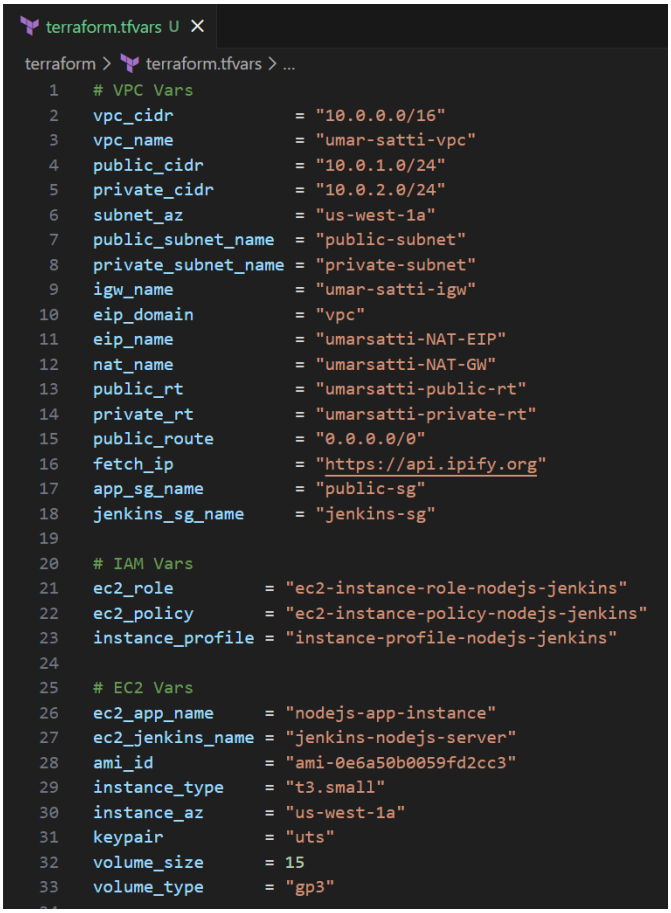
- Declares all input variables required by the root module.
- Groups variables logically by module:
  - VPC
  - IAM
  - EC2
- Enables flexible configuration without modifying core Terraform code.

4. terraform.tfvars

- Supplies concrete values for all declared variables.
- Defines environment-specific settings such as:
  - VPC CIDR and naming
  - EC2 AMI and instance sizes

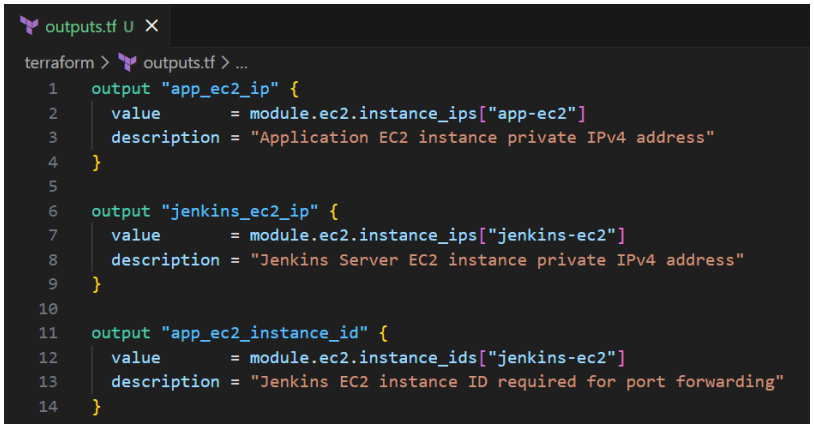
- ECS task sizing and launch type
- ALB listener and target group configuration

- Allows easy reuse of the same Terraform code across environments.



5. outputs.tf

- Exposes key infrastructure outputs after Terraform execution.
- Provides the following outputs for display after infrastructure creation:
  - Private IP addresses of Application and Jenkins EC2 instances.
  - Instance ID of Jenkins EC2 instance required for port forwarding.



3.2.2 Terraform Modules Directory

1. VPC Module

The VPC module provisions the complete networking layer, including subnets, routing, Internet gateway, NAT gateway, and security groups.

vpc/main.tf

- Defines a custom VPC with DNS support and hostname enabled.
- Creates a public and private subnet in **us-west-1a** AZ.
- Provisions Internet gateway and NAT gateway for public subnet.
- Configures public route table with internet access and Private route table with outbound internet access via NAT gateway.
- Implements security groups for Application EC2 (port 22, 3000, and 80) and Jenkins EC2 (port 8080 and port 22 from a single IP)
- Uses local variables to simplify subnet and routing mappings.

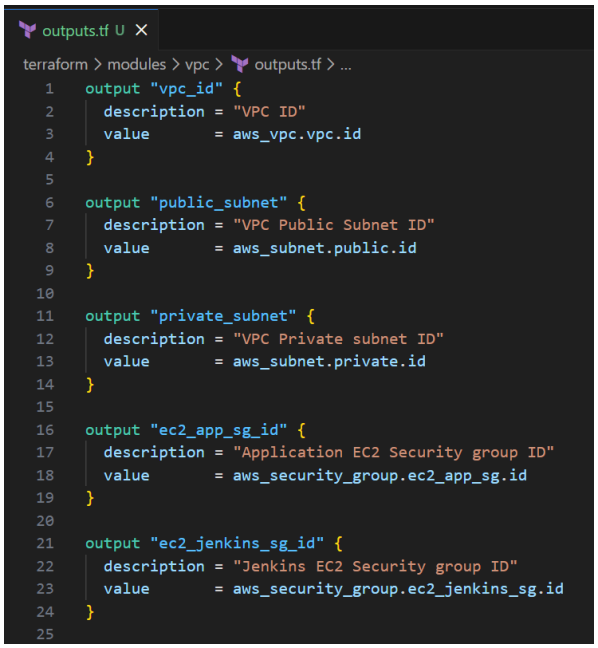
vpc/variables.tf

Declares input variables required to configure the VPC:

- CIDR blocks for VPC, public subnet, and private subnet.
- Naming for VPC, IGW, NAT, Subnets, Route tables, and Security groups.
- Internet and route parameters.

vpc/outputs.tf

Exposes key networking resources to other modules including VPC ID, public and private subnet IDs, and security group IDs for EC2 instances.



2. IAM Module

The IAM module provisions roles and policies required by application and Jenkins EC2 instances. These are attached to the instances as IAM Instance profiles.

iam/main.tf

Creates an **IAM role** with trust relationship for EC2 service as well as an Instance Profile for attaching the role to EC2 instances. Uses AWS Managed policies as well as external JSON document for inline policies. Include the following policy permissions:

- **AmazonSSMManagedInstanceCore** (for Session Manager access)
- **CloudWatchAgentServerPolicy** (for CloudWatch agent)

main.tf U

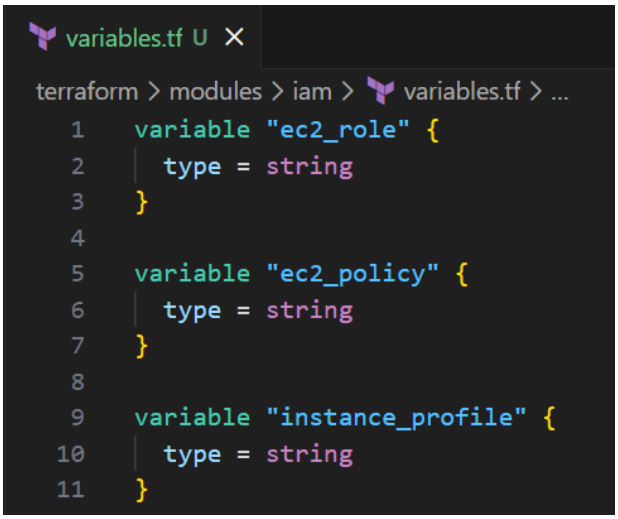
terraform > modules > iam > main.tf > ...

```hcl
# EC2 Trust relationship
data "aws_iam_policy_document" "ec2_assume_role" {
  statement {
    effect = "Allow"

    principals {
      type        = "Service"
      identifiers = ["ec2.amazonaws.com"]
    }

    actions = ["sts:AssumeRole"]
  }
}

# EC2 IAM Role
resource "aws_iam_role" "ec2_instance_role" {
  name               = var.ec2_role
  path               = "/"
  assume_role_policy = data.aws_iam_policy_document.ec2_assume_role.json
}

# Attach AWS Managed Policies
resource "aws_iam_role_policy_attachment" "ssm_policy" {
  role       = aws_iam_role.ec2_instance_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
}

resource "aws_iam_role_policy_attachment" "cwagent_policy" {
  role       = aws_iam_role.ec2_instance_role.name
  policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
}

# Instance Profile
resource "aws_iam_instance_profile" "instance_profile" {
  name = var.instance_profile
  role = aws_iam_role.ec2_instance_role.name
}
```

iam/variables.tf

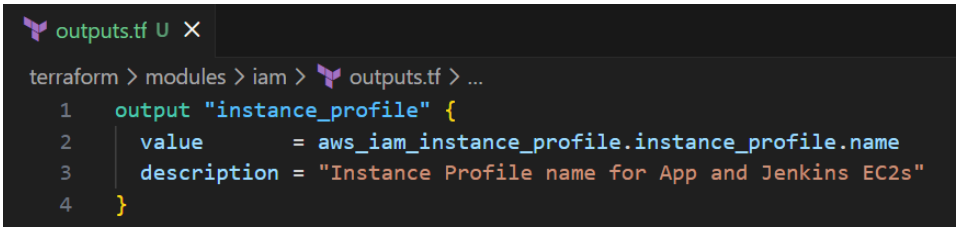
Declares input variables required to configure the IAM:

- EC2 role and policy names
- Instance profile name



iam/outputs.tf

Exposes the IAM instance profile name to be used by EC2 module.



3. EC2 Module

The EC2 module provisions the Jenkins Master and Jenkins Agent instances in private subnets.

ec2/main.tf

Uses a local map to define **application EC2** and **Jenkins EC2** instance. Both instances use the same AMI, instance type, IAM instance profile, and storage.

Application EC2:

- Placed in the public subnet.
- Attached to a security group allowing port 22, 80, and 3000 traffic.
- Uses a pre-configured key pair named **uts**.
- Bootstrapped using **app_ec2.sh** script file.

Jenkins EC2:

- Placed in the private subnet
- Uses a restricted security group by allowing port 22/8080 traffic from **My IP**.
- Bootstrapped using **jenkins_ec2.sh** script file.
- Does not use a key pair.

ec2/variables.tf

Declares input variables required to configure the EC2:

- EC2 names and AMI IDs.
- Instance IDs and Availability zones.
- EBS volume type and size.

Declares input variables that are referenced from other modules:

- Public and private subnets from VPC module.
- EC2 security groups from VPC module.
- IAM Instance profile from IAM module.

ec2/outputs.tf

Outputs private IPv4 addresses and Instance IDs of both EC2 instances.

outputs.tf U

terraform > modules > ec2 > outputs.tf > ...

```
output "instance_ips" {
  value       = { for i, instance in aws_instance.ec2 : i => instance.private_ip }
  description = "Private IPv4 address of EC2 instances using maps"
}

output "instance_ids" {
  value       = { for i, instance in aws_instance.ec2 : i => instance.id }
  description = "Instance IDs of EC2 instances using maps"
}
```

3.2.3 Supporting Terraform Files

This section describes user data scripts and policy documents used by Terraform to bootstrap EC2 instances and define IAM permissions.

Application EC2 Bootstrap Script (app_ec2.sh)

This script is executed automatically when the Application EC2 instance is created to prepare the runtime environment for a Node.js application.

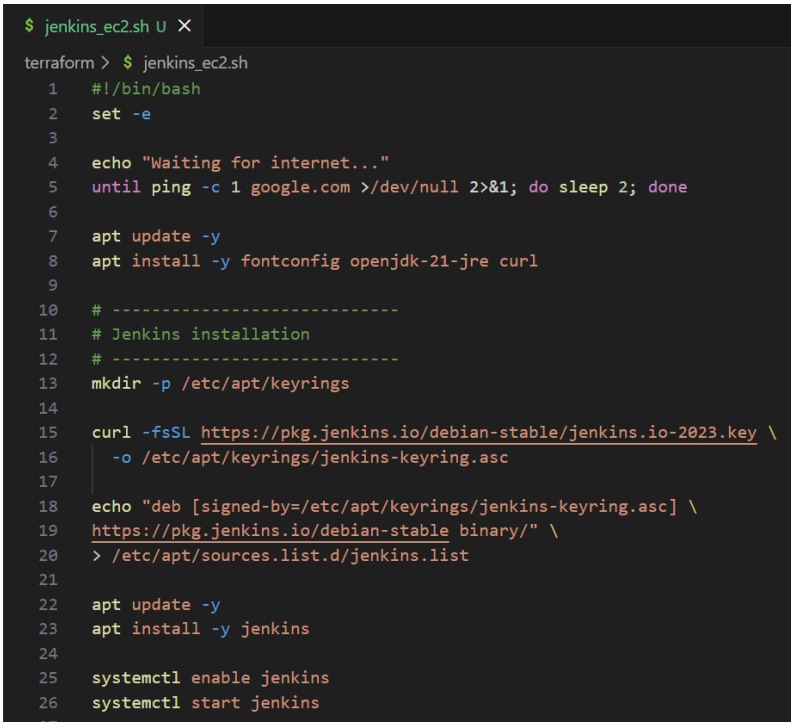
- Waits for full internet connectivity to ensure reliable package installation.
- Installs core system utilities including curl, Git, Nginx, and unzip.
- Installs Node.js 20 system-wide along with npm.
- Installs PM2 and configures it to start on boot using systemd.
- Creates structured application directories for rollback.
- Configures Nginx as a reverse proxy to forward HTTP traffic to Node.js app.
- Installs and configures the Amazon CloudWatch Agent.

This ensures the application server is production-ready immediately after provisioning, with process management, reverse proxying, and centralized logging in place.

Jenkins EC2 Bootstrap Script (jenkins_ec2.sh)

This script provisions the Jenkins EC2 instance used for CI/CD pipeline execution.

- Waits for internet availability before starting installation steps.
- Installs OpenJDK 21, which is required to run Jenkins.
- Adds the official Jenkins package repository and GPG key.
- Installs Jenkins from the stable repository.
- Enables and starts the Jenkins service automatically on boot.

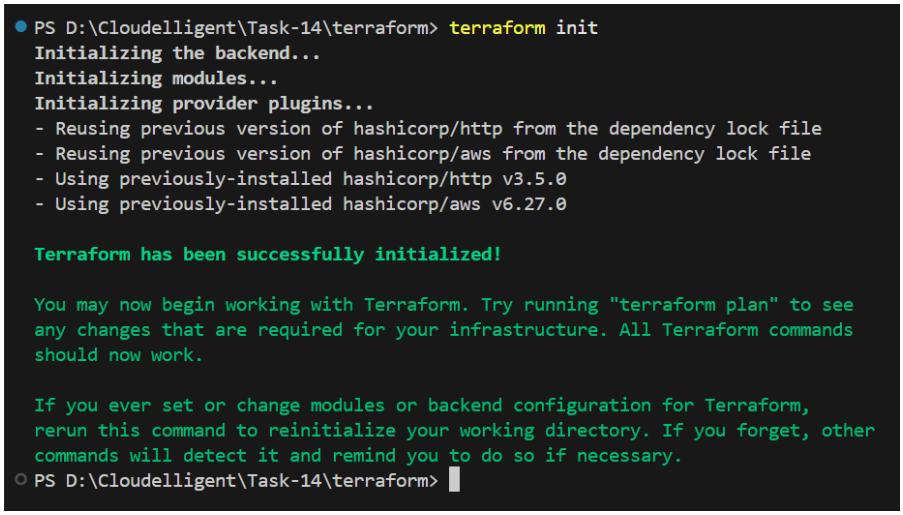


This guarantees that the Jenkins server is fully operational and accessible as soon as the EC2 instance is launched.

3.3 Terraform Commands

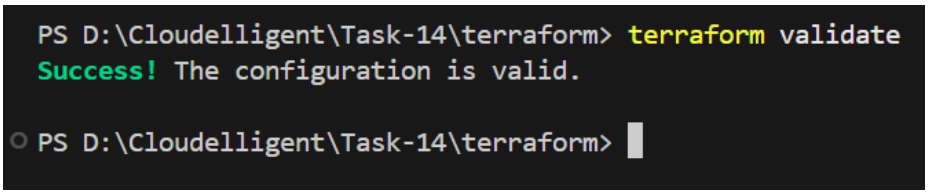
1. Terraform init

Initializes the Terraform working directory by downloading required providers and modules, configuring the remote backend, and preparing Terraform.



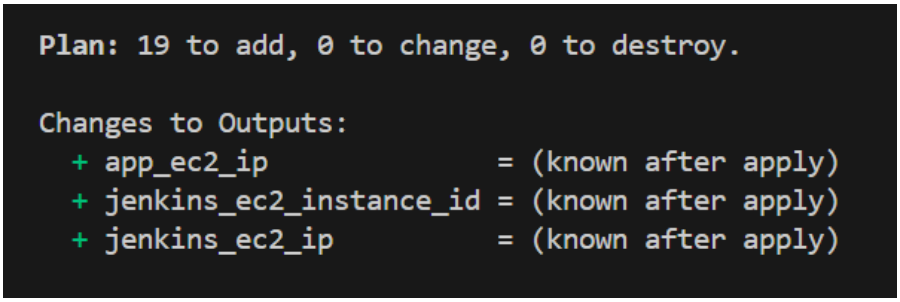
2. Terraform validate

Checks the Terraform configuration for syntax and logical errors without creating resources, ensuring the configuration is valid and consistent.



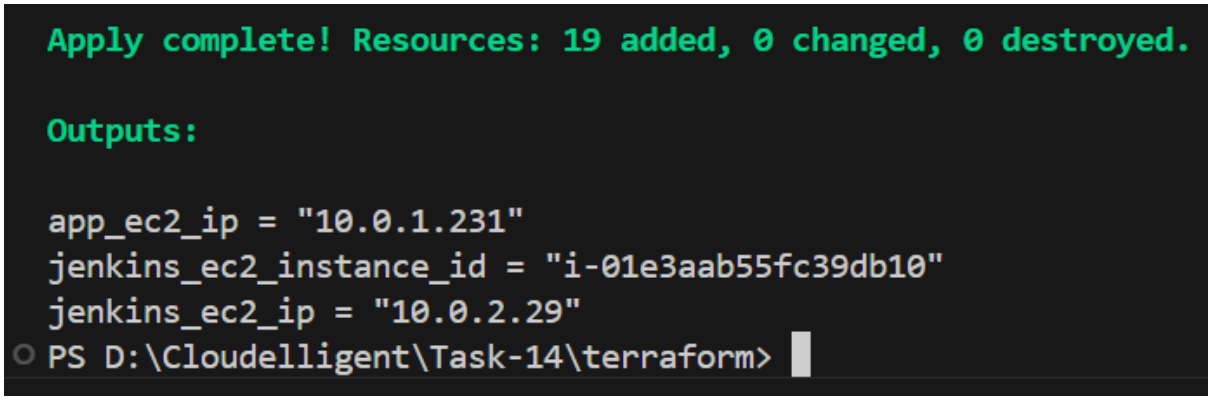
3. Terraform plan

Creates an execution plan that previews the AWS resources Terraform will create and any changes that will be applied, allowing review before deployment.



4. Terraform apply

The terraform apply command provisions the AWS infrastructure as defined in the Terraform configuration. Upon confirmation, Terraform creates all required resources defined in VPC, IAM, and EC2 modules.



Terraform also outputs key infrastructure details for later use:

- **Application EC2 Private IP:** 10.0.1.231
- **Jenkins EC2 Private IP:** 10.0.2.29
- **Jenkins EC2 instance ID:** i-01e3aab55fc39db10

These outputs are used for:

- Connecting Jenkins Master and Agent nodes.
- Accessing the deployed application through the ALB.
- Verifying successful infrastructure provisioning.

4. Validate Infrastructure in AWS

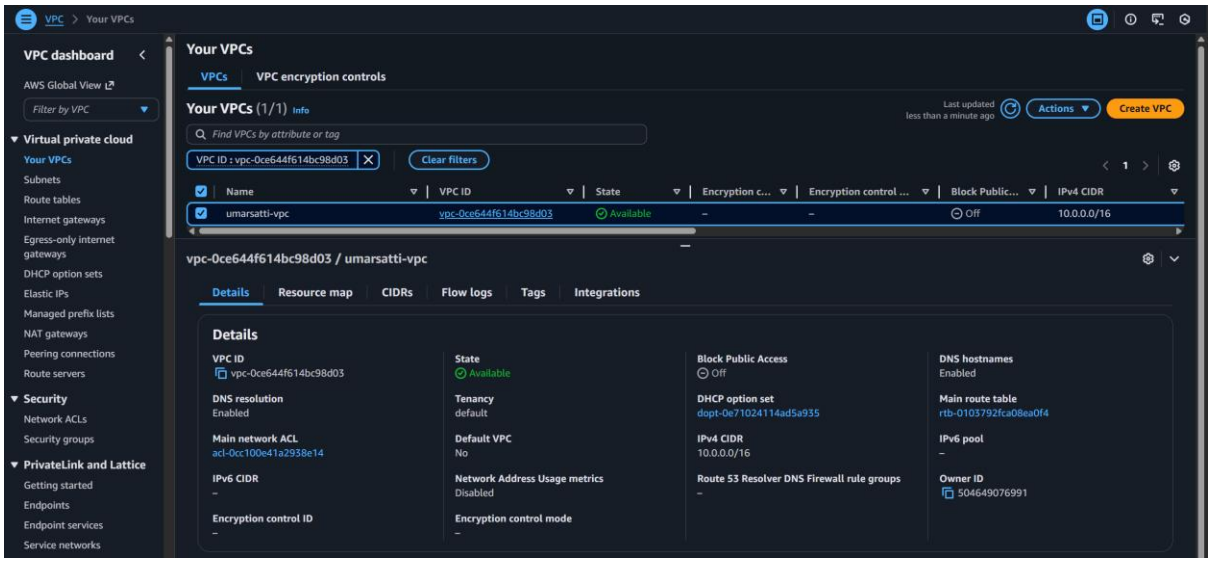
This section validates that all AWS resources created using Terraform are provisioned correctly and functioning as expected.

4.1 VPC and Networking Validation

1. Verify VPC

In the AWS Console, navigate to **VPC** service. Select **Your VPCs** and verify:

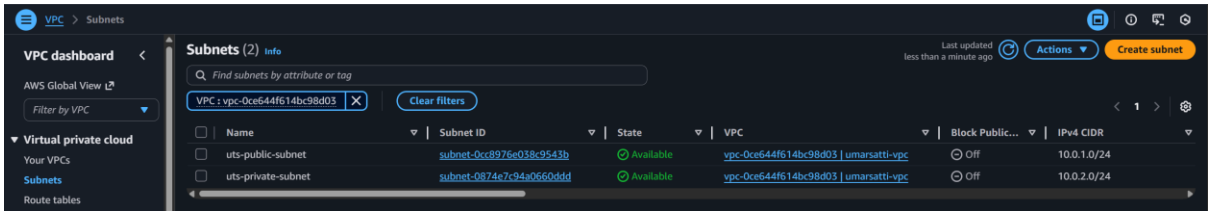
- VPC created by Terraform exists.
- Correct IPv4 CIDR block as defined in Terraform.
- DNS hostnames and DNS resolution are enabled.
- VPC name matches the Terraform configuration.
- The following shows the VPC configuration:
    - **Name:** umarsatti-vpc
    - **VPC ID:** vpc-0ce644f614bc98d03
    - **IPv4 CIDR:** 10.0.0.0/16



2. Verify Subnets

Navigate to **Subnets** section in the VPC console.

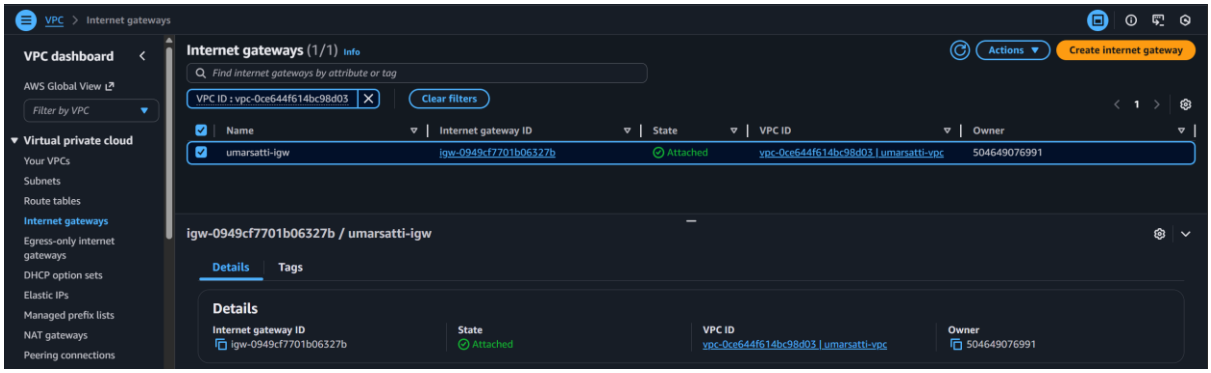
- Confirm creation of public and private subnets.
- The following shows the subnets configuration:
    - **Names:** uts-public-subnet & uts-private-subnet
    - **IPv4 CIDRs:** 10.0.1.0/24 & 10.0.2.0/24
    - **Subnet IDs:** subnet-0cc8976e038c9543b & subnet-0874e7c94a0660ddd



3. Internet Gateway

Navigate to **Internet Gateways** in the VPC console.

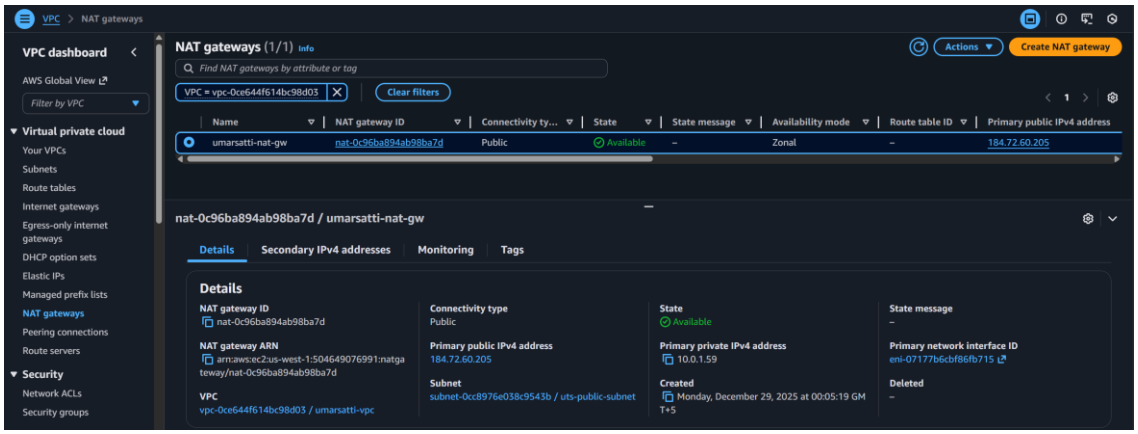
- Confirm that Internet gateway exists and **attached** to VPC.
- The following shows the IGW configuration:
  - **Name:** umarsatti-igw
  - **Internet gateway ID:** igw-0949cf7701b06327b
  - **State:** Attached



4. NAT Gateway

Navigate to **NAT gateways** in the VPC console.

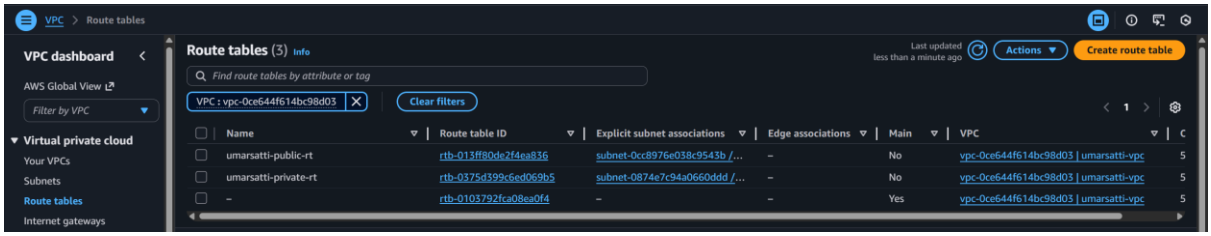
- Confirm that NAT Gateway exists in public subnet.
- Status shows **Available**.
- The following shows the NAT gateway configuration:
  - **Name:** umarsatti-nat-gw
  - **Nat gateway ID:** nat-0c96ba894ab98ba7d
  - **Elastic IP:** 184.72.60.205



5. Route Tables

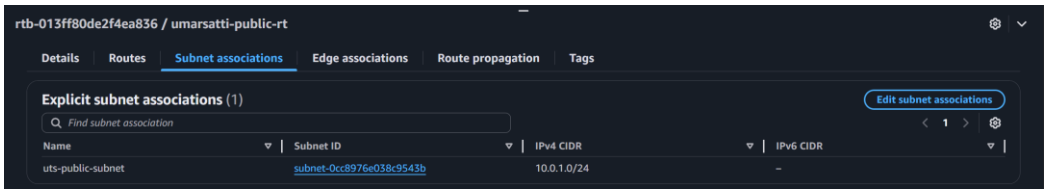
Navigate to **Route Tables** in the VPC console. Confirm that public and private route tables are created. The following shows the configuration:

- **Public route table name:** umarsatti-public-rt
- **Private route table name:** umarsatti-private-rt

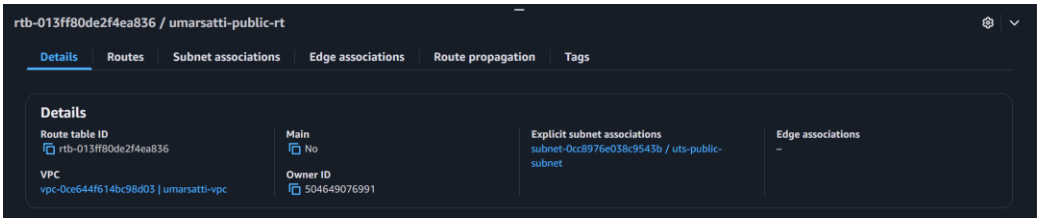


Select the **Public route table (umarsatti-public-rt)** and confirm the following.

- Explicit Association with public subnet. (in Subnet Associations tab).

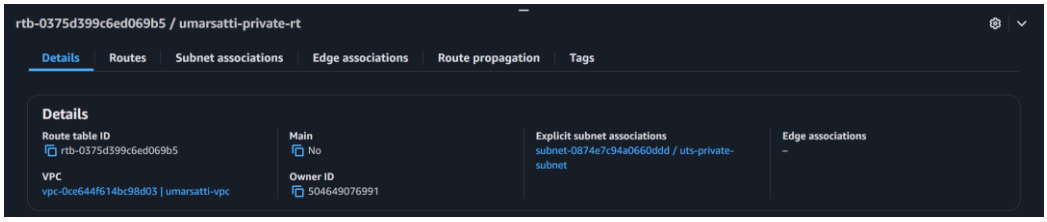


- Contains route **0.0.0.0/0 → Internet Gateway** (in Routes tab).

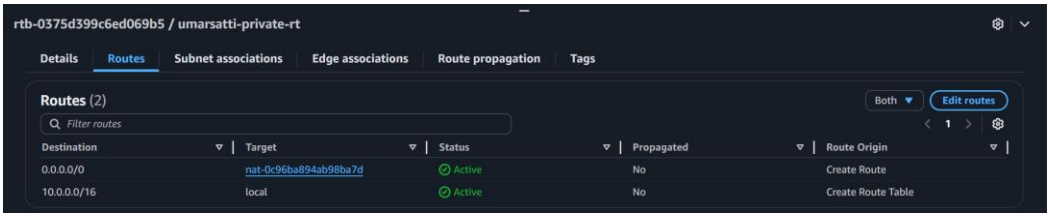


Select the **Private route table (umarsatti-private-rt)** and confirm the following.

- Explicit Association with public subnet. (in Subnet associations tab).

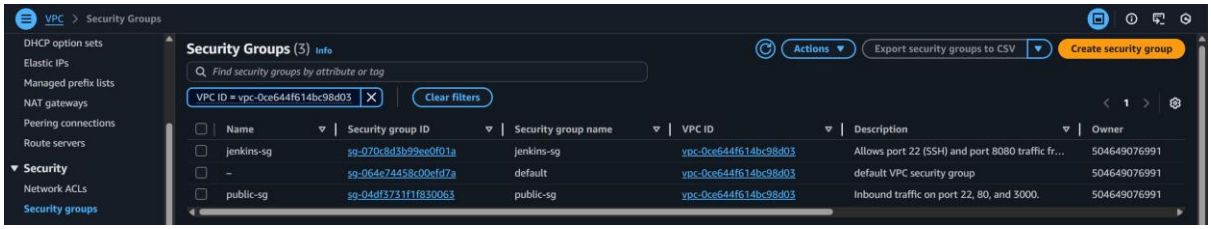


- Contains route **0.0.0.0/0 → NAT Gateway** (in Routes tab).



5. Security Groups

Navigate to **Security Groups** in the VPC console. Confirm that both the security groups are created. The following shows the security group configuration:



Application EC2 Security group

- **Inbound rules:** Traffic on Port 22, 80, and 3000.
- **Outbound rules:** All traffic allowed (0.0.0.0/0)

Jenkins EC2 Security group

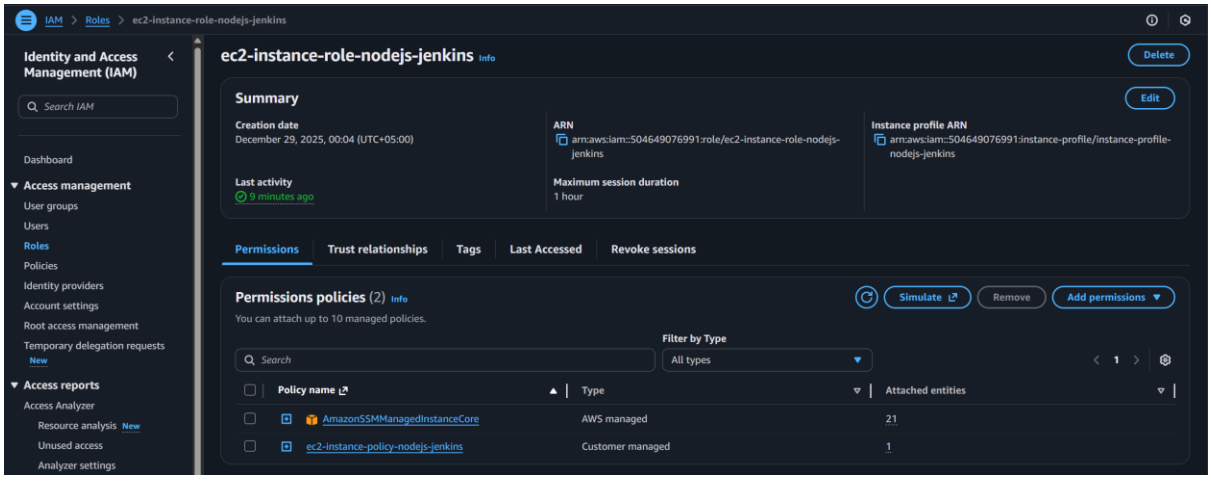
- **Inbound rules:** Traffic on Port 22 and 8080.
- **Outbound rules:** All traffic allowed (0.0.0.0/0)

4.2 IAM Validation

1. EC2 Instance Role

Navigate to **IAM** console and select **Roles**. Use the search bar to verify that the IAM role named **"ec2-instance-role-nodejs-jenkins"** exists. Select this IAM role and confirm the following:

- Trust relationship and Instance profile exists.
- Attached policy matches **instance_profile.json**.
- Permissions include SSM, EC2 messages, and Cloudwatch.

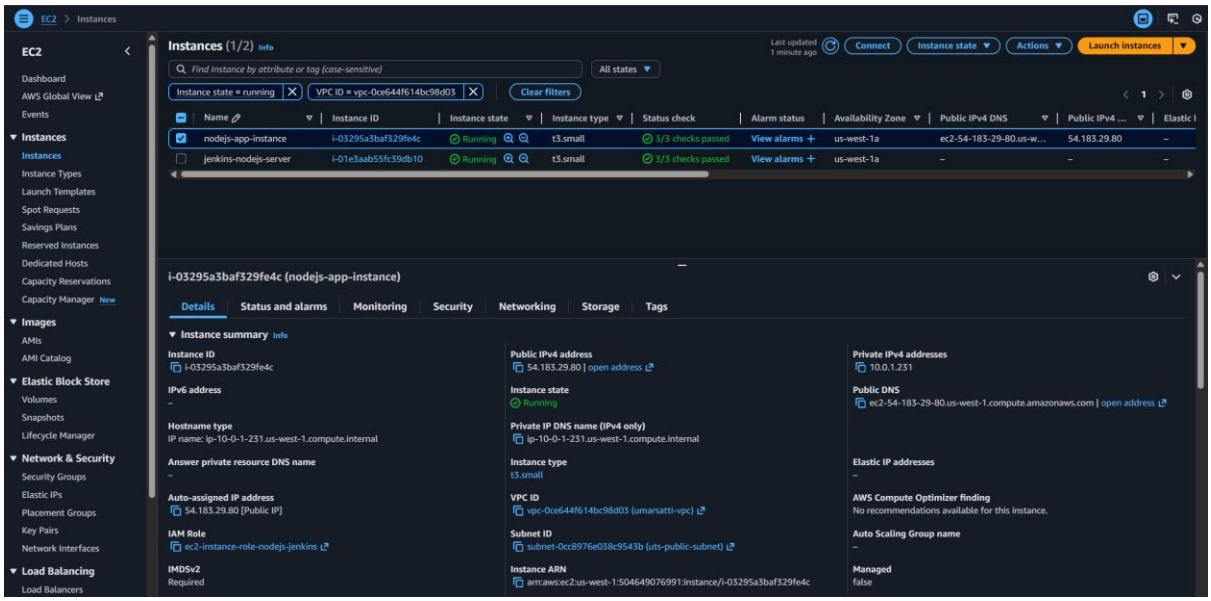


4.3 EC2 Validation

1. Application EC2 instance

Navigate to **EC2** console and select **Instances**. Select the application EC2 instance and verify the following:

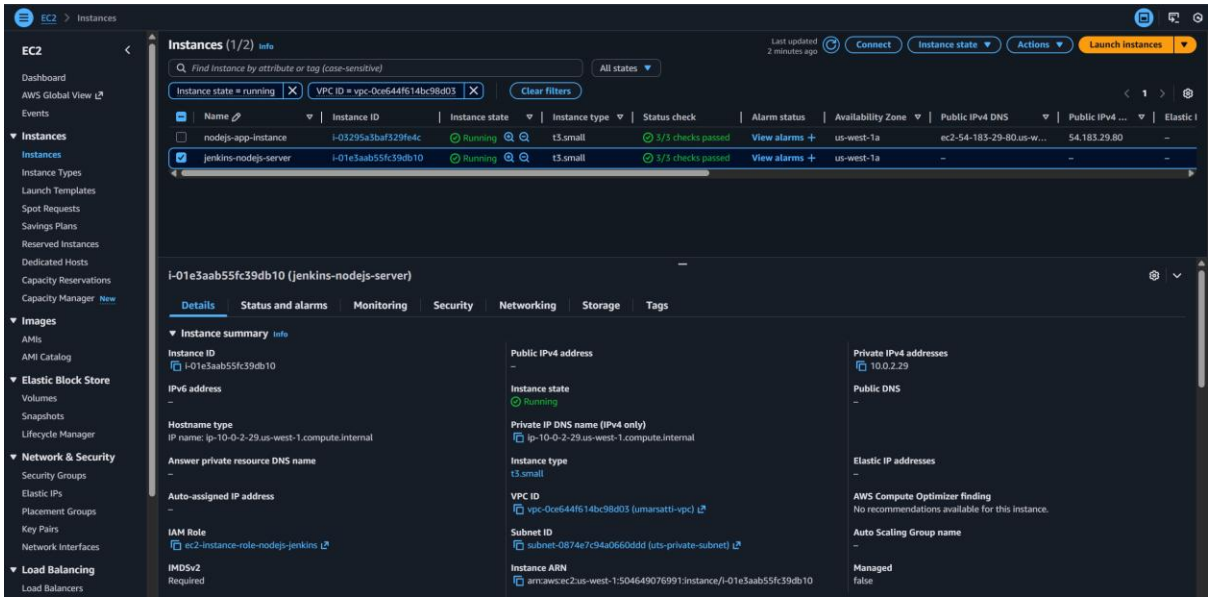
- Application EC2 **(nodejs-app-instance)** instance is created.
- Instance is running and status checks are passed.
- Located in **public subnet** with a public IP assigned to it.
- Has a private IPv4 address of 10.0.1.xx.
- IAM instance profile attached.
- Correct security group attached to it.
- Instance type matches Terraform configuration.



2. Jenkins EC2 instance

Navigate to **EC2** console and select **Instances**. Select the Jenkins EC2 instance and verify the following:

- Jenkins EC2 **(jenkins-nodejs-server)** instance is created.
- Instance is running and status checks are passed.
- Located in **private subnet** without a public IPv4 address.
- Has a private IPv4 address of 10.0.2.xx.
- IAM instance profile attached.
- Correct security group attached to it.
- Instance type matches Terraform configuration.



3. User Data Installations on Application EC2

Select the Application EC2 instance and click **Connect**. Use **SSM Session Manager** to connect to the instance. Verify the installations by using the following commands:

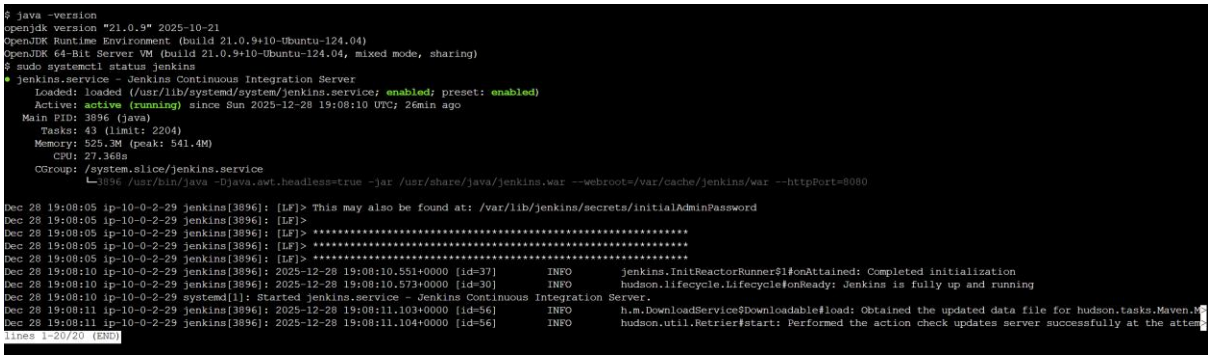
- For Node.js: ***node -v***
- For Npm: ***npm -v***
- For PM2: ***pm2 -v***
- For Nginx: ***sudo systemctl status nginx***
- For CloudWatch agent:
    - ***sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status***

```
$ node -v
v20.19.6
$ npm -v
10.8.2
$ pm2 -v
6.0.14
$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-12-28 19:06:37 UTC; 24min ago
       Docs: man:nginx(8)
   Main PID: 2975 (nginx)
      Tasks: 3 (limit: 2204)
     Memory: 2.4M (peak: 2.6M)
        CPU: 16ms
     CGroup: /system.slice/nginx.service
             ├─2975 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
             ├─2976 "nginx: worker process"
             └─2977 "nginx: worker process"

Dec 28 19:06:37 ip-10-0-1-231 systemd[1]: Starting nginx.service - A high performance web server and a reverse proxy server...
Dec 28 19:06:37 ip-10-0-1-231 systemd[1]: Started nginx.service - A high performance web server and a reverse proxy server.
$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status
{
  "status": "running",
  "starttime": "2025-12-28T19:06:44+00:00",
  "configstatus": "configured",
  "version": "1.300062.0b1304"
}
$
```

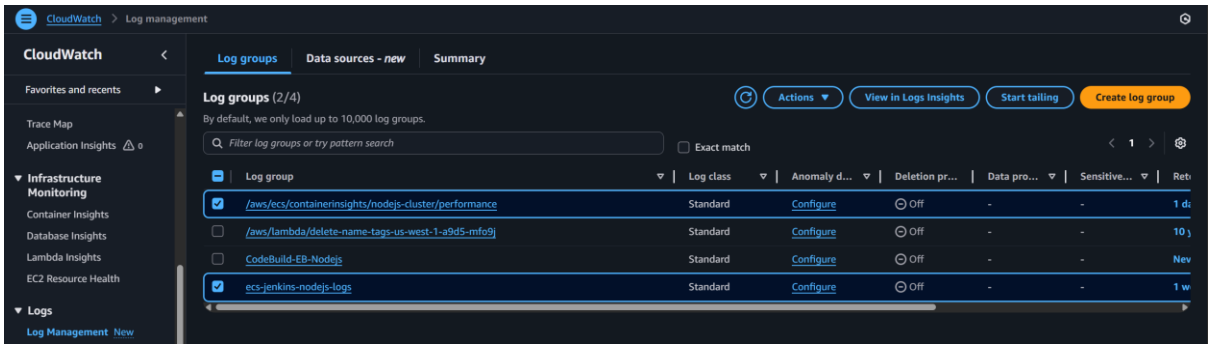
4. User Data Installations on Jenkins EC2

Select the Application EC2 instance and click **Connect**. Use **SSM Session Manager** to connect to the instance. Verify the installations by using the following commands:

- For Java/OpenJDK: ***java -version***
- For Jenkins: ***sudo systemctl status jenkins***



4.4 CloudWatch Validation (Pre-Deployment)



4.4 End-to-End Validation (Pre-Deployment)

Navigate to **EC2** console and select the Application EC2 instance named **nodejs-app-instance**. Copy the **Public IPv4** and paste it in a browser. Verify the following:

- URL: http://54.183.29.80
- Gives a **502 Bad Gateway** error because the application does not exist yet.

5. Jenkins Setup

This section describes the setup and validation of a Jenkins-based CI/CD pipeline to build, push, and deploy the Node.js application to Application EC2 instance. It covers accessing Jenkins UI, configuring Jenkins plugins, installing Node.js tools, storing credentials for private key, creating Jenkins pipeline script, and validating the automated deployment.

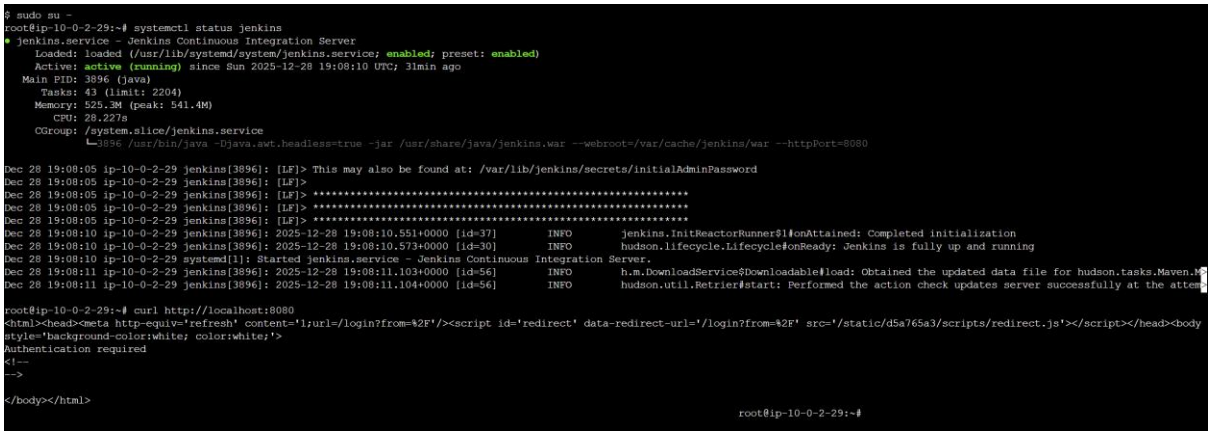
5.1 Use Port Forwarding to Access Jenkins UI

Since the Jenkins EC2 instance is in a private subnet, access from a local machine (Windows 11) is performed using **SSM port forwarding**.

Step 1: Verify Jenkins is Running

Connect to the Jenkins EC2 instance using **SSM Session Manager** and then verify the following:

- Check Jenkins service status. Confirm that it is active and running.
    - *sudo systemctl status Jenkins*
- Verify local access from the EC2 instance.
    - *curl http://localhost:8080*
    - The Jenkins landing page HTML should be returned.



Step 2: Access Jenkins UI via Port Forwarding

Pre-requisites for Port-forwarding on Windows machine include installation of AWS CLI and Sessions Manager Plugin. Verify using the following commands:

- ***aws --version***
- ***session-manager-plugin***

```
PS C:\Users\umars> aws --version
aws-cli/2.22.7 Python/3.12.6 Windows/11 exe/AMD64
PS C:\Users\umars> session-manager-plugin

The Session Manager plugin was installed successfully. Use the AWS CLI to start a session.
```

Confirm AWS CLI authentication using the following command. Expected output should include an AWS account ID along with an IAM identity:

- ***aws sts get-caller-identity***

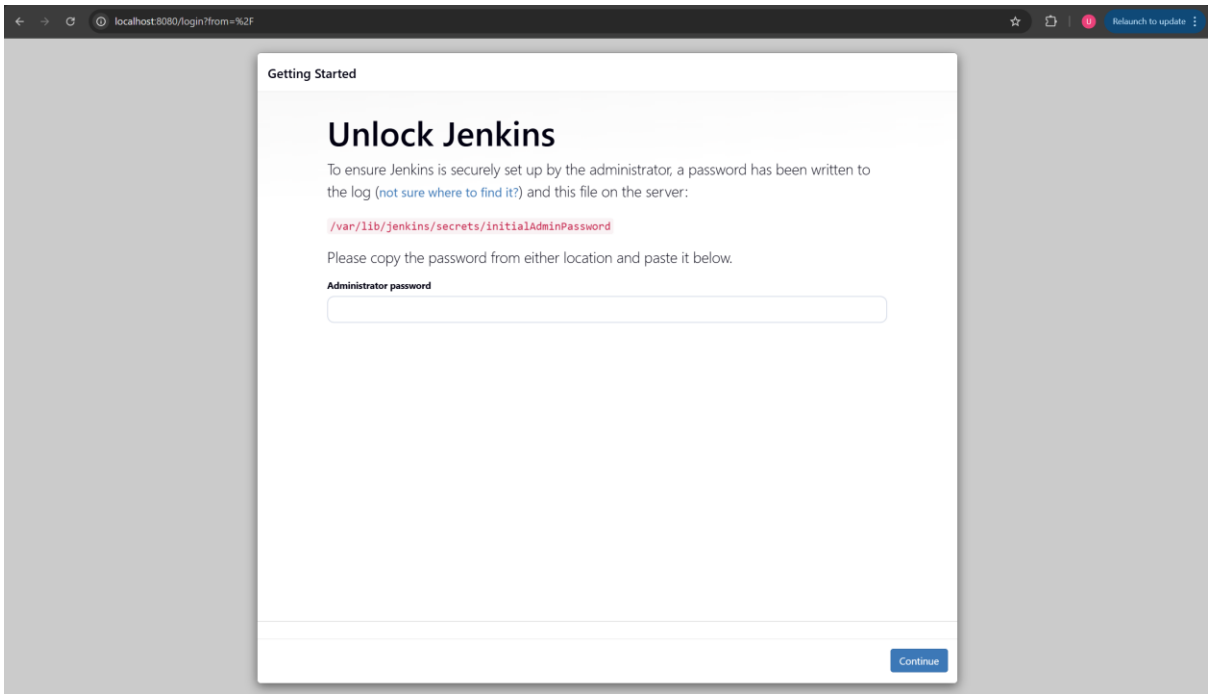
```
PS C:\Users\umars> aws sts get-caller-identity
{
    "UserId": "AROAXK73N2D7R4SMA6NA4:Umar.satti",
    "Account": "504649076991",
    "Arn": "arn:aws:sts::504649076991:assumed-role/AWSReservedSSO_AdministratorAccess_d0a7cfeb88c39771/Umar.satti"
}
```

Start the SSM port forwarding session using the following commands:

```
aws ssm start-session `
--target <Jenkins-EC2-Instance-ID> `
--document-name AWS-StartPortForwardingSession `
--parameters "portNumber=8080,localPortNumber=8080"
```

```
PS C:\Users\umars> aws ssm start-session `
>> --target i-01e3aab55fc39db10 `
>> --document-name AWS-StartPortForwardingSession `
>> --parameters "portNumber=8080,localPortNumber=8080"

Starting session with SessionId: Umar.satti-zyyhtcfv5y2us8t8ddc5k76uiy
Port 8080 opened for sessionId Umar.satti-zyyhtcfv5y2us8t8ddc5k76uiy.
Waiting for connections...

Connection accepted for session [Umar.satti-zyyhtcfv5y2us8t8ddc5k76uiy]
```

Open a browser on the Windows machine and navigate to local host URL on port 8080. It should display a Jenkins UI page. URL: ***http://localhost:8080***



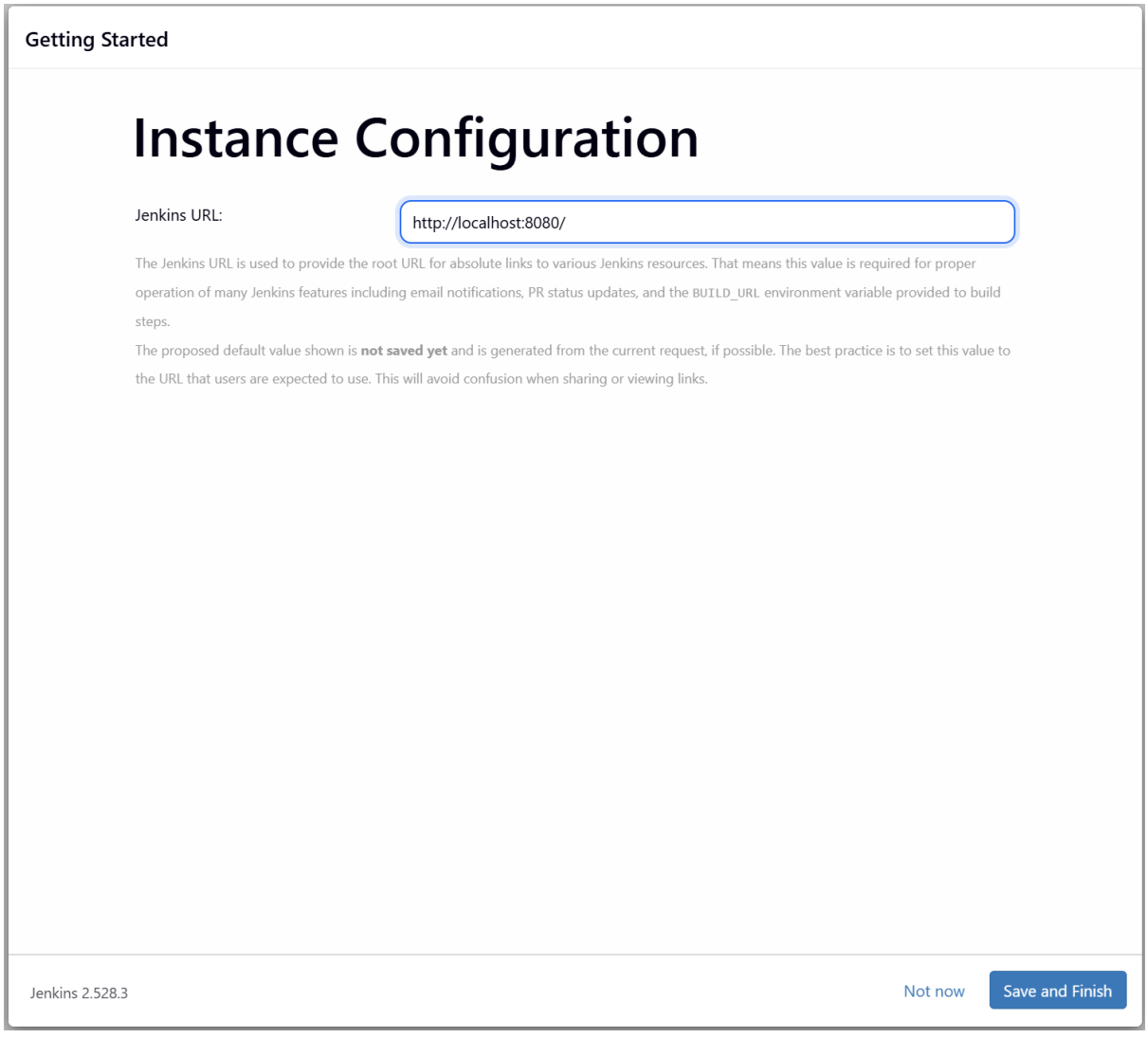
Once on the Jenkins **Unlock Page**, use the password stored inside the Jenkins EC2 instance to set up Jenkins. Use the following command to retrieve initial password:

- ***sudo cat /var/lib/jenkins/secrets/initialAdminPassword***
- **Note:** Run this command on Jenkins EC2 instance (jenkins-nodejs-server)

```
root@ip-10-0-2-29:~# cat /var/lib/jenkins/secrets/initialAdminPassword
315a1a1a89584e2790cf7e4651d5c257
root@ip-10-0-2-29:~#
```

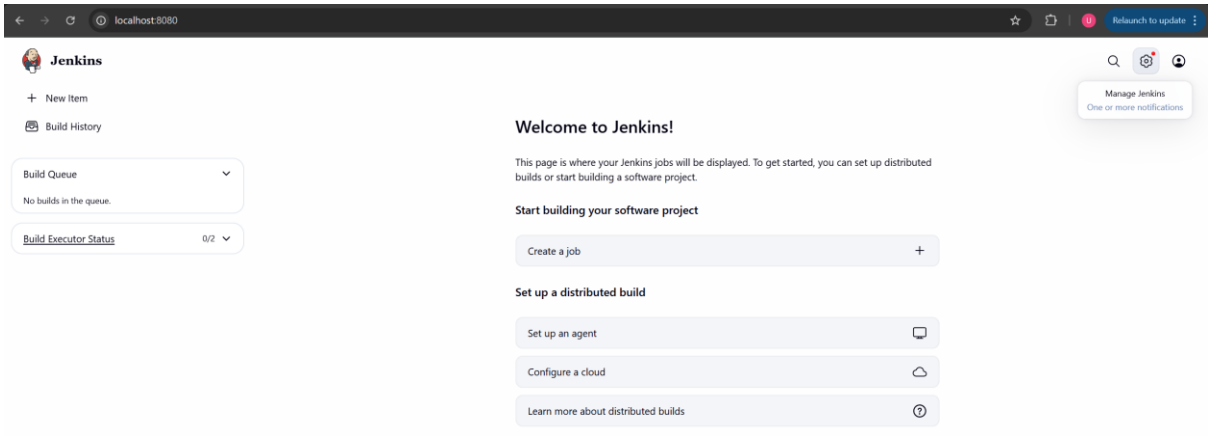
Perform the following actions after adding the initial password:

- Click **Install suggested plugins**.
- Create an admin account (username, password, name, email).
- Set Jenkins URL: ***http://localhost:8080/***
- Click **Starting using Jenkins.**

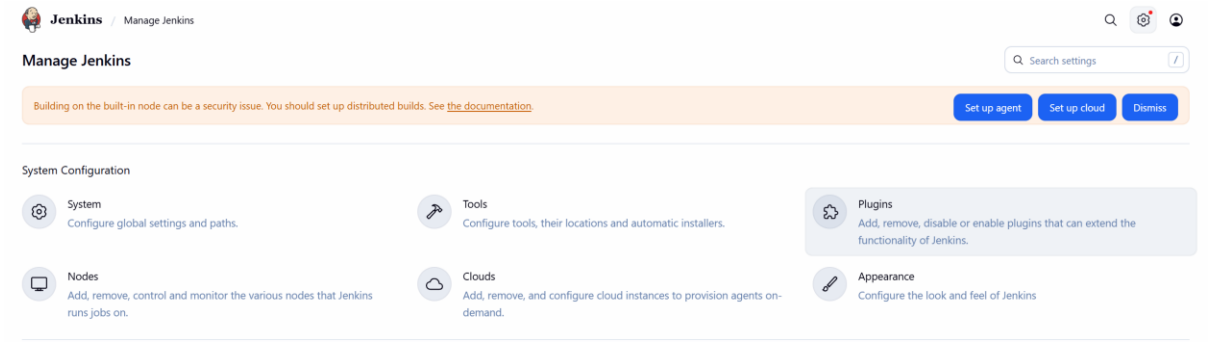


5.2 Install Plugins

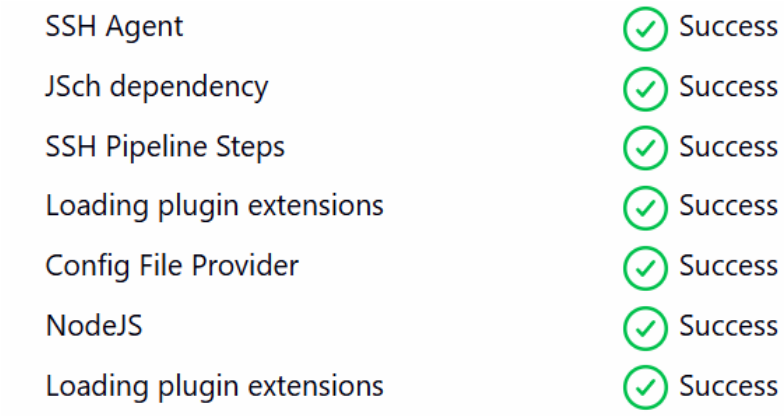
On the Jenkins Dashboard, select **Manage Jenkins** (settings icon) on the top right.



Under System Configuration, choose **Plugins**.

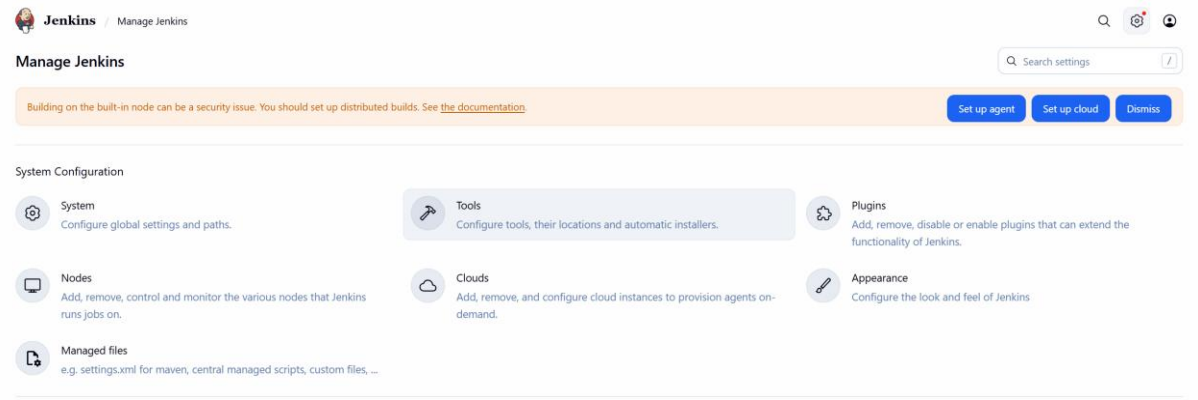


Click Available plugins. Search for NodeJS, SSH Agent, and SSH Pipeline Steps plugins. Select and download these three plugins.



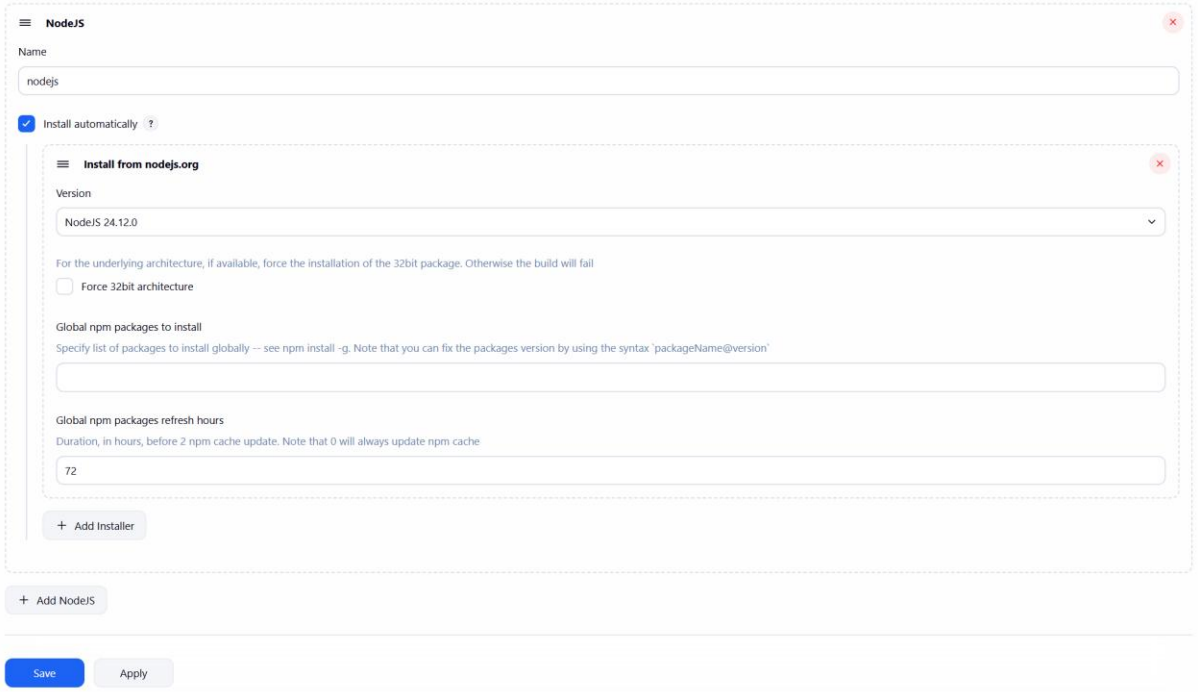
5.3 Install NodeJS Tool

On the Jenkins Dashboard, select Manage Jenkins (settings icon) on the top right. Under System Configuration, choose **Tools**.



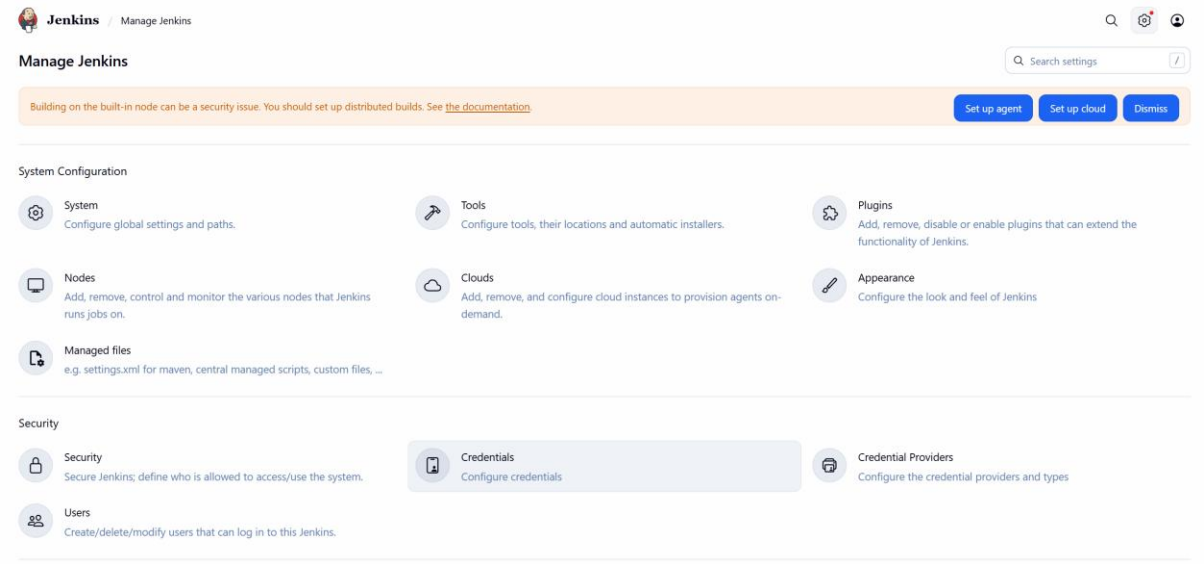
Scroll down to the end. It should display NodeJS installations tool. Click Add NodeJS button and add the following:

- **Name:** nodejs
- Select **Install automatically**
- **Version:** NodeJS 24.12.0
- Click **Apply** and **Save**

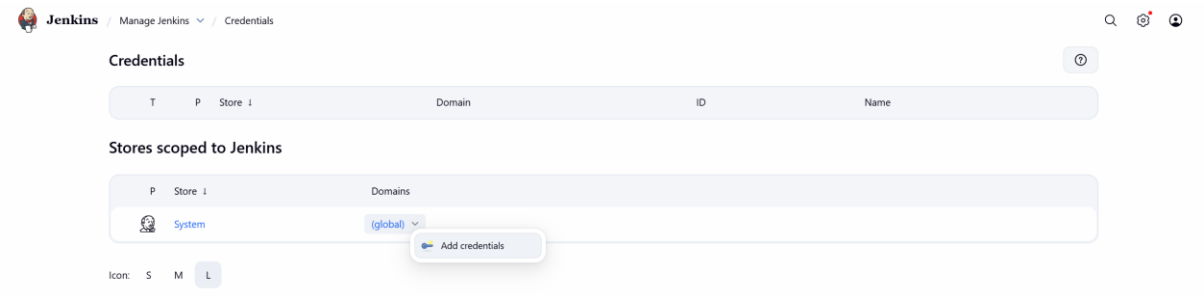


5.4: Add SSH Key Credentials

Go back to **Manage Jenkins** page (settings icon on the top right). Under Security section, choose **Credentials**.

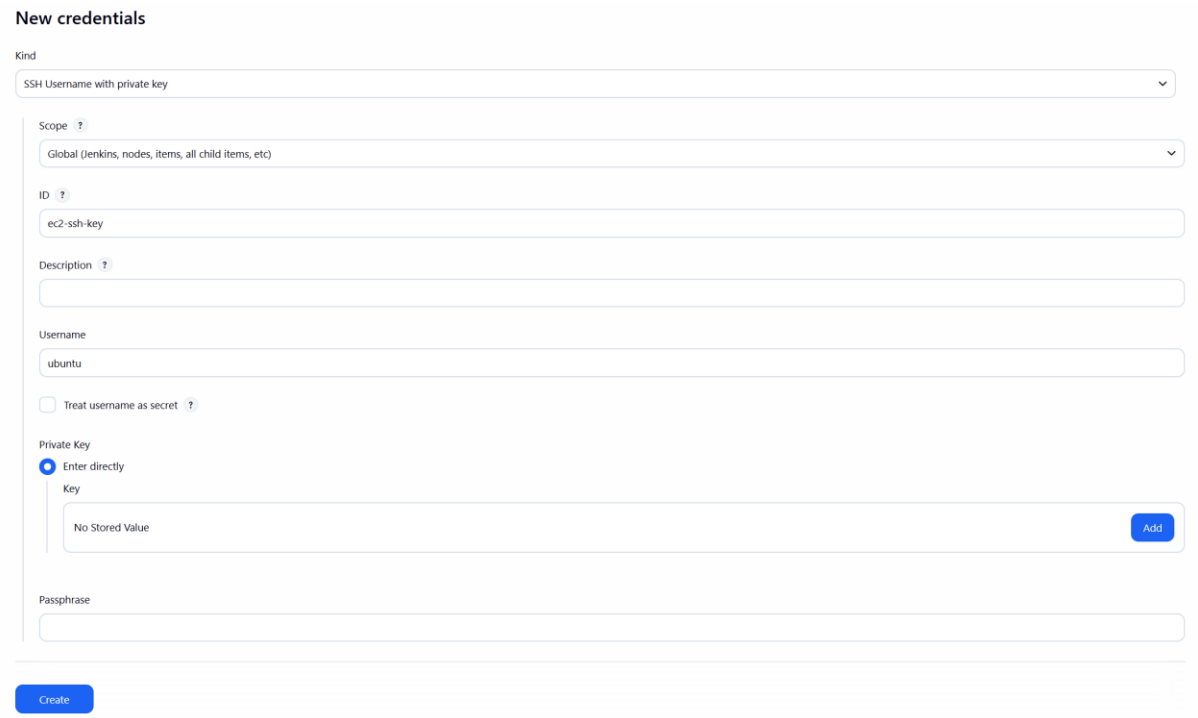


Under Stores scoped to Jenkins, it should display a text called global (highlighted in blue under Domains column). Toggle the drop-down and click **Add credentials**.

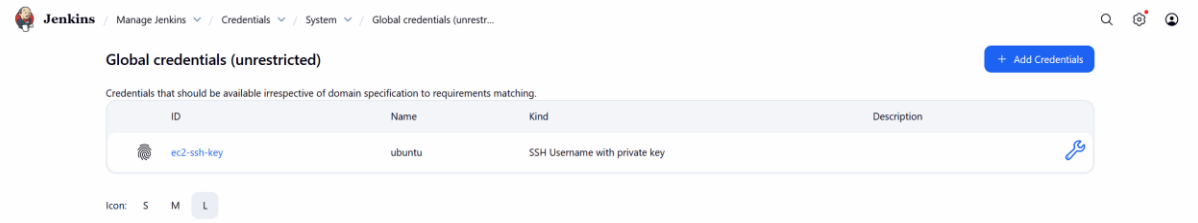


Add the credentials for SSH using the following configuration:

- **Kind:** SSH Username with private key
- **Scope:** Global
- **ID:** ec2-ssh-key
- **Description (optional):** Leave it blank
- **Username:** ubuntu
- Select the **Private Key button** (where it says Enter directly). Click the **Add** button and insert the SSH private key using the contents of the **.pem** file (EC2 key pair).
- Click **Create**

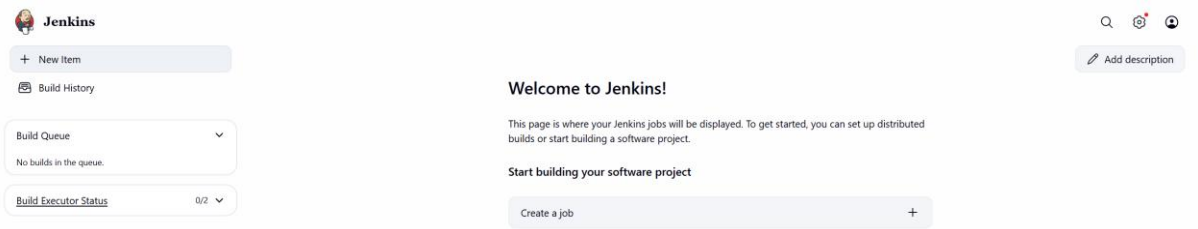


The following image shows the SSH key added to the Global credentials.



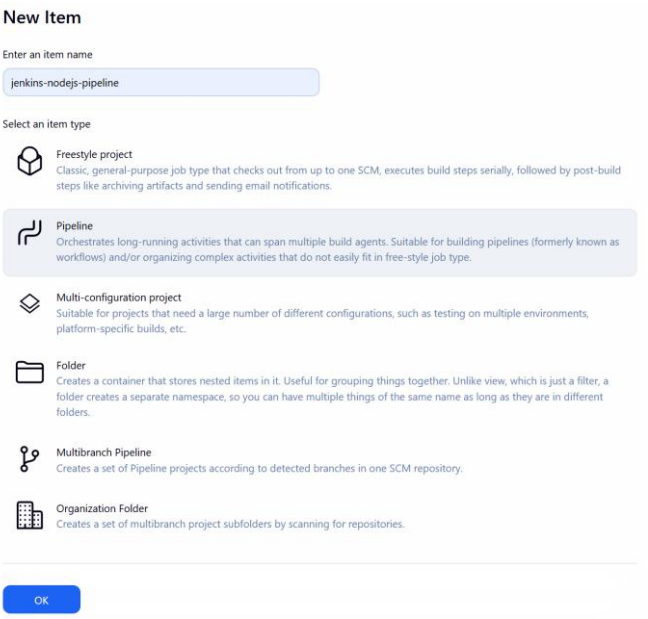
5.5 Pipeline Configuration

From the Jenkins Dashboard, click **New Item** located on the top right.



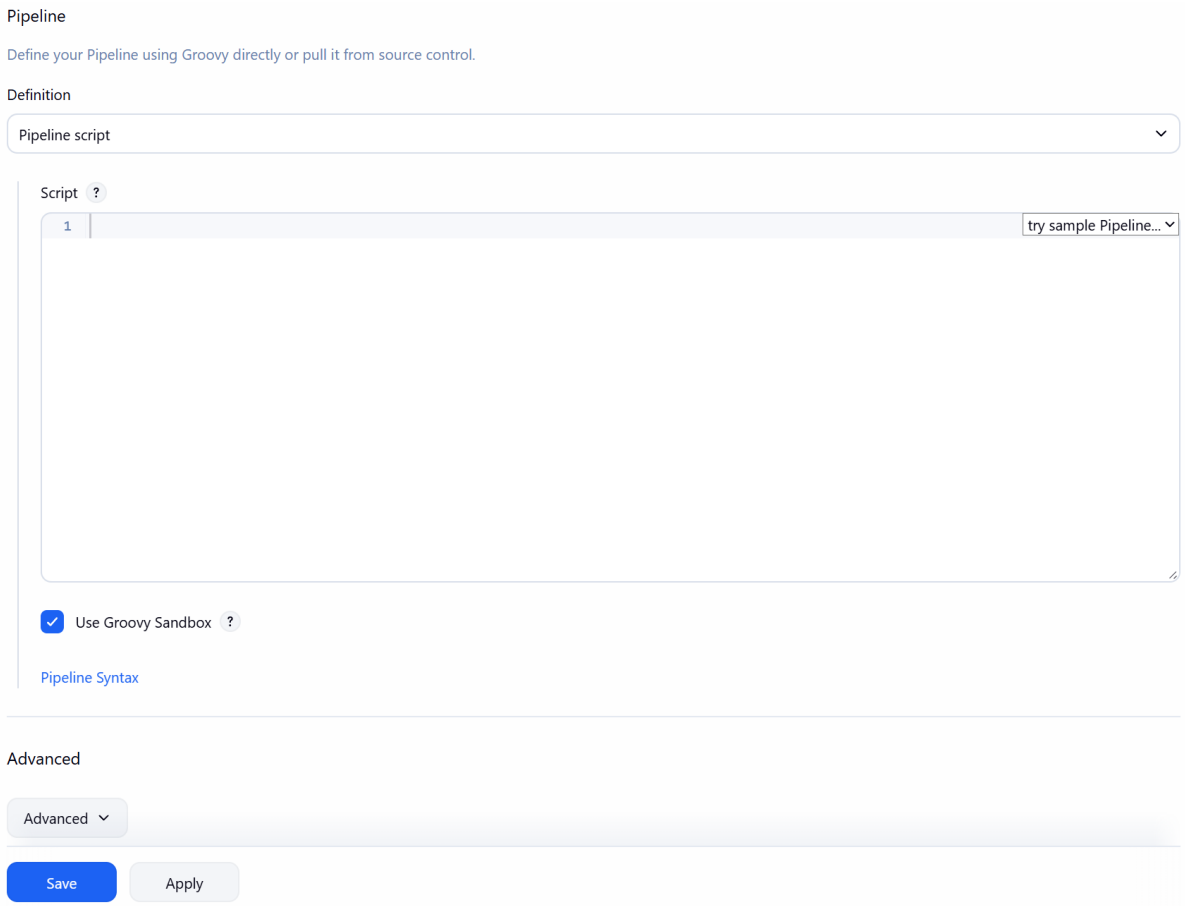
Create a New Item with the following configuration:

- **Item name:** jenkins-nodejs-pipeline
- **Item type:** Pipeline (2nd option)
- Click **OK**

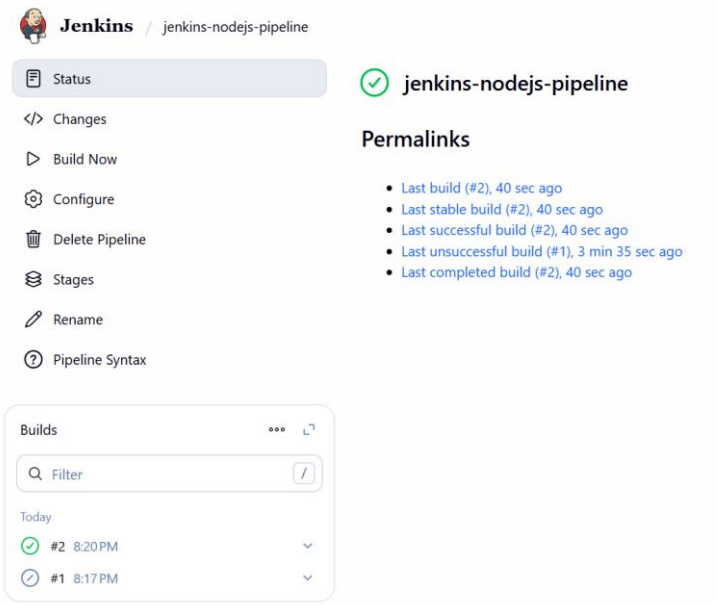


Scroll down to the Pipeline section. Perform the following configurations:

- **Definition:** Pipeline script
- Add the groovy script which performs checkout, build, test, and deploy stages
- Click **Apply** and **Save**
- **Note:** Script is stored in GitHub with the groovy file extension.



On the Pipeline page (Jenkins/Jenkins-nodejs-pipeline), run the Build job. Observe the Console Output for any failure or success. If issue occurs, debug it using the description provided in the Console Output.



5.6 Pipeline Execution and Console Output

Jenkins Pipeline Execution and Console Output

This section explains the Jenkins pipeline execution flow and the corresponding console output produced during a successful build, test, and deployment of the Node.js application to an EC2 instance.

Pipeline Initialization

The pipeline execution is manually triggered. Jenkins initializes the execution environment, including the configured Node.js tool, to ensure consistent runtime behavior throughout the pipeline.

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/jenkins-nodejs-pipeline
```

Source Code Checkout

In the Checkout stage, Jenkins retrieves the application source code from the GitHub repository and checks out the main branch.

- Connects to the GitHub repository.
- Fetches the latest code from the main branch.
- Checks out a specific commit for traceability.

```
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/jenkins-nodejs-pipeline/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/Umarsatti1/Task-14-Jenkins-Setup-using-Terraform-for-Nodejs-EC2-Deployment.git # timeout=10
Fetching upstream changes from https://github.com/Umarsatti1/Task-14-Jenkins-Setup-using-Terraform-for-Nodejs-EC2-Deployment.git
 > git --version # timeout=10
 > git --version # 'git version 2.43.0'
 > git fetch --tags --force --progress -- https://github.com/Umarsatti1/Task-14-Jenkins-Setup-using-Terraform-for-Nodejs-EC2-Deployment.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision bcfa2577dc9708dbd952d19f55c63af799041845 (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f bcfa2577dc9708dbd952d19f55c63af799041845 # timeout=10
 > git branch -a -v --no-abbrev # timeout=10
 > git branch -D main # timeout=10
 > git checkout -b main bcfa2577dc9708dbd952d19f55c63af799041845 # timeout=10
Commit message: "Update index.html"
 > git rev-list --no-walk bcfa2577dc9708dbd952d19f55c63af799041845 # timeout=10
```

Build Stage

The Build stage installs app dependencies and validates the runtime environment. Node.js and npm versions are confirmed before installing required packages.

- Verifies Node.js and npm versions.
- Installs dependencies using npm install.
- Confirms successful installation with no reported vulnerabilities.

```
[Pipeline] sh
+ node -v
v24.12.0
+ npm -v
11.6.2
+ npm install

up to date, audited 66 packages in 584ms

22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Test Stage

During the Test stage, automated tests defined in the project are executed. Successful test completion ensures the application is stable before deployment.

- Executes the test suite using npm test.
- Confirms all tests pass successfully.

```
[Pipeline] sh
+ npm test

> node-app@1.0.0 test
> echo "Tests passed!"

Tests passed!
```

Deployment to EC2

In the Deploy to EC2 stage, Jenkins uses secure SSH credentials to connect to the application EC2 instance and deploy the application using a blue/green-style deployment strategy. PM2 is used for application process management.

- Establishes SSH connection using Jenkins-managed credentials.
- Prepares a new release directory and copies application files.
- Switches between release directories to activate the new version.
- Installs production dependencies on the EC2 instance.
- Restarts or starts the application using PM2 and saves the process state.

```
[Pipeline] sshagent
[ssh-agent] Using credentials ubuntu
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-oNng9Ohnm8EY/agent.5190
SSH_AGENT_PID=5193
Running ssh-add (command line suppressed)
Identity added: /var/lib/jenkins/workspace/jenkins-nodejs-pipeline@tmp/private_key_13804105537811865009.key (/var/lib/jenkins/workspace/jenkins-nodejs-pipeline@tmp/private_key_13804105537811865009.key)
[ssh-agent] Started.
[Pipeline] {
[Pipeline] script
[Pipeline] {
[Pipeline] sh
+ ssh -o StrictHostKeyChecking=no ubuntu@10.0.1.231
                                sudo rm -rf /var/www/nodeapp_new
                                sudo mkdir -p /var/www/nodeapp_new
                                sudo chown ubuntu:ubuntu /var/www/nodeapp_new

Warning: Permanently added '10.0.1.231' (ED25519) to the list of known hosts.
```

```
[Pipeline] sh
+ scp -o StrictHostKeyChecking=no -r app.js package.json package-lock.json public ubuntu@10.0.1.231:/var/www/nodeapp_new/
[Pipeline] sh
+ ssh -o StrictHostKeyChecking=no ubuntu@10.0.1.231
                                    set -e

                                    echo "Switching releases..."

                                    sudo rm -rf /var/www/nodeapp_previous || true
                                    sudo mv /var/www/nodeapp_current /var/www/nodeapp_previous || true
                                    sudo mv /var/www/nodeapp_new /var/www/nodeapp_current

                                    sudo chown -R ubuntu:ubuntu /var/www/nodeapp_current

                                    cd /var/www/nodeapp_current
                                    npm install --omit=dev

                                    pm2 restart nodeapp || pm2 start app.js --name nodeapp
                                    pm2 save

Switching releases...

added 65 packages, and audited 66 packages in 1s

22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New major version of npm available! 10.8.2 -> 11.7.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.7.0
npm notice To update run: npm install -g npm@11.7.0
npm notice
```

```
[PM2] Spawning PM2 daemon with pm2_home=/home/ubuntu/.pm2
[PM2] PM2 Successfully daemonized
Use --update-env to update environment variables
[PM2][ERROR] Process or Namespace nodeapp not found
[PM2] Starting /var/www/nodeapp_current/app.js in fork_mode (1 instance)
[PM2] Done.
┌────┬─────────┬─────────────┬─────────┬─────────┬──────┬────────┬────┬────────┬──────┬────────┬──────────┬──────────┐
│ id │ name    │ namespace   │ version │ mode    │ pid  │ uptime │ ↺  │ status │ cpu  │ mem    │ user     │ watching │
├────┼─────────┼─────────────┼─────────┼─────────┼──────┼────────┼────┼────────┼──────┼────────┼──────────┼──────────┤
│ 0  │ nodeapp │ default     │ 1.0.0   │ fork    │ 3807 │ 0s     │ 0  │ online │ 0%   │ 34.8mb │ ubuntu   │ disabled │
└────┴─────────┴─────────────┴─────────┴─────────┴──────┴────────┴────┴────────┴──────┴────────┴──────────┴──────────┘
[PM2] Saving current process list...
[PM2] Successfully saved in /home/ubuntu/.pm2/dump.pm2
```

Health Check and Pipeline Completion

After deployment, Jenkins performs an automated health check to verify application availability. Upon successful validation, the pipeline completes and is marked as successful.

- Sends an HTTP request to the application health endpoint.
- Confirms a 200 OK response.
- Marks the pipeline execution as **SUCCESS**.

```
[Pipeline] sh
+ curl -s -o /dev/null -w %{http_code} http://10.0.1.231:3000/health
+ STATUS=200
+ [ 200 -ne 200 ]
+ echo Health check passed!
Health check passed!
```

5.7 Validation

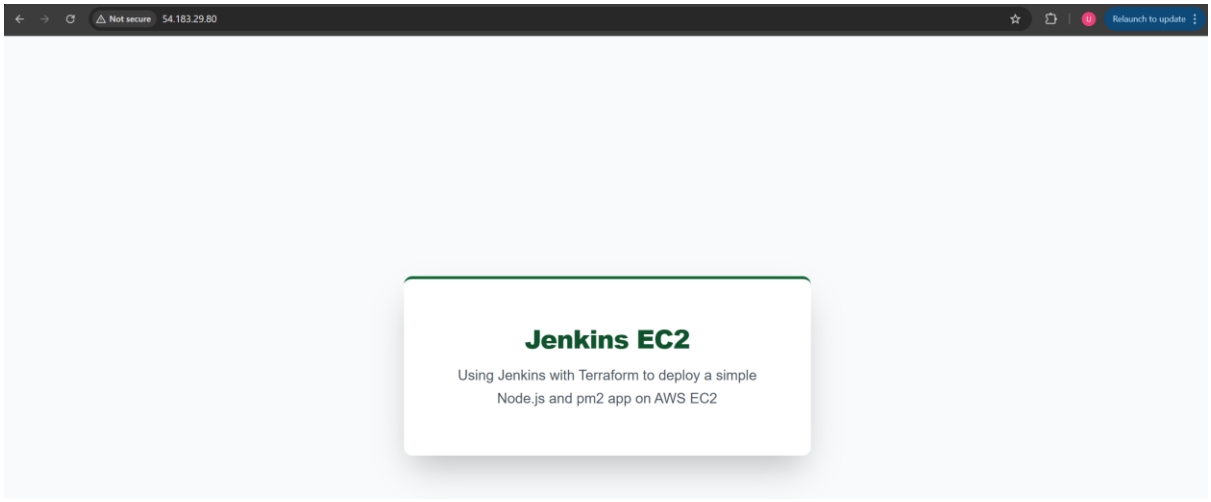
Navigate to the Pipeline page, select the pipeline named **jenkins-nodejs-pipeline**, and click console output. Ensure that each stage completes without errors.

```
$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 5193 killed;
[ssh-agent] Stopped.
[Pipeline] // sshagent
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Application Access:

Navigate to EC2 console. Select the nodejs-app-instance and copy the instance Public IPv4 address. Perform the following:

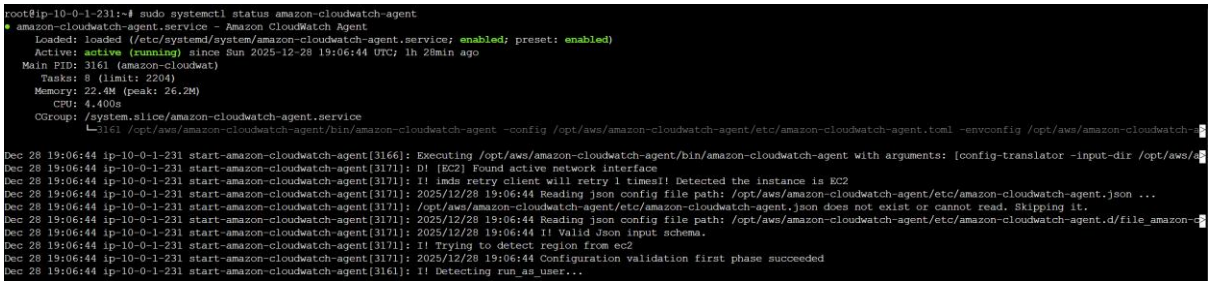
- Open this URL in a browser: http://<App-EC2-Public-IP>
  - ***http://54.183.29.80***
- Verify that the Node.js application loads successfully.



Application Health Monitoring

The Node.js application exposes a /health endpoint used for automated and manual health validation. Health checks are performed locally on the EC2 instance and externally via the public IP. Successful HTTP 200 responses confirm that the application, PM2 process manager, and Nginx reverse proxy are functioning correctly.

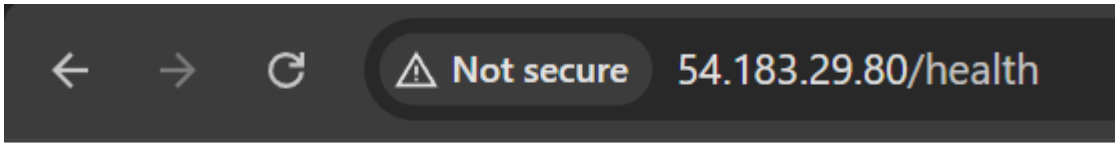
The Amazon CloudWatch Agent runs continuously on the application EC2 instance and is responsible for shipping logs and instance metrics to AWS CloudWatch. The agent service is enabled at boot and operates without errors, ensuring observability of application behavior.



- /health endpoint returns HTTP 200
- CloudWatch Agent is active and stable

- Confirms application availability post-deployment

```
root@ip-10-0-1-231:~# curl http://54.183.29.80/health
Health status: OKroot@curl http://localhost:3000/health
Health status: OKroot@ip-10-0-1-231:~#
```



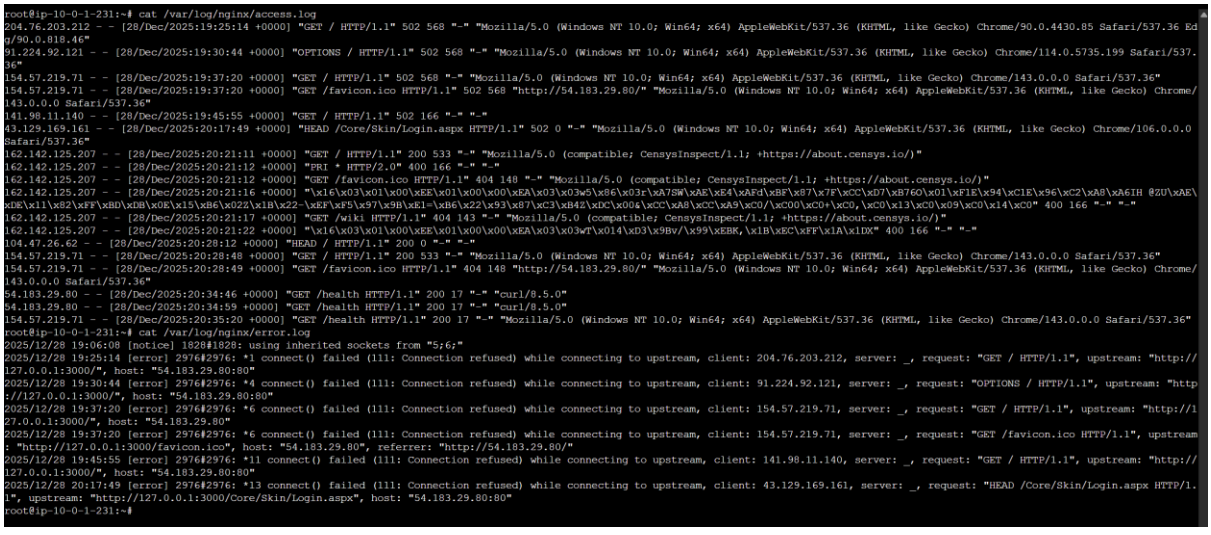
Health status: OK

Nginx Access and Error Logs

Nginx serves as a reverse proxy in front of the Node.js application. Access and error logs are stored under /var/log/nginx/. During application restarts or deployments, transient 502 Bad Gateway errors may appear when the upstream Node.js service is temporarily unavailable.

Once the application process is restarted via PM2, Nginx successfully routes traffic and returns HTTP 200 responses. Occasional scanner or bot traffic from external IPs is expected and does not indicate a misconfiguration.

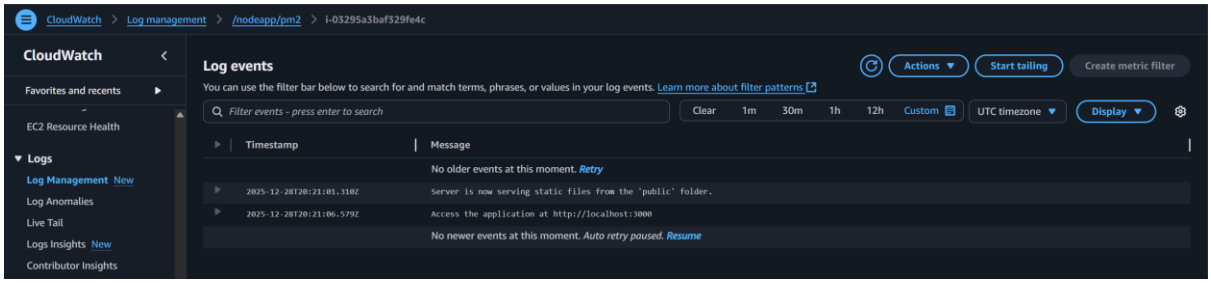
- 502 errors occurred when the application was not deployed
- Subsequent requests returned HTTP 200
- Reverse proxy behavior is correct and stable



```
root@ip-10-0-1-231:~# cat /var/log/nginx/access.log
204.76.203.212 - - [28/Dec/2025:19:25:14 +0000] "GET / HTTP/1.1" 502 568 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85 Safari/537.36 Edg/90.0.818.46"
91.224.92.121 - - [28/Dec/2025:19:30:44 +0000] "OPTIONS / HTTP/1.1" 502 568 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199 Safari/537.36"
154.57.219.71 - - [28/Dec/2025:19:37:20 +0000] "GET / HTTP/1.1" 502 568 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36"
154.57.219.71 - - [28/Dec/2025:19:37:20 +0000] "GET /favicon.ico HTTP/1.1" 502 568 "http://54.183.29.80/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36"
141.98.11.140 - - [28/Dec/2025:19:45:55 +0000] "GET / HTTP/1.1" 502 166 "-" "-"
43.129.169.161 - - [28/Dec/2025:20:17:49 +0000] "HEAD /Core/Skin/Login.aspx HTTP/1.1" 502 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36"
162.142.125.207 - - [28/Dec/2025:20:21:11 +0000] "GET / HTTP/1.1" 200 533 "-" "Mozilla/5.0 (compatible; CensysInspect/1.1; +https://about.censys.io/)"
162.142.125.207 - - [28/Dec/2025:20:21:12 +0000] "PRI * HTTP/2.0" 400 166 "-" "-"
162.142.125.207 - - [28/Dec/2025:20:21:12 +0000] "GET /favicon.ico HTTP/1.1" 404 148 "-" "Mozilla/5.0 (compatible; CensysInspect/1.1; +https://about.censys.io/)"
162.142.125.207 - - [28/Dec/2025:20:21:16 +0000] "\x16\x03\x01\x00\xEE\x01\x00\x00\xEA\x03\x03w5\x86\x03r\xA78W\xAE\xE4\xAFd\xBF\x87\x7F\xCC\xD7\xB760\x01\xF1E\x94\xC1E\x96\xC2\xA8\xA6IH @ZU\xAE\xDE\x11\x82\xFF\xBD\xDB\x0E\x15\xB6\x022\x1B\x22-\xEF\xF5\x97\x9B\xE1~\xB6\x22\x93\x87\xC3\xB42\xDC\x00&\xCC\xA8\xCC\xA9\xC0/\xC00\xCO+\xCO,\xC0\x13\xC0\x09\xC0\x14\xC0" 400 166 "-" "-"
162.142.125.207 - - [28/Dec/2025:20:21:17 +0000] "GET /wiki HTTP/1.1" 404 143 "-" "Mozilla/5.0 (compatible; CensysInspect/1.1; +https://about.censys.io/)"
162.142.125.207 - - [28/Dec/2025:20:21:22 +0000] "\x16\x03\x01\x00\xEE\x01\x00\x00\xEA\x03\x03wT\x014\xD3\x9Bv/\x99\xEBK,\x1B\xEC\xFF\x1A\x1DX" 400 166 "-" "-"
104.47.26.62 - - [28/Dec/2025:20:28:12 +0000] "HEAD / HTTP/1.1" 200 0 "-" "-"
154.57.219.71 - - [28/Dec/2025:20:28:48 +0000] "GET / HTTP/1.1" 200 533 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36"
154.57.219.71 - - [28/Dec/2025:20:28:49 +0000] "GET /favicon.ico HTTP/1.1" 404 148 "http://54.183.29.80/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36"
54.183.29.80 - - [28/Dec/2025:20:34:46 +0000] "GET /health HTTP/1.1" 200 17 "-" "curl/8.5.0"
54.183.29.80 - - [28/Dec/2025:20:34:59 +0000] "GET /health HTTP/1.1" 200 17 "-" "curl/8.5.0"
154.57.219.71 - - [28/Dec/2025:20:35:20 +0000] "GET /health HTTP/1.1" 200 17 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36"
root@ip-10-0-1-231:~# cat /var/log/nginx/error.log
2025/12/28 19:06:08 [notice] 1828#1828: using inherited sockets from "5;6;"
2025/12/28 19:25:14 [error] 2976#2976: *1 connect() failed (111: Connection refused) while connecting to upstream, client: 204.76.203.212, server: _, request: "GET / HTTP/1.1", upstream: "http://127.0.0.1:3000/", host: "54.183.29.80:80"
2025/12/28 19:30:44 [error] 2976#2976: *4 connect() failed (111: Connection refused) while connecting to upstream, client: 91.224.92.121, server: _, request: "OPTIONS / HTTP/1.1", upstream: "http://127.0.0.1:3000/", host: "54.183.29.80:80"
2025/12/28 19:37:20 [error] 2976#2976: *6 connect() failed (111: Connection refused) while connecting to upstream, client: 154.57.219.71, server: _, request: "GET / HTTP/1.1", upstream: "http://127.0.0.1:3000/", host: "54.183.29.80"
2025/12/28 19:37:20 [error] 2976#2976: *6 connect() failed (111: Connection refused) while connecting to upstream, client: 154.57.219.71, server: _, request: "GET /favicon.ico HTTP/1.1", upstream: "http://127.0.0.1:3000/favicon.ico", host: "54.183.29.80", referrer: "http://54.183.29.80/"
2025/12/28 19:45:55 [error] 2976#2976: *11 connect() failed (111: Connection refused) while connecting to upstream, client: 141.98.11.140, server: _, request: "GET / HTTP/1.1", upstream: "http://127.0.0.1:3000/", host: "54.183.29.80:80"
2025/12/28 20:17:49 [error] 2976#2976: *13 connect() failed (111: Connection refused) while connecting to upstream, client: 43.129.169.161, server: _, request: "HEAD /Core/Skin/Login.aspx HTTP/1.1", upstream: "http://127.0.0.1:3000/Core/Skin/Login.aspx", host: "54.183.29.80:80"
root@ip-10-0-1-231:~#
```

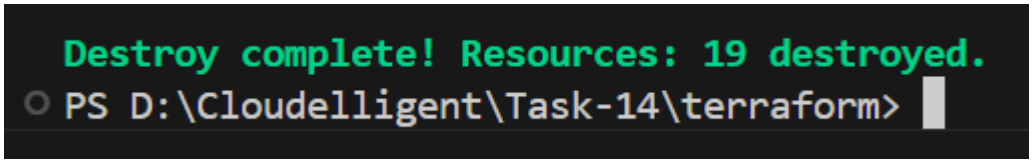
CloudWatch Application Logs (PM2)

PM2 application logs are collected from the .pm2/logs directory and streamed to AWS CloudWatch Logs. This enables centralized log access without requiring SSH access to the EC2 instance. Startup messages and runtime logs confirm successful application initialization and process management.



6. Clean Up

The ***terraform destroy -auto-approve*** command was executed to remove all AWS resources created by Terraform. This ensures that no infrastructure is left running, helping prevent unnecessary costs. The successful completion message confirms that all **19** resources were **destroyed**, as shown in the screenshot.



7. Troubleshooting

The following issues were encountered during the provisioning of infrastructure using Terraform and during the execution of the Jenkins CI/CD pipeline for Node.js EC2 deployment. Each issue outlines the observed problem, the identified root cause, and the solution that was applied.

Issue 1: EC2 Instance Loses Connectivity During Pipeline Execution

Problem
The application EC2 instance became unresponsive or lost connectivity while the Jenkins pipeline was running, causing the deployment to fail.

Root Cause
The pipeline included an npm build command which caused excessive resource consumption on the EC2 instance. Increasing the EC2 instance size did not fully mitigate the issue, indicating that the build process was unnecessary and unstable for the deployment workflow.

Solution
The npm build command was removed entirely from the Jenkins pipeline. Since the application did not require a build step for runtime execution, removing this command stabilized the EC2 instance and resolved the connectivity issue.

Issue 2: Node.js and PM2 Commands Not Found During Deployment

Problem
The Jenkins pipeline failed with the following errors during deployment:

npm: command not found

pm2: command not found

Root Cause
Node.js and PM2 were expected to be available on the Jenkins instance. However, Jenkins is only responsible for orchestrating the deployment and should not require Node.js tooling locally. These commands must be executed on the target EC2 instance instead.

Solution
The pipeline was corrected to ensure:

- Jenkins performs only build and orchestration tasks.

- All Node.js, npm, and PM2 commands are executed remotely on the application EC2 instance via SSH.

This separation of responsibilities resolved the command not found errors.

Issue 3: Malformed SSH Heredoc in Jenkins Pipeline

Problem
The Jenkins pipeline failed with ***exit code 127***, indicating that the EOF marker in an SSH heredoc was being interpreted as a command.

Root Cause
The heredoc syntax was incorrectly formatted:

- The EOF marker was indented.
- Bash could not correctly detect the end of the heredoc block. This caused Jenkins to misinterpret the script structure.

Solution
The heredoc was corrected by:

- Ensuring EOF started at column 1 with no indentation.
- Removing indentation inside the heredoc block.
- Adding set -e for immediate failure on errors.
- Using npm install --omit=dev to remove npm warnings.

This resolved the syntax error and allowed the SSH commands to execute correctly.

Issue 4: Incorrect Security Group Mapping in Terraform EC2 Module

Problem
Terraform failed during planning with the following error:

Incorrect attribute value type
Inappropriate value for attribute "vpc_security_group_ids": set of string required

Root Cause
The vpc_security_group_ids attribute expects a list of strings, but a single string value was passed from the module variable.

Solution
The configuration was updated to wrap the security group value in a list:

Changed

- ***each.value.security_group*** to ***[each.value.security_group]***

Updated the EC2 resource to:

- ***vpc_security_group_ids = [each.value.security_group]***

This resolved the type mismatch error.

Issue 5: SSH Key Pair Not Assigned to EC2 Instance

Problem
The application EC2 instance was launched without an SSH key pair, preventing SSH access after provisioning.

Root Cause
The key_name attribute was missing from the aws_instance resource block. As a result, Terraform did not pass the existing key pair to AWS during instance creation.

Solution
The EC2 resource was updated to explicitly map the key pair:

- Added key_name = each.value.key_name to the EC2 resource.
- Used conditional assignment in the locals map to assign the SSH key only to the application EC2 instance and null to the Jenkins instance.

This ensured secure SSH access while maintaining least-privilege configuration.