

Task 14

**Deploying Node.js
Application On EC2 Using
Terraform and Jenkins**

Umar Satti

Table of Contents

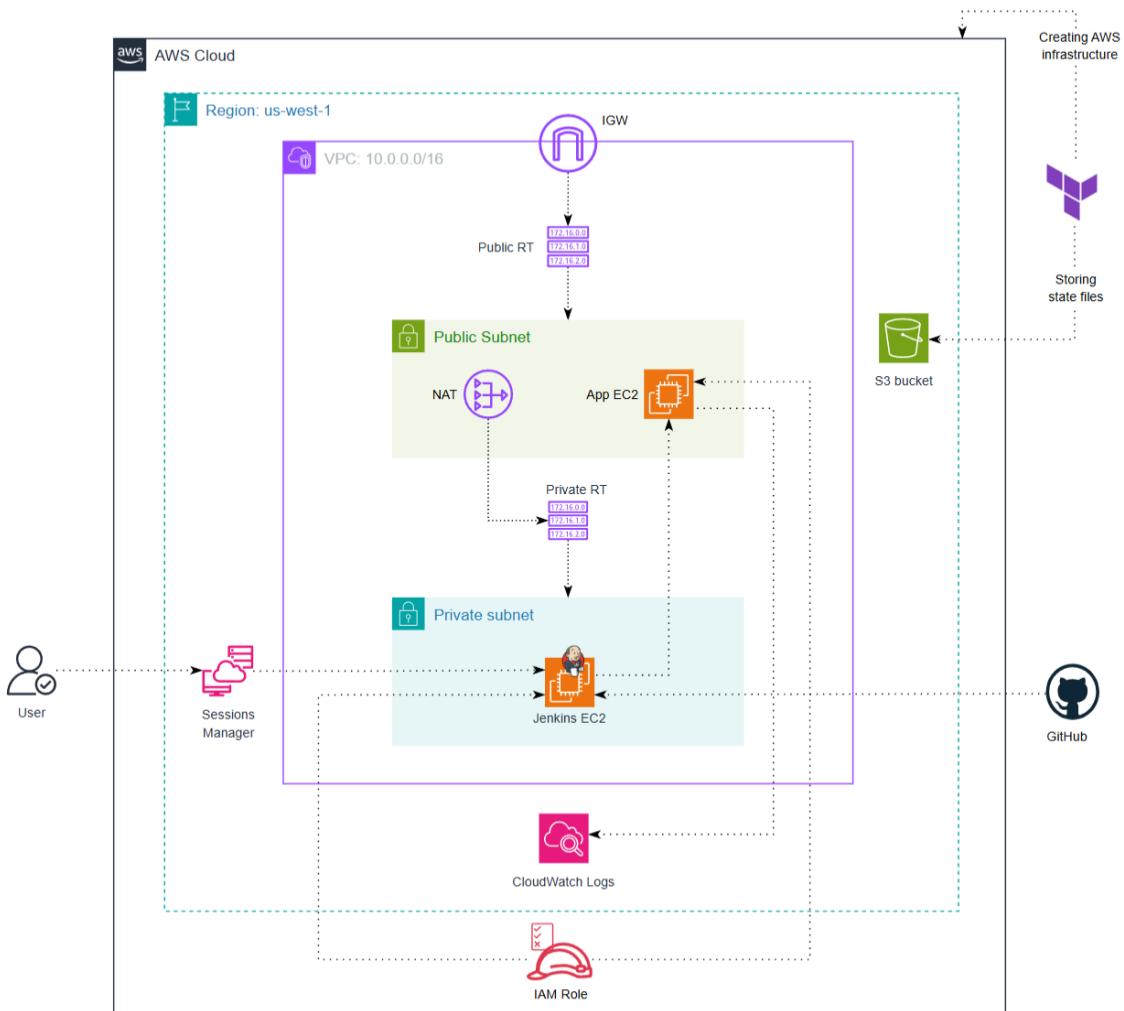
1. Task Description	3
2. Architecture Diagram	3
3. Project Structure.....	4
3.1 Application Files	4
3.2 Terraform Infrastructure Code.....	4
3.2.1 Root Terraform Files	4
3.2.2 Terraform Modules Directory	8
3.2.3 Supporting Terraform Files	11
3.3 Terraform Commands	13
4. Validate Infrastructure in AWS	15
4.1 VPC and Networking Validation.....	15
4.2 IAM Validation.....	18
4.3 EC2 Validation	19
4.4 CloudWatch Validation (Pre-Deployment)	21
4.4 End-to-End Validation (Pre-Deployment)	21
5. Jenkins Setup	22
5.1 Use Port Forwarding to Access Jenkins UI	22
5.2 Install Plugins	25
5.3 Install NodeJS Tool	26
5.4: Add SSH Key Credentials.....	27
5.5 Pipeline Configuration	28
5.6 Pipeline Execution and Console Output.....	30
5.7 Validation	33
6. Clean Up	37
7. Troubleshooting.....	38

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

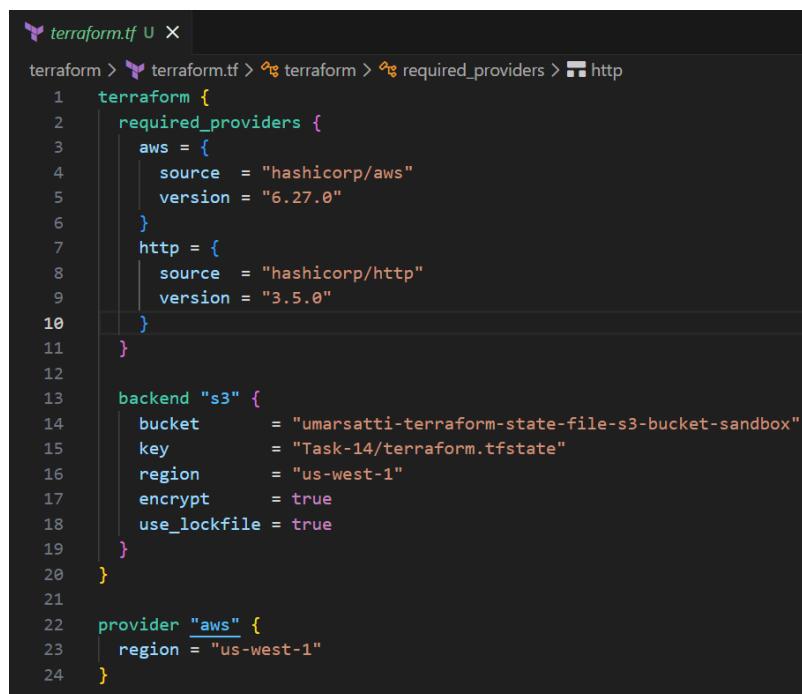
This directory contains Terraform configuration used to provision and manage all AWS infrastructure required for Jenkins, EC2, VPC, IAM, and CloudWatch. The setup follows a modular design to improve reusability and consistency.

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**.



```
terraform > terraform.tf > terraform > required_providers > http
1   terraform {
2     required_providers {
3       aws = {
4         source  = "hashicorp/aws"
5         version = "6.27.0"
6       }
7       http = {
8         source  = "hashicorp/http"
9         version = "3.5.0"
10    }
11  }
12
13  backend "s3" {
14    bucket      = "umarsatti-terraform-state-file-s3-bucket-sandbox"
15    key         = "Task-14/terraform.tfstate"
16    region      = "us-west-1"
17    encrypt     = true
18    use_lockfile = true
19  }
20}
21
22 provider "aws" {
23   region = "us-west-1"
24 }
```

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.

```

terraform > terraform.tfvars > ...
1   # VPC Vars
2   vpc_cidr          = "10.0.0.0/16"
3   vpc_name          = "umarsatti-vpc"
4   public_cidr       = "10.0.1.0/24"
5   private_cidr      = "10.0.2.0/24"
6   subnet_az         = "us-west-1a"
7   public_subnet_name = "public-subnet"
8   private_subnet_name = "private-subnet"
9   igw_name          = "umarsatti-igw"
10  eip_domain        = "vpc"
11  eip_name          = "umarsatti-nat-eip"
12  nat_name          = "umarsatti-nat-gw"
13  public_rt          = "umarsatti-public-rt"
14  private_rt         = "umarsatti-private-rt"
15  public_route       = "0.0.0.0/0"
16  fetch_ip          = "https://api.ipify.org"
17  app_sg_name        = "public-sg"
18  jenkins_sg_name    = "jenkins-sg"
19
20  # IAM Vars
21  ec2_role           = "ec2-instance-role-nodejs-jenkins"
22  ec2_policy          = "ec2-instance-policy-nodejs-jenkins"
23  instance_profile    = "instance-profile-nodejs-jenkins"
24
25  # EC2 Vars
26  ec2_app_name        = "nodejs-app-instance"
27  ec2_jenkins_name     = "jenkins-nodejs-server"
28  ami_id              = "ami-0e6a50b0059fd2cc3"
29  instance_type        = "t3.small"
30  instance_az          = "us-west-1a"
31  keypair              = "uts"
32  volume_size          = 15
33  volume_type          = "gp3"
34

```

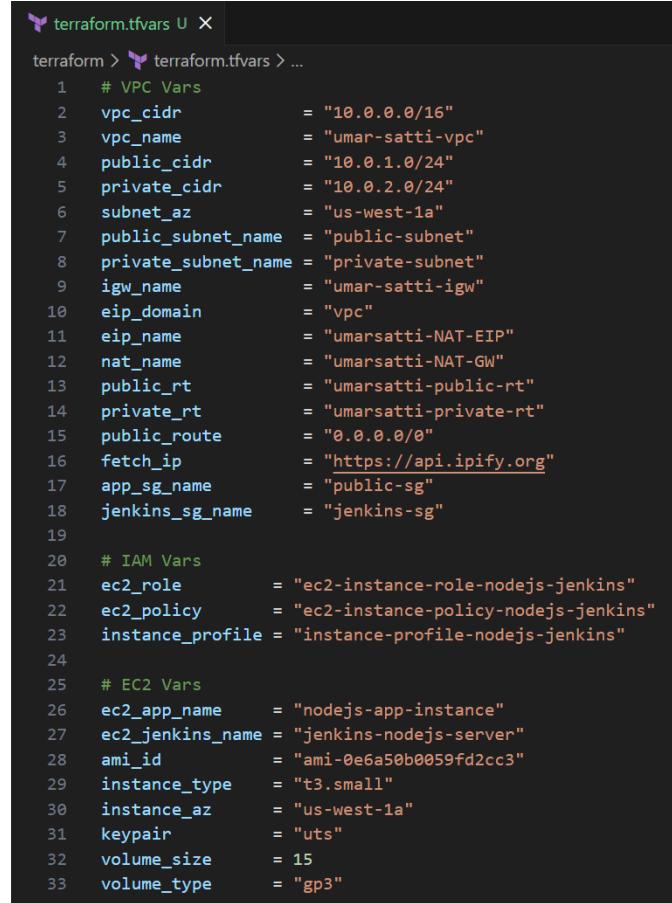
3. variables.tf

- 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.



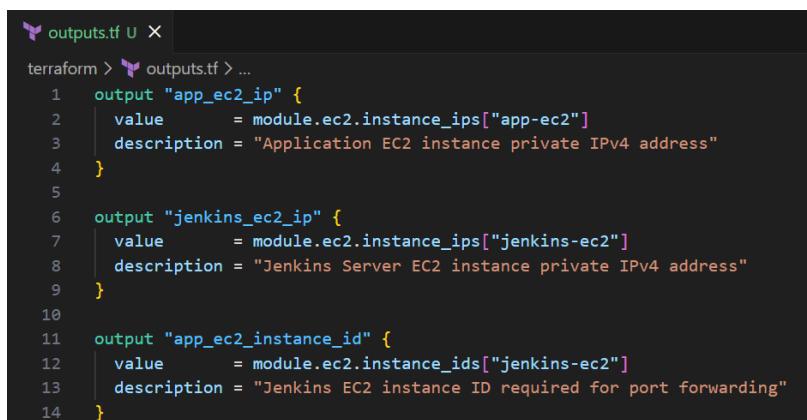
```

terraform > terraform.tfvars > ...
1  # VPC Vars
2  vpc_cidr      = "10.0.0.0/16"
3  vpc_name       = "umar-satti-vpc"
4  public_cidr    = "10.0.1.0/24"
5  private_cidr   = "10.0.2.0/24"
6  subnet_az     = "us-west-1a"
7  public_subnet_name = "public-subnet"
8  private_subnet_name = "private-subnet"
9  igw_name       = "umar-satti-igw"
10 eip_domain    = "vpc"
11 eip_name       = "umarsatti-NAT-EIP"
12 nat_name       = "umarsatti-NAT-GW"
13 public_rt      = "umarsatti-public-rt"
14 private_rt     = "umarsatti-private-rt"
15 public_route   = "0.0.0.0/0"
16 fetch_ip      = "https://api.ipify.org"
17 app_sg_name    = "public-sg"
18 jenkins_sg_name = "jenkins-sg"
19
20 # IAM Vars
21 ec2_role        = "ec2-instance-role-nodejs-jenkins"
22 ec2_policy      = "ec2-instance-policy-nodejs-jenkins"
23 instance_profile = "instance-profile-nodejs-jenkins"
24
25 # EC2 Vars
26 ec2_app_name    = "nodejs-app-instance"
27 ec2_jenkins_name = "jenkins-nodejs-server"
28 ami_id          = "ami-0e6a50b0059fd2cc3"
29 instance_type   = "t3.small"
30 instance_az     = "us-west-1a"
31 keypair         = "uts"
32 volume_size     = 15
33 volume_type     = "gp3"
...

```

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.



```

terraform > outputs.tf > ...
1  output "app_ec2_ip" {
2    value      = module.ec2.instance_ips["app-ec2"]
3    description = "Application EC2 instance private IPv4 address"
4  }
5
6  output "jenkins_ec2_ip" {
7    value      = module.ec2.instance_ips["jenkins-ec2"]
8    description = "Jenkins Server EC2 instance private IPv4 address"
9  }
10
11 output "app_ec2_instance_id" {
12   value      = module.ec2.instance_ids["jenkins-ec2"]
13   description = "Jenkins EC2 instance ID required for port forwarding"
14 }

```

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.

```
outputs.tf U
terraform > modules > vpc > outputs.tf > ...
1  output "vpc_id" {
2    description = "VPC ID"
3    value        = aws_vpc.vpc.id
4  }
5
6  output "public_subnet" {
7    description = "VPC Public Subnet ID"
8    value        = aws_subnet.public.id
9  }
10
11 output "private_subnet" {
12   description = "VPC Private subnet ID"
13   value        = aws_subnet.private.id
14 }
15
16 output "ec2_app_sg_id" {
17   description = "Application EC2 Security group ID"
18   value        = aws_security_group.ec2_app_sg.id
19 }
20
21 output "ec2_jenkins_sg_id" {
22   description = "Jenkins EC2 Security group ID"
23   value        = aws_security_group.ec2_jenkins_sg.id
24 }
25
```

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:

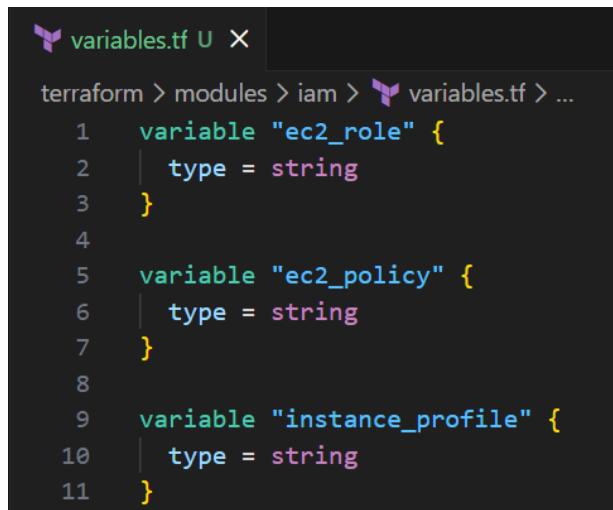
- **AmazonSSMManagedInstanceCore** (for Session Manager access)
- **CloudWatchAgentServerPolicy** (for CloudWatch agent)

```
main.tf  U X
terraform > modules > iam > main.tf > ...
1   # EC2 Trust relationship
2   data "aws_iam_policy_document" "ec2_assume_role" {
3     statement {
4       effect = "Allow"
5
6       principals {
7         type      = "Service"
8         identifiers = ["ec2.amazonaws.com"]
9       }
10
11      actions = ["sts:AssumeRole"]
12    }
13  }
14
15 # EC2 IAM Role
16 resource "aws_iam_role" "ec2_instance_role" {
17   name          = var.ec2_role
18   path          = "/"
19   assume_role_policy = data.aws_iam_policy_document.ec2_assume_role.json
20 }
21
22 # Attach AWS Managed Policies
23 resource "aws_iam_role_policy_attachment" "ssm_policy" {
24   role      = aws_iam_role.ec2_instance_role.name
25   policy_arn = "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
26 }
27
28 resource "aws_iam_role_policy_attachment" "cwagent_policy" {
29   role      = aws_iam_role.ec2_instance_role.name
30   policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
31 }
32
33 # Instance Profile
34 resource "aws_iam_instance_profile" "instance_profile" {
35   name = var.instance_profile
36   role = aws_iam_role.ec2_instance_role.name
37 }
```

iam/variables.tf

Declares input variables required to configure the IAM:

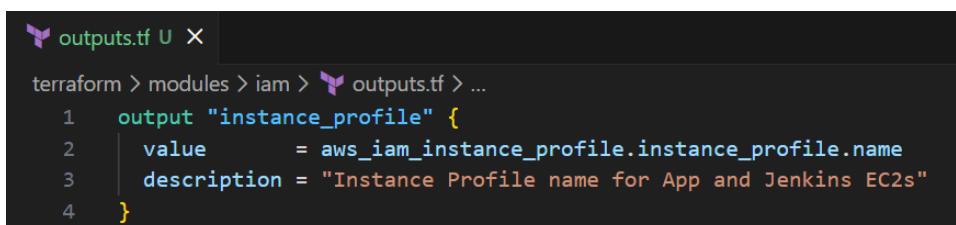
- EC2 role and policy names
- Instance profile name



```
variables.tf
variable "ec2_role" {
  type = string
}
variable "ec2_policy" {
  type = string
}
variable "instance_profile" {
  type = string
}
```

iam/outputs.tf

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



```
outputs.tf
output "instance_profile" {
  value     = aws_iam_instance_profile.instance_profile.name
  description = "Instance Profile name for App and Jenkins EC2s"
}
```

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:

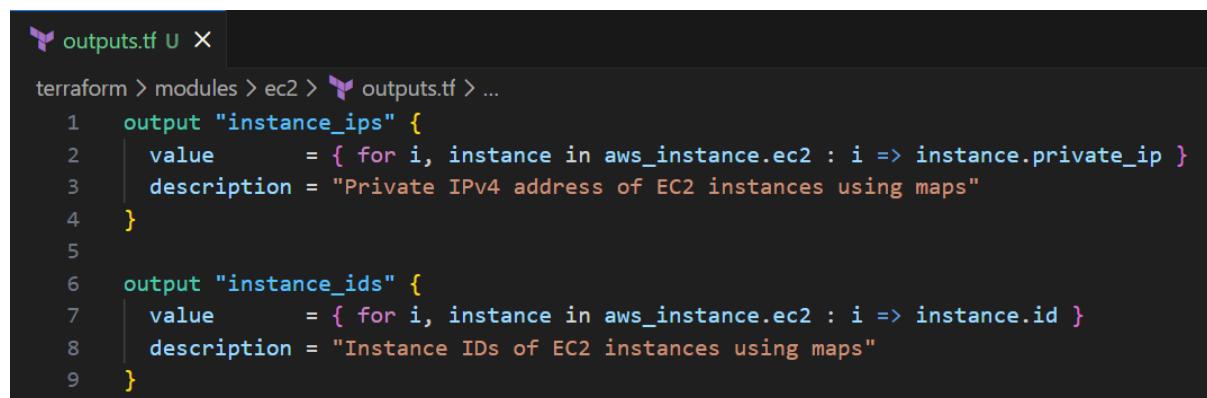
- EC2 names and AMI IDs.
- Instance IDs and Availability zones.
- EBS volume type and size.

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
terraform > modules > ec2 > outputs.tf > ...
1   output "instance_ips" {
2     value      = { for i, instance in aws_instance.ec2 : i => instance.private_ip }
3     description = "Private IPv4 address of EC2 instances using maps"
4   }
5
6   output "instance_ids" {
7     value      = { for i, instance in aws_instance.ec2 : i => instance.id }
8     description = "Instance IDs of EC2 instances using maps"
9 }
```

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.

```
$ jenkins_ec2.sh U X
terraform > $ jenkins_ec2.sh
1  #!/bin/bash
2  set -e
3
4  echo "Waiting for internet..."
5  until ping -c 1 google.com >/dev/null 2>&1; do sleep 2; done
6
7  apt update -y
8  apt install -y fontconfig openjdk-21-jre curl
9
10 # -----
11 # Jenkins installation
12 # -----
13 mkdir -p /etc/apt/keyrings
14
15 curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key \
| -o /etc/apt/keyrings/jenkins-keyring.asc
16
17 echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/" \
> /etc/apt/sources.list.d/jenkins.list
18
19 apt update -y
20 apt install -y jenkins
21
22 systemctl enable jenkins
23 systemctl start jenkins
```

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.

```
● PS D:\Cloudelligent\Task-14\terraform> terraform init
  Initializing the backend...
  Initializing modules...
  Initializing provider plugins...
    - Reusing previous version of hashicorp/http from the dependency lock file
    - Reusing previous version of hashicorp/aws from the dependency lock file
    - Using previously-installed hashicorp/http v3.5.0
    - Using previously-installed hashicorp/aws v6.27.0

  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.
○ PS D:\Cloudelligent\Task-14\terraform>
```

2. Terraform validate

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

```
PS D:\Cloudelligent\Task-14\terraform> terraform validate
Success! The configuration is valid.

○ PS D:\Cloudelligent\Task-14\terraform>
```

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.

```
Plan: 19 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + app_ec2_ip          = (known after apply)
  + jenkins_ec2_instance_id = (known after apply)
  + jenkins_ec2_ip        = (known after apply)
```

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.

```
Apply complete! Resources: 19 added, 0 changed, 0 destroyed.
```

Outputs:

```
app_ec2_ip = "10.0.1.231"
jenkins_ec2_instance_id = "i-01e3aab55fc39db10"
jenkins_ec2_ip = "10.0.2.29"
```

```
PS D:\Cloudelligent\Task-14\terraform> █
```

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

The screenshot shows the AWS VPC console interface. On the left, there's a navigation sidebar with options like VPC dashboard, AWS Global View, Filter by VPC, Virtual private cloud (with subnets, route tables, internet gateways), Security (Network ACLs, Security groups), PrivateLink and Lattice (Getting started, Endpoints, Endpoint services, Service networks), and a general security section. The main area is titled 'Your VPCs' and shows a table with one item: 'vpc-0ce644f614bc98d03 / umarsatti-vpc'. The table columns include VPC ID, Name, State, Encryption controls, Block Public Access, and IPv4 CIDR. Below this table, there's a detailed view for the selected VPC, showing sections for Details, Resource map, CIDRs, Flow logs, Tags, and Integrations. The 'Details' section contains various configuration parameters such as VPC ID, State (Available), Block Public Access (Off), and more.

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

Subnets (2) Info							
Find subnets by attribute or tag		Actions					
VPC : vpc-0ce644f614bc98d03		Create subnet					
Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR		
uts-public-subnet	subnet-0cc8976e038c9543b	Available	vpc-0ce644f614bc98d03 umarsatti-vpc	Off	10.0.1.0/24		
uts-private-subnet	subnet-0874e7c94a0660ddd	Available	vpc-0ce644f614bc98d03 umarsatti-vpc	Off	10.0.2.0/24		

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

Internet gateways (1/1) Info							
Find internet gateways by attribute or tag		Actions					
VPC ID : vpc-0ce644f614bc98d03		Create internet gateway					
Name	Internet gateway ID	State	VPC ID	Owner			
umarsatti-igw	igw-0949cf7701b06327b	Attached	vpc-0ce644f614bc98d03 umarsatti-vpc	504649076991			

igw-0949cf7701b06327b / umarsatti-igw	
Details	Tags
Details Internet gateway ID: igw-0949cf7701b06327b Attached	VPC ID: vpc-0ce644f614bc98d03 umarsatti-vpc Owner: 504649076991

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

NAT gateways (1/1) Info							
Find NAT gateways by attribute or tag		Actions					
VPC : vpc-0ce644f614bc98d03		Create NAT gateway					
Name	NAT gateway ID	Connectivity type	State	State message	Availability mode	Route table ID	Primary public IPv4 address
umarsatti-nat-gw	nat-0c96ba894ab98ba7d	Public	Available	-	Zonal	-	184.72.60.205

nat-0c96ba894ab98ba7d / umarsatti-nat-gw	
Details	Secondary IP4 addresses
Details NAT gateway ID: nat-0c96ba894ab98ba7d NAT gateway ARN: arn:aws:ec2:us-west-1:504649076991:natgateway/nat-0c96ba894ab98ba7d VPC: vpc-0ce644f614bc98d03 umarsatti-vpc	Connectivity type : Public Primary public IPv4 address : 184.72.60.205 Subnet : subnet-0cc8976e038c9543b / uts-public-subnet State : Available Primary private IPv4 address : 10.0.1.59 Created : Monday, December 29, 2025 at 00:05:19 GM T+5 State message : - Primary network interface ID : eni-0717b6cbf86fb715 Deleted : -

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

Name	Route table ID	Explicit subnet associations	Main	VPC
umarsatti-public-rt	rtb-013ff80de2f4ea836	subnet-0cc8976e038c9543b / ...	No	vpc-0ce644f614bc98d03 umarsatti-vpc
umarsatti-private-rt	rtb-0375d399c6ed069b5	subnet-0874e7c94a0660ddd / ...	No	vpc-0ce644f614bc98d03 umarsatti-vpc
-	rtb-0105792fca08ea0f4	-	Yes	vpc-0ce644f614bc98d03 umarsatti-vpc

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

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

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
uts-public-subnet	subnet-0cc8976e038c9543b	10.0.1.0/24	-

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

Route table ID rtb-013ff80de2f4ea836	Main No	Explicit subnet associations subnet-0cc8976e038c9543b / uts-public-subnet	Edge associations -
VPC vpc-0ce644f614bc98d03 umarsatti-vpc	Owner ID 504649076991		

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

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

Route table ID rtb-0375d399c6ed069b5	Main No	Explicit subnet associations subnet-0874e7c94a0660ddd / uts-private-subnet	Edge associations -
VPC vpc-0ce644f614bc98d03 umarsatti-vpc	Owner ID 504649076991		

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

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	nat-0c96ba894ab98ba7d	Active	No	Create Route
10.0.0.0/16	local	Active	No	Create Route Table

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:

Name	Security group ID	Security group name	VPC ID	Description	Owner
jenkins-sg	sg-070c8d5b99e0f01a	jenkins-sg	vpc-0ce644f614bc98d03	Allows port 22 (SSH) and port 8080 traffic fr...	504649076991
-	sg-064e74458c00ef7a	default	vpc-0ce644f614bc98d03	default VPC security group	504649076991
public-sg	sg-04df5751f1fb30d03	public-sg	vpc-0ce644f614bc98d03	Inbound traffic on port 22, 80, and 3000.	504649076991

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.

Summary

Creation date: December 29, 2025, 00:04 (UTC+05:00)

Last activity: 9 minutes ago

ARN: arn:aws:iam::504649076991:role/ec2-instance-role-nodejs-jenkins

Instance profile ARN: arn:aws:iam::504649076991:instance-profile/instance-profile-nodejs-jenkins

Maximum session duration: 1 hour

Permissions

Permissions policies (2)

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
AmazonSSMManagedInstanceCore	AWS managed	21
ec2-instance-policy-nodejs-jenkins	Customer managed	1

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.

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like Dashboard, AWS Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager, Images, Elastic Block Store, Network & Security, and Load Balancing. The main area displays a table of instances. The first row has a checkbox, Name (nodejs-app-instance), Instance ID (i-03295a3baf329fe4c), Instance state (Running), Instance type (t3.small), Status check (3/3 checks passed), Alarm status (View alarms), Availability Zone (us-west-1a), Public IPv4 DNS (ec2-54-183-29-80.us-west-1.compute.amazonaws.com), and Public IPv4 (54.183.29.80). The second row has a checkbox, Name (jenkins-nodejs-server), Instance ID (i-01e3aab55fc39db10), Instance state (Running), Instance type (t3.small), Status check (3/3 checks passed), Alarm status (View alarms), Availability Zone (us-west-1a), Public IPv4 DNS (empty), and Public IPv4 (empty). At the bottom of the table, there are buttons for Connect, Instance state, Actions, and Launch instances.

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.

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like Dashboard, AWS Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces, and Load Balancing.

The main area displays 'Instances (1/2) Info'. It lists two instances:

- nodejs-app-instance**: Instance ID i-03295a3ba5329fe4c, Status Running, Instance type t3.small, Status check 3/3 checks passed, View alarms, Availability Zone us-west-1a, Public IPv4 DNS ec2-54-183-29-80.us-west-1.amazonaws.com, Public IPv4 54.183.29.80, Elastic IP -.
- jenkins-nodejs-server**: Instance ID i-01e3aab55fc39db10, Status Running, Instance type t3.small, Status check 3/3 checks passed, View alarms, Availability Zone us-west-1a, Public IPv4 DNS ec2-54-183-29-80.us-west-1.amazonaws.com, Public IPv4 54.183.29.80, Elastic IP -.

Below the instance list, there's a detailed view for the jenkins-nodejs-server instance. The 'Details' tab is selected, showing the following configuration:

- Instance ID:** i-01e3aab55fc39db10
- IPv4 address:** -
- Hostname type:** IP name: ip-10-0-2-29.us-west-1.compute.internal
- Answer private resource DNS name:** -
- Auto-assigned IP address:** -
- IAM Role:** e2-instance-role-nodejs-jenkins
- IMDv2:** Required
- Public IPv4 address:** -
- Private IP4 DNS name (IPv4 only):** ip-10-0-2-29.us-west-1.compute.internal
- Instance type:** t3.small
- VPC ID:** vpc-0ce644f614bc98d03 (umarsatti-vpc)
- Subnet ID:** subnet-0874e7c94a0660ddd (uts-private-subnet)
- Instance ARN:** arn:aws:ec2:us-west-1:504649076991:instance/i-01e3aab55fc39db10
- Public DNS:** -
- Elastic IP addresses:** -
- AWS Compute Optimizer finding:** No recommendations available for this instance.
- Auto Scaling Group name:** -
- Managed:** false

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"
               └─2978 "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**

```
# java -version
openjdk version "21.0.9" 2025-10-21
openjdk Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
  Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
  Active: active (running) since Sun 2025-12-28 19:08:10 UTC; 26min ago
    Main PID: 3896 (java)
      Tasks: 43 (limit: 2204)
        Memory: 525.3M (peak: 541.4M)
        CPU: 27.368s
       CGroup: /system.slice/jenkins.service
               └─3896 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF] > This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF]> ****
Dec 28 19:08:10 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:08:10.551+0000 [id=37]     INFO    jenkins.InitReactorRunner$1#onAttained: Completed initialization
Dec 28 19:08:10 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:08:10.573+0000 [id=30]     INFO    hudson.lifecycle.Lifecycle.onReady: Jenkins is fully up and running
Dec 28 19:08:10 ip-10-0-2-29 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Dec 28 19:08:11 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:08:11.103+0000 [id=56]     INFO    h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.X
Dec 28 19:08:11 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:08:11.104+0000 [id=56]     INFO    hudson.util.Retrive#start: Performed the action check updates server successfully at the atten
lines 1-20/29 (END)
```

4.4 CloudWatch Validation (Pre-Deployment)

The screenshot shows the AWS CloudWatch Log Management interface. On the left, there's a sidebar with navigation links like 'CloudWatch', 'Favorites and recents', 'Trace Map', 'Application Insights', 'Infrastructure Monitoring' (with sub-options 'Container Insights', 'Database Insights', 'Lambda Insights', 'EC2 Resource Health'), and 'Logs' (with sub-option 'Log Management'). The main area is titled 'Log groups (2/4)' and shows a table of log groups. The table has columns for 'Log group' (with checkboxes), 'Log class', 'Anomaly d...', 'Deletion pr...', 'Data pro...', 'Sensitive...', and 'Ret'. There are four log groups listed: '/aws/ecs/containerinsights/nodejs-cluster/performance' (selected), '/aws/lambda/delete-name-tags-us-west-1-a9d5-mfo9', 'CodeBuild-EB-Nodejs', and 'ecs-jenkins-nodejs-logs' (selected). A search bar at the top says 'Filter log groups or try pattern search'.

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.

```
root@ip-10-0-2-29:~# sudo su -  
root@ip-10-0-2-29:~# systemctl status jenkins  
● jenkins.service - Jenkins Continuous Integration Server  
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)  
     Active: active (running) since Sun 2025-12-28 19:08:10 UTC; 3min ago  
       Main PID: 3896 (java)  
         Tasks: 43 (limit: 2204)  
        Memory: 525.3M (peak: 541.4M)  
          CPU: 28.227s  
        CGroup: /system.slice/jenkins.service  
               └─3896 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080  
  
Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF] > This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword  
Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF]  
Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF] ****  
Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF] > ****  
Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF] > ****  
Dec 28 19:08:05 ip-10-0-2-29 jenkins[3896]: [LF] > ****  
Dec 28 19:08:10 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:09:10.551+0000 [id=37]      INFO    jenkins.InitReactorRunner$1@onAttained: Completed initialization  
Dec 28 19:08:10 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:09:10.573+0000 [id=30]      INFO    hudson.lifecycle.Lifecycle$OnReady: Jenkins is fully up and running  
Dec 28 19:08:10 ip-10-0-2-29 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.  
Dec 28 19:08:11 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:09:11.103+0000 [id=56]      INFO    h.m.DownloadService$Downloadable$load: Obtained the updated data file for hudson.tasks.Maven.M  
Dec 28 19:08:11 ip-10-0-2-29 jenkins[3896]: 2025-12-28 19:09:11.104+0000 [id=56]      INFO    hudson.util.Retrier#start: Performed the action check updates server successfully at the attempt 1  
root@ip-10-0-2-29:~# curl http://localhost:8080  
<html><head><meta http-equiv="refresh" content="1;url=/login?from=%2F"/><script id='redirect' data-redirect-url='/_login?from=%2F' src='/static/dsa765a3/scripts/redirect.js'></script></head><body>  
Authentication required  
<!--  
>  
</body></html>  
root@ip-10-0-2-29:~#
```

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": "AROAXX73N2D7R4SMA6NA4:Umar.satti",
    "Account": "504649076991",
    "Arn": "arn:aws:sts::504649076991:assumed-role/AWSReservedSSO_AdministratorAccess_d0a7cfb88c39771/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"

```

```

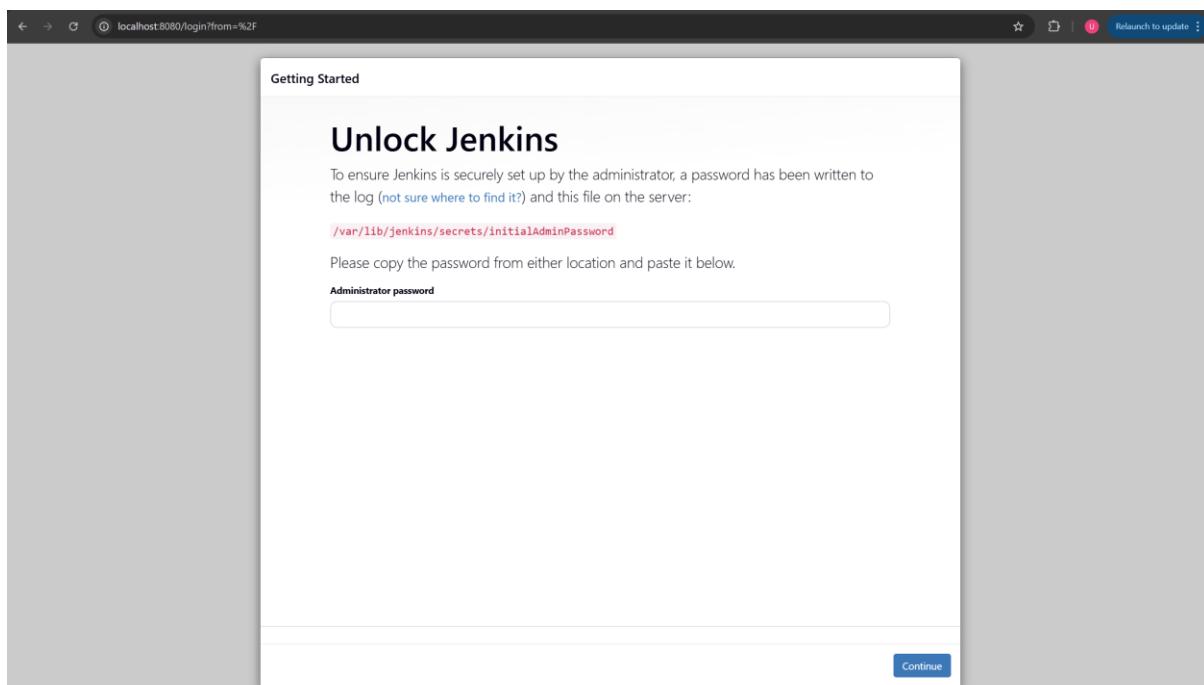
PS C:\Users\umars> aws ssm start-session \
>> --target i-01e3aab55fc39db10 \
>> --document-name AWS-StartPortForwardingSession \
>> --parameters "portNumber=8080,localPortNumber=8080"

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
315a1ala89584e2790cf7e4651d5c257
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**.

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

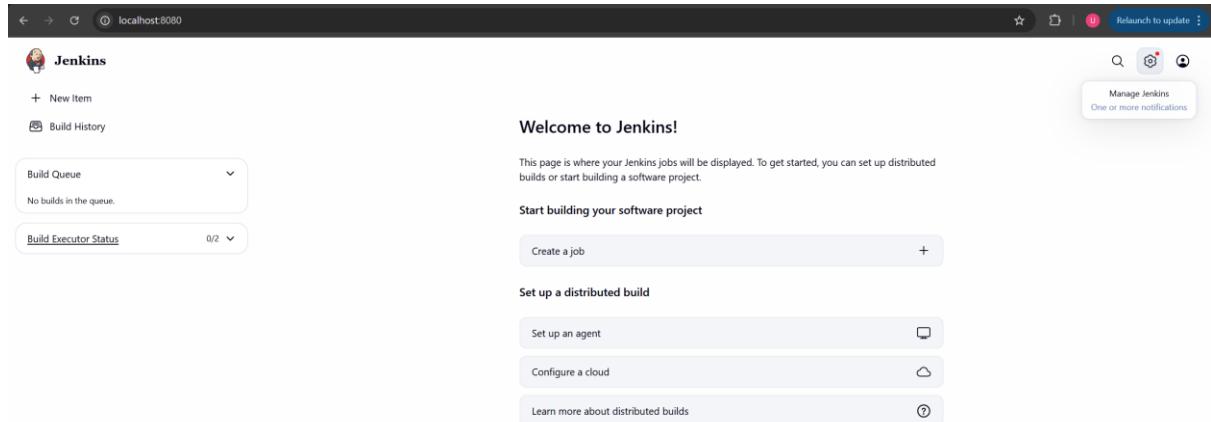
Jenkins 2.528.3

Not now

Save and Finish

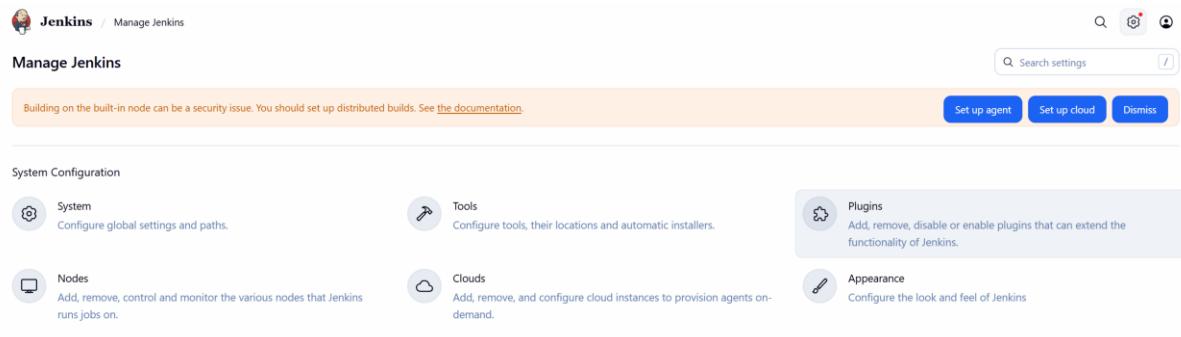
5.2 Install Plugins

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



The screenshot shows the Jenkins dashboard at localhost:8080. The main header says "Welcome to Jenkins!". Below it, there's a message: "This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project." There are several buttons and links: "Create a job", "Set up a distributed build", "Set up an agent", "Configure a cloud", and "Learn more about distributed builds". On the left, there are sections for "Build Queue" (No builds in the queue) and "Build Executor Status" (0/2). A "Manage Jenkins" button is visible in the top right corner.

Under System Configuration, choose **Plugins**.



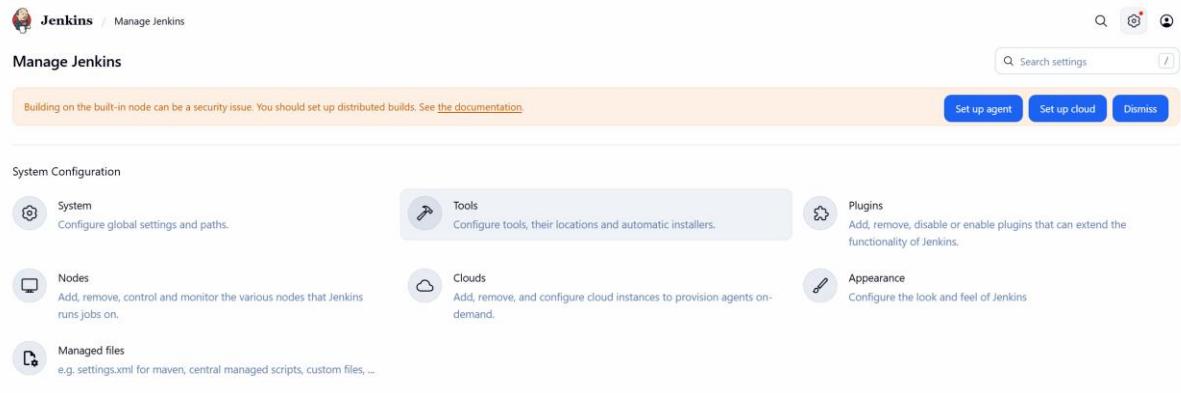
The screenshot shows the "Manage Jenkins" page under "System Configuration". It lists several sections: "System", "Tools", "Nodes", "Clouds", "Plugins" (which is highlighted), and "Appearance". A prominent orange bar at the top contains the text: "Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#)". Below this bar are three buttons: "Set up agent", "Set up cloud", and "Dismiss".

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

SSH Agent	 Success
JSch dependency	 Success
SSH Pipeline Steps	 Success
Loading plugin extensions	 Success
Config File Provider	 Success
NodeJS	 Success
Loading plugin extensions	 Success

5.3 Install NodeJS Tool

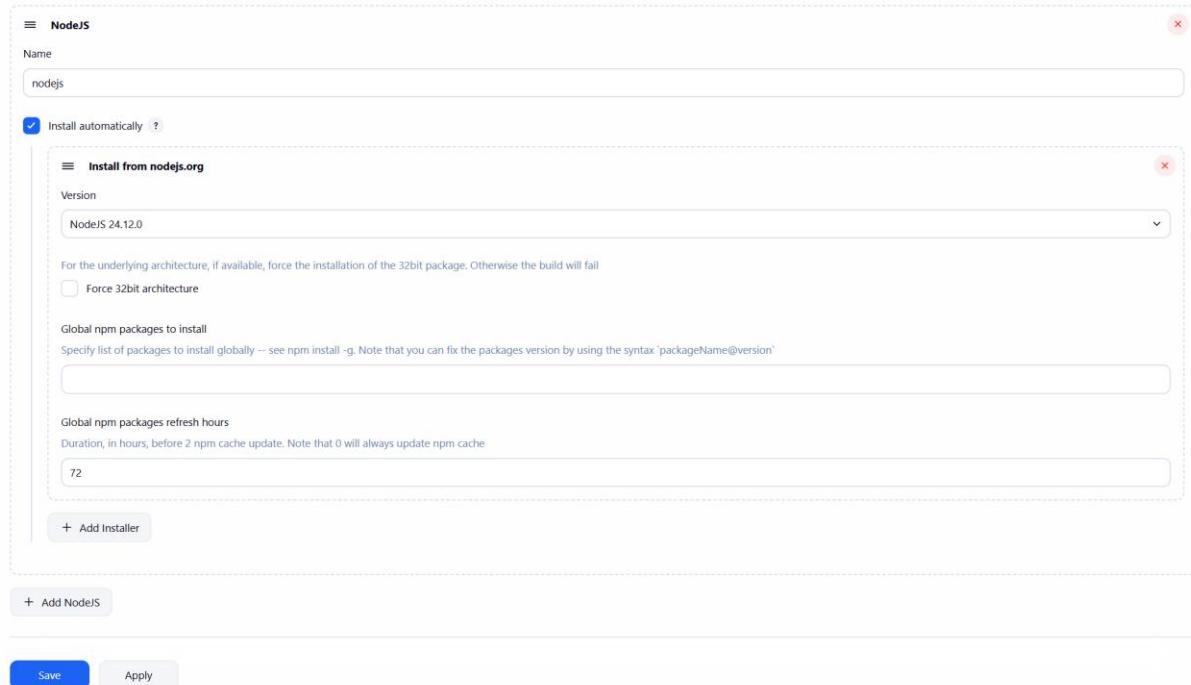
On the Jenkins Dashboard, select Manage Jenkins (settings icon) on the top right. Under System Configuration, choose **Tools**.



The screenshot shows the Jenkins Manage Jenkins interface. At the top, there's a navigation bar with the Jenkins logo and a search bar labeled "Search settings". Below the navigation bar, a message says "Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#)". There are three buttons at the top right: "Set up agent", "Set up cloud", and "Dismiss". The main area is titled "System Configuration" and contains several sections: "System" (Configure global settings and paths), "Tools" (selected, Configure tools, their locations and automatic installers), "Clouds" (Add, remove, and configure cloud instances to provision agents on-demand), and "Plugins" (Add, remove, disable or enable plugins that can extend the functionality of Jenkins). "Appearance" is also listed. A note at the bottom of the configuration section says "e.g. settings.xml for maven, central managed scripts, custom files, ...".

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**



The screenshot shows the "NodeJS" configuration dialog. It has a "Name" field containing "nodejs" and an "Install automatically" checkbox checked. Below this is a "Version" field set to "NodeJS 24.12.0". A note says "For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail" with a "Force 32bit architecture" checkbox. A "Global npm packages to install" section allows specifying a list of packages to install globally using the syntax "packageName@version". A "Global npm packages refresh hours" section specifies a duration of "72". At the bottom, there are "Save" and "Apply" buttons.

5.4: Add SSH Key Credentials

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

The screenshot shows the Jenkins 'Manage Jenkins' interface. In the 'System Configuration' section, there are links for 'System', 'Tools', 'Nodes', 'Clouds', 'Managed files', and 'Appearance'. In the 'Security' section, there are links for 'Security', 'Credentials' (which is highlighted in blue), and 'User'. The 'Credentials' link is underlined and has a tooltip 'Configure 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**.

The screenshot shows the 'Credentials' page. At the top, there is a table header with columns: T, P, Store, Domain, ID, and Name. Below the header, there is a section titled 'Stores scoped to Jenkins' with a table. The table has columns: P, Store, Domains, and a 'Add credentials' button. The 'Domains' column for the first row shows '(global)' with a dropdown arrow. The 'Add credentials' button is highlighted with a red box.

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**

New credentials

Kind
SSH Username with private key

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

ID ?
ec2-ssh-key

Description ?

Username
ubuntu

Treat username as secret ?

Private Key
 Enter directly
Key
No Stored Value Add

Passphrase

Create

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

Jenkins / Manage Jenkins / Credentials / System / Global credentials (unrestricted...)

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
ec2-ssh-key	ubuntu	SSH Username with private key	🔧

Icon: S M L

5.5 Pipeline Configuration

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

Jenkins

+ New Item

Build History

Welcome to Jenkins!

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

Start building your software project

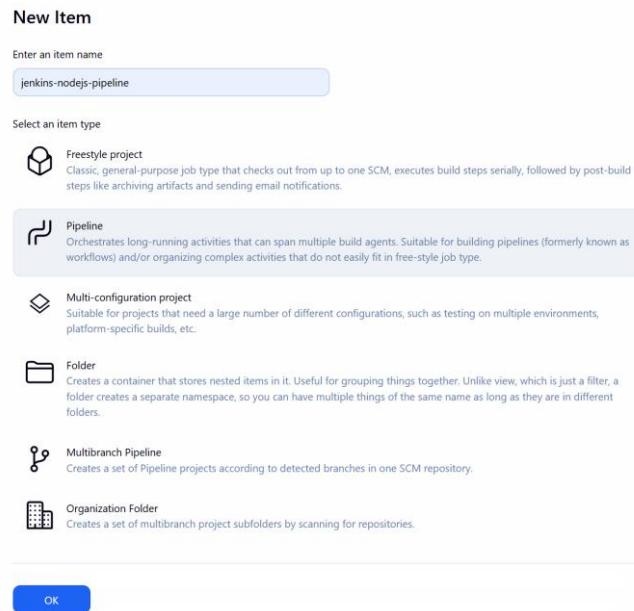
Build Queue
No builds in the queue.

Build Executor Status
0/2

Create a job +

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.

Pipeline
Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script

Script ?
 try sample Pipeline... ▾

Use Groovy Sandbox ?

Pipeline Syntax

Advanced

Advanced ▾

Save Apply

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.

The screenshot shows the Jenkins Pipeline page for the 'jenkins-nodejs-pipeline' job. At the top, there's a navigation bar with a Jenkins logo and the path 'jenkins-nodejs-pipeline'. Below it is a sidebar with links like Status, Changes, Build Now, Configure, Delete Pipeline, Stages, Rename, and Pipeline Syntax. The main content area has a green checkmark icon next to the pipeline name 'jenkins-nodejs-pipeline'. Under 'Permalinks', there's a list of build links. The 'Builds' section contains a table with columns for build number, timestamp, and status. It shows build #2 from today at 8:20 PM and build #1 from today at 8:17 PM.

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/Umarsatt1/Task-14-Jenkins-Setup-using-Terraform-for-Nodejs-EC2-Deployment.git # timeout=10
Fetching upstream changes from https://github.com/Umarsatt1/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/Umarsatt1/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 bcfaf2577dc9708dbd952d19f55c63af799041845 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f bcfaf2577dc9708dbd952d19f55c63af799041845 # timeout=10
> git branch -a -v --no-abbrev # timeout=10
> git branch -D main # timeout=10
> git checkout -b main bcfaf2577dc9708dbd952d19f55c63af799041845 # timeout=10
Commit message: "Update index.html"
> git rev-list --no-walk bcfaf2577dc9708dbd952d19f55c63af799041845 # 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-oNng90hm8EY/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

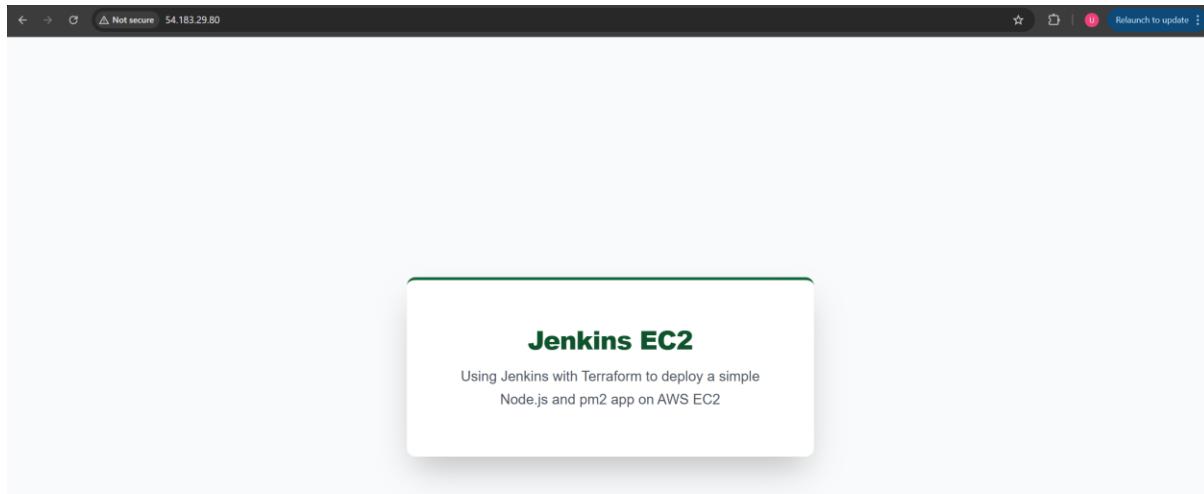
Navigate to the Pipeline page, select the pipeline named **jenkins-nodejs-pipeline**, and click console output. Ensure that each stage completes without errors.

```
$ 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.

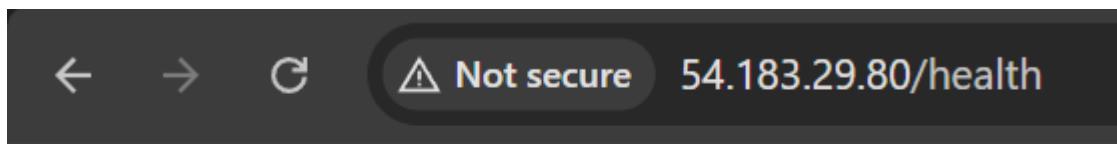
```
root@ip-10-0-1-231:~# sudo systemctl status amazon-cloudwatch-agent
● amazon-cloudwatch-agent.service - Amazon CloudWatch Agent
   Loaded: loaded (/etc/systemd/system/amazon-cloudwatch-agent.service; enabled; preset: enabled)
   Active: active (running) since Sun 2025-12-28 19:06:44 UTC; 1h 28min ago
     Main PID: 3161 (amazon-cloudwatch)
       Tasks: 8 (limit: 2204)
      Memory: 22.4M (peak: 26.2M)
        CPU: 4.400ms
       CGroup: /system.slice/amazon-cloudwatch-agent.service
               └─3161 /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -config /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml -envconfig /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.env

Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3166]: Executing /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent with arguments: [config-translator -input-dir /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml -envconfig /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.env]
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3166]: I1 [002] The configuration file /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml is valid.
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3171]: I1 [mds] retry client will retry 1 times!! Detected the instance is EC2
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3171]: 2025/12/28 19:06:44 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json ...
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3171]: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json does not exist or cannot read. Skipping it.
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3171]: 2025/12/28 19:06:44 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_amazon-cloudwatch-agent.json ...
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3171]: 2025/12/28 19:06:44 I1 Valid Json input schema.
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3171]: I1 Trying to detect region from ec2
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3171]: 2025/12/28 19:06:44 Configuration validation first phase succeeded
Dec 28 19:06:44 ip-10-0-1-231 start-amazon-cloudwatch-agent[3161]: I1 Detecting run_as_user...
```

- /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

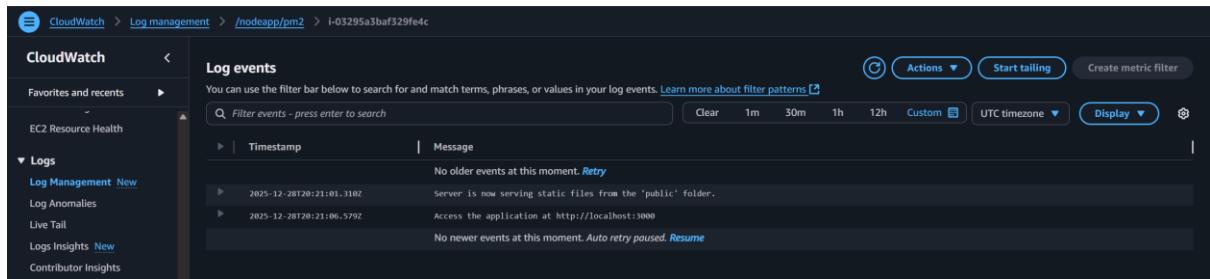
Nginx serves as a reverse proxy in front of the Node.js application. Access and error logs are stored under `/var/log/nginx/`. During application restarts or deployments, transient 502 Bad Gateway errors may appear when the upstream Node.js service is temporarily unavailable.

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

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.



The screenshot shows the AWS CloudWatch Log Management interface. The left sidebar navigation includes CloudWatch, Favorites and recents, EC2 Resource Health, and Logs (with sub-options: Log Management, Log Anomalies, Live Tail, Logs Insights, and Contributor Insights). The main content area is titled "Log events" and displays two log entries:

Timestamp	Message
2025-12-28T20:21:01.310Z	Server is now serving static files from the 'public' folder.
2025-12-28T20:21:06.579Z	Access the application at http://localhost:3000

At the top right, there are buttons for Actions, Start tailing, Create metric filter, and a filter bar with a search input and time range selector (Custom, UTC timezone, Display).

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.

```
Destroy complete! Resources: 19 destroyed.  
○ PS D:\Cloudelligent\Task-14\terraform> █
```

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.