TASK 2

AWS EC2 WordPress Deployment with Terraform

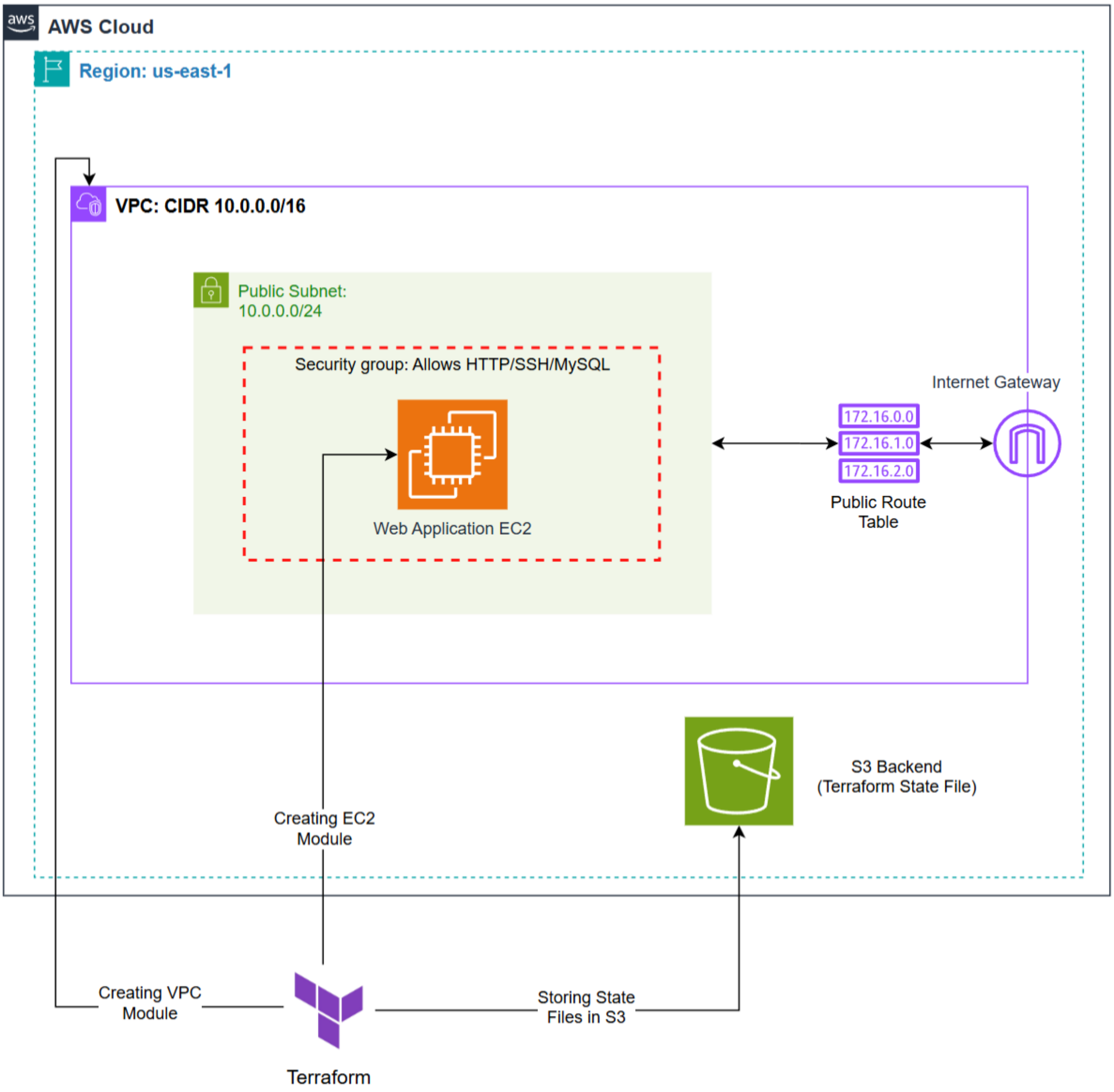
Umar Satti

Table of Contents

Task Objective

Provision an EC2 instance on AWS using Terraform, install WordPress, set up a MySQL database on the instance, and configure WordPress to use that MySQL database. This setup should utilize a user data script for automation.

Architecture Diagram



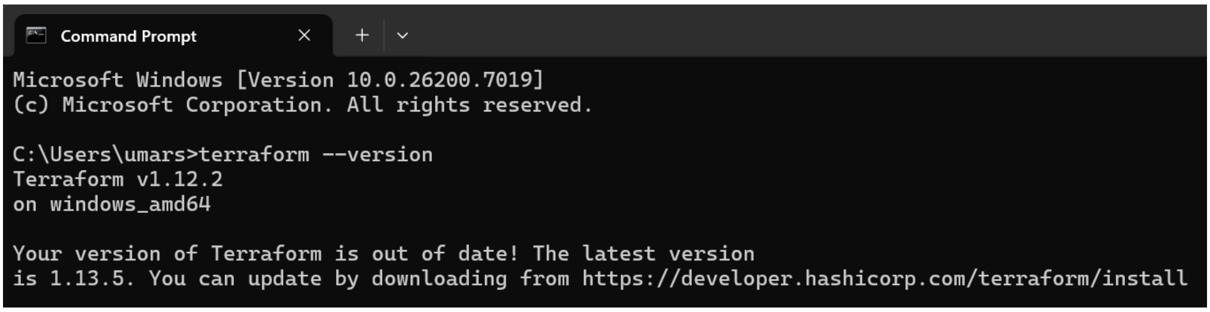
WordPress Deployment on AWS

This guide walks through step-by-step instructions for deploying a WordPress application using Terraform on AWS

Task 1.1: Install Terraform on your local machine or in a development environment

Steps:

1. Navigate to HashiCorp official website to install the Terraform CLI for Windows (or any operating system). Link: ***https://developer.hashicorp.com/terraform/install***
2. Extract the terraform file and store it in the desired location. Note down the path as this is required later.
3. Open “System Properties” and then click “Environment Variables” at the bottom of the tab.
4. Navigate to the “user variables” and double-click or edit the “Path” option.
5. Add the path that was noted in the second step i.e. “C://Terraform” as this folder contains the terraform.exe file.
6. To verify if Terraform is functional, open a Command Prompt or PowerShell and type the following command: “terraform --version”.
7. This command will display terraform version and operating system. Additionally, it also confirms that terraform is downloaded and accessible from local CLI.



```
Microsoft Windows [Version 10.0.26200.7019]
(c) Microsoft Corporation. All rights reserved.

C:\Users\umars>terraform --version
Terraform v1.12.2
on windows_amd64

Your version of Terraform is out of date! The latest version
is 1.13.5. You can update by downloading from https://developer.hashicorp.com/terraform/install
```

Task 1.2: Configure AWS CLI with your credentials to allow Terraform to interact with your AWS account (on Windows)

Steps:

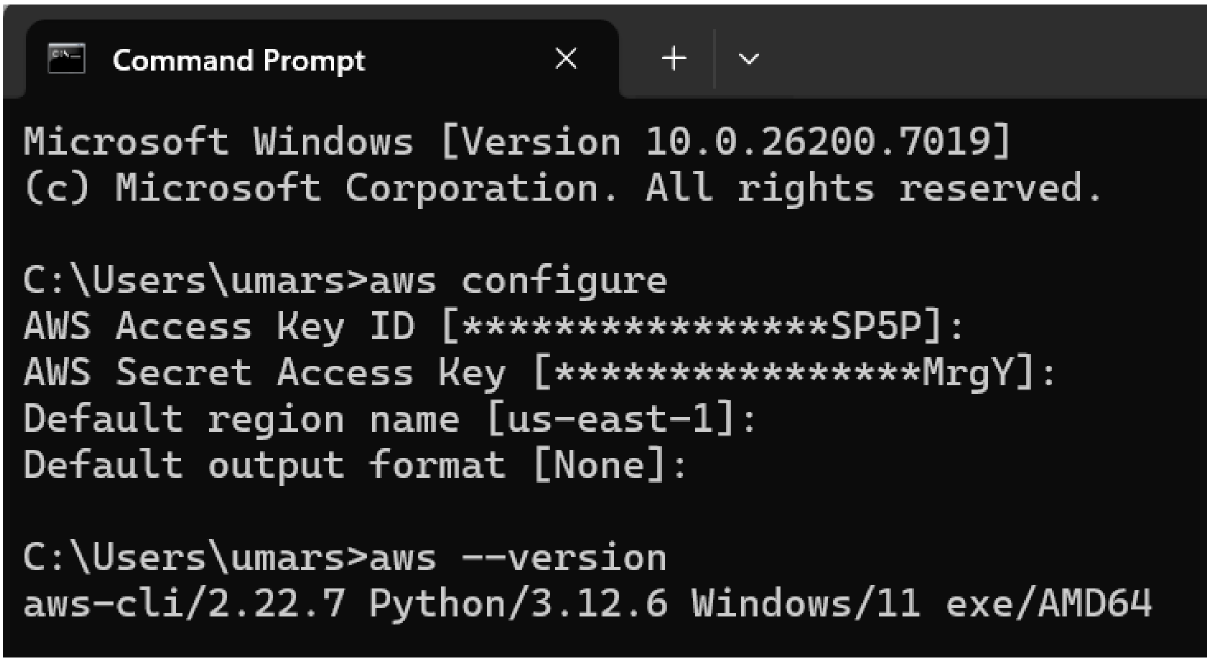
1. Download the AWS CLI MSI installer for Windows (64-bit) from the following link: ***https://awscli.amazonaws.com/AWSCLIV2.msi***
2. Run the AWS CLI installer by choosing custom path. Open “System Properties” and then click “Environment Variables” at the bottom of the tab.

3. Navigate to the “user variables” and double-click or edit the “Path” option.
4. Add the path that was noted in the second step as this folder contains the AWSCLIV2.msi file.
5. Use “aws --version” to confirm that AWS CLI was successfully installed and the path was correctly configured.
6. Use “aws configure” command and input the AWS account credentials.
7. Enter the AWS Access Key ID, AWS Secret Access Key, Default region name, and Default output format.

Note: I downloaded AWS CLI from the official AWS documentation page and then added this **C:\Users\umars\Downloads\AWSCLIV2.msi** path in the environment variables. To confirm that AWS CLI was properly installed, I used the command “aws --version” which displays the aws-cli version 2.22.7.

I also configured AWS CLI using my personal AWS account credentials:

- Access Key ID: ending with SP5P
- Secret Access Key: ending with MrgY
- Default region: us-east-1
- Output format: Default (JSON)



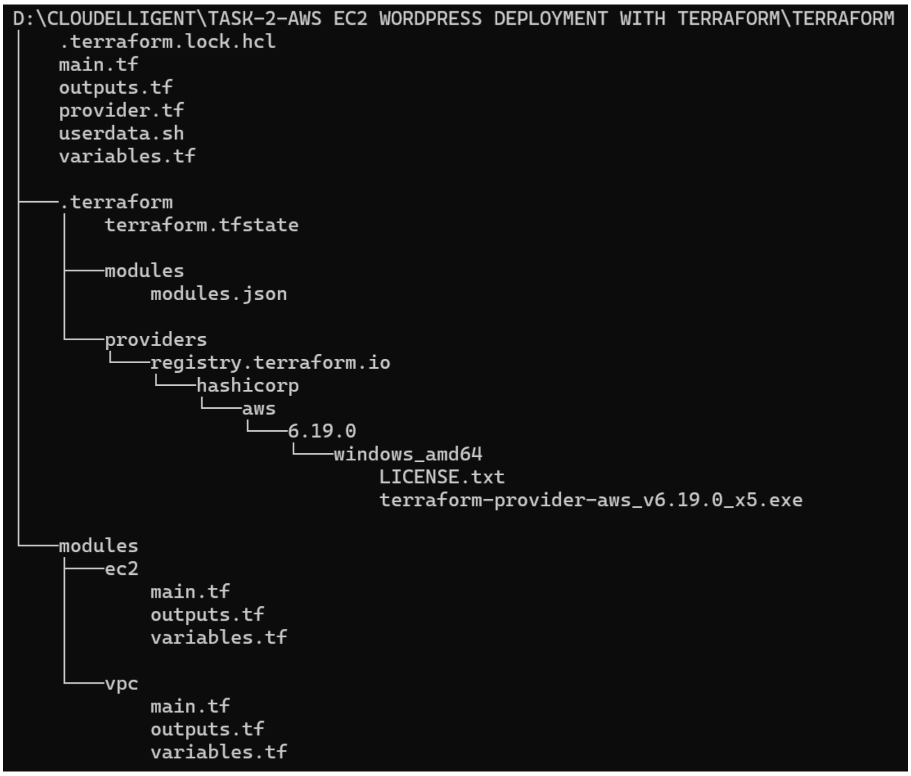
```
Microsoft Windows [Version 10.0.26200.7019]
(c) Microsoft Corporation. All rights reserved.

C:\Users\umars>aws configure
AWS Access Key ID [****************SP5P]:
AWS Secret Access Key [****************MrgY]:
Default region name [us-east-1]:
Default output format [None]:

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

Task 1.3: Create Terraform Project Structure

Steps:

1. Create a provider.tf (or terraform.tf) file in the root directory. This file defines the AWS provider, provider version, AWS region, and the S3 backend for storing Terraform state files.
2. Create an S3 bucket in the same AWS region to store Terraform state files (explained in Task 1.4 below).
3. Create a main.tf file in the root directory. This connects modules together, passes outputs from the VPC module to the EC2 module, and passes variables between them.
4. Create a variables.tf file in the root directory. This file defines default values (key-value pairs) for parameters such as VPC name, CIDR block, EC2 AMI ID, and instance type.
5. Create an outputs.tf file in the root directory. This file exposes important module outputs such as EC2 public IP and public DNS after deployment.
6. Create a modules directory that contains two submodules i.e. vpc and ec2. Each submodule contains its own main.tf, variables.tf, and outputs.tf files.
7. Create a user data bash script (userdata.sh) either in the root directory or inside the EC2 module. This script automates WordPress setup by installing Apache, PHP, MySQL, creating a database, and configuring WordPress automatically on instance boot.

The terraform project directory structure should look like this:



```
D:\CLOUDELLIGENT\TASK-2-AWS EC2 WORDPRESS DEPLOYMENT WITH TERRAFORM\TERRAFORM
│   .terraform.lock.hcl
│   main.tf
│   outputs.tf
│   provider.tf
│   userdata.sh
│   variables.tf
│
├───.terraform
│   │   terraform.tfstate
│   │
│   ├───modules
│   │       modules.json
│   │
│   └───providers
│       └───registry.terraform.io
│           └───hashicorp
│               └───aws
│                   └───6.19.0
│                       └───windows_amd64
│                               LICENSE.txt
│                               terraform-provider-aws_v6.19.0_x5.exe
│
└───modules
    ├───ec2
    │       main.tf
    │       outputs.tf
    │       variables.tf
    │
    └───vpc
            main.tf
            outputs.tf
            variables.tf
```

Task 1.4: Create S3 Bucket for Terraform Remote Backend

Steps:

1. Log in to the AWS Management Console. Navigation to S3 using the search bar at the top.
2. Click on **Create Bucket** button.
3. Choose **General Purpose**, add a globally unique bucket name, and make sure the AWS Region is the same as Terraform.
4. Click **Create Bucket**.
5. Update provider.tf or terraform.tf file in root directory to reference this S3 bucket in the backend block.

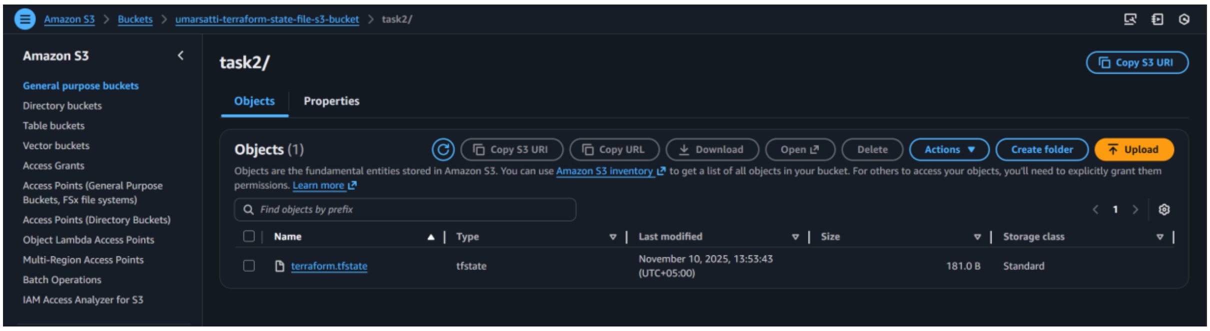
Once the S3 bucket is created in the AWS Management Console and referenced in the Terraform backend configuration, Terraform automatically begins storing and versioning state files in this bucket.

In this case, the S3 bucket named **umarsatti-terraform-state-file-s3-bucket** is used as the remote backend, as defined in the **provider.tf** file. The backend block ensures all state information is centralized, secure, and persistent across multiple users or workstations.

The screenshot you included shows the exact file path inside your S3 bucket:
S3 > Buckets > umarsatti-terraform-state-file-s3-bucket > task2 > terraform.tfstate

This confirms that:

- Terraform successfully initialized the backend and wrote the state file to the S3 bucket.
- The **terraform.tfstate** file contains metadata about all deployed AWS resources (VPC, Subnet, EC2, etc.).
- Every terraform plan, apply, or destroy operation reads and updates this file automatically.
- The locking mechanism (enabled by **use_lockfile = true**) ensures that no two processes modify the state simultaneously, preventing state corruption.



Amazon S3 > Buckets > umarsatti-terraform-state-file-s3-bucket > task2/

task2/

Objects (1)

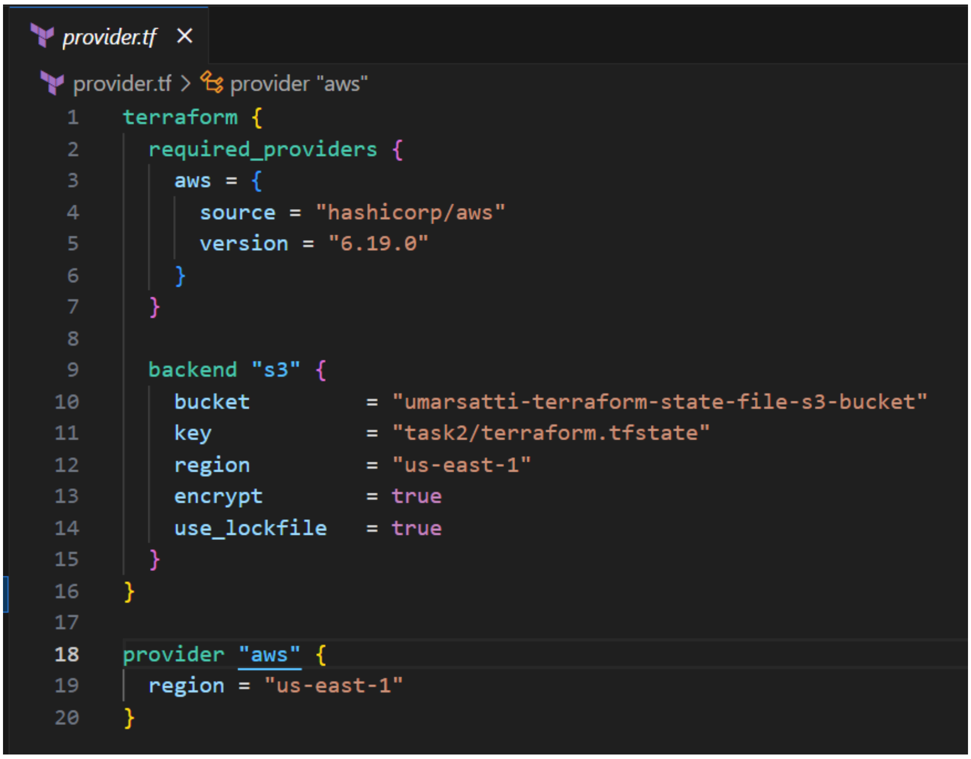
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
terraform.tfstate	tfstate	November 10, 2025, 13:53:43 (UTC+05:00)	181.0 B	Standard

Task 1.5: Root Directory Files

This section explains each Terraform configuration file located in the project’s root directory and their purpose in connecting the VPC and EC2 modules.

provider.tf:



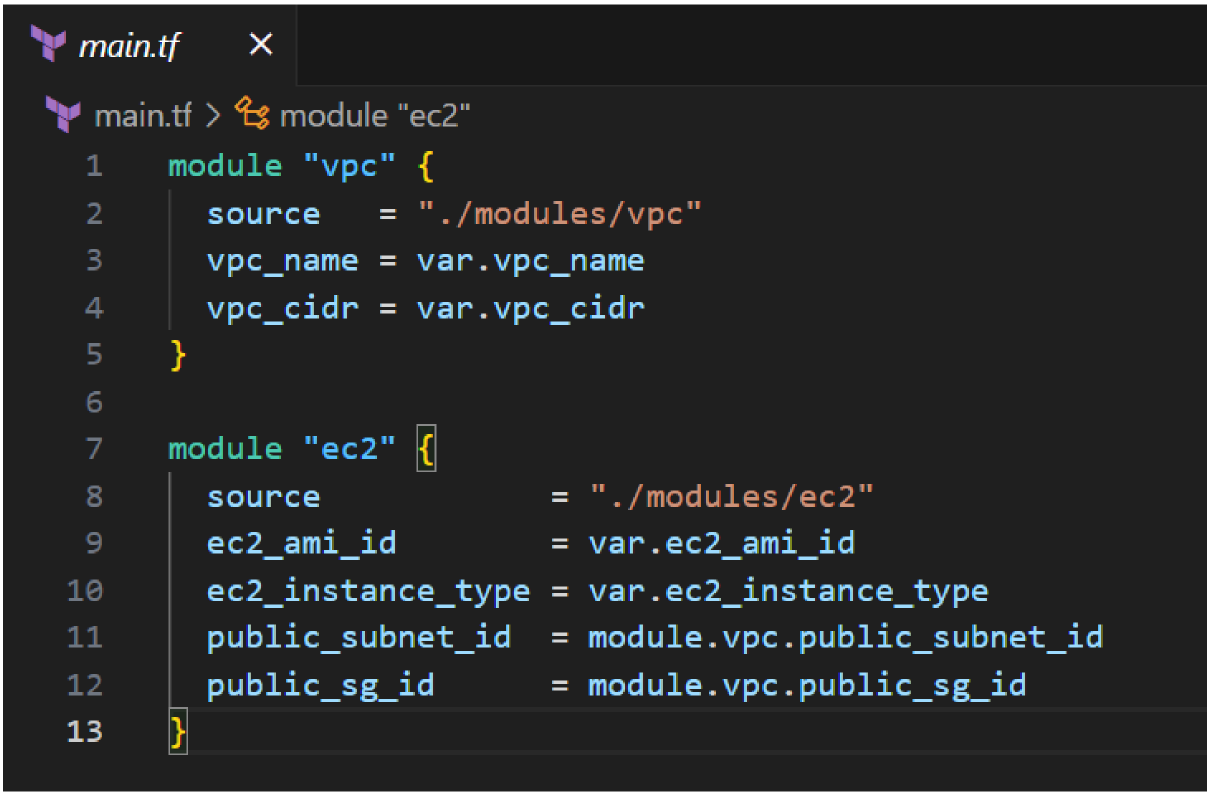
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "6.19.0"
    }
  }

  backend "s3" {
    bucket         = "umarsatti-terraform-state-file-s3-bucket"
    key            = "task2/terraform.tfstate"
    region         = "us-east-1"
    encrypt        = true
    use_lockfile   = true
  }
}

provider "aws" {
  region = "us-east-1"
}
```

This file defines the **AWS provider** configuration and the **remote backend** for storing the Terraform state file securely in an Amazon S3 bucket.

- The terraform block specifies that this project uses the **AWS provider** sourced from HashiCorp, version 6.19.0.
- The backend "s3" configuration ensures that the Terraform state file (terraform.tfstate) is stored in an **S3 bucket** named **umarsatti-terraform-state-file-s3-bucket** under the path **task2/terraform.tfstate**.
- State file **encryption** is enabled, and **use_lockfile = true** prevents multiple users from updating the state simultaneously.
- The provider "aws" block defines the **AWS region (us-east-1)** where all infrastructure resources will be created.

Using a remote backend improves collaboration and ensures the state file is safe even if the local environment changes.

main.tf:



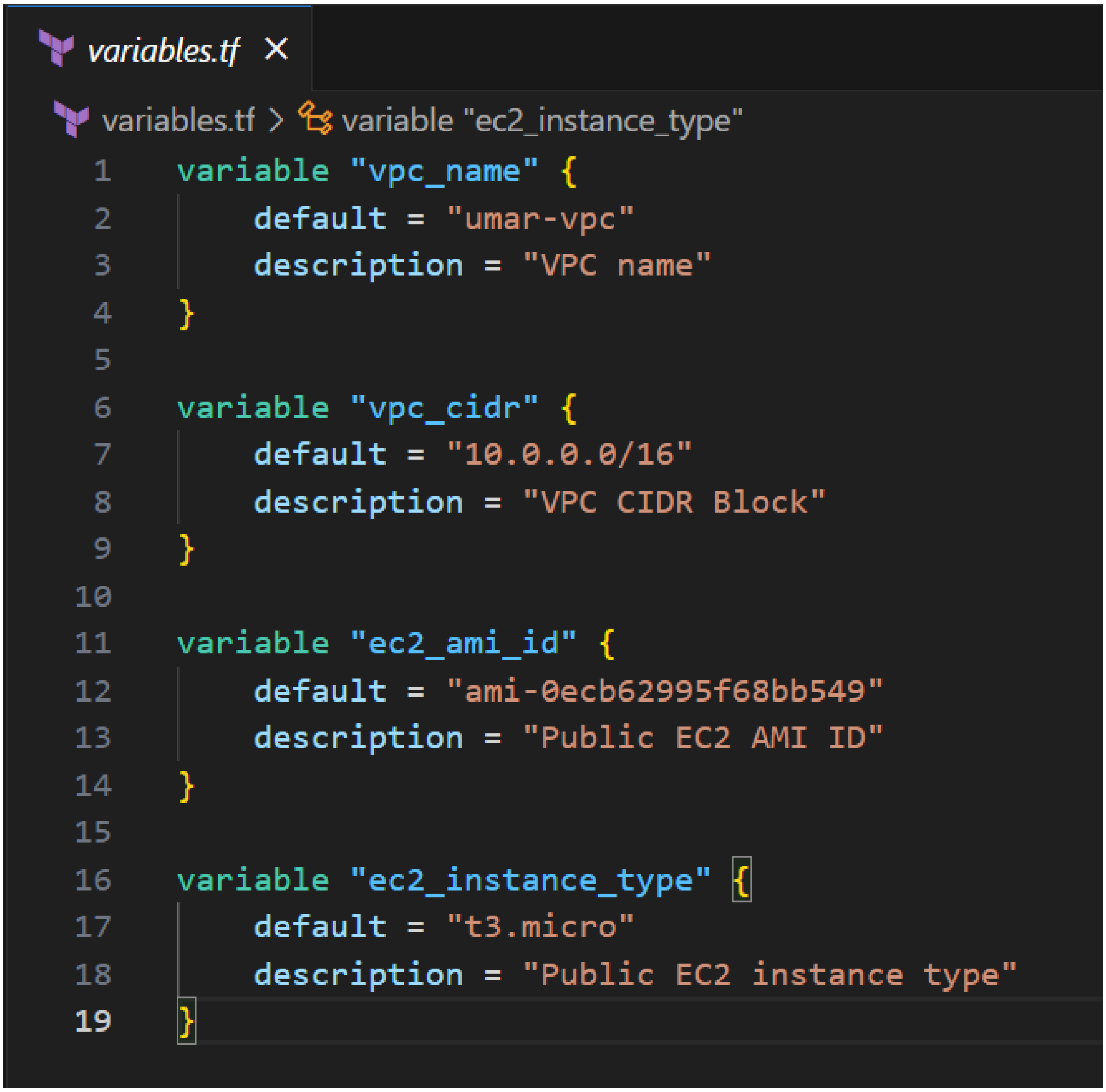
```
module "vpc" {
  source   = "./modules/vpc"
  vpc_name = var.vpc_name
  vpc_cidr = var.vpc_cidr
}

module "ec2" {
  source            = "./modules/ec2"
  ec2_ami_id        = var.ec2_ami_id
  ec2_instance_type = var.ec2_instance_type
  public_subnet_id  = module.vpc.public_subnet_id
  public_sg_id      = module.vpc.public_sg_id
}
```

This file defines how Terraform composes the overall infrastructure using **modules** for networking and compute layers.

- The **VPC module** (./modules/vpc) provisions all networking components including VPC, subnets, internet gateway, route tables, and security groups. The variables vpc_name and vpc_cidr are passed from the root level variables file.
- The **EC2 module** (./modules/ec2) provisions an EC2 instance that hosts WordPress. It references the **subnet ID** and **security group ID** outputs from the VPC module (module.vpc.public_subnet_id and module.vpc.public_sg_id) to ensure proper network placement.

This modular structure promotes reusability and clean separation of concerns, where each module handles a specific AWS service.

variables.tf:

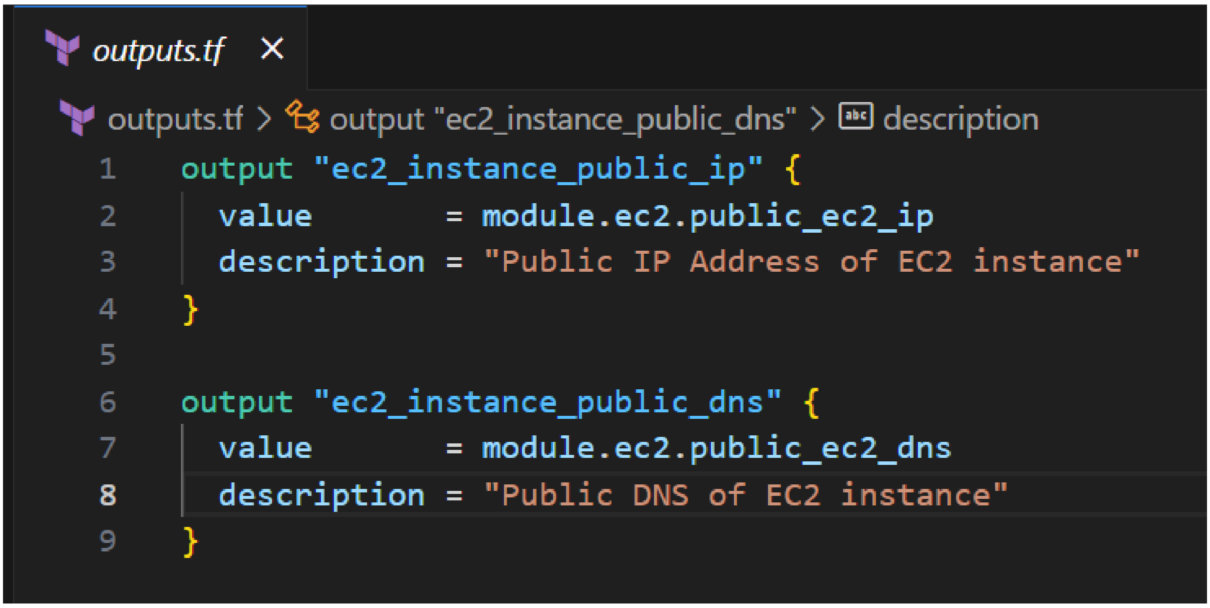


```
variable "vpc_name" {
    default = "umar-vpc"
    description = "VPC name"
}

variable "vpc_cidr" {
    default = "10.0.0.0/16"
    description = "VPC CIDR Block"
}

variable "ec2_ami_id" {
    default = "ami-0ecb62995f68bb549"
    description = "Public EC2 AMI ID"
}

variable "ec2_instance_type" {
    default = "t3.micro"
    description = "Public EC2 instance type"
}
```

This file defines the **input variables** for the Terraform project.

- **vpc_name** and **vpc_cidr** are used by the VPC module to define the VPC name and IPv4 CIDR range.
- **ec2_ami_id** specifies the Amazon Machine Image used to create the EC2 instance. This is an Ubuntu AMI.
- **ec2_instance_type** defines the compute capacity of the instance. A t3.micro is used for this task.

outputs.tf:

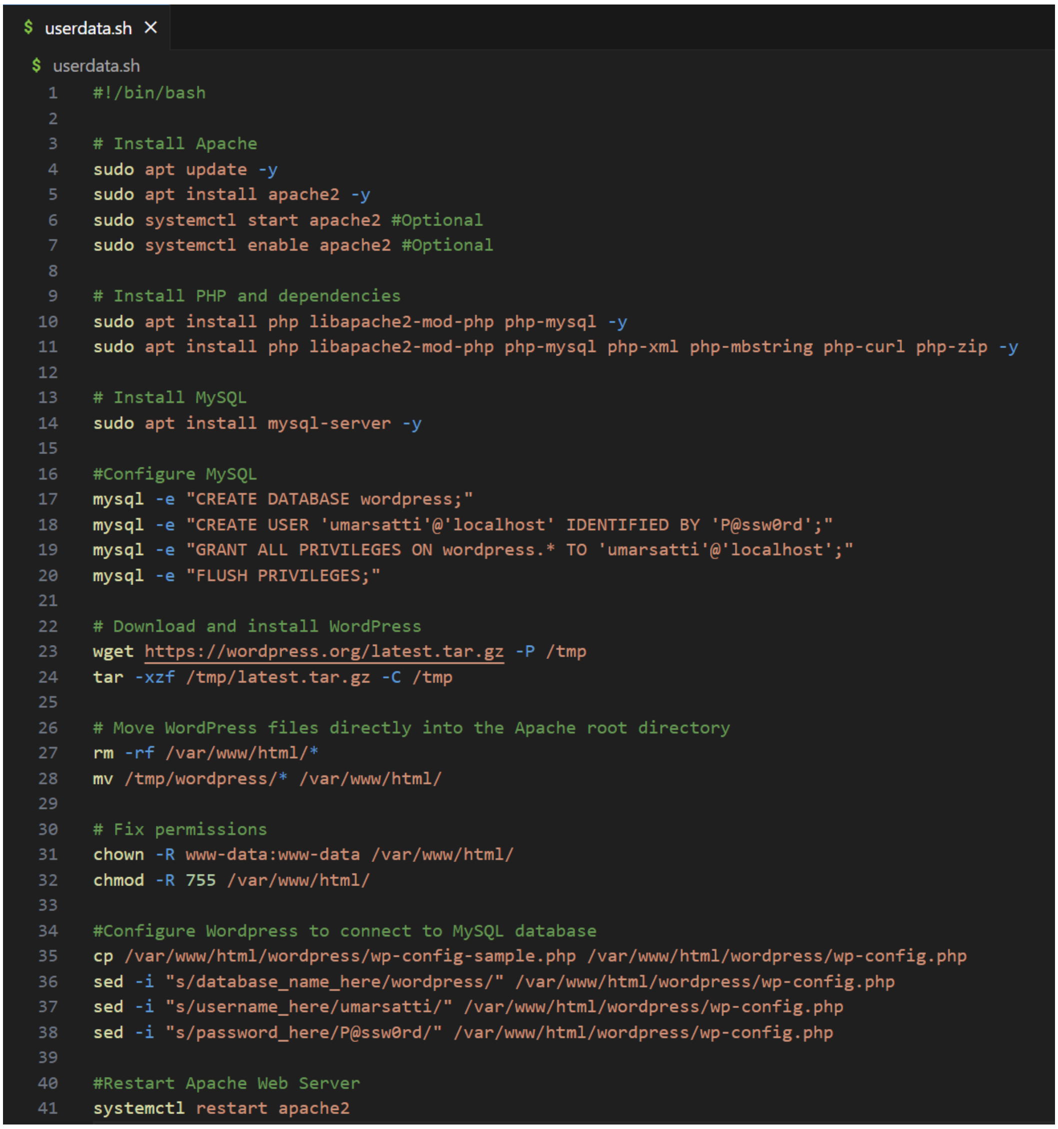


```
output "ec2_instance_public_ip" {
  value       = module.ec2.public_ec2_ip
  description = "Public IP Address of EC2 instance"
}

output "ec2_instance_public_dns" {
  value       = module.ec2.public_ec2_dns
  description = "Public DNS of EC2 instance"
}
```

The outputs file displays key information about deployed resources after a successful terraform apply execution.

- **ec2_instance_public_ip** outputs the public IP address of the EC2 instance, allowing direct access to the WordPress application.
- **ec2_instance_public_dns** outputs the public DNS name of the EC2 instance, which can also be used to access WordPress in a browser.

userdata.sh (User Data script):



```bash
#!/bin/bash

# Install Apache
sudo apt update -y
sudo apt install apache2 -y
sudo systemctl start apache2 #Optional
sudo systemctl enable apache2 #Optional

# Install PHP and dependencies
sudo apt install php libapache2-mod-php php-mysql -y
sudo apt install php libapache2-mod-php php-mysql php-xml php-mbstring php-curl php-zip -y

# Install MySQL
sudo apt install mysql-server -y

#Configure MySQL
mysql -e "CREATE DATABASE wordpress;"
mysql -e "CREATE USER 'umarsatti'@'localhost' IDENTIFIED BY 'P@ssw0rd';"
mysql -e "GRANT ALL PRIVILEGES ON wordpress.* TO 'umarsatti'@'localhost';"
mysql -e "FLUSH PRIVILEGES;"

# Download and install WordPress
wget https://wordpress.org/latest.tar.gz -P /tmp
tar -xzf /tmp/latest.tar.gz -C /tmp

# Move WordPress files directly into the Apache root directory
rm -rf /var/www/html/*
mv /tmp/wordpress/* /var/www/html/

# Fix permissions
chown -R www-data:www-data /var/www/html/
chmod -R 755 /var/www/html/

#Configure Wordpress to connect to MySQL database
cp /var/www/html/wordpress/wp-config-sample.php /var/www/html/wordpress/wp-config.php
sed -i "s/database_name_here/wordpress/" /var/www/html/wordpress/wp-config.php
sed -i "s/username_here/umarsatti/" /var/www/html/wordpress/wp-config.php
sed -i "s/password_here/P@ssw0rd/" /var/www/html/wordpress/wp-config.php

#Restart Apache Web Server
systemctl restart apache2
```

This user data script automates the deployment and configuration of the WordPress environment on the EC2 instance at startup.

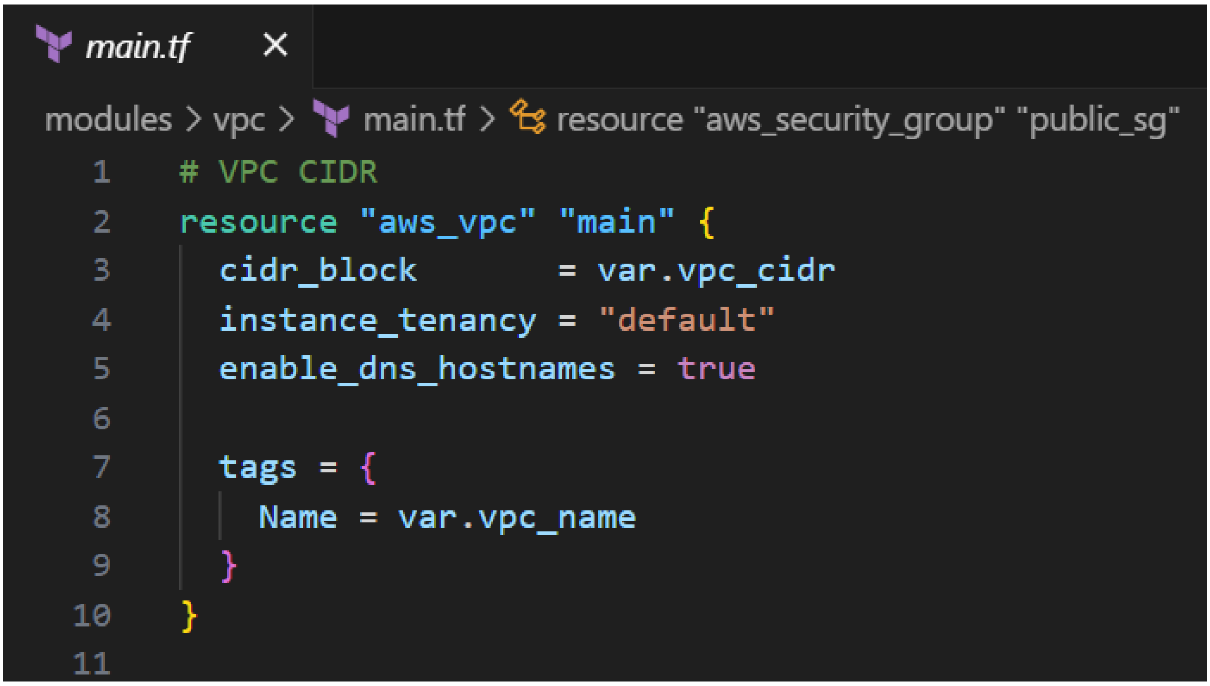
- Installs and starts the **Apache web server** and **PHP** with all required extensions for WordPress.
- Installs **MySQL server**, creates a **wordpress** database, and sets up a dedicated user named **umarsatti** with full privileges.
- Downloads and extracts the latest version of **WordPress**, sets file permissions, and configures database credentials in **wp-config.php**.

This automation ensures a fully functional **WordPress** site is ready immediately after the EC2 instance is launched.

Task 1.6: Configure VPC Module

This section explains each Terraform configuration file located in the VPC modules (modules/vpc) directory.

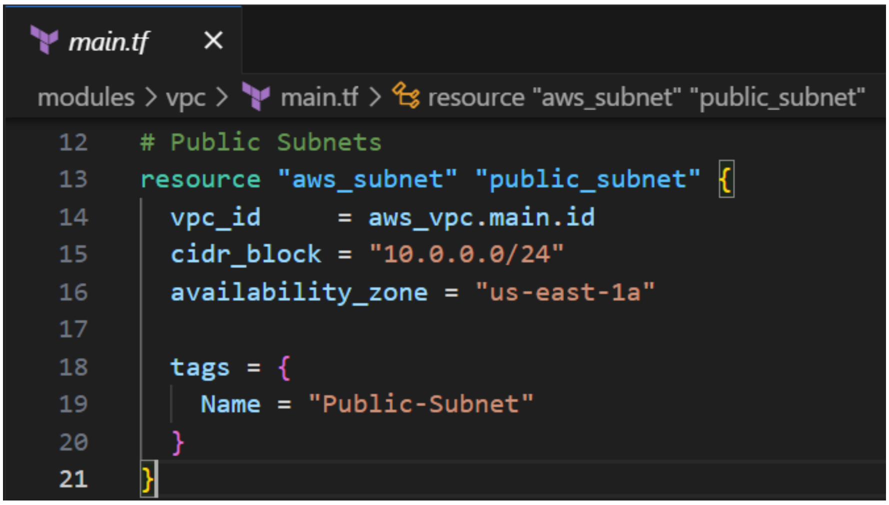
main.tf:


main.tf ×
modules > vpc > main.tf > resource "aws_security_group" "public_sg"

```
# VPC CIDR
resource "aws_vpc" "main" {
  cidr_block       = var.vpc_cidr
  instance_tenancy = "default"
  enable_dns_hostnames = true

  tags = {
    Name = var.vpc_name
  }
}

```

This block defines the VPC networking environment.

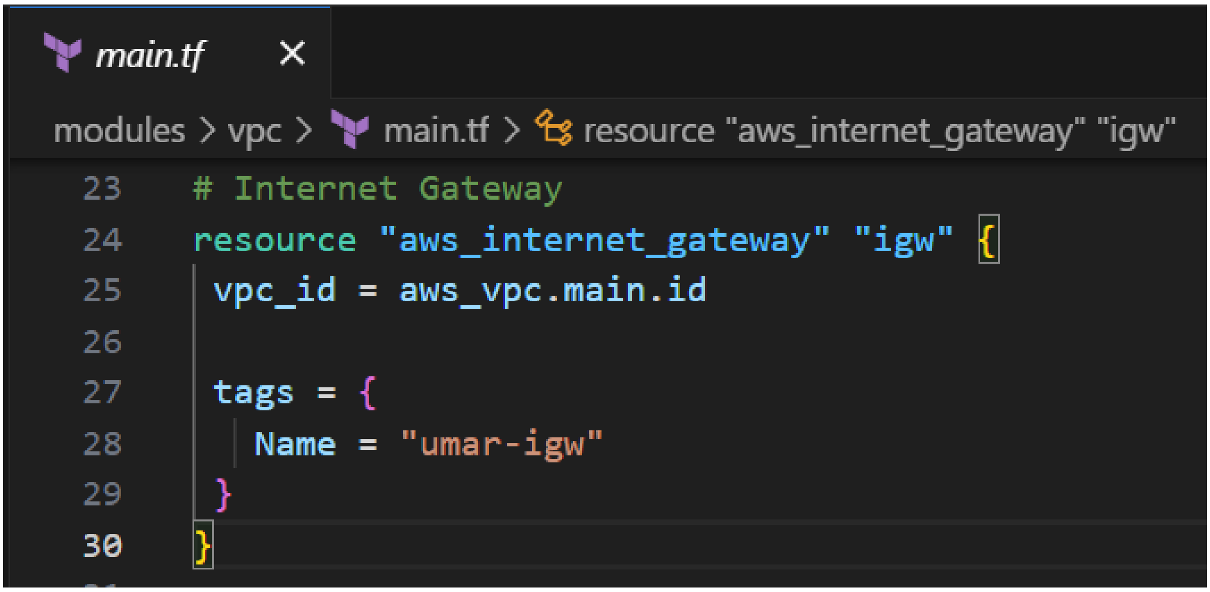
- The **cidr_block** value is dynamically passed from the variables.tf file
- **enable_dns_hostnames = true** ensures that instances within the VPC automatically receive DNS hostnames
- The VPC is tagged with a dynamic name (var.vpc_name) for better visibility and resource management inside AWS.


main.tf
modules > vpc > main.tf > resource "aws_subnet" "public_subnet"

```
# Public Subnets
resource "aws_subnet" "public_subnet" {
  vpc_id      = aws_vpc.main.id
  cidr_block = "10.0.0.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "Public-Subnet"
  }
}
```

This block creates a **public subnet** within the previously defined VPC.

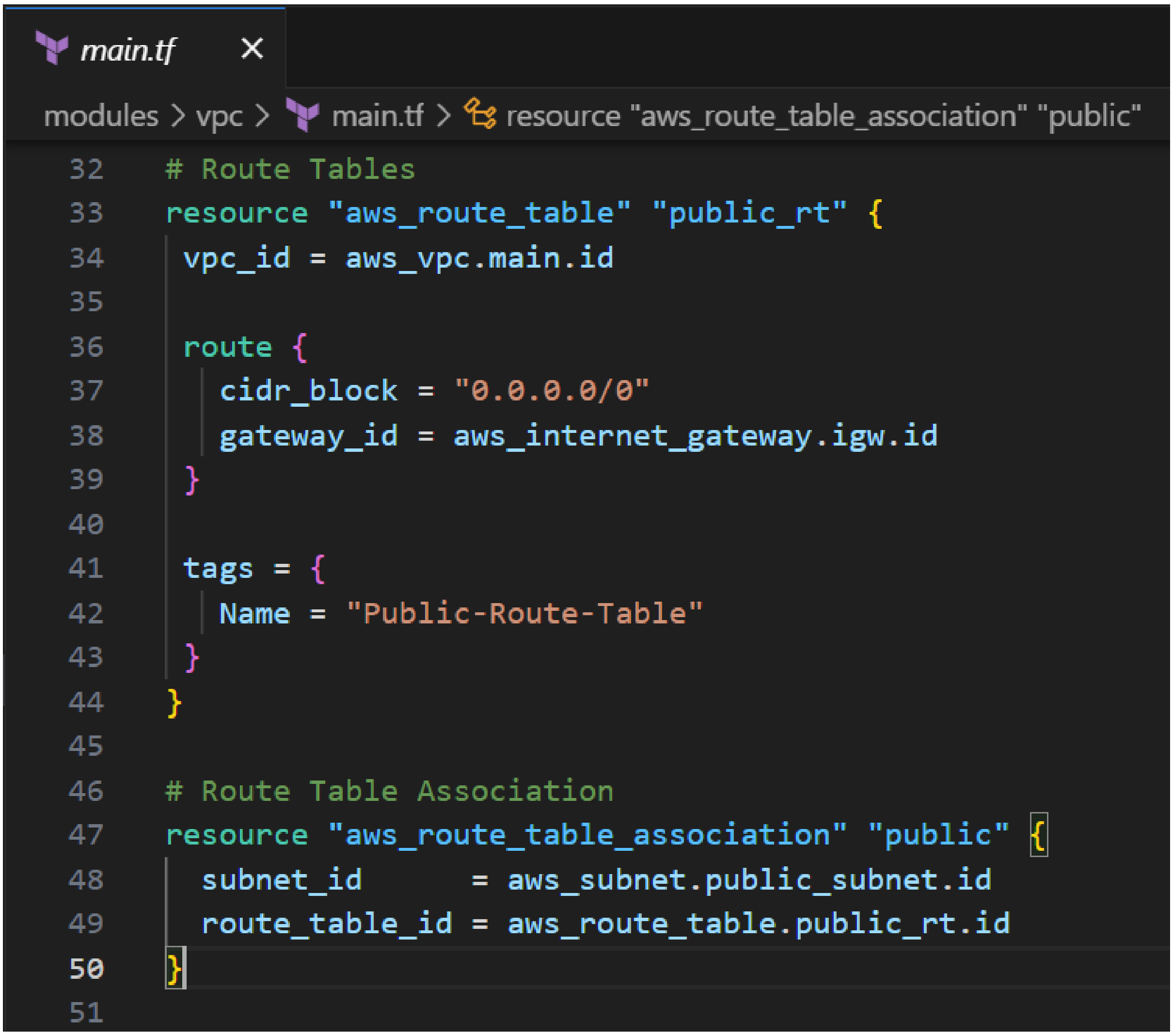
- The subnet uses the CIDR block 10.0.0.0/24
- It resides in the **us-east-1a** Availability Zone
- The **vpc_id** references the VPC created earlier, ensuring proper network association.
- Name of this subnet (using tags) is “Public-Subnet”.


main.tf
modules > vpc > main.tf > resource "aws_internet_gateway" "igw"

```
# Internet Gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "umar-igw"
  }
}
```

This block provisions an **Internet Gateway (IGW)** and attaches it to the VPC.

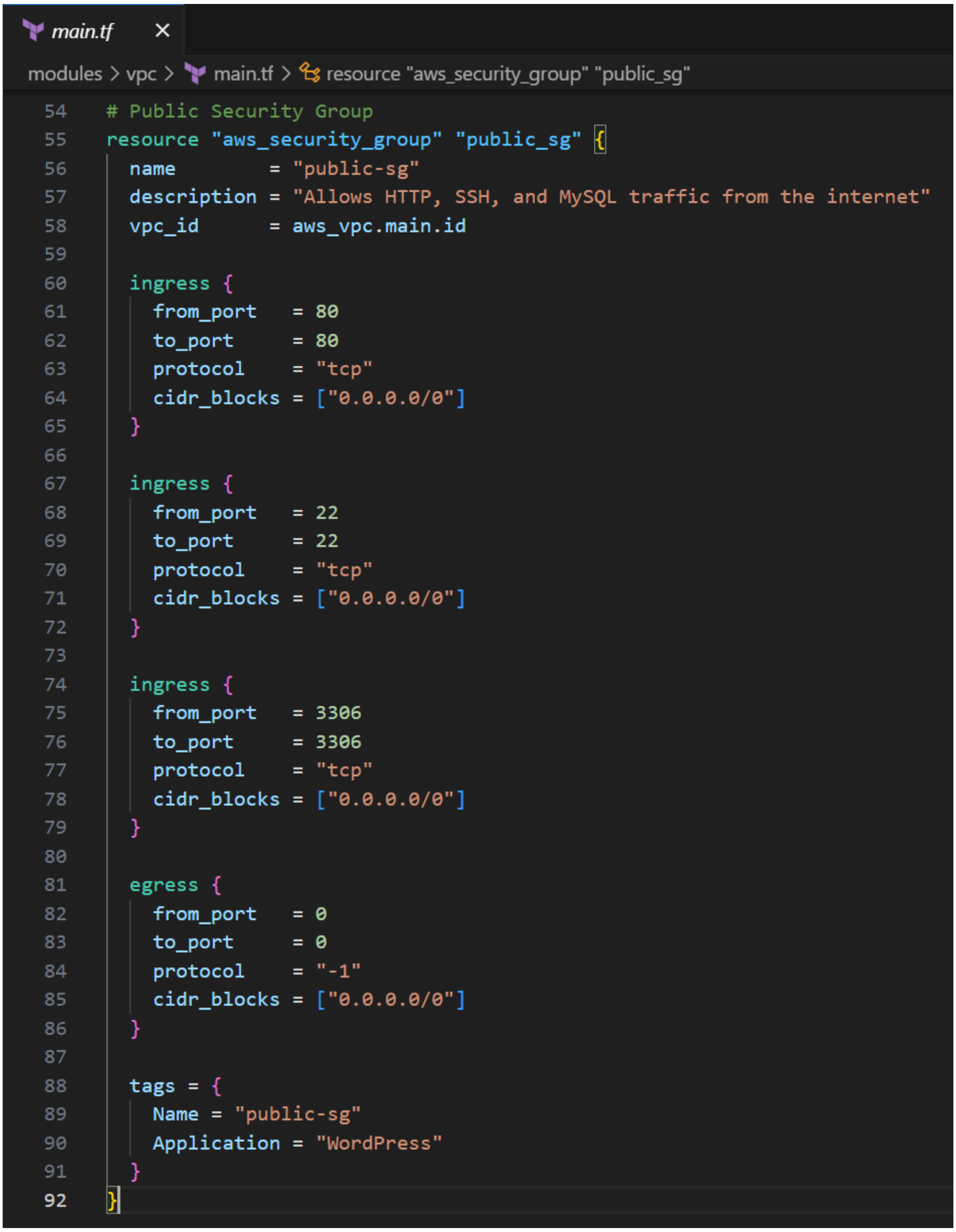
- The IGW enables instances in the public subnet to communicate with the internet.
- The vpc_id interpolation ties the IGW to the specific VPC, ensuring connectivity.
- Tagged as “umar-igw” for easy identification.


main.tf
modules > vpc > main.tf > resource "aws_route_table_association" "public"

```
# Route Tables
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "Public-Route-Table"
  }
}

# Route Table Association
resource "aws_route_table_association" "public" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_rt.id
}
```

This section sets up routing for the public subnet:

- The **Route Table** directs all outbound traffic (0.0.0.0/0) through the **Internet Gateway**, enabling external connectivity.
- The **Route Table Association** binds the “Public-Subnet” to this route table, ensuring that all instances in that subnet use the IGW for internet access.
- This is a fundamental step to make EC2 instances publicly reachable.



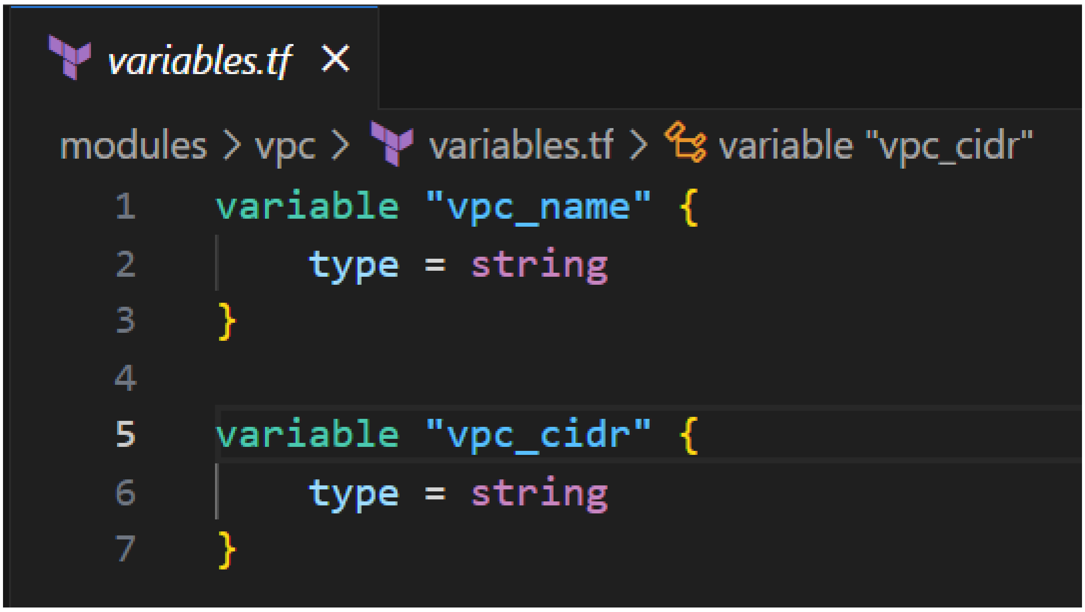
main.tf
modules > vpc > main.tf > resource "aws_security_group" "public_sg"

```
# Public Security Group
resource "aws_security_group" "public_sg" {
  name        = "public-sg"
  description = "Allows HTTP, SSH, and MySQL traffic from the internet"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "public-sg"
    Application = "WordPress"
  }
}
```

This block defines the **Security Group** for the public EC2 instance.

- It allows inbound connections on:
  - **Port 22 (SSH)** for secure remote access.
  - **Port 80 (HTTP)** for WordPress website access.
  - **Port 3306 (MySQL)** for database communication.
- Outbound traffic is unrestricted, allowing the instance to reach external services.
- The security group is tagged with the name "**public-sg**".

variables.tf:

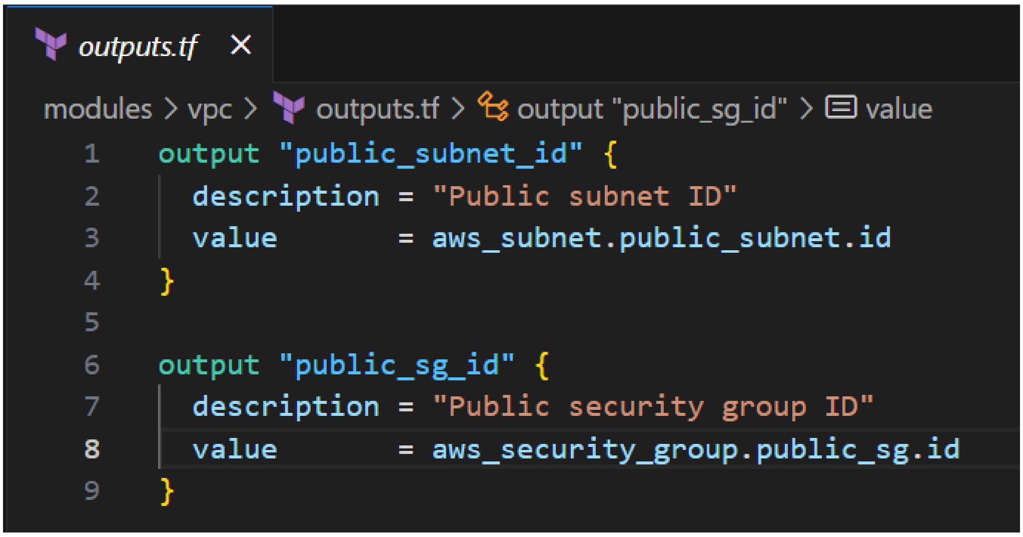


```hcl
variable "vpc_name" {
    type = string
}

variable "vpc_cidr" {
    type = string
}
```

These variables parameterize the VPC configuration:

- **vpc_name** allows the VPC name to be dynamically set
- **vpc_cidr** defines the CIDR block (e.g., 10.0.0.0/16) used for the VPC network.

outputs.tf:



```hcl
output "public_subnet_id" {
  description = "Public subnet ID"
  value       = aws_subnet.public_subnet.id
}

output "public_sg_id" {
  description = "Public security group ID"
  value       = aws_security_group.public_sg.id
}
```

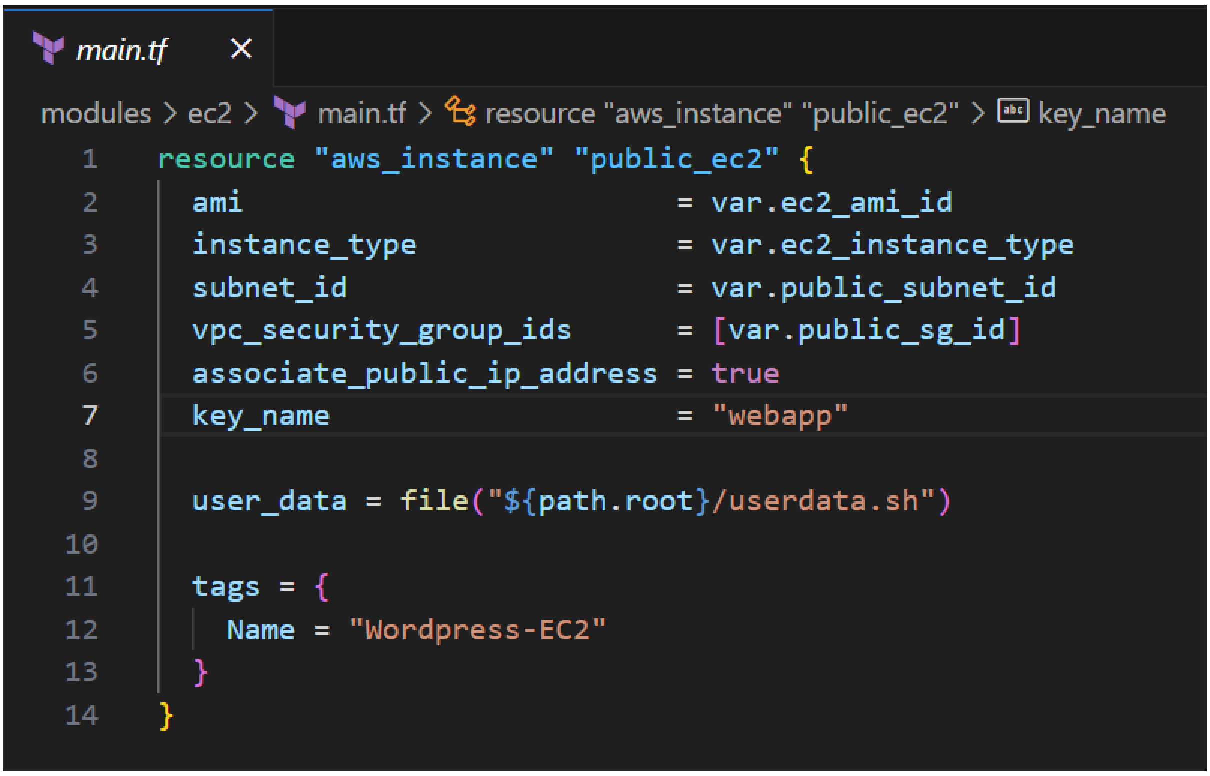
Outputs make key resource identifiers available to other modules (like EC2).

- **public_subnet_id** exposes the subnet's ID so it can be referenced by the EC2 instance.
- **public_sg_id** makes the security group ID accessible to other resources. This is how the VPC module integrates with the EC2 module.

Task 1.7: Configure EC2 Module

This section explains each Terraform configuration file located in the EC2 modules (/modules/ec2) directory.

main.tf:

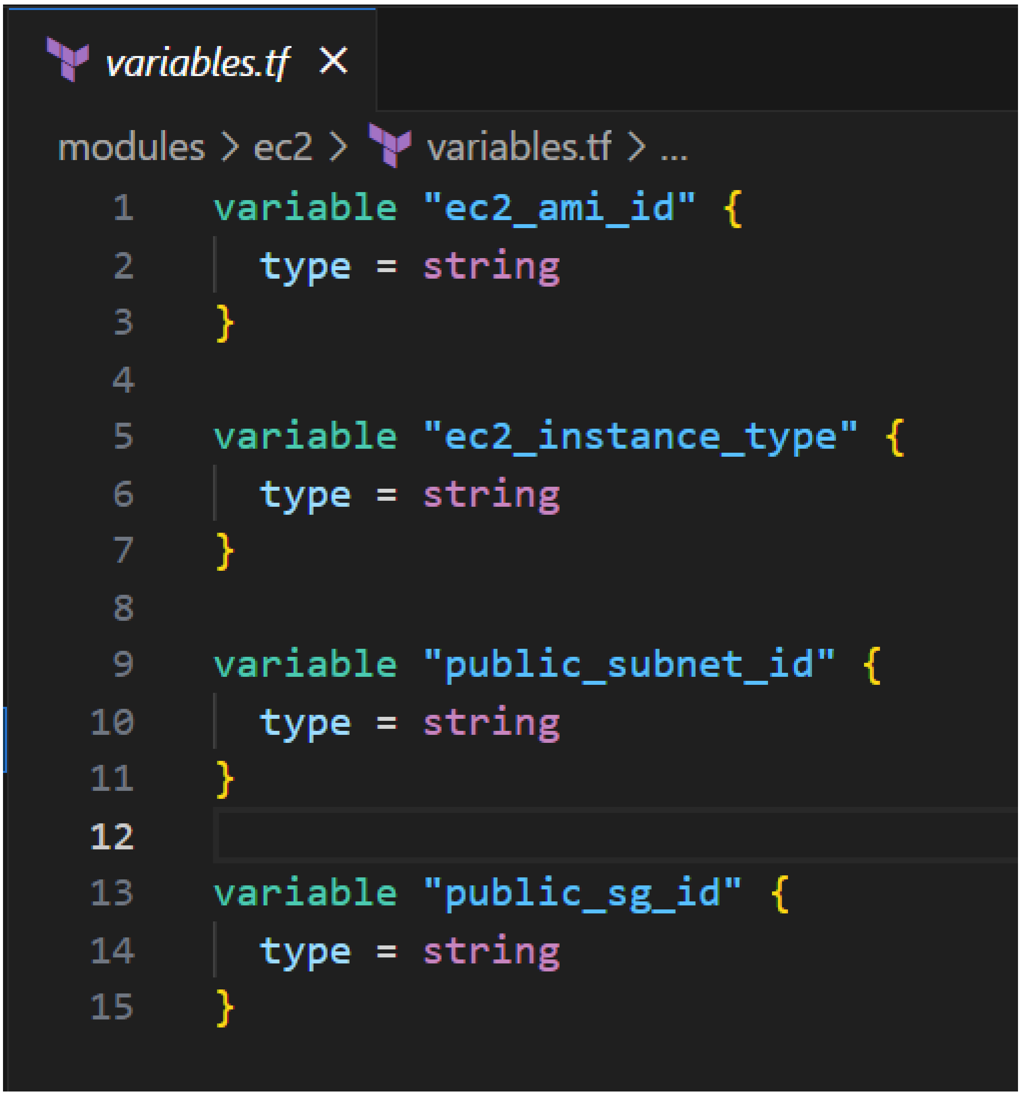

main.tf ×
modules > ec2 > main.tf > resource "aws_instance" "public_ec2" > key_name

```
resource "aws_instance" "public_ec2" {
  ami                         = var.ec2_ami_id
  instance_type               = var.ec2_instance_type
  subnet_id                   = var.public_subnet_id
  vpc_security_group_ids      = [var.public_sg_id]
  associate_public_ip_address = true
  key_name                    = "webapp"

  user_data = file("${path.root}/userdata.sh")

  tags = {
    Name = "Wordpress-EC2"
  }
}
```

This block provisions an **EC2 instance** that will host the WordPress application.

- Uses variables for AMI ID and instance type.
- The instance launches within the **Public Subnet** and is associated with the **Public Security Group** from the VPC module.
- **associate_public_ip_address = true** ensures that the instance receives a public IP, making it accessible over the internet.
- The **user_data** script automates installation and setup of Apache, PHP, MySQL, and WordPress on startup.
- The EC2 instance is tagged as “Wordpress-EC2” for easy identification.

variables.tf:



variables.tf ×

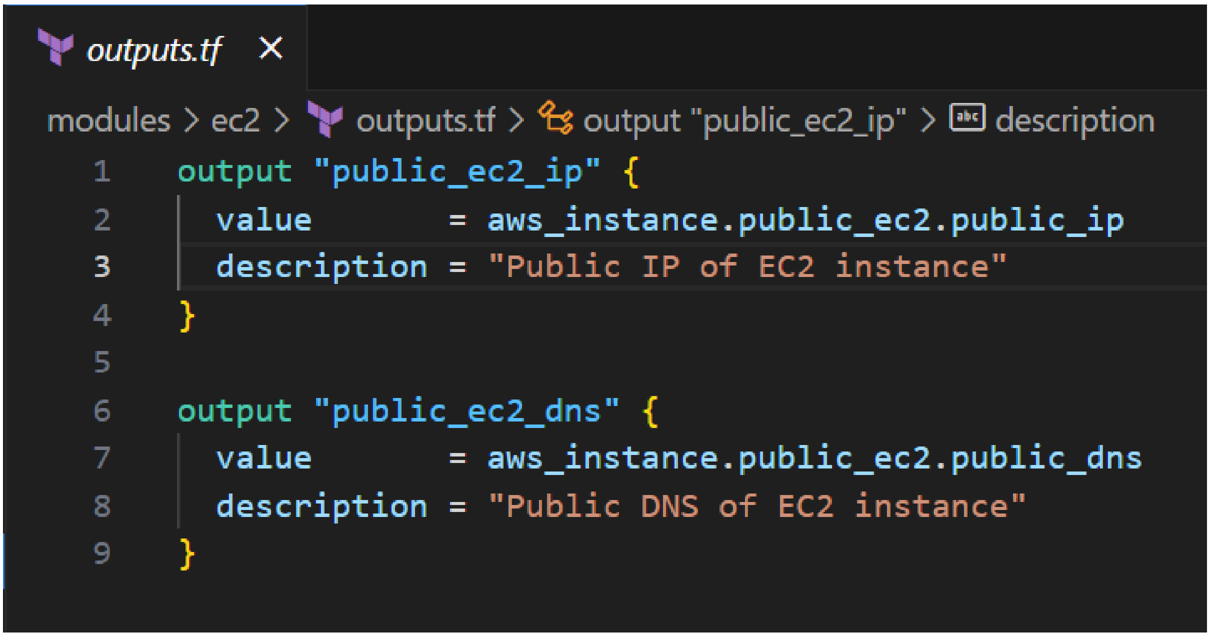
modules > ec2 > variables.tf > ...

```
variable "ec2_ami_id" {
  type = string
}

variable "ec2_instance_type" {
  type = string
}

variable "public_subnet_id" {
  type = string
}

variable "public_sg_id" {
  type = string
}
```

These variables define parameters required for EC2 deployment:

- **ec2_ami_id** specifies the AMI image used for instance creation.
- **ec2_instance_type** controls the instance's compute capacity.
- **public_subnet_id** and **public_sg_id** inherit values from the VPC module outputs, enabling proper network placement and security configuration.

outputs.tf:



outputs.tf ×
modules > ec2 > outputs.tf > output "public_ec2_ip" > description

```
output "public_ec2_ip" {
  value       = aws_instance.public_ec2.public_ip
  description = "Public IP of EC2 instance"
}

output "public_ec2_dns" {
  value       = aws_instance.public_ec2.public_dns
  description = "Public DNS of EC2 instance"
}
```

These outputs expose connection details of the deployed EC2 instance:

- **public_ec2_ip** displays the instance’s public IP address for direct access.
- **public_ec2_dns** provides the public DNS name, which can be used to access WordPress via a browser.
- These outputs make it easy to verify and connect to the running instance after terraform apply.

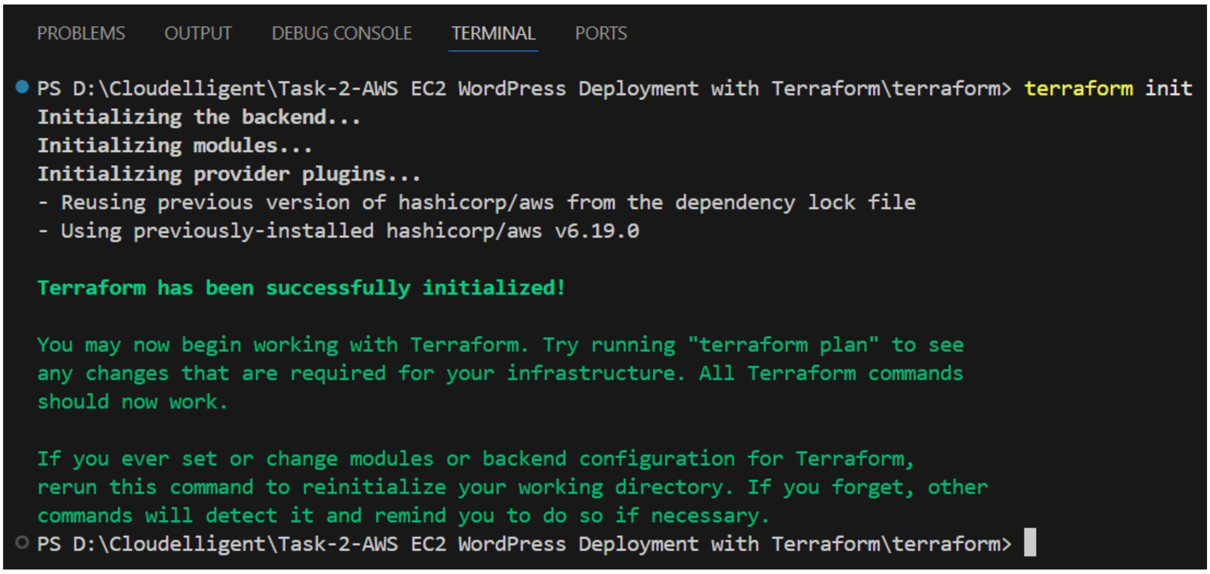
Task 1.8: Running Terraform Commands to Deploy Infrastructure

This section documents the series of Terraform CLI commands executed to deploy, validate, and destroy the WordPress environment.

Note: To perform this task, the user must be in the root directory of Terraform project where the provider.tf and the root main.tf files are stored.

Step 1: terraform init

Initializes the working directory by downloading the required provider plugins and connecting them to the configured backend (S3).

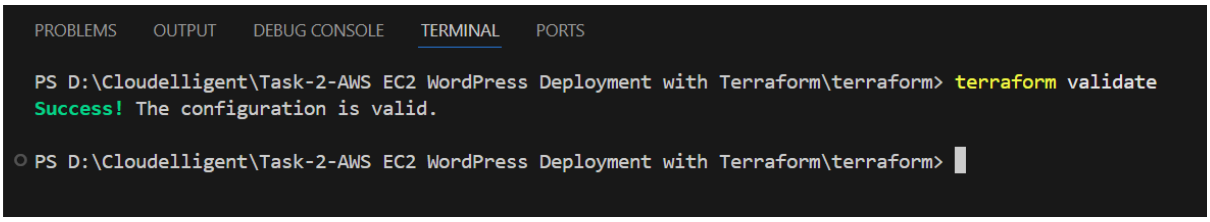


```
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform> terraform init
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.19.0

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-2-AWS EC2 WordPress Deployment with Terraform\terraform>
```

By running this command, Terraform confirms initialization, backend setup, and provider readiness.

Step 2: terraform validate

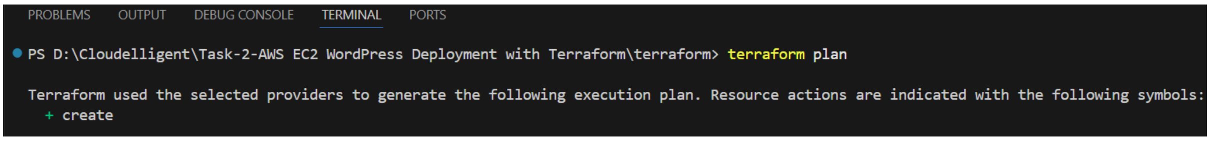
Performs a syntax and logic check on all configuration files in the directory. Outputs an error if the logic or syntax is incorrect.



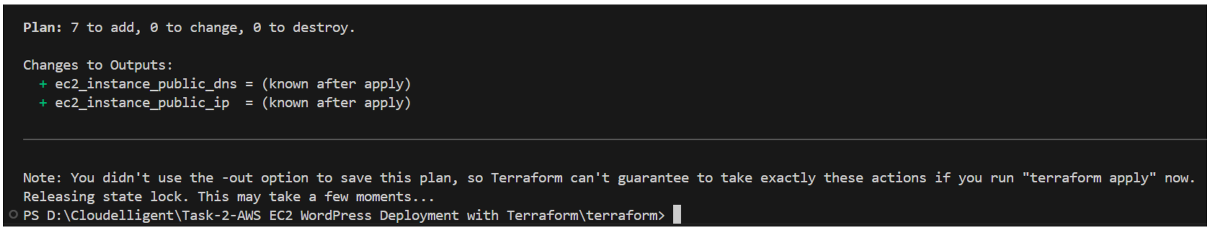
```
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform> terraform validate
Success! The configuration is valid.

PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform>
```

Step 3: terraform plan

Generates an execution plan showing all actions Terraform will perform to reach the desired state (resource creation, updates, or deletions).



```
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform> terraform plan

 Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
   + create
```

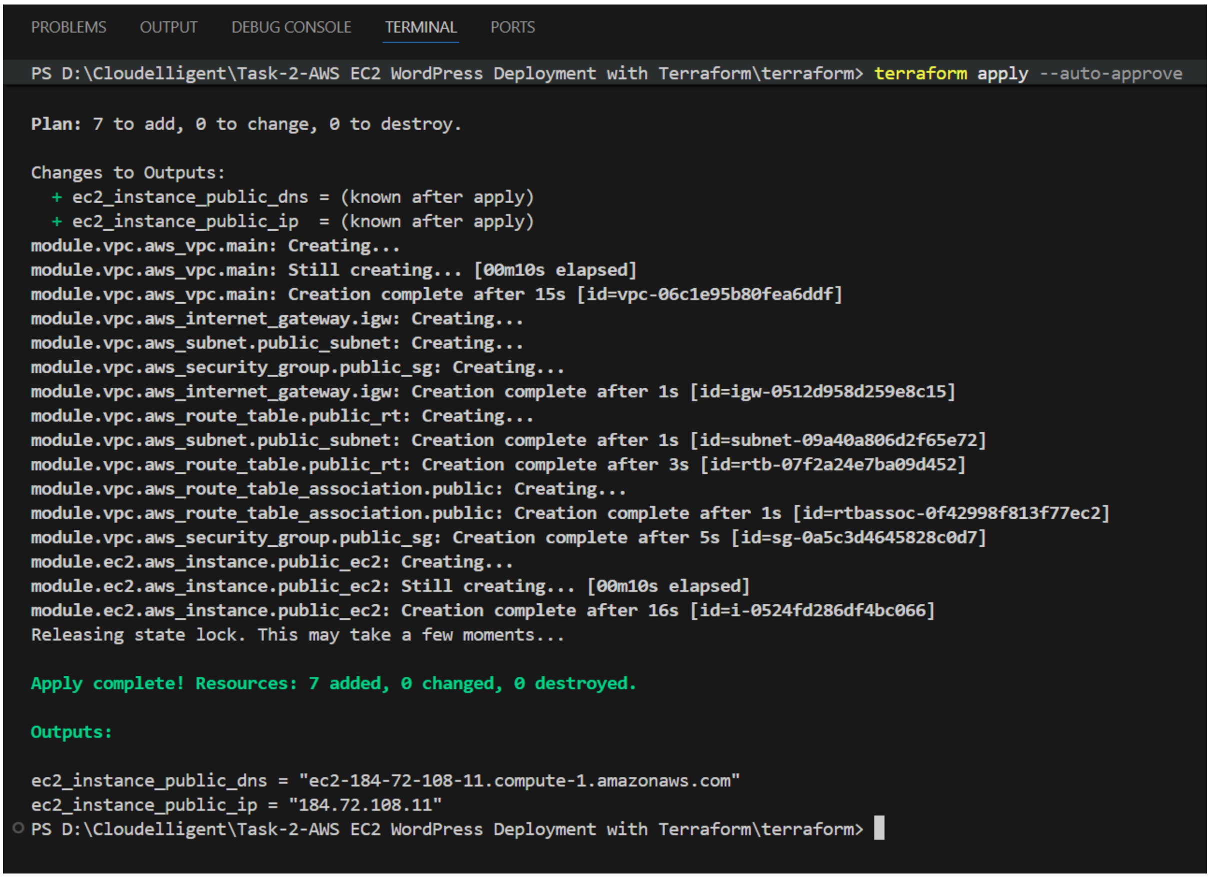


```
Plan: 7 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + ec2_instance_public_dns = (known after apply)
  + ec2_instance_public_ip  = (known after apply)

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
Releasing state lock. This may take a few moments...
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform>
```

Plan shows **7 to add, 0 to change, 0 to destroy**, confirming all required AWS resources are queued for creation.

Step 4: terraform apply –auto-approve (terraform apply)

Executes the plan and provisions the resources defined in the Terraform configuration without manual confirmation.

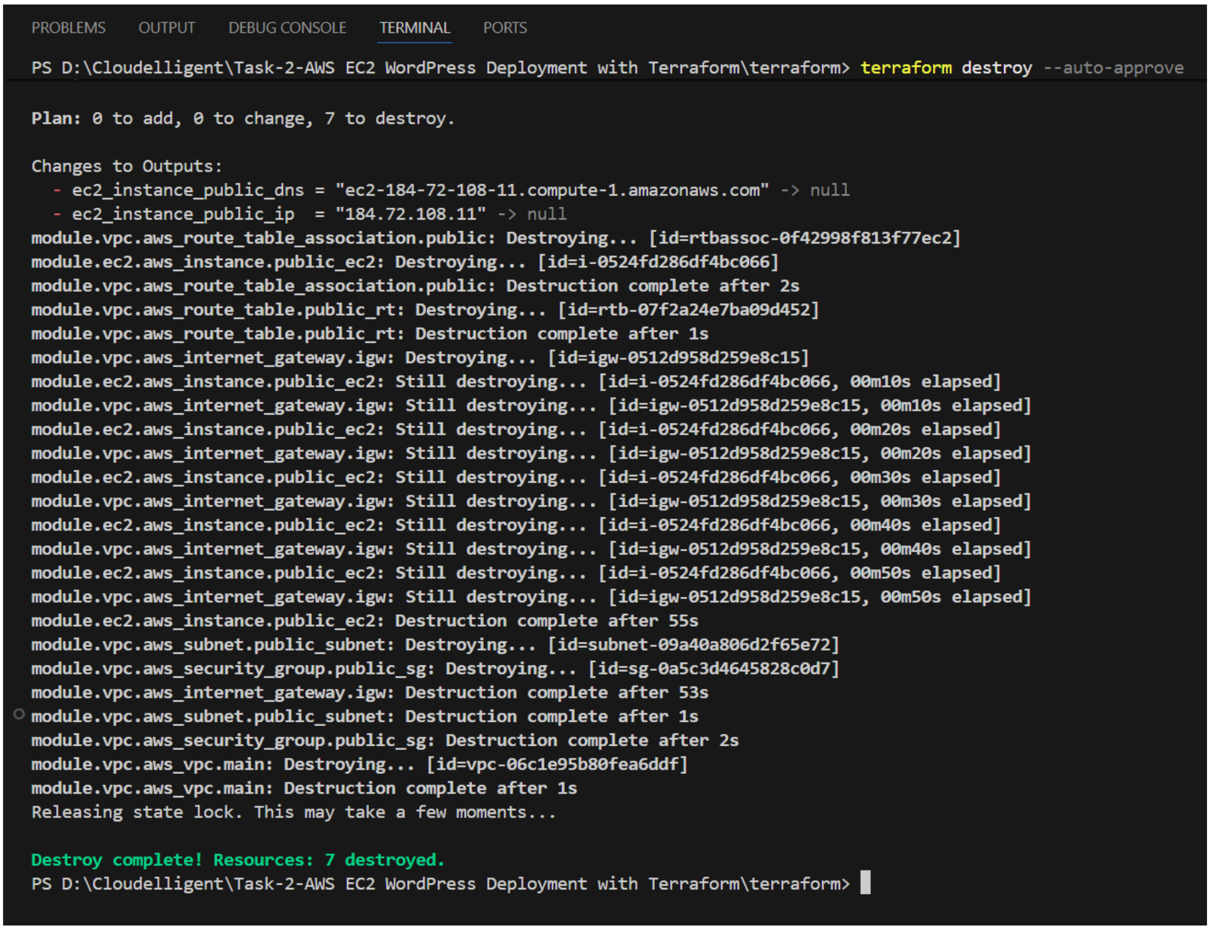


```
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform> terraform apply --auto-approve

Plan: 7 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + ec2_instance_public_dns = (known after apply)
  + ec2_instance_public_ip  = (known after apply)
module.vpc.aws_vpc.main: Creating...
module.vpc.aws_vpc.main: Still creating... [00m10s elapsed]
module.vpc.aws_vpc.main: Creation complete after 15s [id=vpc-06c1e95b80fea6ddf]
module.vpc.aws_internet_gateway.igw: Creating...
module.vpc.aws_subnet.public_subnet: Creating...
module.vpc.aws_security_group.public_sg: Creating...
module.vpc.aws_internet_gateway.igw: Creation complete after 1s [id=igw-0512d958d259e8c15]
module.vpc.aws_route_table.public_rt: Creating...
module.vpc.aws_subnet.public_subnet: Creation complete after 1s [id=subnet-09a40a806d2f65e72]
module.vpc.aws_route_table.public_rt: Creation complete after 3s [id=rtb-07f2a24e7ba09d452]
module.vpc.aws_route_table_association.public: Creating...
module.vpc.aws_route_table_association.public: Creation complete after 1s [id=rtbassoc-0f42998f813f77ec2]
module.vpc.aws_security_group.public_sg: Creation complete after 5s [id=sg-0a5c3d4645828c0d7]
module.ec2.aws_instance.public_ec2: Creating...
module.ec2.aws_instance.public_ec2: Still creating... [00m10s elapsed]
module.ec2.aws_instance.public_ec2: Creation complete after 16s [id=i-0524fd286df4bc066]
Releasing state lock. This may take a few moments...

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

ec2_instance_public_dns = "ec2-184-72-108-11.compute-1.amazonaws.com"
ec2_instance_public_ip = "184.72.108.11"
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform>
```

Resources (VPC, Subnet, Security Group, EC2, etc.) are created successfully, followed by public IP and DNS outputs for WordPress instance. Additionally, the user can open their browser and navigate to the EC2 Public IP/DNS to view the live WordPress site.

Step 5: terraform destroy --auto-approve (or terraform destroy)

Destroys all resources previously created by Terraform, cleaning up the AWS environment.



```
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform> terraform destroy --auto-approve

Plan: 0 to add, 0 to change, 7 to destroy.

Changes to Outputs:
  - ec2_instance_public_dns = "ec2-184-72-108-11.compute-1.amazonaws.com" -> null
  - ec2_instance_public_ip  = "184.72.108.11" -> null
module.vpc.aws_route_table_association.public: Destroying... [id=rtbassoc-0f42998f813f77ec2]
module.ec2.aws_instance.public_ec2: Destroying... [id=i-0524fd286df4bc066]
module.vpc.aws_route_table_association.public: Destruction complete after 2s
module.vpc.aws_route_table.public_rt: Destroying... [id=rtb-07f2a24e7ba09d452]
module.vpc.aws_route_table.public_rt: Destruction complete after 1s
module.vpc.aws_internet_gateway.igw: Destroying... [id=igw-0512d958d259e8c15]
module.ec2.aws_instance.public_ec2: Still destroying... [id=i-0524fd286df4bc066, 00m10s elapsed]
module.vpc.aws_internet_gateway.igw: Still destroying... [id=igw-0512d958d259e8c15, 00m10s elapsed]
module.ec2.aws_instance.public_ec2: Still destroying... [id=i-0524fd286df4bc066, 00m20s elapsed]
module.vpc.aws_internet_gateway.igw: Still destroying... [id=igw-0512d958d259e8c15, 00m20s elapsed]
module.ec2.aws_instance.public_ec2: Still destroying... [id=i-0524fd286df4bc066, 00m30s elapsed]
module.vpc.aws_internet_gateway.igw: Still destroying... [id=igw-0512d958d259e8c15, 00m30s elapsed]
module.ec2.aws_instance.public_ec2: Still destroying... [id=i-0524fd286df4bc066, 00m40s elapsed]
module.vpc.aws_internet_gateway.igw: Still destroying... [id=igw-0512d958d259e8c15, 00m40s elapsed]
module.ec2.aws_instance.public_ec2: Still destroying... [id=i-0524fd286df4bc066, 00m50s elapsed]
module.vpc.aws_internet_gateway.igw: Still destroying... [id=igw-0512d958d259e8c15, 00m50s elapsed]
module.ec2.aws_instance.public_ec2: Destruction complete after 55s
module.vpc.aws_subnet.public_subnet: Destroying... [id=subnet-09a40a806d2f65e72]
module.vpc.aws_security_group.public_sg: Destroying... [id=sg-0a5c3d4645828c0d7]
module.vpc.aws_internet_gateway.igw: Destruction complete after 53s
module.vpc.aws_subnet.public_subnet: Destruction complete after 1s
module.vpc.aws_security_group.public_sg: Destruction complete after 2s
module.vpc.aws_vpc.main: Destroying... [id=vpc-06c1e95b80fea6ddf]
module.vpc.aws_vpc.main: Destruction complete after 1s
Releasing state lock. This may take a few moments...

Destroy complete! Resources: 7 destroyed.
PS D:\Cloudelligent\Task-2-AWS EC2 WordPress Deployment with Terraform\terraform>
```

Displays **Plan: 0 to add, 0 to change, 7 to destroy** and confirm complete deletion of all deployed resources.

The state file in S3 will also be updated automatically to reflect the destroyed state.

Task 1.9: Validate Infrastructure Deployment and WordPress Installation

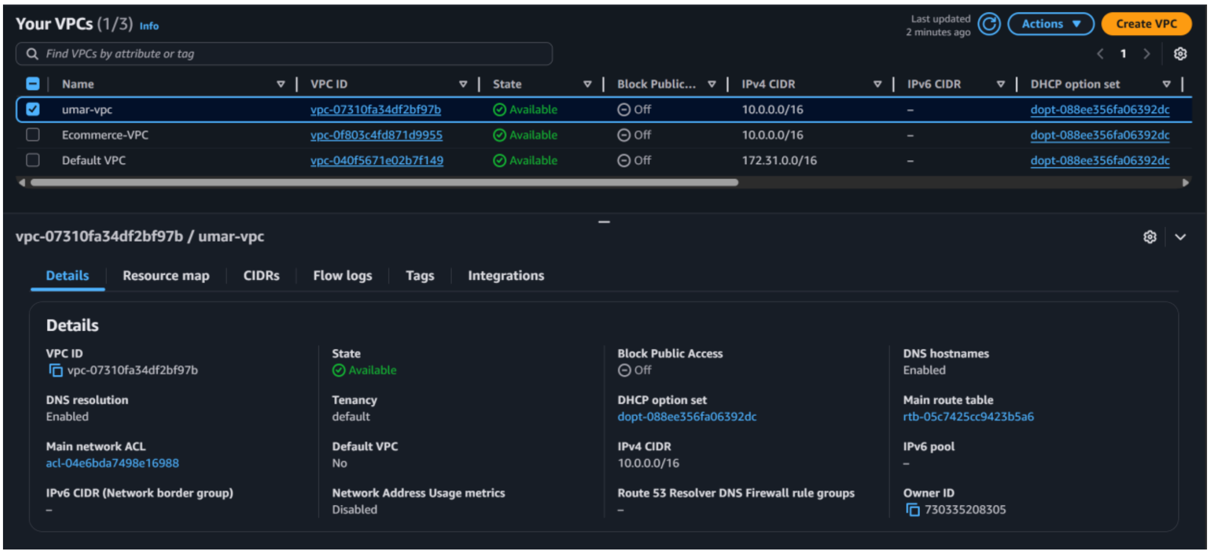
This task demonstrates the successful deployment and verification of all AWS and application-level components created using Terraform. It includes screenshots of the AWS infrastructure, validation commands from the EC2 instance, and the full WordPress installation process.

Step 1: Verify Deployed AWS Infrastructure

After running **terraform apply** command, Terraform provisions all the required networking and compute resources. In this step, please confirm that these resources exist and are configured correctly.

Verify VPC Creation

1. Sign in to the AWS Management Console.
2. Navigate to **VPC** service using the search bar at the top and then click **Your VPCs.**
3. Verify that a VPC named **umar-vpc** has been created.
   - CIDR Block: 10.0.0.0/16
   - DNS Hostnames: Enabled
   - This VPC acts as the isolated network for your WordPress environment.

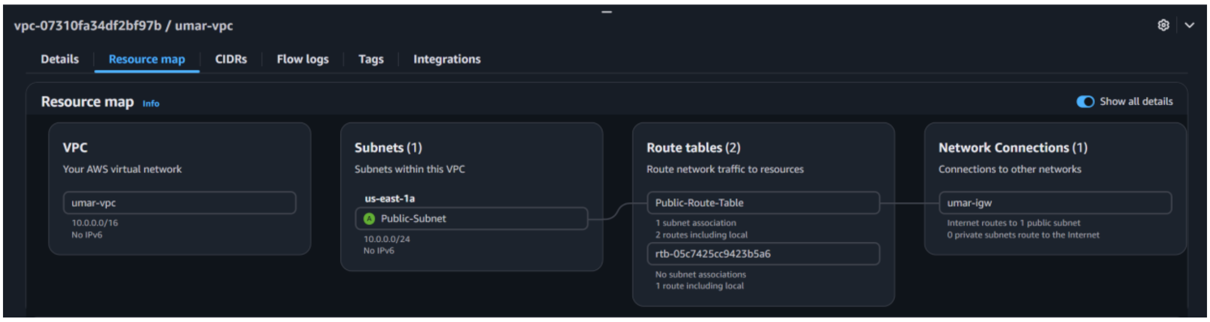


	Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR	DHCP option set
☑	umar-vpc	vpc-07310fa34df2bf97b	✓ Available	⊖ Off	10.0.0.0/16	–	dopt-088ee356fa06392dc
□	Ecommerce-VPC	vpc-0f803c4fd871d9955	✓ Available	⊖ Off	10.0.0.0/16	–	dopt-088ee356fa06392dc
□	Default VPC	vpc-040f5671e02b7f149	✓ Available	⊖ Off	172.31.0.0/16	–	dopt-088ee356fa06392dc

vpc-07310fa34df2bf97b / umar-vpc

Details

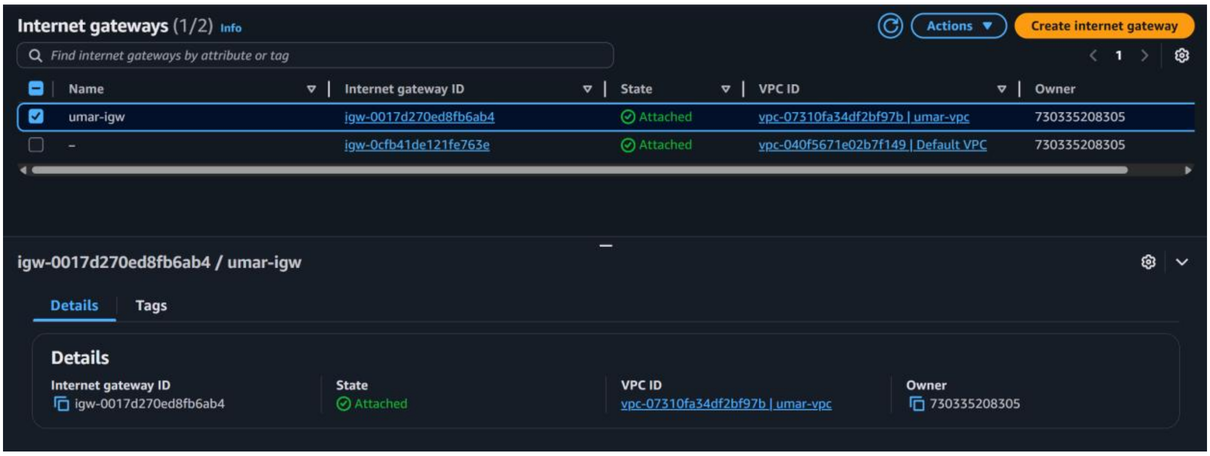
VPC ID vpc-07310fa34df2bf97b	State ✓ Available	Block Public Access ⊖ Off	DNS hostnames Enabled
DNS resolution Enabled	Tenancy default	DHCP option set dopt-088ee356fa06392dc	Main route table rtb-05c7425cc9423b5a6
Main network ACL acl-04e6bda7498e16988	Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool –
IPv6 CIDR (Network border group) –	Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups –	Owner ID 730335208305



vpc-07310fa34df2bf97b / umar-vpc

Verify Internet Gateway (IGW) Creation

1. In the VPC console, choose **Internet Gateways**.
2. Confirm that an Internet Gateway named **umar-igw** exists and is attached to your VPC. This gateway allows the EC2 instance to communicate with the internet.


Internet gateways (1/2) Info
Actions ▼
Create internet gateway
Find internet gateways by attribute or tag
Details
Tags

	Name	Internet gateway ID	State	VPC ID	Owner
☑	umar-igw	igw-0017d270ed8fb6ab4	Attached	vpc-07310fa34df2bf97b | umar-vpc	730335208305
☐	–	igw-0cfb41de121fe763e	Attached	vpc-040f5671e02b7f149 | Default VPC	730335208305

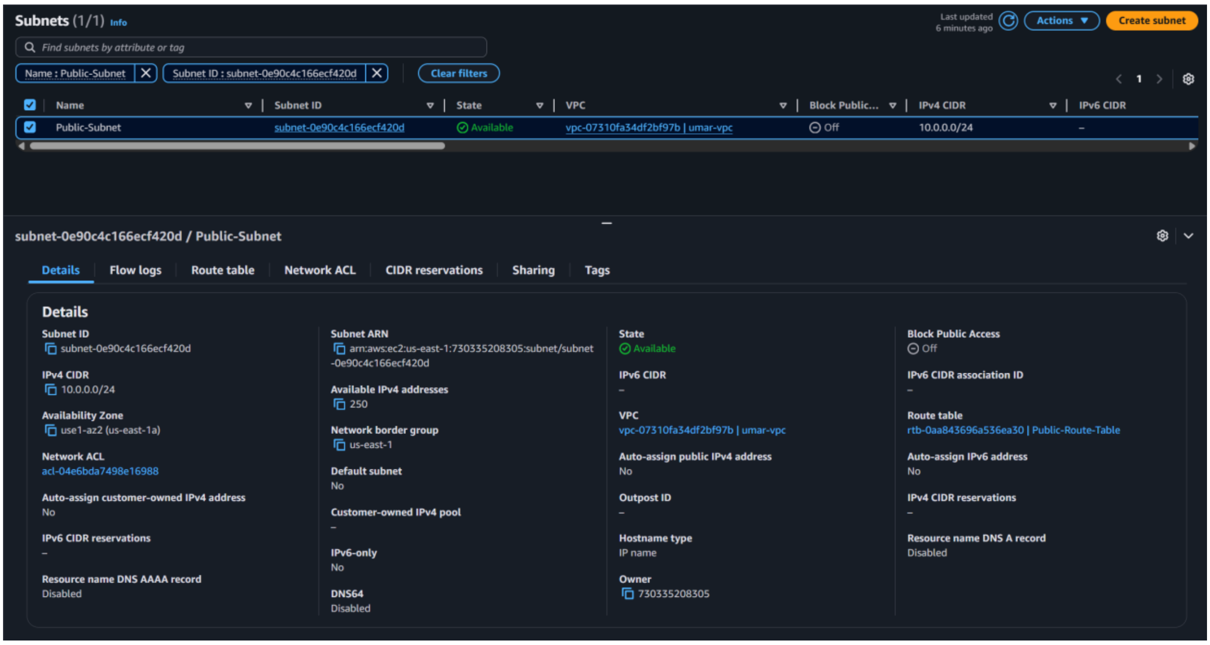
igw-0017d270ed8fb6ab4 / umar-igw

Details

Internet gateway ID: igw-0017d270ed8fb6ab4
State: Attached
VPC ID: vpc-07310fa34df2bf97b | umar-vpc
Owner: 730335208305

Verify Subnet Creation

1. In the VPC console, choose **Subnets located** on the left navigation panel.
2. Confirm that a Subnet named **Public-Subnet** exists with the IPv4 CIDR 10.0.0.0/24 and the same VPC ID (example: vpc-07310fa34df2bf97b).



	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR	IPv6 CIDR
☑	Public-Subnet	subnet-0e90c4c166ecf420d	Available	vpc-07310fa34df2bf97b | umar-vpc	Off	10.0.0.0/24	–

subnet-0e90c4c166ecf420d / Public-Subnet

Details

Subnet ID: subnet-0e90c4c166ecf420d
IPv4 CIDR: 10.0.0.0/24
Availability Zone: use1-az2 (us-east-1a)
Network ACL: acl-04e6bda7498e16988
Auto-assign customer-owned IPv4 address: No
IPv6 CIDR reservations: –
Resource name DNS AAAA record: Disabled

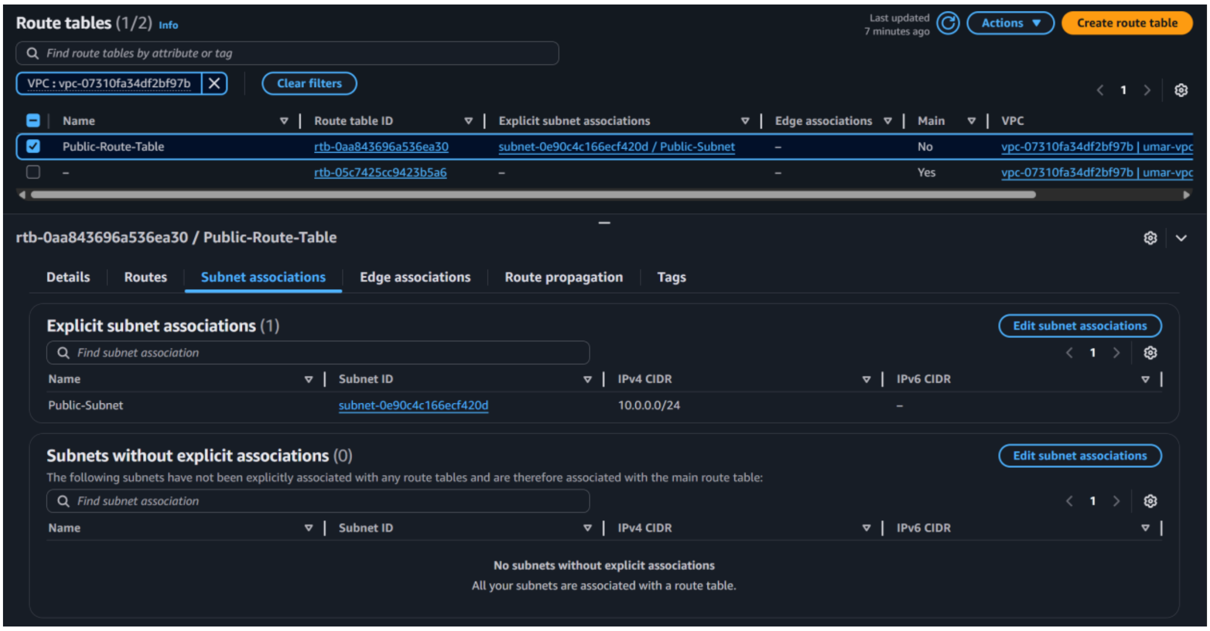
Subnet ARN: arn:aws:ec2:us-east-1:730335208305:subnet/subnet-0e90c4c166ecf420d
Available IPv4 addresses: 250
Network border group: us-east-1
Default subnet: No
Customer-owned IPv4 pool: –
IPv6-only: No
DNS64: Disabled

State: Available
IPv6 CIDR: –
VPC: vpc-07310fa34df2bf97b | umar-vpc
Auto-assign public IPv4 address: No
Outpost ID: –
Hostname type: IP name
Owner: 730335208305

Block Public Access: Off
IPv6 CIDR association ID: –
Route table: rtb-0aa843696a536ea30 | Public-Route-Table
Auto-assign IPv6 address: No
IPv4 CIDR reservations: –
Resource name DNS A record: Disabled

Verify Route Table and Routes Creation

1. In the VPC console, choose **Route tables** located on the left navigation panel.
2. Select the route table named **Public-Route-Table**.
3. Check that it contains a route to the Internet Gateway with the destination **0.0.0.0/0**.
4. Under **Subnet Associations**, verify that the **Public Subnet** is associated with this route table as shown under **Explicit subnet associations (1)**.



Route tables (1/2) Info

	Name	Route table ID	Explicit subnet associations	Edge associations	Main	VPC
☑	Public-Route-Table	rtb-0aa843696a536ea30	subnet-0e90c4c166ecf420d / Public-Subnet	–	No	vpc-07310fa34df2bf97b | umar-vpc
□	–	rtb-05c7425cc9423b5a6	–	–	Yes	vpc-07310fa34df2bf97b | umar-vpc

rtb-0aa843696a536ea30 / Public-Route-Table

Explicit subnet associations (1)

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
Public-Subnet	subnet-0e90c4c166ecf420d	10.0.0.0/24	–

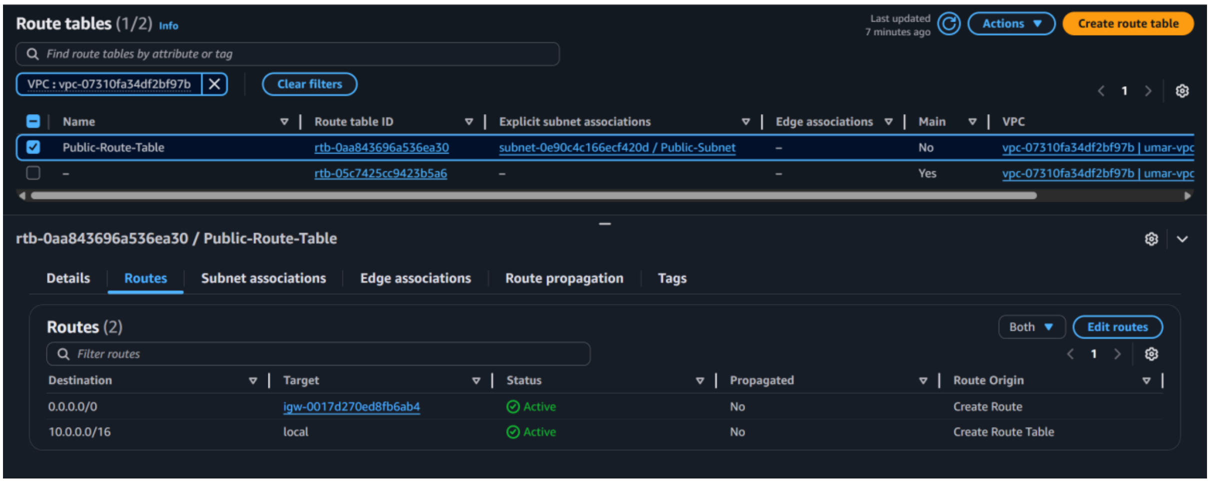
Subnets without explicit associations (0)

The following subnets have not been explicitly associated with any route tables and are therefore associated with the main route table:

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR

No subnets without explicit associations

All your subnets are associated with a route table.



Route tables (1/2) Info

	Name	Route table ID	Explicit subnet associations	Edge associations	Main	VPC
☑	Public-Route-Table	rtb-0aa843696a536ea30	subnet-0e90c4c166ecf420d / Public-Subnet	–	No	vpc-07310fa34df2bf97b | umar-vpc
□	–	rtb-05c7425cc9423b5a6	–	–	Yes	vpc-07310fa34df2bf97b | umar-vpc

rtb-0aa843696a536ea30 / Public-Route-Table

Routes (2)

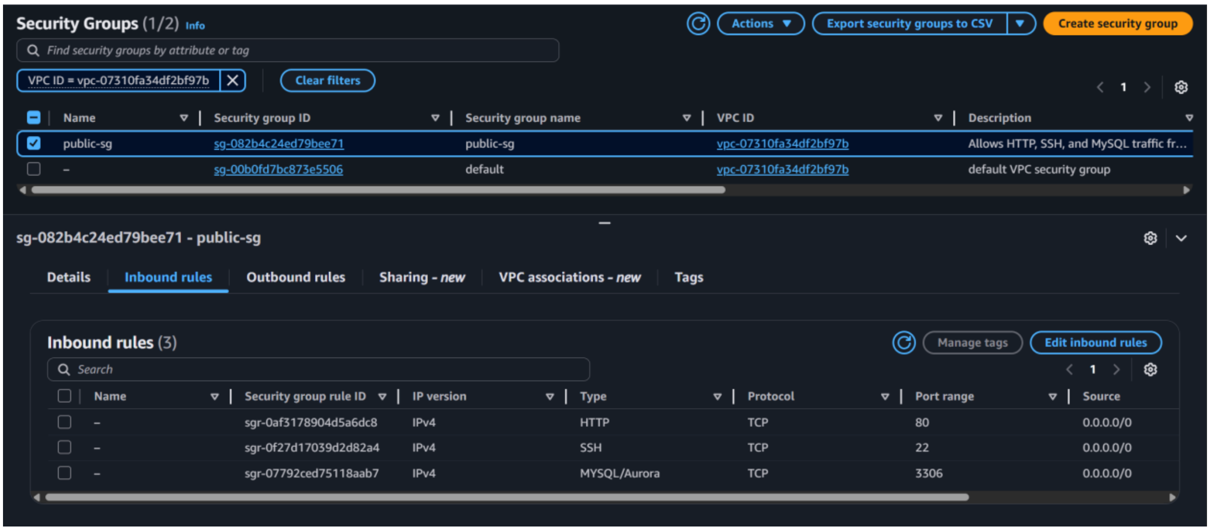
Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	igw-0017d270ed8fb6ab4	✓ Active	No	Create Route
10.0.0.0/16	local	✓ Active	No	Create Route Table

Verify Security Group Creation

1. In the VPC console, choose **Security groups** located on the left navigation panel.
2. Locate the **public-sg** security group
3. It should have the following inbound rules:

Type	Protocol	Port range	Source
HTTP	TCP	80	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
MySQL/Aurora	TCP	3306	0.0.0.0/0

4. Outbound rules should allow all traffic.


Security Groups (1/2) Info
Actions
Export security groups to CSV
Create security group
Find security groups by attribute or tag
VPC ID = vpc-07310fa34df2bf97b
Clear filters

☐	Name	Security group ID	Security group name	VPC ID	Description
☑	public-sg	sg-082b4c24ed79bee71	public-sg	vpc-07310fa34df2bf97b	Allows HTTP, SSH, and MySQL traffic fr...
☐	–	sg-00b0fd7bc873e5506	default	vpc-07310fa34df2bf97b	default VPC security group

sg-082b4c24ed79bee71 - public-sg

☐	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
☐	–	sgr-0af3178904d5a6dc8	IPv4	HTTP	TCP	80	0.0.0.0/0
☐	–	sgr-0f27d17039d2d82a4	IPv4	SSH	TCP	22	0.0.0.0/0
☐	–	sgr-07792ced75118aab7	IPv4	MYSQL/Aurora	TCP	3306	0.0.0.0/0

At this point, your VPC, Internet Gateway, subnet, route table, routes and security group are confirmed as functional.

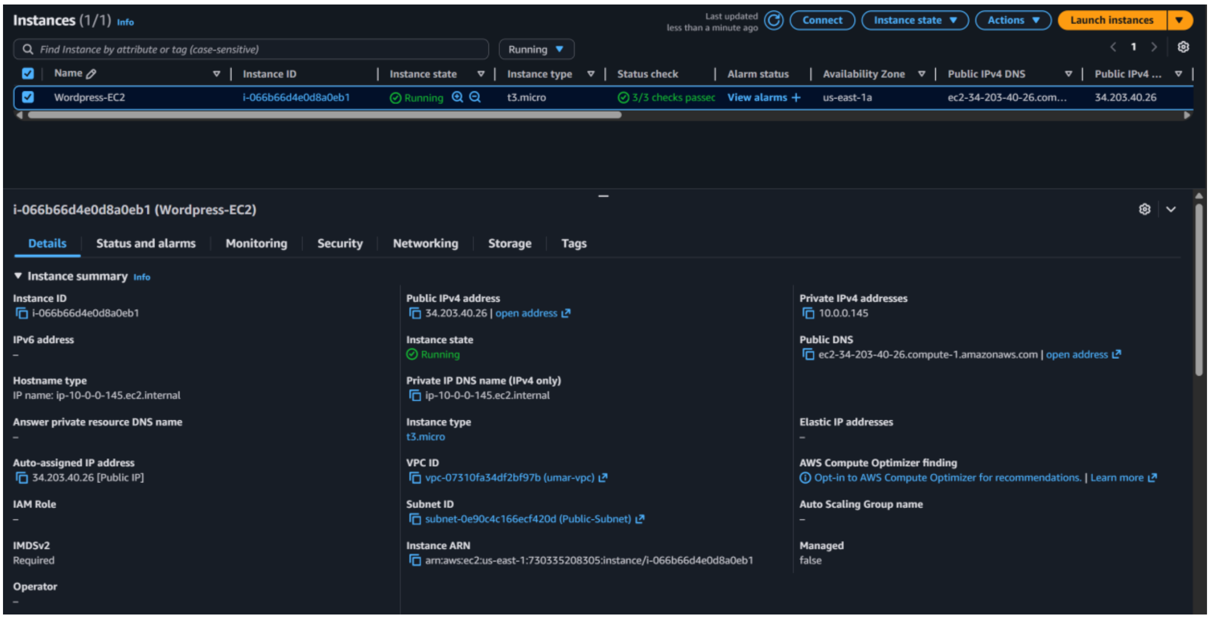
Step 2: Verify EC2 Instance Deployment

Next, verify that Terraform successfully created the EC2 instance and that it is running.

To verify EC2 deployment status

1. In the AWS Console, open EC2 service using the search bar and click **Instances** located on the left navigation panel.
2. Confirm that an instance named **WordPress-EC2** is in the Running state.
   - Instance Type: t3.micro
   - AMI: Ubuntu Server 24.04
   - Subnet: Public-Subnet
   - Public IP: Automatically assigned (example: 34.203.40.26)

This instance hosts the WordPress website.


Instances (1/1) Info
Last updated less than a minute ago
Connect
Instance state ▼
Actions ▼
Launch instances ▼
Find Instance by attribute or tag (case-sensitive)
Running ▼
1

☑	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
☑	Wordpress-EC2	i-066b66d4e0d8a0eb1	✓ Running	t3.micro	✓ 3/3 checks passec	View alarms +	us-east-1a	ec2-34-203-40-26.com...	34.203.40.26

i-066b66d4e0d8a0eb1 (Wordpress-EC2)

Instance ID
i-066b66d4e0d8a0eb1

IPv6 address
–

Hostname type
IP name: ip-10-0-0-145.ec2.internal

Answer private resource DNS name
–

Auto-assigned IP address
34.203.40.26 [Public IP]

IAM Role
–

IMDSv2
Required

Operator
–

Public IPv4 address
34.203.40.26 | open address

Instance state
✓ Running

Private IP DNS name (IPv4 only)
ip-10-0-0-145.ec2.internal

Instance type
t3.micro

VPC ID
vpc-07310fa34df2bf97b (umar-vpc)

Subnet ID
subnet-0e90c4c166ecf420d (Public-Subnet)

Instance ARN
arn:aws:ec2:us-east-1:730335208305:instance/i-066b66d4e0d8a0eb1

Private IPv4 addresses
10.0.0.145

Public DNS
ec2-34-203-40-26.compute-1.amazonaws.com | open address

Elastic IP addresses
–

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name
–

Managed
false

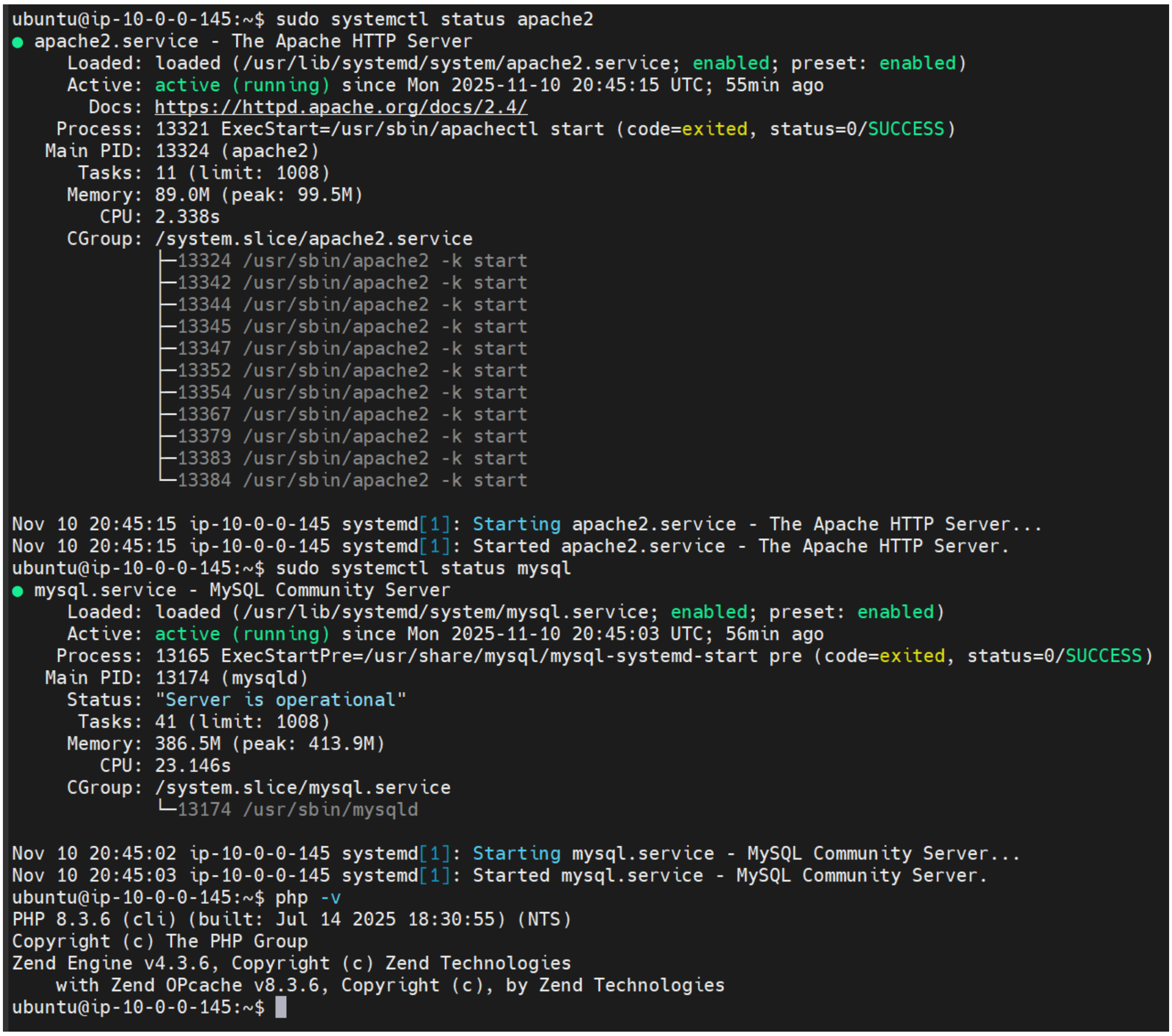
Step 3: Verify Services Running on the EC2 Instance

Terraform's user data script automatically installs Apache, PHP, and MySQL during instance boot. The next step is to SSH into the EC2 instance and verify that all services are running.

Connect to EC2 Instance

1. For this to work, it is important to have the key pair saved on local machine.
2. Make sure the key pair (.pem file) has necessary permission. If not, run "chmod 400 <keypair.pem>
3. Once that is done, use the following command to SSH into the instance:
   - ssh -i "webapp.pem" ubuntu@<your-ec2-public-ip>
   - Example: ssh -i "webapp.pem" ubuntu@ 34.203.40.26

4. Once connected to the instance, verify that the Apache, PHP, and MySQL are installed. To verify services, run the following commands:
   - sudo systemctl status apache2
   - sudo systemctl status mysql
   - php -v
5. Expected results
   - Apache2 status shows **active (running).**
   - MySQL service shows **active (running).**
   - PHP version is displayed (e.g., PHP 8.3.6).



```
ubuntu@ip-10-0-0-145:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-11-10 20:45:15 UTC; 55min ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 13321 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 13324 (apache2)
      Tasks: 11 (limit: 1008)
     Memory: 89.0M (peak: 99.5M)
        CPU: 2.338s
     CGroup: /system.slice/apache2.service
             ├─13324 /usr/sbin/apache2 -k start
             ├─13342 /usr/sbin/apache2 -k start
             ├─13344 /usr/sbin/apache2 -k start
             ├─13345 /usr/sbin/apache2 -k start
             ├─13347 /usr/sbin/apache2 -k start
             ├─13352 /usr/sbin/apache2 -k start
             ├─13354 /usr/sbin/apache2 -k start
             ├─13367 /usr/sbin/apache2 -k start
             ├─13379 /usr/sbin/apache2 -k start
             ├─13383 /usr/sbin/apache2 -k start
             └─13384 /usr/sbin/apache2 -k start

Nov 10 20:45:15 ip-10-0-0-145 systemd[1]: Starting apache2.service - The Apache HTTP Server...
Nov 10 20:45:15 ip-10-0-0-145 systemd[1]: Started apache2.service - The Apache HTTP Server.
ubuntu@ip-10-0-0-145:~$ sudo systemctl status mysql
● mysql.service - MySQL Community Server
     Loaded: loaded (/usr/lib/systemd/system/mysql.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-11-10 20:45:03 UTC; 56min ago
    Process: 13165 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
   Main PID: 13174 (mysqld)
     Status: "Server is operational"
      Tasks: 41 (limit: 1008)
     Memory: 386.5M (peak: 413.9M)
        CPU: 23.146s
     CGroup: /system.slice/mysql.service
             └─13174 /usr/sbin/mysqld

Nov 10 20:45:02 ip-10-0-0-145 systemd[1]: Starting mysql.service - MySQL Community Server...
Nov 10 20:45:03 ip-10-0-0-145 systemd[1]: Started mysql.service - MySQL Community Server.
ubuntu@ip-10-0-0-145:~$ php -v
PHP 8.3.6 (cli) (built: Jul 14 2025 18:30:55) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.3.6, Copyright (c) Zend Technologies
    with Zend OPcache v8.3.6, Copyright (c), by Zend Technologies
ubuntu@ip-10-0-0-145:~$ 
```

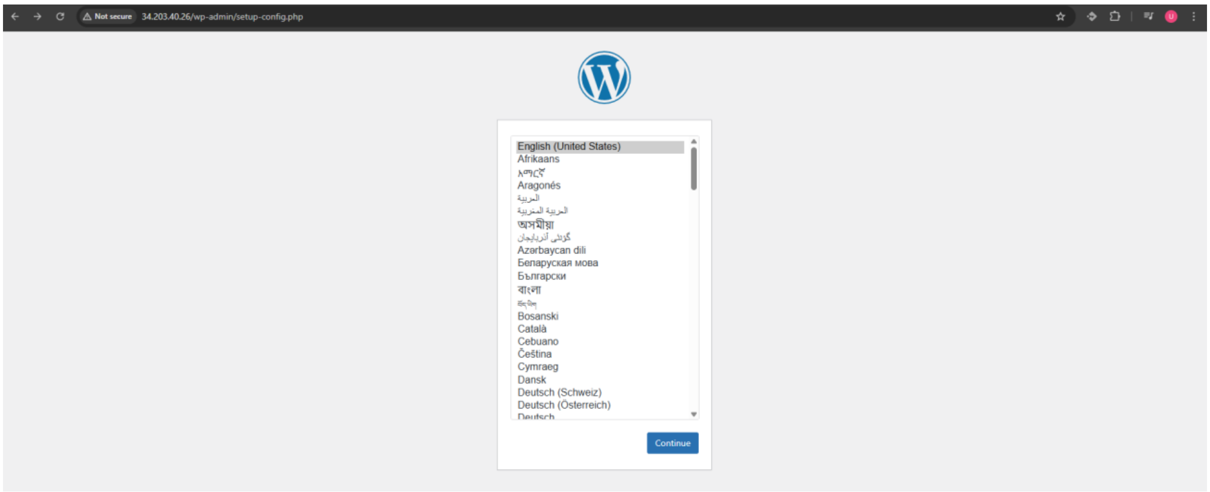
These results confirm that Apache, MySQL, and PHP are installed and the EC2 instance is fully configured to host WordPress.

Step 4: Access and Configure WordPress

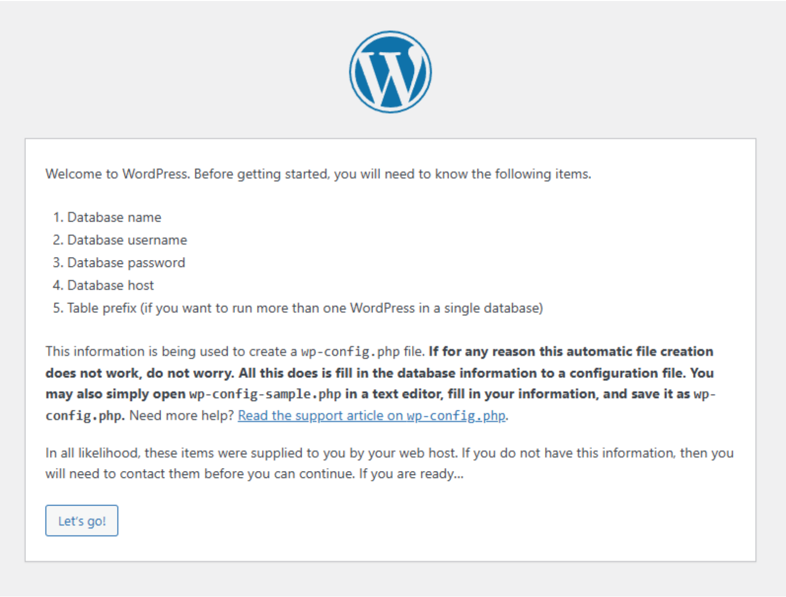
Now that the infrastructure and services are operational, access the WordPress setup page in your browser.

To open WordPress setup:

1. In browser, go to **http://<your-ec2-public-ip>/wp-admin/setup-config.php**
    - Example: http://34.203.40.26/wp-admin/setup-config.php



2. **“Welcome to WordPress”** setup screen should display. Select **English** and click **Continue** to proceed.
3. Click “Let’s go!” button to proceed.



Welcome to WordPress. Before getting started, you will need to know the following items.

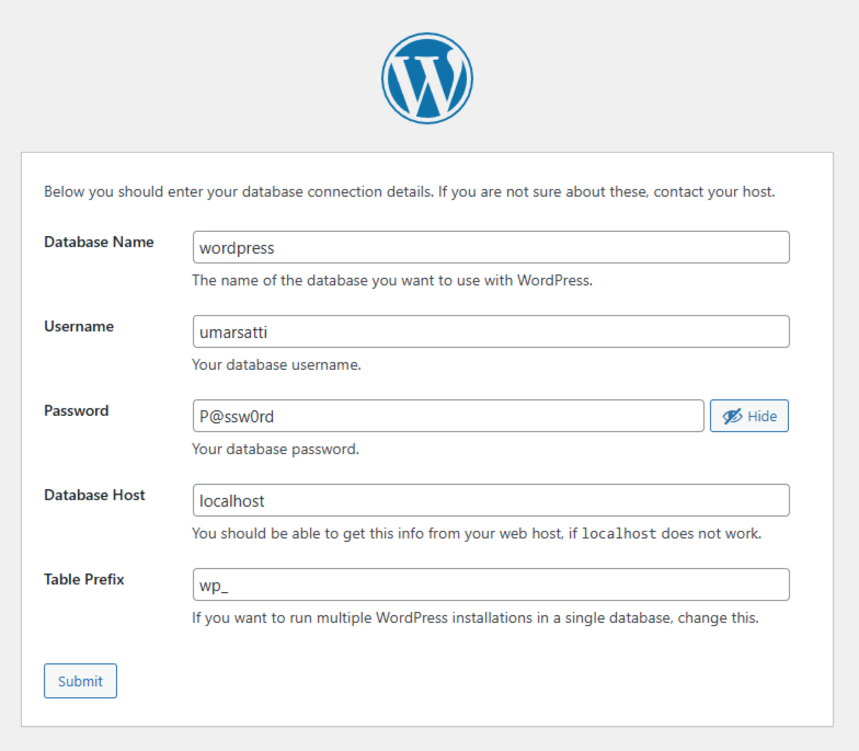
1. Database name
2. Database username
3. Database password
4. Database host
5. Table prefix (if you want to run more than one WordPress in a single database)

This information is being used to create a `wp-config.php` file. **If for any reason this automatic file creation does not work, do not worry. All this does is fill in the database information to a configuration file. You may also simply open `wp-config-sample.php` in a text editor, fill in your information, and save it as `wp-config.php`.** Need more help? Read the support article on `wp-config.php`.

In all likelihood, these items were supplied to you by your web host. If you do not have this information, then you will need to contact them before you can continue. If you are ready…

To provide database connection details:

1. On the Database Configuration page, enter the following:
   - Database Name: wordpress
   - Username: umarsatti
   - Password: P@ssw0rd
   - Database Host: localhost
   - Table Prefix: _wp
2. Click **Submit** button. If successful, WordPress will confirm that it can connect to the database.



Below you should enter your database connection details. If you are not sure about these, contact your host.

Database Name: wordpress
The name of the database you want to use with WordPress.

Username: umarsatti
Your database username.

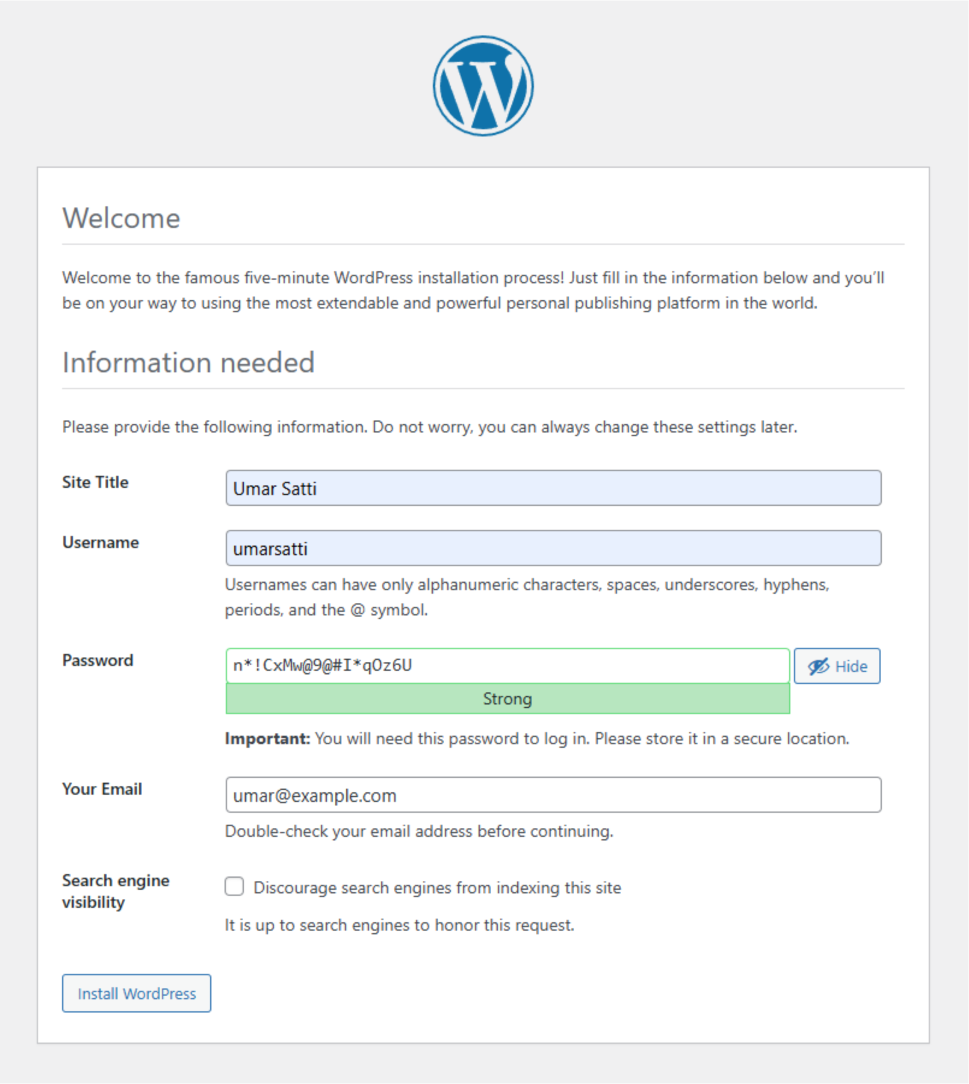
Password: P@ssw0rd
Your database password.

Database Host: localhost
You should be able to get this info from your web host, if `localhost` does not work.

Table Prefix: wp_
If you want to run multiple WordPress installations in a single database, change this.

Step 5: Install WordPress

1. Once WordPress verifies database connection, it will display the installation page. To complete installation, fill out the required fields:
   - Site Title: Umar Satti
   - Username: umarsatti
   - Password: n*!CxMw@9@#I*qOz6U
   - Email: umar@example.com
2. Click Install WordPress.
3. After a few seconds, a success message is displayed:
   - “Success! WordPress has been installed. Thank you and enjoy!”



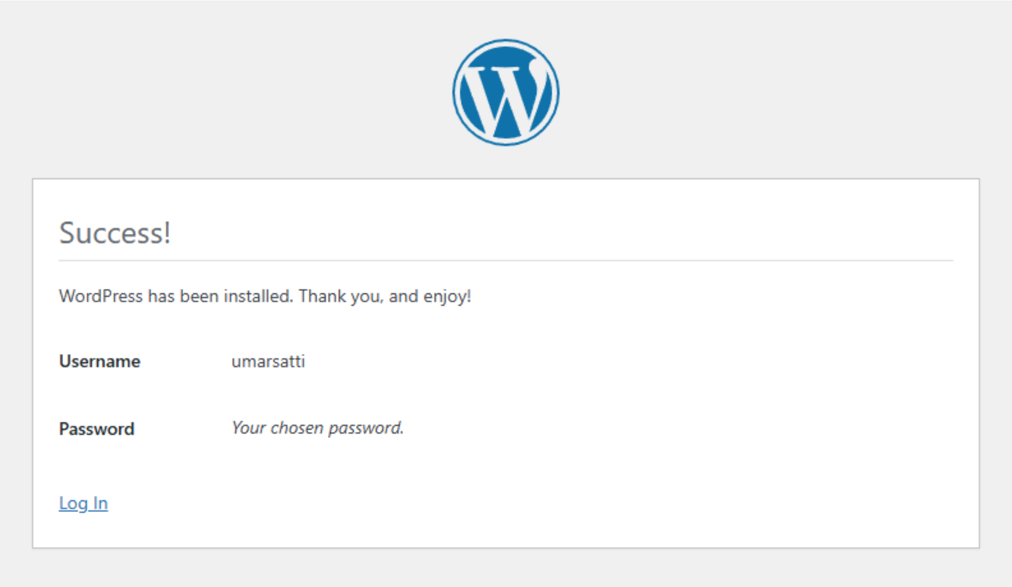
Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Do not worry, you can always change these settings later.

Site Title	Umar Satti
Username	umarsatti Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.
Password	n*!CxMw@9@#I*q0z6U Hide Strong **Important:** You will need this password to log in. Please store it in a secure location.
Your Email	umar@example.com Double-check your email address before continuing.
Search engine visibility	□ Discourage search engines from indexing this site It is up to search engines to honor this request.



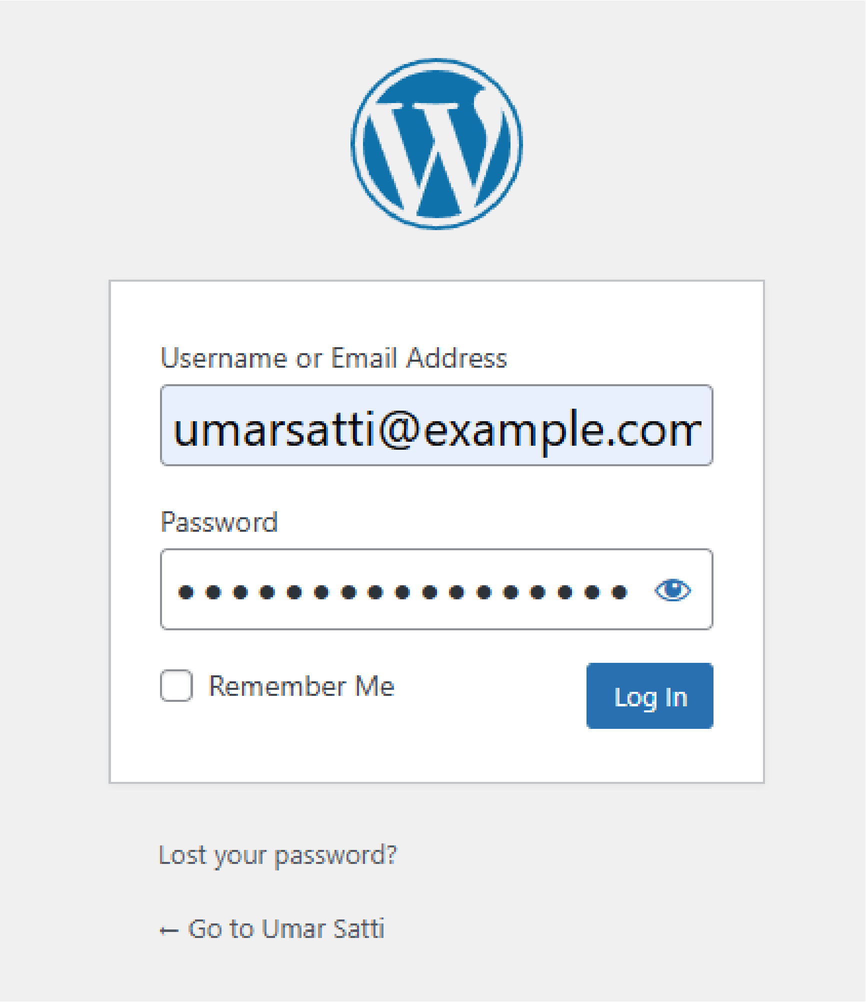
Success!

WordPress has been installed. Thank you, and enjoy!

Username	umarsatti
Password	*Your chosen password.*

Step 6: Log In to Your WordPress Admin Dashboard

1. After installation is completed, click **Log In** and use the credentials provided earlier.
2. Go to **http://<your-ec2-public-ip>/wp-login.php**
    - Example: http://34.203.40.26/wp-login.php
3. Enter the username (umarsatti) and password (P@ssw0rd).
4. Click **Log In.**



Username or Email Address

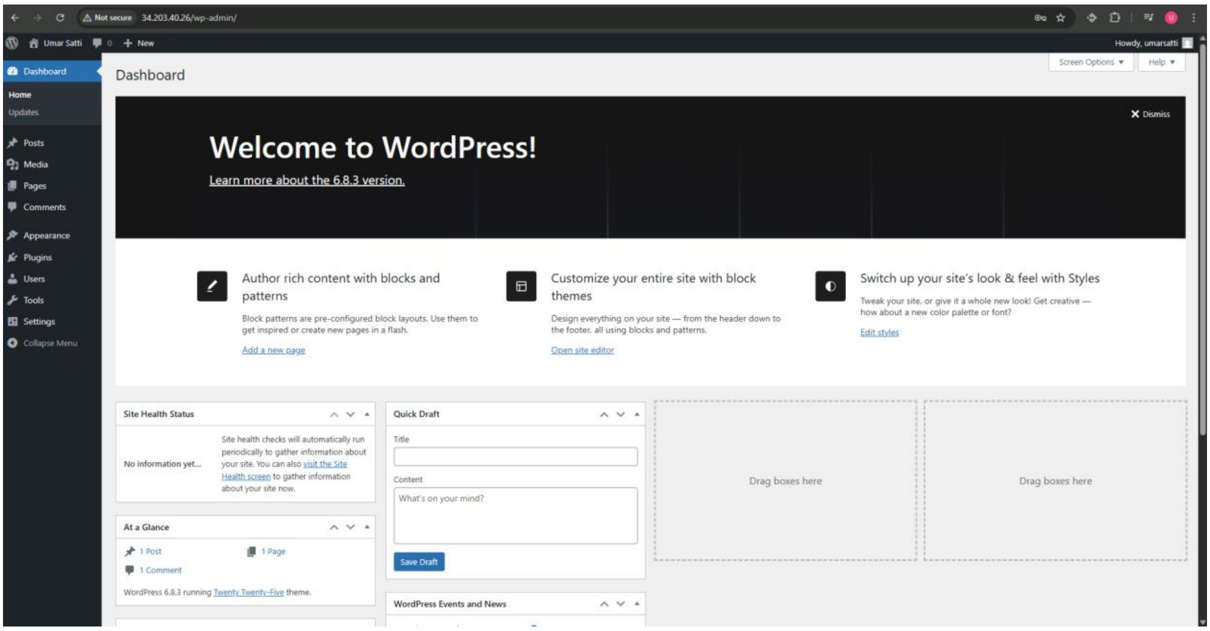
umarsatti@example.com

Password

•••••••••••••••••

□ Remember Me

The page should automatically redirect to the WordPress Admin Dashboard, confirming that the website is now fully functional.



Welcome to WordPress!

Learn more about the 6.8.3 version.

Author rich content with blocks and patterns

Block patterns are pre-configured block layouts. Use them to get inspired or create new pages in a flash.

Customize your entire site with block themes

Design everything on your site — from the header down to the footer, all using blocks and patterns.

Switch up your site's look & feel with Styles

Tweak your site, or give it a whole new look! Get creative — how about a new color palette or font?

No information yet...

Site health checks will automatically run periodically to gather information about your site. You can also visit the Site Health screen to gather information about your site now.

WordPress 6.8.3 running Twenty Twenty-Five theme.

The AWS infrastructure, provisioned by Terraform, is now complete and verified. The deployment and initial configuration of the WordPress application were successful, meaning the application is healthy and accessible. The components defined in Terraform (VPC, EC2, security groups) and the custom user data script integrated smoothly to create a live WordPress environment. This showcases how infrastructure-as-code simplifies complex deployments.

Task 1.10: Troubleshooting and Lessons Learned

During developing and deploying the Terraform-based WordPress environment, several challenges were encountered, particularly around automation, module communication, and non-interactive scripting. This section documents the issues faced, their underlying causes, and the steps taken to resolve them, so that others following this guide can avoid similar pitfalls.

Issue 1: EC2 User Data Script Fails to Execute MySQL Commands

Problem Description:
While running the Terraform deployment, the EC2 instance launched successfully, but the user data script failed to create the MySQL database and user as expected.
The issue occurred because the mysql command normally runs in interactive mode, waiting for manual user input. However, terraform user data scripts execute in a non-interactive shell at instance boot, meaning that any command requiring interaction will hang indefinitely.

Root Cause:
Using commands such as **sudo mysql** followed by **CREATE DATABASE wordpress;** caused the script to stall since it opened the MySQL shell without automatically executing any SQL statements.

Solution:
To ensure commands execute non-interactively, I used the **mysql -e** flag, which allows SQL commands to run directly from the command line. Here is the code:

- mysql -e “CREATE DATABASE wordpress;”
- mysql -e “CREATE USER ‘umarsatti’@’localhost’ IDENTIFIED BY ‘P@ssw0rd’;”
- mysql -e “GRANT ALL PRIVILEGES ON wordpress.* TO ‘umarsatti’@’localhost’;”
- mysql -e “FLUSH PRIVILEGES;”
- mysql -e “EXIT;”

This approach executes SQL statements inline, allowing the EC2 user data script to complete without requiring human input.

Issue 2: Incorrect Implementation of EC2 User Data in Terraform

Problem Description:
Initially, there was confusion about how to properly define and attach a user data script to an EC2 instance in Terraform as it is not clearly defined in Terraform documentation. The script failed to execute or was not recognized, leading to an incomplete WordPress setup.

Root Cause:
User data was not being passed correctly to the **aws_instance** resource. Terraform requires

the script to be provided as a string or a file reference. It cannot automatically interpret arbitrary text blocks unless properly formatted.

Solution

User data scripts should be written as **Bash shell scripts** (with **#!/bin/bash** at the top).
To properly link the script to the Terraform EC2 resource, the following syntax should be used within the **aws_instance** resource:

- user_data = file("${path.root}/userdata.sh")

This tells Terraform to read the contents of the external userdata.sh file located in the root module and pass it to the EC2 instance. This method ensures the script is uploaded in the correct format and executed at instance launch. Alternatively, if the user data script is in the same directory as EC2 module, it should have the following syntax:

- user_data = file("${path.module}/userdata.sh")

Issue 3: Sharing Outputs Between Terraform Modules

Problem Description:
There was confusion about how to pass values such as subnet IDs and security group IDs from the **VPC module** to the **EC2 module**. Terraform modules are isolated by design, so variables defined in one module are not directly accessible in another.

Root Cause:
Terraform modules don't share variables automatically. To pass data between them, outputs from one module must explicitly be referenced as inputs (input variables) in another module.

Solution:
The solution involved using **outputs.tf** in the VPC module and referencing those outputs in the EC2 module using variables.

Conclusion

This project successfully demonstrated how to automate infrastructure provisioning and WordPress deployment on AWS using Terraform. By following this documentation, users can create the same setup to launch an EC2 WordPress environment with minimal manual configuration.