

TASK 2

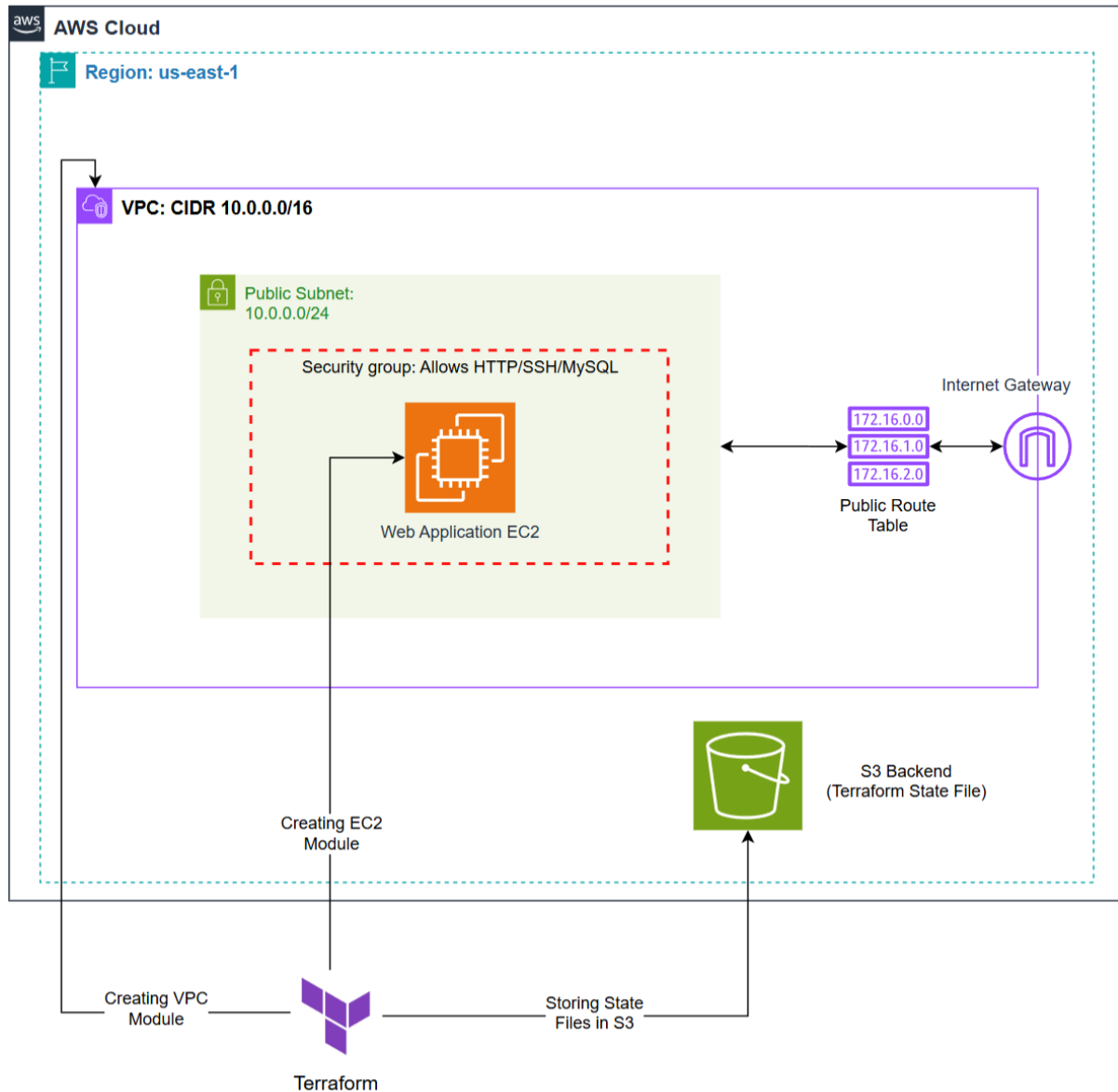
AWS EC2 WordPress Deployment with Terraform

Umar Satti

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

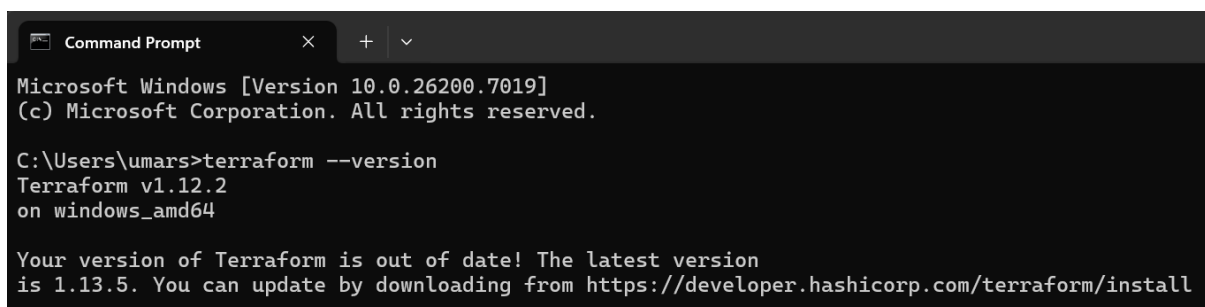


Set Up Terraform Environment

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Your version of Terraform is out of date!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

Steps:

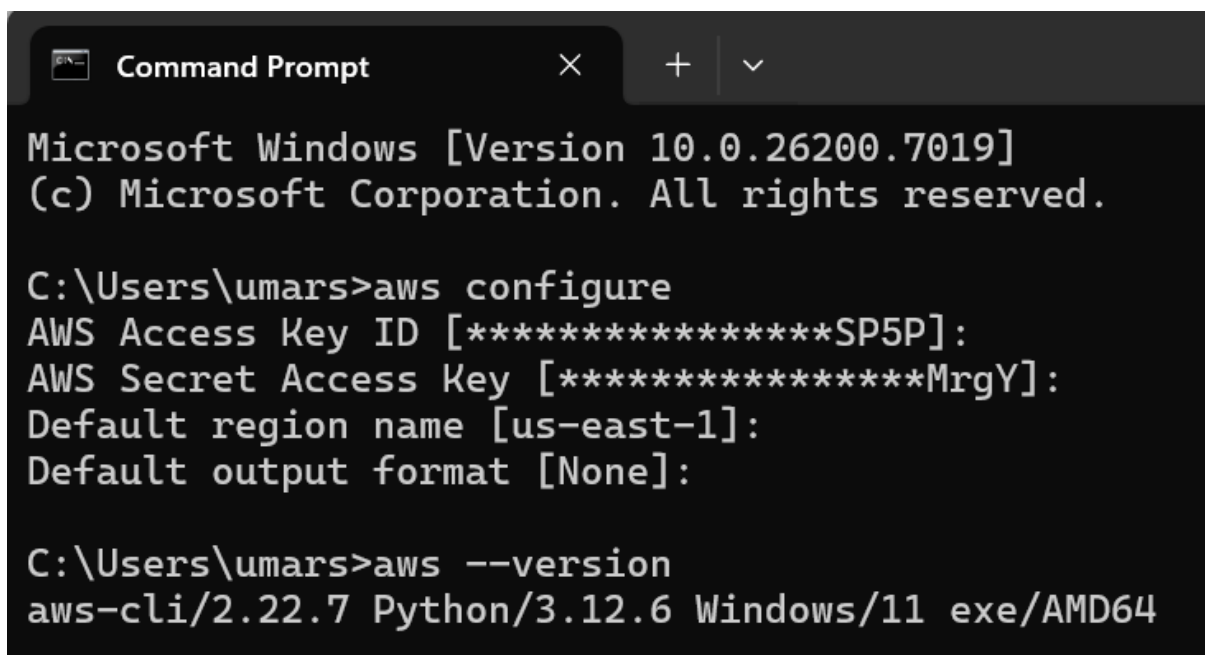
1. Download the AWS CLI MSI installer for Windows (64-bit) from the following link: <https://awscli.amazonaws.com/AWSCLIV2.msi>
2. Run the AWS CLI installer by choosing custom path. Open “System Properties” and then click “Environment Variables” at the bottom of the tab.
3. Navigate to the “user variables”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

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
```

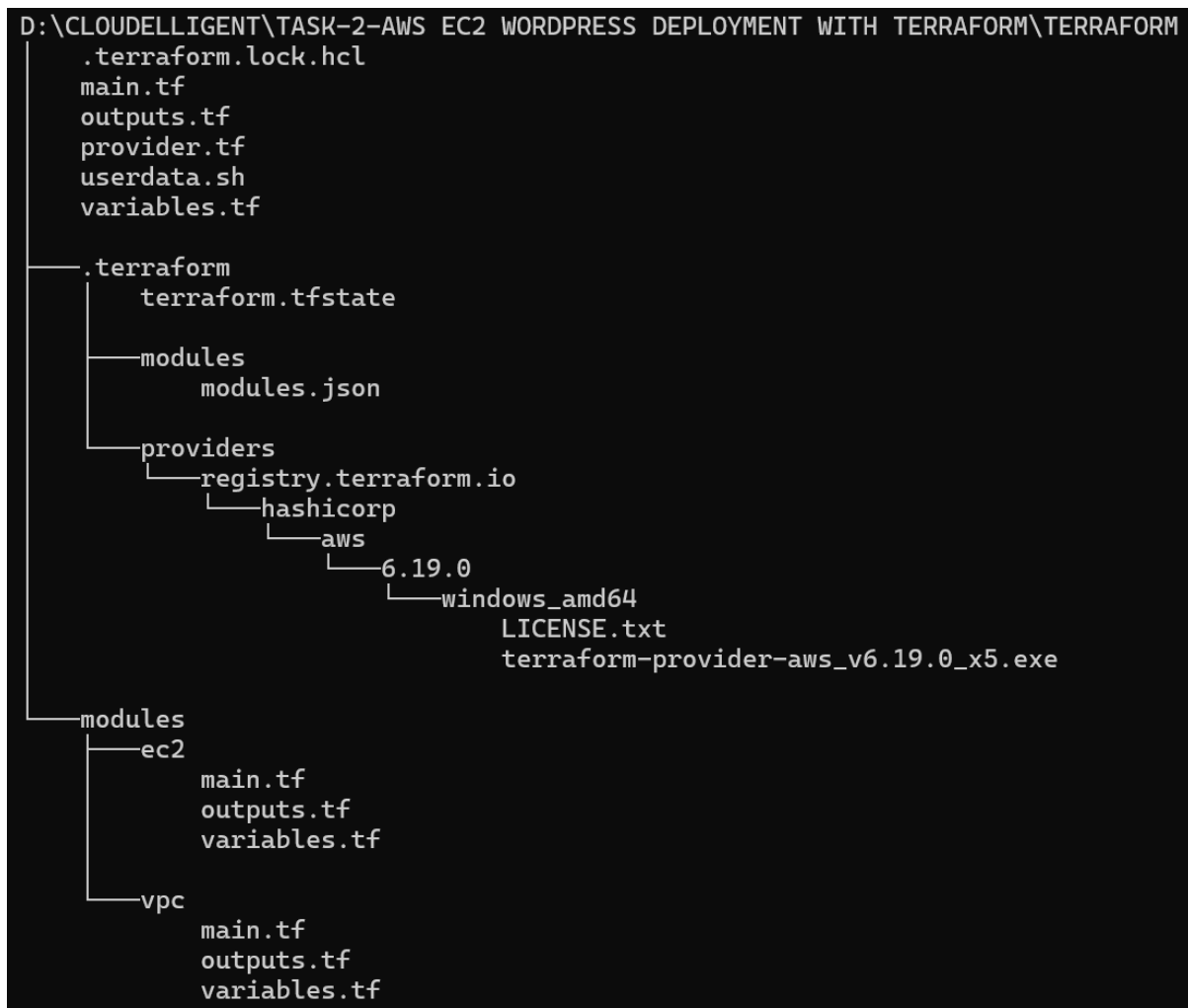
Task 1.3: Create Terraform Project Structure

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.

My terraform project directory structure looks like this:



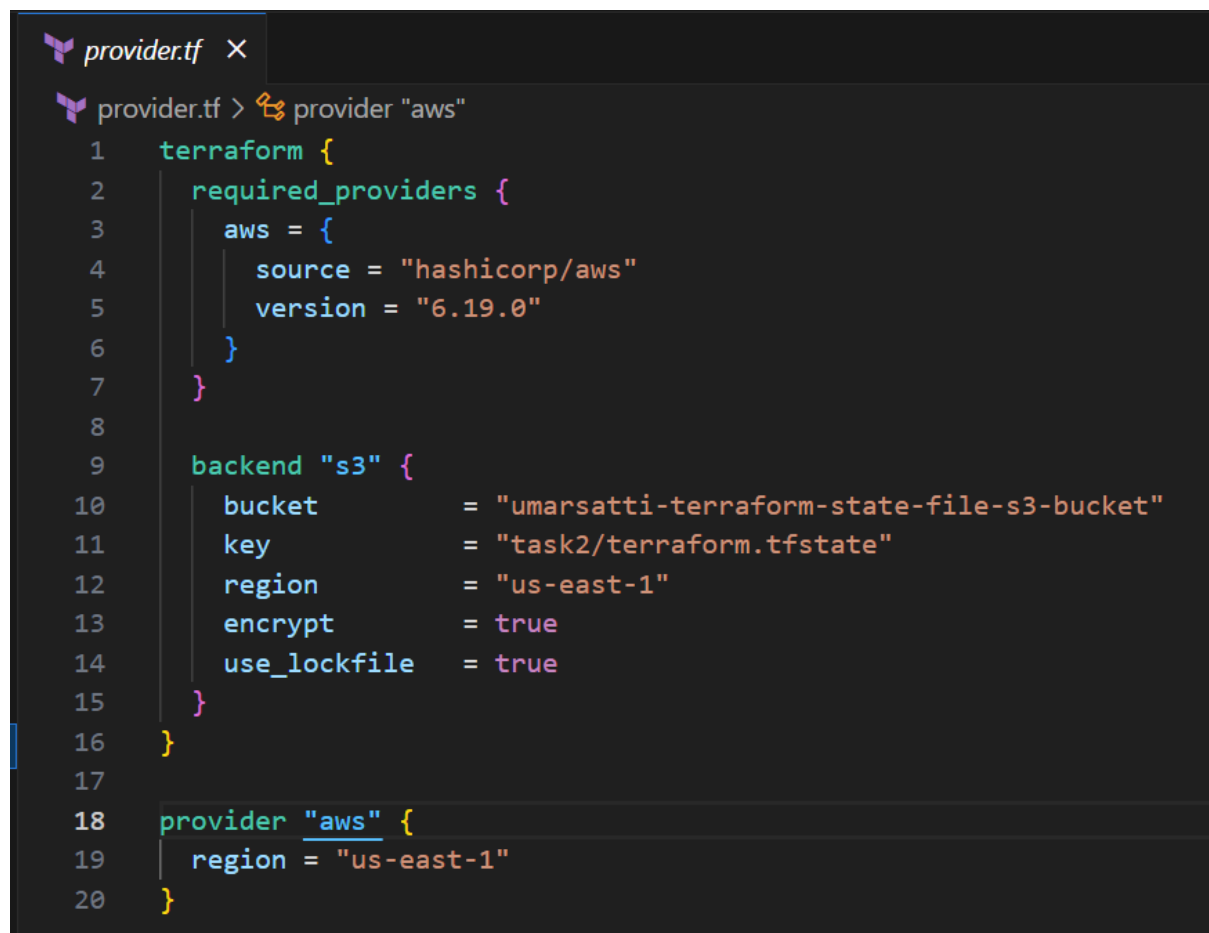
Task 1.4: Create S3 Bucket for Terraform Remote Backend

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

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.

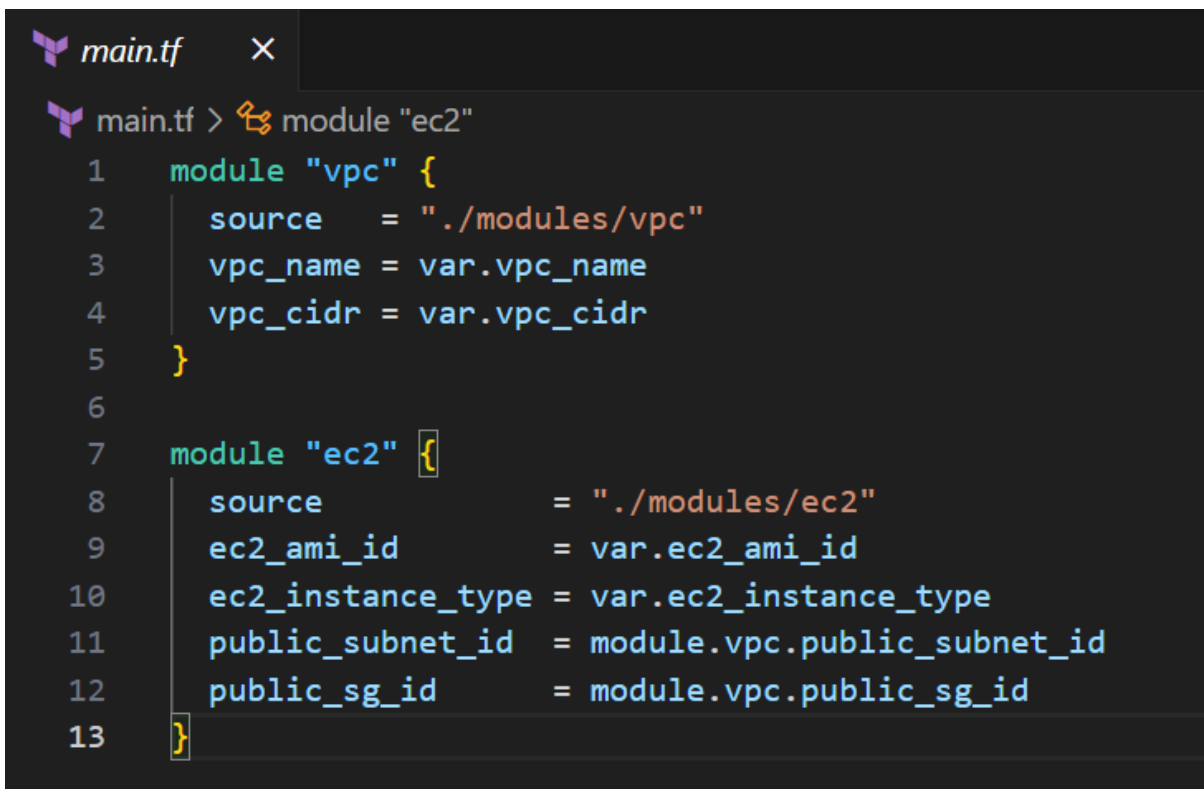


```
provider.tf
1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "6.19.0"
6     }
7   }
8
9   backend "s3" {
10     bucket      = "umarsatti-terraform-state-file-s3-bucket"
11     key         = "task2/terraform.tfstate"
12     region      = "us-east-1"
13     encrypt     = true
14     use_lockfile = true
15   }
16 }
17
18 provider "aws" {
19   region = "us-east-1"
20 }
```

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 "ec2"
1  module "vpc" {
2      source      = "../modules/vpc"
3      vpc_name    = var.vpc_name
4      vpc_cidr    = var.vpc_cidr
5  }
6
7  module "ec2" {
8      source              = "../modules/ec2"
9      ec2_ami_id          = var.ec2_ami_id
10     ec2_instance_type   = var.ec2_instance_type
11     public_subnet_id    = module.vpc.public_subnet_id
12     public_sg_id        = module.vpc.public_sg_id
13 }
```

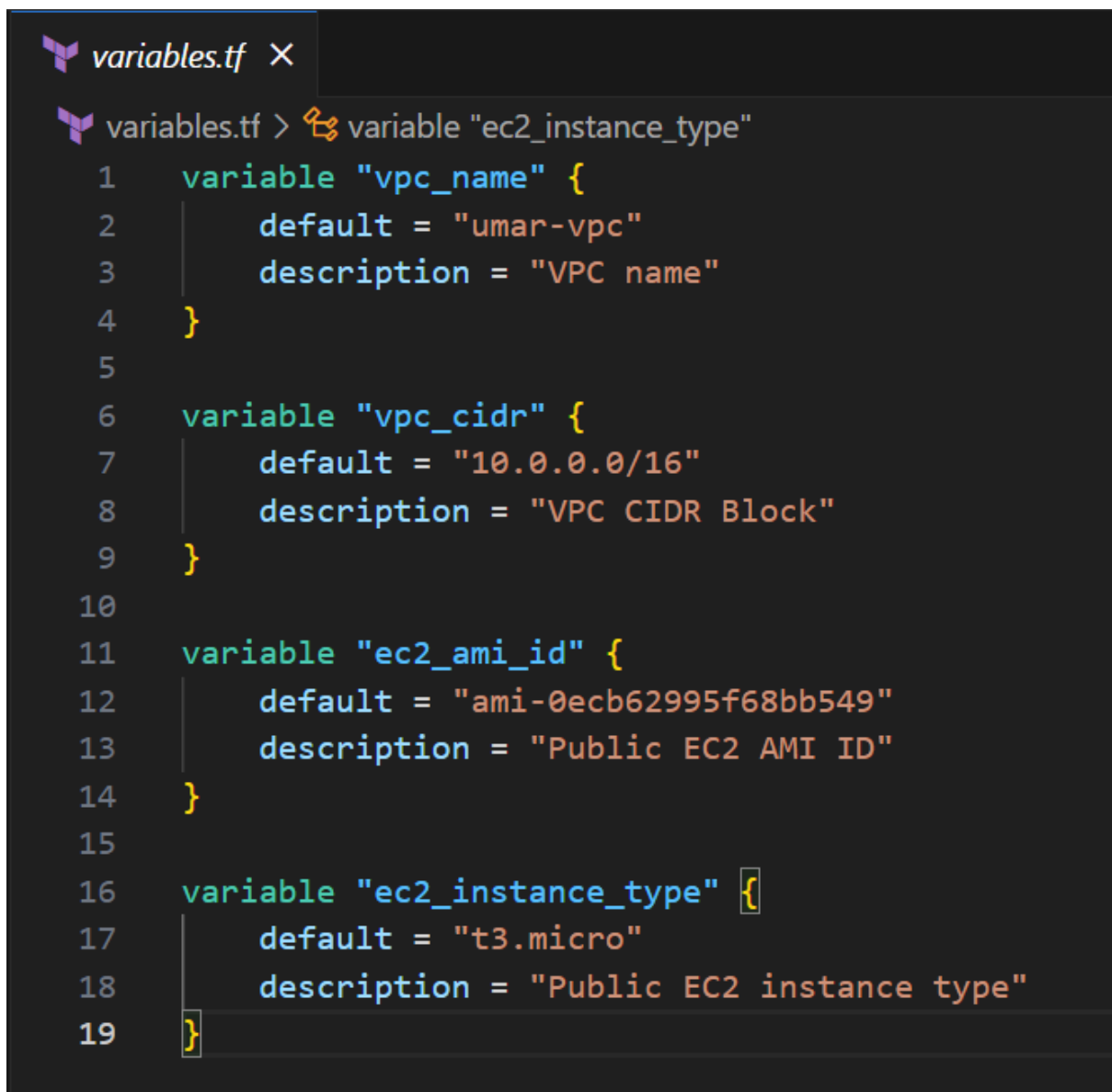
This file defines how Terraform composes the overall infrastructure using **modules** for networking and compute layers.

- 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:



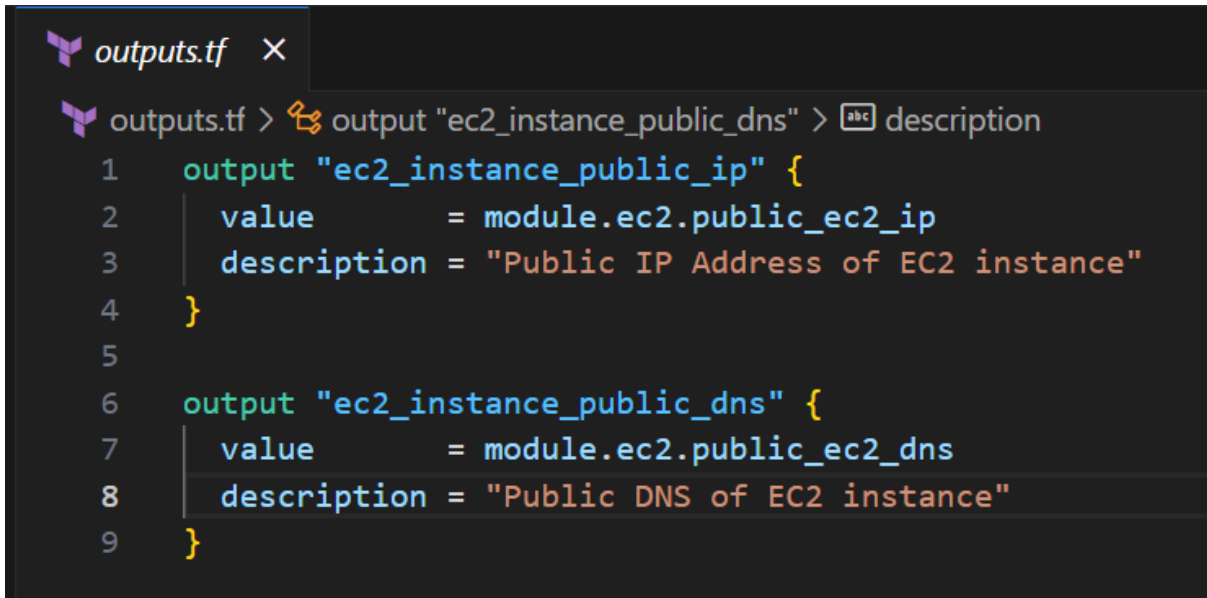
```
variables.tf ×  
variables.tf > variable "ec2_instance_type"  
1  variable "vpc_name" {  
2      default = "umar-vpc"  
3      description = "VPC name"  
4  }  
5  
6  variable "vpc_cidr" {  
7      default = "10.0.0.0/16"  
8      description = "VPC CIDR Block"  
9  }  
10  
11 variable "ec2_ami_id" {  
12     default = "ami-0ecb62995f68bb549"  
13     description = "Public EC2 AMI ID"  
14 }  
15  
16 variable "ec2_instance_type" {  
17     default = "t3.micro"  
18     description = "Public EC2 instance type"  
19 }
```

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



```
outputs.tf X
outputs.tf > output "ec2_instance_public_dns" > description
1  output "ec2_instance_public_ip" {
2      value      = module.ec2.public_ip
3      description = "Public IP Address of EC2 instance"
4  }
5
6  output "ec2_instance_public_dns" {
7      value      = module.ec2.public_dns
8      description = "Public DNS of EC2 instance"
9  }
```

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):

```
$ userdata.sh X
$ userdata.sh
1  #!/bin/bash
2
3  # Install Apache
4  sudo apt update -y
5  sudo apt install apache2 -y
6  sudo systemctl start apache2 #Optional
7  sudo systemctl enable apache2 #Optional
8
9  # Install PHP and dependencies
10 sudo apt install php libapache2-mod-php php-mysql -y
11 sudo apt install php libapache2-mod-php php-mysql php-xml php-mbstring php-curl php-zip -y
12
13 # Install MySQL
14 sudo apt install mysql-server -y
15
16 #Configure MySQL
17 mysql -e "CREATE DATABASE wordpress;"
18 mysql -e "CREATE USER 'umarsatti'@'localhost' IDENTIFIED BY 'P@ssw0rd';"
19 mysql -e "GRANT ALL PRIVILEGES ON wordpress.* TO 'umarsatti'@'localhost';"
20 mysql -e "FLUSH PRIVILEGES;"
21
22 # Download and install WordPress
23 wget https://wordpress.org/latest.tar.gz -P /tmp
24 tar -xzf /tmp/latest.tar.gz -C /tmp
25
26 # Move WordPress files directly into the Apache root directory
27 rm -rf /var/www/html/*
28 mv /tmp/wordpress/* /var/www/html/
29
30 # Fix permissions
31 chown -R www-data:www-data /var/www/html/
32 chmod -R 755 /var/www/html/
33
34 #Configure Wordpress to connect to MySQL database
35 cp /var/www/html/wordpress/wp-config-sample.php /var/www/html/wordpress/wp-config.php
36 sed -i "s/database_name_here/wordpress/" /var/www/html/wordpress/wp-config.php
37 sed -i "s/username_here/umarsatti/" /var/www/html/wordpress/wp-config.php
38 sed -i "s/password_here/P@ssw0rd/" /var/www/html/wordpress/wp-config.php
39
40 #Restart Apache Web Server
41 systemctl restart apache2
```

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

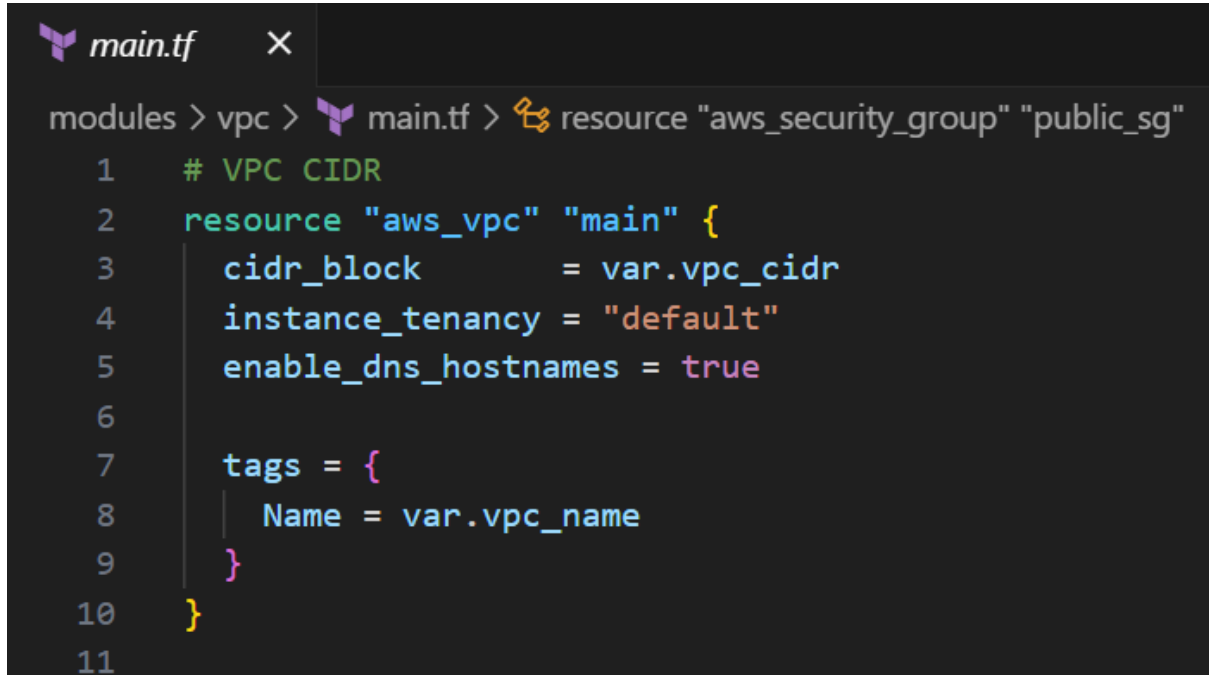
- Installs and starts the **Apache web server** and **PHP** with all required extensions for WordPress.
- Installs **MySQL server**, creates a **wordpress** database, and sets up a dedicated user named **umarsatti** with full privileges.
- Downloads and extracts the latest version of **WordPress**, sets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

A screenshot of a code editor with a dark theme. The editor shows a file named 'main.tf' with a breadcrumb path 'modules > vpc > main.tf'. The code defines an AWS VPC resource. Line 1 is a comment '# VPC CIDR'. Line 2 starts the resource block 'resource "aws_vpc" "main" {'. Lines 3-5 define 'cidr_block = var.vpc_cidr', 'instance_tenancy = "default"', and 'enable_dns_hostnames = true'. Lines 7-9 define a 'tags' block with 'Name = var.vpc_name'. Line 10 closes the resource block with '}'. Line 11 is an empty line.

```
modules > vpc > main.tf > resource "aws_security_group" "public_sg"
1  # VPC CIDR
2  resource "aws_vpc" "main" {
3      cidr_block      = var.vpc_cidr
4      instance_tenancy = "default"
5      enable_dns_hostnames = true
6
7      tags = {
8          Name = var.vpc_name
9      }
10 }
11
```

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"

12  # Public Subnets
13  resource "aws_subnet" "public_subnet" {
14      vpc_id      = aws_vpc.main.id
15      cidr_block  = "10.0.0.0/24"
16      availability_zone = "us-east-1a"
17
18      tags = {
19          Name = "Public-Subnet"
20      }
21  }
```

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"

23  # Internet Gateway
24  resource "aws_internet_gateway" "igw" {
25      vpc_id = aws_vpc.main.id
26
27      tags = {
28          Name = "umar-igw"
29      }
30  }
```

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

32  # Route Tables
33  resource "aws_route_table" "public_rt" {
34    vpc_id = aws_vpc.main.id
35
36    route {
37      cidr_block = "0.0.0.0/0"
38      gateway_id = aws_internet_gateway.igw.id
39    }
40
41    tags = {
42      Name = "Public-Route-Table"
43    }
44  }
45
46  # Route Table Association
47  resource "aws_route_table_association" "public" {
48    subnet_id      = aws_subnet.public_subnet.id
49    route_table_id = aws_route_table.public_rt.id
50  }
51
```

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"

54 # Public Security Group
55 resource "aws_security_group" "public_sg" {
56     name           = "public-sg"
57     description    = "Allows HTTP, SSH, and MySQL traffic from the internet"
58     vpc_id         = aws_vpc.main.id
59
60     ingress {
61         from_port   = 80
62         to_port     = 80
63         protocol    = "tcp"
64         cidr_blocks = ["0.0.0.0/0"]
65     }
66
67     ingress {
68         from_port   = 22
69         to_port     = 22
70         protocol    = "tcp"
71         cidr_blocks = ["0.0.0.0/0"]
72     }
73
74     ingress {
75         from_port   = 3306
76         to_port     = 3306
77         protocol    = "tcp"
78         cidr_blocks = ["0.0.0.0/0"]
79     }
80
81     egress {
82         from_port   = 0
83         to_port     = 0
84         protocol    = "-1"
85         cidr_blocks = ["0.0.0.0/0"]
86     }
87
88     tags = {
89         Name        = "public-sg"
90         Application = "WordPress"
91     }
92 }
```

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:

```
variables.tf X
modules > vpc > variables.tf > variable "vpc_cidr"
1  variable "vpc_name" {
2    |   type = string
3    | }
4
5  variable "vpc_cidr" {
6    |   type = string
7    | }
```

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:

```
outputs.tf X
modules > vpc > outputs.tf > output "public_sg_id" > value
1  output "public_subnet_id" {
2    |   description = "Public subnet ID"
3    |   value       = aws_subnet.public_subnet.id
4    | }
5
6  output "public_sg_id" {
7    |   description = "Public security group ID"
8    |   value       = aws_security_group.public_sg.id
9    | }
```

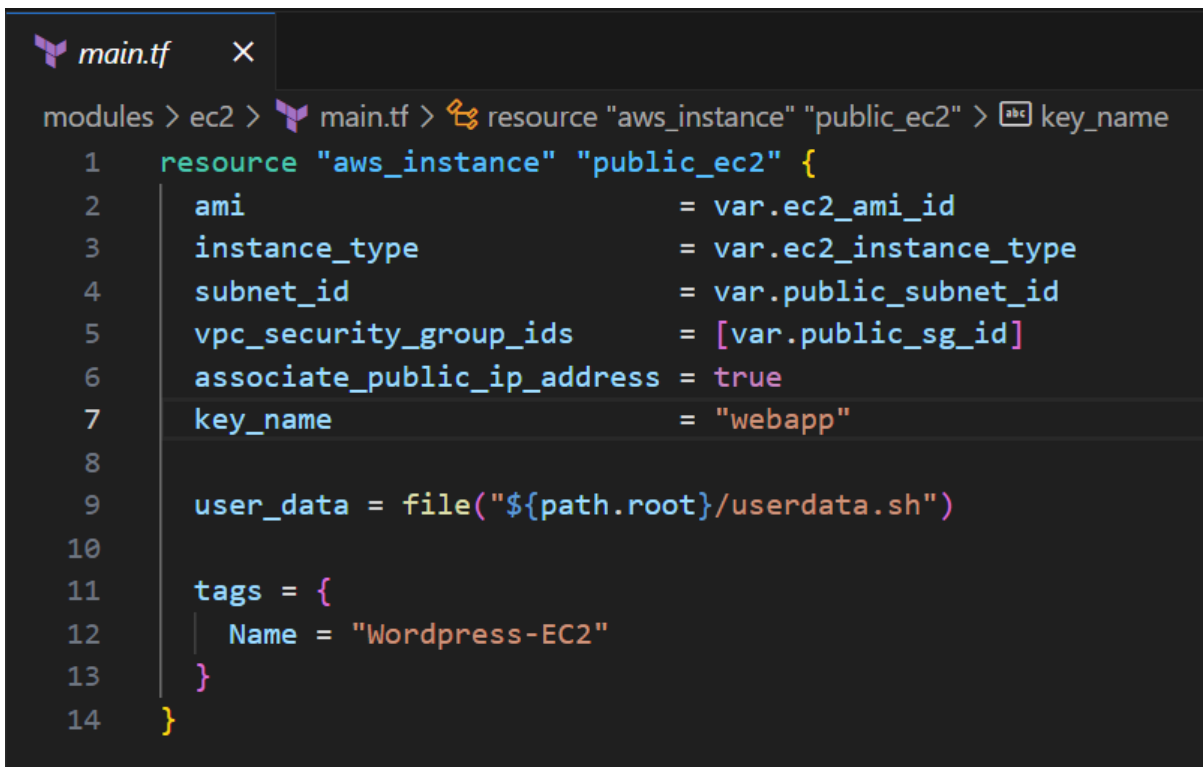
Outputs make key resource identifiers available to other modules (like EC2).

- **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
1  resource "aws_instance" "public_ec2" {
2      ami                    = var.ec2_ami_id
3      instance_type         = var.ec2_instance_type
4      subnet_id             = var.public_subnet_id
5      vpc_security_group_ids = [var.public_sg_id]
6      associate_public_ip_address = true
7      key_name               = "webapp"
8
9      user_data = file("${path.root}/userdata.sh")
10
11     tags = {
12         Name = "Wordpress-EC2"
13     }
14 }

```

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:

A screenshot of a code editor window titled 'variables.tf'. The editor shows a Terraform configuration file with four variable definitions. The first variable is 'ec2_ami_id' of type string. The second is 'ec2_instance_type' of type string. The third is 'public_subnet_id' of type string. The fourth is 'public_sg_id' of type string. The code is written in a dark-themed editor with syntax highlighting. The breadcrumb navigation at the top shows 'modules > ec2 > variables.tf > ...'.

```
modules > ec2 > variables.tf > ...  
1  variable "ec2_ami_id" {  
2    |   type = string  
3  }  
4  
5  variable "ec2_instance_type" {  
6    |   type = string  
7  }  
8  
9  variable "public_subnet_id" {  
10   |   type = string  
11  }  
12  
13 variable "public_sg_id" {  
14   |   type = string  
15  }
```

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
1  output "public_ec2_ip" {
2      value      = aws_instance.public_ec2.public_ip
3      description = "Public IP of EC2 instance"
4  }
5
6  output "public_ec2_dns" {
7      value      = aws_instance.public_ec2.public_dns
8      description = "Public DNS of EC2 instance"
9  }
```

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.