AWS EC2 WordPress Deployment with Terraform

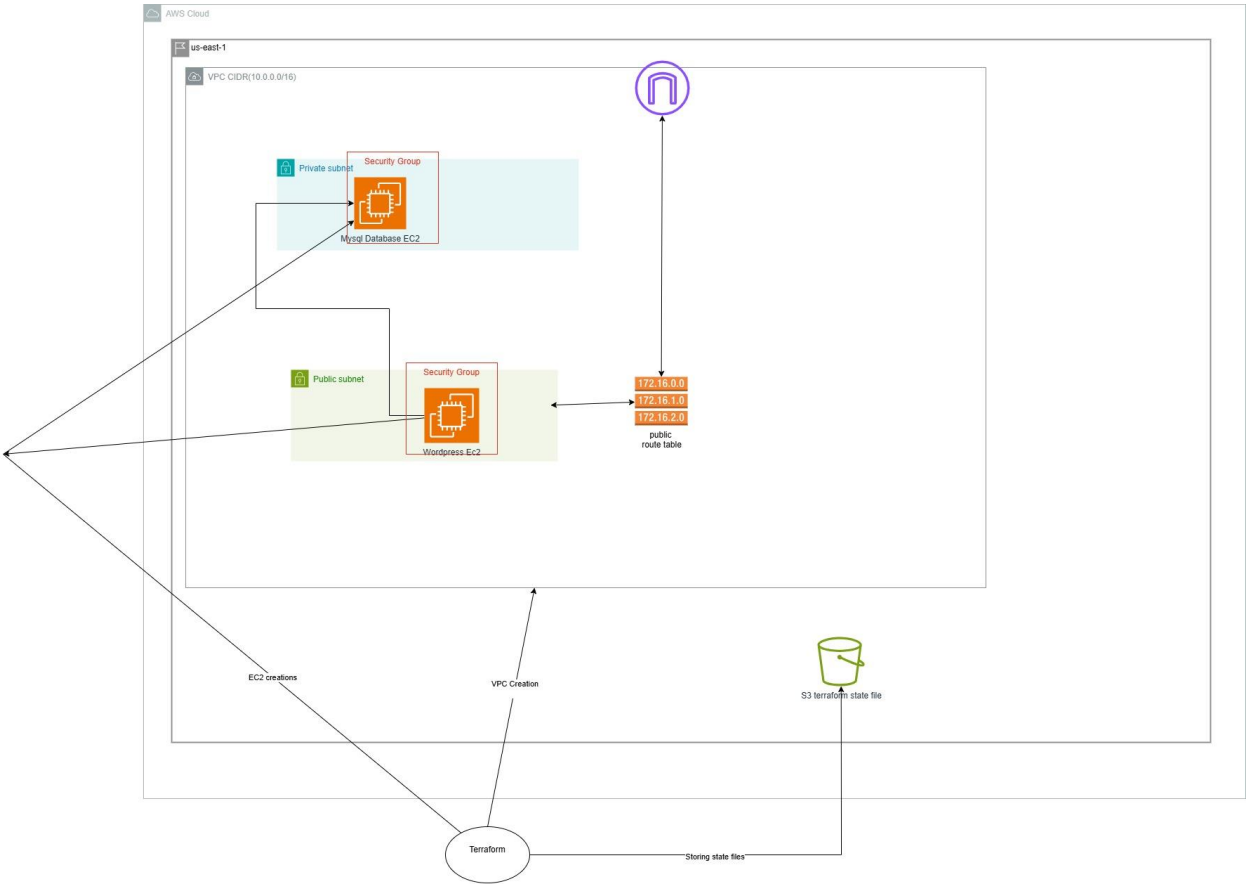
Author: Sardar Noor Ul Hassan
Role: Cloud Intern – Cloudelligent

1. Project Overview

This project demonstrates how to automate the deployment of a full WordPress website using Terraform on AWS.
It provisions two EC2 instances:

- One for **WordPress (web server)**
- One for **MySQL (database server)**

Both instances are automatically configured using **user data scripts**, and all networking components (VPC, subnets, security groups, internet gateway, NAT gateway, etc.) are created through Terraform.
This setup represents a two-tier cloud architecture, following real-world DevOps practices.



2. Set Up Terraform Environment

Before writing any Terraform code, the environment was prepared.

Steps followed:

1. Installed Terraform on the local machine.
2. Installed and configured **AWS CLI** using the command:
3. aws configure

Entered:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name: us-west-2

4. Verified connection:
5. aws sts get-caller-identity

This ensured that Terraform and AWS CLI can communicate with my AWS account.

Reason:
Without AWS CLI credentials, Terraform cannot authenticate and deploy resources.

---

3. Create Terraform Configuration Files

Terraform uses a declarative approach. Each file defines a specific part of infrastructure.

main.tf

Defined:

- AWS provider

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 6.0"
    }
  }
}
```

- VPC, subnets, route tables

```
# ---- VPC ------
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = { Name = "wordpress-vpc" }
}

# -------- Subnets -----
resource "aws_subnet" "public" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.1.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-west-2a"
  tags = { Name = "public-subnet" }
}
```

```
resource "aws_subnet" "private" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.2.0/24"
  availability_zone = "us-west-2a"
  tags = { Name = "private-subnet" }
}
```

- Internet Gateway and NAT Gateway

```
# ---------- Internet Gateway & Route Tables -----
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id
  tags   = { Name = "wordpress-igw" }
}
```

```
# ---------- NAT Gateway ----
resource "aws_eip" "nat_eip" {
  domain = "vpc"
  tags   = { Name = "nat-eip" }
}
```

- Security groups

```
# ---------- Security Groups ----
resource "aws_security_group" "wp_sg" {
  name   = "wp-sg"
  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"] # restrict to your IP in production
  }

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

  tags = { Name = "wordpress-sg" }
}
```

```
resource "aws_security_group" "db_sg" {
  name   = "db-sg"
  vpc_id = aws_vpc.main.id

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

  tags = { Name = "db-sg" }
}

# Link WP SG to DB SG
resource "aws_security_group_rule" "allow_wp_to_db" {
  type                     = "ingress"
  from_port                = 3306
  to_port                  = 3306
  protocol                 = "tcp"
  security_group_id        = aws_security_group.db_sg.id
  source_security_group_id = aws_security_group.wp_sg.id
}
```

- EC2 instances for MySQL and WordPress

```
# ---------- EC2 Instances ---
resource "aws_instance" "mysql" {
  ami                    = var.db_ami
  instance_type          = "t3.micro"
  subnet_id              = aws_subnet.private.id
  vpc_security_group_ids = [aws_security_group.db_sg.id]
  key_name               = var.key_name
  associate_public_ip_address = false

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

  tags = { Name = "mysql-instance" }
}

locals {
  wp_userdata = templatefile("${path.module}/userdata-wordpress.sh", {
    db_host = aws_instance.mysql.private_ip
  })
}
```

```
resource "aws_instance" "wordpress" {
  ami                    = var.wp_ami
  instance_type          = "t3.micro"
  subnet_id              = aws_subnet.public.id
  vpc_security_group_ids = [aws_security_group.wp_sg.id]
  key_name               = var.key_name
  associate_public_ip_address = true

  user_data = local.wp_userdata
  user_data_replace_on_change = true

  tags = { Name = "wordpress-instance" }
}
```

It also contained user_data blocks to automate installation of software on both servers.

variables.tf

Defined configurable values like:

- AMI IDs (Amazon Linux 2023)
- Key pair name for SSH access

variables.tf

```
variable "wp_ami" {
  description = "Amazon Linux 2023 AMI for WordPress (us-west-2)"
  type        = string
  default     = "ami-04f9aa2b7c7091927"
}

variable "db_ami" {
  description = "Amazon Linux 2023 AMI for MySQL (us-west-2)"
  type        = string
  default     = "ami-04f9aa2b7c7091927"
}

variable "key_name" {
  description = "EC2 key pair name"
  type        = string
  default     = "noor-key"
}

```

outputs.tf

Displayed useful information after deployment, such as:

- Public IP of WordPress instance
- Private IP of MySQL instance

Reason:
Separating variables and outputs makes the code reusable and readable.

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

mysql_private_ip = "10.0.2.69"
subnet_ids = {
  "private" = "subnet-0b7dd67a43cff5c1a"
  "public" = "subnet-0fc32681129deb88d"
}
vpc_id = "vpc-058220c77e708a2f3"
wordpress_public_ip = "44.250.108.202"
PS D:\CLOUDELLIGENT INTERNSHIP\Task1(Terraform Wordpress)>
```

---

4. Networking Configuration

A custom **VPC** was created with the CIDR block 10.0.0.0/16.

Inside the VPC:

- **Public subnet (10.0.1.0/24):** for the WordPress EC2 instance.
- **Private subnet (10.0.2.0/24):** for the MySQL EC2 instance.

Internet Gateway (IGW) was attached to the VPC for public internet access.
NAT Gateway was added in the public subnet so that private subnet instances (like the MySQL server) can reach the internet for package updates.

Two **route tables** were used:

- Public route table → connected to IGW
- Private route table → connected to NAT Gateway

Reason:
This separation ensures WordPress is reachable from the internet but MySQL remains secure inside the private subnet.

---

5. Provision EC2 Instances

MySQL Instance

- Placed in private subnet.
- No public IP.
- Installed **MariaDB** using user data script:
- dnf -y install mariadb105-server
- systemctl enable mariadb
- systemctl start mariadb
- Created database wordpress_db and user noor with password onePlus1@.

WordPress Instance

- Placed in public subnet with public IP.
- Installed Apache, PHP, and WordPress using user data.
- Connected to MySQL private IP using Terraform variable substitution.

Reason:
Two-tier separation improves security and scalability.

---

6. User Data Script Explanation

userdata-mysql.sh

```
#!/bin/bash
set -euxo pipefail

dnf -y update
dnf -y install mariadb105-server

systemctl enable mariadb
systemctl start mariadb

# Allow private connections
sed -i 's/^\(bind-address\|#bind-address\).*/bind-address=0.0.0.0/' /etc/my.cnf.d/maria
systemctl restart mariadb

mysql -uroot <<'SQL'
CREATE DATABASE wordpress_db;
CREATE USER 'noor'@'%' IDENTIFIED BY 'onePlus1@';
GRANT ALL PRIVILEGES ON wordpress_db.* TO 'noor'@'%';
FLUSH PRIVILEGES;
SQL
```

- Updates system packages.
- Installs and starts MariaDB service.
- Configures it to accept connections from the WordPress instance only.
- Creates WordPress database and user.

userdata-wordpress.sh

```
#!/bin/bash
set -euxo pipefail

# Template variable from Terraform
DB_HOST="${db_host}"

dnf -y update
dnf -y install httpd php php-mysqlnd wget unzip

systemctl enable httpd
systemctl start httpd

cd /var/www/html
wget https://wordpress.org/latest.zip
unzip -q latest.zip
cp -r wordpress/* .
rm -rf wordpress latest.zip

chown -R apache:apache /var/www/html

mv wp-config-sample.php wp-config.php
sed -i "s/database_name_here/wordpress_db/" wp-config.php
sed -i "s/username_here/noor/" wp-config.php
sed -i "s/password_here/onePlus1@/" wp-config.php
sed -i "s/localhost/${db_host}/" wp-config.php

systemctl restart httpd
```

- Updates system packages.
- Installs Apache, PHP, and WordPress automatically.
- Edits wp-config.php to replace database details:
- sed -i "s/localhost/${db_host}/" wp-config.php
- Starts Apache web server.

Reason:
User data scripts automate configuration at the instance's first boot — no manual login needed.

7. Security Group Configuration

- **wp-sg:**
    - Allows inbound HTTP (80) and SSH (22) from anywhere (0.0.0.0/0).
- **db-sg:**
    - Allows inbound MySQL (3306) only from the wp-sg security group.

Reason:
This restricts access to the database only from the web server, not from the public internet.

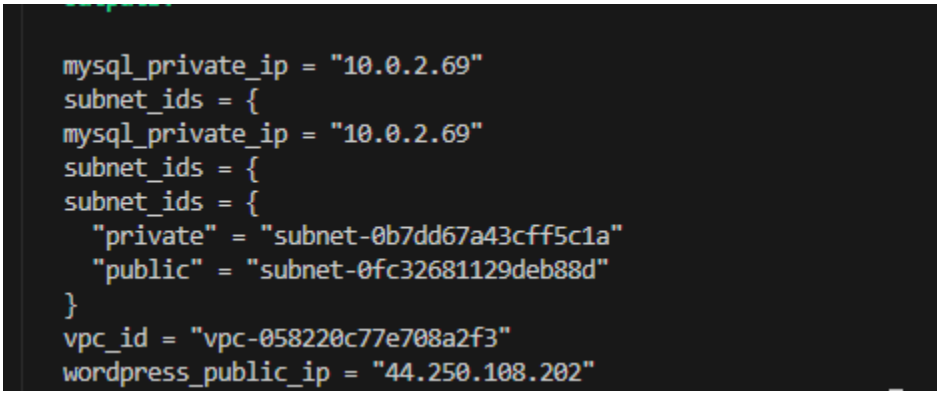
---

8. Output Public IP and Test the Setup

After running:

terraform apply -auto-approve

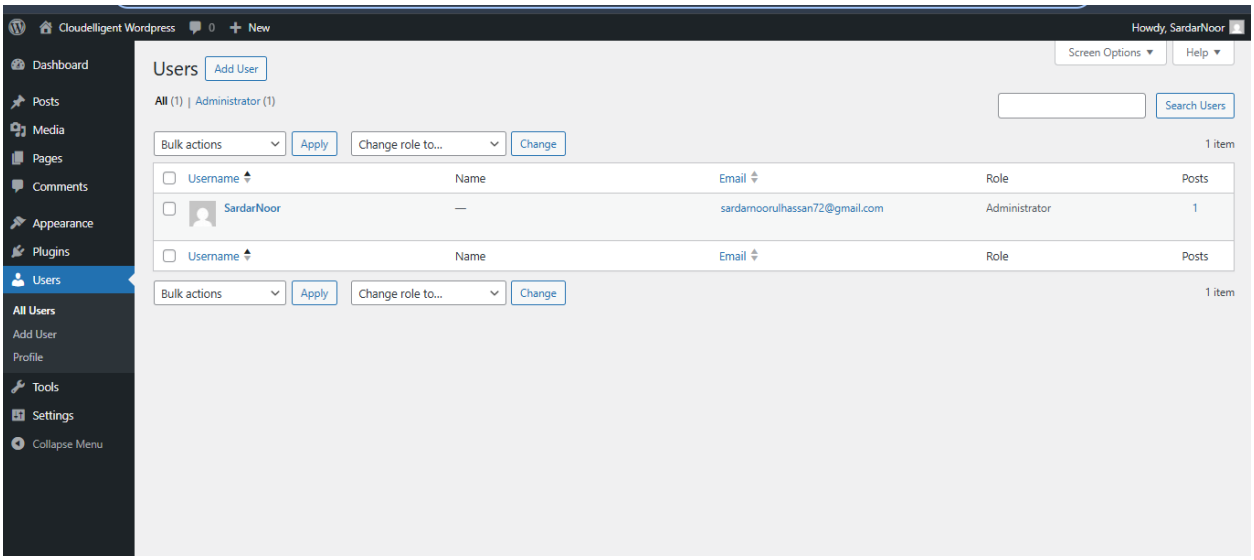
Terraform printed:



Testing Steps:

1. Opened http://44.250.108.202 (wordpress public Ip) in the browser.
2. WordPress installation screen appeared → verified web server working.
3. Completed installation → success message displayed.

4. Logged in to dashboard → verified site operational.



Reason:
Successful connection between web server and database confirms that Terraform's automation and networking worked perfectly.

9. Problems Faced and Solutions

Problem / Error	Cause	Solution Implemented
source_security_group_id not supported	AWS provider v5+ changed security group syntax	Created a separate aws_security_group_rule resource for inbound MySQL rule
Invalid value for vars map: DB_HOST	Wrong case (${DB_HOST} instead of ${db_host})	Changed to lowercase in userdata-wordpress.sh
InvalidAMIID.NotFound	AMI ID didn't exist in us-west-2 region	Replaced with correct ami-04f9aa2b7c7091927 from console
AWS CLI error InvalidAccessKeyId	CLI not configured	Ran aws configure and added Access/Secret Keys
MariaDB service not found	Used Ubuntu commands on Amazon Linux	Replaced apt with dnf and package mariadb105-server

Problem / Error	Cause	Solution Implemented
Terraform version mismatch (provider v6)	Local lock file had v6.20.0	Changed provider version to ~> 6.0 and reinitialized with terraform init -upgrade

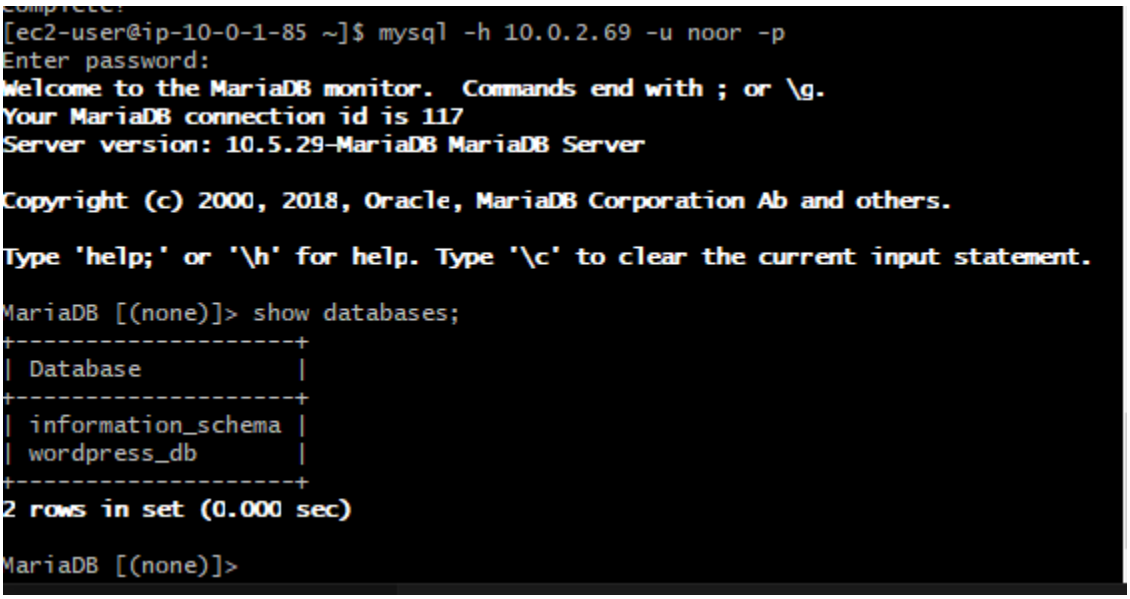
---

10. Final Verification

Both servers are running:

- WordPress accessible publicly via HTTP.
- MySQL instance accessible only privately.
- WordPress dashboard loads successfully.

Successfully connected to the wordpress_db.



11. Conclusion

This project achieved complete automation of a two-tier WordPress application on AWS using Terraform.
Key takeaways:

- Understood how Infrastructure as Code simplifies deployments.
- Learned how to structure Terraform files, handle variables, and debug errors.
- Practiced secure VPC networking using public and private subnets.
- Gained experience writing user data scripts for automatic configuration.

Future improvements:

- Store Terraform state in S3 with DynamoDB locking for collaboration.
- Add an Application Load Balancer for scalability.