Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Gutama Morocho	26/01/2024
	Nombre: Marco Paúl	

Actividad grupal. Despliegue de MEAN multicapa mediante Terraform

Introduccion

Este proyecto diseña, implementa y valida una infraestructura en la nube utilizando Terraform sobre la plataforma de Amazon Web Services (AWS). La infraestructura desplegada está enfocada en soportar una aplicación basada en Nginx y Node.js, junto con una base de datos MongoDB.

Para garantizar la modularidad, escalabilidad y seguridad del despliegue, se ha diseñado una plantilla de Terraform que divide la infraestructura en módulos independientes: VPC, seguridad, balanceador de carga, computación y NAT Gateway. Además, se ha incluido un balanceador de carga (ALB) para distribuir el tráfico entre las instancias de la aplicación y mejorar la disponibilidad del sistema. Finalmente, se han generado salidas (outputs) en Terraform para obtener información clave, como las IPs públicas y privadas de las instancias, el DNS del balanceador y la IP pública del NAT Gateway.

Para los componentes de la infraestructura, se han utilizado imágenes de máquina (AMIs) previamente preparadas: para las instancias de Nginx y Node.js se empleó una AMI generada en una actividad anterior mediante Packer, aprovechando el trabajo previo de configuración y personalización de la imagen; en el caso de MongoDB, se ha seleccionado una AMI pública del catálogo de AWS proporcionada por Bitnami, que ofrece una configuración preestablecida y optimizada del gestor de base de datos, facilitando su implementación y reduciendo la complejidad de la configuración inicial.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Gutama Morocho	26/01/2024
	Nombre: Marco Paúl	

Diagrama de la Estructura de Terraform

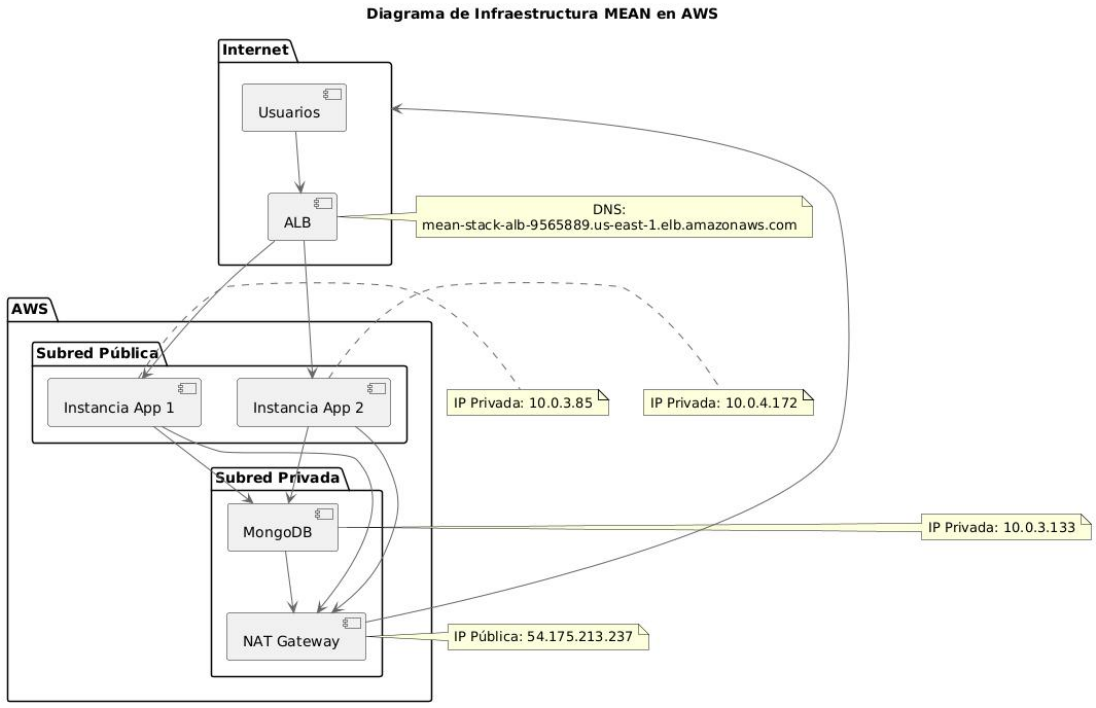

Figura 1. Diagrama infraestructura MEAN en AWS

1. Internet y Usuarios:

Los usuarios acceden a la aplicación a través de Internet utilizando el DNS del ALB (mean-stack-alb-9565889.us-east-1.elb.amazonaws.com).

2. Application Load Balancer (ALB):

- El ALB recibe el tráfico HTTP/HTTPS de los usuarios y lo redirige a las instancias de la aplicación (Nginx + Node.js) en las subredes privadas.
- El ALB está asociado a un grupo de seguridad que permite tráfico HTTP/HTTPS desde Internet.

3. Instancias de la Aplicación:

- Las instancias de la aplicación están en subredes privadas y tienen IPs privadas (10.0.3.85 y 10.0.4.172).
- Estas instancias están asociadas a un grupo de seguridad que permite tráfico HTTP solo desde el ALB.

4. Instancia de MongoDB:

- La instancia de MongoDB también está en una subred privada y tiene una IP privada (10.0.3.133).
- Está asociada a un grupo de seguridad que permite tráfico en el puerto 27017 solo desde las instancias de la aplicación.

5. NAT Gateway:

- El NAT Gateway permite que las instancias en las subredes privadas (instancias de la aplicación y MongoDB) accedan a Internet para actualizaciones o descargas de paquetes.
- Tiene una IP pública (54.175.213.237) que se utiliza para el tráfico saliente.

6. Subredes:

- Subredes públicas: Contienen el ALB.
- Subredes privadas: Contienen las instancias de la aplicación y MongoDB.

Criterio 1

Estructura del proyecto

El proyecto está organizado de forma modular para garantizar su mantenibilidad, escalabilidad y reutilización. A continuación, se describe la estructura de carpetas y archivos creados:

```
.
├── main.tf              # Archivo principal que define los módulos y proveedores
├── variables.tf         # Variables globales del proyecto
├── outputs.tf           # Salidas del proyecto (IPs, DNS, etc.)
├── terraform.tfvars     # Valores de las variables definidas en variables.tf
├── versions.tf          # Versiones de Terraform y proveedores
└── modules/             # Carpeta que contiene los módulos de Terraform
    ├── vpc/             # Módulo para la creación de la VPC y subredes
    │   ├── main.tf
    │   ├── variables.tf
    │   └── outputs.tf
    ├── security/        # Módulo para la creación de grupos de seguridad
    │   ├── main.tf
    │   ├── variables.tf
    │   └── outputs.tf
    ├── loadbalancer/    # Módulo para la creación del balanceador de carga (ALB)
    │   ├── main.tf
    │   ├── variables.tf
    │   └── outputs.tf
```

```
├── compute/           # Módulo para la creación de instancias EC2 (app y MongoDB)
│   ├── main.tf
│   ├── variables.tf
│   └── outputs.tf
└── nat/               # Módulo para la creación del NAT Gateway
    ├── main.tf
    ├── variables.tf
    └── outputs.tf
```

A continuacion se explica el contenido de los diferentes templates del proyecto:

main.tf (raiz):

El archivo main.tf en la raíz del proyecto es el punto de entrada principal de la configuración de Terraform. Este archivo se encarga de integrar y orquestar todos los módulos que componen la infraestructura, definiendo cómo interactúan entre sí y con los recursos de AWS. A continuación, se explica su funcionamiento:

1. Proveedor de AWS

El archivo comienza configurando el proveedor de AWS, que es el plugin de Terraform que permite interactuar con los servicios de AWS. Aquí se especifica la región donde se desplegarán los recursos.

```
provider "aws" {
  region = var.aws_region
}
```

2. MóduloVPC

Este módulo crea la VPC (Virtual Private Cloud) y sus componentes asociados, como subredes públicas y privadas, tablas de rutas y el Internet Gateway. Crea la red virtual (VPC) y divide el espacio de direcciones IP en subredes públicas y privadas.

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

3. Módulo security

Este módulo define los grupos de seguridad que actúan como firewalls virtuales para controlar el tráfico entrante y saliente de los recursos. Crea grupos de seguridad para el ALB, las instancias de la aplicación y MongoDB.

```
module "security" {
 source = "./modules/security"
 vpc_id = module.vpc.vpc_id
}
```

4. Módulo loadbalancer

Este módulo crea el Application Load Balancer (ALB), el Target Group y el Listener para distribuir el tráfico entre las instancias de la aplicación. Configura el balanceador de carga para distribuir el tráfico entre las instancias de la aplicación.

```
module "loadbalancer" {
 source          = "./modules/loadbalancer"
 vpc_id          = module.vpc.vpc_id
 public_subnets    = module.vpc.public_subnet_ids
 security_group_id = module.security.alb_security_group_id
 app_instances     = module.compute.app_instance_ids
}
```

5. Módulo compute

Este módulo se encarga de crear las instancias EC2 para la aplicación (Nginx + Node.js) y MongoDB. Despliega las instancias de la aplicación en subredes privadas y la instancia de MongoDB en una subred privada.

```
module "compute" {
 source                = "./modules/compute"
 vpc_id                = module.vpc.vpc_id
 public_subnet_ids       = module.vpc.public_subnet_ids
 private_subnet_ids      = module.vpc.private_subnet_ids
 app_security_group_id     = module.security.app_security_group_id
 mongodb_security_group_id = module.security.mongodb_security_group_id
 app_instance_count      = var.app_instance_count
 app_instance_type       = var.app_instance_type
 mongodb_instance_type     = var.mongodb_instance_type
 app_ami                 = var.app_ami
 mongodb_ami             = var.mongodb_ami
```

```
}
```

6. Módulo nat

Este módulo crea un NAT Gateway para permitir que las instancias en subredes privadas accedan a Internet. Permite que las instancias en subredes privadas accedan a Internet para actualizaciones o descargas de paquetes.

```
module "nat" {
 source          = "./modules/nat"
 vpc_id          = module.vpc.vpc_id
 public_subnet_id   = module.vpc.public_subnet_ids[0]
 private_subnet_ids = module.vpc.private_subnet_ids
}
```

variables.tf (raiz):

El archivo variables.tf define las variables que se utilizan en el proyecto de Terraform. Estas variables permiten personalizar el despliegue de la infraestructura sin modificar directamente el código, lo que facilita la reutilización y adaptación del proyecto.

```
variable "aws_region" {
 description = "Region de AWS para implementar la infraestructura"
 type       = string
 default    = "us-west-2"
}

variable "vpc_cidr" {
 description = "Bloque CIDR para la VPC"
 type       = string
 default    = "10.0.0.0/16"
}

variable "public_subnets" {
 description = "Lista de bloques CIDR para subredes publicas"
 type       = list(string)
 default    = ["10.0.1.0/24", "10.0.2.0/24"]
}

variable "private_subnets" {
 description = "Lista de bloques CIDR para subredes privadas"
 type       = list(string)
 default    = ["10.0.3.0/24", "10.0.4.0/24"]
```

```
}

variable "app_instance_count" {
 description = "Numero de instancias de aplicacion (nginx + node)"
 type        = number
 default     = 2
}

variable "app_instance_type" {
 description = "Tipo de instancia para servidores de aplicacion (nginx + node)"
 type        = string
 default     = "t2.micro"
}

variable "mongodb_instance_type" {
 description = "Tipo de instancia para servidor de MongoDB"
 type        = string
 default     = "t2.micro"
}

variable "app_ami" {
 description = "ID de AMI para instancias de aplicacion (nginx + node)"
 type        = string
}

variable "mongodb_ami" {
 description = "ID de AMI para instancia de MongoDB"
 type        = string
}
```

terraform.tfvars (raiz):

El archivo terraform.tfvars es un archivo de configuración de variables en Terraform. Su función principal es asignar valores concretos a las variables definidas en el archivo variables.tf.

```
aws_region = "us-east-1"  # Región de AWS donde se desplegará la infraestructura
vpc_cidr = "10.0.0.0/16"  # Bloque CIDR principal de la VPC
# Definición de subredes privadas
private_subnets = ["10.0.3.0/24", "10.0.4.0/24"]
# Dos subredes privadas en diferentes zonas de disponibilidad

# Número de instancias de aplicación a desplegar
```

```
app_instance_count = 2

# Tipo de instancia para servidores de aplicación
app_instance_type = "t2.micro"

# Tipo de instancia para servidor MongoDB
mongodb_instance_type = "t2.micro"

# AMI para instancias de aplicación (Nginx + Node.js)
app_ami = "ami-0764ed87e47544185"
# AMI generada previamente con Packer

# AMI para instancia MongoDB
mongodb_ami = "ami-004e8f8be5386db67"
# AMI pública de Bitnami
```

Módulo VPC

module/vpc/main.tf

1. Creacion de la VPC

```
resource "aws_vpc" "main" {
  cidr_block = var.vpc_cidr
  enable_dns_hostnames = true
  enable_dns_support = true

  tags = {
    Name = "mean-stack-vpc"
  }
}
```

cidr_block: Define el rango de direcciones IP para la VPC.

enable_dns_hostnames y enable_dns_support: Habilitan el soporte DNS para la VPC.

tags: Asigna un nombre descriptivo a la VPC.

2. Creacion de Subredes Publicas

```
resource "aws_subnet" "public" {
  count = length(var.public_subnets)
  vpc_id = aws_vpc.main.id
  cidr_block = var.public_subnets[count.index]
  availability_zone = data.aws_availability_zones.available.names[count.index]
  map_public_ip_on_launch = true
```

```
 tags = {
  Name = "public-subnet-${count.index + 1}"
 }
}
```

- count: Crea una subred por cada bloque CIDR definido en var.public_subnets.
- cidr_block: Asigna el bloque CIDR correspondiente a cada subred.
- availability_zone: Distribuye las subredes en diferentes zonas de disponibilidad.
- map_public_ip_on_launch: Asigna automáticamente una IP pública a las instancias lanzadas en estas subredes.
- tags: Asigna nombres descriptivos a las subredes.

3. Creación de Subredes Privadas

```
resource "aws_subnet" "private" {
 count = length(var.private_subnets)
 vpc_id = aws_vpc.main.id
 cidr_block = var.private_subnets[count.index]
 availability_zone = data.aws_availability_zones.available.names[count.index]

 tags = {
  Name = "private-subnet-${count.index + 1}"
 }
}
```

- Similar a las subredes públicas, pero sin asignar IPs públicas automáticamente.

4. Creación del Internet Gateway (IGW)

```
resource "aws_internet_gateway" "main" {
 vpc_id = aws_vpc.main.id

 tags = {
  Name = "mean-stack-igw"
 }
}
```

- vpc_id: Asocia el IGW con la VPC creada.

5. Creación de la Tabla de Rutas Pública

```
resource "aws_route_table" "public" {
```

```
 vpc_id = aws_vpc.main.id

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

 tags = {
  Name = "mean-stack-public-rt"
 }
}
```

- route: Define una ruta predeterminada (0.0.0.0/0) que redirige el tráfico a Internet a través del IGW.

6. Asociación de Subredes Públicas con la Tabla de Rutas

```
resource "aws_route_table_association" "public" {
 count = length(aws_subnet.public)
 subnet_id = aws_subnet.public[count.index].id
 route_table_id = aws_route_table.public.id
}
```

- subnet_id: Identificador de la subred pública.
- route_table_id: Identificador de la tabla de rutas pública.

7. Obtención de Zonas de Disponibilidad

```
data "aws_availability_zones" "available" {
 state = "available"
}
```

- Obtiene las zonas de disponibilidad disponibles en la región especificada.

Módulo Security

module/security/main.tf

1. Grupo de Seguridad para el ALB (Application Load Balancer)

```
resource "aws_security_group" "alb" {
 name = "mean-alb-sg"
 vpc_id = var.vpc_id
```

```
 ingress {
  from_port = 80
  to_port = 80
  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"]
 }
}
```

- ingress: Permite tráfico HTTP (puerto 80) desde cualquier dirección IP (0.0.0.0/0).
- egress: Permite todo el tráfico saliente.
- vpc_id: Asocia el grupo de seguridad con la VPC.

2. Grupo de Seguridad para las Instancias de la Aplicación

```
resource "aws_security_group" "app" {
 name = "mean-app-sg"
 vpc_id = var.vpc_id

 ingress {
  from_port = 80
  to_port = 80
  protocol = "tcp"
  security_groups = [aws_security_group.alb.id]
 }

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

- ingress: Permite tráfico HTTP (puerto 80) solo desde el grupo de seguridad del ALB.
- egress: Permite todo el tráfico saliente.
- vpc_id: Asocia el grupo de seguridad con la VPC.

3. Grupo de Seguridad para MongoDB

```
resource "aws_security_group" "mongodb" {
 name = "mean-mongodb-sg"
 vpc_id = var.vpc_id

 ingress {
  from_port = 27017
  to_port = 27017
  protocol = "tcp"
  security_groups = [aws_security_group.app.id]
 }

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

- ingress: Permite tráfico en el puerto 27017 (MongoDB) solo desde el grupo de seguridad de la aplicación.
- egress: Permite todo el tráfico saliente.
- vpc_id: Asocia el grupo de seguridad con la VPC.

Módulo compute

module/compute/main.tf

1. Creación de Instancias de la Aplicación (Nginx + Node.js)

```
resource "aws_instance" "app" {
 count = var.app_instance_count
 ami = var.app_ami
 instance_type = var.app_instance_type
 subnet_id = var.private_subnet_ids[count.index % length(var.private_subnet_ids)]
```

```
 vpc_security_group_ids = [var.app_security_group_id]

 tags = {
  Name = "mean-app-${count.index + 1}"
 }
}
```

- count: Define el número de instancias a crear, según var.app_instance_count.
- ami: Especifica la AMI (Amazon Machine Image) para las instancias de la aplicación.
- instance_type: Define el tipo de instancia EC2 (por ejemplo, t2.micro).
- subnet_id: Asigna las instancias a subredes privadas.
- vpc_security_group_ids: Asocia las instancias con el grupo de seguridad de la aplicación.

2. Creación de la Instancia de MongoDB

```
resource "aws_instance" "mongodb" {
 ami = var.mongodb_ami
 instance_type = var.mongodb_instance_type
 subnet_id = var.private_subnet_ids[0]
 vpc_security_group_ids = [var.mongodb_security_group_id]

 tags = {
  Name = "mean-mongodb"
 }
}
```

- ami: Especifica la AMI para la instancia de MongoDB.
- instance_type: Define el tipo de instancia EC2 (por ejemplo, t2.micro).
- subnet_id: Asigna la instancia a una subred privada.
- vpc_security_group_ids: Asocia la instancia con el grupo de seguridad de MongoDB.
- tags: Asigna un nombre descriptivo a la instancia.

Módulo nat

module/nat/main.tf

1. Creación de una IP Elástica (EIP) para el NAT Gateway

```
resource "aws_eip" "nat" {
 domain = "vpc"
}
```

- domain = "vpc": Especifica que la IP elástica se usará dentro de una VPC.

2. Creación del NAT Gateway

```
resource "aws_nat_gateway" "main" {
 allocation_id = aws_eip.nat.id
 subnet_id = var.public_subnet_id

 tags = {
  Name = "mean-stack-nat"
 }
}
```

- allocation_id: Asocia la IP elástica creada anteriormente con el NAT Gateway.
- subnet_id: Especifica la subred pública donde se desplegará el NAT Gateway.
- tags: Asigna un nombre descriptivo al NAT Gateway.

3. Creación de la Tabla de Rutas Privada

```
resource "aws_route_table" "private" {
 vpc_id = var.vpc_id

 route {
  cidr_block = "0.0.0.0/0"
  nat_gateway_id = aws_nat_gateway.main.id
 }

 tags = {
  Name = "mean-stack-private-rt"
 }
}
```

- route: Define una ruta predeterminada (0.0.0.0/0) que redirige el tráfico saliente a Internet a través del NAT Gateway.
- tags: Asigna un nombre descriptivo a la tabla de rutas.

4. Asociación de Subredes Privadas con la Tabla de Rutas

```
resource "aws_route_table_association" "private" {
```

```
 count = length(var.private_subnet_ids)
 subnet_id = var.private_subnet_ids[count.index]
 route_table_id = aws_route_table.private.id
}
```

- subnet_id: Identificador de la subred privada.
- route_table_id: Identificador de la tabla de rutas privada.

El móduloBalanceador de carga de describe en la sección Criterio 2.

Criterio 2

Balanceador de carga (módulo loadbalancer):

El módulo loadbalancer tiene como objetivo crear y configurar un Application Load Balancer (ALB) en AWS. El ALB distribuye el tráfico entrante entre las instancias de la aplicación, mejorando la disponibilidad, escalabilidad y tolerancia a fallos. Este módulo también configura un Target Group (grupo de destino) y un Listener (oyente) para redirigir el tráfico a las instancias correctas.

module/loadbalancer/main.tf:

El archivo main.tf dentro del módulo loadbalancer define los recursos necesarios para crear y configurar el ALB, el Target Group y el Listener. A continuación, se explica cada porción del código:

1. Creación del Application Load Balancer (ALB)

```
resource "aws_lb" "app" {
 name            = "mean-stack-alb"
 internal        = false
 load_balancer_type = "application"
 security_groups    = [var.security_group_id]
 subnets         = var.public_subnets

 tags = {
  Name = "mean-stack-alb"
 }
}
```

- name: Asigna un nombre al ALB.

- internal: Define si el ALB es interno (true) o público (false).
- load_balancer_type: Especifica que es un balanceador de carga de tipo "application".
- security_groups: Asocia el ALB con un grupo de seguridad para controlar el tráfico.
- subnets: Especifica las subredes públicas donde se desplegará el ALB.
- tags: Asigna un nombre descriptivo al ALB.

2. Creación del Target Group (Grupo de Destino)

```
resource "aws_lb_target_group" "app" {
 name     = "mean-stack-tg"
 port     = 80
 protocol = "HTTP"
 vpc_id   = var.vpc_id

 health_check {
  path = "/"
  port = "traffic-port"
 }
}
```

- El Target Group que define a qué instancias se redirigirá el tráfico.
- name: Asigna un nombre al Target Group.
- port: Especifica el puerto de destino (en este caso, 80 para HTTP).
- protocol: Define el protocolo de comunicación (HTTP).
- vpc_id: Asocia el Target Group con la VPC.
- health_check: Configura un chequeo de salud para verificar el estado de las instancias

3. Creación del Listener

```
resource "aws_lb_listener" "app" {
 load_balancer_arn = aws_lb.app.arn
 port              = 80
 protocol          = "HTTP"

 default_action {
```

```
  type             = "forward"
  target_group_arn = aws_lb_target_group.app.arn
 }
}
```

- Crea un Listener que redirige el tráfico entrante al Target Group.
- load_balancer_arn: Asocia el Listener con el ALB.
- port: Especifica el puerto de escucha (en este caso, 80 para HTTP).
- protocol: Define el protocolo de comunicación (HTTP).
- default_action: Configura la acción predeterminada, que es redirigir el tráfico al Target Group.

4. Registro de Instancias en el Target Group

```
resource "aws_lb_target_group_attachment" "app" {
 count            = length(var.app_instances)
 target_group_arn = aws_lb_target_group.app.arn
 target_id        = var.app_instances[count.index]
 port             = 80
}
```

- count: Itera sobre la lista de instancias de la aplicación.
- target_group_arn: Asocia las instancias con el Target Group.
- target_id: Especifica el ID de la instancia a registrar.
- port: Define el puerto de destino (en este caso, 80 para HTTP)

Despues de ejecutar terraform apply, se tiene en outputs el dns del balanceador de carga, el cual es “mean-stack-alb-9565889.us-east-1.elb.amazonaws.com” (El resultado total que se genera en outputs se detalla en la sección Criterio 3). Al acceder al dns del balanceador desde el navegador se tiene:

http://mean-stack-alb-9565889.us-east-1.elb.amazonaws.com/

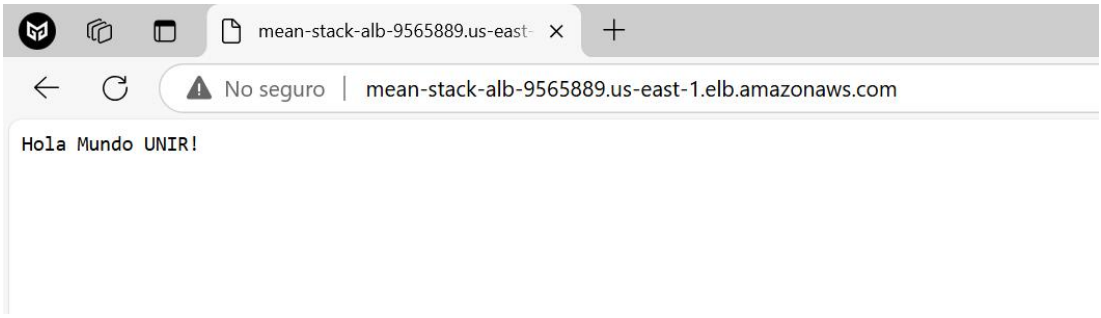

Figura 2. Resultado al ingresar al dns de ALB en el navegador

Criterio 3

Output Terraform

El archivo outputs.tf en la raíz del proyecto define las salidas (outputs) de Terraform. Estas salidas permiten extraer información importante sobre los recursos creados, como direcciones IP, nombres DNS y otros datos relevantes. Estos valores se pueden utilizar para consultas posteriores, integración con otras herramientas o simplemente para mostrar información útil al usuario después de aplicar la configuración de Terraform.

outputs.tf (raiz):

1. app_private_ips

```
output "app_private_ips" {
  description = "IPs privadas de las instancias de la aplicacion (nginx + node)"
  value       = module.compute.app_private_ips
}
```

- Muestra las IPs privadas de las instancias de la aplicación (Nginx + Node.js).
- module.compute.app_private_ips: Obtiene las IPs privadas de las instancias de la aplicación desde el módulo compute.

2. mongodb_private_ip

```
output "mongodb_private_ip" {
  description = "IP privada de la instancia de MongoDB"
  value       = module.compute.mongodb_private_ip
}
```

- Muestra la IP privada de la instancia de MongoDB.

- module.compute.mongodb_private_ip: Obtiene la IP privada de la instancia de MongoDB desde el módulo compute.
- Esta IP es necesaria para que las instancias de la aplicación se conecten a la base de datos.

3. alb_dns

```
output "alb_dns" {
 description = "Nombre DNS del balanceador de carga de la aplicacion"
 value       = module.loadbalancer.alb_dns
}
```

- Muestra el nombre DNS del Application Load Balancer (ALB).
- module.loadbalancer.alb_dns: Obtiene el nombre DNS del ALB desde el módulo loadbalancer.
- Este nombre DNS es el punto de entrada público para acceder a la aplicación desde Internet.

4. nat_public_ip

```
output "nat_public_ip" {
 description = "IP pública del NAT Gateway"
 value       = module.nat.nat_public_ip
}
```

- Muestra la IP pública del NAT Gateway.
- module.nat.nat_public_ip: Obtiene la IP pública del NAT Gateway desde el módulo nat.
- Esta IP es útil para verificar la configuración del NAT Gateway y su conectividad con Internet.

Cómo se Generan las Salidas

- Las salidas obtienen sus valores de los módulos correspondientes (compute, loadbalancer, nat).
- Cada módulo define sus propias salidas internas (por ejemplo, app_private_ips en el módulo compute), que luego se exponen en el archivo outputs.tf de la raíz.

- La clave value en cada bloque output especifica de dónde se obtiene el valor. Por ejemplo, module.compute.app_private_ips obtiene las IPs privadas de las instancias de la aplicación desde el módulo compute.

A continuación los outputs de cada modulo:

modules/vpc/output.tf:

```
output "vpc_id" {
 description = "ID de la VPC creada"
 value       = aws_vpc.main.id
}

output "public_subnet_ids" {
 description = "IDs de subredes publicas"
 value       = aws_subnet.public[*].id
}

output "private_subnet_ids" {
 description = "IDs de subredes privadas"
 value       = aws_subnet.private[*].id
}

output "public_subnet_cidrs" {
 description = "Bloques CIDR de subredes públicas"
 value       = aws_subnet.public[*].cidr_block
}
```

modules/security/output.tf:

```
output "alb_security_group_id" {
 description = "ID del grupo de seguridad del ALB"
 value       = aws_security_group.alb.id
}

output "app_security_group_id" {
 description = "ID del grupo de seguridad de la aplicacion (nginx + node)"
 value       = aws_security_group.app.id
}

output "mongodb_security_group_id" {
 description = "ID del grupo de seguridad de MongoDB"
 value       = aws_security_group.mongodb.id
}
```

modules/loadbalancer/output.tf:

```
output "alb_arn" {
  description = "ARN del Balanceador de Carga de Aplicaciones"
  value       = aws_lb.app.arn
}

output "alb_dns" {
  description = "Nombre DNS del Balanceador de Carga de Aplicaciones"
  value       = aws_lb.app.dns_name
}

output "target_group_arn" {
  description = "ARN del grupo de destino"
  value       = aws_lb_target_group.app.arn
}
```

modules/compute/output.tf:

```
output "app_instance_ids" {
  description = "IDs de instancias de aplicación"
  value       = aws_instance.app[*].id
}

output "app_public_ips" {
  description = "IPs públicas de instancias de aplicacion (nginx + node)"
  value       = aws_instance.app[*].public_ip
}

output "app_private_ips" {
  description = "IPs privadas de instancias de aplicacion (nginx + node)"
  value       = aws_instance.app[*].private_ip
}

output "mongodb_instance_id" {
  description = "ID de instancia de MongoDB"
  value       = aws_instance.mongodb.id
}

output "mongodb_private_ip" {
  description = "IP privada de instancia de MongoDB"
  value       = aws_instance.mongodb.private_ip
}
```

modules/nat/output.tf:

```
output "nat_public_ip" {
 description = "IP pública del NAT Gateway"
 value       = aws_eip.nat.public_ip
}
output "nat_gateway_id" {
 description = "ID del NAT Gateway"
 value       = aws_nat_gateway.main.id
}
```

Al ejecutar el comando “terraform apply”, la infraestructura se crea correctame. A continuacion se presenta una captura del log generado, y en donde se presenta la salidas definidas en el archivo outputs.tf (raiz).

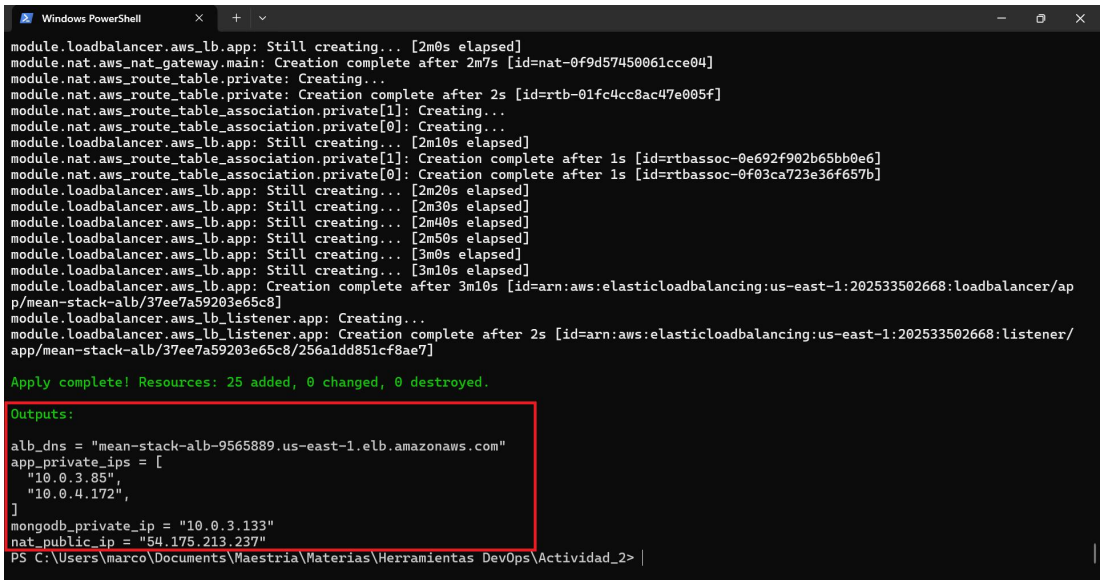

Figura 3. Salida de la ejecución del comando “terraform apply”

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Gutama Morocho	26/01/2024
	Nombre: Marco Paúl	

Verficaciones infraestructura creada

1. Verificar ALB

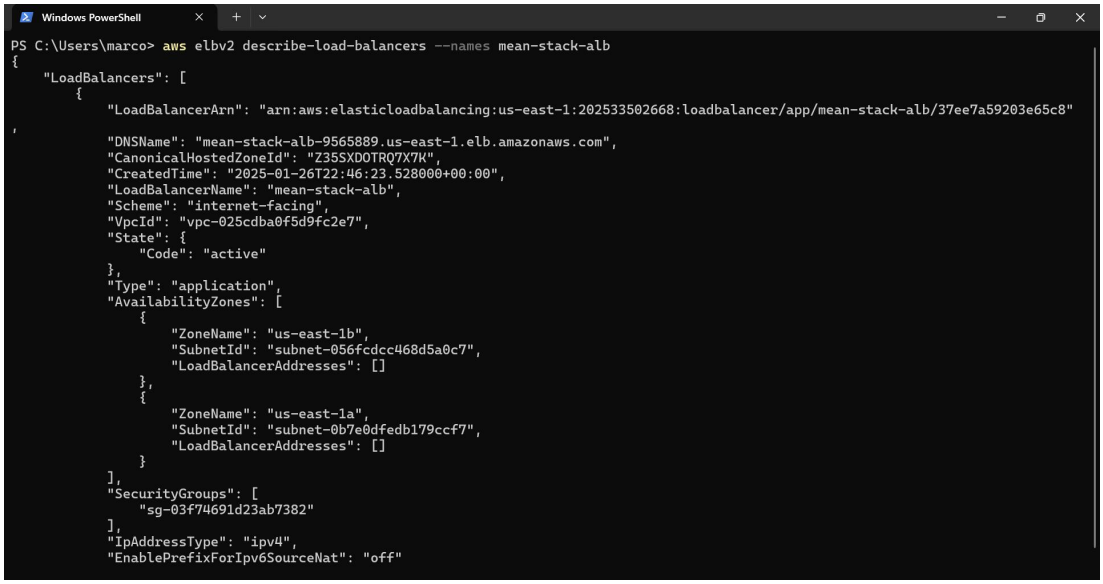

Figura 4. Estado del ALB

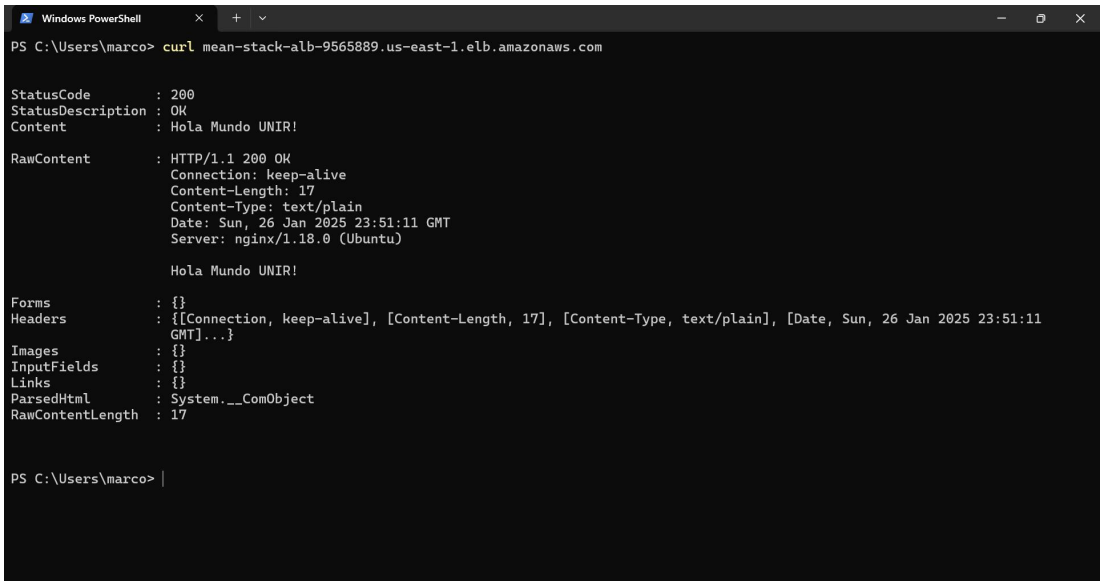

Figura 5. Test conectividad ALB

Se valida el correcto estado del ALB, asi como la respuesta de la aplicacion al acceder al dns del ALB.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Gutama Morocho	26/01/2024
	Nombre: Marco Paúl	

2. Verificar instancias

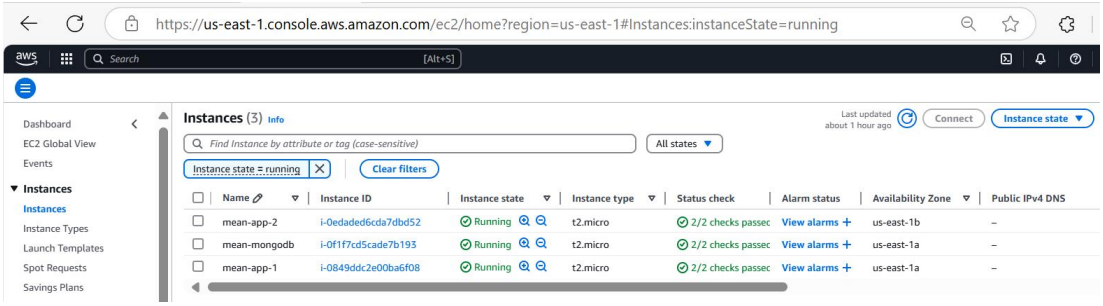

Figura 6. Instancias en paltaforma AWS

Se valida que las instancias han sido creadas y estan ejecutandose correctamente.

3. Verficar networking

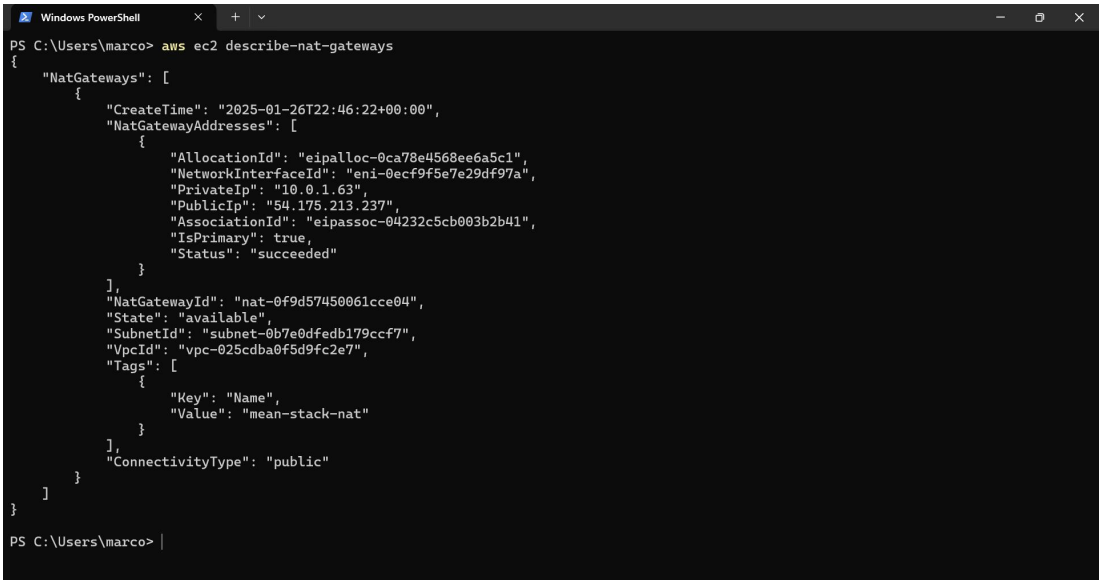

Figura 7. Estado NAT Gateway

El estado del NAT Gateway ha sido creado y configurado correctamente.