**UNIVERSIDAD INTERNACIONAL DE LA RIOJA (UNIR)**  
*La Universidad en Internet*

*Herramientas DevOps*

*(MEXDEVOPS)*

**Maestría en Desarrollo y Operaciones de Software**  
**2025**

Actividad grupal: Despliegue de MEAN multicapa mediante Terraform

Equipo 2\_H

Índice

[Plantemento del proyecto 4](#_Toc201351838)

[Instalación de Terraform 4](#_Toc201351839)

[Ejercicio 1: Modularización y grupos de seguridad 5](#_Toc201351840)

[Creción de un proyecto de Terraform 5](#_Toc201351841)

[Estructura del proyecto 5](#_Toc201351842)

[Selección de nube 5](#_Toc201351843)

[Terraform plan 6](#_Toc201351844)

[Creación de un Key-Pair 6](#_Toc201351845)

[Obtención de una AMI adecuada 6](#_Toc201351846)

[Creación de los recursos 7](#_Toc201351847)

[Resultados ejercicio 1 7](#_Toc201351848)

[Eliminar recursos 8](#_Toc201351849)

[Ejercicio 2: Balanceador de carga en AWS con Terraform 9](#_Toc201351850)

[Justificación y diseño 9](#_Toc201351851)

[Implementación con Terraform 9](#_Toc201351852)

[Subredes públicas 10](#_Toc201351853)

[Asociación de instancias y target group 10](#_Toc201351854)

[Problemas encontrados y solución 10](#_Toc201351855)

[Resultados ejercicio 2 11](#_Toc201351856)

[Ejercicio 3: Outputs de Terraform y NAT Gateway 12](#_Toc201351857)

[¿Por qué exponer estos outputs? 12](#_Toc201351858)

[Cambios realizados: Uso del NAT Gateway 12](#_Toc201351859)

[Resultado ejercicio 3 12](#_Toc201351860)

[Conclusión 13](#_Toc201351861)

[Estructura final del proyecto: 13](#_Toc201351862)

[Referencias 14](#_Toc201351863)

# Plantemento del proyecto

En esta actividad, el objetivo es desplegar una arquitectura MEAN multicapa en AWS utilizando Terraform. El propósito es familiarizarnos con la infraestructura como código (IaC) y la modularización de recursos en la nube (AWS), permitiendo la creación, configuración y destrucción de entornos completos de manera reproducible y automatizada.

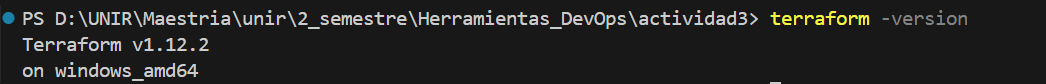
El stack MEAN se dividirá en dos capas principales:

* Capa de aplicación: Una instancia EC2 que ejecuta Node.js y Nginx, donde se encuentra la aplicación.
* Capa de base de datos: Una instancia EC2 que ejecuta MongoDB.

# Instalación de Terraform

En nuestro caso instalaremos Terraform con la herramienta choco en Windows.

Texto

El contenido generado por IA puede ser incorrecto. 

# Ejercicio 1: Modularización y grupos de seguridad

## Creación de un proyecto de Terraform

Para eso es necesario dentro de la carpeta donde vamos a tener nuestro código de Terraform usar terraform init:

Texto

El contenido generado por IA puede ser incorrecto.

## Estructura del proyecto

Para mantener el código organizado y reutilizable, se ha optado por una estructura modular. Cada módulo encapsula la lógica de un componente específico de la infraestructura:

* modules/network: VPC, subred y gateway.
* modules/sg: Define los grupos de seguridad para la app y la base de datos.
* modules/ec2\_app: Lanza la instancia EC2 para la aplicación, instala Node.js, Nginx y la app.
* modules/ec2\_db: Lanza la instancia EC2 para MongoDB y la inicializa.
* main.tf: Orquesta el despliegue, conectando los módulos y pasando las variables necesarias.

## Selección de nube

La nube que seleccionamos para el desarrollo del proyecto es AWS lo cual los objetos usados dentro de nuestros templates de Terraform son específicos para la nube de Amazon. Es importante mencionar que se usa también un usuario que tiene los suficientes permisos y acceso a uso de aws cli para que nuestro proyecto de Terraform pueda crear los recursos desde nuestra máquina.

## Terraform plan

Antes de aplicar nuestro proyecto de terraform es recomendable usar el comando terraform plan ya que nos ayuda a previsualizar los cambios que se realizarán a la infraestructura según la configuración de nuestro código:

Texto

El contenido generado por IA puede ser incorrecto.

### Creación de un Key-Pair

Para que se pueda acceder las máquinas es necesario la creación de un key-pair para que durante la creación de las EC2 con Terraform este pueda configurarlas y así nosotros posteriormente podamos acceder a ellas sin problemas, para ello podemos hacerlo fácilmente dentro de la consola de AWS:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

### Obtención de una AMI adecuada

Usaremos máquinas Ubuntu 20.04 y para obtener la id podemos usar el siguiente comando:

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

## Creación de los recursos

Una vez comprobado que la planeación es correcta podemos aplicar nuestras plantillas y ver los resultados.

terraform apply

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

## Resultados ejercicio 1

Las máquinas son creadas que son inicializados con scripts que preparan tanto la aplicación como la base de datos, las redes y security groups.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

### Eliminar recursos

Para evitar que nos cobren una vez terminado de probar que todo funcionó podemos eliminarlo con: terraform destroy

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

# Ejercicio 2: Balanceador de carga en AWS con Terraform

En este ejercicio, extendimos la infraestructura del stack MEAN desplegada en el Ejercicio 1 para incluir un balanceador de carga (Application Load Balancer, ALB) en AWS. El objetivo es distribuir el tráfico HTTP entrante entre las instancias de la capa de aplicación, mejorando la disponibilidad y escalabilidad del sistema.

## Justificación y diseño

El uso de un Application Load Balancer permite:

* Balancear el tráfico entre varias instancias de aplicación.
* Mejorar la tolerancia a fallos y la disponibilidad.
* Facilitar el escalado horizontal.
* Centralizar la entrada de tráfico HTTP/HTTPS.

En AWS, el ALB requiere al menos dos subredes públicas en diferentes zonas de disponibilidad para garantizar alta disponibilidad.

## Implementación con Terraform

Se creó un módulo específico para el ALB (modules/alb). El módulo recibe como parámetros el VPC, las subredes, el security group y las instancias objetivo.

**Fragmento de código del módulo ALB:**

resource "aws\_lb" "this" {

name = "mean-alb"

internal = false

load\_balancer\_type = "application"

security\_groups = [var.app\_sg\_id]

subnets = var.subnet\_ids

}

### Subredes públicas

Durante la implementación, AWS arrojó el siguiente error:

ValidationError: At least two subnets in two different Availability Zones must be specified

Esto nos llevó a modificar el módulo de red para crear dos subredes públicas en diferentes zonas de disponibilidad:

resource "aws\_subnet" "public\_a" {

...

availability\_zone = data.aws\_availability\_zones.available.names[0]

}

resource "aws\_subnet" "public\_b" {

...

availability\_zone = data.aws\_availability\_zones.available.names[1]

}

Las subredes se asociaron a la tabla de rutas pública para tener acceso a Internet.

### Asociación de instancias y target group

El ALB enruta el tráfico al grupo de destino (target group), que contiene la(s) instancia(s) de la aplicación:

resource "aws\_lb\_target\_group\_attachment" "app" {

count = length(var.target\_instance\_ids)

target\_group\_arn = aws\_lb\_target\_group.this.arn

target\_id = var.target\_instance\_ids[count.index]

port = 80

}

## Problemas encontrados y solución

* Error de subredes insuficientes: El principal reto fue el requisito de AWS de tener dos subredes en diferentes zonas de disponibilidad para el ALB.
* Solución: Se modificó el módulo de red para crear y asociar dos subredes públicas.
* Asociación de tabla de rutas: Fue necesario asociar ambas subredes a la tabla de rutas pública para garantizar el acceso externo.

## Resultados ejercicio 2

El balanceador de carga se creó correctamente.

Al acceder a ese DNS desde el navegador, la aplicación responde correctamente, demostrando que el tráfico es balanceado y la arquitectura es altamente disponible.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

<http://mean-alb-457328202.us-west-1.elb.amazonaws.com/productos>

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

# Ejercicio 3: Outputs de Terraform y NAT Gateway

El objetivo de este ejercicio es exponer, mediante un archivo outputs.tf, los datos clave de la infraestructura desplegada con Terraform:

* IP públicas y privadas de cada nodo (aplicación y base de datos)
* DNS del balanceador de carga (ALB)
* IP pública del NAT Gateway utilizado para la salida a Internet de la base de datos

### ¿Por qué exponer estos outputs?

Los outputs de Terraform permiten obtener de forma automática y ordenada la información esencial de la infraestructura, facilitando la verificación, la documentación y el acceso a los recursos desplegados. Esto es especialmente útil en entornos colaborativos y automatizados, donde los datos de red y acceso deben estar disponibles para otros equipos o scripts.

## Cambios realizados: Uso del NAT Gateway

En la arquitectura inicial, tanto la aplicación como la base de datos estaban en subredes públicas, lo que permitía el acceso directo a Internet. Sin embargo, por buenas prácticas de seguridad, es recomendable que la base de datos esté en una subred privada, sin acceso directo desde Internet.

Para que la instancia de MongoDB (en la subred privada) pueda acceder a Internet (por ejemplo, para descargar actualizaciones o paquetes), se implementó un NAT Gateway en una subred pública. El NAT Gateway permite que los recursos en subredes privadas realicen conexiones salientes a Internet, pero evita conexiones entrantes no autorizadas.

## Resultado ejercicio 3

Tras ejecutar terraform apply, los outputs generados fueron:

Texto

El contenido generado por IA puede ser incorrecto.

# Conclusión

A lo largo de esta actividad grupal logramos desplegar una arquitectura MEAN multicapa en AWS utilizando Terraform, aplicando buenas prácticas de modularización y seguridad. En el primer ejercicio, estructuramos la infraestructura en módulos reutilizables y configuramos los grupos de seguridad para aislar correctamente la aplicación y la base de datos. En el segundo ejercicio, integramos un Application Load Balancer, mejorando la disponibilidad y escalabilidad del sistema, y aprendimos la importancia de contar con subredes en distintas zonas de disponibilidad. Finalmente, en el tercer ejercicio, expusimos los datos clave de la infraestructura mediante outputs de Terraform, y reforzamos la seguridad moviendo la base de datos a una subred privada con acceso a Internet a través de un NAT Gateway.

### Estructura final del proyecto:

Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

# Referencias

* HashiCorp. (2025). Terraform Documentation. https://developer.hashicorp.com/terraform/docs
* HashiCorp. (2025). Terraform AWS Provider Documentation. https://registry.terraform.io/providers/hashicorp/aws/latest/docs
* Amazon Web Services. (2025). Application Load Balancer. https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html
* Amazon Web Services. (2025). VPCs and Subnets. https://docs.aws.amazon.com/vpc/latest/userguide/VPC\_Subnets.html
* Amazon Web Services. (2025). NAT Gateways. https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html
* Amazon Web Services. (2025). Security Groups for Your VPC. https://docs.aws.amazon.com/vpc/latest/userguide/VPC\_SecurityGroups.html
* MongoDB Inc. (2025). MongoDB Manual. https://www.mongodb.com/docs/manual/
* Terraform AWS Modules. (2025). terraform-aws-ec2-instance. https://github.com/terraform-aws-modules/terraform-aws-ec2-instance
* Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley.
* Fowler, M. (2013). Infrastructure as Code. https://martinfowler.com/bliki/InfrastructureAsCode.html