

Asignatura	Datos del alumno	Fecha
<b>Herramientas DevOps</b>	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## Actividad 2. Despliegue de MEAN multicapa mediante Terraform

1. Resumen del sistema general que hay que desplegar
2. Implementación de Packer en el proyecto
3. Proceso de creación y despliegue
4. Problema 1: Despliegue en Azure, incompatibilidad de versiones y porqué AWS
5. Angular en producción y proceso de despliegue del frontend
6. Problema 2: Ejecución de comando interactivo que bloquea el proceso automático de aprovisionamiento
7. Funcionamiento de la conexión entre frontend, backend y base de datos
8. **Criterio 1: Template de Terraform con modularización y grupos de seguridad**
9. **Criterio 2: Inclusión de balanceador de carga**
10. Problema 3: Inconsistencia de recursos estáticos y cierre de sesión del balanceador
11. **Criterio 3: Outputs**
12. **Proceso de despliegue con Terraform en AWS**
13. **Vídeo de la experimentación**  
<https://youtu.be/zRGhkBebEbA>

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

En esta memoria detallo el proceso completo de desarrollo y despliegue del sistema, los principales problemas que he encontrado durante la elaboración del proyecto, los cuales no fueron precisamente pocos, y además, me detendré en algunos conceptos específicos que considero especialmente relevantes y que aportan un conocimiento valioso para proyectos futuros. Para ello, incluiré capturas de pantalla que respaldarán visualmente las explicaciones y, como complemento final, proporcionaré un enlace a un video que documenta el proceso de experimentación, la demostración del despliegue automatizado y el funcionamiento del sistema en producción.

## 1. Resumen del sistema general que hay que desplegar

El objetivo principal de este proyecto es desplegar de forma automática un sistema MEAN multicapa funcional en la nube utilizando la herramienta de Infraestructura como Código (IaC) Terraform. Este sistema, diseñado para ser modular y escalable, se compone de un balanceador de carga, varias instancias que ejecutan el aplicativo, y una instancia adicional para la base de datos.

A este sistema se le conoce como stack MEAN, ya que está formado fundamentalmente por: MongoDB, Express, Angular y Nodejs.

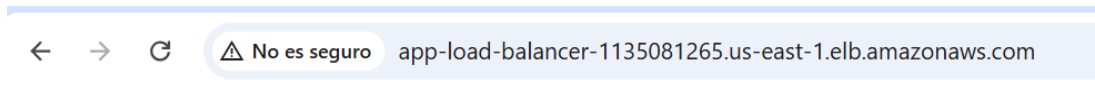
### A. Componentes software del sistema

A nivel de software, el sistema incluye los siguientes elementos principales:

- **Servidor web Nginx:** Gestiona el tráfico entrante, sirve el frontend y redirige las peticiones al backend.
- **Aplicación frontend en Angular:** Una aplicación ligera que gestiona la interfaz de usuario y realiza peticiones al backend.
- **Backend con Aplicación en ExpressJS y Node.js:** Proporciona la lógica del servidor y las conexiones necesarias para interactuar con la base de datos. Nodejs posibilita el entorno de ejecución de js en el lado del servidor para que trabaje el framework de Express.
- **Base de datos no relacional MongoDB:** Encargada de la persistencia de datos.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

La funcionalidad del aplicativo es sencilla, pero suficiente para demostrar el correcto funcionamiento de todo el sistema. Básicamente, es un *"Hola Mundo"* que no solo indica la instancia en la que se está ejecutando, sino que también verifica la correcta conexión entre las tres capas del sistema: frontend, backend y base de datos.



## Hola Mundo desde el Angular de la instancia 1

Hola desde el backend conectado correctamente a MongoDB!

### B. Capas físicas e infraestructura del sistema

A nivel de infraestructura, el sistema está compuesto por dos capas físicas distribuidas en distintas instancias:

**1. Primera capa - Primera instancia (frontend y backend):** Esta instancia combina múltiples servicios esenciales para el funcionamiento del sistema. Está aprovisionada con Nginx, Angular, ExpressJS, Node.js... Además de múltiples software auxiliares como:

- **PM2:** Gestor de procesos para Node.js, asegurando que el backend se mantenga activo y operativo.
- **NPM:** Gestor de paquetes para manejar las dependencias del proyecto.

**2. Segunda capa - Segunda instancia (persistencia de datos):** Dedicada exclusivamente a la base de datos, esta instancia está aprovisionada con **MongoDB**, una base de datos no relacional.

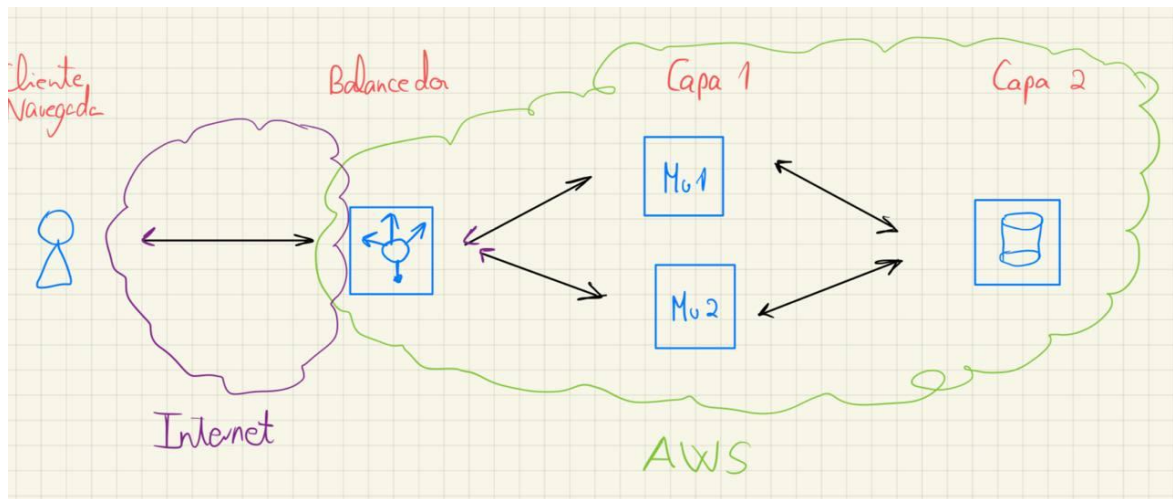
### Funcionalidad general

El sistema está diseñado para garantizar una integración fluida entre todas las capas. Cuando un cliente realiza una petición al sistema, esta pasa por el balanceador de carga y es distribuida a las instancias activas que dan servicio como la primera capa. La instancia de la primera capa que recibe el tráfico gestiona tanto el frontend como

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

el backend, y esta misma realiza solicitudes a la instancia de la segunda capa para acceder a la base de datos.

El despliegue automatizado con Terraform asegura que todos los componentes del sistema, tanto a nivel de infraestructura como de software, sean configurados correctamente, garantizando un entorno listo para producción.



## 2. Implementación de Packer en el proyecto

Para llevar a cabo este sistema, decidí implementar **Packer** como herramienta para crear de forma automática las imágenes base con las que se contruirían las instancias de la primera capa del sistema (capa del aplicativo), tal y como lo hice en el ejercicio 1. La elección de Packer se basó en varios motivos, pero si tuviera que destacar alguno, sería la posibilidad de aprovechar la plantilla que ya había desarrollado previamente. Esto no solo me permitió ahorrar tiempo, sino también poder basarme en la misma estructura del proyecto y darle mayor riqueza técnica al proyecto.

Además, como parte del diseño incluía desplegar un balanceador de carga y varias instancias idénticas en la primera capa para repartir la carga entre ellas durante la experimentación, me parecía lógico y eficiente contar con una imagen base ya aprovisionada y lista para ser utilizada. Esta imagen garantizaría que todas las instancias fueran idénticas, con excepción del contenido HTML el cual tuve que modificar para que se apreciara en la interfaz a que instancia se accede con el balanceador, sobre esto profundizaré más adelante.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

### 3. Proceso de creación y despliegue

El flujo de trabajo comienza con la integración de Terraform y Packer. Terraform fue configurado para llamar a Packer, el cual se encargó de crear la imagen base. Packer levanta una instancia en AWS que es aprovisionada con todos los componentes que portará la imagen, tras la creación de la imagen esta instancia se elimina de forma automática. Para este aprovisionamiento, utilicé **Ansible**, que automatizó la instalación y configuración de gran parte del sistema necesario en la instancia, **dejando únicamente una configuración mínima pendiente que se realizaría en fases posteriores con Terraform (hablaré más tarde sobre esto).**

#### Resumen de aprovisionamiento de instancia Packer mediante Ansible y jerarquía de ficheros

Como el aprovisionamiento es para la imagen base de las instancias de la primera capa del sistema, es decir, la instancia que contiene el aplicativo completo se instalan y configuran Nginx, Angular, Node.js, PM2... prácticamente todos los distintos programas utilizados en el proyecto dejando fuera MongoDB, que está en la segunda capa.

Para entender mejor cómo se realiza este aprovisionamiento, es necesario explicar la estructura de la carpeta **/provisioner** y su subcarpeta **/angular\_app**, ya que ambas contienen los ficheros esenciales que serán copiados a las instancias durante el proceso.

Dentro de la carpeta **Provisioner**, se encuentran los archivos que se utilizarán para configurar y aprovisionar la instancia. Estos ficheros son copiados desde la máquina local (donde se ejecutan Terraform y Packer) hacia la instancia que levanta packer en la nube, utilizando Packer como intermediario. Una vez copiados, Ansible los utiliza para realizar operaciones de configuración y aprovisionamiento.

Los ficheros principales son:

- 1. app.js.** Este archivo contiene el código fuente del backend desarrollado con Express.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

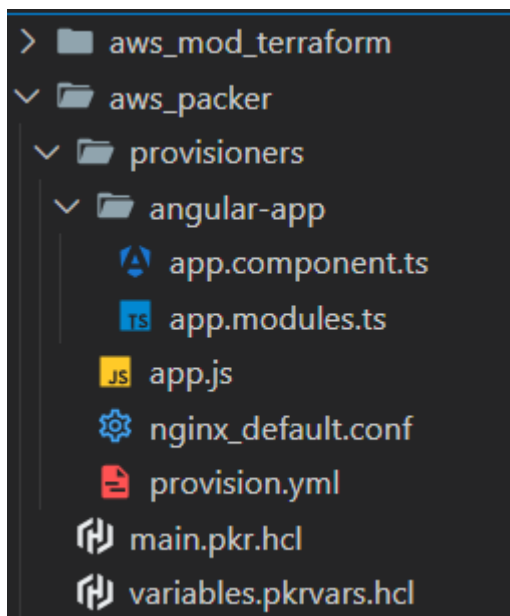
**2. nginx-default.conf.** Este es el archivo de configuración de Nginx. Define cómo se gestionará el tráfico HTTP, cómo se servirán los archivos estáticos generados por Angular y cómo se redirigirán las peticiones hacia el backend alojado en Express.

**3. provision.yml.** Este es el único archivo que **no se copia directamente a la instancia**, ya que es el archivo principal utilizado por Ansible para definir y ejecutar todas las tareas de aprovisionamiento. Contiene las instrucciones necesarias para instalar y configurar los servicios mencionados anteriormente.

Dentro de la subcarpeta **angular\_app** se encuentran los archivos relacionados con el frontend que serán copiados al proyecto de Angular:

**1. app.component.ts.** Este archivo contiene la lógica principal del componente principal del frontend. En este caso, se utiliza para personalizar el "Hola Mundo" que se muestra al cliente.

**2. app.module.ts.** Define los módulos y dependencias del proyecto Angular, siendo un archivo esencial para configurar y estructurar la aplicación frontend.



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

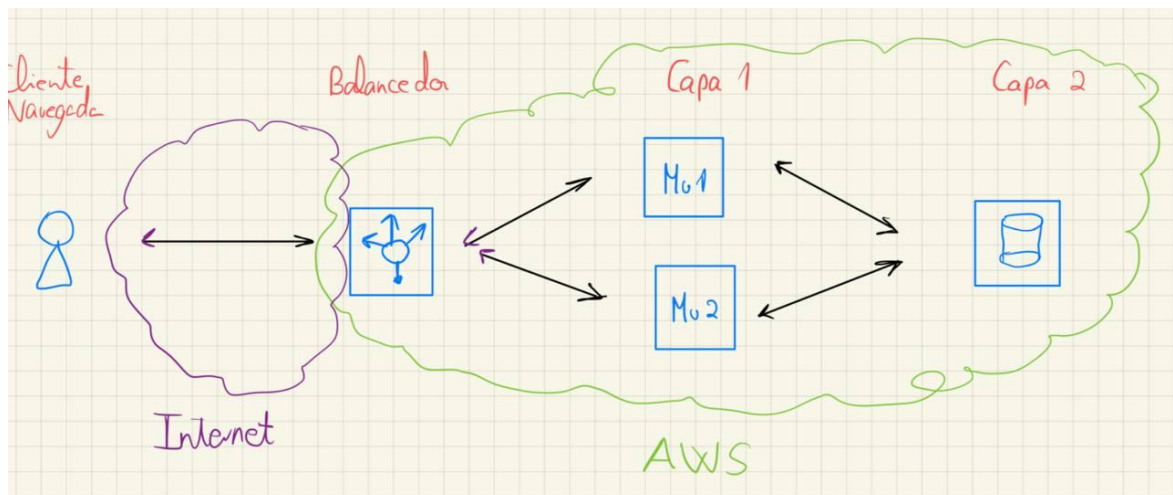
## Resumen del despliegue con Terraform

Una vez creada la imagen base con Packer, el siguiente paso es que Terraform se encarga de levantar todo el conjunto y ecosistema de infraestructura para levantar las instancias y poner en producción el sistema. Aparte de las redes, subredes, puertas de enlace... los principales elementos de infra levantados por Terraform son:

**1. Primera capa con dos instancias idénticas:** Basadas en la imagen creada, estas instancias contienen la misma configuración de software y se utilizarán para ejecutar tanto el frontend como el backend. **Para completar su configuración, Terraform se encarga mediante scripting de bash de ajustar el aprovisionamiento y hacer tareas que no se podían realizar en la creación de la imagen.**

**2. Segunda capa con una instancia para la BD:** Esta instancia, dedicada exclusivamente a MongoDB, es aprovisionada también con scripting de bash en terraform.

**3. Un balanceador de carga:** Un balanceador de carga para distribuir el tráfico de manera equitativa entre las dos instancias idénticas que manejan el frontend y el backend. Esto asegura que el sistema sea capaz de gestionar solicitudes de manera eficiente, incluso bajo cargas elevadas.



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## 4. Problema 1. Despliegue en Azure, incompatibilidad de versiones y porqué AWS.

Aunque mi solución final fue desplegar en AWS, mi primera decisión fue hacerlo en Azure. Esta elección no fue arbitraria, sino que respondió a varios factores prácticos. Uno de los principales fue que las cuentas de AWS para estudiantes tienen ciertas limitaciones en cuanto al tiempo de sesión y a la gestión de credenciales. En experiencias previas, me había encontrado con que estas restricciones y suponían una pérdida de tiempo significativa, algo que no podía permitirme para este proyecto. Por lo que aunque estoy más familiarizado con AWS y me resulta una plataforma más intuitiva, opté inicialmente por Azure.

Para implementar el proyecto en Azure, partí de la plantilla que había diseñado previamente utilizando Packer y Terraform en el primer proyecto. Aunque me ahorró tiempo, tuve que realizar muchas modificaciones para adaptarla al nuevo entorno. Sin embargo, aquí surgió el principal problema que definió mi decisión de abandonar Azure y migrar a AWS: **la incompatibilidad de versiones**.

Mi objetivo era crear con Packer una imagen lo más completa posible, lo que incluía instalar Angular CLI, además de otras dependencias necesarias como Node.js y Express. Para simplificar el proceso, mi estrategia consistía en enviar únicamente a Angular los ficheros esenciales, como el código fuente básico (en este caso, un "Hola Mundo") y el archivo de dependencias, a través de Packer hacia la instancia. Una vez allí, Ansible se encargaría de todo el resto de aprovisionamiento y configuración completa de la instancia para la imagen. (posteriormente Terraform aprovisiona un poco más)

### El problema con Angular CLI y Node.js

El mayor inconveniente surgió cuando intenté instalar Angular CLI en la instancia desplegada en Azure. Esta herramienta requería específicamente al menos la versión 18 de Node.js. Sin embargo, el sistema operativo de base que estaba disponible en la cuenta gratuita de Azure (Ubuntu 18.04) no era compatible con esta versión de Node.



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

Por tanto, estaba limitado a instalar Node.js en su versión 16, algo que ya había utilizado previamente con Packer, pero que era incompatible con la versión más reciente de Angular.

Probé usar una versión anterior de Angular CLI (la 14), que era compatible con Node 16. Sin embargo, al probar esta configuración, me encontré con otro problema inesperado: Angular CLI no funcionaba correctamente. Intenté generar proyectos básicos, pero la herramienta no descargaba todas las dependencias necesarias, y los proyectos resultantes eran defectuosos o directamente inservibles. Esto, evidentemente, imposibilitaba avanzar en el desarrollo.

```
User> az vm image list-skus --location "eastus" --publisher "Canonical" --offer "UbuntuServer" --output table
Location      Name
-----
eastus        12.04.5-LTS
eastus        14.04.0-LTS
eastus        14.04.2-LTS
eastus        14.04.3-LTS
eastus        14.04.4-LTS
eastus        14.04.5-LTS
eastus        16.04-DAILY-LTS
eastus        16.04-LTS
eastus        16.04.0-LTS
eastus        16.04-daily-lts-gen2
eastus        16.04-lts-gen2
eastus        16.04.0-lts-gen2
eastus        18.04-DAILY-LTS
eastus        18.04-LTS
eastus        18.10
eastus        18.04-daily-lts-arm64
eastus        18.04-daily-lts-gen2
eastus        18.04-lts-arm64
eastus        18.04-lts-gen2
eastus        19.04
eastus        19.10-DAILY
eastus        19.04-daily-gen2
eastus        19.04-gen2
eastus        19.10-daily-gen2
```

## Decisión de migrar a AWS

Tras dedicar un día completo a intentar resolver este problema sin éxito, decidí investigar las alternativas que AWS ofrecía en su nivel gratuito. Descubrí que AWS permitía el uso de una imagen base de Ubuntu 20.04, la cual era plenamente compatible con Node 18 y, por ende, con la versión más reciente de Angular CLI. Este cambio solucionó finalmente el problema.

- Compatibilidades AngularCLI y Nodejs

<https://stackoverflow.com/questions/60248452/is-there-a-compatibility-list-for-angular-angular-cli-and-node-js>

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## ► Recursos levantados en Azure

Microsoft Azure

Search resources, services, and docs (G+/I)

Home >

All resources

Directorio predeterminado (aleingmar@gmail.onmicrosoft.com)

+ Create Manage view Refresh Export to CSV Open query Assign tags Delete

Filter for any field... Subscription equals all Resource group equals all Type equals all Location equals all Add filter

0 Unsecure resources 0 Recommendations 11 Changed resources

Group by resource group List view

Name	Type	Resource group	Location	Subscription
networkwatcherrg				
packer-images				
pkri-resource-group-yipvo54jea				
pkriyipvo54jea	Public IP address	pkri-Resource-Group-yipvo54jea	East US	Azure for Students
pkrmijipvo54jea	Network Interface	pkri-Resource-Group-yipvo54jea	East US	Azure for Students
pkrozyipvo54jea	Disk	PKR-RESOURCE-GROUP-YIPVO...	East US	Azure for Students
pkrsyipvo54jea	Network security group	pkri-Resource-Group-yipvo54jea	East US	Azure for Students
pkrmvijipvo54jea	Virtual machine	pkri-Resource-Group-yipvo54jea	East US	Azure for Students
pkrmvijipvo54jea	Virtual network	pkri-Resource-Group-yipvo54jea	East US	Azure for Students

< Previous Page 1 of 1 Next > Showing 1 to 11 of 11 records.

Give feedback

## 5. Angular en producción y proceso de despliegue del frontend

Antes de este proyecto, nunca había trabajado directamente con Angular ni con ningún framework de frontend, por lo que ha sido una experiencia completamente nueva para mí. Hasta ahora, mi conocimiento en este ámbito se limitaba al uso de frameworks más sencillos como Hugo, que, aunque también genera contenido estático, tiene un enfoque y un funcionamiento distinto.

Uno de los aprendizajes más significativos en este proceso ha sido entender cómo funciona Angular en un entorno de producción. A nivel básico, desarrollar una aplicación en Angular implica programar los distintos módulos, componentes y funcionalidades del proyecto en el entorno de desarrollo. Sin embargo, cuando el objetivo es desplegar la aplicación en un servidor accesible desde dispositivos remotos —es decir, llevarla a producción— el flujo cambia sustancialmente. En este punto, Angular proporciona herramientas para compilar todo el código desarrollado y transformarlo en una serie de ficheros estáticos en formato HTML y JavaScript, los cuales son esenciales para su correcto funcionamiento en un servidor.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

```

+ Building...
module.instances.aws_instance.mongodb (remote-exec): Synchronizing state of mongodb.service with SysV service script with /lib/systemd/system
Initial chunk files | Names | Raw size | Estimated transfer size
module.instances.aws_instance.web_server[1] (remote-exec): main-NOFQ6WPG.js | main | 183.78 kB | 49.86 kB
module.instances.aws_instance.web_server[1] (remote-exec): polyfills-FFHMD2TL.js | polyfills | 34.52 kB | 11.28 kB
module.instances.aws_instance.web_server[1] (remote-exec): styles-5TNURTS0.css | styles | 0 bytes | 0 bytes

+ Building...

module.instances.aws_instance.web_server[1] (remote-exec): Application bundle generation complete. [14.218 seconds]

module.instances.aws_instance.web_server[1] (remote-exec): Output location: /home/ubuntu/angular-app/dist/angular-app
+ Building...
Initial chunk files | Names | Raw size | Estimated transfer size
module.instances.aws_instance.web_server[0] (remote-exec): main-E4XTM3PG.js | main | 183.78 kB | 49.86 kB
module.instances.aws_instance.web_server[0] (remote-exec): polyfills-FFHMD2TL.js | polyfills | 34.52 kB | 11.28 kB
module.instances.aws_instance.web_server[0] (remote-exec): styles-5TNURTS0.css | styles | 0 bytes | 0 bytes

module.instances.aws_instance.web_server[0] (remote-exec): | Initial total | 218.30 kB | 61.14 kB

```

En mi caso, esta dinámica se integró perfectamente con la metodología que había planteado para el proyecto. Como mencioné anteriormente, el proceso consistía en utilizar Packer y Ansible para copiar crear el proyecto de Angular y copiar a la instancia los ficheros base personalizados de Angular y que ya los tuviera en su interior la imagen base. **Pero el proyecto no es compilado ni los ficheros estáticos generados hasta el aprovisionamiento mediante scripting de Terraform.**

- Parte del proceso de Angular que se realiza en la creación de la imagen mediante Ansible

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

```
# Instalar Angular CLI globalmente
- name: Instalar la última versión de Angular CLI
  shell: |
    npm install -g @angular/cli@latest
  args:
    warn: false
    executable: /bin/bash

# Crear un nuevo proyecto Angular llamado angular-app
- name: Crear proyecto Angular
  shell: |
    ng new angular-app --defaults
  args:
    chdir: /home/ubuntu # Directorio donde se creará el proyecto
    executable: /bin/bash

# Reemplazar archivos en el proyecto Angular
- name: Reemplazar app.component.ts
  copy:
    src: /tmp/app.component.ts
    dest: /home/ubuntu/angular-app/src/app/app.component.ts
    owner: ubuntu
    group: ubuntu
    mode: '0644'

- name: Reemplazar app.module.ts
  copy:
    src: /tmp/app.modules.ts
    dest: /home/ubuntu/angular-app/src/app/app.module.ts
    owner: ubuntu
    group: ubuntu
    mode: '0644'
```

- Parte del proceso de Angular realizado en el aprovisionamiento con Terraform.

```
provisioner "remote-exec" {
  inline = [
    "sudo pm2 start /home/ubuntu/app.js",
    "BACKEND_URL=http://${self.public_ip}",
    "sudo sed -i 's|__BACKEND_URL__|\"$BACKEND_URL\"|g' /home/ubuntu/angular-app/src/app/app.component.ts",
    "sudo sed -i 's|__NUM_INST__|\"${count.index + 1}\"|g' /home/ubuntu/angular-app/src/app/app.component.ts",
    "cd /home/ubuntu/angular-app",
    "sudo npm install",
    "(sleep 7; echo 'n'; sleep 20; echo 'N') | sudo ng build --configuration=production",
    "sudo mkdir -p /var/www/angular-app/dist",
    "sudo cp -r dist/angular-app/* /var/www/angular-app/dist/",
    "sudo chown -R www-data:www-data /var/www/angular-app/dist",
    "sudo chmod -R 755 /var/www/angular-app/dist",
    "sudo systemctl restart nginx"
  ]
}
```

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

Estos ficheros estáticos representaban la versión final y optimizada de la aplicación, lista para ser servida en un entorno de producción.

Una vez generados estos ficheros, el siguiente paso fue trasladarlos al directorio de producción de Nginx, configurado previamente en la instancia. Esto fue crucial, ya que Nginx se encarga de recibir las peticiones HTTP dirigidas a la dirección IP del servidor y de servir los ficheros estáticos correspondientes como respuesta.

### ¿Por qué separar este proceso en dos partes y no hacerlo directamente todo en la creación de la imagen?

No hacerlo todo con Packer y Ansible tiene su explicación, como mencionaremos en el apartado de funcionamiento de las conexiones, la conexión directa entre el frontend y backend no existe. La conexión se realiza directa cliente-backend, es los ficheros del frontend que al procesarlo el navegador de cliente provoca que este busque un recurso en el backend. Por lo tanto para direccionar al navegador al backend hay que indicarle la ip de la instancia y ponerla en los ficheros del frontend y esa información se obtiene **tras** el despliegue de la instancia con Terraform, por lo tanto, para poder realizar esto necesitamos:

1. Terraform despliega instancia con aplicativo
2. Conocer su ip
3. Añadir la ip pública mediante el aprovisionamiento de la instancia con Terraform a los ficheros de Angular copiados anteriormente en la creación de la imagen
4. Generar estáticos
5. Mandarlos a que lo sirva NginX

(Pasa lo mismo con la información de que instancia es la que manda el frontend).

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## 6. Problema 2. Ejecución de comando interactivo que bloquea el proceso automático de aprovisionamiento.

Uno de los problemas más frustrantes y a la vez inesperados que encontré durante el desarrollo del proyecto fue la ejecución de un comando interactivo que bloqueaba el proceso automático de aprovisionamiento en Terraform. Este inconveniente, que a simple vista parecía algo trivial, terminó costándome muchas horas de trabajo y experimentación para resolverlo.

### Contexto del problema

Durante el aprovisionamiento automático, utilizando scripting en Bash dentro del proceso de Terraform, tenía que ejecutar un comando específico, **ng build**, que se encargaba de generar los ficheros estáticos del proyecto Angular. Este paso era fundamental, ya que los estáticos serían servidos posteriormente por Nginx en la capa de frontend. Sin embargo, al ejecutar el comando, surgió un problema crítico: **ng build** realizaba dos preguntas interactivas relacionadas con permisos y encuestas. En un entorno manual, estas preguntas pueden responderse fácilmente, pero en un flujo automatizado como el mío, donde no hay intervención humana, el proceso se quedaba bloqueado indefinidamente al esperar una respuesta. Esto interrumpía por completo el aprovisionamiento automático, haciendo que la instancia nunca completara su configuración.

### Intentos de solución inicial

Intenté numerosas estrategias para resolver este problema, pero ninguna funcionaba. Estas fueron algunas de las soluciones que probé sin éxito: Uso de variables de entorno, modificación del archivo de configuración de Angular CLI, opciones del comando...

Hasta que probé el comando **yes**, que me hizo que se me encendiera la bombilla: implementé el comando **yes**, que envía respuestas automáticas, como un flujo continuo de "no". Sin embargo, esto no funcionó porque **yes** enviaba respuestas demasiado rápido, lo que causaba que el proceso se quedara bloqueado y entrara en

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

un bucle continuo. Por lo que pensé: *¿y si consiguiera mandarlo con el timing perfecto...?*

Probé a enviar las respuestas mediante **echo**, intentando sincronizar el flujo con el comando **ng build**. Pero sin un control adecuado de los tiempos, el proceso seguía fallando, ya que las respuestas no coincidían con el momento exacto en que las preguntas eran generadas.

### La solución: uso de sleep y control de tiempos

Finalmente, después de muchas pruebas y errores, encontré una solución efectiva utilizando una **combinación de echo y sleep** para sincronizar las respuestas con las preguntas del comando. Este fue el enfoque que utilicé:

Configuré el comando para que tanto el comando **ng build** como las respuestas generadas por **echo** y **sleep** se ejecutaran simultáneamente, comenzando en el mismo instante (segundo 0).

Tras ejecutar **ng build**, el comando generaba la primera pregunta. Añadí un **sleep** de 7 segundos para esperar a que la pregunta apareciera completamente antes de enviar la primera respuesta con **echo "no"**.

Luego, configuré un segundo **sleep** para esperar 13 segundos adicionales (20 segundos en total desde el inicio), lo que daba tiempo al comando para generar la segunda pregunta. Una vez lista, envié la segunda respuesta con otro **echo "no"**.

Con este enfoque, las respuestas se enviaron después de que las preguntas eran generadas, permitiendo que el comando **ng build** completara su ejecución sin interrupciones. Esto resolvió el problema y permitió que el proceso de aprovisionamiento automático continuara sin errores.

```
sudo npm install ,
"(sleep 7; echo 'n'; sleep 20; echo 'N') | sudo ng build --configuration=production",
"..."
```

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

```

module.instances.aws_instance.web_server[0] (remote-exec): Would you like to enable
module.instances.aws_instance.web_server[0] (remote-exec): autocompletion? This will set up your
module.instances.aws_instance.web_server[0] (remote-exec): terminal so pressing TAB while typing
module.instances.aws_instance.web_server[0] (remote-exec): Angular CLI commands will show possible
module.instances.aws_instance.web_server[0] (remote-exec): options and autocomplete arguments.
module.instances.aws_instance.web_server[1] (remote-exec): Would you like to enable
module.instances.aws_instance.web_server[1] (remote-exec): autocompletion? This will set up your
module.instances.aws_instance.web_server[1] (remote-exec): terminal so pressing TAB while typing
module.instances.aws_instance.web_server[1] (remote-exec): Would you like to enable
module.instances.aws_instance.web_server[1] (remote-exec): autocompletion? This will set up your
module.instances.aws_instance.web_server[1] (remote-exec): terminal so pressing TAB while typing
module.instances.aws_instance.web_server[1] (remote-exec): Would you like to enable
module.instances.aws_instance.web_server[1] (remote-exec): autocompletion? This will set up your
module.instances.aws_instance.web_server[1] (remote-exec): terminal so pressing TAB while typing
module.instances.aws_instance.web_server[1] (remote-exec): Angular CLI commands will show possible
module.instances.aws_instance.web_server[1] (remote-exec): options and autocomplete arguments.
module.instances.aws_instance.web_server[1] (remote-exec): (Enabling autocompletion will modify
module.instances.aws_instance.web_server[1] (remote-exec): configuration files in your home
module.instances.aws_instance.web_server[1] (remote-exec): directory.) no
module.instances.aws_instance.web_server[1] (remote-exec): Ok, you won't be prompted again. Should you change your min
ng completion
module.instances.aws_instance.web_server[1] (remote-exec):
module.instances.aws_instance.web_server[1] (remote-exec): Would you like to share pseudonymous
module.instances.aws_instance.web_server[1] (remote-exec): usage data about this project with the
module.instances.aws_instance.web_server[1] (remote-exec): Angular Team
module.instances.aws_instance.web_server[1] (remote-exec): at Google under Google's Privacy Policy
module.instances.aws_instance.web_server[1] (remote-exec): at https://policies.google.com/privacy.
module.instances.aws_instance.web_server[1] (remote-exec): Would you like to enable
module.instances.aws_instance.web_server[0] (remote-exec): autocompletion? This will set up your
module.instances.aws_instance.web_server[0] (remote-exec): terminal so pressing TAB while typing
module.instances.aws_instance.web_server[0] (remote-exec): Angular CLI commands will show possible
module.instances.aws_instance.web_server[0] (remote-exec): options and autocomplete arguments.
module.instances.aws_instance.web_server[0] (remote-exec): (Enabling autocompletion will modify
module.instances.aws_instance.web_server[0] (remote-exec): configuration files in your home
module.instances.aws_instance.web_server[0] (remote-exec): directory.) no
module.instances.aws_instance.web_server[0] (remote-exec): Ok, you won't be prompted again. Should you change your min
ng completion

```

## 7. Funcionamiento de la conexión entre frontend, backend y base de datos

Una de las partes más relevantes y complejas del proyecto fue diseñar e implementar la conexión entre el frontend (Angular), el backend (ExpressJS) y la base de datos (MongoDB), asegurando que todos los componentes interactuaran correctamente y sin conflictos en el entorno de producción.

### Flujo de comunicación entre cliente-frontend y cliente-backend

El flujo comienza cuando el cliente realiza una petición HTTP a la instancia que aloja el proyecto. Esta petición llega al puerto 80, gestionado por Nginx, que sirve los archivos estáticos generados previamente por Angular. Es importante destacar que Angular **no** es un proceso activo en el servidor, ya que su única función durante el aprovisionamiento fue generar los ficheros estáticos. Estos archivos, al llegar al



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

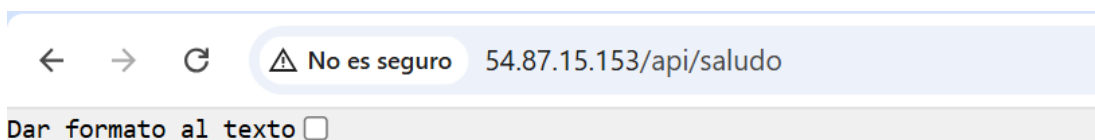
navegador del cliente, contienen el código JavaScript necesario para ejecutar el frontend y realizar llamadas al backend.

**La conexión directa entre frontend y backend no existe**, si no que el navegador del cliente está por medio. La conexión entre frontend y backend se basa en que el frontend le dice al navegador del cliente que llame al backend él y le da su dirección.

**¿Cómo es eso posible?.** Estos ficheros estáticos contienen llamadas a la ruta **/api/saludo**, lo que provoca que el navegador del cliente mande la petición correspondiente a la instancia para obtener el recurso. Nginx escucha otra vez esta petición redirige la petición al puerto 3000 de su instancia, donde se encuentra escuchando el proceso de ExpressJS. Este backend, a su vez, procesa la lógica de negocio y realiza una conexión con la base de datos para completar la solicitud.

```
// Realizar la solicitud al backend para obtener el mensaje
fetch(`${this.backendUrl}/api/saludo`)
  .then(response => {
    if (!response.ok) {
      throw new Error('Error en la solicitud');
    }
  })
```

```
# Redirige solicitudes a /api/ al backend Express
location /api/ {
  proxy_pass http://localhost:3000; # Cambia el puerto si tu backend usa otro
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection 'upgrade';
  proxy_set_header Host $host;
  proxy_cache_bypass $http_upgrade;
}
```



```
{"mensaje": "Hola desde el backend conectado correctamente a MongoDB!"}
```

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## Conexión entre backend y base de datos

La conexión entre ExpressJS y MongoDB presenta una complejidad adicional debido a que ambas instancias están separadas físicamente y por lo tanto, no podemos seguir la misma estrategia que antes para que el backend conozca la ip de la instancia que tiene MongoDB.

### Creación de una red privada común

Para resolver esto y garantizar la comunicación entre el backend y la base de datos, se configuró una red privada común utilizando Terraform. Así podríamos otorgar una ip fija privada a la instancia de MongoDB y dejarla permanente en el backend. Esta red privada permite que las instancias se comuniquen de manera segura dentro de un mismo rango de direcciones IP privadas. Cada instancia se configura con **una dirección IP privada fija**, lo que asegura que las direcciones no cambien incluso si se reinician las instancias o se despliega nuevamente la infraestructura.

En este diseño:

- La instancia que aloja MongoDB no tiene asignada una dirección IP pública, ya que no es necesario que sea accesible desde fuera de la red privada.
- Las instancias que ejecutan el backend tienen tanto una dirección IP pública (para recibir tráfico desde el cliente) como una dirección IP privada (para comunicarse con MongoDB).

```
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose'); // cliente de MongoDB

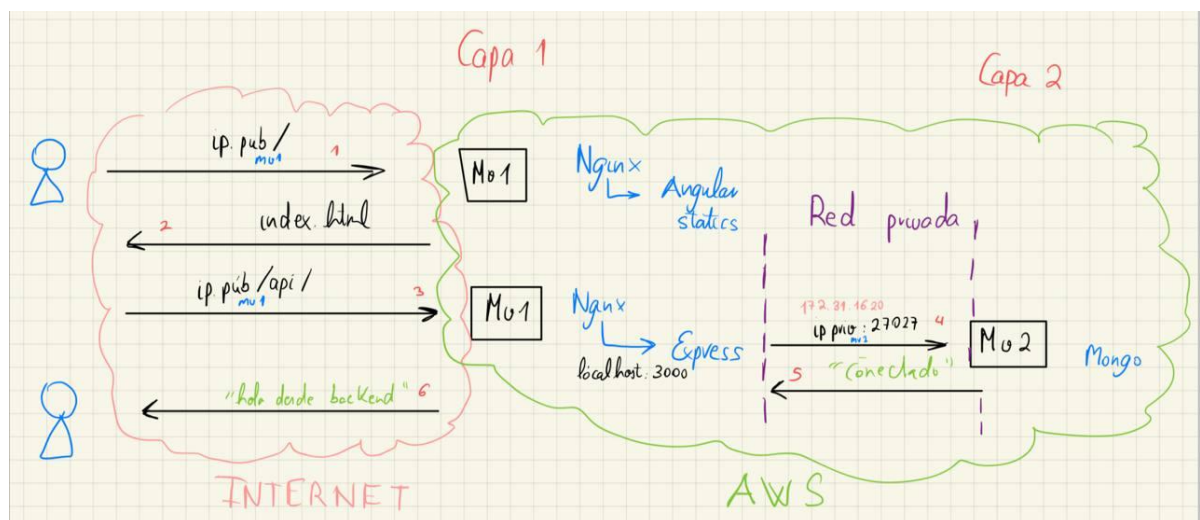
const app = express(); // Asegúrate de que la variable `app`
const PORT = 3000;

// URL de MongoDB
const MONGO_URL = "mongodb://172.31.16.20:27017";
```

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## Configuración de la conexión en ExpressJS

Con la dirección IP privada fija de la instancia de MongoDB conocida de antemano, se configuró directamente en el archivo app.js del backend. Esta dirección se establece junto con el puerto predeterminado de MongoDB (27017) para garantizar la conexión sin necesidad de tráfico externo. Dado que ambas instancias están dentro de la misma subred, la comunicación se realiza exclusivamente a nivel interno, evitando salir al tráfico público y reduciendo así posibles vulnerabilidades.



## Configuración BD y verificación de la conexión backend-bd

En este proyecto, no era necesario realizar una configuración avanzada de la base de datos ni crear colecciones o documentos. El objetivo principal era demostrar la conexión entre el backend y la base de datos. Por ello, la lógica del backend incluye una respuesta simple para verificar si la conexión fue exitosa o si hubo algún problema.

Si la conexión es exitosa, el backend responde al cliente con un mensaje como: *"Hola desde el backend, me he podido conectar correctamente a la base de datos"*. Si la conexión falla por cualquier motivo (como un timeout o una base de datos inactiva), el mensaje devuelto indicará que el backend no pudo establecer la conexión.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## 8. Criterio 1: Template de terraform con modularización y grupos de seguridad

### A. Modularización en Terraform

El uso de módulos en Terraform es una de las características clave que mejora la gestión del código y la reutilización de recursos. Aunque al principio puede parecer complejo, especialmente por la gestión de variables entre los módulos, este enfoque ofrece una gran flexibilidad y escalabilidad.

#### ¿Qué es la modularización?

En esencia, modularizar en Terraform significa dividir el fichero principal **main.tf** en múltiples partes o "categorías", organizadas en subcarpetas. Esto permite separar la lógica del proyecto según su contexto, como redes, seguridad, instancias, balanceadores de carga, etc. De este modo, en lugar de tener un único **main.tf** extenso y difícil de manejar, cada módulo representa una pieza específica de la infraestructura.

#### Estructura de un módulo

Cada módulo tiene su propia subcarpeta con los siguientes ficheros principales:

- **main.tf**: Contiene la lógica específica del módulo, como la configuración de redes o la definición de instancias.
- **variables.tf**: Define/declara las variables necesarias para el módulo, como direcciones IP, tamaños de instancias, nombres de recursos, etc.
- **outputs.tf**: Define las salidas de información que el módulo exportará para ser utilizadas por otros módulos o por el fichero principal.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## Ficheros principales y gestión de variables

### Main.tf

El fichero **main.tf** principal, ubicado en la raíz de la carpeta de Terraform, es el encargado de orquestar y llamar a todos los módulos. Este fichero:

Llama a los módulos mediante la directiva **module**, especificando la ruta de cada uno.

Asigna los valores a las variables utilizadas por los módulos.

### Outputs.tf

Por otro lado, el fichero **outputs.tf** principal contiene las salidas finales de Terraform, como direcciones IP, URLs o identificadores de recursos, que se generan al final del despliegue.

## Gestión de variables y salidas

Uno de los aspectos más desafiantes de la modularización es la gestión de variables y salidas entre los módulos y el fichero principal. Aquí, Terraform utiliza un enfoque estructurado:

- 1. Variables en los módulos:** Cada módulo define sus propias variables en su **variables.tf**, pero no asigna valores a estas. Los valores se proporcionan desde el fichero principal **main.tf**.
- 2. Outputs en los módulos:** Los módulos exportan información clave a través de **outputs.tf**, como direcciones IP o identificadores de recursos. Estas salidas pueden ser utilizadas por otros módulos o por el **main.tf** principal.
- 3. Variables globales:** En el contexto principal de Terraform, existe un fichero **variables.tf** global, donde se declaran las variables compartidas entre los módulos, como el rango de IPs de una red o los nombres comunes de los recursos.
- 4. Asignación de valores:** Los valores de las variables globales y de los módulos se asignan en el **main.tf** principal, asegurando que toda la infraestructura esté correctamente parametrizada. También se puede utilizar un fichero **terraform.tfvars** para ello

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## Ventajas de la modularización

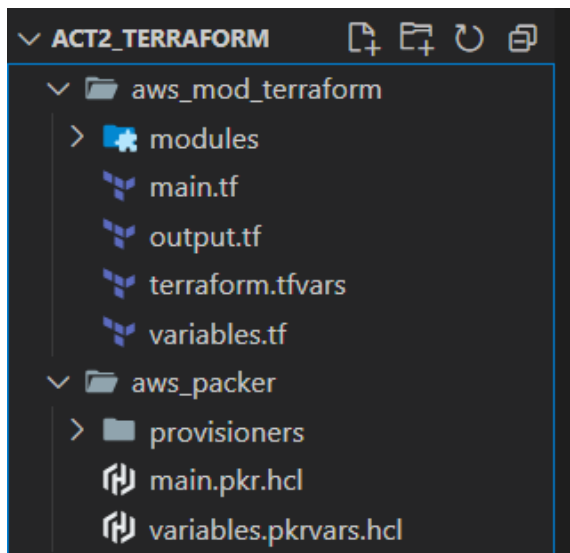
1. **Escalabilidad:** Al dividir el código en módulos, es más fácil añadir nuevas funcionalidades sin afectar al resto del proyecto.
2. **Reutilización:** Los módulos pueden ser reutilizados en otros proyectos con configuraciones similares.
3. **Mantenimiento:** Al estar organizado por categorías, resulta más sencillo localizar y modificar partes específicas del código.
4. **Colaboración:** Permite que diferentes miembros de un equipo trabajen en módulos independientes sin interferir entre sí.

## Mi forma de modularizar la plantilla

En mi implementación de Terraform, he decidido dividir la lógica en cinco módulos bien diferenciados.

### 1. Módulo de Imagen

Este módulo se encarga de la integración con Packer para crear y recuperar la imagen base que será utilizada por las instancias. En este módulo se define la lógica de Terraform para invocar a Packer.



### 2. Módulo de Instancias

En este módulo se encuentra toda la lógica para el despliegue de las instancias del sistema. Entre otras cosas por ejemplo, el aprovisionamiento por parte de Terraform

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

de las instancias. Esto incluye tanto las instancias de la **primera capa** (frontend y backend) como la instancia dedicada a la **segunda capa** (persistencia de datos con MongoDB). Además, aquí se definen y gestionan las direcciones IP:

- **IPs elásticas:** Se asignan a las instancias de la primera capa para permitir el acceso público a través de Internet. (ips públicas)
- **IPs privadas:** Se utilizan para la comunicación interna entre las instancias, asegurando que el tráfico interno sea seguro y no dependa de la red pública.

### 3. Módulo de Balanceador de Carga

Este módulo gestiona la configuración y despliegue del balanceador, asegurándose de que esté correctamente conectado a las instancias y que distribuya el tráfico de manera equilibrada.

### 4. Módulo de Redes

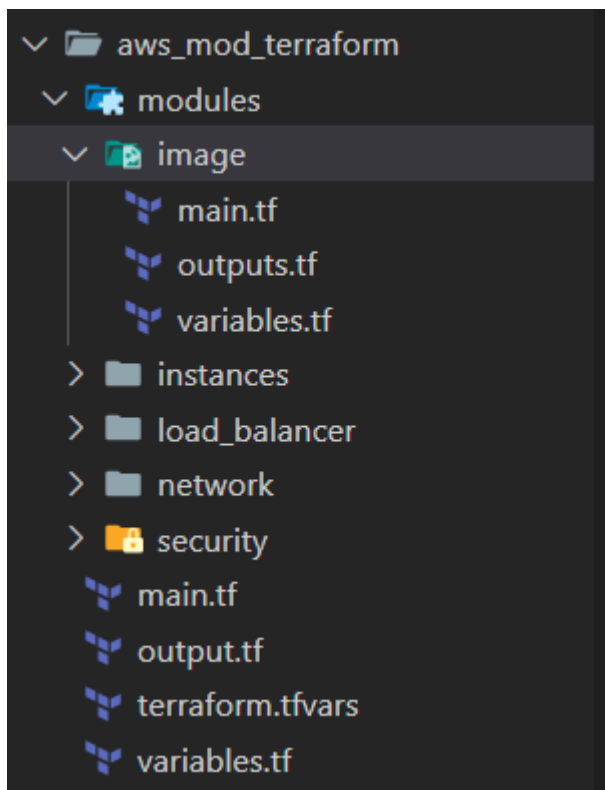
El módulo de redes se encarga de crear y gestionar toda la infraestructura de red del proyecto. Esto incluye:

- **Red principal:** Una red privada que conecta todas las instancias y otros recursos.
- **Subredes:** Se crean subredes específicas para los diferentes elementos del sistema.
- **Rutas:** Se configuran las rutas necesarias para garantizar la conectividad entre las subredes y, en algunos casos, hacia el exterior.

### 5. Módulo de Seguridad

Este módulo gestiona las claves de acceso SSH y los **grupos de seguridad**, que definen las reglas de tráfico permitidas hacia y desde las instancias. Se han creado dos grupos de seguridad principales:

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	



## B. Grupos de seguridad definidos

### 1. Grupo de Seguridad de la Primera Capa

Este grupo aplica a las instancias de la primera capa, permitiendo:

- **Conexión SSH (puerto 22):** Para acceder de manera remota y gestionar las instancias.
- **Conexión HTTP (puerto 80):** Para servir el frontend a los usuarios.
- **Protocolo ICMP (ping):** Utilizado inicialmente para verificar la conectividad entre las instancias.
- **Conexión HTTPS (puerto 443):** Aunque no se utiliza en este proyecto, se deja abierto para posibles implementaciones futuras.

### 2. Grupo de Seguridad de la Segunda Capa

Este grupo aplica a la instancia de la base de datos y tiene reglas más restrictivas, permitiendo:

- **Conexión SSH (puerto 22):** Para gestión remota.
- **Protocolo ICMP (ping):** Para pruebas de conectividad interna.



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	


- **Conexión al puerto 27017:** Puerto predeterminado para MongoDB, utilizado para las solicitudes del backend.

En este grupo no se permite tráfico HTTP o HTTPS, ya que la base de datos no está configurada para recibir solicitudes externas y solo se accede a través de la red interna.

## Configuración de Tráfico

Ambos grupos de seguridad tienen configuraciones similares para el **ingreso** y el **egreso** de tráfico:

- **Ingreso (ingress):** Se define específicamente qué tipos de tráfico pueden entrar en cada instancia, limitando el acceso según los puertos y protocolos mencionados.
- **Egreso (egress):** Se permite cualquier tipo de tráfico de salida desde las instancias, lo que facilita la comunicación hacia otros recursos o servicios externos.

Security Groups (4) [Info](#)  [Actions](#) [Export security groups to CSV](#) [Create security group](#)

<input type="checkbox"/>	Name	Security group ID	Security group name	VPC ID
<input type="checkbox"/>	-	<a href="#">sg-085c668a4e660219b</a>	mongodb-sg	<a href="#">vpc-0443ccf3a68f59ef2</a> <a href="#">[</a>
<input type="checkbox"/>	-	<a href="#">sg-0807c85cc1a0f6ea4</a>	default	<a href="#">vpc-0443ccf3a68f59ef2</a> <a href="#">[</a>
<input type="checkbox"/>	-	<a href="#">sg-02def6ced495ee534</a>	Instance_stack_MEAN-sg	<a href="#">vpc-0443ccf3a68f59ef2</a> <a href="#">[</a>
<input type="checkbox"/>	-	<a href="#">sg-048fc29d347159a2f</a>	default	<a href="#">vpc-00020ebe526c7d69c</a>

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## 9. Criterio 2: Inclusión de balanceador de carga

- Capturas del balanceador de carga, el grupo objetivo al que manda el tráfico (dos instancias funcionales) y la configuración del balanceo de carga. Round robin, stickiness...

### Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

<input checked="" type="checkbox"/>	Name	DNS name	State	VPC ID	Availability Zones
<input checked="" type="checkbox"/>	<a href="#">app-load-balancer</a>	<a href="#">app-load-balancer-203890...</a>	Active	vpc-00020ebe526c7d69c	2 Availability Zones

### app-load-balancer-target-group

**Details**
  
[arn:aws:elasticloadbalancing:us-east-1:590184066359:targetgroup/app-load-balancer-target-group/521aa572fcd40dff](#)

<b>Target type</b> Instance	<b>Protocol : Port</b> HTTP: 80	<b>Protocol version</b> HTTP1	<b>VPC</b> <a href="#">vpc-0443ccf3a68f59ef2</a>
<b>IP address type</b> IPv4	<b>Load balancer</b> <a href="#">app-load-balancer</a>		

2	2	0	0	0	0
Total targets	Healthy	Unhealthy	Unused	Initial	Draining
	0 Anomalous				

#### Traffic configuration

**Load balancing algorithm**  
Round robin

**Slow start duration**  
0 seconds

#### Target selection configuration

**Stickiness**  
On

**Stickiness type**  
Load balancer generated cookie

**Stickiness duration**  
5 seconds

**Cross-zone load balancing**  
Inherit settings from load balancer attributes

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## 10. Problema 3. Inconsistencia de recursos estáticos y cierre de sesión del balanceador

Uno de los problemas más inesperados y complicados que enfrenté durante el desarrollo del proyecto fue relacionado con el balanceador de carga. Este problema, que al principio parecía un fallo intermitente sin un patrón claro, me llevó mucho tiempo para entenderlo y, finalmente, resolverlo.

### Contexto del problema

Cuando desplegué el balanceador de carga utilizando Terraform, inicialmente no pensé que me daría tantos problemas, ya que en proyectos anteriores había utilizado balanceadores sin mayores inconvenientes. Sin embargo, en este caso había una diferencia clave: la aplicación que estaba implementando era más compleja. Mientras que en proyectos anteriores solo servía un simple **index.html**, ahora estaba trabajando con una aplicación completa en Angular, que incluía múltiples rutas a recursos adicionales como archivos JavaScript y CSS.

El comportamiento del sistema era completamente inconsistente. Al acceder directamente a las instancias mediante sus IPs públicas, todo funcionaba perfectamente. Sin embargo, cuando utilizaba el balanceador de carga y accedía al sistema mediante su DNS, los resultados eran impredecibles:

- A veces, el balanceador servía correctamente el contenido de la **instancia 1**.
- En otras ocasiones, mostraba el contenido de la **instancia 2**.
- A menudo, me encontraba con una pantalla en blanco.
- En algunos casos, el navegador arrojaba errores extraños en la consola, relacionados con tipos de contenido.

Esta falta de un patrón claro complicaba enormemente la búsqueda del problema. Parecía que el balanceador distribuía la carga correctamente, ya que ambas instancias estaban "healthy" y, en ciertos momentos, devolvía respuestas de ambas. Sin embargo, algo estaba fallando en el proceso, y no lograba identificar qué era.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## Investigación y análisis

Para encontrar la causa del problema, comencé revisando todos los aspectos posibles:

1. **Estado del balanceador de carga:** Verifiqué en la interfaz de AWS que el balanceador reconociera correctamente ambas instancias como saludables y en la misma región. Esto descartó problemas básicos de configuración.
2. **Caché del navegador:** Desactivé el caché en las herramientas de desarrollo del navegador, para asegurarme de que no se tratara de un problema de almacenamiento en el cliente.
3. **Errores en la consola del navegador:** Empecé a notar un patrón en los errores que aparecían. Cuando la pantalla quedaba en blanco, el navegador reportaba un problema relacionado con el tipo de contenido de uno de los archivos estáticos, en particular un archivo llamado **main-hash.js**, que es generado por Angular al compilar el proyecto.



4. **Pruebas con cURL:** Realicé peticiones directas al balanceador utilizando **cURL** para analizar las respuestas. Descubrí que, cuando se solicitaba el recurso concreto main-hash.js equivocado a la instancia que no era, este devolvía un contenido con la cabecera de tipo de contenido equivocado.
  - ▶ En el primer caso el tipo es equivocado (el balanceador pide el main de la otra instancia a la instancia que no es)
  - ▶ En la segunda petición el balanceador le pide el main a la instancia correcta, me devuelve el tipo correcto

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

```
ubuntu@ip-172-31-16-10:~$ curl -I http://app-load-balancer-652797655.us-east-1.elb.amazonaws.com/main-L6D6YZQN.js
HTTP/1.1 200 OK
Date: Tue, 14 Jan 2025 13:32:36 GMT
Content-Type: text/html
Content-Length: 483
Connection: keep-alive
Server: nginx/1.18.0 (Ubuntu)
Last-Modified: Tue, 14 Jan 2025 13:01:26 GMT
ETag: "67866026-1e3"
Accept-Ranges: bytes

ubuntu@ip-172-31-16-10:~$ curl -I http://app-load-balancer-652797655.us-east-1.elb.amazonaws.com/main-L6D6YZQN.js
HTTP/1.1 200 OK
Date: Tue, 14 Jan 2025 13:32:37 GMT
Content-Type: application/javascript
Content-Length: 183780
Connection: keep-alive
Server: nginx/1.18.0 (Ubuntu)
Last-Modified: Tue, 14 Jan 2025 13:01:35 GMT
ETag: "6786602f-2cde4"
Accept-Ranges: bytes
```

- Cuando no entiende el main (no se llama así el main de esa instancia) te da un tipo equivocado → Pantalla en blanco

```
ubuntu@ip-172-31-16-10:~$ curl -I http://52.73.123.71/main-L6D6YZQN.js
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 14 Jan 2025 13:32:16 GMT
Content-Type: text/html
Content-Length: 483
Last-Modified: Tue, 14 Jan 2025 13:01:26 GMT
Connection: keep-alive
ETag: "67866026-1e3"
Accept-Ranges: bytes
```

- Este es el correcto, cuando entiende este main ese es el tipo que te debe de dar

```
ubuntu@ip-172-31-16-10:~$ curl -I http://52.73.123.71/main-VXPILI52.js
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 14 Jan 2025 13:36:15 GMT
Content-Type: application/javascript
Content-Length: 183778
Last-Modified: Tue, 14 Jan 2025 13:01:26 GMT
Connection: keep-alive
ETag: "67866026-2cde2"
Accept-Ranges: bytes
```

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## Descubrimiento del problema

El problema estaba en cómo Angular generaba los recursos estáticos. Cuando se compila un proyecto en Angular, se generan archivos como **main-hash.js** que incluyen un hash único en su nombre. Esto significa que el archivo **main-hash.js** de la **instancia 1** tenía un hash diferente al de la **instancia 2**.

El balanceador de carga estaba configurado con un algoritmo de distribución **round-robin** sin persistencia de sesión. Así, cuando el cliente solicitaba el archivo **main-hash.js**:

1. Si el balanceador redirigía la petición a la misma instancia que había generado el `index.html` inicial, todo funcionaba correctamente.
2. Si el balanceador redirigía la petición a la otra instancia, esta devolvía una respuesta diferente. Esto generaba un error de tipo de contenido en el navegador, y la pantalla quedaba en blanco.

## Solución

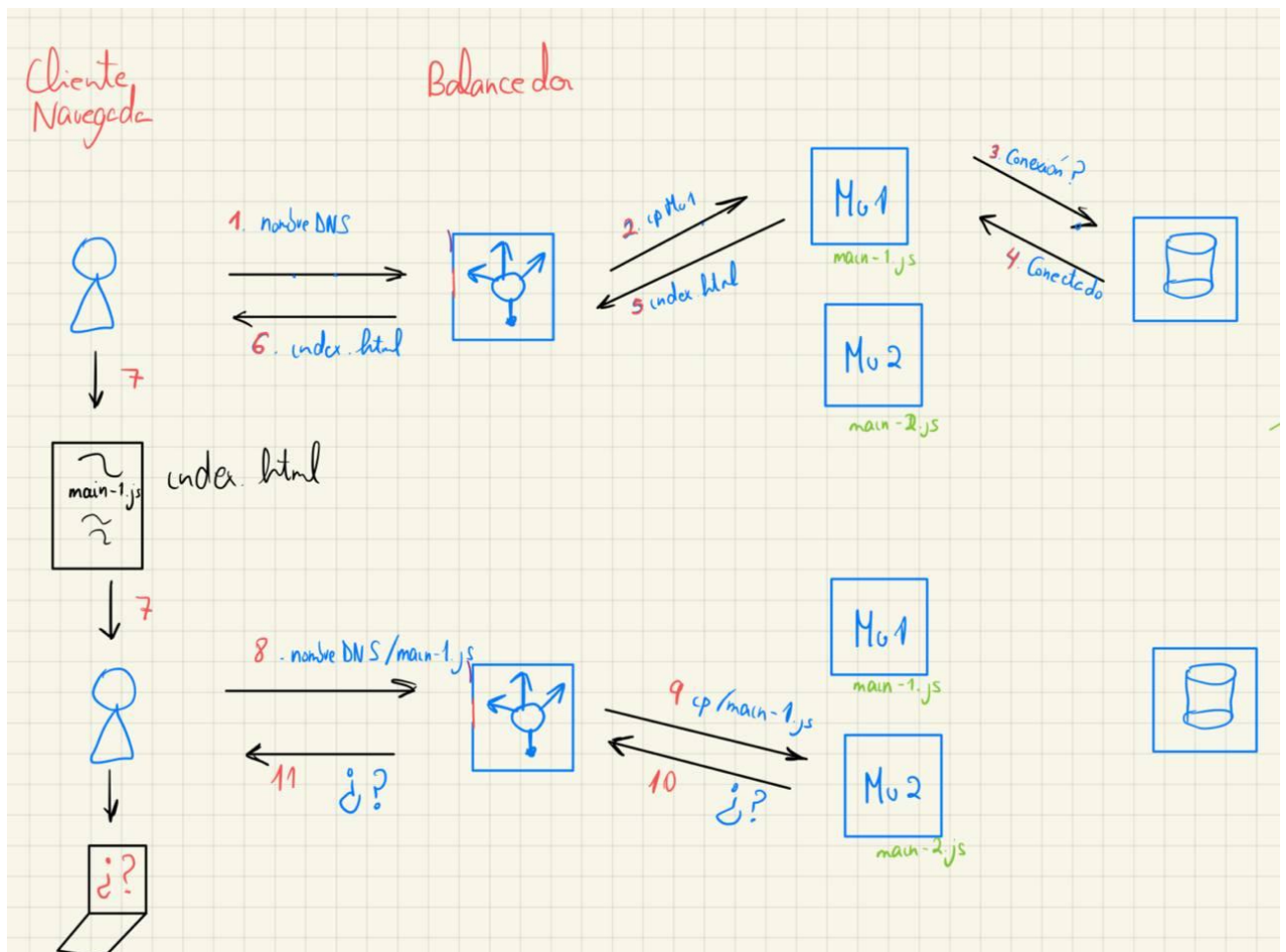
La solución fue más sencilla de lo que esperaba, pero solo después de entender completamente el problema. Configuré el balanceador de carga para que utilizara **sticky sessions** con un tiempo de persistencia de 5 segundos.

Con esta configuración, el balanceador recuerda a qué instancia asignó la primera petición de un cliente y redirige todas las peticiones subsiguientes del mismo cliente a esa instancia durante el período de persistencia. De esta forma:

1. El cliente solicita el índice a través del balanceador de carga.
2. El balanceador redirige la petición a una instancia (por ejemplo, la **instancia 1**).
3. Todas las peticiones subsecuentes del cliente, como las solicitudes para **main-hash.js**, se redirigen también a la **instancia 1**.
4. Esto evita inconsistencias en los recursos estáticos y asegura que todo funcione correctamente.

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

```
# Configuración de Sticky Sessions
# Con esto conseguimos que el balanceador recuerde que instancia mando el in
# Si no me daba un gran problema de inconsistencia en la carga de los recurs
# 5s debería ser mas q de sobra
stickiness {
  type           = "lb_cookie" # Usamos cookies gestionadas por el ALB
  cookie_duration = 5           # Duración de 10 segundos
}
```



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## 11. Criterio 3: Outputs

- IP públicas de cada nodo.
- IP privadas de cada nodo.
- DNS del balanceador.
- IP publica objeto nat gateway para instance mongodb.

```

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.

Outputs:

load_balancer_dns = "app-load-balancer-1787038451.us-east-1.elb.amazonaws.com"
mongodb_nat_gateway_public_ip = "18.214.151.191"
web_server_private_ips = [
    "172.31.16.10",
    "172.31.16.11",
]
web_server_public_ips = [
    "98.85.155.40",
    "34.194.88.84",
]
web_server_ssh_commands = [
    "ssh -i id_rsa ubuntu@98.85.155.40",
    "ssh -i id_rsa ubuntu@34.194.88.84",
]
aws_terraform>

```



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

## 12. Capturas del proceso de despliegue con Terraform en AWS

- Creación del proyecto de terraform y descarga de dependencias

```
aws_terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Finding latest version of hashicorp/null...
- Finding latest version of hashicorp/tls...
- Installing hashicorp/aws v5.83.1...
- Installed hashicorp/aws v5.83.1 (signed by HashiCorp)
- Installing hashicorp/null v3.2.3...
- Installed hashicorp/null v3.2.3 (signed by HashiCorp)
- Installing hashicorp/tls v4.0.6...
- Installed hashicorp/tls v4.0.6 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
aws_terraform>
```

- Validación de plantilla de terraform

```
aws_terraform> terraform validate
Success! The configuration is valid.
```

- Aplicación de configuración definida en la plantilla de terraform

```
aws_terraform> terraform apply -var "aws_access_key=$env:PKR_VAR_aws_access_key" -var "aws_secret_key=$env:PKR_VAR_aws_secret_key" -var "aws_session_token=$env:PKR_VAR_aws_session_token"
data.aws_vpc.default: Reading...
data.aws_vpc.default: Read complete after 2s [id=vpc-06cb256503c358a72]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create
<- read (data resources)

Terraform will perform the following actions:
```



Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

- Las 4 instancias que se generan durante el proceso, la instancia que levanta packer, crea la imagen y después de tira, las dos instancias que soportan el aplicativo en la capa 1 y la instancia con la bd en la capa 2.

**Instances (4)** Info Last updated less than a minute ago Connect Instance state Actions Launch instances

Find Instance by attribute or tag (case-sensitive) All states < 1 >

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm s
<input type="checkbox"/>	Instance_stack_MEAN-2	i-07cb0282a2598f62b	Running	t2.micro	2/2 checks passec	<a href="#">View al</a>
<input type="checkbox"/>	Instance_stack_MEAN-1	i-0f62e698727659d2f	Running	t2.micro	2/2 checks passec	<a href="#">View al</a>
<input type="checkbox"/>		i-06ac3c7cf4ece3685	Terminated	t2.micro	-	<a href="#">View al</a>
<input type="checkbox"/>	MongoDB-Instance	i-0d4f38b3c589bc84f	Running	t2.micro	2/2 checks passec	<a href="#">View al</a>

- Despliegue de balanceador de carga

**Load balancers (1/1)** Actions Create load balancer

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Filter load balancers < 1 > ⚙

<input checked="" type="checkbox"/>	Name	DNS name	State	VPC ID	Availability Zones
<input checked="" type="checkbox"/>	app-load-balancer	app-load-balancer-203890...	Active	vpc-00020ebe526c7d69c	2 Availability Zones

- Captura de diferentes recursos levantados por terraform necesarios para desplegar el servicio y el correcto funcionamiento del sistema.

**Elastic IP addresses (3)** Actions

Find resources by attribute or tag

<input type="checkbox"/>	Name	Allocated IPv4 addr...	Type
<input type="checkbox"/>	Web-Server-EIP-1	<a href="#">100.28.195.142</a>	Public IP
<input type="checkbox"/>	MongoDB-EIP	<a href="#">3.232.105.156</a>	Public IP
<input type="checkbox"/>	Web-Server-EIP-2	<a href="#">54.205.212.51</a>	Public IP

**Key pairs (2)** Info Actions Create key pair

Find Key Pair by attribute or tag < 1 > ⚙

<input type="checkbox"/>	Name	Type	Created	Fingerprint	ID
<input type="checkbox"/>	unir	rsa	2025/01/15 09:09 GMT+10	23:e9:69:5f:00:02:9d:8f:fe...	key-0f9c036d3fb56539f
<input type="checkbox"/>	vockey	rsa	2025/01/12 14:25 GMT+10	16:78:9c:c0:47:1a:72:2c:b6...	key-0ec71e0006b85dd99

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

**Your VPCs (2)** [Info](#) Last updated less than a minute ago [Actions](#) [Create VPC](#)

<input type="checkbox"/>	Name	VPC ID	State	Block Public...	IPv4 CIDR
<input type="checkbox"/>	-	<a href="#">vpc-00020ebe526c7d69c</a>	Available	Off	172.31.0.0/16
<input type="checkbox"/>	CustomVPC	<a href="#">vpc-0443ccf3a68f59ef2</a>	Available	Off	172.31.16.0/24

**Subnets (8)** [Info](#) Last updated less than a minute ago [Actions](#) [Create subnet](#)

<input type="checkbox"/>	Name	Subnet ID	State	VPC
<input type="checkbox"/>	-	<a href="#">subnet-0925096eb2e62c8d5</a>	Available	<a href="#">vpc-00020ebe526c7d69c</a>
<input type="checkbox"/>	PublicSubnet1	<a href="#">subnet-09878078b3d921e04</a>	Available	<a href="#">vpc-0443ccf3a68f59ef2</a>   <a href="#">Custo...</a>
<input type="checkbox"/>	-	<a href="#">subnet-017384d8f249dd6ae</a>	Available	<a href="#">vpc-00020ebe526c7d69c</a>
<input type="checkbox"/>	-	<a href="#">subnet-07eeef2231e47a115</a>	Available	<a href="#">vpc-00020ebe526c7d69c</a>
<input type="checkbox"/>	-	<a href="#">subnet-0445c3360768e7952</a>	Available	<a href="#">vpc-00020ebe526c7d69c</a>
<input type="checkbox"/>	-	<a href="#">subnet-0d901bcd6f1f8f032</a>	Available	<a href="#">vpc-00020ebe526c7d69c</a>
<input type="checkbox"/>	PublicSubnet2	<a href="#">subnet-065edac1d74700f57</a>	Available	<a href="#">vpc-0443ccf3a68f59ef2</a>   <a href="#">Custo...</a>
<input type="checkbox"/>	-	<a href="#">subnet-07b06a48f83175c1d</a>	Available	<a href="#">vpc-00020ebe526c7d69c</a>

**Route tables (3)** [Info](#) Last updated 1 minute ago [Action](#)

<input type="checkbox"/>	Name	Route table ID	Explicit subnet associ...	
<input type="checkbox"/>	-	<a href="#">rtb-0866af37c63bbf02d</a>	-	-
<input type="checkbox"/>	PublicRouteTable	<a href="#">rtb-03e9360cd23fd3351</a>	2 subnets	-
<input type="checkbox"/>	-	<a href="#">rtb-0210debe3b0a7b08b</a>	-	-

- Comprobación de la conectividad entre instancias de la capa 1 y 2 usando el comando ping

```
Last login: Sun Jan 12 02:24:15 2025 from 116.255.16.57
ubuntu@ip-172-31-16-10:~$ ping 172.31.16.20
PING 172.31.16.20 (172.31.16.20) 56(84) bytes of data.
64 bytes from 172.31.16.20: icmp_seq=1 ttl=64 time=1.63 ms
64 bytes from 172.31.16.20: icmp_seq=2 ttl=64 time=1.77 ms
64 bytes from 172.31.16.20: icmp_seq=3 ttl=64 time=1.41 ms
```

Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

- Comprobación de conexión entre los procesos de Express en la instancia con el aplicativo y el proceso de MongoDB con la instancia de la capa 2

```
exception: connect failed
ubuntu@ip-172-31-16-10:~$ mongo --host 172.31.16.20 --port 27017
MongoDB shell version v3.6.8
connecting to: mongodb://172.31.16.20:27017/
Implicit session: session { "id" : UUID("923b622a-7ad7-4881-b211-19f6640acd53") }
MongoDB server version: 3.6.8
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
Server has startup warnings:
2025-01-12T02:49:17.023+0000 I STORAGE [initandlisten]
2025-01-12T02:49:17.023+0000 I STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2025-01-12T02:49:17.023+0000 I STORAGE [initandlisten] **      See http://dochub.mongodb.org/core/prodnotes-filesystem
2025-01-12T02:49:18.179+0000 I CONTROL [initandlisten]
2025-01-12T02:49:18.179+0000 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2025-01-12T02:49:18.179+0000 I CONTROL [initandlisten] **      Read and write access to data and configuration is unrestricted.
2025-01-12T02:49:18.179+0000 I CONTROL [initandlisten]
>
```

```
← → ↻ ⚠ No es seguro 35.175.221.195/api/saludo
Dar formato al texto ☐
```

```
{"mensaje":"Hola desde el backend, pero no se pudo conectar a MongoDB."}
```

- Output final del despliegue de sistema con terraform

```
Apply complete! Resources: 0 added, 1 changed, 0 destroyed.

Outputs:

load_balancer_dns = "app-load-balancer-1787038451.us-east-1.elb.amazonaws.com"
mongodb_nat_gateway_public_ip = "18.214.151.191"
web_server_private_ips = [
  "172.31.16.10",
  "172.31.16.11",
]
web_server_public_ips = [
  "98.85.155.40",
  "34.194.88.84",
]
web_server_ssh_commands = [
  "ssh -i id_rsa ubuntu@98.85.155.40",
  "ssh -i id_rsa ubuntu@34.194.88.84",
]
❖ aws_terraform>
```

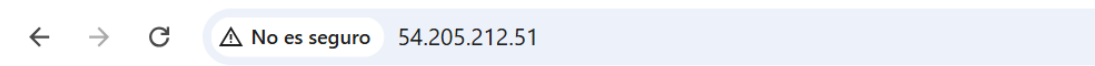
Asignatura	Datos del alumno	Fecha
Herramientas DevOps	Apellidos: Inglés Martínez	15/01/2025
	Nombre: Alejandro	

- Acceso final al servicio desde el navegador usando las ip de las instancias directamente



## Hola Mundo desde el Angular de la instancia 1

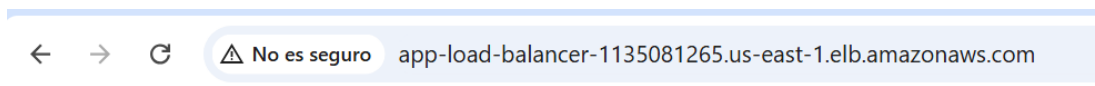
Hola desde el backend conectado correctamente a MongoDB!



## Hola Mundo desde el Angular de la instancia 2

Hola desde el backend conectado correctamente a MongoDB!

- Acceso final al servicio desde el navegador usando el nombre DNS del balanceador.



## Hola Mundo desde el Angular de la instancia 1

Hola desde el backend conectado correctamente a MongoDB!

### 13. Vídeo de la experimentación

<https://youtu.be/zRGhkBebEbA>