TERRAFORM: despliegue de MEAN multicapa mediante Terraform

Marco Speranza López

Índice

[1. Introducción 5](#_Toc153827560)

[2. Objetivoa 5](#_Toc153827561)

[3. Requisitos Previos 6](#_Toc153827562)

[4. Pasos a Seguir 6](#_Toc153827563)

[1. Creación de una cuenta AWS: 6](#_Toc153827564)

[2. Conexión con AWS desde mi host: 7](#_Toc153827565)

[3. Configuración Inicial de Packer: 8](#_Toc153827566)

[5. Creación de la plantilla de Packer 9](#_Toc153827567)

[6. Estructura de mi directorio. 11](#_Toc153827568)

[1. index.js 11](#_Toc153827569)

[2. package.json 11](#_Toc153827570)

[3. script.sh 11](#_Toc153827571)

[4. template.pkr.hc 13](#_Toc153827572)

[7. Creación de la AMI 13](#_Toc153827573)

[1. Inicialización de la Plantilla de Packer: 13](#_Toc153827574)

[2. Validación de la Plantilla de Packer: 13](#_Toc153827575)

[3. Construcción de la AMI con Packer: 14](#_Toc153827576)

[8. Lanzamiento de una Instancia EC2 14](#_Toc153827577)

[1. Seleccionar la AMI Personalizada: 14](#_Toc153827578)

[2. Asignar un Par de Claves SSH: 16](#_Toc153827579)

[3. Revisar y Lanzar: 16](#_Toc153827580)

[4. Acceso a la Aplicación: 16](#_Toc153827581)

[9. Resultado 17](#_Toc153827582)

[10. Conexión SSH 17](#_Toc153827583)

[1. Proceso Inicial 18](#_Toc153827584)

[2. Error Inicial 18](#_Toc153827585)

[3. Pasos para la Conexión SSH 19](#_Toc153827586)

[11. Verificación Automatizada Post-Despliegue 19](#_Toc153827587)

[Anexo: Cumplimiento de los Criterios de Evaluación 21](#_Toc153827588)

[Criterio 1 21](#_Toc153827589)

[Criterio 2 21](#_Toc153827590)

[Criterio 3 21](#_Toc153827591)

[Criterio 4 21](#_Toc153827592)

[Criterio 5 21](#_Toc153827593)

# Introducción

# Objetivos

El objetivo principal de esta actividad es familiarizarme con las herramientas de automatización de infraestructura y el despliegue de aplicaciones en la nube. Aprenderé a utilizar Packer para crear una imagen de instancia EC2 en AWS que estará preconfigurada con NGINX como servidor web y Node.js como entorno de ejecución para tu aplicación.

# Requisitos Previos

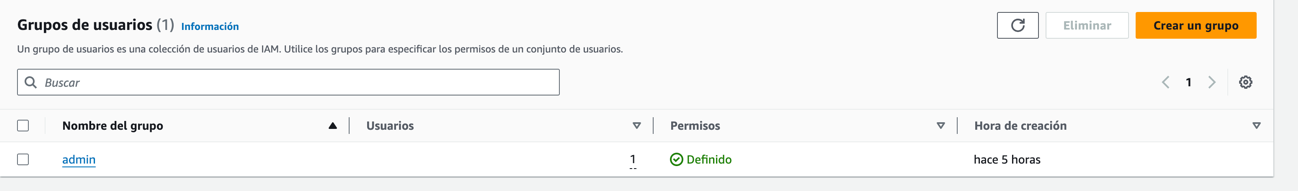
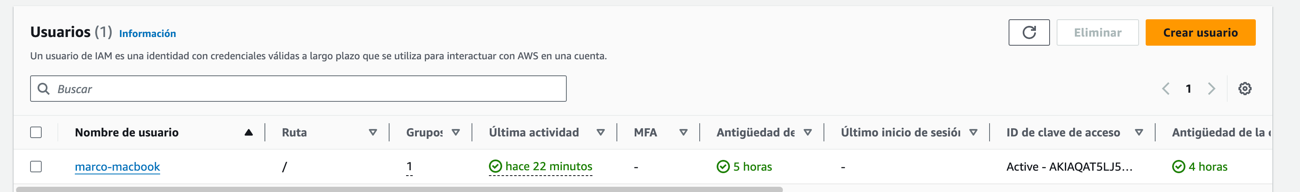
* Una cuenta de AWS con acceso a AWS (Identity and Access )Management (IAM) para crear instancias EC2 y AMIs.
* Packer instalado en mi entorno de desarrollo.
* Un par de claves SSH para acceder a las instancias EC2.
* Conocimiento básico de NGINX, Node.js y Linux.

# Pasos a Seguir

Esta guía está dividida en varios pasos que me llevarán desde la configuración inicial hasta el despliegue de mi aplicación en una instancia EC2 de AWS:

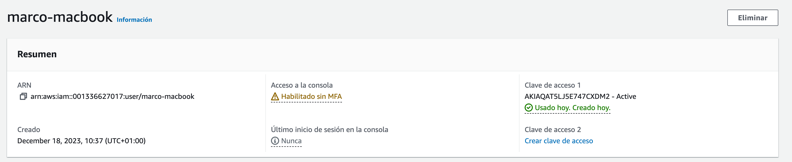
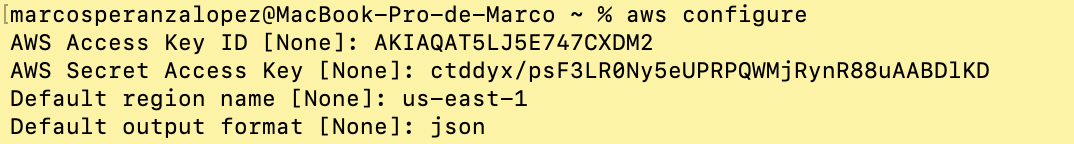
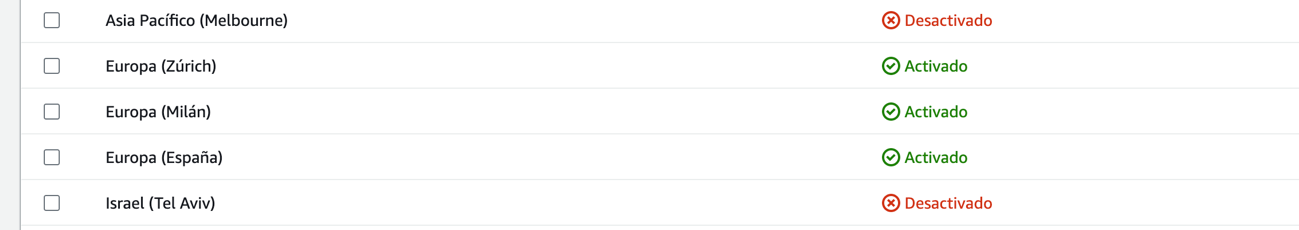
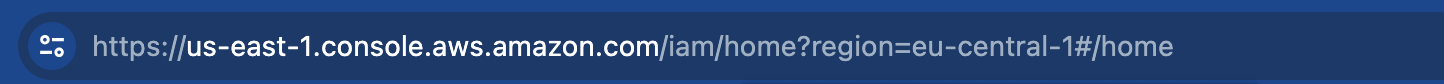
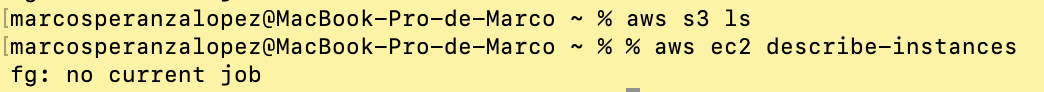
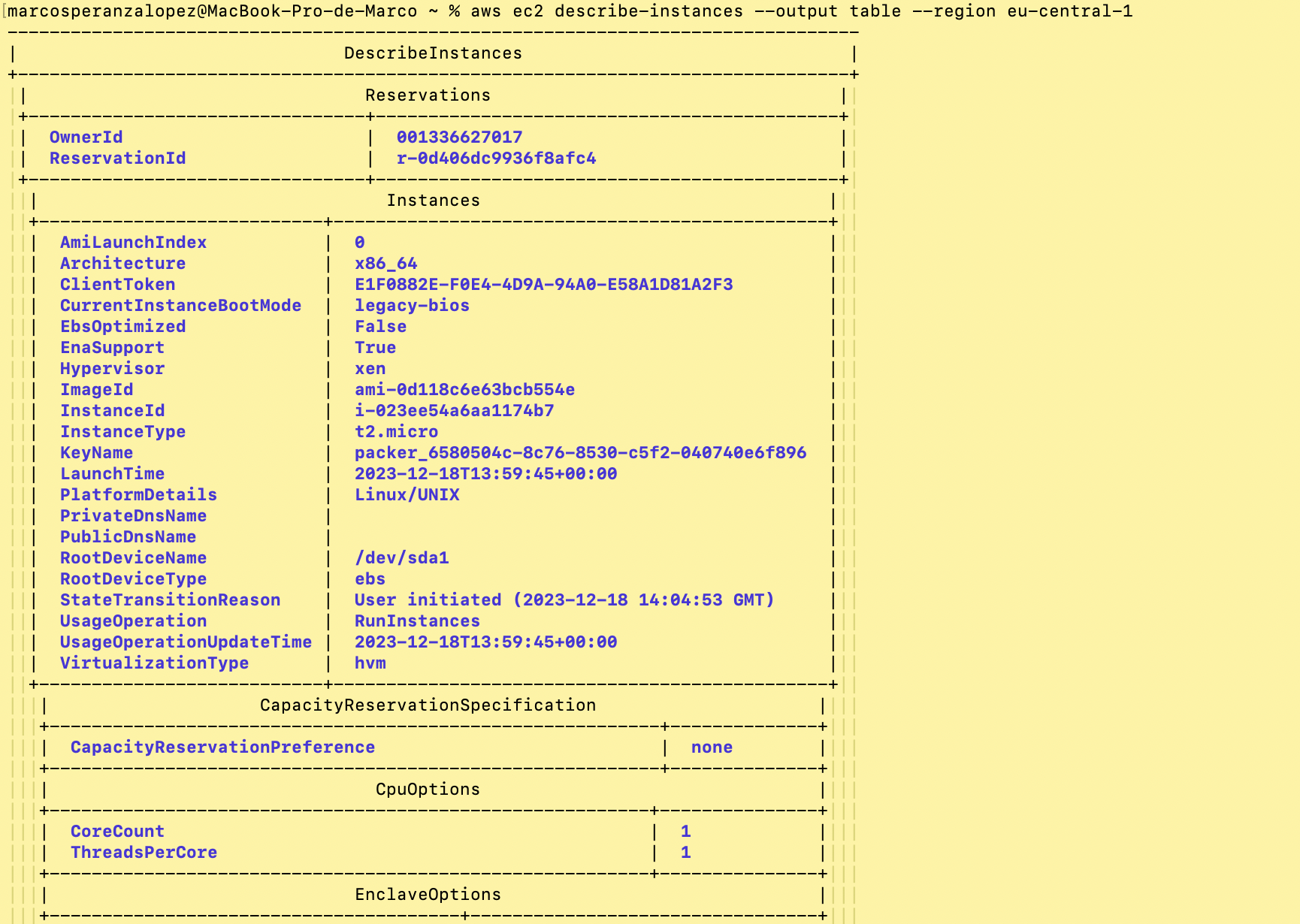
## Creación de una cuenta AWS:

Será necesario crear una cuenta AWS, que contenga:

* Un grupo de usuarios que llamare “admin”, y que contenga un rol de –“system administrator” dentro del IAM.
* Un usuario con nombre “marco-macbook” que este contenido en el grupo previamente definido, además de añadir una ID de clave de acceso, que posteriormente añadiremos en nuestro host, desde donde nos conectaremos aws, para lanzar la instancia.

## Conexión con AWS desde mi host:

Tras la creación de usuario, se añadirá una clave de acceso, desde donde nosotros podremos acceder a nuestro AWS.

1. Este es mi usuario con su respectiva clave de acceso.
2. A continuación, añadimos la clave de acceso, en mi host a través el comando “***aws configure***”
3. Más adelante cambié la región donde opera, debido a que tengo que tenerla activada en mi panel de AWS, cambiándola a **“eu-central-1”**. Además, a la hora de crear la imagen, será necesario definir en que región opera, y donde en donde está habilitada. Esto se puede añadir en la adminsitración de regiones. 
4. Podemos hacer una prueba de conexión con los siguientes comandos: 
5. Otra forma de ver la instancia seria a través de la descripción de instancias según la región. (Si cambiase de región mi instancia estaría vacia)  
   

## Configuración Inicial de Packer:

En esta etapa, nos enfocaremos en la configuración inicial de Packer para crear una AMI de EC2 con NGINX y Node.js preinstalados. Primero, debemos asegurarnos de que Packer esté instalado en nuestro entorno de desarrollo. En este caso, utilizaremos un MacBook como host y el gestor de paquetes Homebrew para la instalación.

A continuación, se detallan los comandos necesarios para la instalación y comprobación de Packer:

1. ***$ homebrew install packer***
2. ***$ packer -v***

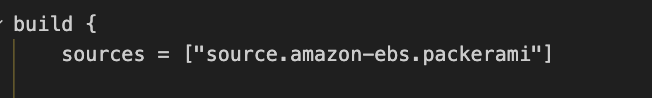
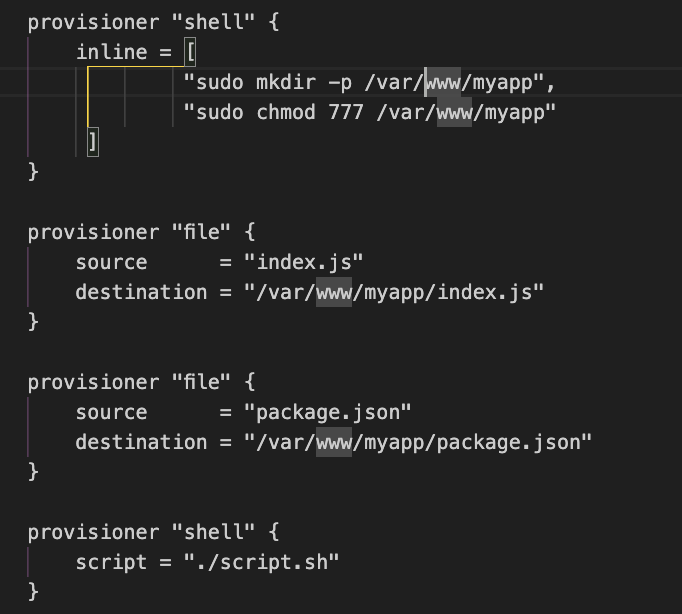
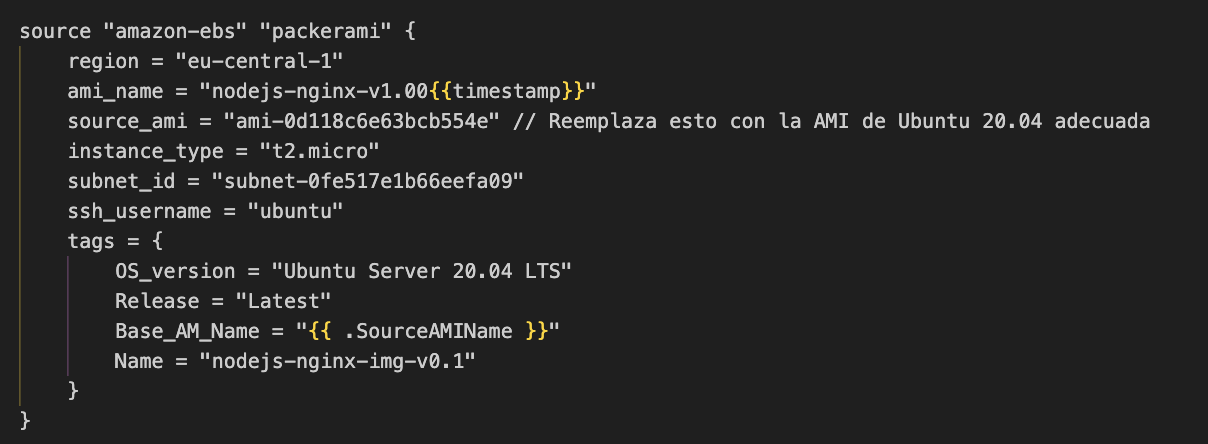
De la misma manera, y con la elección de la nube AWS, se instalará AWS CLI. La AWS CLI nos ayudará a interactuar con AWS y configurar el entorno para el aprovisionamiento de instancias EC2.

1. **$ homebrew install awscli**
2. **$ aws –version**

Con Packer y la AWS CLI correctamente instalados, hemos completado la configuración inicial necesaria para crear nuestra AMI personalizada. En los siguientes pasos, nos adentraremos en la creación de la plantilla Packer y el aprovisionamiento de la instancia EC2.

# Creación de la plantilla de Packer

La plantilla de Packer es un archivo JSON que define cómo se creará tu imagen. El esqueleto básico para la plantilla incluirá:

* **Builders**: Configuración de la imagen base, tipo de instancia, y detalles de la conexión SSH.
* **Provisioners**: Scripts que instalan y configuran Nginx, Node.js, y tu aplicación. Están dentro del builder.
* **Source**: Esta sección describe la fuente de la imagen base que se utilizará como punto de partida para tu AMI. Puede ser una AMI existente, una instantánea o incluso una imagen de disco personalizada.

En esta sección, hemos declarado la plantilla de Packer y especificado los plugins necesarios para interactuar con Amazon Web Services (AWS). Aquí, estamos usando el plugin "amazon" en su versión 0.0.2 o superior, que se encuentra en la fuente de GitHub de HashiCorp. En esta sección, estamos configurando el origen (source) de nuestra imagen. Hemos especificado varios parámetros importantes:

* region: La región de AWS en la que se creará la imagen. En este caso, estamos utilizando "eu-central-1".
* ami\_name: El nombre de la AMI que se creará, que incluye una marca de tiempo para garantizar la unicidad de los nombres de las AMIs.
* source\_ami: La AMI base que se utilizará como punto de partida para nuestra imagen. Asegúrate de reemplazar esto con la AMI adecuada de Ubuntu 20.04 según tus requisitos.
* instance\_type: El tipo de instancia EC2 que se utilizará durante el proceso de creación de la AMI. Aquí, hemos elegido "t2.micro" para una instancia de tamaño pequeño.
* subnet\_id: La ID de la subred de AWS en la que se ejecutará temporalmente la instancia durante el proceso de creación de la AMI.
* ssh\_username: El nombre de usuario para la conexión SSH a la instancia EC2.
* tags: Etiquetas que se asignarán a la AMI para una fácil identificación y organización.
* **Packer**: En esta sección, declaras las configuraciones específicas de Packer, como plugins y versiones requeridas. Esto asegura que Packer use las herramientas adecuadas para el proceso de creación de AMI.

# Estructura de mi directorio.

La estructura de directorio para el proyecto se organiza de la siguiente manera para facilitar el desarrollo, la configuración y el despliegue de la aplicación con Packer:

## index.js

El archivo principal de la aplicación Node.js que inicia el servidor web o realiza el procesamiento principal de la aplicación.

## package.json

Este archivo contiene la configuración del proyecto Node.js, incluyendo las dependencias necesarias para que la aplicación se ejecute correctamente.

## script.sh

El script.sh es un script de bash que automatiza la preparación de un servidor Ubuntu para hospedar una aplicación Node.js con NGINX como servidor web proxy inverso. A continuación, se detalla la función de cada sección del script:

#### Actualización del Sistema:

Se actualizan los paquetes del sistema operativo con sudo apt update y luego se aplica la actualización con sudo apt upgrade -y.

#### Instalación de NGINX y Node.js:

2. Se instalan NGINX, Node.js y NPM (Node Package Manager) utilizando sudo apt install -y nginx nodejs npm.

#### Instalación de una Versión Específica de Node.js:

3. Se utiliza un script proporcionado por NodeSource (para la versión 18.x) que configura el repositorio de Node.js adecuado y luego se instala esa versión específica con sudo apt install -y nodejs.

#### Compilación de Dependencias de Node.js:

4. Se instala build-essential para proporcionar las herramientas necesarias para la compilación de dependencias de Node.js.

#### Instalación de Dependencias de la Aplicación Node.js:

5. Se cambia la propiedad del directorio de la aplicación a ubuntu:ubuntu y se navega a este directorio para instalar las dependencias de Node.js con npm install.

#### Configuración del Servicio Node.js:

6. Se crea un archivo de configuración del servicio systemd para la aplicación Node.js, permitiendo que el sistema gestione la aplicación como un servicio. Este servicio se inicia con el servidor y se reinicia automáticamente en caso de fallo.

#### Habilitación e Inicio del Servicio Node.js:

7. Se habilita el servicio para que se inicie en el arranque del sistema con sudo systemctl enable nodeapp.service y se inicia el servicio con sudo systemctl start nodeapp.service.

#### Configuración de NGINX como Proxy Inverso:

8. Se configura NGINX para actuar como un proxy inverso que pasa las solicitudes al servidor Node.js que escucha en el puerto 3000. La configuración se escribe en el archivo predeterminado de sitios disponibles de NGINX.

#### Limpieza y Reinicio de NGINX:

9. Se eliminan enlaces simbólicos obsoletos de los sitios habilitados de NGINX y se establecen enlaces nuevos para asegurarse de que la configuración más reciente esté activa. Finalmente, se reinicia el servicio NGINX

## template.pkr.hc

La plantilla de configuración de Packer en formato HCL (HashiCorp Configuration Language), que define cómo se construye la imagen de la máquina virtual. Hacemos referia a este apartado en la sección 5.

# Creación de la AMI

La creación de la AMI (Amazon Machine Image) es un paso crítico en el proceso de despliegue de nuestra aplicación. Utilizamos Packer para automatizar la creación de esta imagen que incluirá nuestra aplicación y las configuraciones necesarias. El siguiente es el proceso que se lleva a cabo:

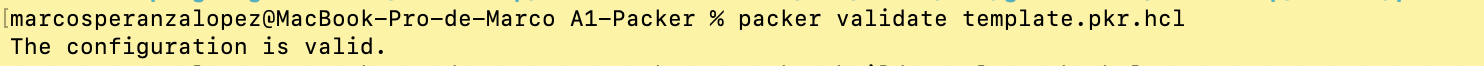
## Inicialización de la Plantilla de Packer:

Ejecutamos packer init template.pkr.hcl para inicializar nuestra plantilla de Packer. Esto descarga e instala los plugins necesarios, como se evidencia por el mensaje que indica que el plugin de Amazon ha sido instalado desde github.com/hashicorp/amazon.

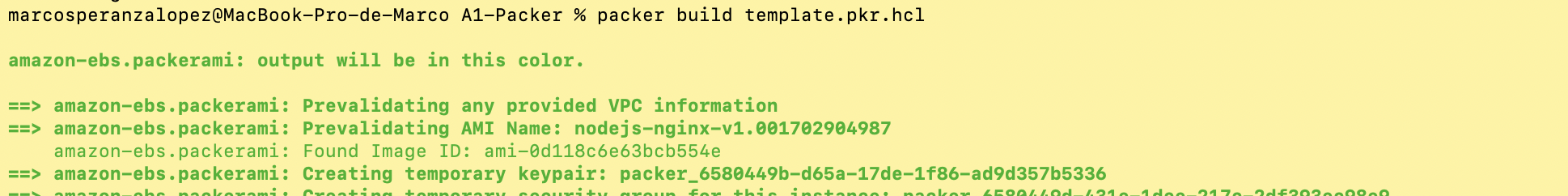


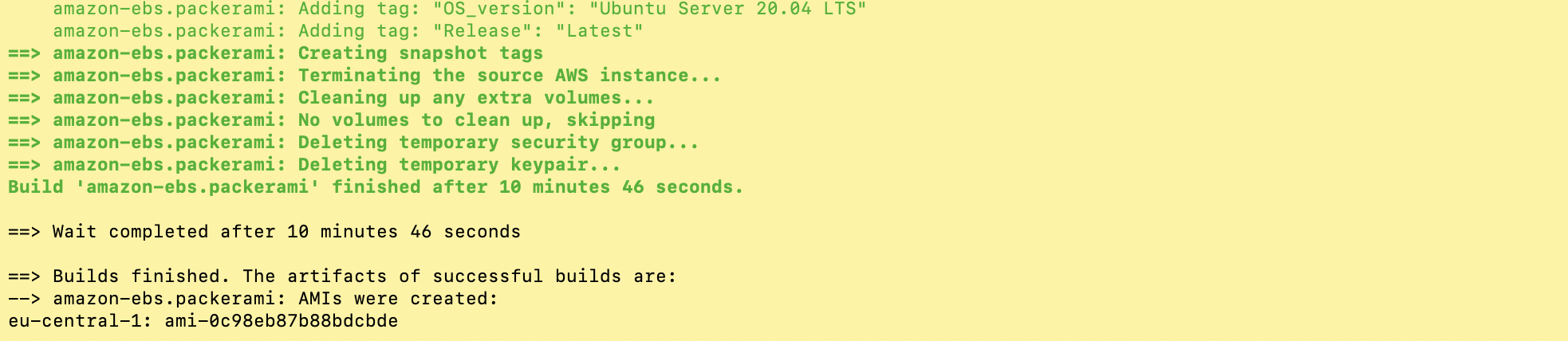
## Validación de la Plantilla de Packer:

Antes de proceder a construir la AMI, validamos nuestra plantilla con packer validate template.pkr.hcl. Este paso asegura que la configuración no tiene errores y es coherente.



## Construcción de la AMI con Packer:

Una vez validada la plantilla, ejecutamos **“*packer build template.pkr.hcl”***. Este comando inicia el proceso de construcción de la AMI basándose en las especificaciones de nuestra plantilla, la cual incluye la configuración del servidor, el software requerido y los scripts de aprovisionamiento.

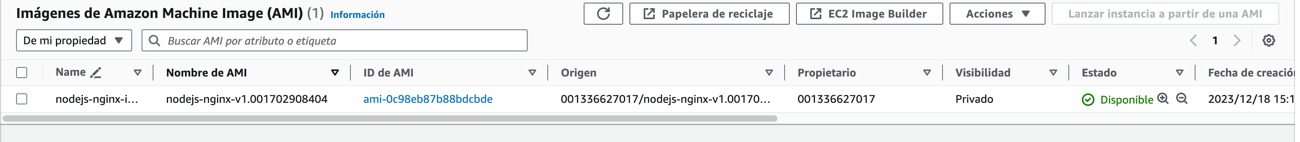


El resultado final de este proceso es una AMI de EC2 que está preconfigurada con las dependencias necesarias y listas para ser desplegadas en la nube de AWS.

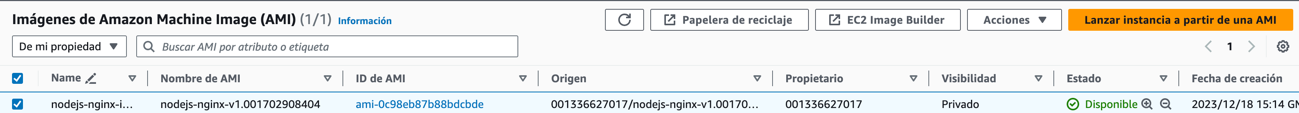
# Lanzamiento de una Instancia EC2

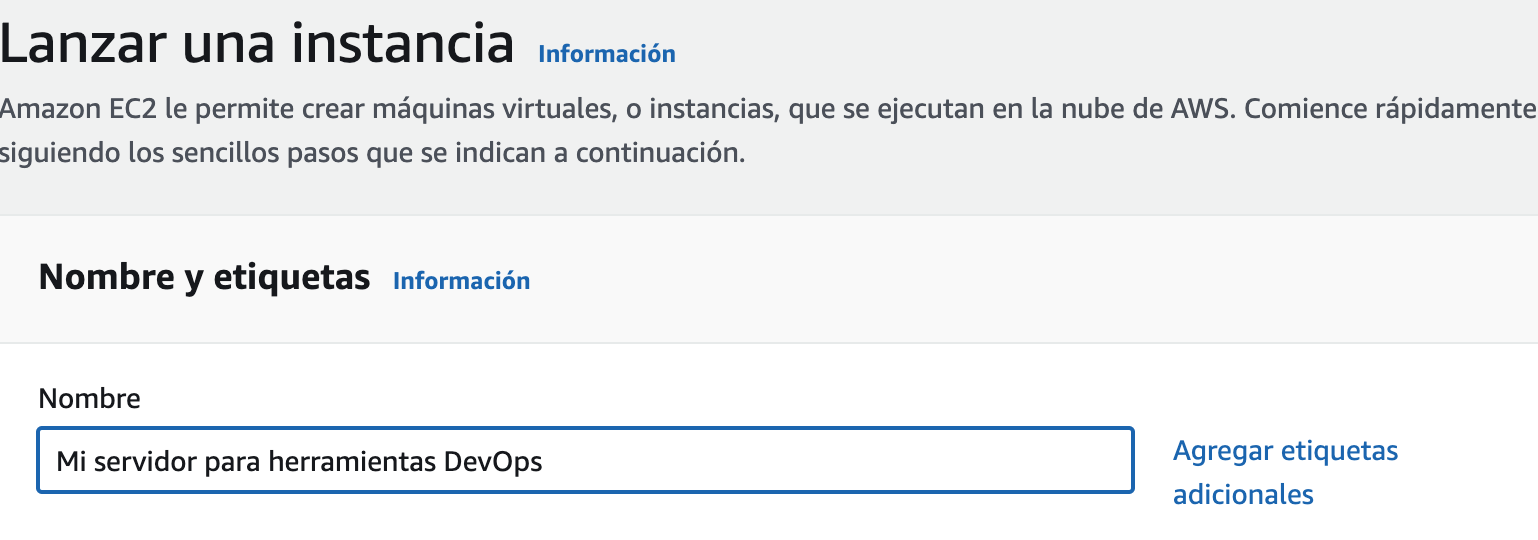
Una vez que la AMI está creada, el siguiente paso es utilizarla para lanzar una instancia EC2 en AWS. Esto permite que la aplicación esté operativa en un entorno de servidor en la nube. A continuación, se detallan los pasos para este proceso:

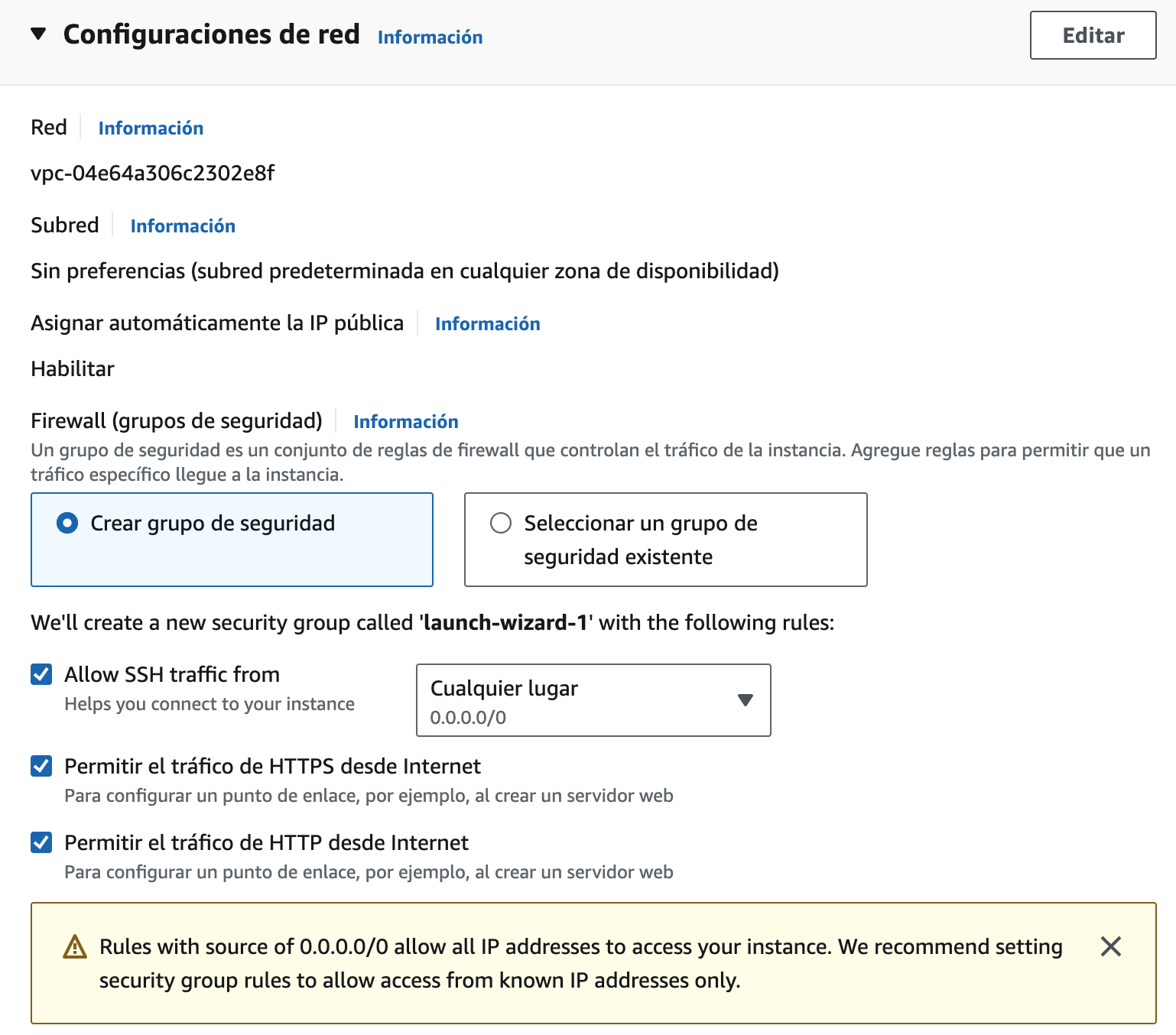
## Seleccionar la AMI Personalizada:

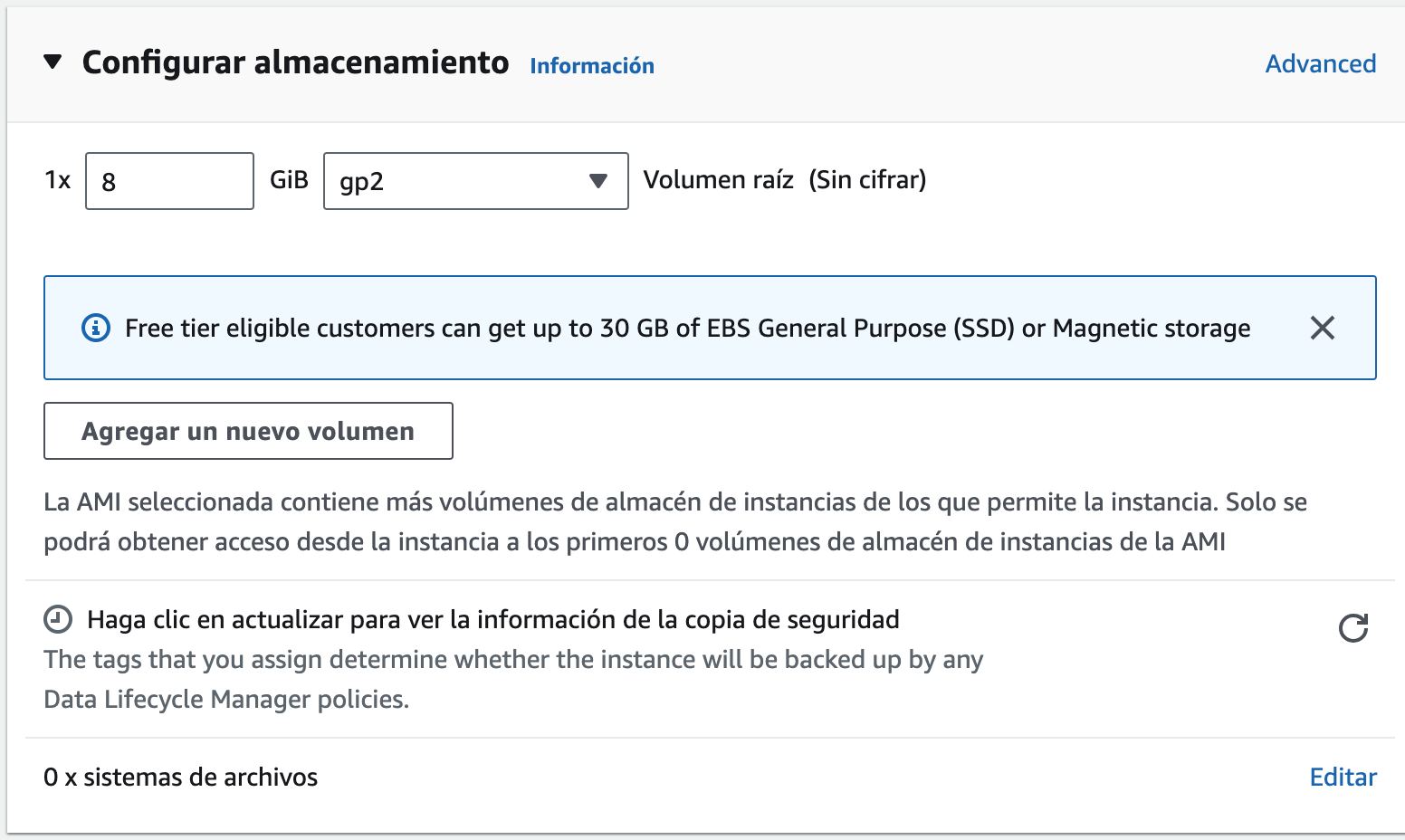
Me dirijo al servicio EC2. En el panel de EC2, buso la sección "AMIs" en el menú lateral, donde encontraré la AMI que acaba de crear con Packer.

Seleccionare la instancia iniciar el proceso de lanzamiento.



Configurarle el lanzamiento de la instancia

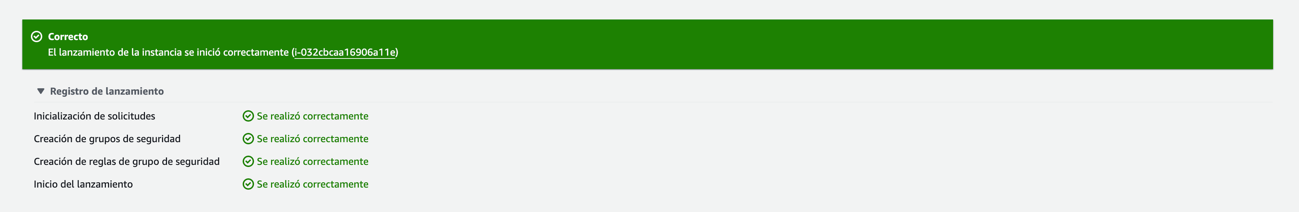
En mi caso voy a habilitar en las configuraciones de la red todo el tráfico.

El almacenamiento lo voy a dejar por defecto.

## Asignar un Par de Claves SSH:

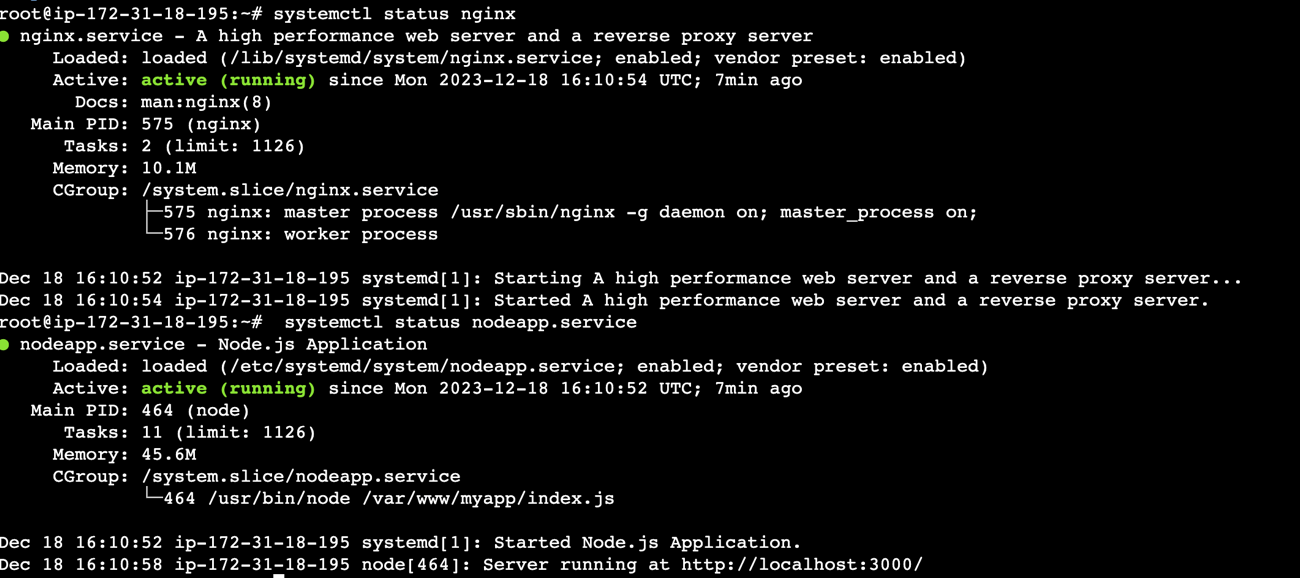
Para el acceso remoto seguro, asocie un par de claves SSH a la instancia. Si no tiene uno, AWS le permitirá crear un nuevo par de claves. Asegúrese de descargar y guardar la clave privada en un lugar seguro.

## Revisar y Lanzar:



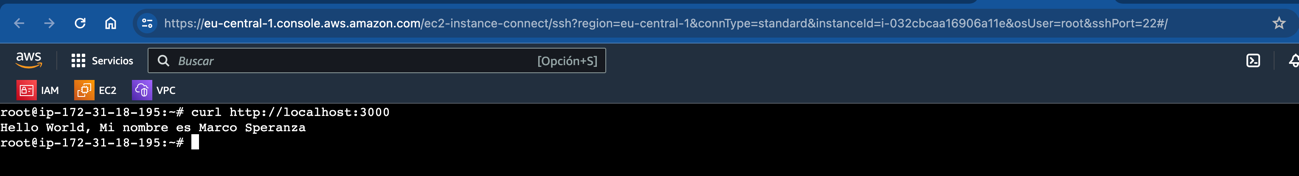
## Acceso a la Aplicación:

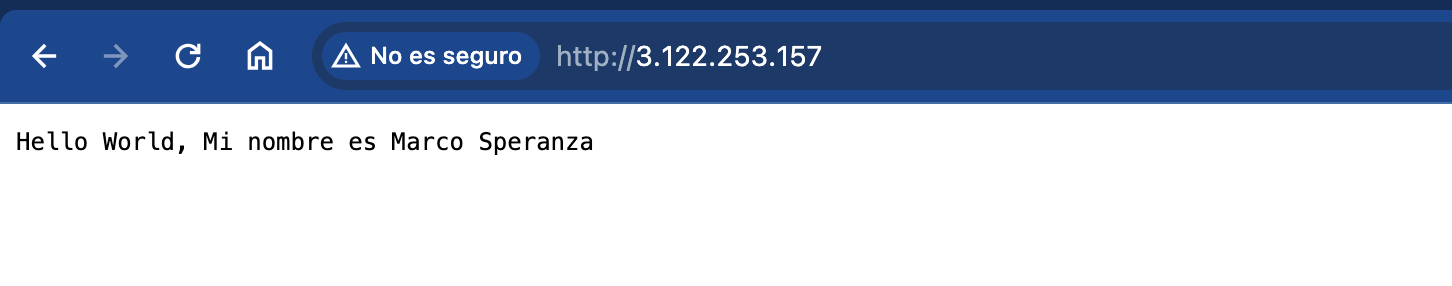
Una vez que la instancia está en ejecución, se puede acceder a ella utilizando la dirección IP pública o el nombre de host DNS proporcionado por AWS. Hemos accedió a través de EC2 Instance, directamente en el puerto.



# Resultado

Tras acceder comprobamos que podemos entrar de manera local en nuestra web, que reside en el localhost. Además de hacerlo a través de la IP pública.





# Conexión SSH

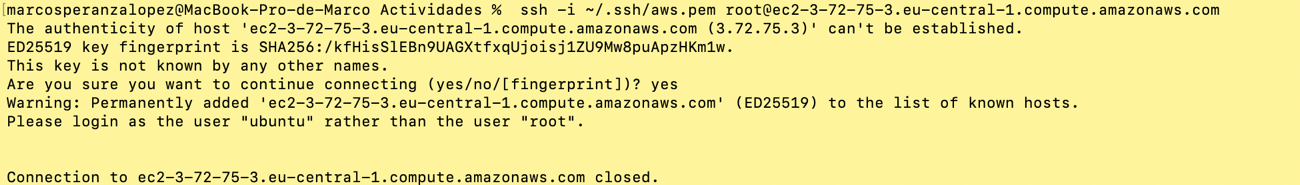
La conexión SSH es un método seguro para acceder a una instancia de Amazon EC2 desde una máquina local. En este caso, se utilizó un Mac para realizar la conexión. Aquí está el proceso detallado, incluido el error inicial y su corrección:

## Proceso Inicial

Al lanzar la instancia EC2, se genera un par de claves SSH (`.pem`). Este archivo de clave privada es esencial para establecer una conexión segura. Sin él, no es posible acceder a la instancia después de su creación.

## Error Inicial

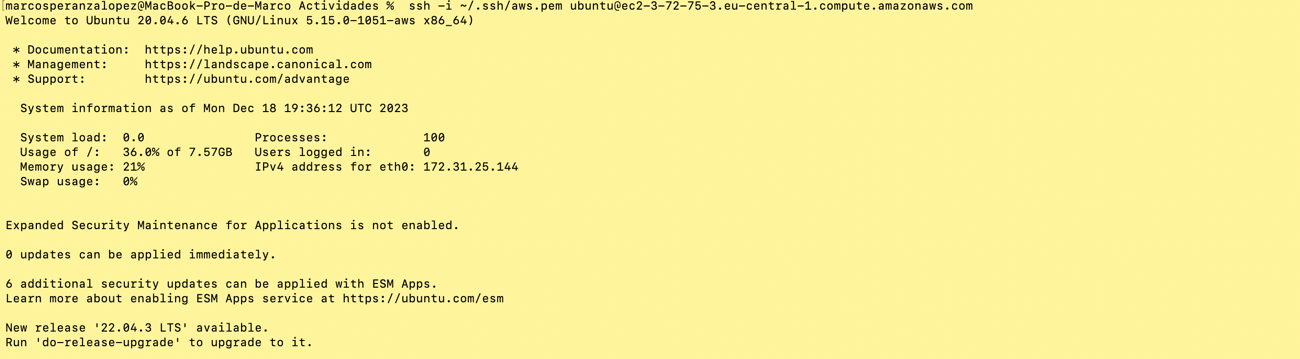
En el intento inicial de conexión, se utilizó el usuario `root` para acceder a la instancia:



Esto falló porque AWS ya no permite el acceso directo como `root` a través de SSH por razones de seguridad. En su lugar, se debe utilizar un nombre de usuario específico de la AMI. Por ejemplo, `ubuntu` para las AMIs de Ubuntu, `ec2-user` para las AMIs de Amazon Linux, etc.

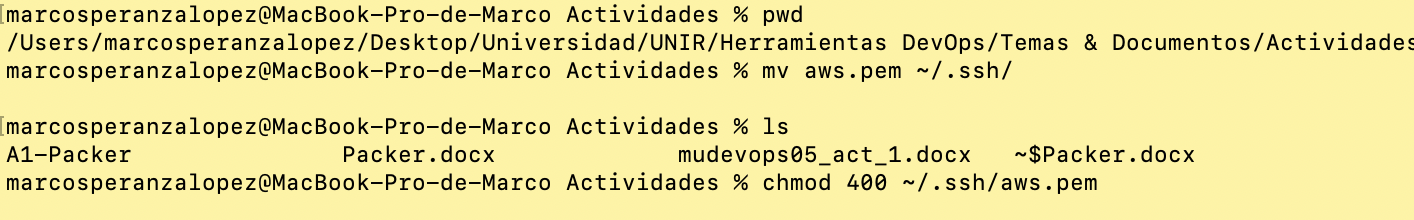
### Corrección y Conexión Exitosa

El problema se corrigió utilizando el nombre de usuario adecuado para la AMI de Ubuntu, que es ‘**ubuntu’**:



## Pasos para la Conexión SSH

1. Verificar Archivo de Clave Privada: Me aseguro de que el archivo `.pem` esté en una ubicación segura y tenga los permisos correctos:



2. Conectar a la Instancia EC2: Utiliza el comando SSH con el nombre de usuario y la dirección DNS pública de la instancia:

3. Acceso Exitoso: Una vez que la conexión se ha establecido, estarás en el shell de la instancia EC2, donde puedes administrar tu servidor como sea necesario.

# Verificación Automatizada Post-Despliegue

Se desarrolló un script de automatización (`automate.sh`) para validar la implementación de la aplicación Node.js. Este procedimiento se basó en la imagen de AMI generada mediante Packer, y se centró en establecer una conexión SSH sin intervención manual a la nueva instancia EC2. El objetivo era verificar que los servicios de Node.js y NGINX estuvieran operativos y que la aplicación respondiera correctamente a las solicitudes HTTP.

Los pasos definidos en el script fueron:

- Iniciar una conexión SSH a la instancia EC2 utilizando la clave privada SSH correspondiente, estableciendo así un canal de comunicación seguro para las operaciones subsiguientes.

- Verificar el estado del servicio Node.js con el comando ‘systemctl status nodeapp.service’, confirmando que el servicio estuviera activo y corriendo.

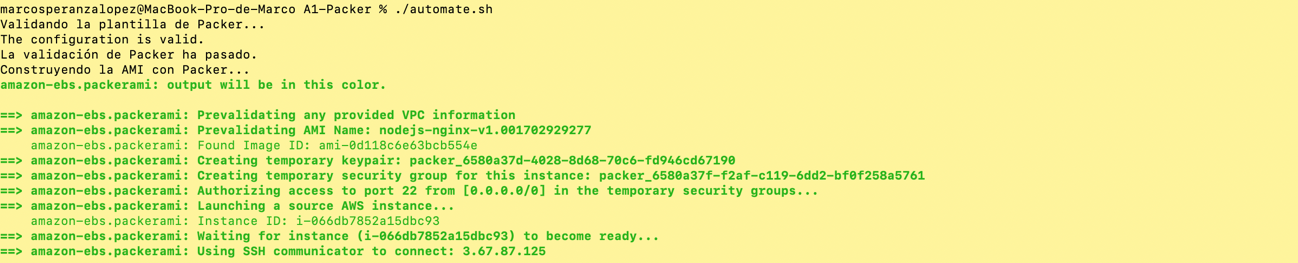
- Comprobar la operatividad de NGINX mediante ‘systemctl status nginx’, asegurando que estuviera configurado para manejar el tráfico HTTP.

- Realizar una solicitud a la aplicación Node.js con ‘curl [http://localhost:3000`](http://localhost:3000%60)’ desde la instancia, obteniendo la respuesta esperada con el mensaje "Hello World". La salida se visualizó en consola para confirmar que la aplicación estaba activa y funcionando correctamente.

- Registrar la salida de estos comandos para documentar la correcta funcionalidad y para servir como evidencia del proceso de despliegue.

El script se ejecutó inmediatamente después de la creación de la AMI por Packer y una vez que la instancia EC2 estaba en estado 'running'. Este enfoque formó parte crucial del pipeline de integración y despliegue continuo, facilitando una automatización completa del proceso de despliegue y validación de la aplicación, y garantizando la calidad y consistencia en cada iteración.

Al concluir la ejecución del script, se confirmó que la instancia EC2 estaba completamente operativa, con la aplicación Node.js accesible y respondiendo adecuadamente a las solicitudes HTTP. Esto no solo demostró la efectividad del despliegue automatizado, sino que también reforzó el concepto de inmutabilidad en el entorno de despliegue.



# Anexo: Cumplimiento de los Criterios de Evaluación

## Criterio 1

Se ha completado con éxito la creación de una imagen de Packer en AWS, demostrando la habilidad de manejar herramientas de infraestructura como código y automatización en la nube.

## Criterio 2

Se ha instalado correctamente el stack necesario para el funcionamiento de la aplicación en AWS, y se ha proporcionado documentación exhaustiva del proceso a través de logs y capturas de pantalla que evidencian cada paso del proceso.

## Criterio 3

El Builder para otra nube pública no se ha desarrollado en el ámbito de esta actividad.

## Criterio 4

El despliegue desde la imagen construida ha mostrado la aplicación funcionando automáticamente sin intervención manual. Se ha incluido evidencia de este proceso mediante pantallazos que muestran tanto el despliegue efectivo como la respuesta satisfactoria al "Hello World" de la aplicación.

## Criterio 5

La explicación de los entregables ha sido detallada, mostrando comprensión profunda del proceso de despliegue. Se han identificado y explicado las limitaciones de la práctica, asegurando una descripción completa y entendimiento del proyecto.