Actividad: Despliegue de Nginx y Nodejs mediante una imagen generada por Packer

CESAR FERNANDO CASTRO PARADA

cesar.castroparada@gmail.com

Table of Contents

[1. Introducción 3](#_Toc183386667)

[1.1. Objetivos 3](#_Toc183386668)

[1.2. Descripción de la Actividad 3](#_Toc183386669)

[2. Ejercicio 1: Creación de una Template de Packer 4](#_Toc183386670)

[2.1. Preparación del Entorno 4](#_Toc183386671)

[2.2. Creación de la Plantilla 6](#_Toc183386672)

[2.3. Resultados Obtenidos 8](#_Toc183386673)

[3. Ejercicio 2: Despliegue Automático Sin Intervención Manual 9](#_Toc183386674)

[3.1. Modificaciones en la Plantilla 9](#_Toc183386675)

[3.2. Implementación en AWS y Uso del CLI 9](#_Toc183386676)

[3.3. Resultados Obtenidos 13](#_Toc183386677)

[4. Ejercicio 3: Despliegue Multinube 15](#_Toc183386678)

[4.1. Justificación del Uso de Google Cloud 15](#_Toc183386679)

[4.2. Configuración del Builder para Google Cloud 16](#_Toc183386680)

[4.3. Resultados Obtenidos 24](#_Toc183386681)

[5. Conclusión y Lecciones Aprendidas 27](#_Toc183386682)

1. Introducción

1.1. Objetivos

En esta actividad, hemos aprendido a utilizar Packer para automatizar la creación de imágenes reutilizables que incluyan Node.js, Nginx y una aplicación sencilla para probar las imágenes y su correcta configuración.

Los objetivos alcanzados son:

* Automatizar la instalación y configuración de Node.js y Nginx en imágenes base de Ubuntu 24.04.
* Realizar despliegues en AWS y GCP sin intervención manual.
* Implementar un despliegue multinube utilizando una única plantilla de Packer.

1.2. Descripción de la Actividad

La actividad se basó en la guía proporcionada por [DigitalOcean](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-20-04), que detalla cómo configurar Node.js con Nginx en Ubuntu 20.04, adaptándola para su automatización mediante Packer y logrando despliegues en nubes públicas, principalmente AWS y GCP.

Es importante resaltar que consiste en 3 fases que ayudan a conocer mas a fondo las funcionalidades de Packer así como la implementación de la automatización con esta herramienta.

Fase 1: Creación de un template de Packer para la creación de una AMI en AWS.

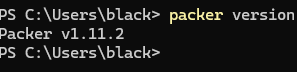
Fase 2: Agregar automatización a la plantilla creada para poder desplegar una instancia a partir de la imagen creada en EC2.

Fase 3: Agregar otro proveedor de nube para crear una plantilla multicloud para crear dos imágenes con la misma configuración en dos nubes diferentes y levantar instancias de maquinas virtuales a partir de las imágenes creadas en sus respetivas nubes, todo esto de manera automática.

2. Ejercicio 1: Creación de una Template de Packer

2.1. Preparación del Entorno

* Instalamos Packer en nuestro entorno local (en mi caso mi laptop personal con Windows 11 Pro) y configuramos las variables de entorno necesarias para usar Packer mediante la consola. (En path, se configura la ruta donde guardaste la descarga de Packer, en mi caso en Program Files: C:\Program Files (x86)\packer\_1.11.2\_windows\_amd64 )



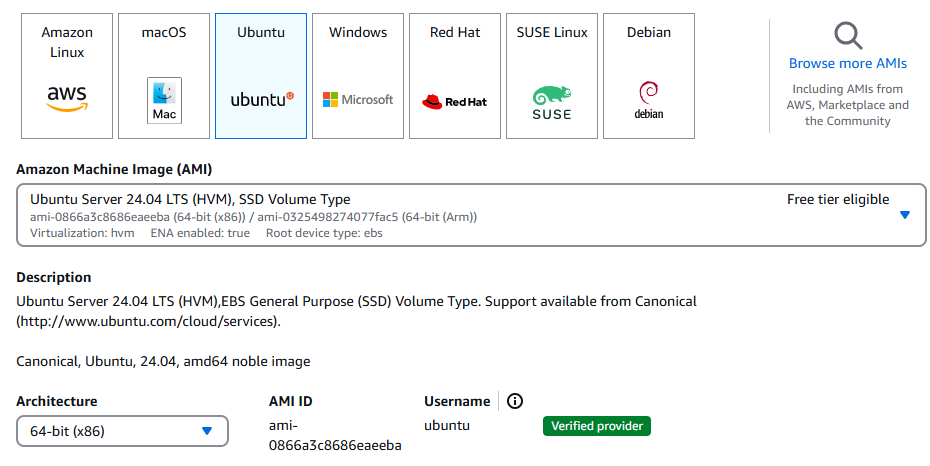
* Instalamos el CLI de AWS desde el sitio web oficial de AWS.

Amazon Web Services (2024). *Installing or updating to the latest version of the AWS CLI.* Amazon Web Services.

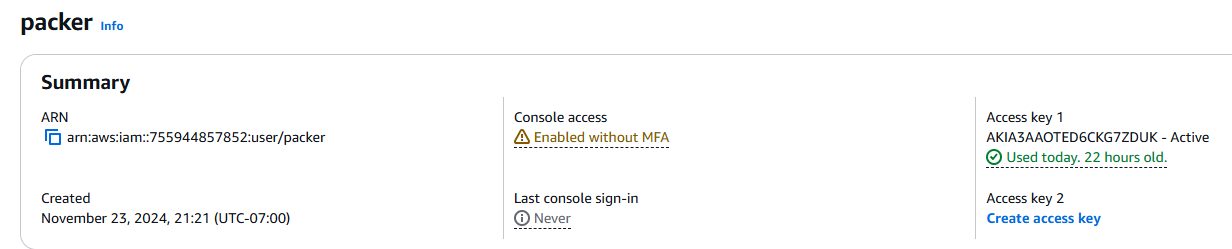
https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

* Seleccionamos una AMI base con Ubuntu 24.04 LTS para la región

us-east-1 con soporte para HVM y almacenamiento EBS. En mi caso seleccione la ami “ami-0866a3c8686eaeeba” directo desde la consola de AWS en la sección de EC2 para mayor practicidad. Es una imagen de Ubuntu 24.04 (Canonical, Ubuntu, 24.04, amd64 noble image) y fue seleccionada por ser parte del free tier de AWS.

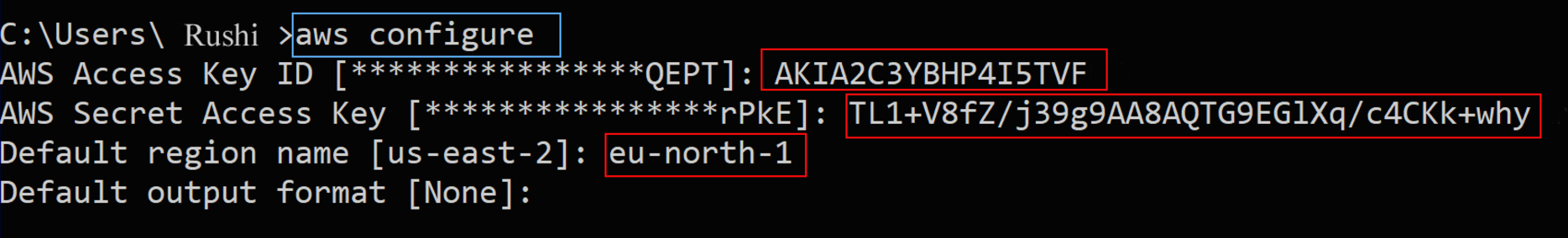


* En AWS IAM, creamos un usuario con los permisos necesarios para poder interactuar con los servicios de EC2 solamente y creamos llaves de acceso para eso usuario para poder interactuar con AWS mediante comandos de consola y su API. (En este caso solo cree un usuario llamado Packer, el cual será usado por Packer para interactuar con AWS EC2.)



* Configuramos las credenciales de AWS en el entorno local mediante el comando aws configure e ingresamos las credenciales que generamos para el usuario “Packer” con permisos para EC2.

Imagen de ejemplo de los comandos:



Estos son los requerimientos mínimos para iniciar a crear imágenes de AWS EC2 de manera local.

2.2. Creación de la Plantilla

Creamos una plantilla de Packer (template.pkr.hcl) que automatiza la creación de una imagen en AWS con Node.js y Nginx preinstalados y configurados. Esta plantilla incluye los siguientes elementos:

1. packer {

2. required\_plugins {

3. amazon = {

4. version = ">= 1.3.3"

5. source = "github.com/hashicorp/amazon"

6. }

7. }

8. }

9.

10. variable "aws\_region" {

11. default = "us-east-1"

12. }

13.

14. variable "instance\_type" {

15. default = "t2.micro"

16. }

17.

18. variable "ami\_id" {

19. default = "ami-0866a3c8686eaeeba"

20. }

21.

22. source "amazon-ebs" "node\_nginx\_image" {

23. region = var.aws\_region

24. instance\_type = var.instance\_type

25. source\_ami = var.ami\_id

26. ssh\_username = "ubuntu"

27. ami\_name = "packer-node-nginx-{{timestamp}}"

28. associate\_public\_ip\_address = true

29. }

30.

31. build {

32. name = "node-nginx-image"

33. sources = ["source.amazon-ebs.node\_nginx\_image"]

34.

35. provisioner "shell" {

36. inline = [

37. "export DEBIAN\_FRONTEND=noninteractive",

38. "sudo apt update -y",

39. "sudo apt upgrade -y",

40. "sudo apt install -y curl gnupg software-properties-common",

41. "curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -",

42. "sudo apt install -y nodejs nginx",

43. "sudo npm install -g pm2",

44. "echo 'const http = require(\"http\"); const hostname = \"localhost\";const port = 3000; const server = http.createServer((req, res) => { res.statusCode = 200; res.setHeader(\"Content-Type\", \"text/plain\"); res.end(\"Hello World!\"); }); server.listen(port, hostname, () => { console.log(`Server running at http://$$hostname:$$port/`); });' > /home/ubuntu/hello.js",

45. "sudo -u ubuntu pm2 start /home/ubuntu/hello.js",

46. "sudo -u ubuntu pm2 save",

47. "sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u ubuntu --hp /home/ubuntu",

48. "sudo systemctl enable pm2-ubuntu || true",

49. "sudo systemctl restart pm2-ubuntu || true",

50. "echo 'server { listen 80; server\_name \_; location / { proxy\_pass http://localhost:3000; 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; } }' | sudo tee /etc/nginx/sites-available/default",

51. "sudo ln -sf /etc/nginx/sites-available/default /etc/nginx/sites-enabled/default",

52. "sudo nginx -t || true",

53. "sudo systemctl restart nginx || true"

54. ]

55. }

56.

57. provisioner "shell" {

58. inline = [

59. "sudo ufw allow 'Nginx Full'",

60. "sudo systemctl enable nginx"

61. ]

62. }

63. }

64.

Código del Archivo template.pkr.hcl

Plugins y Variables:

Se configura el plugin requerido para AWS.

Las variables permiten definir parámetros reutilizables como la región, tipo de instancia y AMI base.

Builder:

Se utiliza amazon-ebs como builder, especificando detalles como la AMI base, el tipo de instancia (t2.micro) y la configuración de red pública.

Provisioners:

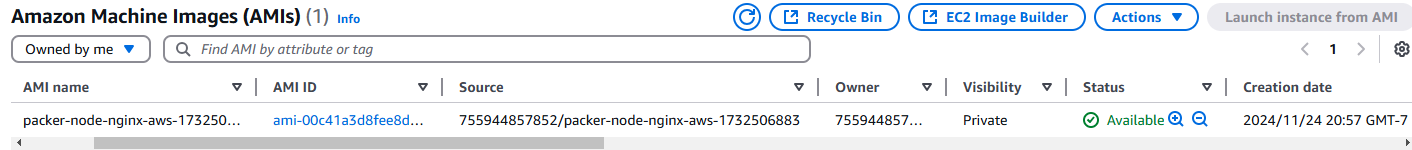
Primer provisioner: Configura el entorno con Node.js, instala Nginx y PM2, ¡y crea un script Node.js básico que responde con “Hello World!". También configura Nginx como proxy inverso y asegura que el script y los servicios inicien automáticamente.

Segundo provisioner: Configura el firewall para permitir tráfico HTTP y HTTPS, y habilita Nginx para iniciar automáticamente.

2.3. Resultados Obtenidos

Una imagen (AMI) con Ubuntu 24.04, lista para ejecutar aplicaciones Node.js con un servidor proxy configurado (Nginx) y una aplicación sencilla para pruebas que se inicia de manera automática cuando se crea una instancia de maquina virtual a partir de esa imagen.

Permite crear instancias listas para producción en AWS.



3. Ejercicio 2: Despliegue Automático Sin Intervención Manual

En esta sección, se implementó un despliegue completamente automatizado utilizando Packer y GitHub Actions para crear una imagen AMI en AWS y lanzar una instancia EC2 basada en esta imagen.

3.1. Modificaciones en la Plantilla

En este caso no fue necesario hacer modificaciones a la plantilla (template.pkr.hcl). Sigue utilizando el plugin de AWS y el mismo procedimiento de provisionamiento, pero ahora está integrada en un flujo CI/CD con GitHub Actions para la automatización. Decidí elegir GitHub Actions por su facilidad para crear flujos de trabajo y porque mis plantillas estaban almacenadas en un repositorio de GitHub para mantener un control de versiones adecuado. Existían otras opciones como usar Ansible o Terraform por ejemplo, pero al final decidí usar GitHub Actions por su practicidad e integración con mi repositorio.

<https://github.com/Lordbear117/packer-image-template>

3.2. Implementación en AWS y Uso del CLI

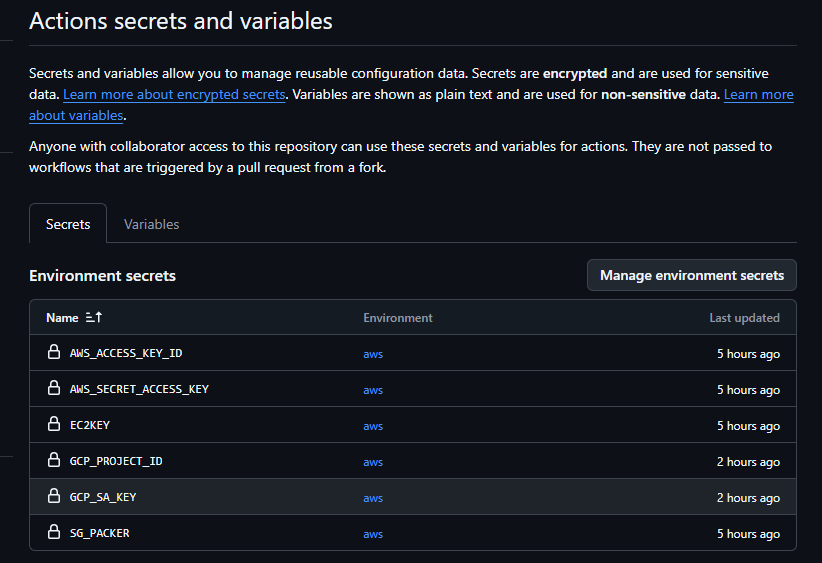
Se utilizó un flujo de trabajo en GitHub Actions para:

* Construir la imagen AMI: Ejecutar la plantilla de Packer en AWS para generar una nueva AMI con Node.js y Nginx.

Desplegar automáticamente la instancia EC2:

* Verificar y detener instancias anteriores con el mismo nombre.
* Lanzar una nueva instancia EC2 basada en la AMI creada.
* Configurar la instancia con parámetros como el grupo de seguridad, la clave SSH y el nombre único.

Para las variables de entorno y llaves secretas de Github fue necesario crear un nuevo entorno de secretos. En este caso mi entorno se llama aws y ahí se crearon las llaves junto con su valor para ser usadas en el flujo de trabajo en un archivo .yml llamado packer-ec2.yml, esto dentro de la carpeta .github/workflows, que es donde se guardan los workflows que GitHub Actions toma de manera automática desde un repositorio.



A continuación, comparto el workflow que realiza todas estas acciones y en donde se puede ver el uso del entorno aws y sus variables secretas.

1. name: Build and Deploy EC2 Instance with Packer and GitHub Actions

2.

3. on:

4.   push:

5.     branches:

6.       - main

7.

8. jobs:

9.   build-ami:

10.     runs-on: ubuntu-latest

11.     environment: aws

12.

13.     steps:

14.       - name: Checkout Code

15.         uses: actions/checkout@v3

16.

17.       - name: Install Packer

18.         uses: hashicorp/setup-packer@v2

19.         with:

20.           version: latest

21.

22.       - name: Configure AWS Credentials

23.         uses: aws-actions/configure-aws-credentials@v3

24.         with:

25.           aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

26.           aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

27.           aws-region: us-east-1

28.

29.       - name: Init Packer

30.         run: packer init template\_automatic.pkr.hcl

31.

32.       - name: Build AMI with Packer

33.         run: packer build template\_automatic.pkr.hcl

34.

35.   deploy-ec2:

36.     needs: build-ami

37.     runs-on: ubuntu-latest

38.     environment: aws

39.

40.     steps:

41.       - name: Configure AWS Credentials

42.         uses: aws-actions/configure-aws-credentials@v3

43.         with:

44.           aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

45.           aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

46.           aws-region: us-east-1

47.

48.       - name: Install jq

49.         run: sudo apt-get install jq -y

50.

51.       - name: Check and Terminate Existing EC2 Instance

52.         run: |

53.           INSTANCE\_NAME="Packer Instance-$(date +%Y%m%d)"

54.           INSTANCE\_ID=$(aws ec2 describe-instances --filters "Name=tag:Name,Values=$INSTANCE\_NAME" --query "Reservations[0].Instances[0].InstanceId" --output text)

55.

56.           if [ "$INSTANCE\_ID" != "None" ]; then

57.             echo "La instancia con el nombre '$INSTANCE\_NAME' ya existe, deteniendo y eliminando..."

58.             # Detener y eliminar la instancia existente

59.             aws ec2 terminate-instances --instance-ids $INSTANCE\_ID

60.             aws ec2 wait instance-terminated --instance-ids $INSTANCE\_ID

61.             echo "Instancia terminada y eliminada."

62.           else

63.             echo "No se encontró una instancia existente con el nombre '$INSTANCE\_NAME'. Lanzando una nueva..."

64.           fi

65.

66.       - name: Launch EC2 Instance

67.         run: |

68.           AMI\_ID=$(aws ec2 describe-images --filters "Name=name,Values=packer-node-nginx-\*" --query "Images | sort\_by(@, &CreationDate)[-1].ImageId" --output text)

69.           DEFAULT\_SUBNET=$(aws ec2 describe-subnets --filters "Name=default-for-az,Values=true" --query "Subnets[0].SubnetId" --output text)

70.

71.           INSTANCE\_NAME="Packer Instance-$(date +%Y%m%d)"  # Nombre único para la nueva instancia

72.

73.           # Lanzar la nueva instancia EC2 con el nombre único

74.           INSTANCE\_DETAILS=$(aws ec2 run-instances --image-id $AMI\_ID --instance-type t2.micro --key-name ${{ secrets.EC2KEY }} --security-group-ids ${{ secrets.SG\_PACKER }} --subnet-id $DEFAULT\_SUBNET --region us-east-1 --associate-public-ip-address --tag-specifications "ResourceType=instance,Tags=[{Key=Name,Value=$INSTANCE\_NAME}]" --output json)

75.

76.           # Obtener el ID de la instancia

77.           INSTANCE\_ID=$(echo $INSTANCE\_DETAILS | jq -r '.Instances[0].InstanceId')

78.           echo "La instancia EC2 ha sido lanzada con ID: $INSTANCE\_ID"

79.

80.           # Esperar a que la instancia esté en estado 'running'

81.           aws ec2 wait instance-running --instance-ids $INSTANCE\_ID

82.           echo "La instancia está en estado 'running'."

83.

84.           # Obtener la IP pública de la instancia

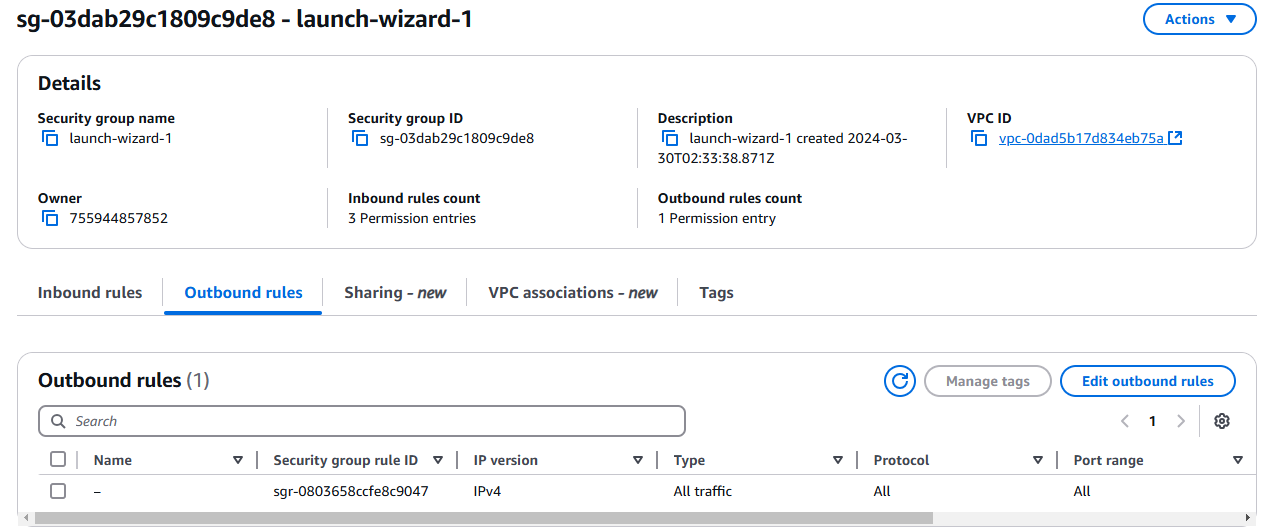
85.           PUBLIC\_IP=$(aws ec2 describe-instances --instance-ids $INSTANCE\_ID --query "Reservations[0].Instances[0].PublicIpAddress" --output text)

86.

87.           echo "La IP pública de la instancia es: $PUBLIC\_IP"

88.

También es necesario crear un grupo de seguridad en AWS para asignarlo a las nuevas instancias que se creen y permitir el trafico por HTTP Y HTTPS a las nuevas instancias. Los grupos de seguridad son propios de AWS y se encuentran a nivel de instancias de EC2. En este caso se tomo un grupo de seguridad llamado launch-wizard-1 y es el que se usara para todas las maquinas virtuales creadas de manera automática.



3.3. Resultados Obtenidos

Flujo CI/CD completado:

* La imagen AMI se creó automáticamente con Packer y se desplegó sin intervención manual.
* Se verificó que las instancias existentes fueran terminadas antes de lanzar nuevas.

Instancia EC2 funcional:

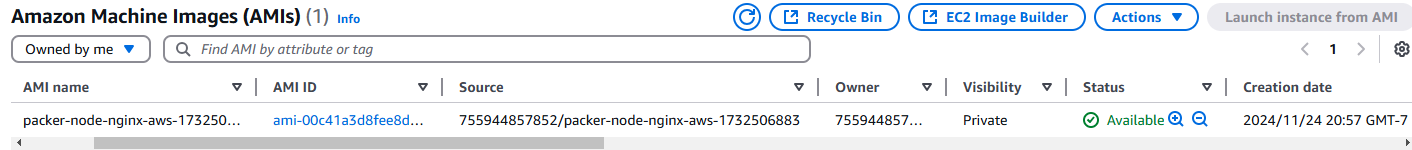
* La instancia inició con Node.js y Nginx configurados, ejecutando el servicio Node.js detrás de un proxy inverso Nginx.
* La IP pública de la instancia fue generada y proporcionada como parte del flujo.

Beneficios:

* Despliegue consistente y reproducible en AWS.
* Minimización de errores manuales.
* Automatización eficiente utilizando herramientas modernas como GitHub Actions.

A continuación, comparto imágenes de los resultados obtenidos en AWS.

AMI creada desde el workflow de GitHub Actions.



Instancia de EC2 creada desde el workflow de GitHub Actions.

A screenshot of a computer

Description automatically generated

Se accedió a la IP publica de la instancia y se puede ver que la aplicación de hello.js está funcionando correctamente desde el navegador.

A screenshot of a computer

Description automatically generated

4. Ejercicio 3: Despliegue Multinube

4.1. Justificación del Uso de Google Cloud

Google Cloud Platform (GCP) fue seleccionado como proveedor secundario para este despliegue multinube debido a las siguientes razones:

* Precio competitivo: GCP ofrece descuentos automáticos por uso prolongado y tiene precios competitivos para máquinas de baja potencia como e2-micro.
* Integración sencilla: Herramientas como gcloud y soporte nativo para Packer facilitan la creación de imágenes personalizadas y despliegues automatizados.
* Cobertura global: GCP tiene regiones en ubicaciones estratégicas que complementan las de AWS, mejorando la redundancia y disponibilidad global.
* Ecosistema robusto: Los servicios como Google Compute Engine y su red global de baja latencia lo convierten en una excelente opción para aplicaciones distribuidas.
* Usabilidad: La curva de aprendizaje de GCP es menor comparada con Azure.
* Interfaz: La interfaz de la consola de GCP es mas amigable con el usuario.

4.2. Configuración del Builder para Google Cloud

El builder de GCP en la plantilla template\_multicloud.pkr.hcl fue configurado con el siguiente bloque para crear imágenes personalizadas con Node.js, Nginx y PM2 instalados:

1. source "googlecompute" "node\_nginx\_image\_gcp" {

2. project\_id = "leafy-tuner-382618" # Cambia por tu ID de proyecto en GCP

3. zone = var.gcp\_zone

4. machine\_type = "e2-micro"

5. source\_image = "ubuntu-2404-noble-amd64-v20241115"

6. image\_name = "packer-node-nginx-gcp-{{timestamp}}"

7. ssh\_username = "ubuntu"

8. }

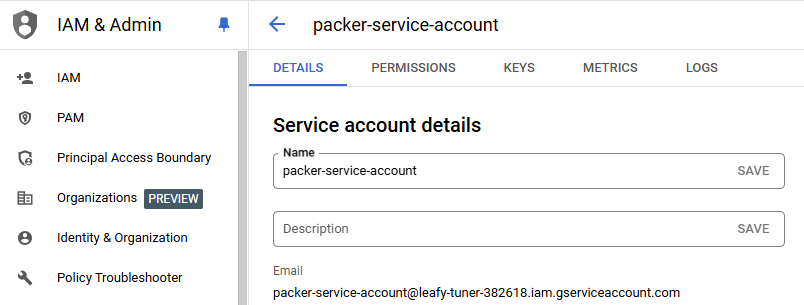
9.

Para obtener la imagen fuente para crear la nueva imagen con Packer, es necesario entrar a la consola de GCP y buscar la versión de Ubuntu que coincidiera con la misma que use en AWS. En este caso la imagen tenía el id de ubuntu-2404-noble-amd64-v20241115.

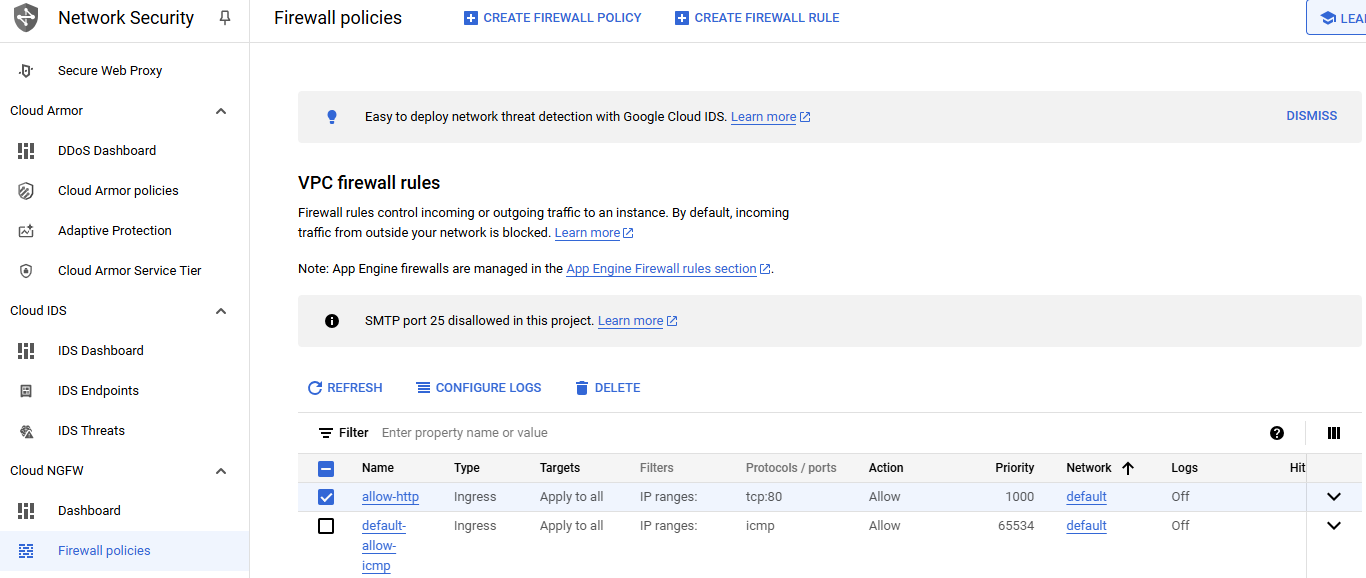
Fue necesario crear un usuario de IAM en GCP con los permisos necesarios para poder interactuar con Compute Engine para la creación de imágenes e instancias de manera automática mediante la API de GCP.

Al usuario lo nombre packer-service-account y fue necesario darle permisos adicionales mediante el siguiente comando para agregar una nueva política de iam a mi usuario packer-service-account:

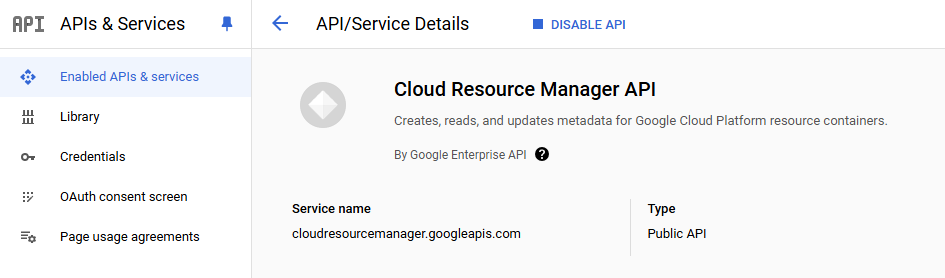
gcloud iam service-accounts add-iam-policy-binding 919631022975-compute@developer.gserviceaccount.com --member="serviceAccount:packer-service-account@<your-project-id>.iam.gserviceaccount.com" --role="roles/iam.serviceAccountUser"



Se necesita crear en Network Security, una nueva regla de firewall para permitir el acceso a las nuevas instancias por HTTP Y HTTPS. Por defecto todas las maquinas virtuales que se creen no tienen alcance a internet y es por ello por lo que es necesario crear una nueva regla. La regla puede ser configurada para abarcar solo ciertas instancias con una etiqueta en especifico o abarcar a todas las instancias creadas.



Es necesario habilitar las APIs de Cloud Resource Manager API y Compute Engine API en GCP para poder interactuar mediante consola con los servicios de Cloud Engine.



A screenshot of a computer

Description automatically generated

Después de configurar todo lo anterior se hicieron dos nuevas modificaciones. Se crearon dos nuevos templates llamados template\_multicloud.pkr.hcl y gcp.pkr.hcl para incluir la creación de imágenes en AWS y GCP con la misma configuración.

template\_multicloud.pkr.hcl

1. packer {

2. required\_plugins {

3. amazon = {

4. version = ">= 1.3.3"

5. source = "github.com/hashicorp/amazon"

6. }

7. googlecompute = {

8. version = ">= 1.1.6"

9. source = "github.com/hashicorp/googlecompute"

10. }

11. }

12. }

13.

14. variable "aws\_region" {

15. default = "us-east-1"

16. }

17.

18. variable "gcp\_region" {

19. default = "us-central1"

20. }

21.

22. variable "gcp\_zone" {

23. default = "us-central1-a"

24. }

25.

26. variable "instance\_type" {

27. default = "t2.micro"

28. }

29.

30. # Definir la variable "ami\_id" con el valor de la AMI de Ubuntu 24.04 free tier

31. variable "ami\_id" {

32. default = "ami-0866a3c8686eaeeba"

33. }

34.

35. # Referencias a los archivos específicos para AWS y GCP

36. source "amazon-ebs" "node\_nginx\_image\_aws" {

37. region = var.aws\_region

38. instance\_type = var.instance\_type

39. source\_ami = var.ami\_id

40. ssh\_username = "ubuntu"

41. ami\_name = "packer-node-nginx-aws-{{timestamp}}"

42. associate\_public\_ip\_address = true

43. }

44.

45. source "googlecompute" "node\_nginx\_image\_gcp" {

46. project\_id = "leafy-tuner-382618" # Cambia por tu ID de proyecto en GCP

47. region = var.gcp\_region

48. zone = var.gcp\_zone

49. image\_family = "packer-node-nginx-gcp"

50. image\_name = "packer-node-nginx-gcp-{{timestamp}}"

51. source\_image = "ubuntu-2404-noble-amd64-v20241115" # Imagen de Ubuntu 24.04 en GCP

52. machine\_type = "e2-micro"

53. ssh\_username = "ubuntu"

54. }

55.

56. build {

57. name = "node-nginx-image"

58. sources = [

59. "source.amazon-ebs.node\_nginx\_image\_aws",

60. "source.googlecompute.node\_nginx\_image\_gcp"

61. ]

62.

63. provisioner "shell" {

64. inline = [

65. "export DEBIAN\_FRONTEND=noninteractive",

66. "sudo apt update -y",

67. "sudo apt upgrade -y",

68. "sudo apt install -y curl gnupg software-properties-common",

69. # Instalación de Node.js, Nginx y PM2

70. "curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -",

71. "sudo apt install -y nodejs nginx",

72. "sudo npm install -g pm2",

73. # Creación de la aplicación hello.js

74. "echo 'const http = require(\"http\"); const hostname = \"localhost\";const port = 3000; const server = http.createServer((req, res) => { res.statusCode = 200; res.setHeader(\"Content-Type\", \"text/plain\"); res.end(\"Hello World!\"); }); server.listen(port, hostname, () => { console.log(`Server running at http://$$hostname:$$port/`); });' > /home/ubuntu/hello.js",

75. "sudo -u ubuntu pm2 start /home/ubuntu/hello.js",

76. "sudo -u ubuntu pm2 save",

77. # Configurar el script de inicio automáticamente

78. "sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u ubuntu --hp /home/ubuntu",

79. "sudo systemctl enable pm2-ubuntu || true",

80. "sudo systemctl restart pm2-ubuntu || true",

81. # Crear el archivo de configuración de Nginx

82. "echo 'server { listen 80; server\_name \_; location / { proxy\_pass http://localhost:3000; 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; } }' | sudo tee /etc/nginx/sites-available/default",

83. "sudo ln -sf /etc/nginx/sites-available/default /etc/nginx/sites-enabled/default",

84. "sudo nginx -t || true",

85. "sudo systemctl restart nginx || true"

86. ]

87. }

88.

89. provisioner "shell" {

90. inline = [

91. # Abrir el puerto 80 en el firewall

92. "sudo ufw allow 'Nginx Full'",

93. # Habilitar Nginx para que se inicie automáticamente

94. "sudo systemctl enable nginx"

95. ]

96. }

97. }

98.

gcp.pkr.hcl

1. source "googlecompute" "node\_nginx\_image\_gcp" {

2. project\_id = "leafy-tuner-382618" # Cambia por tu ID de proyecto en GCP

3. zone = "us-central1-a"

4. machine\_type = "e2-micro"

5. source\_image = "ubuntu-2404-noble-amd64-v20241115" # Imagen base en Google Cloud

6. image\_name = "packer-node-nginx-gcp-{{timestamp}}"

7. ssh\_username = "ubuntu"

8. metadata = {

9. "startup-script" = "#!/bin/bash

10. sudo apt-get update -y

11. sudo apt-get upgrade -y

12. sudo apt-get install -y curl gnupg software-properties-common

13. # Instalación de Node.js, Nginx y PM2

14. curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -

15. sudo apt-get install -y nodejs nginx

16. sudo npm install -g pm2

17. # Creación de la aplicación hello.js

18. echo 'const http = require(\\\"http\\\"); const hostname = \\\\\"localhost\\\\\"; const port = 3000; const server = http.createServer((req, res) => { res.statusCode = 200; res.setHeader(\\\"Content-Type\\\", \\\"text/plain\\\"); res.end(\\\"Hello World!\\\"); }); server.listen(port, hostname, () => { console.log(\\`Server running at http://$$hostname:$$port/\\`); });' > /home/ubuntu/hello.js

19. sudo -u ubuntu pm2 start /home/ubuntu/hello.js

20. sudo -u ubuntu pm2 save

21. sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u ubuntu --hp /home/ubuntu

22. sudo systemctl enable pm2-ubuntu || true

23. sudo systemctl restart pm2-ubuntu || true

24. # Configurar el archivo de configuración de Nginx

25. echo 'server { listen 80; server\_name \_; location / { proxy\_pass http://localhost:3000; 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; } }' | sudo tee /etc/nginx/sites-available/default

26. sudo ln -sf /etc/nginx/sites-available/default /etc/nginx/sites-enabled/default

27. sudo nginx -t || true

28. sudo systemctl restart nginx || true

29. # Abrir el puerto 80 en el firewall

30. sudo ufw allow 'Nginx Full'

31. # Habilitar Nginx para que se inicie automáticamente

32. sudo systemctl enable nginx"

33. }

34.

Se realizó un nuevo workflow para incluir la creación de una instancia de Compute Engine de GCP, además de incluir nuevas llaves secretas en el repositorio de GitHub para guardar las llaves de acceso de la cuenta IAM de Google Cloud. Este nuevo workflow tiene el propósito de crear de manera automática las imágenes en ambos proveedores cloud para posteriormente, una vez terminado este paso, verificar la existencia de las imágenes y proceder a aprovisionar una instancia en ambos proveedores cloud. Todo lo anterior de manera automática dentro del mismo workflow separado en diferentes pasos.

Se ejecutará cada vez que exista un nuevo push hacia la rama main con nuevos cambios a las templates de Packer.

1. name: Build and Deploy EC2 and Google Cloud VM with Packer

2.

3. on:

4.   push:

5.     branches:

6.       - main

7.

8. jobs:

9.   build-images:

10.     runs-on: ubuntu-latest

11.     environment: aws

12.

13.     steps:

14.       - name: Checkout Code

15.         uses: actions/checkout@v3

16.

17.       - name: Install Packer

18.         uses: hashicorp/setup-packer@v2

19.         with:

20.           version: latest

21.

22.       - name: Configure AWS Credentials

23.         uses: aws-actions/configure-aws-credentials@v3

24.         with:

25.           aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

26.           aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

27.           aws-region: us-east-1

28.

29.       - id: "auth"

30.         uses: "google-github-actions/auth@v2"

31.         with:

32.           credentials\_json: "${{ secrets.GCP\_SA\_KEY }}"

33.

34.       - name: Set up Google Cloud Credentials

35.         uses: google-github-actions/setup-gcloud@v2

36.         with:

37.           project\_id: ${{ secrets.GCP\_PROJECT\_ID }}

38.

39.       - name: Init Packer for Multicloud

40.         run: packer init template\_multicloud.pkr.hcl

41.

42.       - name: Build Images for AWS and Google Cloud

43.         run: packer build template\_multicloud.pkr.hcl

44.

45.   deploy-instance-google:

46.     needs: build-images

47.     runs-on: ubuntu-latest

48.     environment: aws

49.

50.     steps:

51.       - id: "auth"

52.         uses: "google-github-actions/auth@v2"

53.         with:

54.           credentials\_json: "${{ secrets.GCP\_SA\_KEY }}"

55.

56.       - name: Set up Google Cloud Credentials

57.         uses: google-github-actions/setup-gcloud@v2

58.         with:

59.           project\_id: ${{ secrets.GCP\_PROJECT\_ID }}

60.

61.       - name: Install jq

62.         run: sudo apt-get install jq -y

63.

64.       - name: Set Google Cloud Project

65.         run: gcloud config set project ${{ secrets.GCP\_PROJECT\_ID }}

66.

67.       - name: Launch Google Compute Engine Instance

68.         run: |

69.           # Obtener el nombre de la imagen más reciente creada por Packer

70.           IMAGE\_NAME=$(gcloud compute images list --filter="name~'packer-\*'" --format="value(name)" --sort-by="~creationTimestamp" | head -n 1)

71.

72.           PROJECT\_ID=$(gcloud config get-value project)

73.

74.           INSTANCE\_NAME="packer-instance-$(date +%Y%m%d)"

75.

76.           # Lanzar la nueva instancia de Google Cloud

77.           gcloud compute instances create $INSTANCE\_NAME \

78.             --image=$IMAGE\_NAME \

79.             --image-project=$PROJECT\_ID \

80.             --zone=us-central1-a \

81.             --machine-type=n1-standard-1 \

82.             --tags=http-server,https-server \

83.             --metadata=startup-script='#!/bin/bash echo "Startup script running!"' \

84.             --project=$PROJECT\_ID

85.

86.           echo "Instancia de Google Cloud $INSTANCE\_NAME creada con éxito."

87.

88.       - name: Get External IP of the Google Cloud Instance

89.         run: |

90.           INSTANCE\_NAME="packer-instance-$(date +%Y%m%d)"

91.           EXTERNAL\_IP=$(gcloud compute instances describe $INSTANCE\_NAME --zone=us-central1-a --format="get(networkInterfaces[0].accessConfigs[0].natIP)")

92.

93.           echo "La IP externa de la instancia de Google Cloud es: $EXTERNAL\_IP"

94.

95.   deploy-instance-aws:

96.     needs: build-images

97.     runs-on: ubuntu-latest

98.     environment: aws

99.

100.     steps:

101.       - name: Configure AWS Credentials

102.         uses: aws-actions/configure-aws-credentials@v3

103.         with:

104.           aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

105.           aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

106.           aws-region: us-east-1

107.

108.       - name: Install jq

109.         run: sudo apt-get install jq -y

110.

111.       - name: Check and Terminate Existing EC2 Instance

112.         run: |

113.           INSTANCE\_NAME="Packer Instance-$(date +%Y%m%d)"

114.           INSTANCE\_ID=$(aws ec2 describe-instances --filters "Name=tag:Name,Values=$INSTANCE\_NAME" --query "Reservations[0].Instances[0].InstanceId" --output text)

115.

116.           if [ "$INSTANCE\_ID" != "None" ]; then

117.             echo "La instancia con el nombre '$INSTANCE\_NAME' ya existe, deteniendo y eliminando..."

118.             # Detener y eliminar la instancia existente

119.             aws ec2 terminate-instances --instance-ids $INSTANCE\_ID

120.             aws ec2 wait instance-terminated --instance-ids $INSTANCE\_ID

121.             echo "Instancia terminada y eliminada."

122.           else

123.             echo "No se encontró una instancia existente con el nombre '$INSTANCE\_NAME'. Lanzando una nueva..."

124.           fi

125.

126.       - name: Launch EC2 Instance

127.         run: |

128.           # Obtener la ID de la imagen de AWS

129.           AMI\_ID=$(aws ec2 describe-images --filters "Name=name,Values=packer-node-nginx-\*" --query "Images | sort\_by(@, &CreationDate)[-1].ImageId" --output text)

130.           DEFAULT\_SUBNET=$(aws ec2 describe-subnets --filters "Name=default-for-az,Values=true" --query "Subnets[0].SubnetId" --output text)

131.

132.           INSTANCE\_NAME="Packer Instance-$(date +%Y%m%d)"  # Nombre único para la nueva instancia

133.

134.           # Lanzar la nueva instancia EC2 con el nombre único

135.           INSTANCE\_DETAILS=$(aws ec2 run-instances --image-id $AMI\_ID --instance-type t2.micro --key-name ${{ secrets.EC2KEY }} --security-group-ids ${{ secrets.SG\_PACKER }} --subnet-id $DEFAULT\_SUBNET --region us-east-1 --associate-public-ip-address --tag-specifications "ResourceType=instance,Tags=[{Key=Name,Value=$INSTANCE\_NAME}]" --output json)

136.

137.           # Obtener el ID de la instancia

138.           INSTANCE\_ID=$(echo $INSTANCE\_DETAILS | jq -r '.Instances[0].InstanceId')

139.           echo "La instancia EC2 ha sido lanzada con ID: $INSTANCE\_ID"

140.

141.           # Esperar a que la instancia esté en estado 'running'

142.           aws ec2 wait instance-running --instance-ids $INSTANCE\_ID

143.           echo "La instancia está en estado 'running'."

144.

145.           # Obtener la IP pública de la instancia

146.           PUBLIC\_IP=$(aws ec2 describe-instances --instance-ids $INSTANCE\_ID --query "Reservations[0].Instances[0].PublicIpAddress" --output text)

147.

148.           echo "La IP pública de la instancia EC2 es: $PUBLIC\_IP"

149.

4.3. Resultados Obtenidos

Despliegue Automático

Los workflows de GitHub Actions, definidos en packer\_multicloud.yml, desplegaron instancias automáticamente en ambos proveedores.

* En AWS: Usando la AMI más reciente creada por Packer, con seguridad configurada mediante grupos de seguridad.
* En GCP: Usando la imagen personalizada más reciente, configurada para iniciar automáticamente el servidor Node.js con Nginx como proxy inverso.

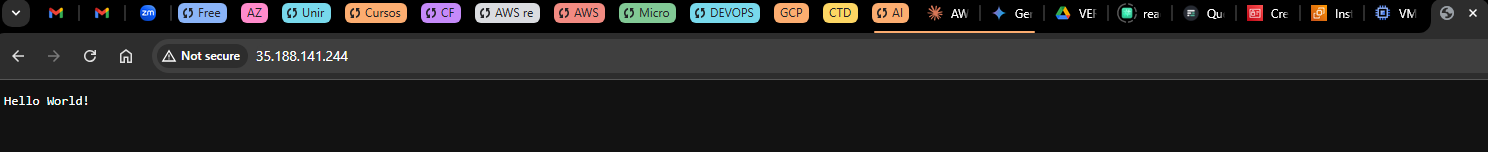
GCP

A screenshot of a computer

Description automatically generated

A screenshot of a computer

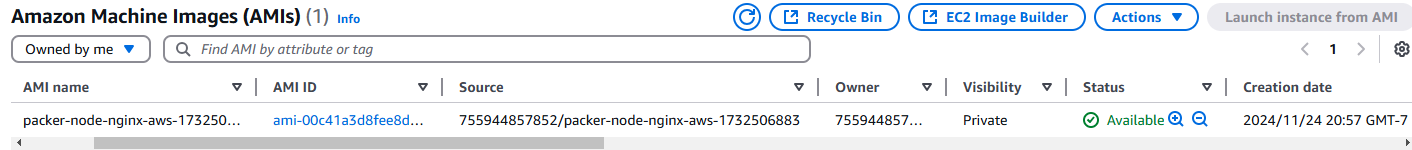
Description automatically generated



A screenshot of a computer

Description automatically generated

AWS



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

5. Conclusión y Lecciones Aprendidas

En este trabajo, se logró integrar y automatizar procesos clave para la configuración y despliegue de aplicaciones mediante herramientas modernas como Packer, Node.js, Nginx y plataformas en la nube como AWS y GCP. El uso de Packer permitió crear imágenes reutilizables y consistentes, optimizando el tiempo y recursos necesarios para gestionar entornos de despliegue. Además, la implementación de un despliegue automatizado garantizó un proceso eficiente y sin intervención manual, cumpliendo con los estándares de escalabilidad y portabilidad requeridos en entornos multinube.

Esta experiencia reafirma la importancia de las prácticas de DevOps en proyectos tecnológicos actuales, donde la automatización y la interoperabilidad entre servicios juegan un papel fundamental en la entrega continua de valor. En conclusión, se estableció una base sólida para el manejo eficiente de infraestructuras modernas y se demostró la viabilidad de implementar soluciones multinube con herramientas confiables y escalables.

Aunque fue necesaria una configuración inicial de usuarios y otras configuraciones de seguridad, el resto de las actividades se pudieron hacer de manera automática y cada cambie nuevo al template de la rama main se verá actualizado y desplegado en los diferentes entornos configurados sin ninguna configuración extra. Solamente será necesario modificar el template almacenado en el repositorio, lo cual nos permitirá tener un control de versiones de nuestro template y de esta manera tener un mayor control sobre lo que se desplegará. Sin duda Packer es una herramienta muy potente que permite reducir los errores humanos y reducir los tiempos de creación de recursos en gran medida, todo esto gracias a la automatización, uno de los pilares de DevOps.

Aprendí mucho al realizar este proyecto y creo que me ayudó bastante mi experiencia previa con servicios que ofrecen los proveedores cloud, me parece que si tiene un grado de dificultad intermedio por todas las configuraciones que se deben realizar en diferentes servicios, todas las dependencias que se deben instalar para realizar la actividad y la gran cantidad de recursos sobre los cuales se tiene que buscar información para poder cumplir con cada paso de las actividades plasmadas. En conclusión, esta actividad me permitió adquirir una gran cantidad de conocimientos nuevos en un corto lapso de tiempo así como reforzar otros conocimientos que no tenia muy afianzados.

REPOSITORIO DE GITHUB:

<https://github.com/Lordbear117/packer-image-template>