TRABAJO DE GRADO
Seminario
Implementación de Arquitectura en AWS con Balanceador de Carga y Contenedores

Corporación Universitaria Remington
Facultad de Ingenierías
Ingeniería en Sistemas

Integrantes:
Elquin Rodrigo Peña Pinilla
Jhon Jarbinson Soto Jimenez
Juan Esteban Giraldo Troncoso

Docente
Juan Pablo Berrio Lopez

Opción de Trabajo de grado Seminario-Diplomado
Amazon Web Services AWS
2025

Tabla de Contenidos

Resumen

Para la primera entrega en el seminario se abordaron tres (3) conceptos fundamentales de Amazon Web Services (AWS): la plataforma, las Virtual Private Clouds (VPC) y las instancias EC2. Debido al avance tecnológico, este ha dado paso a nuevas formas de administrar recursos computacionales, entre las innovaciones más significativas es la computación en la nube, la cual permite el acceso remoto y escalable a infraestructura de tecnología.

Se diseñó, desplegó y documentó una red en AWS que incluye dos instancias EC2 (una Windows y una Linux), asegurando su accesibilidad pública, conectividad entre ellas y la instalación de un servidor web funcional en cada instancia.

Se creo una red virtual de AWS en los cuales se configuraron dos datacenters compuestos por dos subredes, una (1) privada y una (1) pública. Se habilitaron una IP pública que permita el acceso desde equipos locales, protegiendo el ingreso a los servidores por medio de reglas que habiliten puertos previamente configurados en las instancias creadas de Windows y Linux.

Para esta segunda entrega del proyecto, se llevó a cabo la implementación de una arquitectura en la nube usando Amazon Web Services (AWS) para la plataforma RápidoYa, esta es una empresa emergente dedicada a conectar restaurantes con clientes mediante entregas rápidas. El objetivo principal fue garantizar la alta disponibilidad, escalabilidad automática y eficiencia en el manejo del tráfico de esta.

Para lograrlo, se configuró un Application Load Balancer (ALB) encargado de distribuir las solicitudes entre múltiples instancias EC2 ubicadas en diferentes zonas de disponibilidad. Cada

instancia fue equipada con un proxy reverso utilizando Nginx y una aplicación de prueba ejecutada dentro de un contenedor Docker.

También se implementó un grupo de Auto Scaling con políticas basadas en el uso del CPU, lo que permite ajustar dinámicamente la cantidad de instancias según la carga del sistema. Además, se realizaron pruebas de funcionamiento, balanceo de carga, recuperación ante fallos y escalado automático, obteniendo resultados satisfactorios.

Con esta entrega se demuestra la viabilidad técnica de una solución moderna y robusta para responder al crecimiento de la plataforma, dejando la base lista para futuras mejoras y una posible automatización completa del despliegue.

Palabras clave

Arquitectura en la nube, AWS, Balanceador de carga, EC2, Autoescalado, Docker, Proxy reverso, Nginx, Alta disponibilidad, Escalabilidad, Infraestructura como servicio, Startup tecnológica, Plataforma de domicilios, Contenedores, Aplicaciones distribuidas, Amazon Web Services, Implementación práctica, Cloud computing, Ingeniería de sistemas, Tecnología en la nube.

Marco conceptual

En la actualidad, el desarrollo de soluciones tecnológicas modernas requiere de infraestructuras que sean flexibles, seguras y fáciles de escalar. Dentro de este contexto, la **computación en la nube** se ha convertido en una de las herramientas más potentes para empresas y desarrolladores, permitiendo desplegar y administrar servicios sin necesidad de contar con servidores físicos propios.

Amazon Web Services (AWS) es una de las plataformas más completas en este campo. Ofrece una variedad de servicios que facilitan desde la creación de redes virtuales hasta la implementación de aplicaciones distribuidas. Uno de los primeros pasos en este tipo de soluciones es la configuración de la **VPC (Virtual Private Cloud)**, que permite organizar los recursos dentro de una red lógica en la nube. Dentro de esta red, es común usar unas **instancias EC2**, que funcionan como servidores virtuales y pueden ejecutar aplicaciones tanto en entornos Windows como Linux.

Cuando se trata de aplicaciones que deben estar disponibles para muchos usuarios y manejar distintas cargas de tráfico, es clave contar con un **balanceador de carga (Application Load Balancer)**. Este se encarga de distribuir las solicitudes entrantes de forma eficiente entre diferentes servidores, evitando cuellos de botella. A su vez, el uso de herramientas como **Auto Scaling** permite que el sistema aumente o disminuya automáticamente el número de servidores en función del uso del CPU u otros criterios definidos.

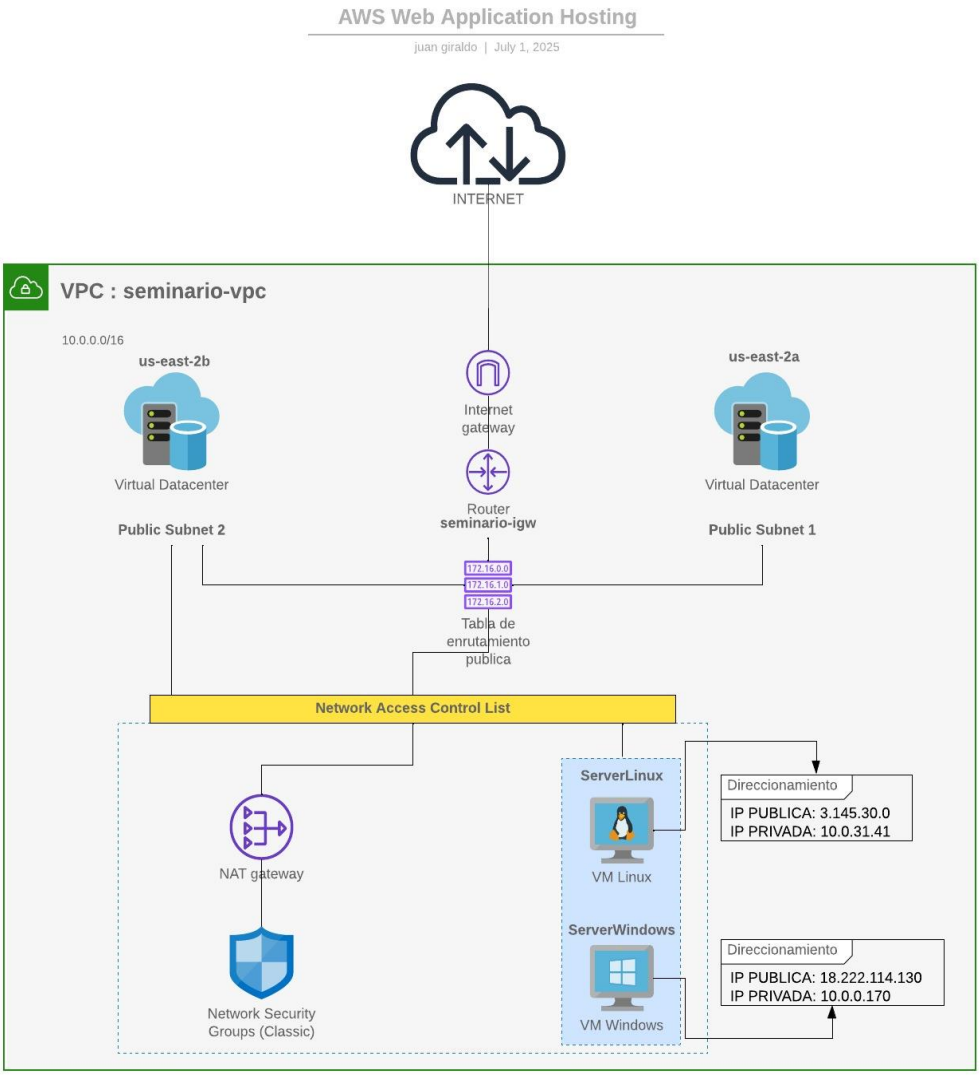
Por otro lado, tecnologías como **Docker** y **Nginx** permiten simplificar el despliegue de aplicaciones, ya que Docker encapsula todo lo necesario para que una aplicación funcione (código, librerías, configuración), mientras que *Nginx* se encarga de manejar el tráfico como un **proxy reverso**, mejorando el rendimiento y la seguridad.

En conjunto, estos conceptos permiten crear arquitecturas robustas, como la desarrollada para la plataforma RápidoYa, que combinan flexibilidad, rendimiento y capacidad de crecimiento, adaptándose a las necesidades de un startup tecnológico que requiere estabilidad desde sus primeras etapas.

Desarrollo e implementación del aprendizaje

Diagrama de Arquitectura

Figura 1



Fuente: Creación propia en Lucid.app

Breve explicación de la red creada

Contexto de la red

Para esta red recreamos una VPC (red virtual de AWS) que nombramos "*seminario-vpc*" para la cual configuramos dos data centers cada uno compuesto por 2 Sub-Redes (1 privada / 1

publica), en las subredes públicas (para permitir su acceso desde internet mediante el igw) creamos 1 instancia EC2 (Servidor/Máquina Virtual en la Nube) Windows que llamamos ServerWindows y otra instancia EC2 de Linux que llamamos ServerLinux, le habilitamos una IP Pública para permitir el acceso desde nuestro equipos locales, protegemos el acceso a los servidores mediante reglas que habilitan puertos previamente configurados en Windows RDP (Puerto 3389) y Linux SSH (Puerto 22) y para el acceso a los Servidores HTTP (Puerto 80).

Tipo de instancias usadas (Linux: Amazon Linux, Ubuntu, etc. | Windows: Versión de Windows Server)

- *AMI de Windows Server 2016 Base. (Apto para la capa gratuita)*
- *AMI de (Imagen de Sistema Operativo) Amazon Linux 2023. (Apto para la capa gratuita)*

Justificación de las configuraciones de red (por ejemplo, uso de VPC, subredes públicas, Internet Gateway)

VPC: Usamos este recurso el cual nos permite crear y personalizar una Red Virtual basadas en nuestros requerimientos.

Subredes públicas: Usamos Una Subred publica para alojar nuestras Máquinas Virtuales para así permitir su acceso desde internet mediante el uso del IGW (Internet Gateway/Router Virtual) y las tablas de enrutamiento.

Internet Gateway: Es el recurso del cual están provistas nuestras Sub-Redes Públicas el cual permite acceder a las máquinas virtuales desde nuestros equipos locales.

AMI usadas son las imágenes del sistema operativo que usamos para crear nuestras Instancias (Linux y Windows) en nuestro caso usamos las aptas para la capa gratuita para así no generar cobros en nuestro proceso de aprendizaje.

Configuraciones realizadas

En AWS, una instancia se refiere a una máquina virtual en ejecución, proporcionada por el servicio **Amazon EC2 (Elastic Compute Cloud)**. Estas instancias permiten ejecutar aplicaciones y sistemas operativos tal como se haría en un servidor físico (Kavis, 2014).

Las instancias EC2 se pueden configurar en cuanto a:

- Tipo de sistema operativo. Windows Server 2016 Base
- Memoria y CPU asignada. Intel Xeon, 1GB Ram
- Almacenamiento y redes. 30 GB SSD, ID VPC, IP Publica, conexión WEB puerto 80
- Acceso remoto mediante claves SSH.

Las instancias son fundamentales en cualquier arquitectura de nube, ya que permiten ejecutar servicios web, entornos de desarrollo, aplicaciones móviles, entre otros.

Pasos para crear las instancias EC2

Proceso Creación VPC: Una VPC (Virtual Private Cloud) es una red virtual privada dentro del entorno de AWS. Permite al usuario definir un entorno de red personalizado donde se pueden lanzar recursos como instancias, bases de datos o balanceadores de carga (Amazon Web Services, 2024).

En una VPC, el usuario tiene control total sobre:

- Rango de direcciones IP privadas.
- Subredes públicas y privadas.
- Tablas de enrutamiento.
- Reglas de firewall mediante grupos de seguridad y listas de control de acceso.

Esto permite aislar recursos críticos y proteger el tráfico interno frente a amenazas externas, manteniendo el principio de seguridad por diseño.

Figura 2

Fuente: Creación VPC en AWS

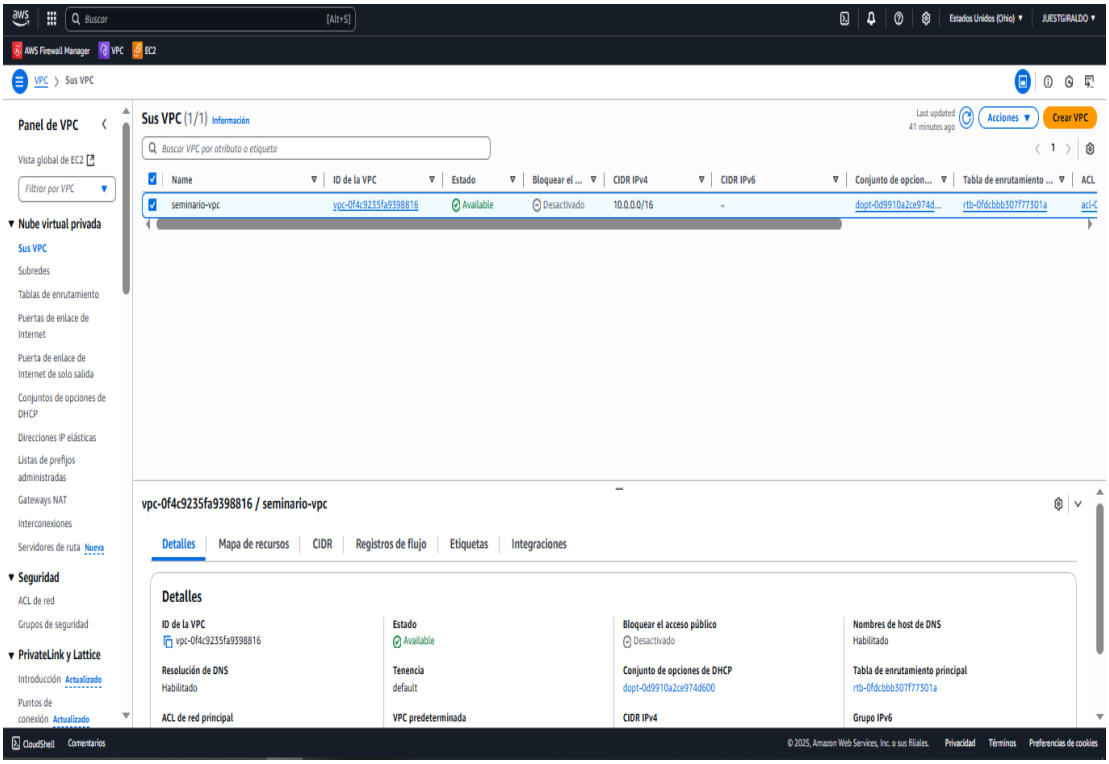
Figura 3

Fuente: Detalles creación VPC en AWS

Figura 4

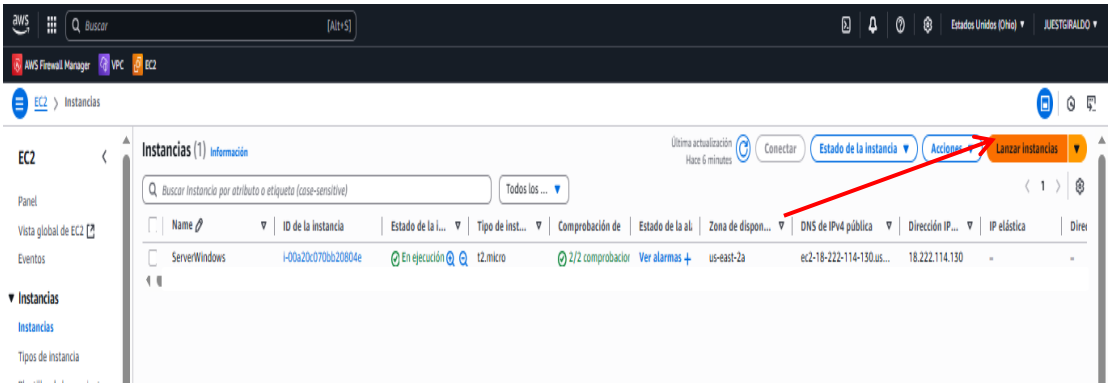
Fuente: Detalles creación VPC en AWS

Figura 5



Fuente: Pantalla en la que se visualiza la VPC creada y disponible en AWS

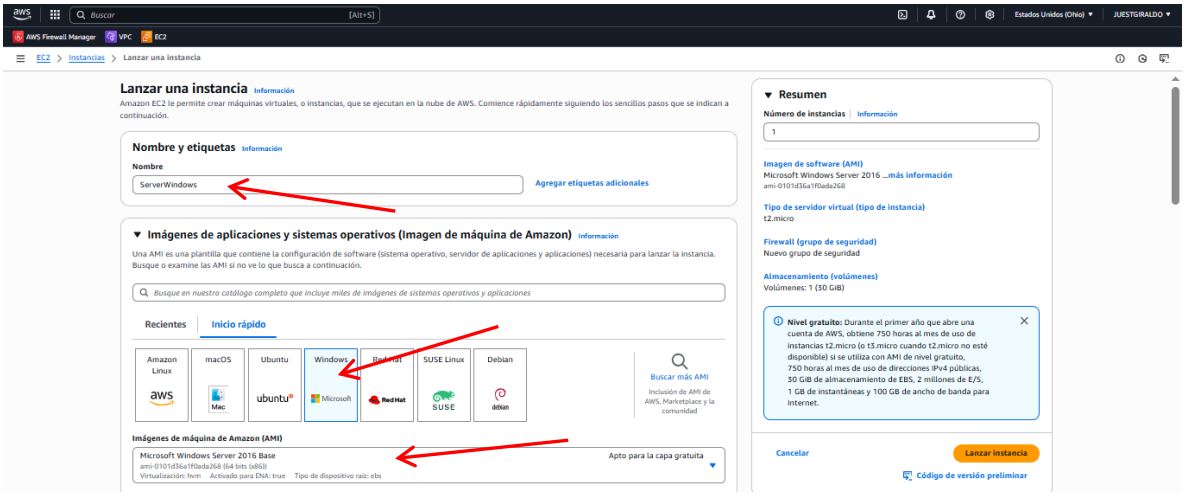
Figura 6



Fuente: Inicio creación Instancias Windows en AWS

Iniciamos dando clic en Lanzar Instancias y asignamos el nombre Server Windows y elegimos la AMI que queremos usar en este caso Microsoft Windows Server 2016 Base – Apto para la capa gratuita.

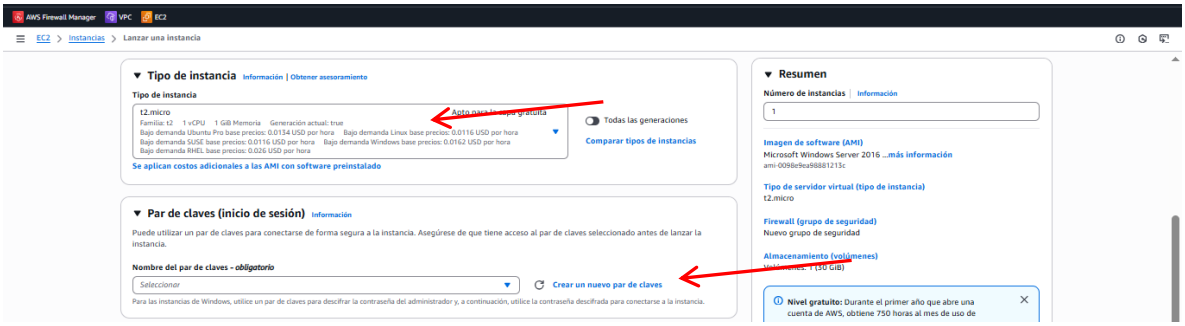
Figura 7



Fuente: Asignación de nombre a la Instancias Windows y selección de AMI en AWS

Seleccionamos el tipo de Instancia (T2) y creamos el Par de claves (archivo que debemos guardar el cual nos permitirá acceder a la instancia de forma segura.

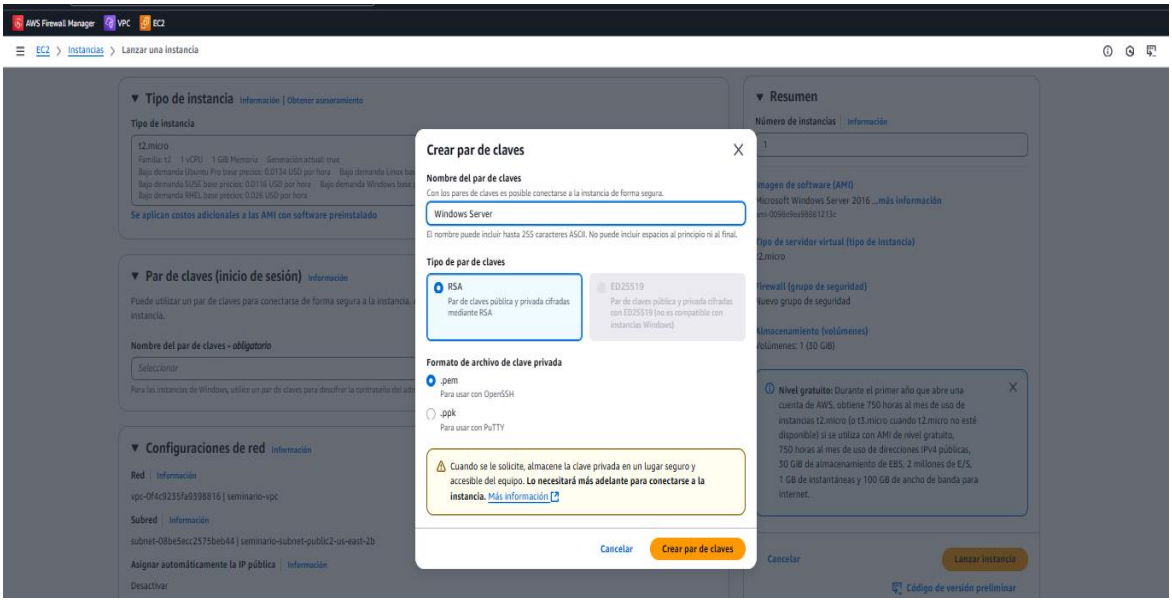
Figura 8



Fuente: Selección tipo de instancia y creación del par de claves en AWS

Asignamos un nombre al par de claves a crear (Windows Server) y damos clic en crear par de claves y guardamos el archivo que se genera en un lugar seguro pues lo necesitaremos próximamente.

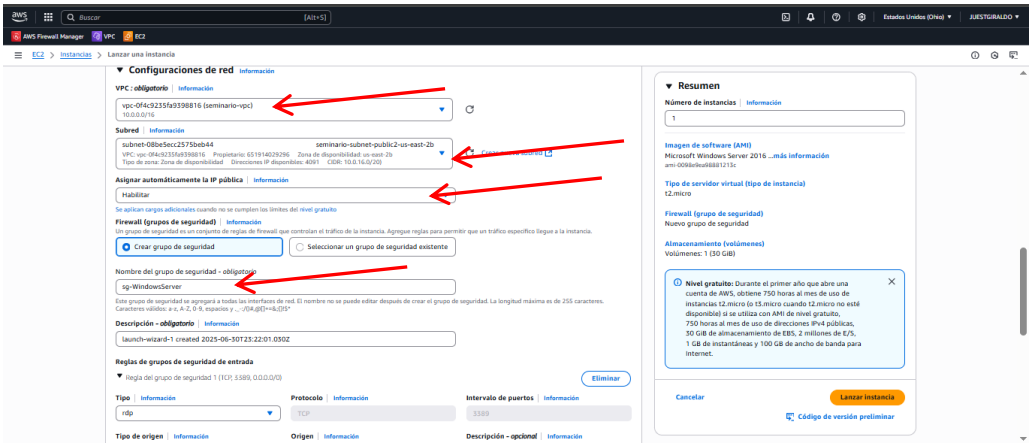
Figura 9



Fuente: Creación par de claves en AWS

Configuramos la red seleccionamos nuestro VPC ***Seminario-vpc***, seleccionamos una subred pública y habilitamos la IP pública para hacer accesible nuestro servidor desde nuestro equipo local y asignamos el nombre ***sg-WindowsServer*** a nuestro grupo de seguridad.

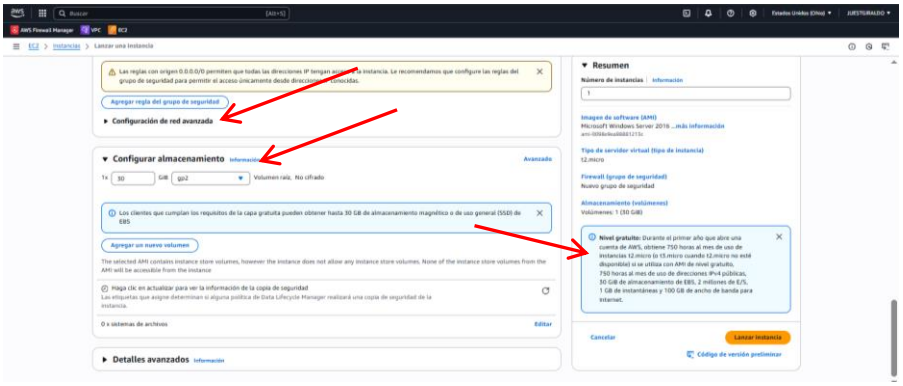
Figura 10



Fuente: Configuración de red en AWS

Configuramos el almacenamiento ***gp2*** con 30 gigas que nos provee AWS en la capa gratuita y finalizamos dando clic en lanzar instancia.

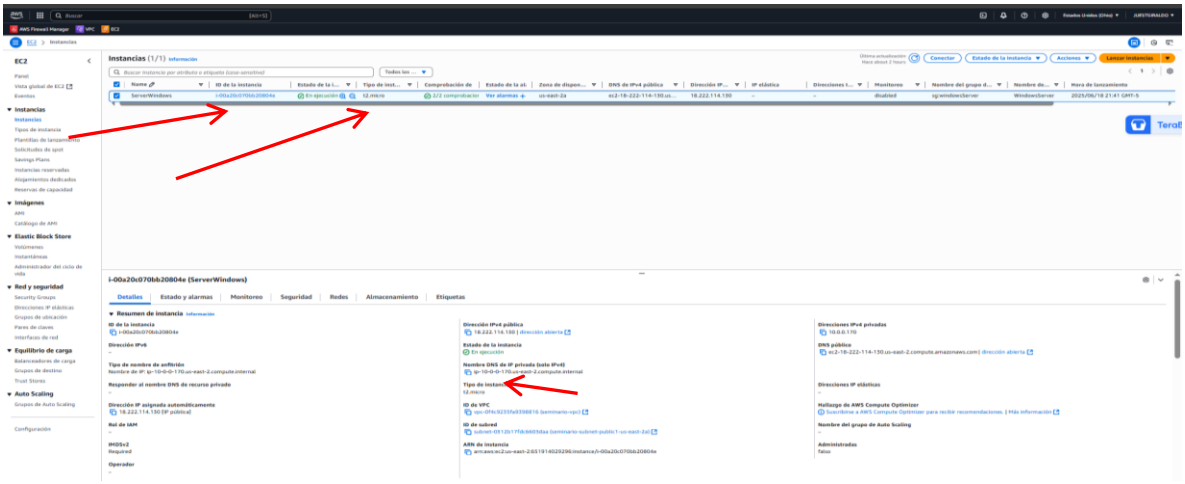
Figura 11



Fuente: Configuración de almacenamiento en AWS

Verificamos que la instancia ya está en ejecución, comprobación de estado ok y contamos con una IP Pública.

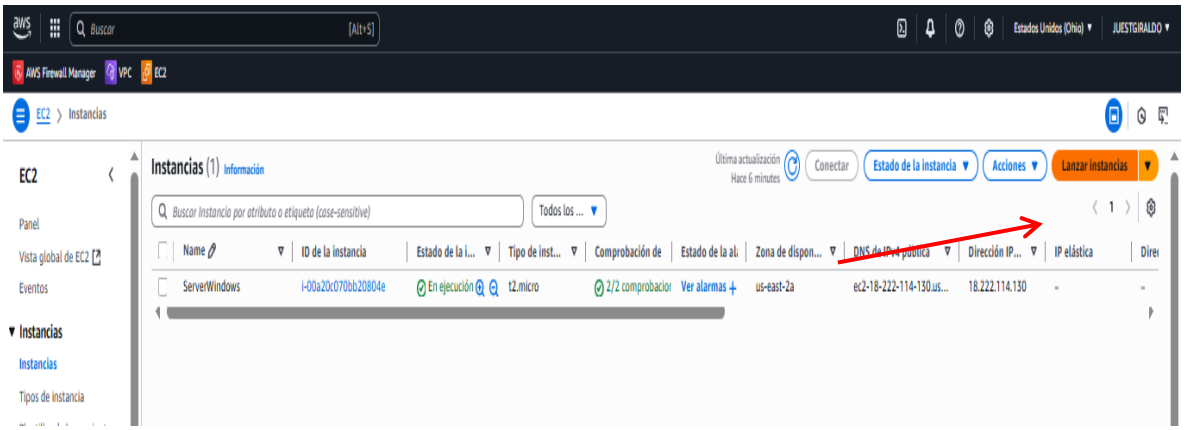
Figura 12



Fuente: Instancia creada, comprobada y en ejecución en AWS

Creación de las Instancias

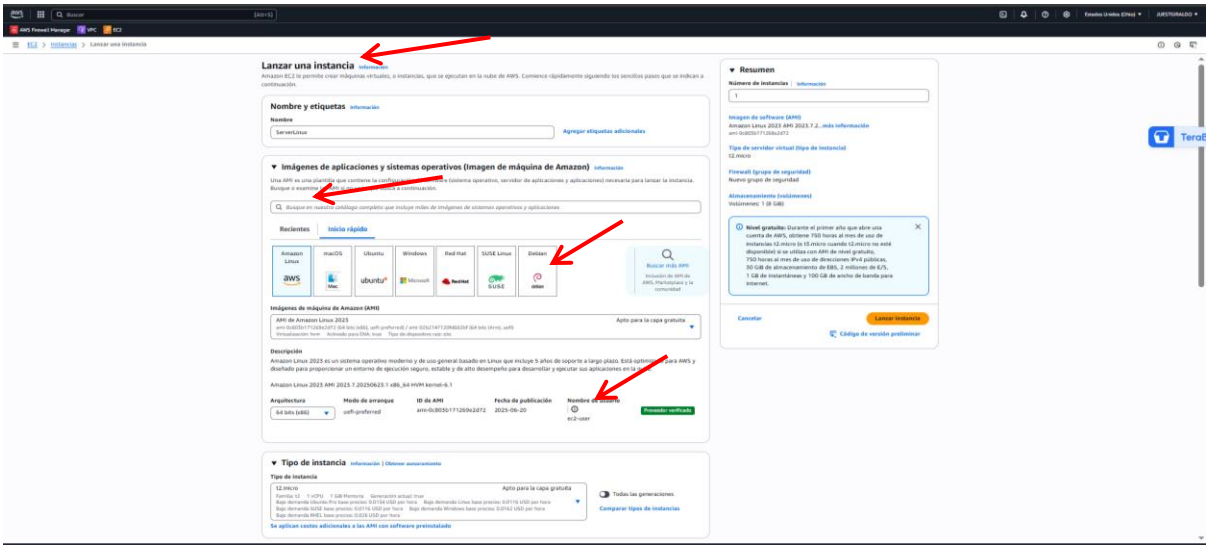
Figura 13



Fuente: Creación de la instancia en AWS

Le asignamos el nombre ***Server Linux*** y elegimos la AMI que queremos usar en este caso AMI de Amazon Linux 2023 – Apto para la capa gratuita, verificamos el tipo de Instancia (T2).

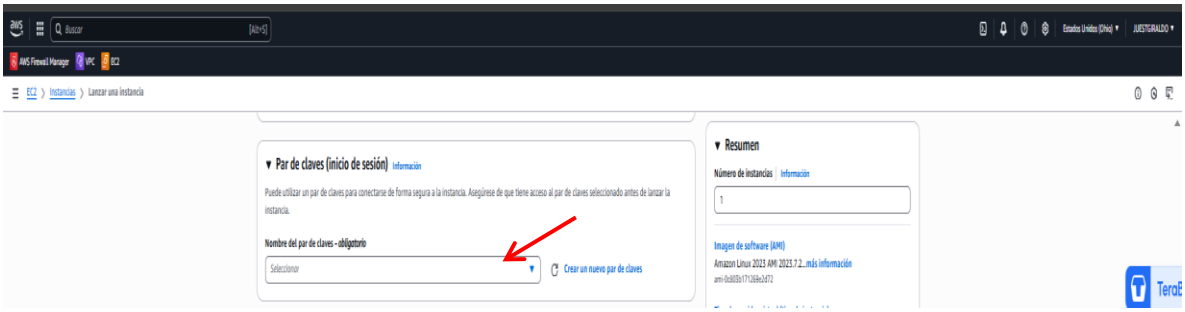
Figura 14



Fuente: Asignación de nombre a la Instancias Linux y selección de AMI en AWS

Creamos el Par de claves (archivo que debemos guardar el cual nos permitirá acceder a la instancia de forma segura) dando clic en crear un nuevo par de claves.

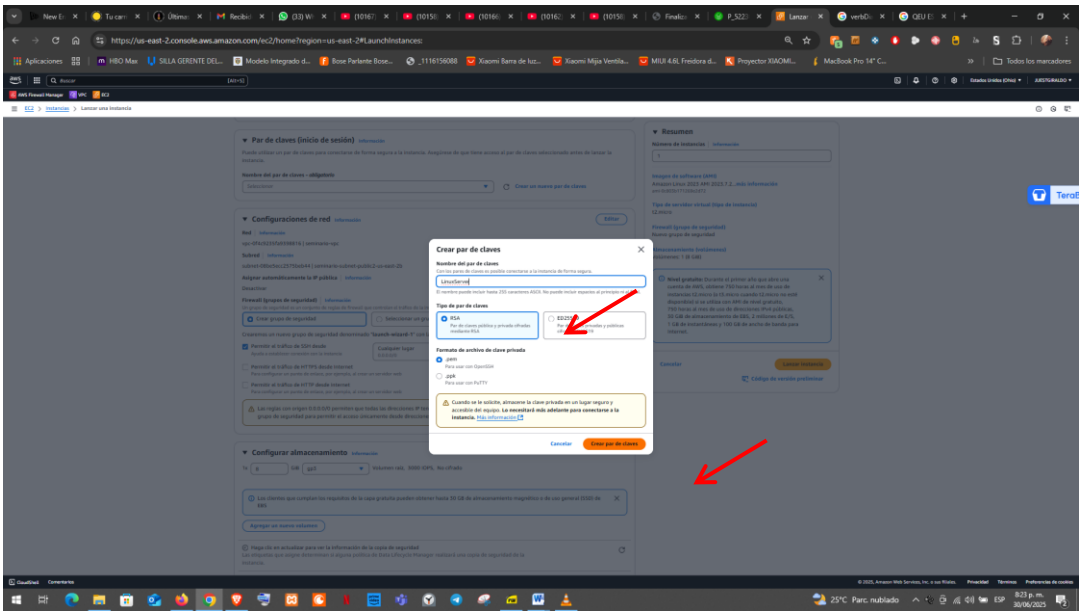
Figura 15



Fuente: Creación de par de claves para el server Linux en AWS

Asignamos un nombre al par de claves a crear (Linux Server) y damos clic en crear par de claves y guardamos el archivo que se genera en un lugar seguro pues lo necesitaremos próximamente.

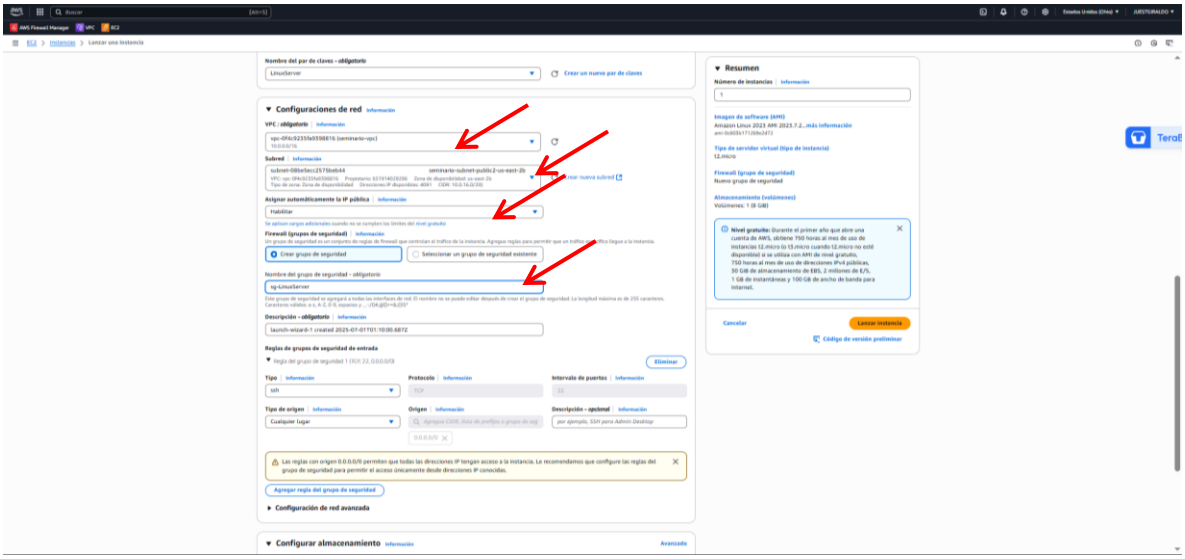
Figura 16



Fuente: Creación par de claves en AWS

Configuramos la red seleccionamos nuestro VPC ***Seminario-vpc***, seleccionamos una subred pública y habilitamos la IP publica para hacer accesible nuestro servidor desde nuestro equipo local y asignamos el nombre ***sg-LinuxServer*** a nuestro grupo de seguridad.

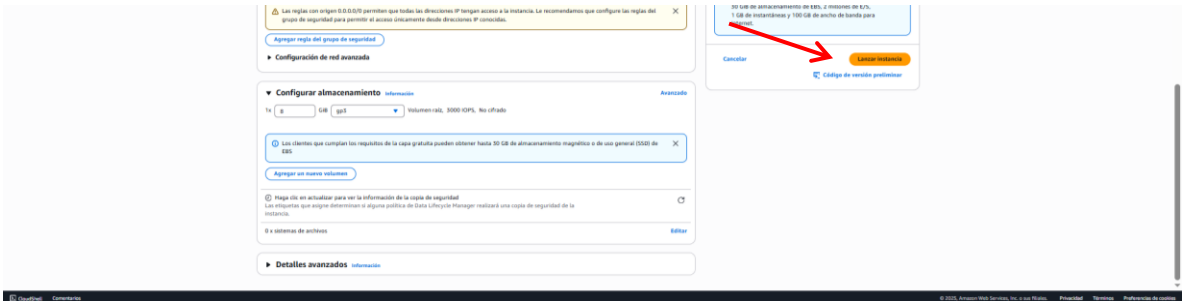
Figura 17



Fuente: Configuración de Red en AWS

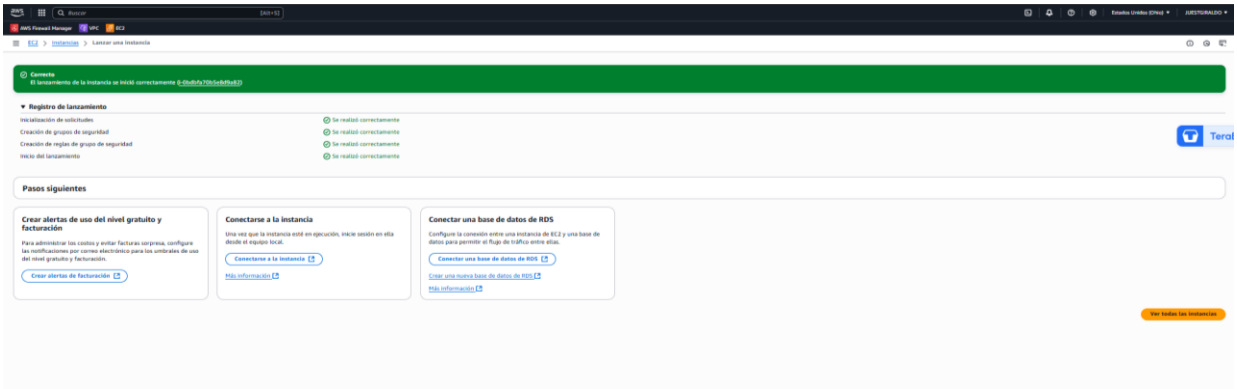
Verificamos la configuramos el almacenamiento ***gp3*** con 8 gigas que nos provee AWS en la capa gratuita y finalizamos dando clic en lanzar instancia.

Figura 18



Fuente: Verificación de almacenamiento asignado y lanzamiento de instancia en AWS.

Figura 19



Fuente: Lanzamiento de la instancia CORRECTO en AWS

Lanzamiento Correcto instancia ***LinuxServer***.

Detalles de los Grupos de Seguridad (puertos abiertos: RDP, SSH, HTTP)

INSTANCIA WINDOWS

Creamos un grupo de seguridad llamado ***sg-WindowsServer*** que por defecto tiene la regla de acceso para el protocolo ***TCP*** con el puerto ***3389*** y usando ***RDP*** (Protocolo de Escritorio Remoto) para permitirnos su admiración.

INSTANCIA LINUX

En el caso de Linux usamos la consola de comandos ***SSH*** y el puerto ***22***.

Asignación de IPs públicas y privadas

En el caso de estas Instancia que necesitamos sean accesible desde fuera de la red de AWS habilitamos la IP Pública en la configuración de las instancias.

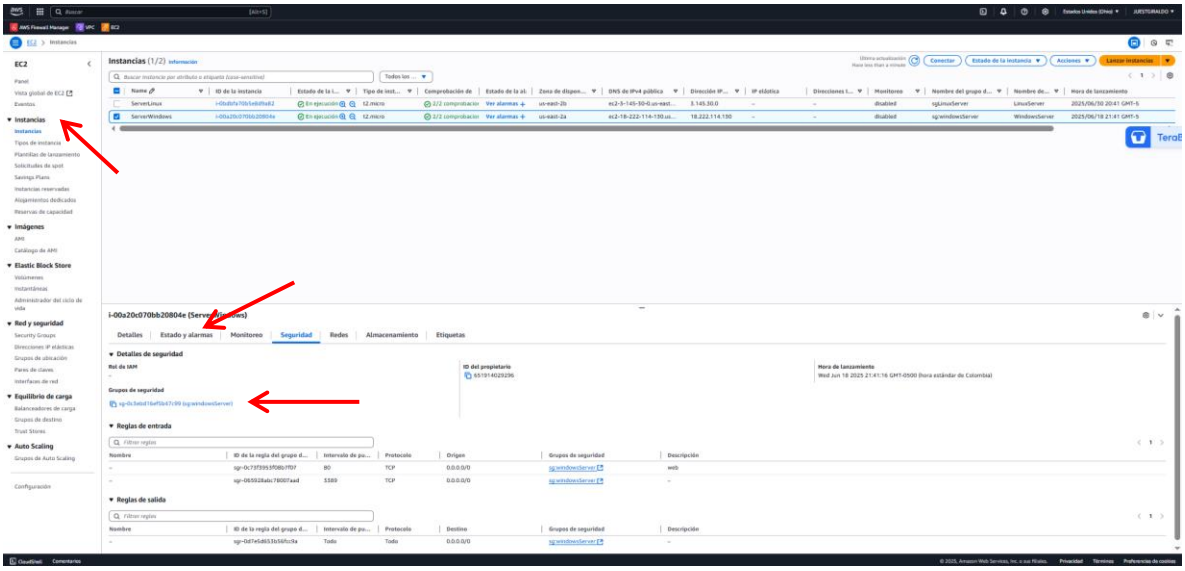
ServerLinux: IP Pública: **3.145.30.0** / IP Privada: **10.0.31.41**.

ServerWindows: IP Pública: **18.222.114.130** / IP Privada: **10.0.0.170**.

Procedimiento de acceso

Cómo acceder a cada servidor (cliente RDP para Windows, SSH para Linux)

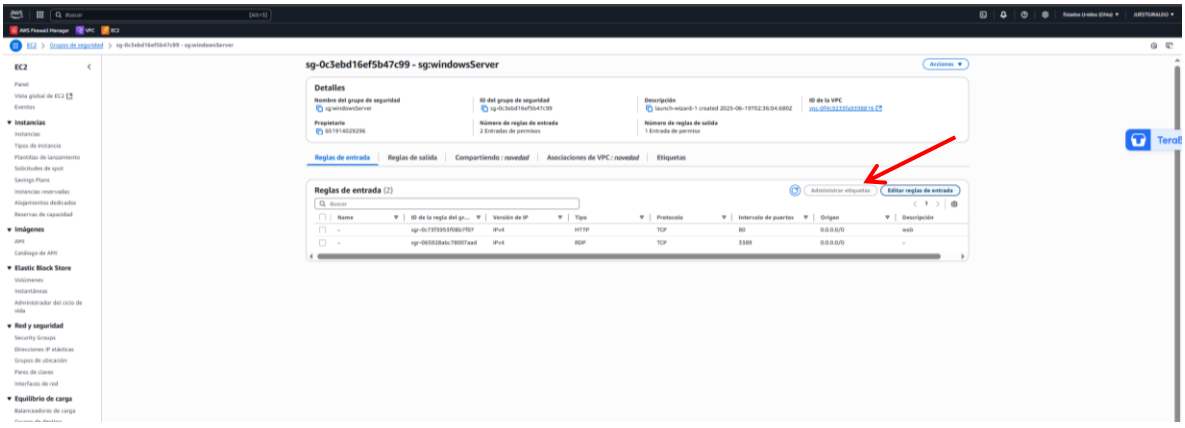
Figura 20



Fuente: Agregar Puerto 80 AL FIREWALL en AWS

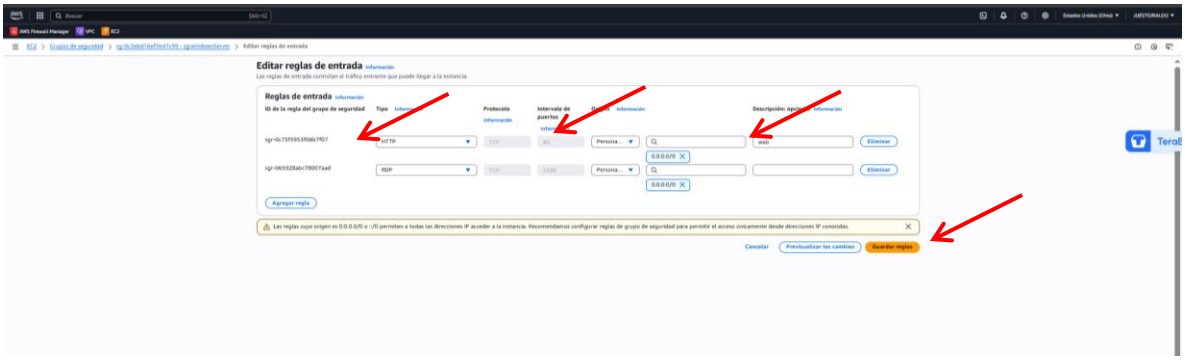
Inicialmente debemos Agregar el puerto 80 al firewall de AWS para permitir el acceso desde nuestro equipo local (pasos aplicados a ambas instancias Windows y Linux).

Figura 21



Fuente: Pasos para agregar el puerto 80 en AWS

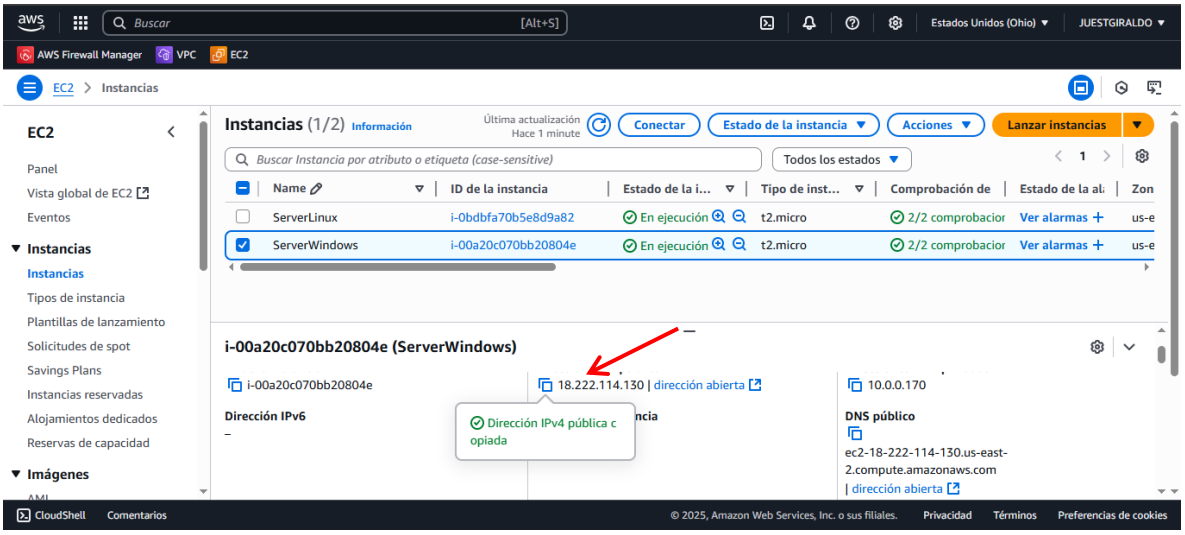
Figura 22



Fuente: Pasos para agregar el puerto en AWS

Acceso A Servidor Windows

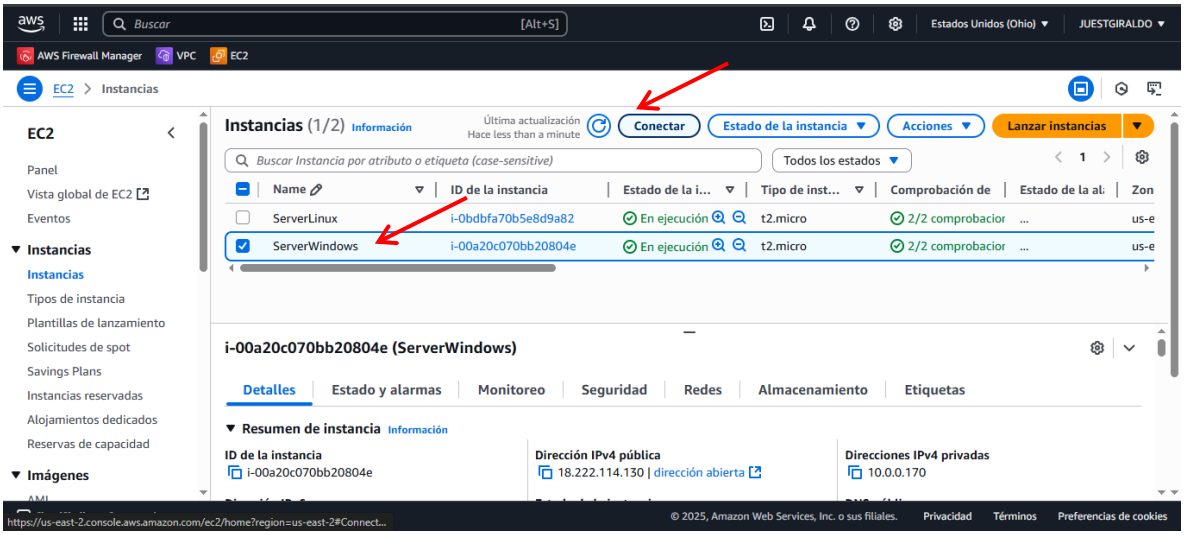
Figura 23



Fuente: Obtenemos la IP Publica (18.222.114.130) en AWS

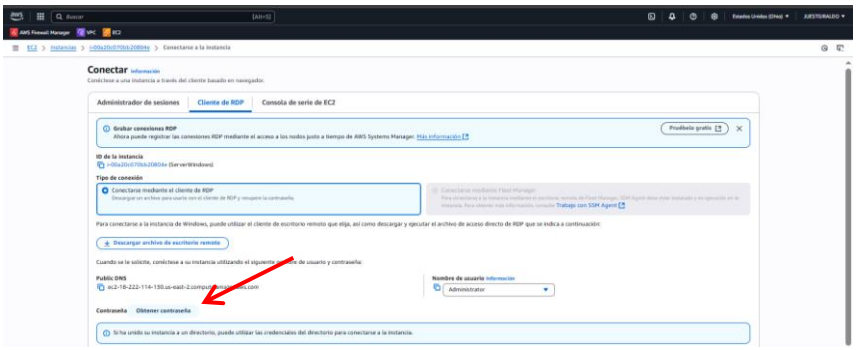
Obtenemos la contraseña de nuestro archivo ***WindowsServer.pem***.

Figura 24



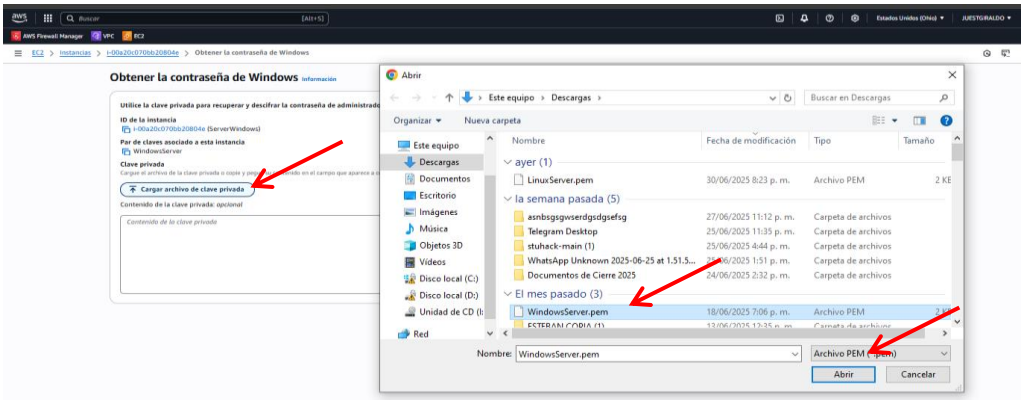
Fuente: Obtención de Contraseña para RDP en AWS

Figura 25



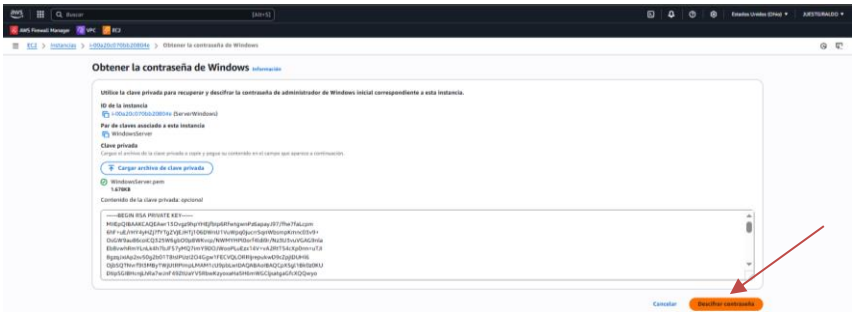
Fuente: Obtención de Contraseña para RDP en AWS

Figura 26



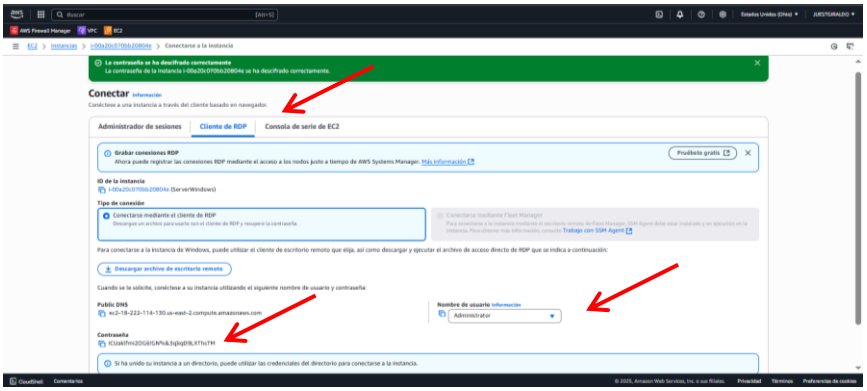
Fuente: Obtención de Contraseña para RDP en AWS

Figura 27



Fuente: Obtención de Contraseña para RDP en AWS

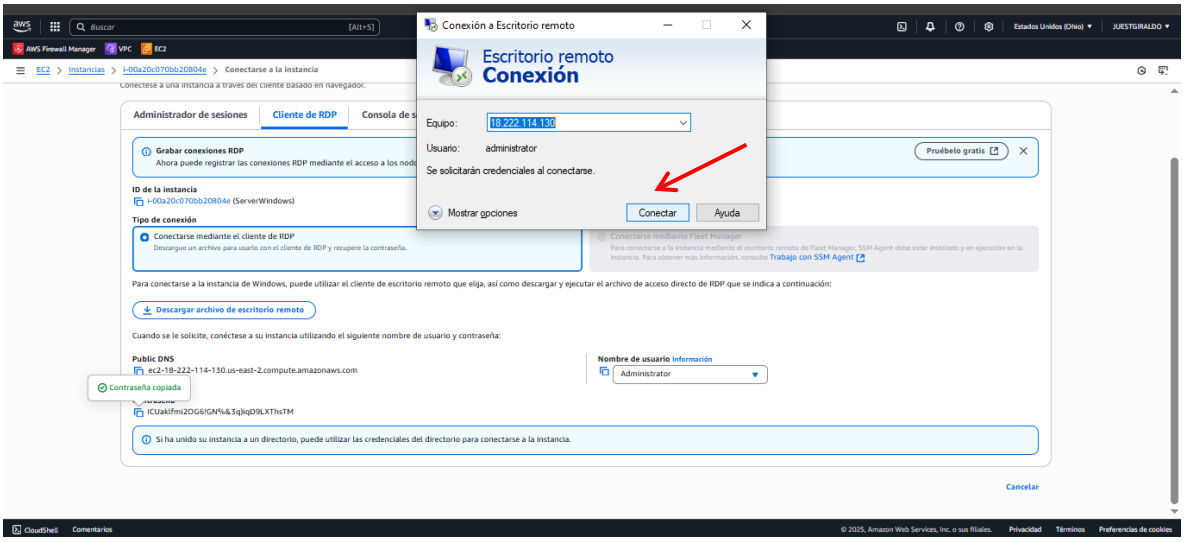
Figura 28



Fuente: Obtención de Contraseña para RDP en AWS

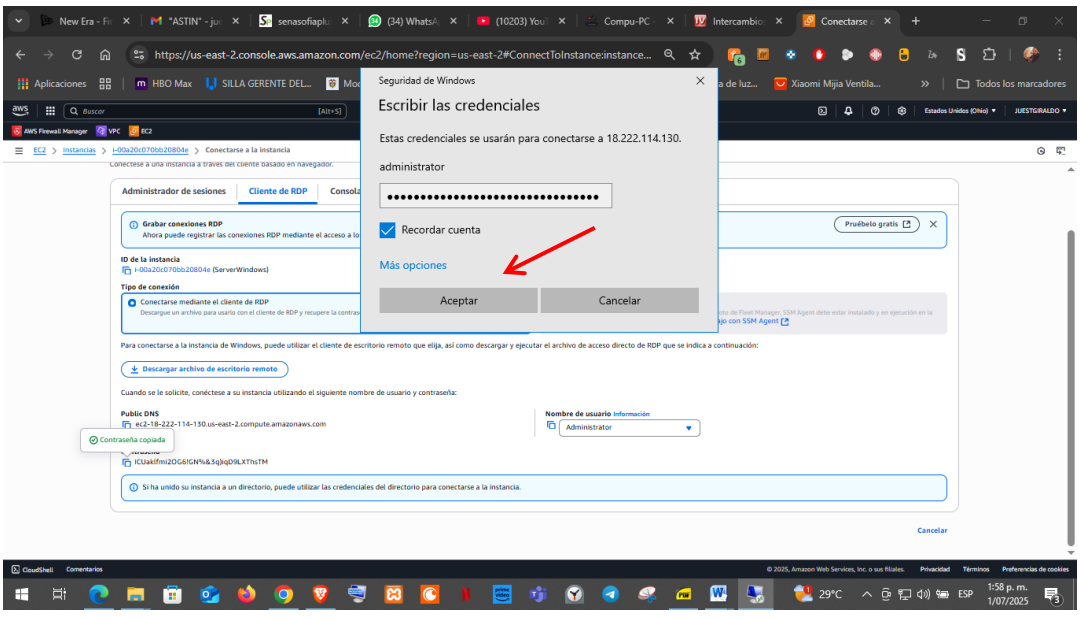
Accedemos al cliente de RDP de Windows, ingresamos IP Pública*: 18.222.114.130,* Usuario*: administrador.* Y la contraseña descifrada previamente.

Figura 29



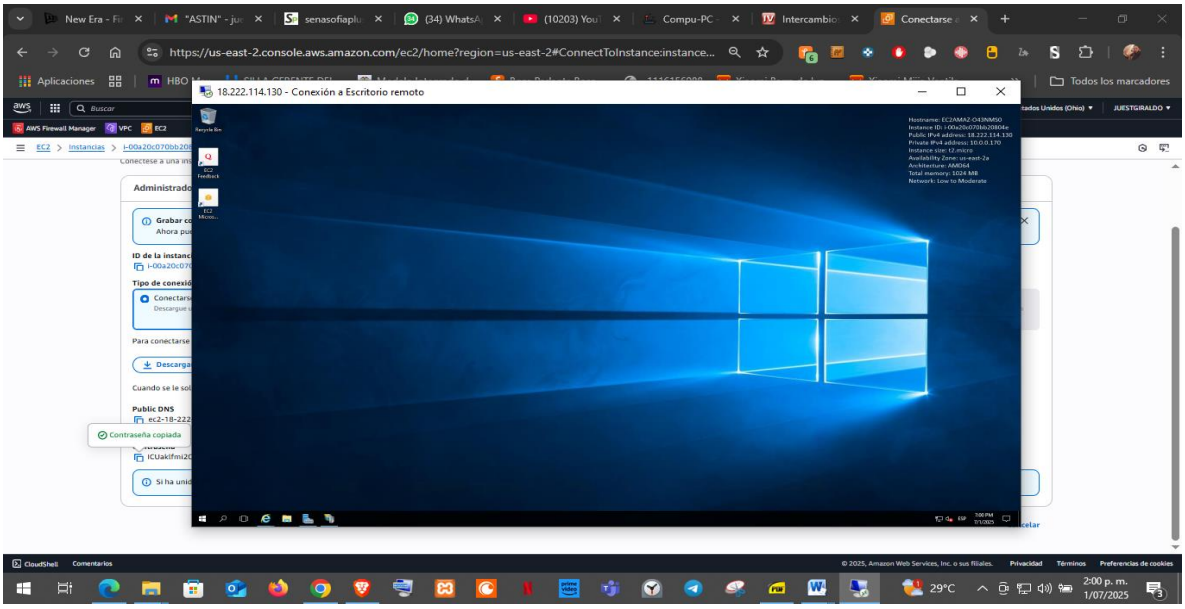
Fuente: Ejecución del Cliente RDP de Windows

Figura 30



Fuente: Ejecución del Cliente RDP de Windows.

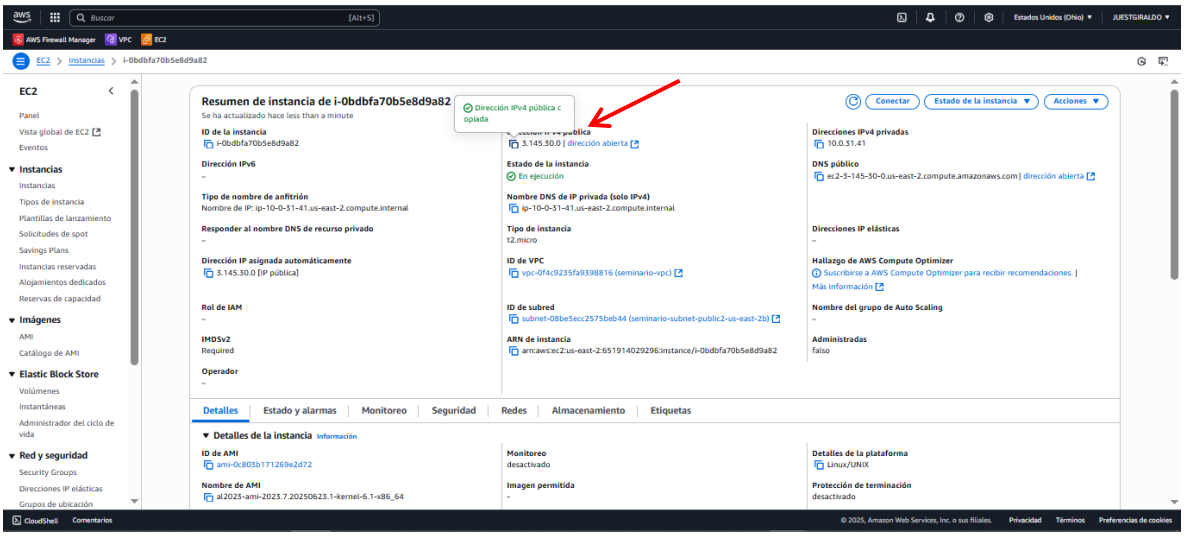
Figura 31



Fuente: Acceso la Instancia Windows Server a través del Cliente RDP de Windows

Acceso A Servidor Linux

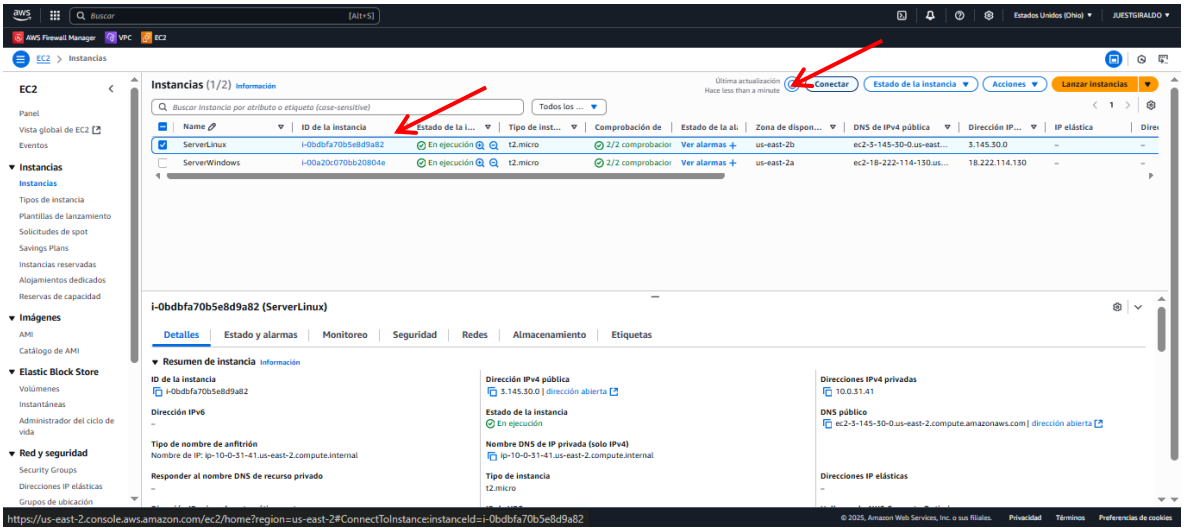
Figura 32



Fuente: Obtención IP PÚBLICA en AWS

Obtenemos la IP Publica: 3.145.30.0 o DNS Público (a Continuación), de nuestro Server Linux.

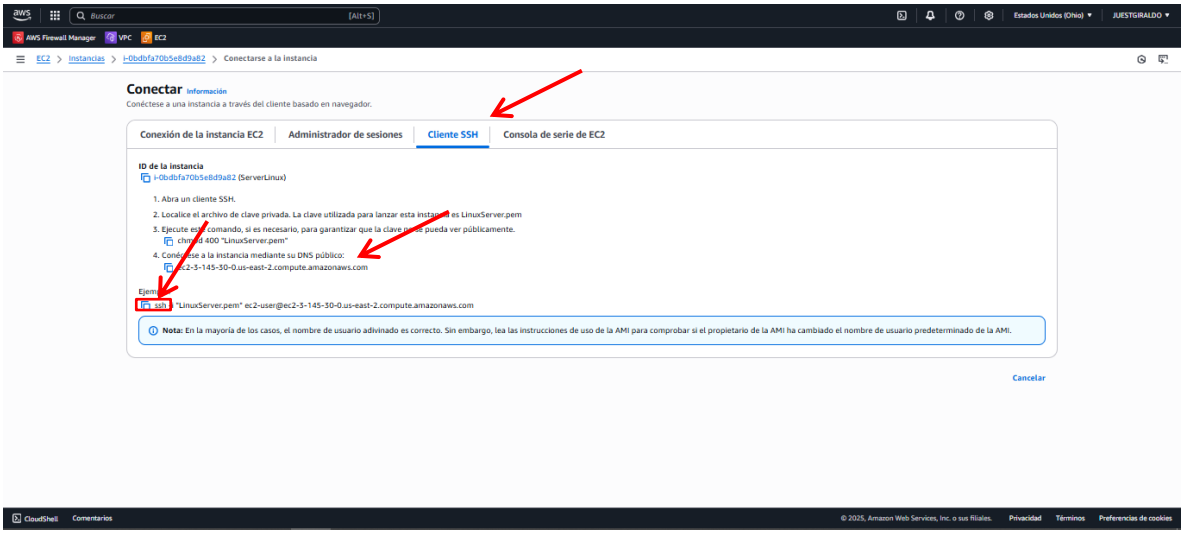
Figura 33



Fuente: Obtención de Acceso para el Cliente SSH en AWS

Obtenemos el acceso a ***ServerLinux*** mediante DNS Público y Usuario.

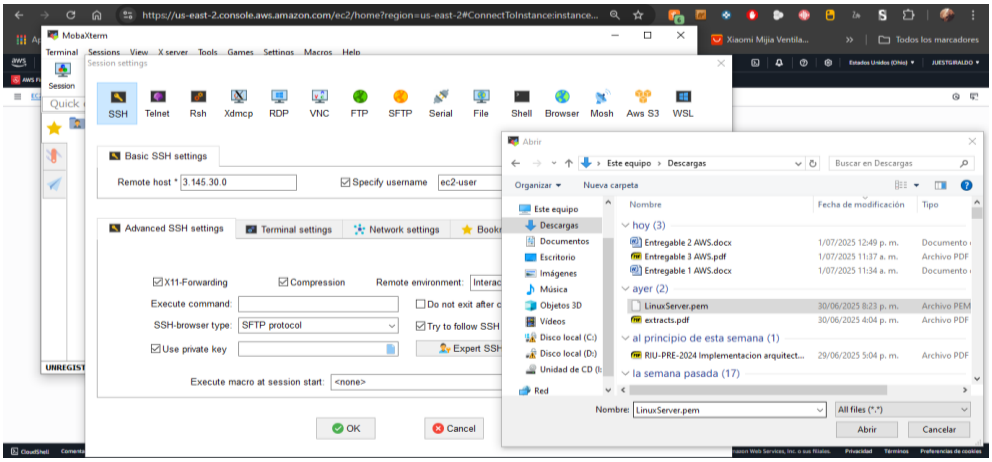
Figura 34



Fuente: Obtención de DNS Público y Usuario para el Cliente SSH en AWS

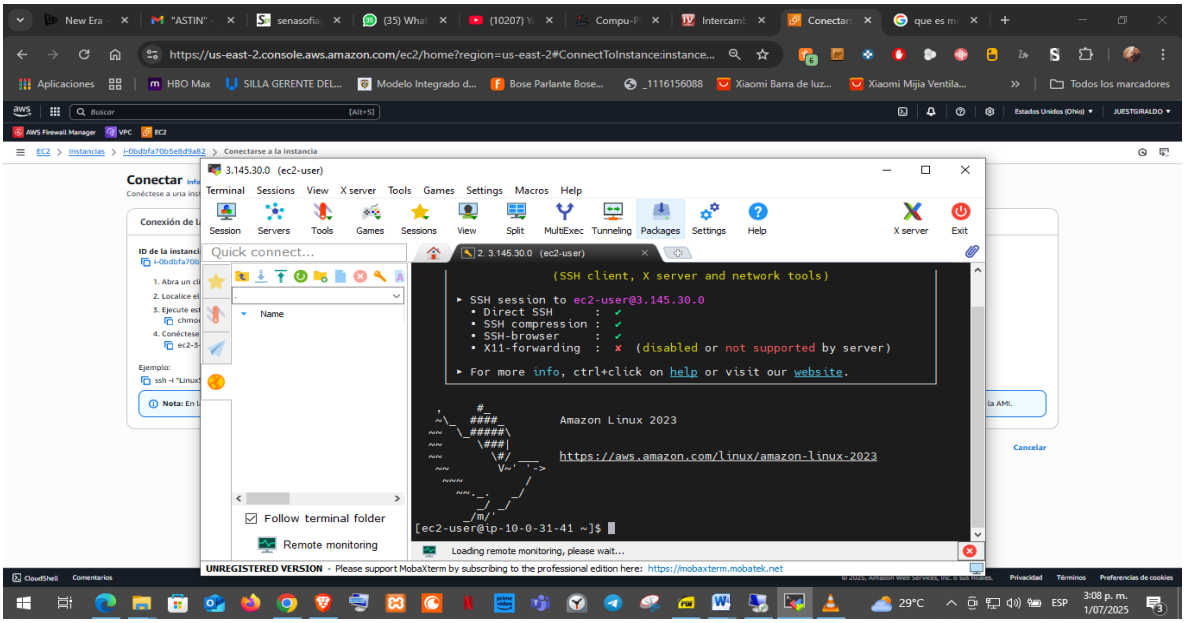
Obtenemos de DNS o IP Público*:* ***ec2-3-145-30-0.us-east-2.compute.amazonaws.com/3.145.30.0*** y Usuario*:* ***ec2-user*** para el Cliente SSH.

Figura 35



Fuente: Acceso la Instancia Linux Server a través de Mobaxterm como cliente de SSH

Figura 36



Fuente: Evidencia de acceso la Instancia Linux Server en Mobaxterm

Observamos que ya tenemos acceso a la instancia ***ServerLinux****.*

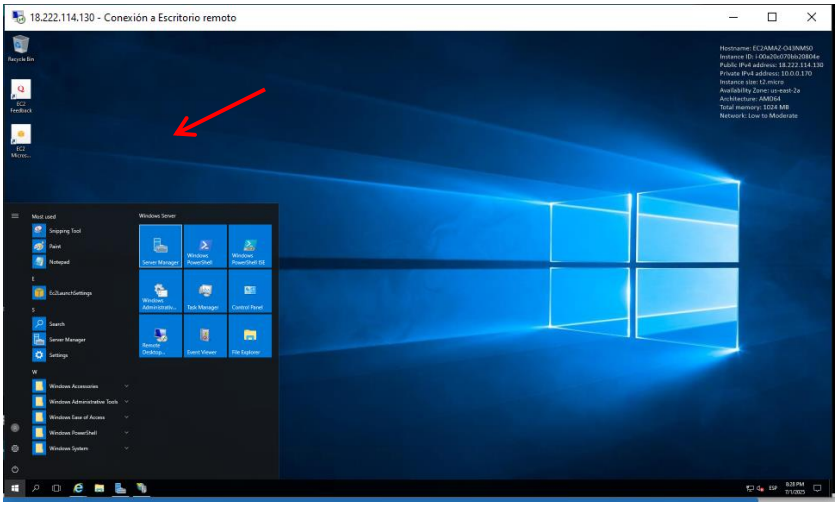
Consideraciones de seguridad, por ejemplo, uso de llaves PEM.

Como consideraciones de seguridad principalmente tenemos el uso de llave PEM Como archivos únicos e irreemplazables para cifrar el acceso a las instancias tanto de Linux Como Windows, se debe tener en cuenta donde almacenamos estos archivos para evitar su pérdida o copia no autorizada.

Configuración del servidor web

Pasos seguidos para instalar IIS en Windows Server.

Figura 37

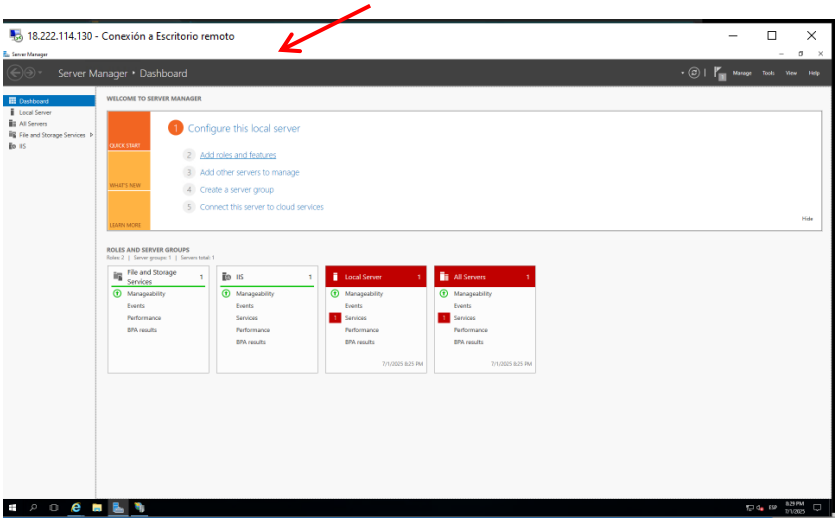


Fuente: Ingreso a Server Manager de la Instancia Windows

Dentro de la instancia Windows Server a través de RDP de Windows, damos clic en el botón de inicio e ingresamos al Server Manager.

Configuración del Web Server de la Instancia Windows.

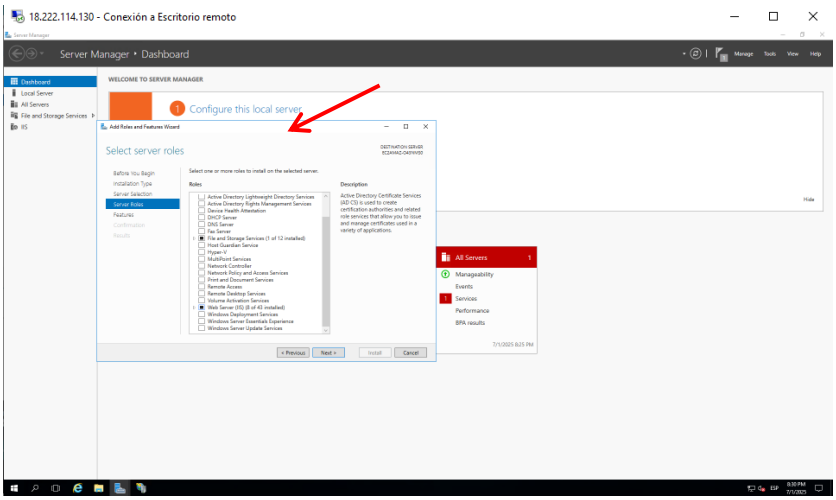
Figura 38



Fuente: Conexión a escritorio remoto en Windows

Damos clic en ***Add roles and features***.

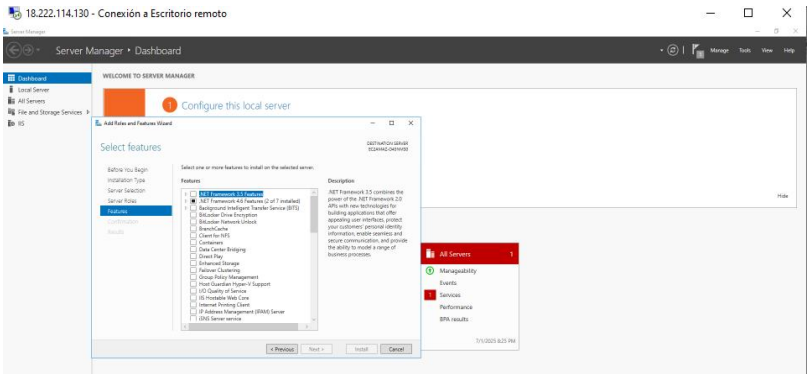
Figura 39



Fuente: Configuración del Web Server de la Instancia Windows

En el asistente damos siguiente hasta que parecen los servicios que deseamos instalar, y seleccionamos *Web Server (IIS)*.

Figura 40

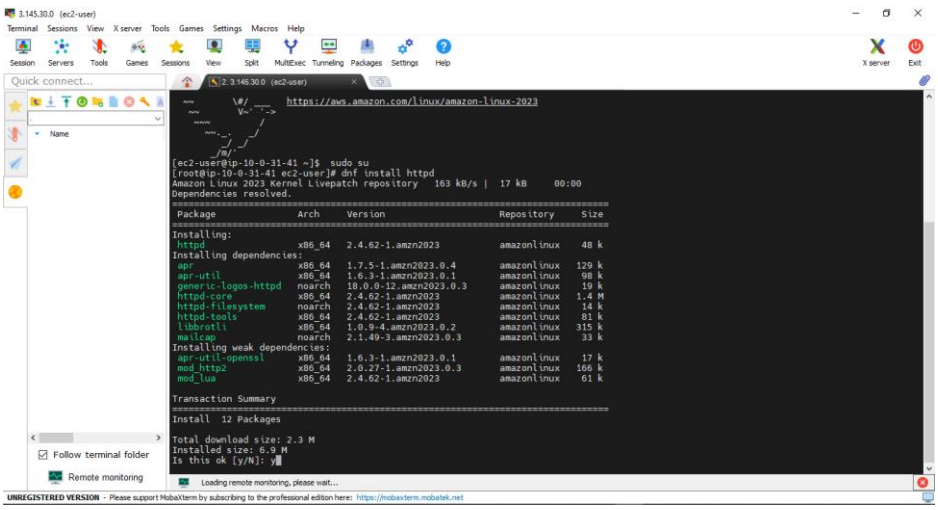


Fuente: Configuración del Web Server de la Instancia Windows

Continuamos dejando la configuración como viene por defecto dando clic en siguiente hasta que aparezca el botón *Install* y damos clic en el.

Pasos Para Instalar Apache O Nginx En Linux.

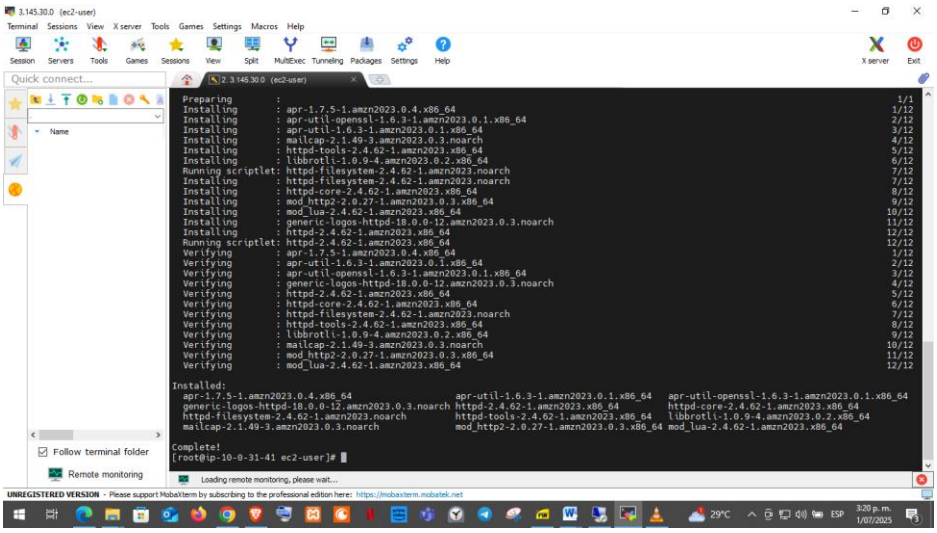
Figura 41



Fuente: Configuración del Web Server de la Instancia Linux

Dentro de la instancia Linux Server a través de ***Mobaxterm*** como cliente SSH, digitamos el comando: sudo su para cambiar a súper usuario, luego digitamos el comando: ***dnf install httpd*** para instalar Apache, digitamos: y para continuar.

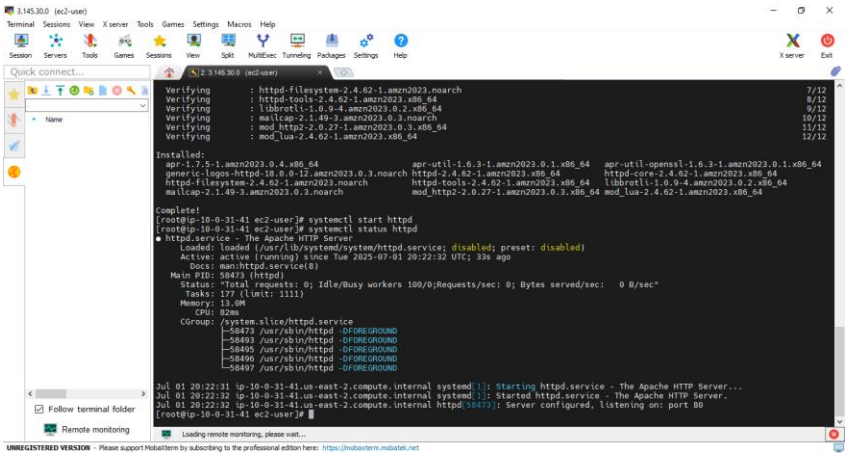
Figura 42



Fuente: Configuración del Web Server de la Instancia Linux

Verificamos la instalación ***Complete!***.

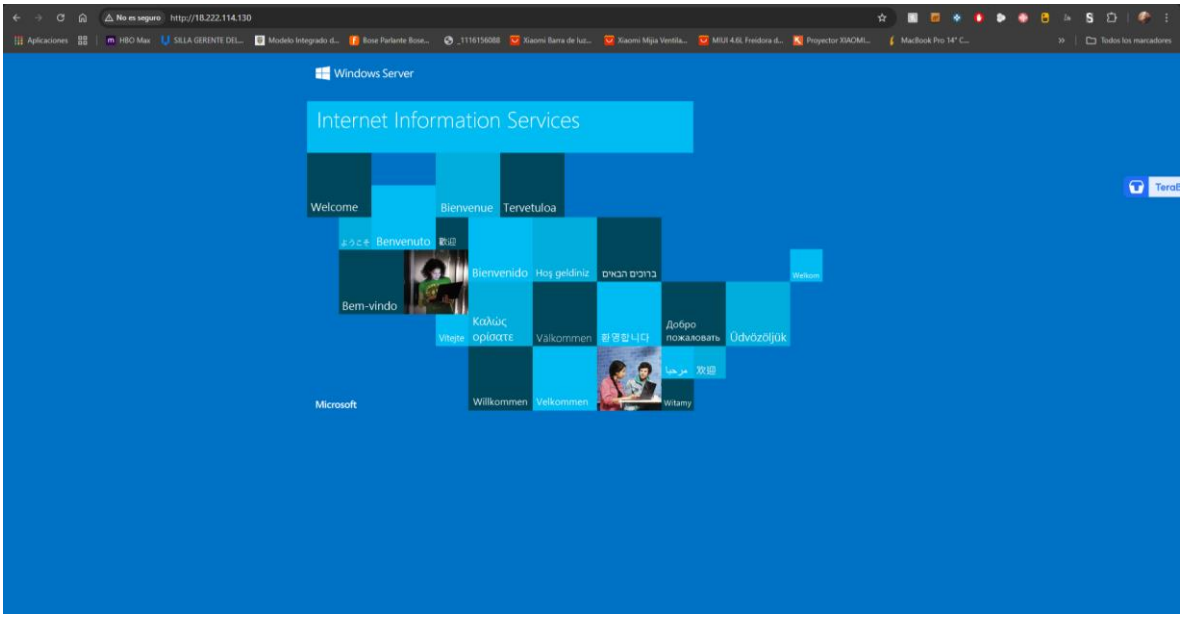
Figura 43



Fuente: Configuración del Web Server de la Instancia Linux

Iniciamos el Servicio con el comando*:* ***systemctl start httpd*** y verificamos su estado con el comando*:* ***systemctl status httpd***.

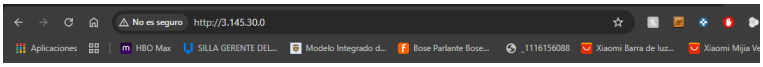
Figura 44



Fuente: Verificación IP Publica de nuestro Windows Server en nuestro equipo local

Pruebas básicas para verificar que los servidores web son accesibles desde Internet (captura de pantallas del navegador).

Figura 45

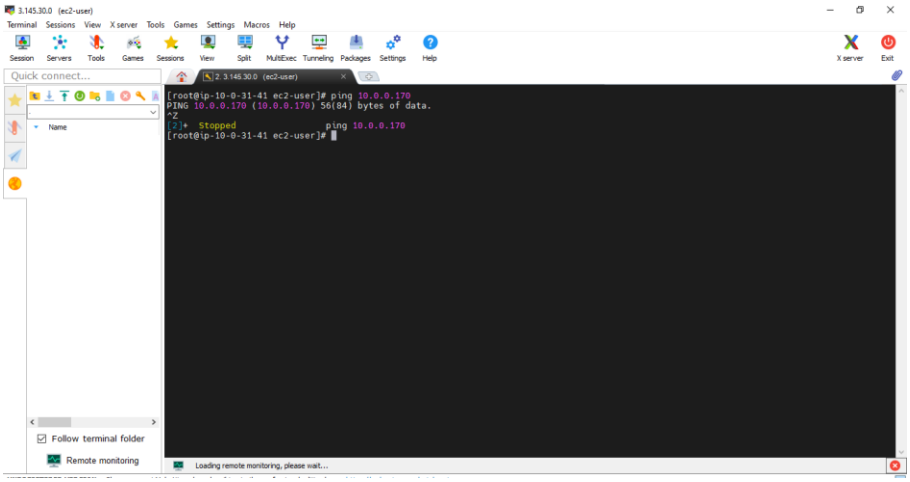


Fuente: Prueba de funcionamiento en navegador en equipo local

Pruebas De Conectividad

Desde la instancia Windows hacer ping a la IP privada de la instancia Linux y viceversa.

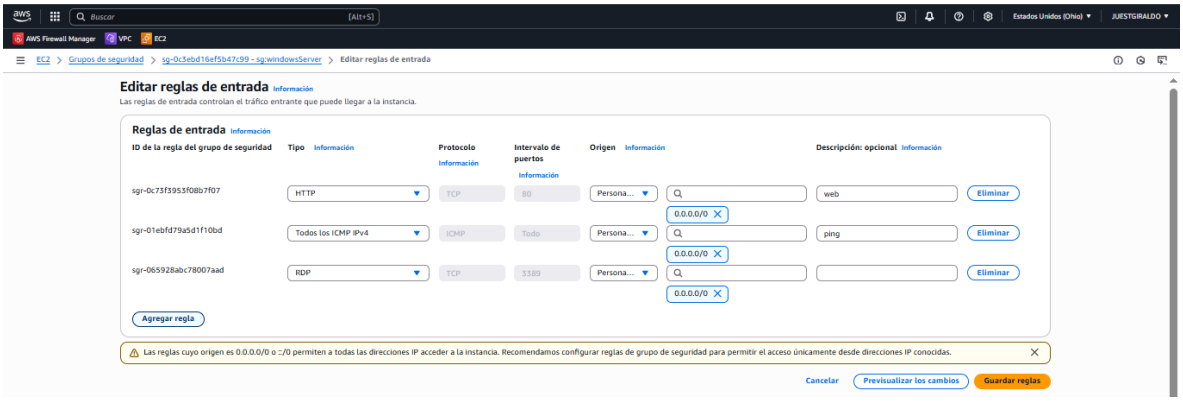
Figura 46



Fuente: Prueba ping desde la Instancia Linux a Windows

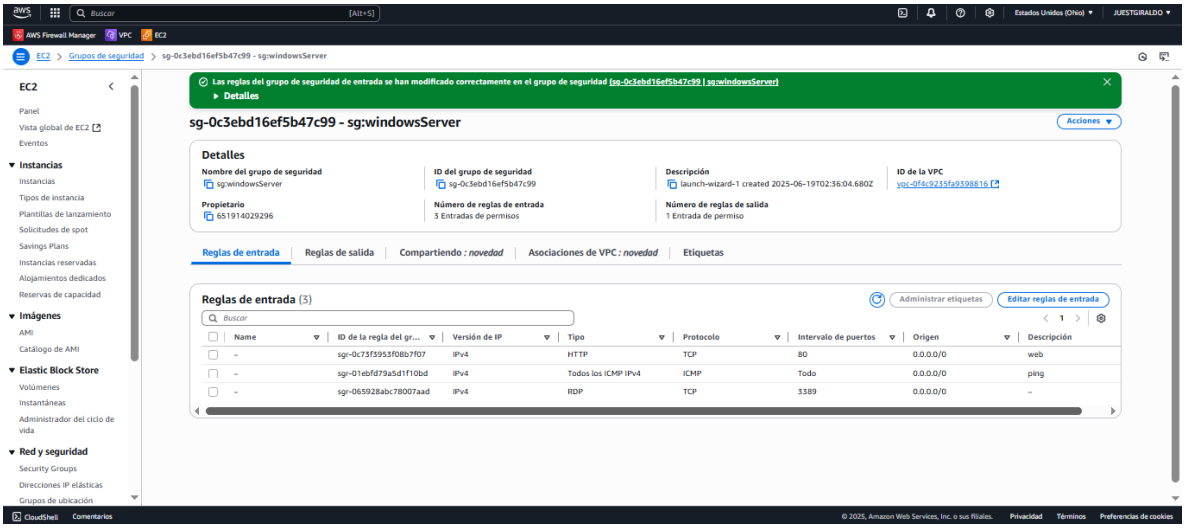
En un primero Intento de Ping entre las Instancias Linux y Windows y viceversa hubo total perdida de paquetes. Para Solucionar lo anterior fue necesario habilitar una nueva regla de entrada para habilitar el ICMP en los grupos de seguridad en ambas en Instancias así.

Figura 47



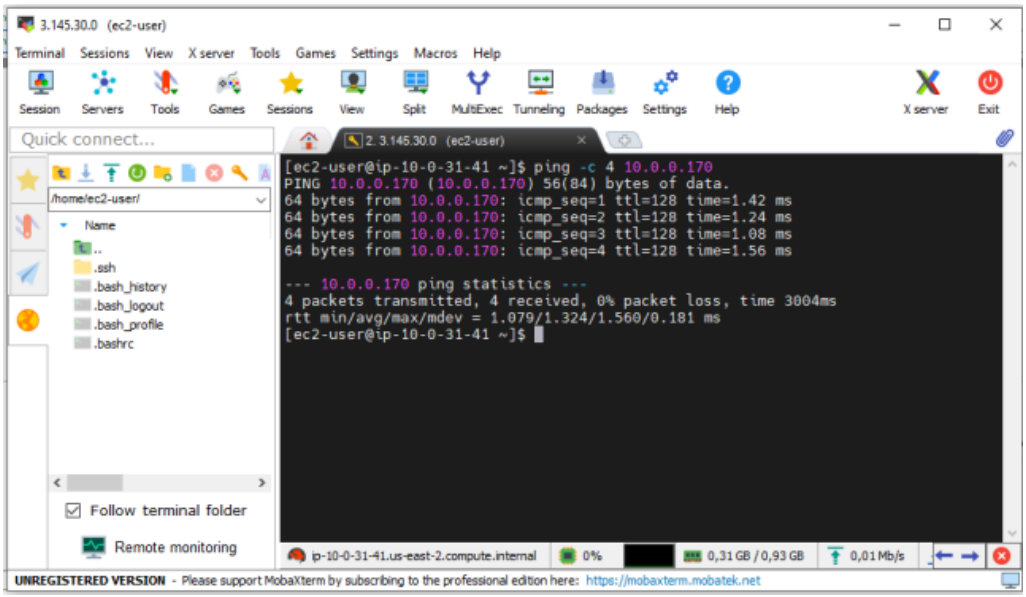
Fuente: Configuración de nueva regla de entrada para el protocolo ICMP en AWS

Figura 48



Fuente: Verificación de regla aplicada en AWS

Figura 49



Presentación de RápidoYa

El startup que decidimos presentar en este proyecto se llama "RápidoYa", la cual es una plataforma digital que nació con la idea de conectar restaurantes con clientes finales a través de entregas rápidas y eficientes. El objetivo principal de RápidoYa es facilitar el proceso de pedido de comida, especialmente en ciudades intermedias donde todavía no hay muchas opciones tecnológicas para este tipo de servicios.

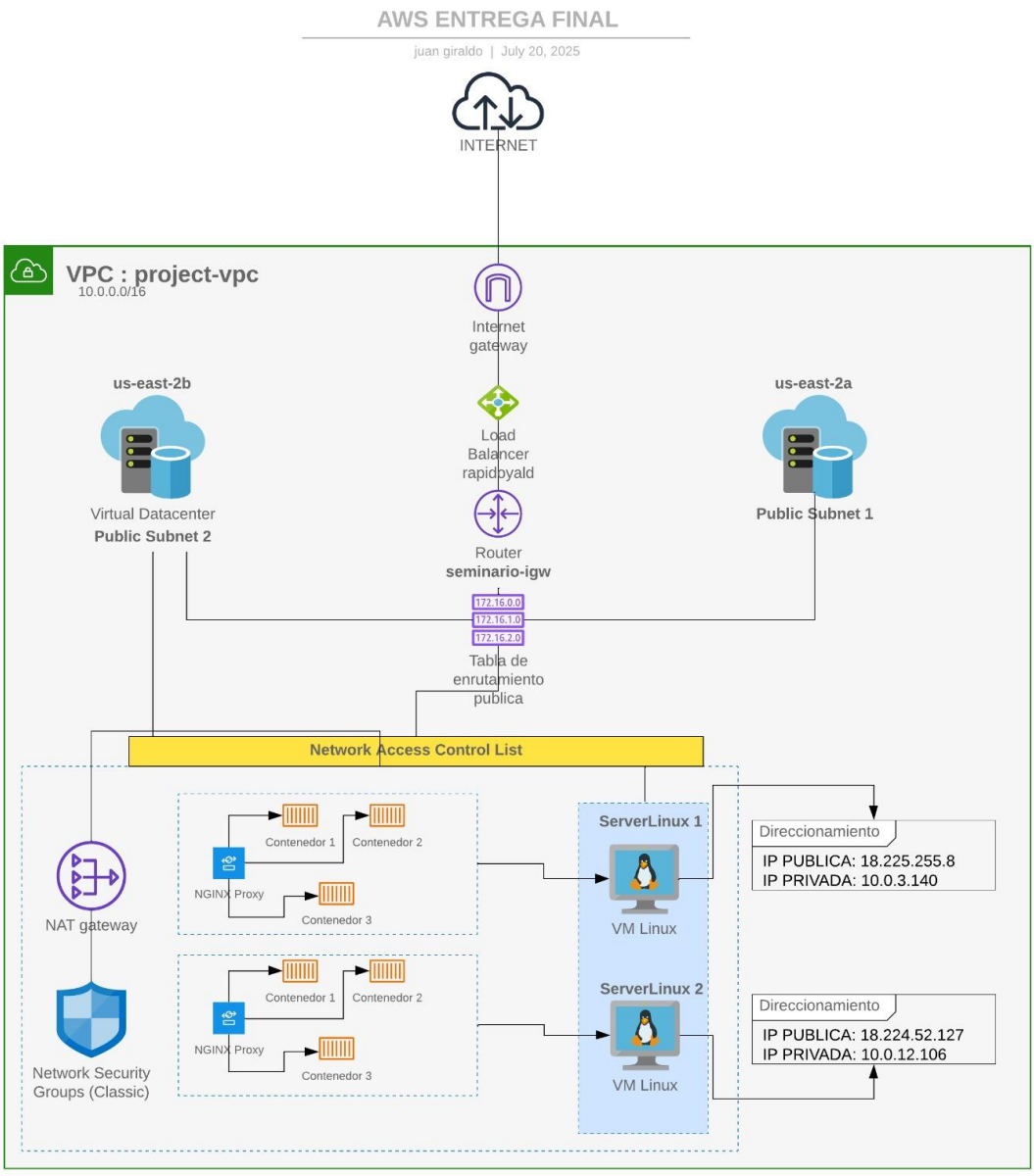
Durante los últimos meses, RápidoYa ha tenido un crecimiento bastante acelerado. Empezaron operando solo en una ciudad, pero gracias a la buena recepción por parte de los usuarios y al aumento en la demanda, han decidido ampliar su cobertura a nuevas regiones del país. Sin embargo, este crecimiento también ha traído nuevos retos, especialmente en el manejo del tráfico web, la disponibilidad del servicio y la escalabilidad de la plataforma.

Hasta ahora, la infraestructura que estaban usando era muy básica: un solo servidor virtual con todo montado allí mismo. Eso generaba cuellos de botella y problemas de disponibilidad cuando muchas personas usaban la app al mismo tiempo. Por eso, el equipo técnico de RápidoYa decidió dar el siguiente paso: migrar su arquitectura a la nube usando Amazon Web Services (AWS), para garantizar una mejor experiencia a los usuarios y estar preparados para seguir creciendo sin preocuparse por limitaciones técnicas.

El objetivo es implementar una arquitectura en AWS que sea escalable, de alta disponibilidad y que pueda manejar altos volúmenes de tráfico sin afectar el rendimiento de la aplicación. Esto incluye el uso de balanceadores de carga, instancias EC2 distribuidas en diferentes zonas de disponibilidad, un proxy reverso con Nginx, contenedores Docker para aislar los servicios, y políticas de Autoescalado para optimizar el uso de recursos.

Diagrama de Arquitectura RapidoYa

Figura 50



Fuente: Creación lucid.app

Diseño de la Arquitectura en AWS RápidoYa

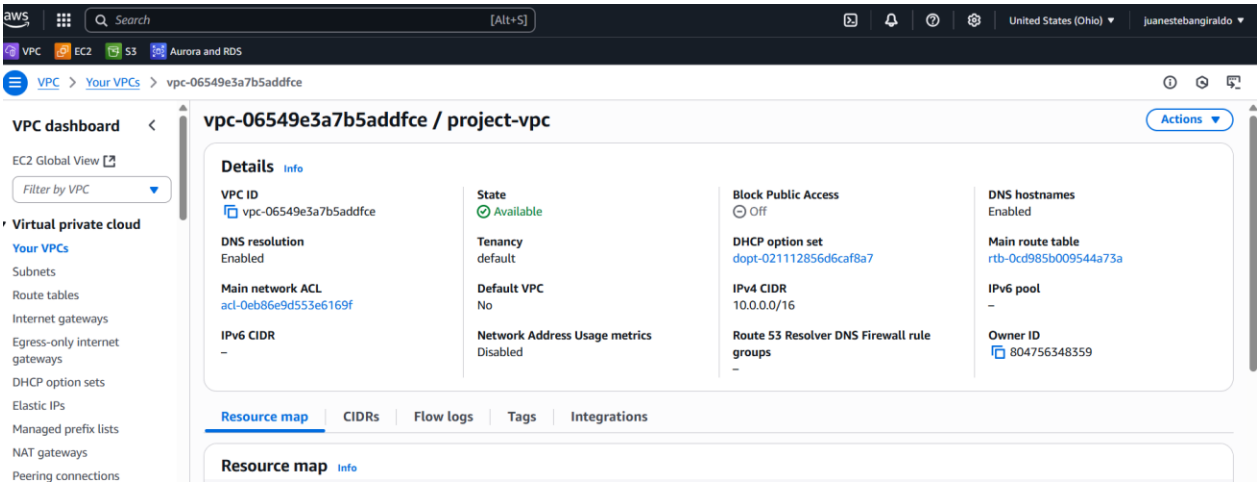
Debido a los problemas de escalabilidad y disponibilidad que presenta RápidoYa, hemos diseñado una arquitectura basada en servicios de AWS que permite distribuir la carga, escalar automáticamente y mantener el servicio siempre disponible. Explicaremos cómo está compuesta esta solución y cómo se comunican los diferentes elementos:

Implementación Paso a Paso en AWS RápidoYa

Configuración de la VPC y subredes

Lo primero fue crear una **VPC personalizada**, que llamamos ***project-vpc*** ya que necesitábamos un entorno de red aislado para desplegar los recursos. Asignamos un bloque de direcciones IP (CIDR) privado, por ejemplo 10.0.0.0/16.

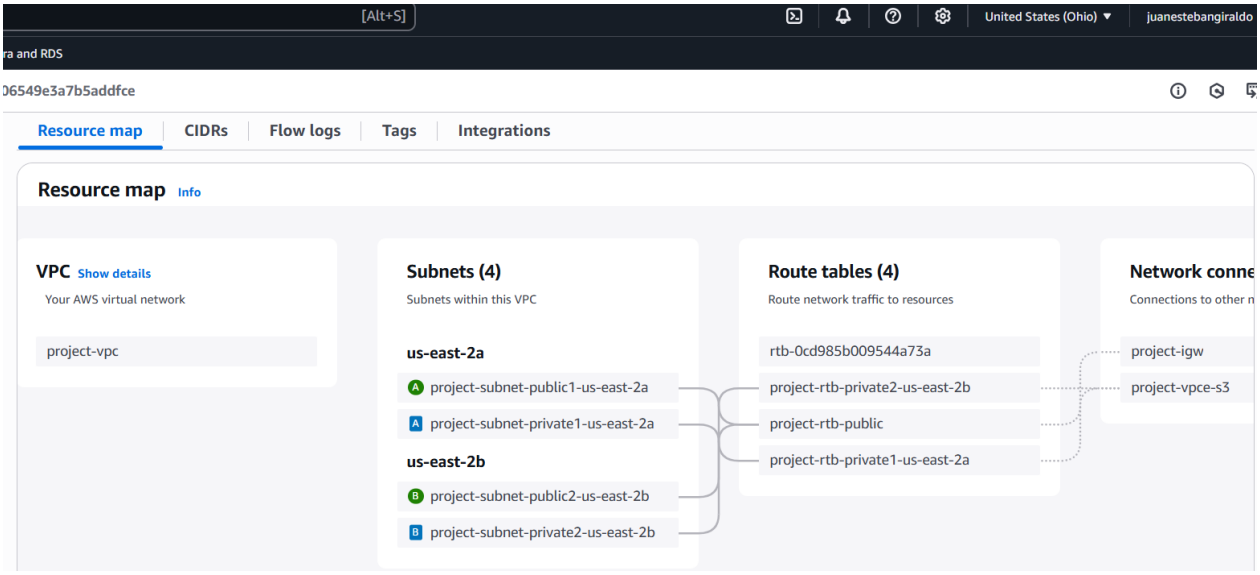
Figura 51



Fuente: VPC creado en AWS

Después, creamos **dos subredes públicas**, una en cada zona de disponibilidad (un ejemplo de ellas es ***us-east-2a*** y ***us-east-2b***), para garantizar alta disponibilidad. Estas subredes permiten que las instancias EC2 se comuniquen hacia afuera (Internet) a través de una Gateway.

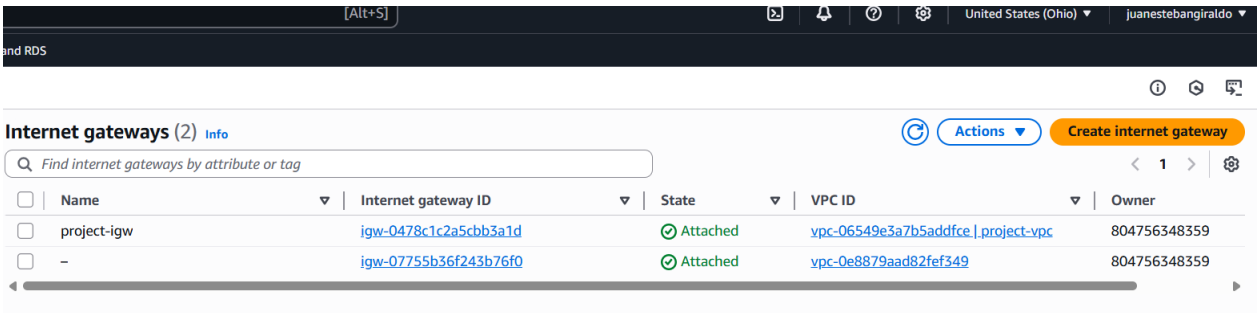
Figura 52



Fuente: Subnets creadas (públicas y privadas) en AWS

También creamos una **Internet Gateway** llamada ***project-igw*** y asociamos a la VPC, para que las instancias pudieran recibir tráfico desde fuera. En la tabla de ruteo agregamos una regla que redirige el tráfico ***0.0.0.0/0*** hacia esa Gateway.

Figura 53

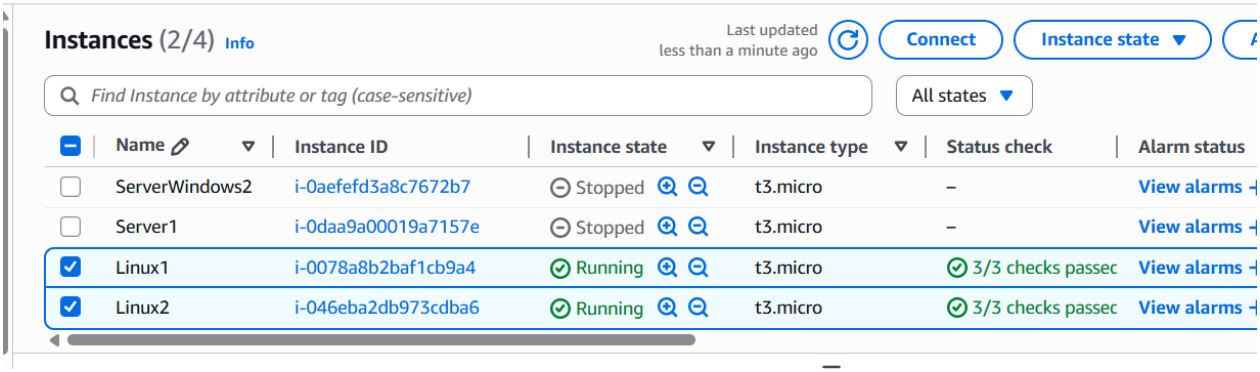


Fuente: Gateway creada en AWS

Creación de instancias EC2

Luego lanzamos dos instancias EC2 con Amazon Linux 2 como sistema operativo. Las configuramos en distintas zonas (una en cada subred).

Figura 54

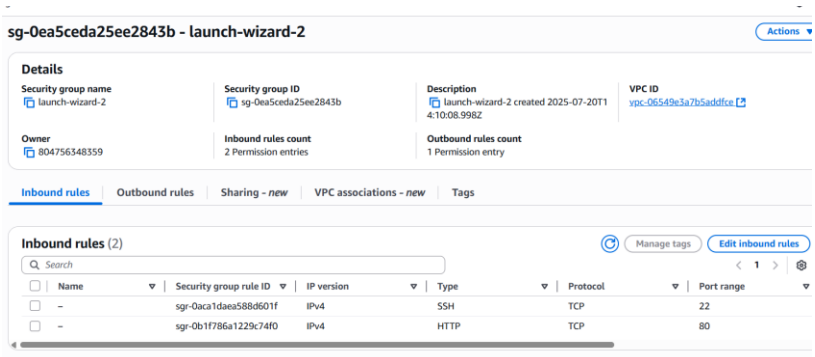


Fuente: Instancias Linux en AWS

Durante el proceso de lanzamiento:

- Asignamos un **par de llaves** para poder obtener acceso por *SSH*.

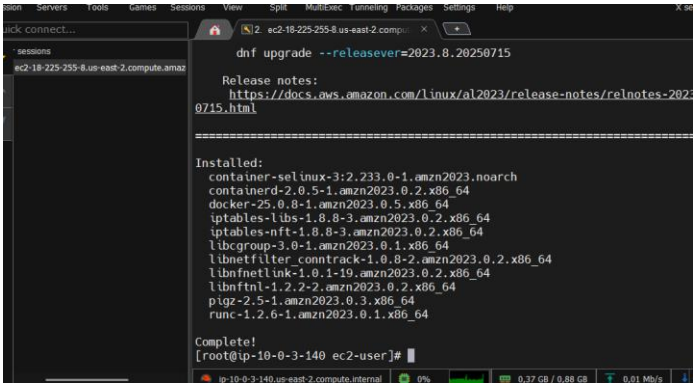
Figura 55



Fuente: 1

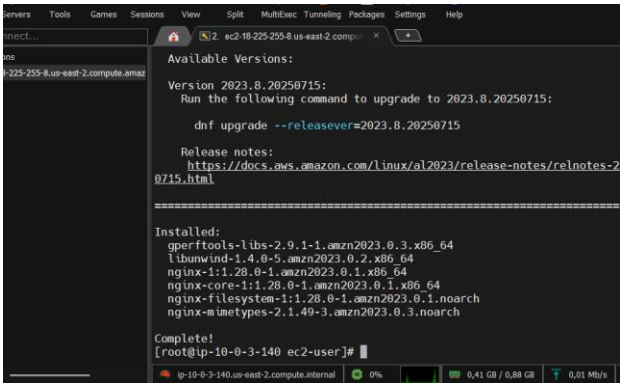
- Seleccionamos el tipo de instancia t3.micro (por temas de presupuesto y porque era suficiente para la prueba y t2.micro ya no se encuentra disponible para uso en prueba gratis).
- Instalamos actualizaciones del sistema y luego Docker y Nginx. Esto lo hicimos por vía terminal con ***mobaxterm***

Figura 56



Fuente: Docker por mobaxterm

Figura 57

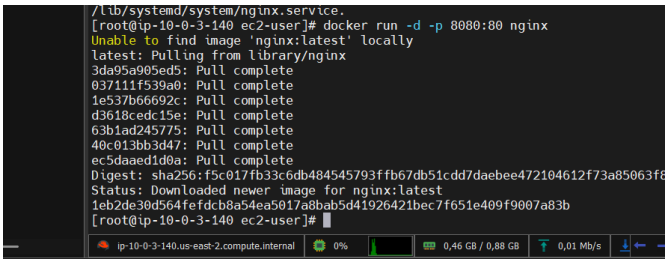


Fuente: NginX por mobaxterm

Configuración del contenedor Docker

Dentro de cada instancia EC2, creamos un archivo Dockerfile sencillo con una app de prueba en Node.js. También probamos usando una imagen pública de NGINX como servidor web para agilizar el despliegue.

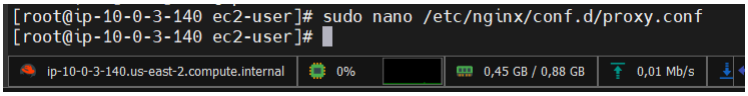
Figura 58



Fuente: Nginx en Docker

Con esto, la aplicación quedó corriendo en el puerto 8080. Desde Nginx (instalado en el sistema) configuramos el proxy reverso para redirigir las peticiones entrantes hacia el contenedor.

Figura 59



Fuente: configuración de Nginx

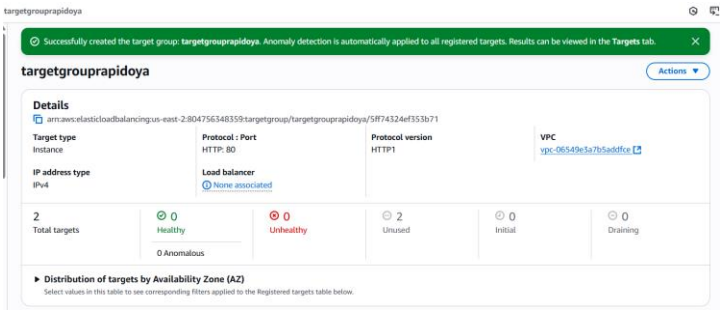
Configuración del Application Load Balancer (ALB)

Desde el servicio de EC2, en "Load Balancers" y creamos un **Application Load Balancer** así:

- Tipo: Application Load Balancer
- Esquema: Internet-facing
- Listeners: Puerto 80 (HTTP)
- VPC: La inicial
- Subredes: Ambas subredes públicas

Luego creamos un **Target Group** donde agregamos las dos instancias EC2 manualmente y configuramos chequeos de salud en la ruta /.

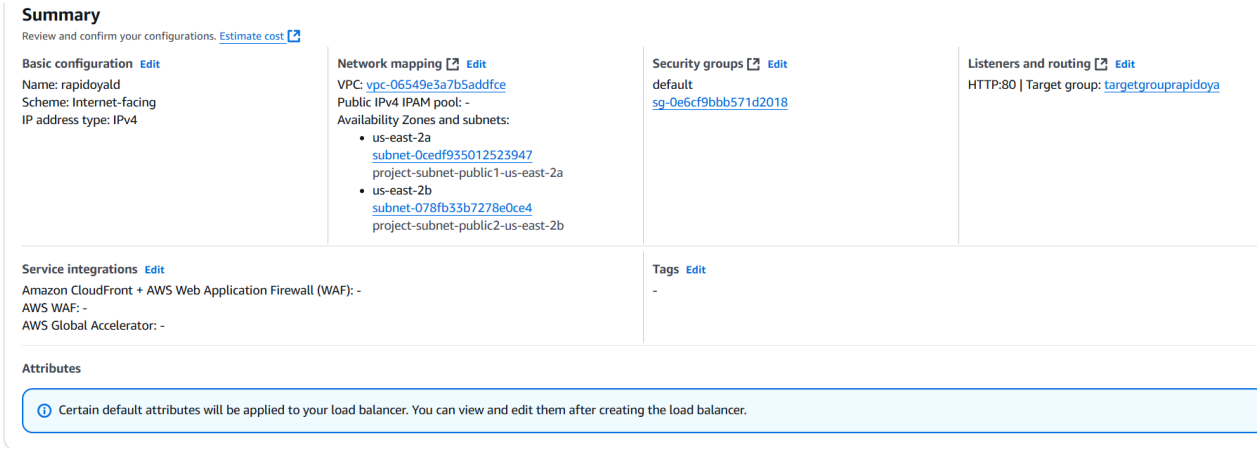
Figura 60



Fuente: targetgrouprapido ya en AWS

Finalmente, asociamos ese Target Group al ALB. Esto permitió que el tráfico entrante se distribuya automáticamente a cualquiera de las dos instancias que estén funcionando.

Figura 61



Fuente: ALB creado en AWS

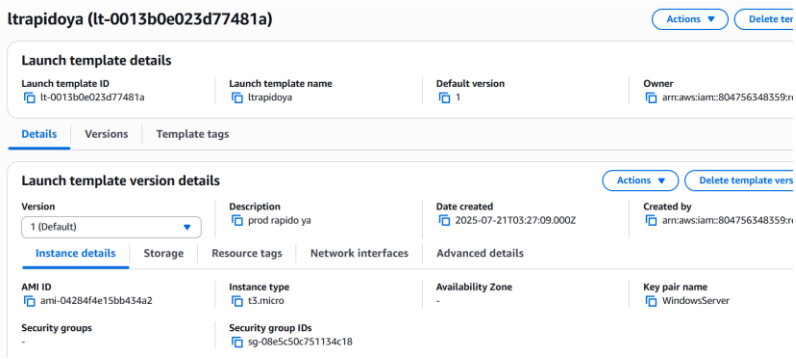
Configuración del Auto Scaling

Desde el panel de Auto Scaling, se configuro un **Launch Template** basado en la configuración de las instancias EC2 que ya se tenía. Luego creamos un **Auto Scaling Group** con lo siguiente:

- Mínimo de instancias: 2
- Máximo: 4
- Política de escalado: basada en el uso del CPU (ejemplo, si el promedio supera el 60% se lanza una nueva instancia)

  Esto nos permite que el sistema se ajuste automáticamente según la carga que reciba.

Figura 62



Fuente: Austo Scaling en AWS

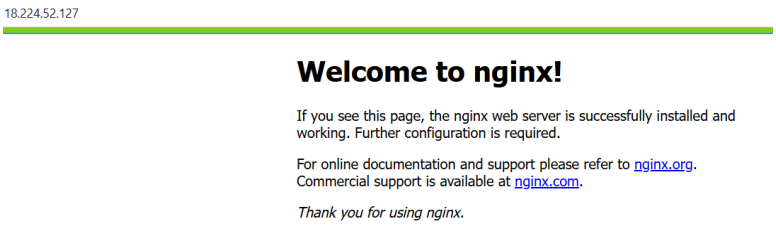
Pruebas

Para probar si el Application Load Balancer estaba funcionando como debía, accedimos a la **URL pública** que nos entregó AWS al finalizar la configuración del ALB con 18.225.255.8 y 18.224.52.127. Se ingreso la URL varias veces, actualizando la página rápidamente, para simular múltiples usuarios.

Figura 63

Fuente: Prueba 1

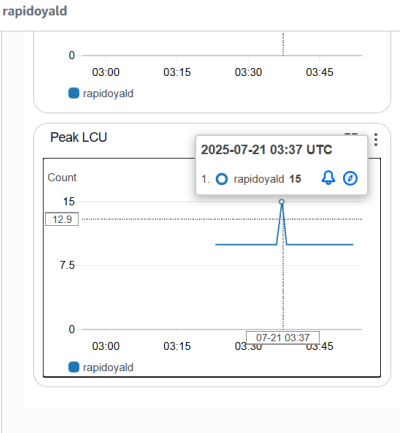
Figura 64



Fuente: Prueba 2

Luego revisamos los logs del servidor Nginx y notamos que las solicitudes estaban llegando de forma alternada a las dos instancias EC2. Esto nos confirmó que el balanceador de carga estaba distribuyendo el tráfico correctamente entre ambas.

Figura 65



Fuente: Balanceador de carga en AWS

Conclusiones

Con esta implementación utilizada en Amazon Web Service, logramos transformar completamente la arquitectura de RápidoYa, pasando de una solución básica y monolítica a una infraestructura moderna, escalable y altamente disponible. Usando servicios clave como EC2, ALB, Auto Scaling y Docker, ahora la plataforma está preparada para soportar una mayor demanda de usuarios sin comprometer el rendimiento ni la estabilidad.

Uno de los logros más importantes cumplidos fue comprobar que el balanceador de carga distribuye correctamente el tráfico entre múltiples instancias, y que el sistema responde bien ante fallos o picos de uso gracias al autoescalado. Además, el uso de contenedores con Docker simplifica el despliegue de la aplicación y permite mayor flexibilidad en el manejo de los servicios internos.

En términos generales, esta arquitectura no solo cumple con los requisitos técnicos del proyecto, sino que también deja una base sólida para seguir creciendo a futuro. A medida que la empresa escale, se podrían integrar otros servicios de AWS como RDS, Elastic Container Service (ECS) o incluso migrar a soluciones serverless para optimizar aún más los recursos.

Este trabajo demostramos que, con una buena planeación y uso adecuado de las herramientas disponibles, es posible crear soluciones robustas que respondan a las necesidades reales de una empresa en crecimiento.

Referencias Bibliográficas

Amazon Web Services. (2023). *What is Amazon EC2?*. AWS Documentation. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Amazon Web Services. (2023). *Elastic Load Balancing - Application Load Balancers*. AWS Documentation. https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html

Amazon Web Services. (2023). *Amazon EC2 Auto Scaling*. AWS Documentation. https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html

Docker Inc. (2023). *Docker Overview*. https://docs.docker.com/get-started/overview/

Nginx Inc. (2023). *Understanding the Reverse Proxy Functionality of NGINX*. https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/

Ministerio de Comercio, Industria y Turismo. (2017). *NTC ISO/IEC 20000-1:2011 – Tecnología de la información – Gestión del servicio*. Icontec.

ISO. (2018). *ISO/IEC 27001:2013 – Information Security Management Systems*. International Organization for Standardization. https://www.iso.org/isoiec-27001-information-security.html

Mora, J. & García, D. (2022). *Implementación de soluciones en la nube con AWS*. Editorial Campus Digital