



TRABAJO DE GRADO
Opción Seminario-Diplomado.

Implementación servicios AWS Seguro

Corporación Universitaria Remington.
Facultad de ingeniería
Ingeniería de Sistemas

Juan Camilo López Alvarez
Ángel Omar Mancipe Pabón
David Santiago Rodríguez Madiedo.

Juan Pablo Berrio López.
Opción de Trabajo de grado Seminario-Diplomado.
2025

Tabla de Contenidos

Resumen.....	3
Marco conceptual y contextual	3
Implementación Servidores	5
VPC.....	5
Creación VPC	6
Video Conexión	9
Video General	9
Implementación Docker.....	9
Ejecución de multiples contenedores.....	10
Conclusiones	18
Referencias.....	20

Resumen

En las organizaciones actuales se ha visto cada vez más la necesidad de sistematizar y automatizar procesos con la información de forma segura, con el propósito de optimizar el tiempo y las habilidades de los colaboradores en sus puestos de trabajo garantizando que los datos estén seguros y disponibles para la toma de decisiones. Por lo cual las áreas de sistemas buscan herramientas como AWS que permiten de una forma costo eficiente, moderna y segura gestionar las diferentes plataformas empresariales aumentando la disponibilidad para la toma de decisiones de los directivos, por esta razón en este trabajo es una muestra real en un entorno controlado del alcance que puede tener la virtualización de servicios de forma segura en la nube utilizando recursos bajo demanda. Este desarrollo académico muestra la parametrización mínima segura que debe tener un servidor Windows y Linux para exponer servicios de forma controlada para interactuar de forma interna y con otras compañías en cualquier lugar del mundo.

Palabras clave

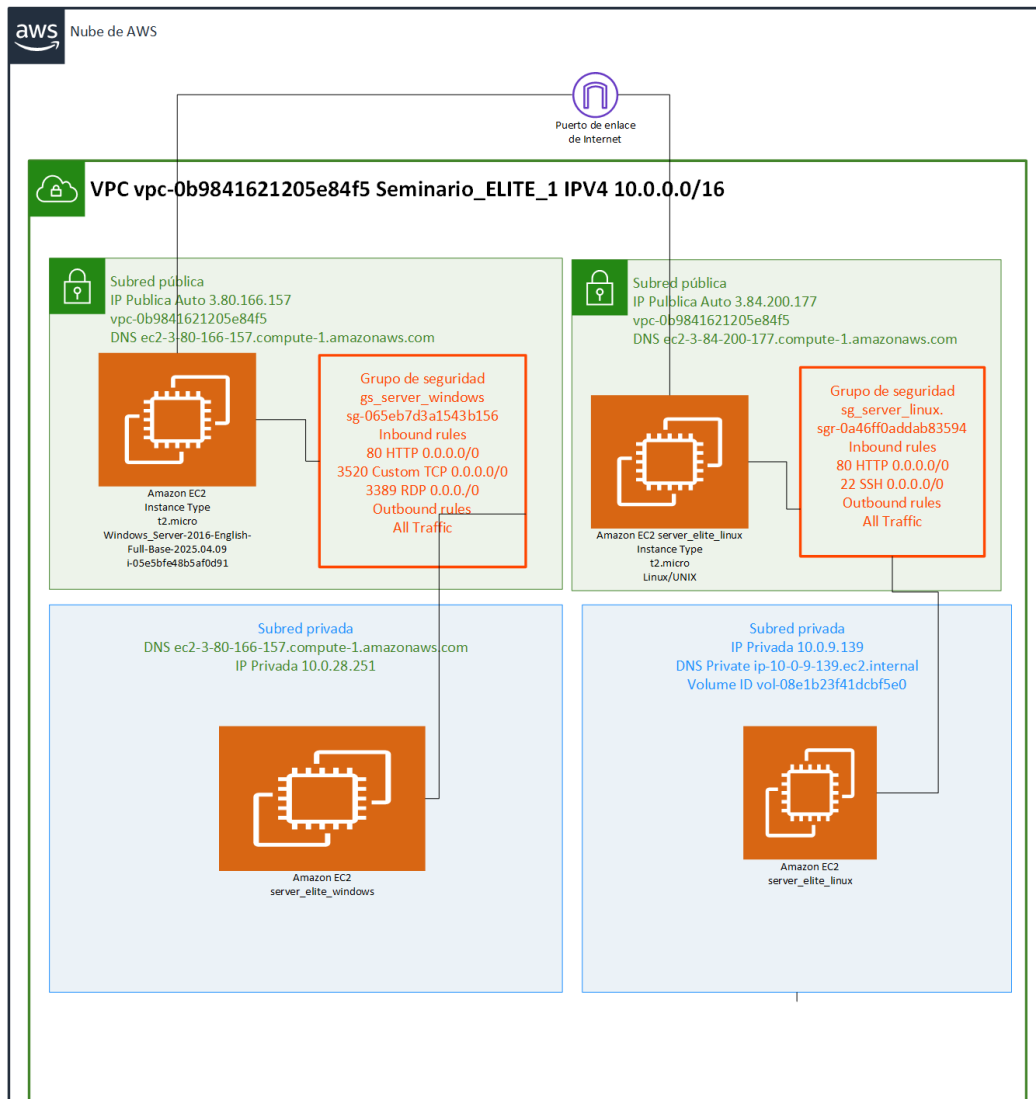
Alta disponibilidad, servicios web, recursos compartidos, capacidades adaptativas, sistemas operativos.

Marco conceptual y contextual

Este ejercicio practico es objeto de la teoría vista en el sermonario de AWS donde se detallo los aspectos mas importantes para realizar una implementación de servidores en la nube de AWS con los criterios de seguridad, conectividad y desarrollo más utilizados para desplegar servicios utilizando virtualización con Linux y Windows en este caso dando como resultado la comparación en rendimiento y el análisis de costos al evidenciar que los sistemas Windows requieren algunas recursos adicionales que encarecen los servicios. Y por tal razón la mayoría de los servicios en la nube se ejecutan bajo sistemas operativos Linux. El desarrollo de esta actividad permitió conocer las generalidades que tiene AWS y sus potencialidades en entornos de prueba, pero abre el espectro para proyectos y soluciones empresariales o académicas en inteligencia artificial para soluciones de procesamiento de información, desarrollo de video juegos, retail, y cualquier área del comercio facilitando en poco tiempo formas de acceder al mercado y su competencia. En este laboratorio se encuentran generalidades pero también pasos básicos para crear instancias, parametrizarlas en seguridad, conectividad y entornos de exposición dependiendo del requerimiento en este caso solo exponer un servicio que mostrara conexión exitosa y ver su polifuncionalidad reutilizando recursos creados entre instancias y servidores.

Implementación Servidores

Figura 1 Diagrama Instancias AWS



VPC

En AWS este **módulo** permite establecer los criterios de conectividad a nivel de red como direccionamiento, tablas de enrutamiento y configuraciones **específicas** de la red la cuales connotaran de **qué** forma vamos a exponer el ingreso y salida de servicios que tengamos en los servidores. Es virtual para los usuarios, pero en los datacenter físicos

lógicamente contarán con la misma infraestructura que se conoce en entornos locales con tecnologías de alta disponibilidad y seguridad.

Creación VPC

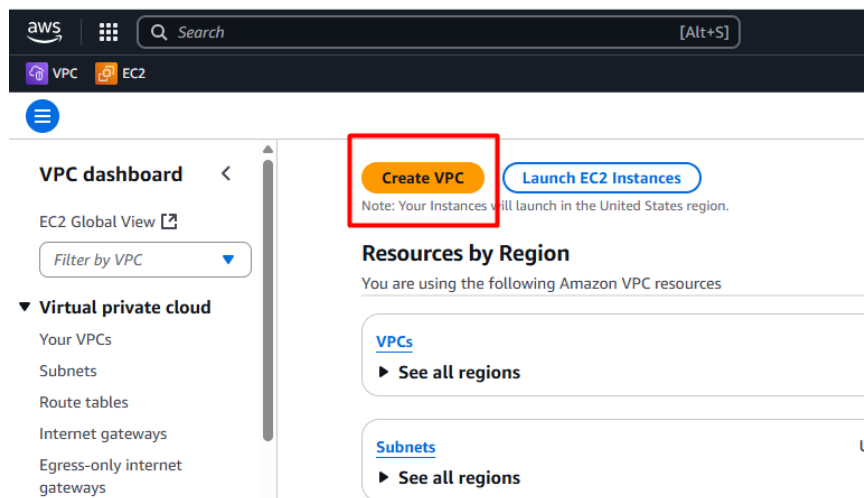
- Iniciamos Sesión en AWS con una cuenta gratuita o paga
- Buscamos la opción VPC

Figura 2 menú VPC



- Seleccionamos la opción Create VPC para iniciar la configuración

Figura 3 Creación VPC



- Según la necesidad configuramos la VPC en este caso por fines académicos lo dejamos así:

Figura 4 Nombre VPC

Create VPC [Info](#)

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, suc

VPC settings

Resources to create [Info](#)

Create only the VPC resource or the VPC and other networking resources.

VPC only

VPC and more

Name tag auto-generation [Info](#)

Enter a value for the Name tag. This value will be used to auto-generate Name tags for all resources in the VPC.

Auto-generate

Nombre_VPC

IPv4 CIDR block [Info](#)

Determine the starting IP and the size of your VPC using CIDR notation.

10.0.0.0/16

65.536 IPs

CIDR block size must be between /16 and /28.

IPv6 CIDR block [Info](#)

No IPv6 CIDR block

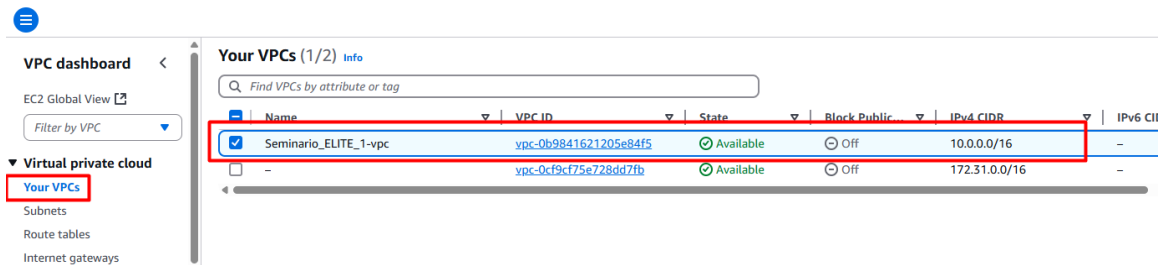
Amazon-provided IPv6 CIDR block

Tenancy [Info](#)

Default

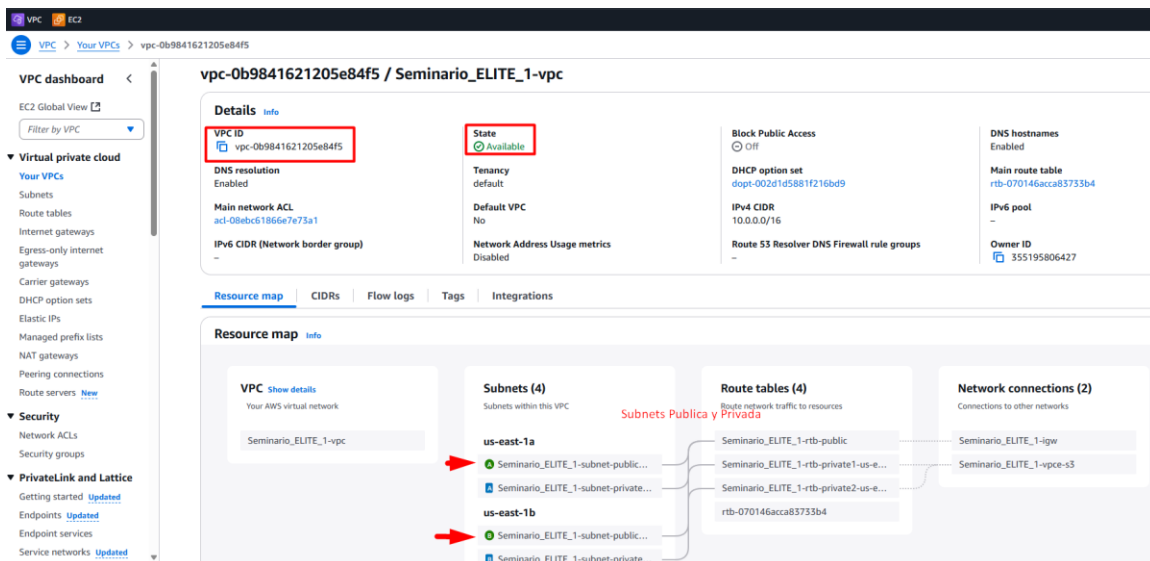
- Luego de crear la VPC en la opción your VPC'S encontraremos la VPC por defecto y las demás que estén creadas en algún momento por nosotros como en este caso Seminario_ELITE_1-vpc

Figura 5 Validación VPC



- Luego seleccionamos la VPC creada y podremos ver todas las opciones con la que esta generada y su estado así:

Figura 6 Información Importante VPC



- En la opción Subnets encontraremos el detalle del direccionamiento, estado de acceso público y privado de nuestra red.

Figura 7 Subnets Referencia

The screenshot shows the AWS Management Console interface for the 'Subnets' page. The left sidebar is titled 'Virtual private cloud' and includes a 'Subnets' link. The main content area displays a table of subnets with columns for Name, Subnet ID, State, and VPC. A red box highlights a selection of subnets, including 'Seminario_ELITE_1-subnet-private1-us-east-1a', 'Seminario_ELITE_1-subnet-private2-us-east-1b', 'Seminario_ELITE_1-subnet-public1-us-east-1a', and 'Seminario_ELITE_1-subnet-public2-us-east-1b'.

Name	Subnet ID	State	VPC
Seminario_ELITE_1-subnet-private1-us-east-1a	subnet-0fa921b2f02757a05	Available	vpc-0cf9cf75e728dd7fb
Seminario_ELITE_1-subnet-private2-us-east-1b	subnet-0e4807a43951f8c95	Available	vpc-0b9841621205e84f5 Sem...
Seminario_ELITE_1-subnet-public1-us-east-1a	subnet-08065dd77d567bebb	Available	vpc-0b9841621205e84f5 Sem...
-	subnet-02014868e779deb35	Available	vpc-0cf9cf75e728dd7fb
-	subnet-03b68a43768056e2d	Available	vpc-0cf9cf75e728dd7fb
Seminario_ELITE_1-subnet-public2-us-east-1b	subnet-0fcc95b52008453f0	Available	vpc-0b9841621205e84f5 Sem...
-	subnet-01dd2f791e4bf6a42	Available	vpc-0cf9cf75e728dd7fb
-	subnet-0599c426a1e797a46	Available	vpc-0cf9cf75e728dd7fb
-	subnet-0a9cc91c8dc91efd	Available	vpc-0cf9cf75e728dd7fb

Video Conexión

[Primera Entrega-20250507_173208-Grabación de la reunión.mp4](#)

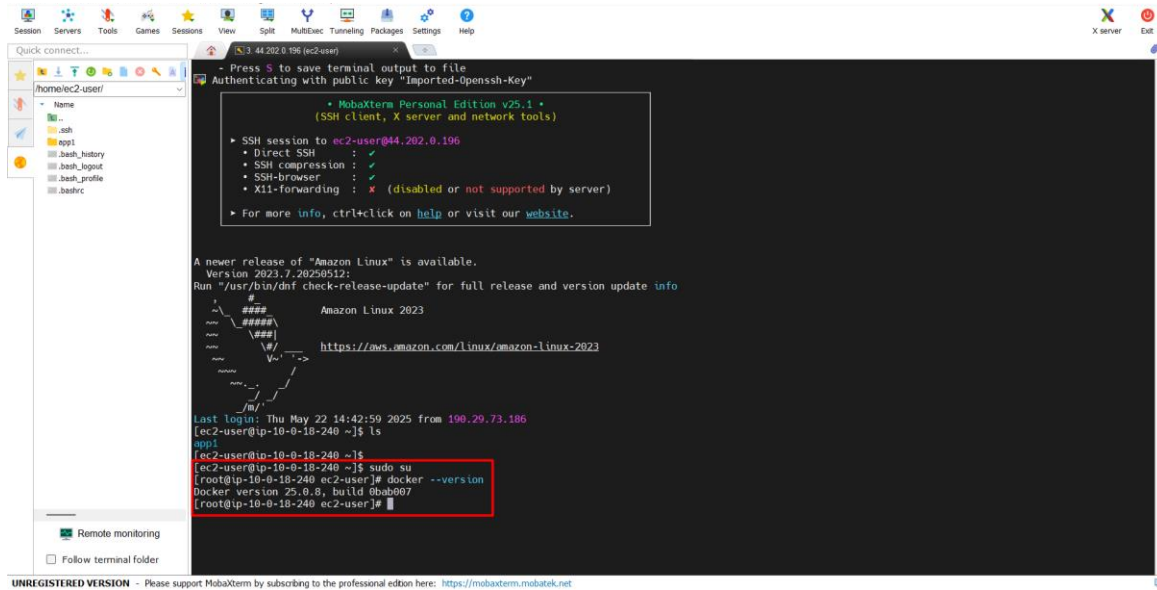
Video General

[Seminario Parte 1-20250507_171030-Grabación de la reunión.mp4](#)

Implementación Docker

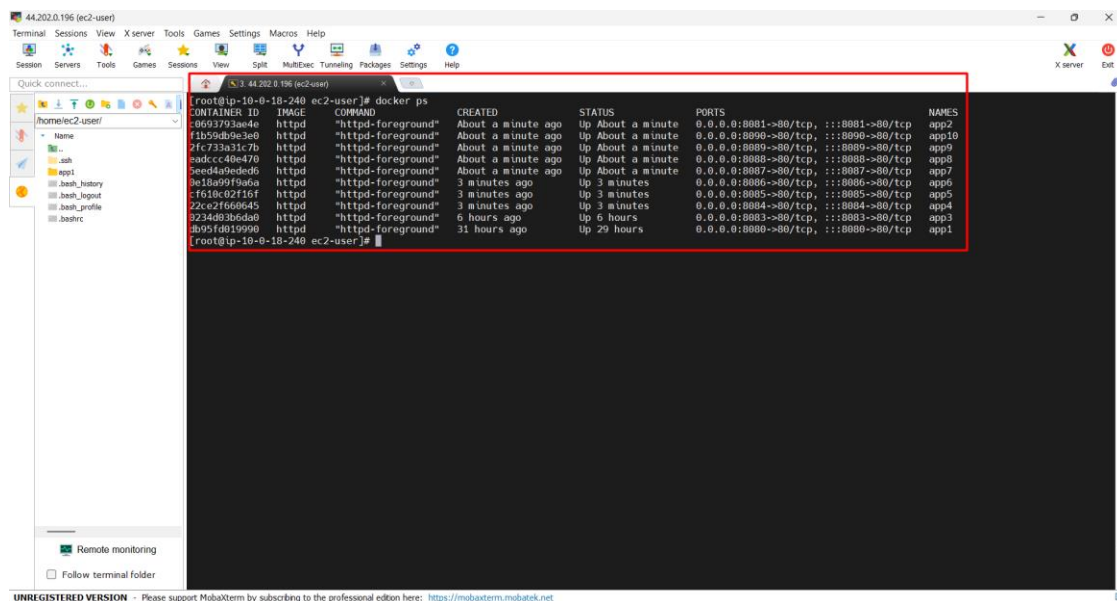
Instalación y puesta en marcha del servicio Docker

Figura 8 Inicio Servicio Docker



Ejecución de multiples contenedores

Figura 9 múltiples Contenedores



Generación de carga alto consumo de CPU

Figura 10 Consumo de CPU

```

[root@ip-10-0-18-240 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; preset: disabled)
   Active: active (running) since Wed 2025-05-21 13:34:40 UTC; 1 day 7h ago
     TriggersBy: ● docker.socket
       Docs: https://docs.docker.com
        Main PID: 1099356 (dockerd)
          Tasks: 169
        Memory: 118.9M
           CPU: 18.296s
      CGroup: /system.slice/docker.service
              └─1099356 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536
                └─1109241 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8880 -container-ip 172.17.0.2 -container-port 80
                └─1109355 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8883 -container-ip 172.17.0.3 -container-port 80
                └─1108368 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8083 -container-ip 172.17.0.3 -container-port 80
                └─1208691 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8884 -container-ip 172.17.0.4 -container-port 80
                └─1208696 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8084 -container-ip 172.17.0.4 -container-port 80
                └─1208982 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8885 -container-ip 172.17.0.5 -container-port 80
                └─1208987 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8085 -container-ip 172.17.0.5 -container-port 80
                └─1201259 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8886 -container-ip 172.17.0.6 -container-port 80
                └─1201274 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8086 -container-ip 172.17.0.6 -container-port 80
                └─1201324 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8887 -container-ip 172.17.0.7 -container-port 80
                └─1201333 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8087 -container-ip 172.17.0.7 -container-port 80
                └─1201919 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8888 -container-ip 172.17.0.8 -container-port 80
                └─1201924 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8088 -container-ip 172.17.0.8 -container-port 80
                └─1202205 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8889 -container-ip 172.17.0.9 -container-port 80
                └─1202210 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8089 -container-ip 172.17.0.9 -container-port 80
                └─1202492 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8890 -container-ip 172.17.0.10 -container-port 80
                └─1202497 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8090 -container-ip 172.17.0.10 -container-port 80
                └─1202855 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8881 -container-ip 172.17.0.11 -container-port 80
                └─1202859 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8081 -container-ip 172.17.0.11 -container-port 80

May 21 13:34:40 ip-10-0-18-240.ec2.internal dockerd[1099356]: time="2025-05-21T13:34:40.137242789Z" level=info msg="Daemon has completed initialization"
May 21 13:34:40 ip-10-0-18-240.ec2.internal dockerd[1099356]: time="2025-05-21T13:34:40.171960762Z" level=info msg="API listen on /run/docker.sock"
May 21 13:34:40 ip-10-0-18-240.ec2.internal systemd[1]: Started docker.service - Docker Application Container Engine.
May 21 16:25:47 ip-10-0-18-240.ec2.internal dockerd[1099356]: 2025/05/21 16:25:47 http: superfluous response.WriteHeader call from go.opentelemetry.io/contrib
May 21 16:27:08 ip-10-0-18-240.ec2.internal dockerd[1099356]: time="2025-05-21T16:27:08.733580596Z" level=info msg="Ignoring event" container=db95fd81998998
May 21 16:27:08 ip-10-0-18-240.ec2.internal dockerd[1099356]: time="2025-05-21T16:27:08.758618611Z" level=warning msg="failed to close stdin: task db95fd81998998"
May 22 16:22:33 ip-10-0-18-240.ec2.internal dockerd[1099356]: 2025/05/22 16:22:33 http: superfluous response.WriteHeader call from go.opentelemetry.io/contrib
May 22 16:22:34 ip-10-0-18-240.ec2.internal dockerd[1099356]: 2025/05/22 16:22:34 http: superfluous response.WriteHeader call from go.opentelemetry.io/contrib
May 22 16:27:03 ip-10-0-18-240.ec2.internal dockerd[1099356]: 2025/05/22 16:27:03 http: superfluous response.WriteHeader call from go.opentelemetry.io/contrib

[root@ip-10-0-18-240 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; preset: disabled)
   Active: active (running) since Wed 2025-05-21 13:34:40 UTC; 1 day 7h ago

```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

Figura 11 Conexión EC2

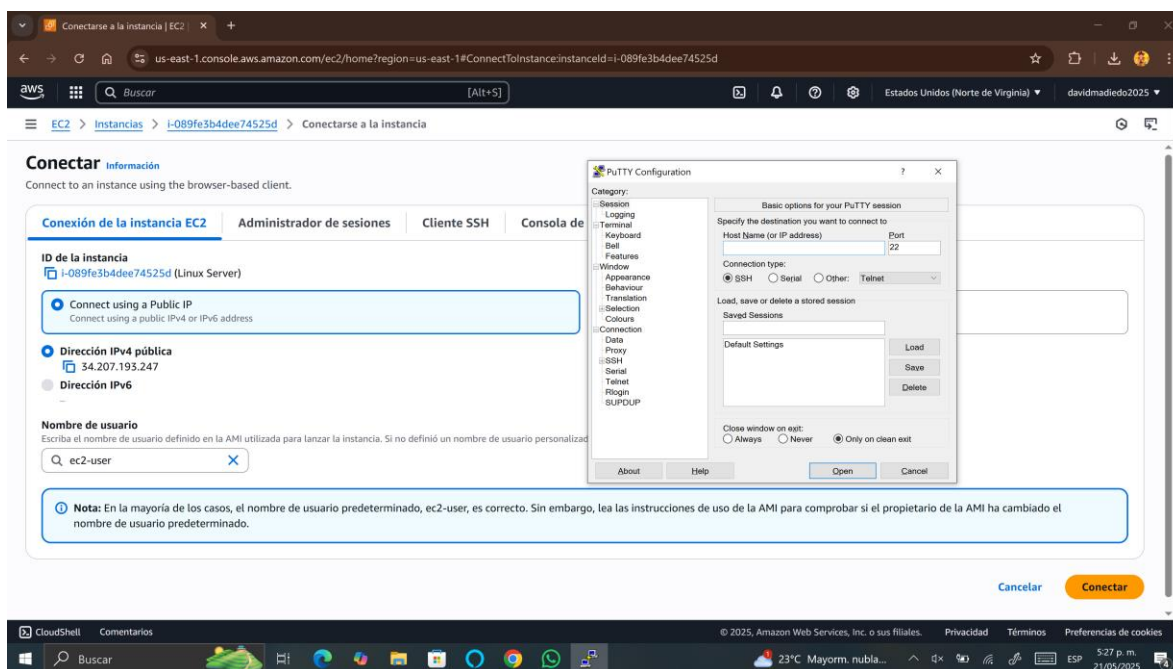


Figura 12 Ruta Conexión

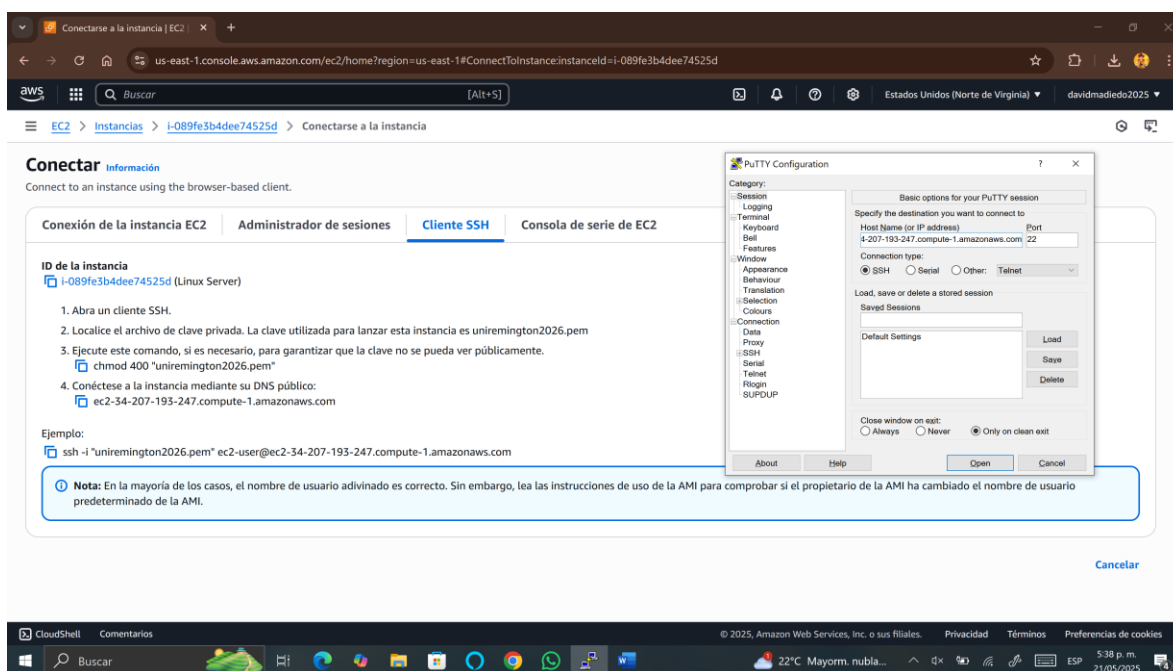


Figura 13 Key

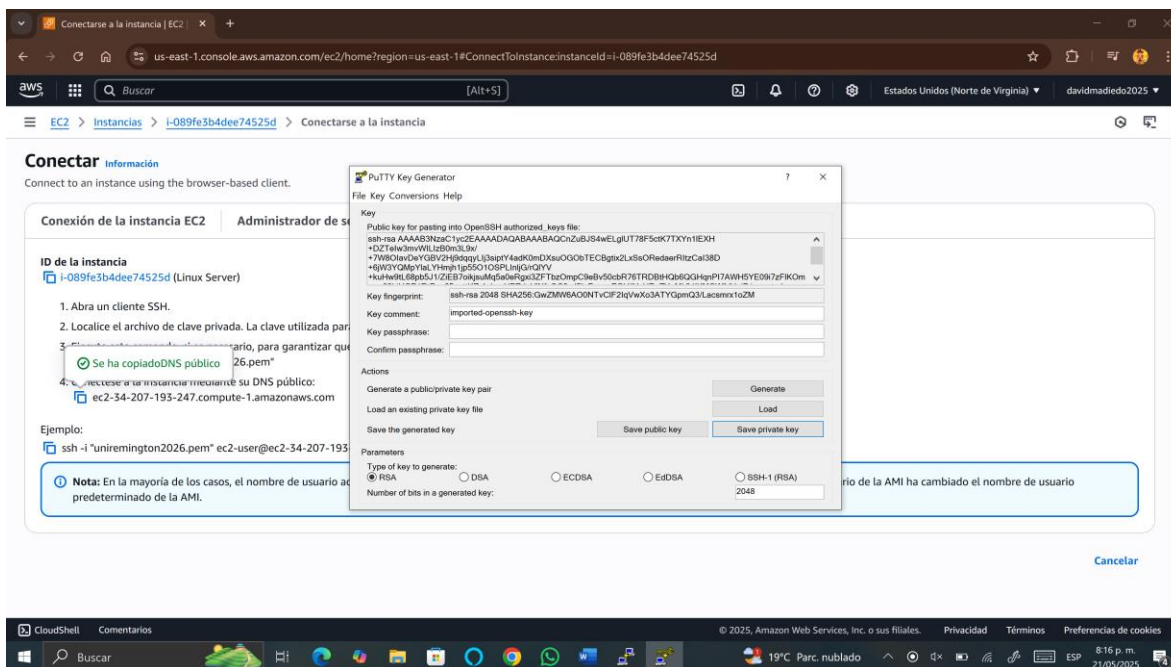


Figura 14 Autenticación

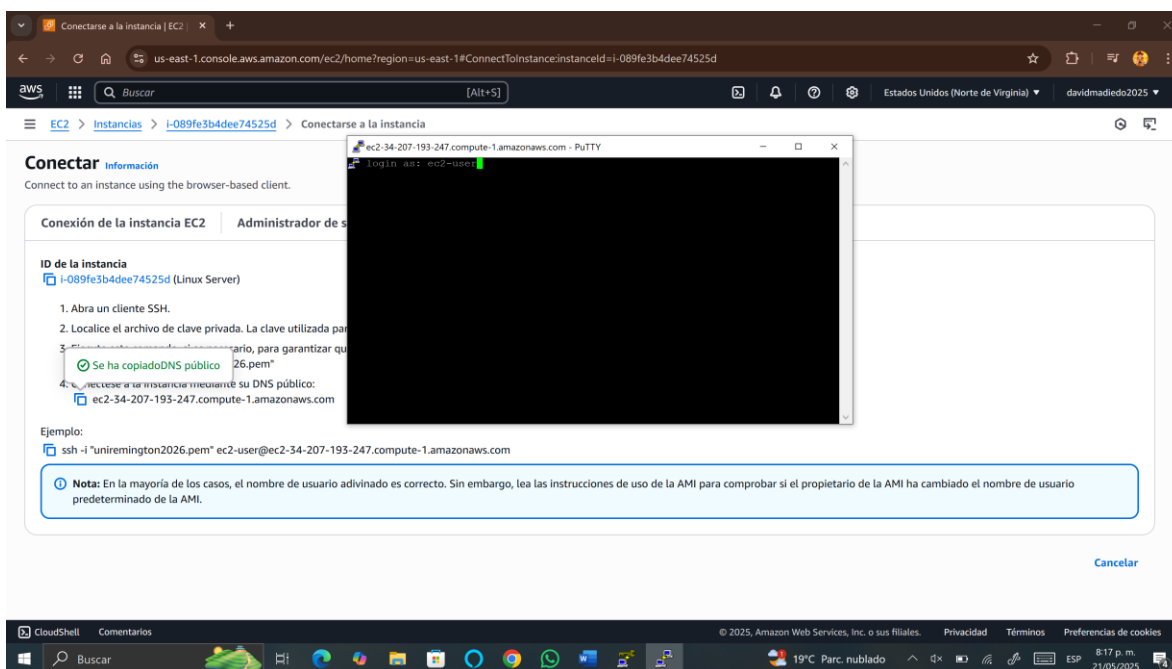


Figura 15 Conexión Completa

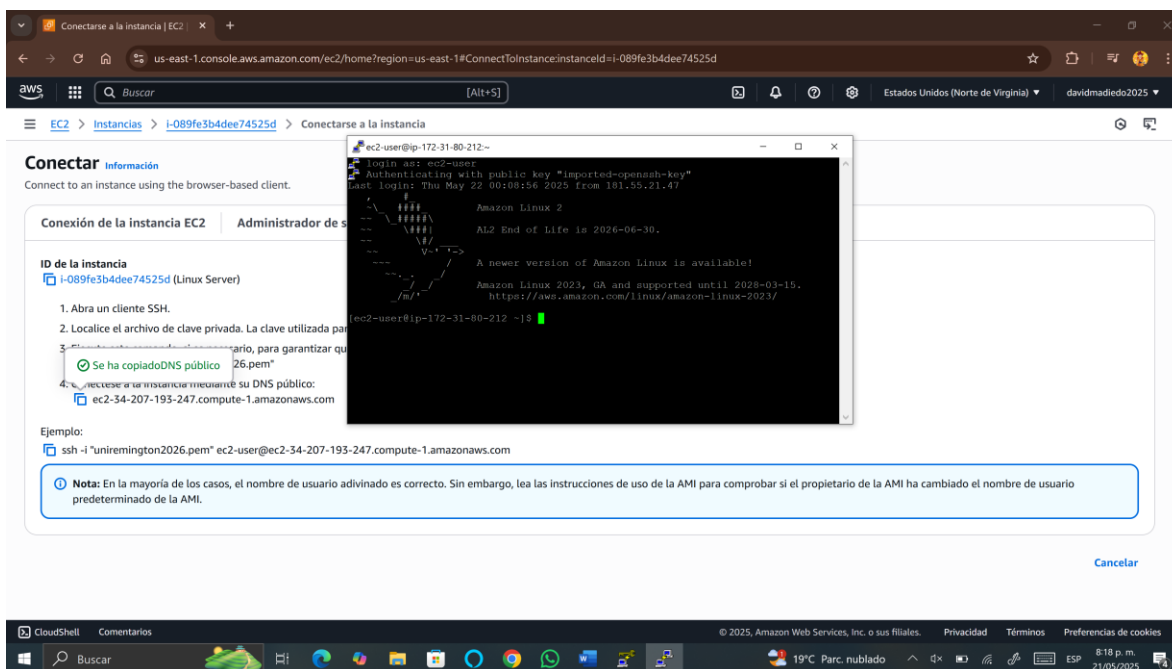


Figura 16 Actualización

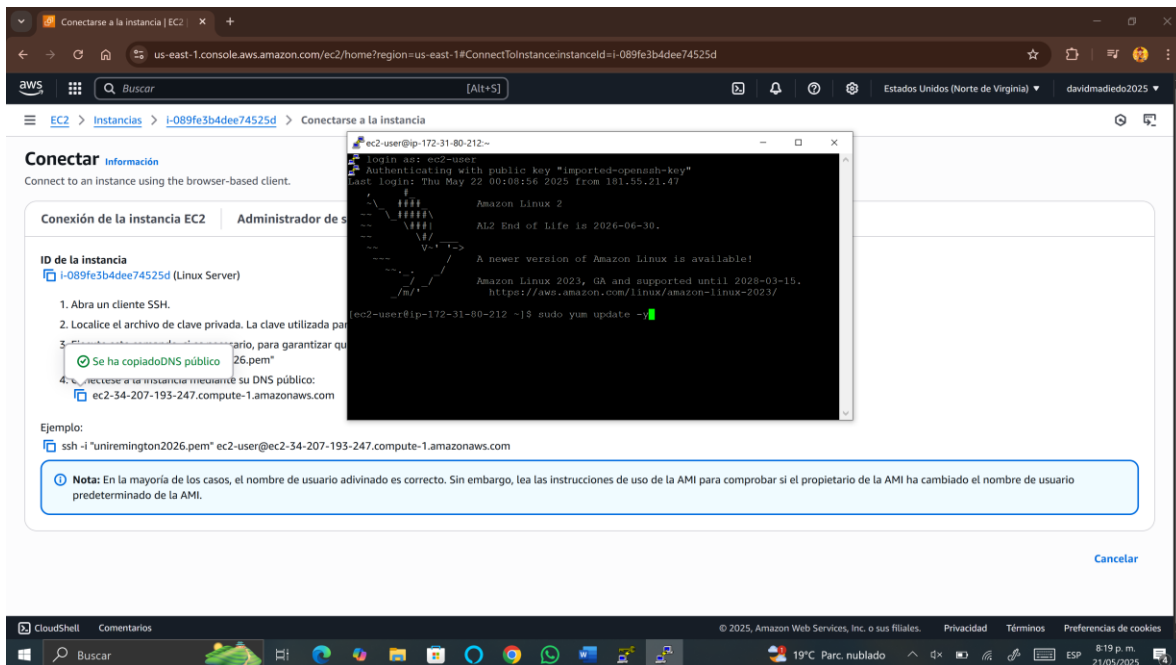


Figura 17 Instalación Paquete

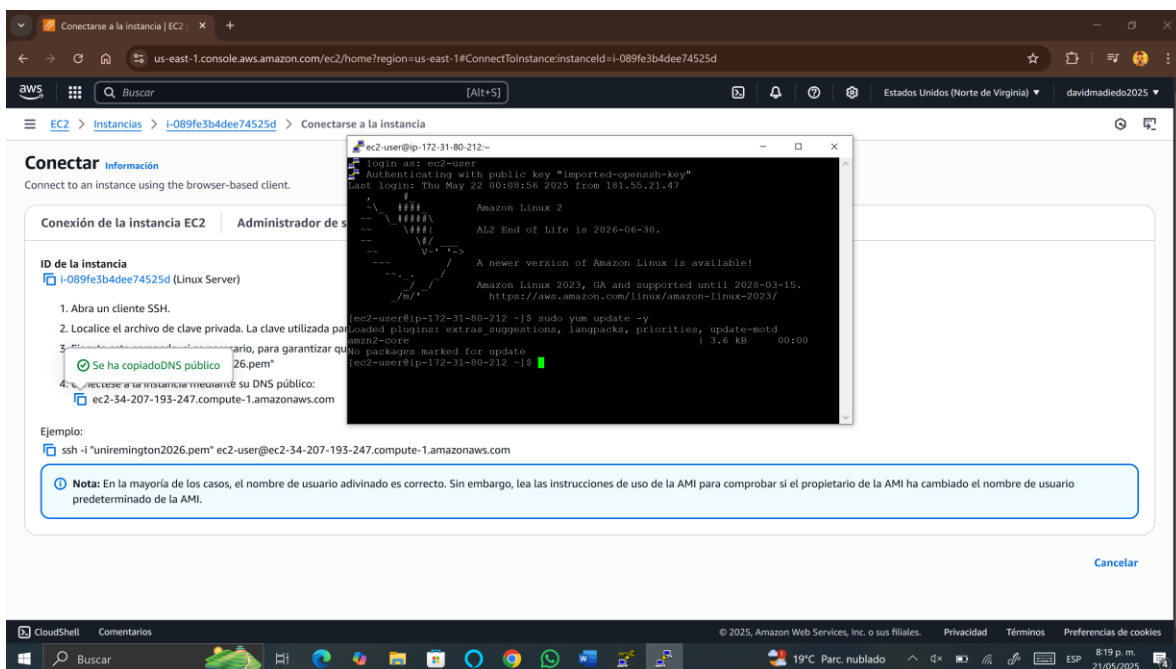


Figura 18 Aplicaciones

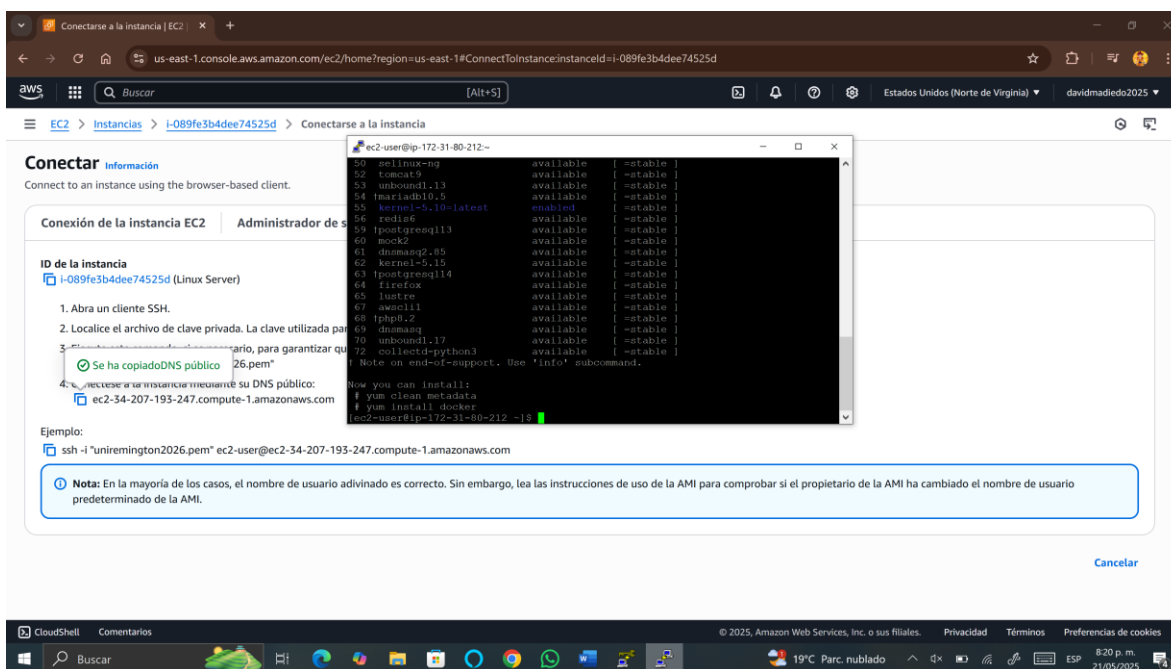


Figura 19 Docker

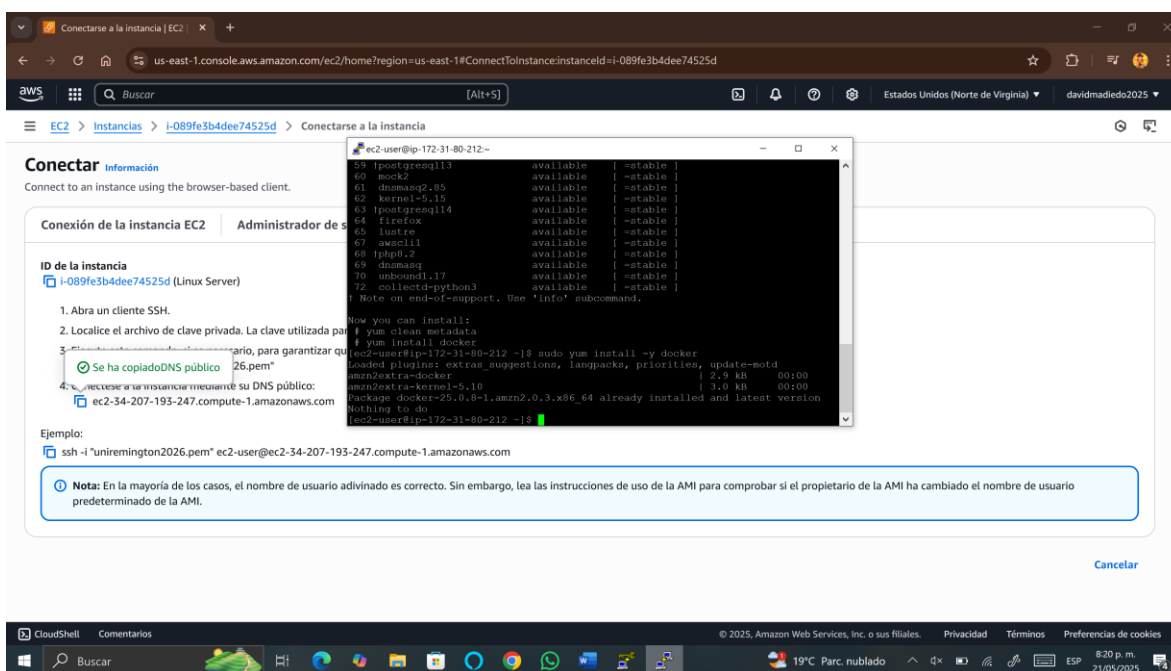


Figura 20 Validación Aplicación

The screenshot shows the AWS Management Console interface for connecting to an EC2 instance. The main window displays the 'Conectar' (Connect) dialog for instance `i-089fe3b4d`. The dialog includes a terminal window with the following output:

```

[ec2-user@ip-172-31-80-212 ~]$ docker version
Client:
Version:           25.0.8
API version:       1.44
Go version:        go1.23.8
Git commit:        0bab007
Build:             Tue Apr 15 21:42:34 2025
OS/Arch:           linux/amd64
Context:           default
Server:
Engine:
Version:           25.0.8
API version:       1.44 (minimum version 1.24)
Go version:        go1.23.8
Git commit:        7190f08
Build:             Tue Apr 15 21:43:05 2025
OS/Arch:           linux/amd64
Experimental:     false
Containerd:

```

Below the terminal, an example SSH command is provided: `ssh -i "uniremington2026.pem" ec2-user@ec2-34-207-193-247.compute-1.amazonaws.com`. A note states: "Nota: En la mayoría de los casos, el nombre de usuario adivinado es correcto. Sin embargo, lea las instrucciones de uso de la AMI para comprobar si el propietario de la AMI ha cambiado el nombre de usuario predeterminado de la AMI."

Figura 21 Instancias

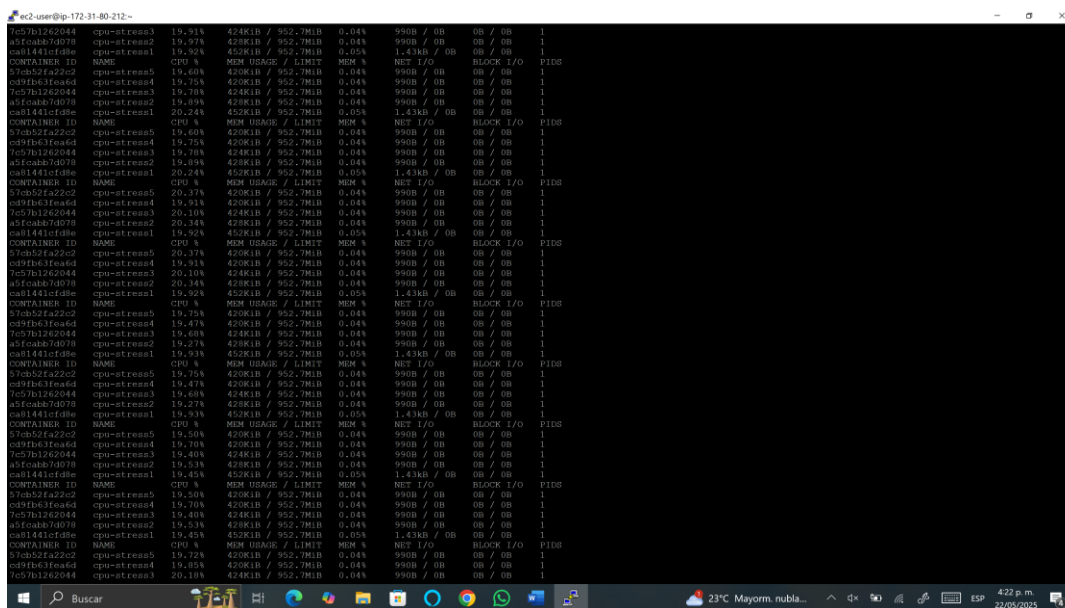
The screenshot shows the AWS Management Console 'Instances' page. The table below lists the instances:

Name	ID de la instancia	Estado de la instancia	Tipo de instancia	Comprobación de estado	Estado de la alarma	Zona de disponibilidad	DNS de la instancia
Linux Server	i-089fe3b4d74525d	En ejecución	t2.micro	2/2 comprobador	Ver alarmas +	us-east-1c	ec2-34-21...
Windows Server	i-08609db397b4c02f7	En ejecución	t2.micro	2/2 comprobador	Ver alarmas +	us-east-1d	ec2-54-9'

The detailed view for instance `i-089fe3b4d74525d (Linux Server)` shows the following metrics:

- Utilización de la CPU (%): 98.6
- Entrada de red (bytes): 467k
- Salida de red (bytes): 112k
- Paquetes de entrada de red: 393
- Paquetes de salida de red (recuento): 190
- Metadatos sin token (recuento): 1
- Uso de créditos de CPU (recuento): 4.42
- Saldo de créditos de CPU (recuento): 144

Figura 22 Log



Conclusiones

Los desarrollos y aplicaciones implementadas en la nube tienen la posibilidad de extender con capacidad como servicio facilitando su configuración física según requerimientos casi que en tiempo real

Los servidores implementados en AWS cuentan con alta disponibilidad y seguridad en poco tiempo de implementación con múltiples sistemas operativos

En AWS la seguridad perimetral cuenta con múltiples capas las cuales facilitan su implementación

Utilizar una herramienta como AWS aumenta la capacidad competitiva de cualquier proyecto o empresa en el mercado generando ahorros si se implementa con estudios técnicos con análisis de requerimientos.

Las herramientas suministradas en el diplomado permiten explorar de forma práctica y teórica temas modernos que facilitan y abren el mercado laboral por medio del conocimiento en una fuente segura

Las bases de la Ingeniería a nivel teórico permiten evaluar cual es el recurso que se debe contratar y los tiempos necesarios optimizando eficiencia y costo de los proyectos que se quieren realizar

Referencias

- AWS. (2025). *Amazon Web Services*. Obtenido de AWS:
<https://aws.amazon.com/ec2/free/>
- docker*. (2013-2025). Obtenido de <https://docs.docker.com/>
- Medina, A. A. (Mayo de 2018). *Udemy*. Obtenido de Introducción a Linux:
<https://www.udemy.com/course/instala-linux-en-tu-pc/learn/lecture/6963630?start=0#overview>
- MobaXterm*. (2008-2025). Obtenido de MobaXterm:
<https://mobaxterm.mobatek.net/documentation.html>
- Vásquez, G. (Noviembre de 2024). *Udemy*. Obtenido de Windows Server 2012 y Linux Ubuntu Server para principiantes: <https://www.udemy.com/course/windows-server-y-linux-para-principiantes/learn/lecture/9470508?start=0#overview>