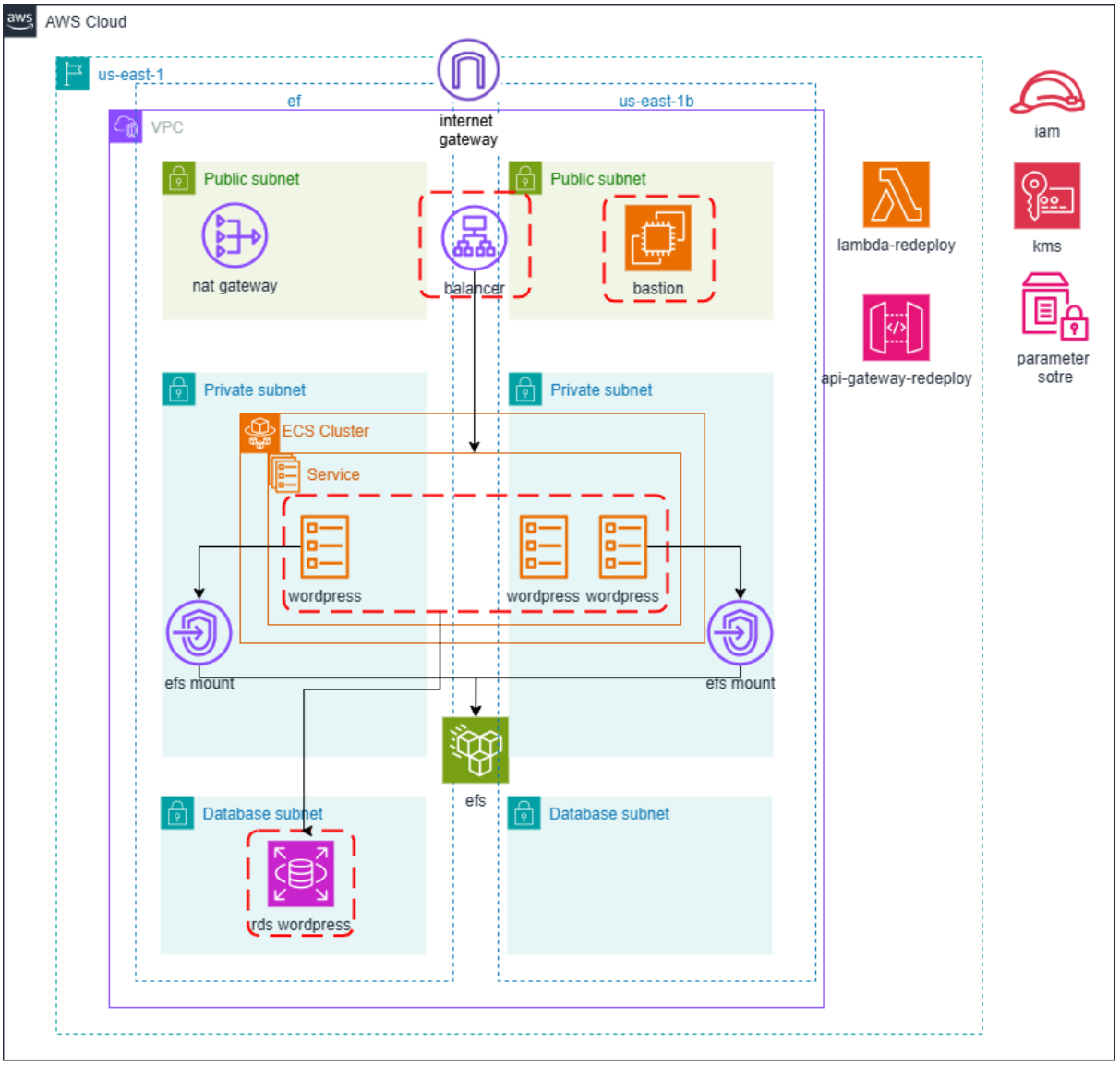
Prueba Técnica
Jorge Ignacio Morales

Prueba Técnica

Paso 1

Para este punto se propone la siguiente arquitectura:



La infraestructura fue desplegada utilizando Terraform. Se implementaron tres capas de subnets para separar lógicamente los recursos a nivel de red:

1. **Subnets públicas**: Destinadas a los recursos que deben ser accesibles desde internet, como el balanceador de carga (ALB) y una instancia bastión, que permite conexiones seguras hacia los servicios internos, incluidos la base de datos y el clúster ECS.
2. **Subnets privadas**: Utilizadas para los servicios internos, como las tareas de ECS Fargate.
3. **Subnets para almacenamiento**: Asignadas al servicio Amazon EFS, configurado para compartir archivos entre las tareas de ECS.

Para optimizar costos, especialmente dado que no es un ambiente productivo, se implementó un único NAT Gateway en lugar de uno por zona de disponibilidad. Esto proporciona conectividad a internet para los recursos privados, como las tareas de ECS.

El servicio de WordPress se implementó en un clúster de ECS Fargate con un servicio asociado a un balanceador de carga (ALB) que distribuye el tráfico. El servicio está configurado con tres tareas por defecto para manejar solicitudes. Los archivos compartidos entre las tareas se almacenan en Amazon EFS, que está configurado con dos puntos de montaje distribuidos entre las subnets privadas.

Para la capa de datos, se implementó una base de datos RDS MySQL, protegida mediante **security groups**. El acceso está restringido únicamente a las tareas de ECS Fargate y a la instancia bastión, lo que garantiza un control estricto de las conexiones.

En cuanto a la gestión de credenciales, se utilizó AWS Systems Manager Parameter Store para almacenar y manejar de forma segura las credenciales de la base de datos. Además, para la encriptación en reposo de los datos, tanto en EFS como en RDS, se configuraron claves gestionadas por AWS KMS.

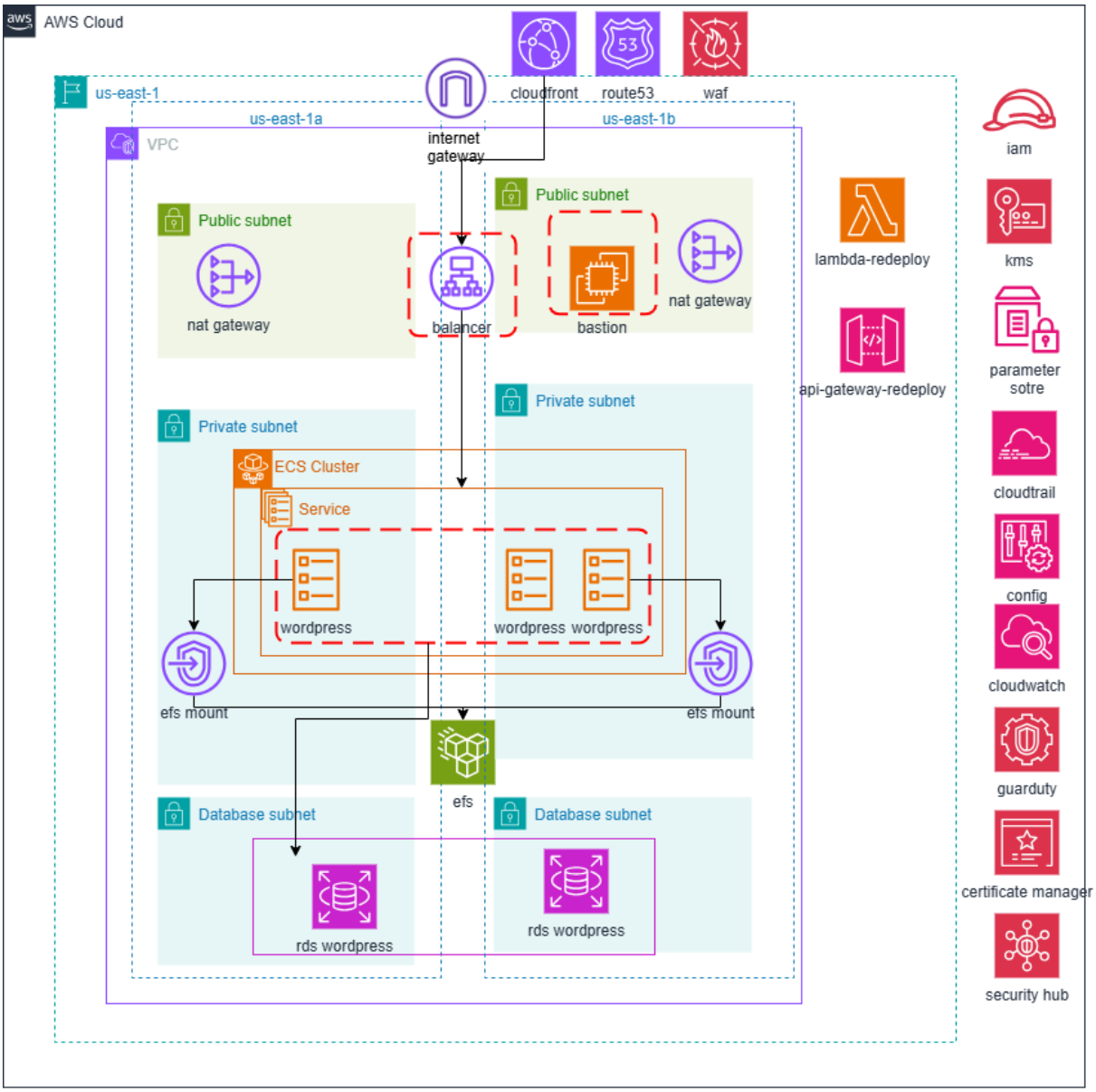
Propuesta para un ambiente productivo

Aunque la solución implementada cumple con los requisitos iniciales, se identificaron áreas de mejora para fortalecer la seguridad, la resiliencia y la alta disponibilidad. Estas mejoras incluyen:

1. **Seguridad**:
    - Incorporar **AWS WAF** para proteger la aplicación contra ataques como inyección SQL o ataques DDoS.
    - Utilizar **AWS Certificate Manager** para manejar certificados SSL/TLS, asegurando conexiones cifradas en el balanceador de carga.
    - Habilitar **AWS Security Hub** para centralizar la evaluación de la seguridad y monitorear configuraciones basadas en estándares de seguridad.
    - Implementar **AWS GuardDuty** para detectar y responder a amenazas potenciales.
2. **Resiliencia**:
    - Aumentar la cantidad de subnets públicas y privadas, distribuyendo los recursos entre al menos tres zonas de disponibilidad (AZs) para mayor tolerancia a fallos.
    - Configurar un segundo NAT Gateway en otra AZ para evitar puntos únicos de fallo.

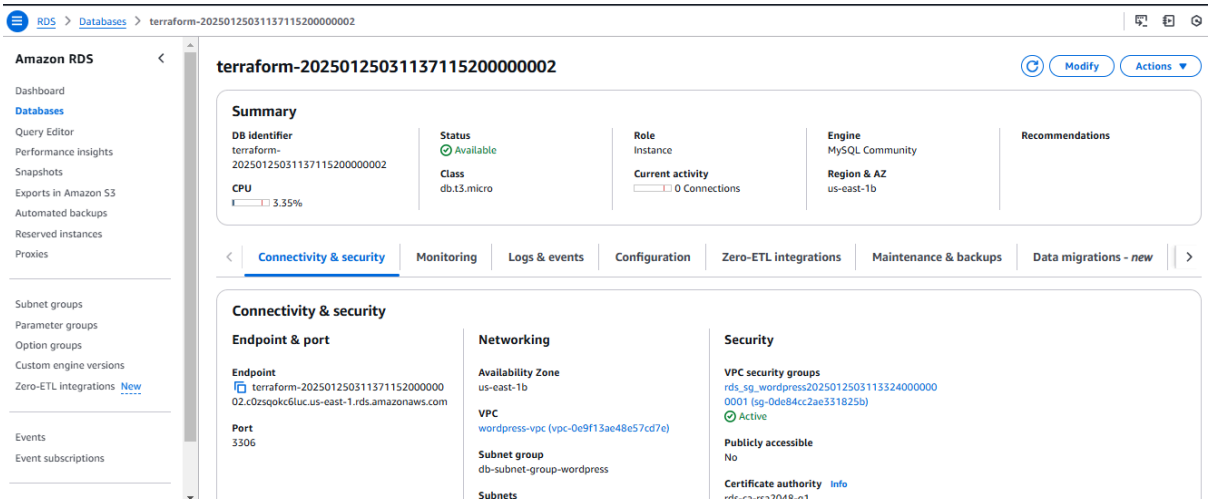
3. **Monitoreo y gobernanza**:
    - Habilitar **AWS Config** para monitorear los cambios en los recursos y garantizar la conformidad con las políticas definidas.
    - Implementar **Amazon CloudWatch** con métricas personalizadas para monitorear el desempeño de las tareas ECS y el uso de recursos.
4. **Disponibilidad**:
    - Configurar autoescalado en las tareas ECS para manejar picos de tráfico.
    - Habilitar réplicas de lectura en RDS para mejorar la disponibilidad y el rendimiento de las consultas.

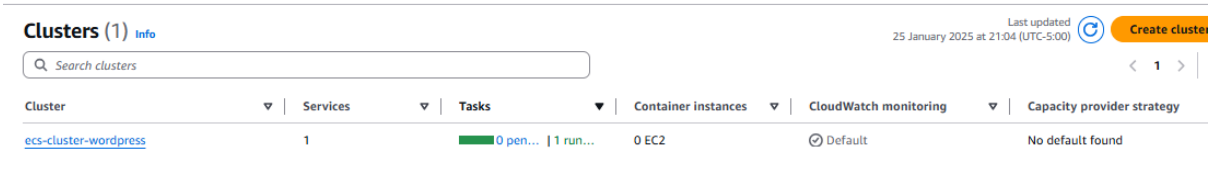
Estas recomendaciones no fueron implementadas en la solución inicial debido a restricciones de tiempo y recursos, pero constituyen una guía sólida para una implementación en un ambiente productivo, asegurando que la solución sea escalable, segura y confiable.

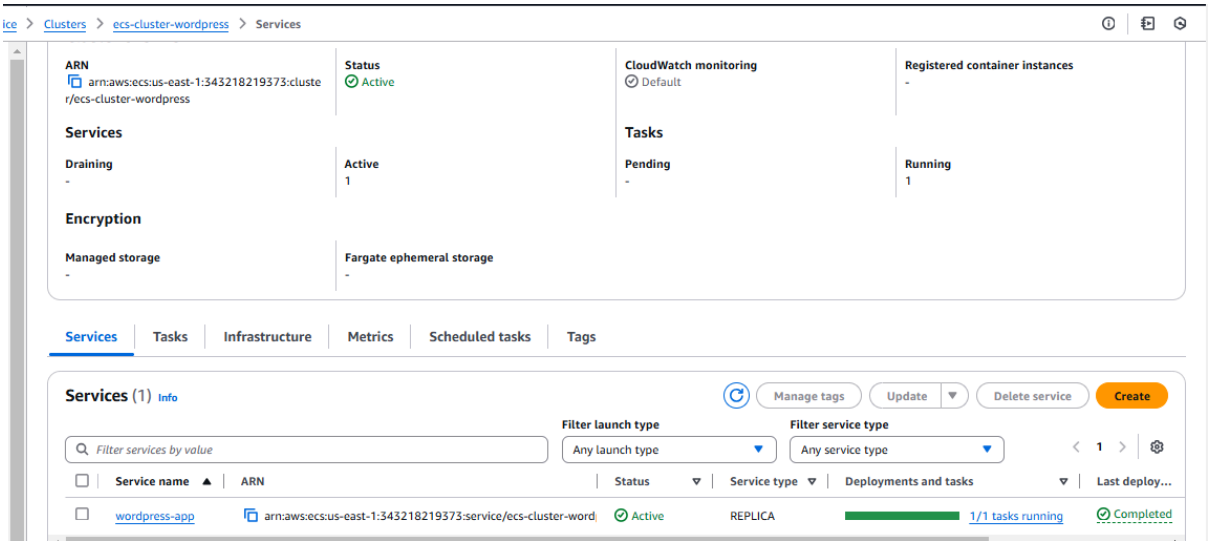


Se adjunta el archivo con el código comprimido de terraform con el que se desplegó la infraestructura

Infraestructura Creada en AWS







wordpress-app Info

Last updated
25 January 2025 at 21:04 (UTC-5:00)

Update service | Delete service

Service overview Info

Status	Tasks (1 Desired)	Task definition: revision	Deployment status
Active	0 pending | 1 running	wordpress:4	Success

Health and metrics | Tasks | Logs | Deployments | Events | Configuration and networking | Service auto scaling | Tags

Status Info

Service name	Service ARN	Deployments current state	Created at
wordpress-app	arn:aws:ecs:us-east-1:343218219373:service/ecs-cluster-wordpress/wordpress-app	1 Completed task	24 January 2025 at 21:39 (UTC-5:00)

Task definitions > wordpress > Revision 4 > Containers

wordpress:4

Deploy | Actions | Create new revision

Overview Info

ARN	Status	Time created	App environment
arn:aws:ecs:us-east-1:343218219373:task-definition/wordpress:4	ACTIVE	25 January 2025 at 00:03 (UTC-5:00)	Fargate

Task role	Task execution role	Operating system/Architecture	Network mode
ecs_task_assume	ecs_task_execution_role	-	awsvpc

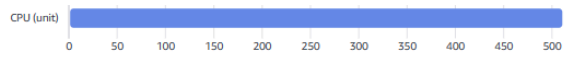
Fault injection
Turned off

Containers | JSON | Task placement | Volumes (0) | Requires attributes | Tags

Task size

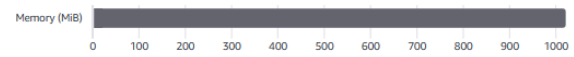
Task CPU
512 units (0,5 vCPU)

Task CPU maximum allocation for containers



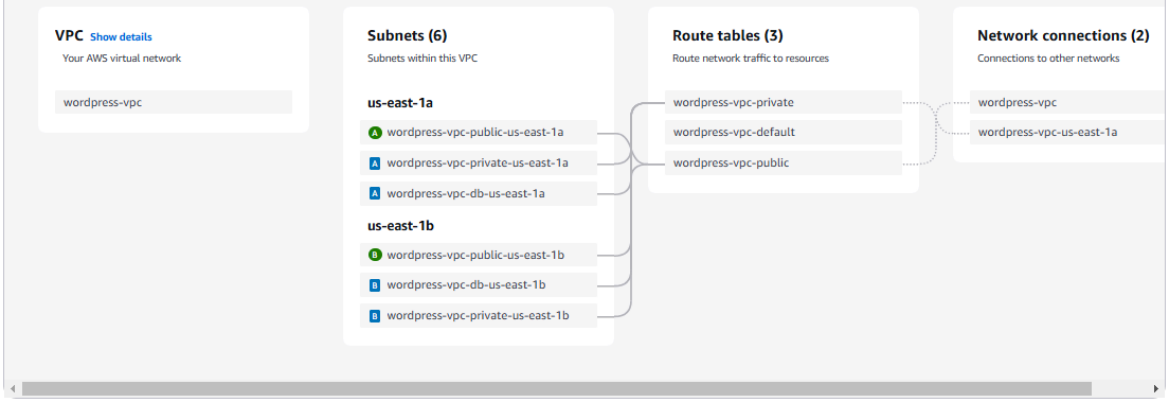
Task memory
1024 MiB (1 GB)

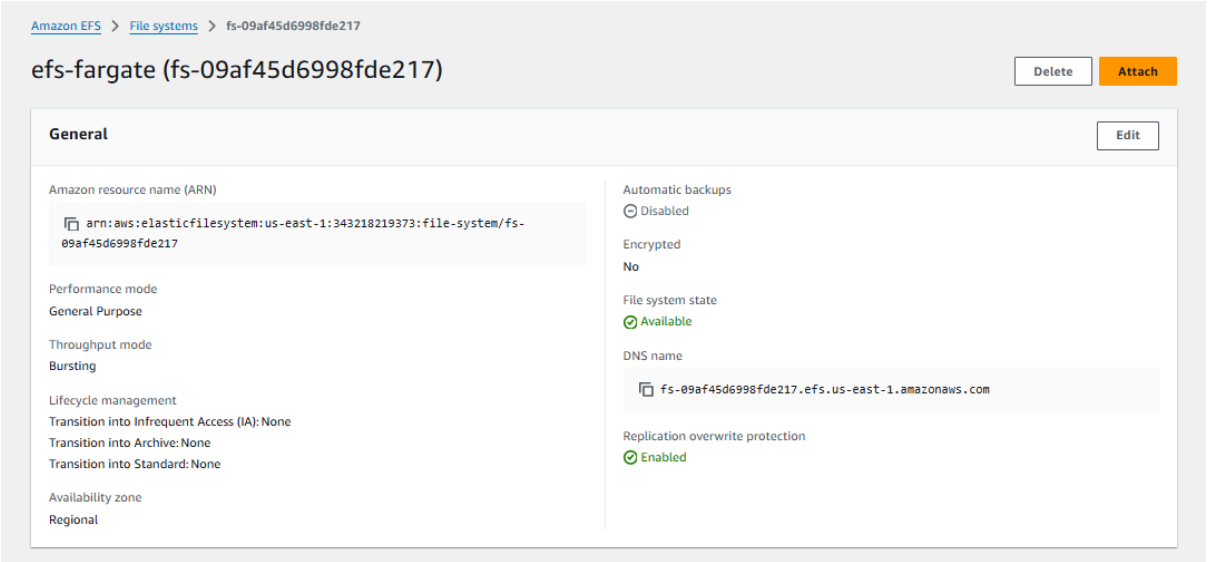
Task memory maximum allocation for container memory reservation



Resource map | CIDRs | Flow logs | Tags | Integrations

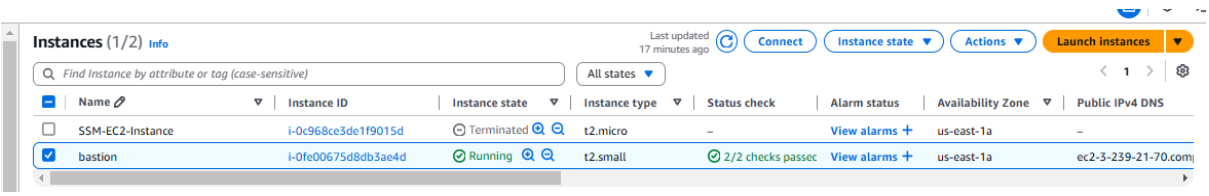
Resource map Info



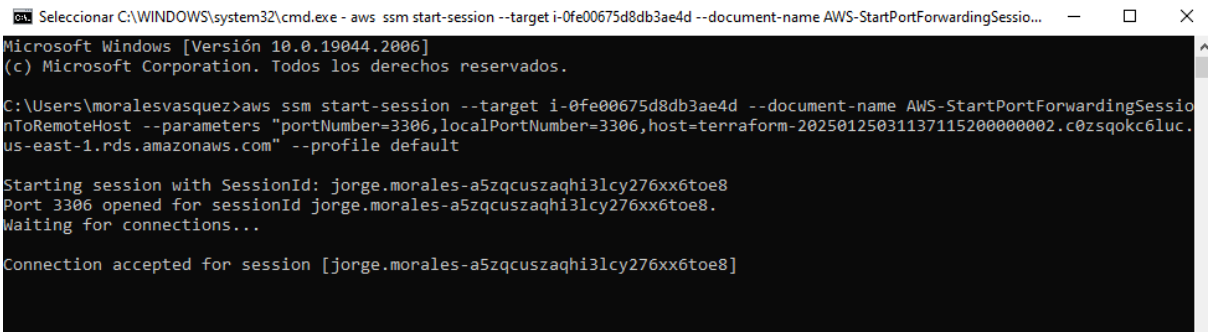


Bastion:

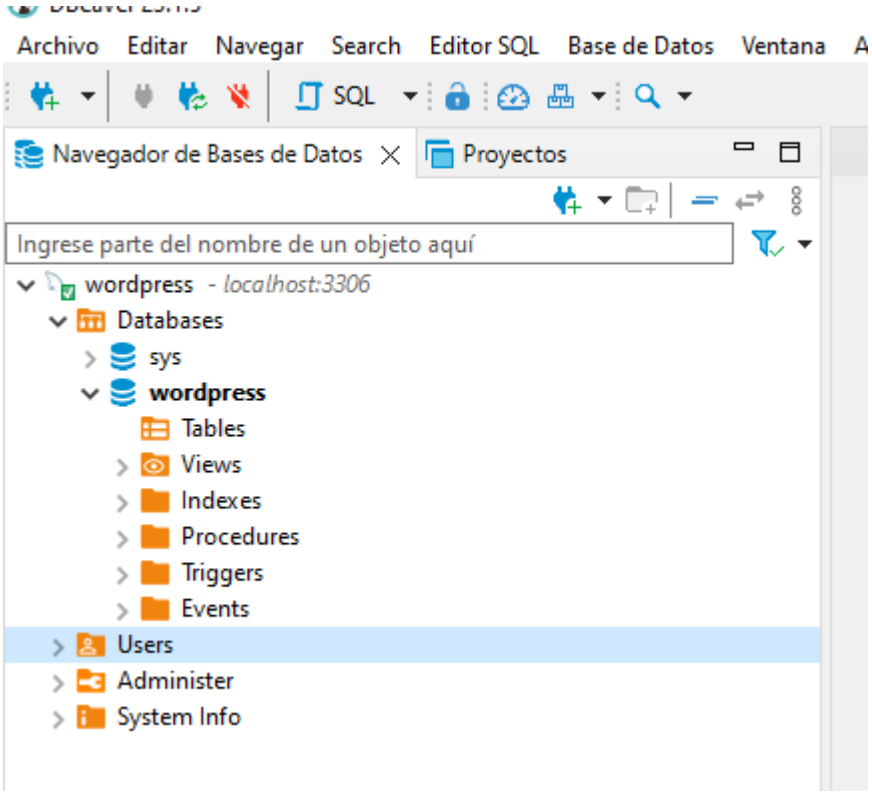
La conexión a la base de datos se realizó utilizando el plugin de Session Manager, el cual crea un túnel seguro entre la máquina local y la instancia desplegada. Este enfoque permite realizar el reenvío de puertos hacia la base de datos con varios beneficios clave. En primer lugar, elimina la necesidad de que la instancia sea pública, lo que mejora la seguridad al reducir la superficie de ataque. En segundo lugar, genera registros detallados de las sesiones, lo que facilita la trazabilidad y auditoría. Por último, el acceso se gestiona mediante roles de IAM y el AWS CLI configurado en la máquina local del usuario, lo que garantiza una autenticación segura y eficiente.



Creación de la conexión con el plugin de session manager



Conectado a la base de datos



Python

Paso 2 - Estrategia de automatización.

Para abordar el desafío de reiniciar diariamente todas las tareas de los servicios en los 1000 clústeres ECS distribuidos en 50 cuentas de AWS, propongo la siguiente solución automatizada:

1. **Accesos y permisos**
    - Se va a implementar una cuenta principal en la cual se va a administrar la automatización y se brindaran los permisos a nivel de roles, en las cuentas destino se debe brindar
2. **Identificación y registro de servicios**
    - Inicialmente, se implementará una función Lambda que ejecutará un script apoyado en AWS Config. Este script tendrá como objetivo identificar todos los clústeres y los servicios asociados en cada cuenta de AWS.
    - Los datos recopilados se almacenarán en una tabla de DynamoDB, la cual incluirá una columna que indique la tanda asignada para el reinicio del servicio. Las tandas se determinarán dividiendo equitativamente la cantidad total de servicios entre las 6 horas disponibles (00:00 a 06:00). Este proceso de identificación y asignación se ejecutará diariamente antes del inicio del ciclo de reinicios.
3. **Estandarización y control mediante tags**
    - Se solicitará a los responsables de los servicios que implementen un **tag estandarizado** llamado `ecs_task_daily_restart`. Este tag deberá contener como valor la última fecha y hora de reinicio del servicio.

    - Los servicios que no tienen este tag serán identificados mediante el script mencionado anteriormente, y se enviará una notificación a los encargados para que asignen el tag correspondiente. Esto permitirá un mejor control del proceso de reinicio y garantizará la inclusión de todos los servicios en la automatización.
4. **Orquestación del reinicio en tandas**
    - Se configurará un **Amazon EventBridge** para desencadenar una **AWS Step Function** que será responsable de ejecutar los reinicios en tandas.
    - La Step Function tomará los datos almacenados en DynamoDB y procesará los servicios de manera ordenada según la tanda asignada. Esto asegurará que no se ejecuten todos los reinicios al mismo tiempo, distribuyendo la carga de trabajo de forma eficiente durante la ventana de tiempo establecida.
5. **Notificación y monitoreo**
    - Para facilitar el monitoreo, se configurarán métricas y alarmas en **Amazon CloudWatch**. Estas alertas permitirán detectar fallos o retrasos en el proceso y notificar a los equipos responsables mediante **Amazon SNS**.
6. **Manejo de servicios no estandarizados**
    - Los servicios desplegados manualmente y que no cuenten con la estandarización requerida serán manejados mediante notificaciones específicas para los encargados, quienes deberán tomar las acciones necesarias para incluirlos en el flujo automatizado.

Con esta solución, se asegura un proceso eficiente, controlado y escalable para reiniciar las tareas de todos los servicios ECS, minimizando riesgos y asegurando la continuidad operativa en todos los clústeres.

Paso 3 – Airflow

Se adjunta el archivo “hello-nequi-dag.py”