Penguin On The Way!

Secure & Highly Available AWS Infrastructure

Cloud Architecture & Infrastructure Documentation

Autor: Carlos Fernández Salazar

Información de contacto

Email: info@carlosfdez.com

Website: www.carlosfdez.com

LinkedIn: www.linkedin.com/in/carlosfernandezsalazar

GitHub: github.com/CarlosFdezSalazar

Índice

Introducción

Introducción al proyecto

Objetivos

Alcance del proyecto

Contexto y limitaciones del entorno (AWS Academy Learner Lab)

Visión general de la arquitectura

Introducción al proyecto

El proyecto ***"Penguin On The Way!"*** Se plantea como un caso práctico de arquitectura cloud con el objetivo de demostrar mis habilidades en diseño, implementación y operación de infraestructura en AWS, orientado a simular un entorno real de producción, aplicando principios del AWS Well-Architected Framework..

La finalidad de este proyecto no es comercial, sino **demostrar, adquirir y practicar competencias técnicas y de diseño de infraestructuras seguras, resilientes y escalables en AWS**, incluyendo la capacidad para:

- Diseñar infraestructuras seguras, escalables y altamente disponibles.
- Aplicar buenas prácticas de segmentación de red, gestión de accesos y políticas de seguridad.
- Implementar Infraestructura como Código (IaC) y pipelines de despliegue automatizado.
- Configurar monitorización, alertas y resiliencia ante fallos, asegurando la continuidad de los servicios.
- Creación y ejecución de scripts con bash
- Establecer conexiones entre aplicaciones hosteadas en servidores (EC2 en este caso) y servicios de AWS

La elección de una plataforma de gestión de tickets se limita a un **caso de uso representativo**, permitiendo centrar el esfuerzo en la infraestructura subyacente y en la demostración de competencias cloud, más que en el desarrollo de la aplicación en sí.

Debido a la naturaleza de la situación planteada, **agradezco todo tipo de feedback** con intención de mejora. Estaré encantado de escuchar y atender a cualquier compañero del sector a través de las vías de contacto presentes en este documento.

Objetivos

Como se mencionó anteriormente, el proyecto tiene como finalidad principal **demostrar competencias en el diseño, implementación y operación de infraestructura cloud en AWS**, aplicando buenas prácticas de arquitectura, seguridad y automatización. Los objetivos específicos incluyen:

- **Diseñar una infraestructura segura y segmentada**, utilizando VPCs, subnets públicas y privadas, Security Groups, NACLs e IAM Roles, siguiendo el principio de mínimo privilegio.

- **Garantizar alta disponibilidad y resiliencia**, mediante la implementación de arquitecturas multi-AZ, Auto Scaling Groups para EC2 y RDS en configuración Multi-AZ con failover automático.
- **Implementar monitoreo, logging y alertas centralizados en CloudWatch y S3**, configurando notificaciones mediante SNS para asegurar la observabilidad y operación continua de la plataforma.
- **Aplicar Infraestructura como Código (IaC) con CloudFormation**, creando despliegues reproducibles y parametrizables, y modularizando stacks para VPC, seguridad, aplicación y bases de datos.
- **Documentar decisiones de arquitectura y operación,** justificando cada elección de servicio y configuración para evidenciar capacidad de análisis y diseño profesional.
- Implementar **estrategias de recuperación y backups**, asegurando persistencia de datos, snapshots automáticos y procedimientos de failover ante desastres.
- **Registrar los tickets de usuario en la base de datos RDS**, demostrando la integración segura entre la capa de aplicación y la base de datos.
- **Enviar notificaciones automáticas de tickets mediante SNS** a los administradores, garantizando la visibilidad de los eventos generados en la plataforma..

Alcance del proyecto

El alcance del proyecto ***"Penguin On The Way!"*** se centra en el **diseño, implementación y operación de una infraestructura cloud segura, escalable y altamente disponible en AWS**, empleando buenas prácticas de arquitectura.

Dentro de este proyecto se incluyen los siguientes elementos:

- **Infraestructura de red**: creación de una VPC multi-AZ, con subnets públicas y privadas, rutas correctamente configuradas, NAT Gateway y segmentación mediante Security Groups y NACLs.
- **Capa de aplicación**: despliegue de instancias EC2 en Auto Scaling Groups, detrás de un Application Load Balancer, como soporte para la ejecución de la aplicación de gestión de tickets. Insisto en mencionar que la aplicación se utiliza únicamente como **caso de uso** para demostrar el funcionamiento de la

infraestructura, no como demostración de cómo diseñar una aplicación profesional.

- **Base de datos**: configuración de Amazon RDS en modo Multi-AZ, con failover automático, cifrado y backups programados para garantizar persistencia y disponibilidad de los datos.
- **Notificaciones y automatización de eventos:** configuración de SNS para que los eventos generados por la aplicación (creación de tickets en RDS) envíen directamente correos electrónicos a los administradores de la plataforma, asegurando comunicación automática y notificación en tiempo real sobre nuevos tickets sin necesidad de funciones Lambda intermedias.
- **Seguridad y control de accesos**: gestión de credenciales mediante Parameter Store, IAM Roles y políticas de mínimo privilegio, así como HTTPS en el ALB para comunicación segura.
- **Observabilidad y monitoreo**: centralización de logs en CloudWatch o S3, métricas de rendimiento y alarmas configuradas mediante SNS para asegurar operación continua y detección temprana de incidentes
- **Despliegue reproducible**: implementación de Infraestructura como Código mediante CloudFormation.
- **Pruebas de resiliencia y continuidad**: validación de Auto Scaling, failover de RDS, actualización de AMIs sin downtime y simulación de caída de AZ para verificar la disponibilidad multi-AZ de la infraestructura.

Contexto y limitaciones del entorno (AWS Academy Learner Lab)

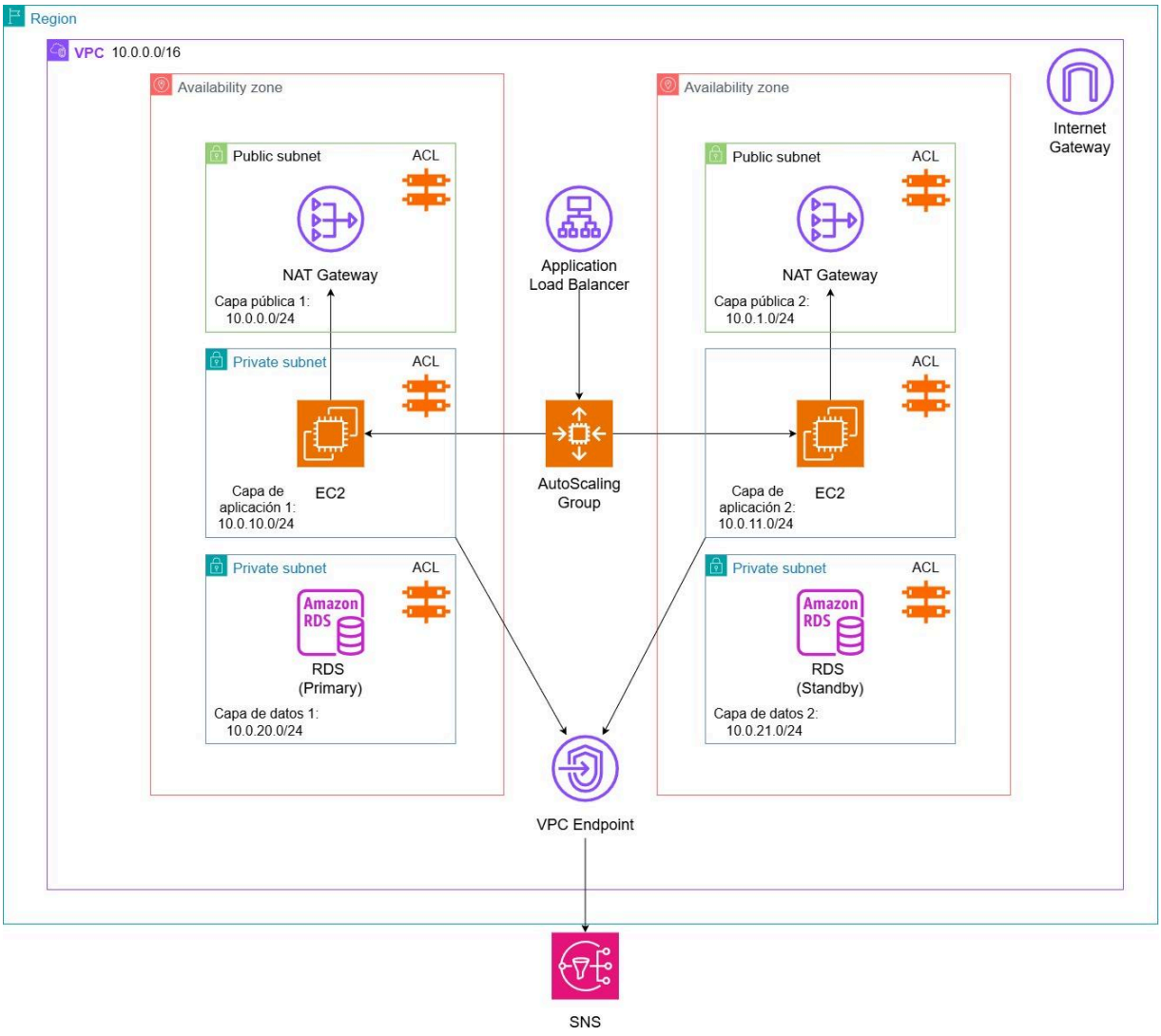
Este proyecto se realiza utilizando **AWS Academy**, concretamente el recurso **AWS Academy Learner Lab**, un entorno de sandbox en el que no todos los servicios y funcionalidades de AWS están disponibles.

En este contexto, algunas configuraciones o mejoras podrán **plantearse de manera teórica**, debido a la imposibilidad de implementarlos en el entorno sandbox mencionado.

Visión general de la arquitectura

La explicación detallada de estos componentes, su configuración y motivación se desarrolla más adelante en este documento.

https://github.com/CarlosFdezSalazar/Penguin-On-the-Way- (diagrama-POTW.drawio)



Documentación de decisiones de diseño

Justificación de servicios y configuraciones

Selección de tipos de instancia y tamaño de RDS/EC2

Principios de arquitectura aplicados (seguridad, resiliencia, escalabilidad)

Documentación de decisiones de diseño

En esta sección se recogen las **principales decisiones de diseño adoptadas** a lo largo del desarrollo de la infraestructura y la aplicación de Penguin On The Way!, así como la justificación técnica detrás de cada una de ellas.

El objetivo es no solo describir qué se ha implementado, sino **explicar por qué se han elegido determinadas soluciones frente a otras**, teniendo en cuenta aspectos como la escalabilidad, la seguridad, la resiliencia, el coste y las buenas prácticas en entornos cloud.

Esta documentación permite **aportar contexto al diseño de la arquitectura, facilitar su comprensión y servir como referencia para futuras mejoras o adaptaciones del sistema**, alineándose con un enfoque profesional propio de entornos reales de producción.

Justificación de servicios y configuraciones

La selección de servicios y configuraciones en este proyecto se ha realizado siguiendo criterios de buenas prácticas en entornos cloud, **priorizando escalabilidad, alta disponibilidad, seguridad y facilidad de mantenimiento.**

Si bien existen soluciones PaaS más óptimas para desplegar una aplicación web como la de Penguin On The Way! (por ejemplo, Elastic Beanstalk o App Runner), se ha optado por el uso de instancias EC2 gestionadas mediante Auto Scaling y Launch Templates. Esta decisión busca demostrar competencias propias de un arquitecto **AWS Associate**, así como habilidades en **Linux, Bash y administración de servidores**, gestionando de manera directa el despliegue, la configuración de Nginx, Gunicorn y la aplicación Flask.

El resto de la arquitectura se apoya en servicios gestionados de AWS, como **RDS para la base de datos y Application Load Balancer para distribución de tráfico**, garantizando tanto eficiencia operativa como integración con funcionalidades nativas (backups automáticos, balanceo de carga y alertas).

La **separación en capas (pública, aplicación y datos)** mediante subnets y reglas de seguridad refuerza el aislamiento y la seguridad del sistema. Además, la incorporación de Auto Scaling y servicios como CloudWatch, SNS y CloudTrail asegura observabilidad, alertado y auditoría, completando una arquitectura alineada con escenarios de producción.

El detalle de cada componente y su configuración se encuentra en la sección de **componentes de infraestructura.**

Selección de tipos de instancia y tamaño de RDS/EC2

Para este proyecto se ha optado por tipos de instancia **EC2 t3.micro** para la capa de aplicación y **RDS db.t3.micro** para la base de datos MySQL.

La elección de estos tamaños responde a un criterio de **mínimos recursos necesarios** para ejecutar correctamente la práctica en un entorno de pruebas como es AWS Learner Lab, que tiene un presupuesto de uso limitado.

Esta selección también permite **optimizar costes**, evitando sobredimensionamiento de los recursos en AWS mientras se mantiene un entorno funcional para experimentar con escalado, snapshots, backups y despliegues automáticos.

Aunque estas instancias no estarían recomendadas para entornos de producción con alta carga, son **suficientes para fines didácticos**, demostrando el manejo de Auto Scaling, Load Balancer y conectividad entre capas de manera eficiente.

Principios de arquitectura aplicados (seguridad, resiliencia, escalabilidad)

En el diseño de **Penguin On The Way!** se han aplicado los principios fundamentales de arquitectura en la nube para garantizar un entorno seguro, resiliente y escalable.

- **Seguridad:** Cada capa cuenta con **Security Groups específicos**, limitando el tráfico entrante y saliente según necesidad (ALB, EC2, RDS, VPC Endpoint), así como ACL´s a nivel de subredes y tablas de ruta hacía NAT Gateway desde la capa de aplicación, asegurando que las instancias EC2 solo salen a internet a través de estas. Además, se ha configurado el **VPC Endpoint para SNS**, evitando tráfico de notificaciones por Internet, y las credenciales de base de datos y variables sensibles se gestionan mediante archivos de entorno protegidos y roles de IAM.
- **Resiliencia:** La base de datos RDS está desplegada en **Multi-AZ**, asegurando alta disponibilidad y tolerancia a fallos. Los **snapshots automáticos y backups programados** permiten recuperación ante errores o borrado accidental de datos.

- **Escalabilidad:** La capa de aplicación cuenta con un **Auto Scaling Group** vinculado a un **Application Load Balancer**, que ajusta dinámicamente el número de instancias EC2 según la carga de solicitudes. Esta configuración permite responder eficientemente a picos de tráfico sin intervención manual, manteniendo la performance y disponibilidad.

En conjunto, estas decisiones reflejan buenas prácticas de arquitectura AWS, adaptadas a un entorno de **pruebas y aprendizaje**, demostrando competencias de diseño de arquitecturas seguras, resilientes y escalables.

Componentes de infraestructura

Arquitectura de red: VPC, subnets, routing, IGW, NAT Gateways, NACLs.

Capa de aplicación: EC2, ALB, Auto Scaling.

Capa de datos: RDS Multi-AZ.

Notificaciones y automatización de eventos: integración mediante SNS.

Diagrama conceptual de la infraestructura.

Componentes de la infraestructura

A continuación, capa por capa, se muestra el inventariado y creación de todos los componentes relativos a:

- **Capa de red:** VPC, subnets, routing, IGW, NAT Gateways, NACLs, Route tables.
- **Capa de aplicación:** EC2, ALB, Auto Scaling
- **Capa de datos:** RDS Multi-AZ, failover, backups
- **Notificaciones y automatización de eventos:** integración mediante SNS
- **Seguridad y monitorización:** Security Groups, IAM Roles, CloudWatch y alarmas para garantizar la protección y visibilidad de la infraestructura.

Como se indica en la sección de "objetivos", este proyecto busca **aplicar IaC** mediante CloudFormation, por lo que la mayoría del desarrollo se hará haciendo uso de código.

Arquitectura de red: VPC, subnets, routing, IGW, NAT Gateways, NACLs, route tables

Comenzando con VPC y subnets, este proyecto hará uso de:

VPC: 10.0.0.0/16 – bloque de direcciones amplio que permite futuras expansiones del proyecto.

- Asociado a un **Internet Gateway (IGW)** que permite la conectividad de la capa pública con internet.
- DNS de AWS habilitado para resolución interna de nombres dentro de la VPC.

Subnets públicas:

- **Public subnet 1 (capa pública 1)**: 10.0.0.0/24
- **Public subnet 2 (capa pública 2):** 10.0.1.0/24
- Contienen los **NAT Gateways** que permiten la salida a internet de recursos privados de manera controlada.
- Cada subnet está asociada a su **Route Table pública**, que enruta 0.0.0.0/0 hacia el IGW.

Subnets privadas de aplicación:

- **Private subnet 1 (capa de aplicación 1):** 10.0.10.0/24
- **Private subnet 2 (capa de aplicación 2):** 10.0.11.0/24
- **Enrutamiento:** salida a internet a través del NAT Gateway de la misma AZ, garantizando alta disponibilidad

Subnets privadas de datos:

- **Private subnet 3 (capa de datos 1):** 10.0.20.0/24
- **Private subnet 4 (capa de datos 2):** 10.0.21.0/24
- Aíslan la base de datos RDS; no tienen acceso directo a internet, solo aceptan tráfico desde la capa de aplicación mediante **Security Groups**.

Justificación de la separación de capas

- Las **subnets públicas alojan NAT Gateways y ALB** (si se decide exponerlo), permitiendo conectividad controlada desde internet.
- Las **subnets privadas de aplicación aíslan las instancias EC2** de acceso directo desde internet, reforzando la seguridad.
- Las **subnets privadas de datos aíslan la base de datos RDS**, protegiendo la información y asegurando que solo la capa de aplicación pueda interactuar con ella.

Routing y acceso a internet

- **Subnets públicas:** ruta por defecto a IGW.
- **Subnets privadas de aplicación:** salida a internet vía NAT Gateway.
- **Subnets privadas de datos:** solo aceptan tráfico desde la capa de aplicación; no tienen ruta directa a internet.

NACLs y Security Groups

- **NACLs:** lista de control de acceso a nivel de subnet; tres NACL distintas se aplican según la capa de infraestructura.

- **Security Groups:** firewall virtual a nivel de recurso (ALB, EC2, RDS, VPC Endpoints). Ejemplos:
    - ALB: permite HTTP desde internet.
    - EC2: solo acepta tráfico HTTP desde ALB.
    - RDS: solo permite tráfico MySQL desde EC2.
    - VPC Endpoint SNS: solo permite tráfico HTTP desde las EC2.

VPC Endpoints

- Se utiliza un **VPC Endpoint de tipo Interface para SNS**.
- Subnets: desplegado en las subnets privadas de aplicación para asegurar conectividad interna.
- Security Group: permite tráfico únicamente desde las EC2 de la aplicación.
- Objetivo: permitir que la aplicación envíe notificaciones a SNS sin necesidad de salida a internet, aumentando seguridad y reduciendo dependencias externas.

Alta disponibilidad y resiliencia

- Duplicación de recursos en **2 AZs**, asegurando que EC2 y RDS continúen operando incluso si una AZ falla.
- La distribución soporta **Auto Scaling** de la capa de aplicación y **failover automático** de RDS Multi-AZ.

VPC Flow Logs

- Registro de todo el tráfico que entra y sale de la VPC.
- Captura: IP de origen/destino, puerto, protocolo, cantidad de bytes/paquetes y acción (aceptada/denegada).
- Destino: CloudWatch Logs para auditoría, monitorización y resolución de problemas.
- Beneficio: facilita verificar la comunicación entre capas y detectar intentos de conexión no autorizados.

Tags y buenas prácticas

- Todos los recursos de red (VPC, subnets, NAT Gateways, IGW, NACLs) incluyen **tags descriptivos** para facilitar auditoría, gestión y mantenimiento futuro.

Capa de aplicación: EC2, ALB, Auto Scaling, SNS

La capa de aplicación se encarga de alojar la aplicación **"Penguin On The Way!"**, la cual permite la gestión de tickets de forma mínima como caso de uso representativo. Esta capa ha sido diseñada para ser **altamente disponible, escalable y segura**, siguiendo buenas prácticas de arquitectura cloud.

Despliegue de instancias EC2

- **Instancias EC2:** Se utilizan instancias EC2 en la capa de aplicación, desplegadas dentro de las subnets privadas de aplicación.
- **Sistema operativo:** Ubuntu 22.04 LTS.
- **Instalación de la app:**
  - Se ejecuta un script de setup que instala Python 3, pip, dependencias de la aplicación (Flask, PyMySQL, Boto3, Gunicorn, Python-dotenv).
  - Se crea un **entorno virtual** para aislar las dependencias de la aplicación.
  - Se configura **Gunicorn como servicio systemd**, asegurando que la app se inicie automáticamente al arrancar la máquina (cabe mencionar que las instancias se inician desde una AMI en Auto Scaling).
- **Seguridad:**
  - Las EC2 no exponen puertos directamente a internet; solo aceptan tráfico HTTP desde el ALB.
  - Se aplican Security Groups específicos para permitir únicamente la comunicación necesaria (ALB → EC2).

Application Load Balancer (ALB)

- **Objetivo:** distribuir tráfico de manera eficiente entre las instancias EC2 y soportar alta disponibilidad.
- **Configuración:**
  - Se despliega en las subnets públicas para exponer un punto de entrada seguro a la aplicación.
  - Redirige tráfico HTTP hacia las EC2 que se encuentran en subnets privadas.

- Configura cabeceras para mantener la trazabilidad y la IP de origen (X-Real-IP, X-Forwarded-For).

Auto Scaling Group (ASG)

- **Objetivo:** garantizar escalabilidad horizontal automática ante cambios de carga.
- **Configuración:**
  - ASG asociado a un **Launch Template** que define el tipo de instancia EC2, AMI, Security Group y script de bootstrap.
  - Se establece un mínimo de instancias (2) para asegurar disponibilidad constante y un máximo que permita escalar según la demanda (4).
  - Health checks del ALB aseguran que las instancias no saludables se eliminen automáticamente y se lancen nuevas instancias.
- **Alta disponibilidad:**
  - Distribución de instancias en 2 AZ, de modo que si una zona falla, la otra sigue operativa.

Comunicación interna y dependencias

- Las EC2 se comunican únicamente con:
  - **RDS Multi-AZ:** para operaciones de escritura de tickets.
  - **VPC Endpoint de SNS:** para publicar notificaciones de creación de tickets hacia los administradores.
- Todo el tráfico está controlado mediante Security Groups y NACLs, manteniendo la aplicación aislada del acceso directo a internet.
- La salida a internet para actualizaciones o dependencias se realiza a través de **NAT Gateways** ubicados en subnets públicas.

Monitorización y logs

- **CloudWatch y S3 Logs:** Los distintos servicios centralizan los logs en Cloud Watch y S3.
- **Métricas de instancias EC2:** Monitorizadas mediante CloudWatch Agent.
- **Alarmas:** Se configuran alarmas de CPU alta, instancias no saludables o fallos en la app que envían notificaciones vía SNS a los administradores.

Resumen de buenas prácticas aplicadas

- Separación clara entre capas públicas, aplicación y datos.
- Auto Scaling y ALB garantizan disponibilidad y distribución de carga.

- Seguridad a nivel de Security Groups y subnets privadas.
- Observabilidad con métricas y logs centralizados.
- Despliegue reproducible mediante scripts de bootstrap y CloudFormation.

Capa de datos: RDS Multi-AZ

La capa de datos es la encargada de almacenar de forma persistente la información generada por la aplicación, en este caso los tickets creados por los usuarios. Se ha diseñado siguiendo principios de **seguridad, aislamiento y fiabilidad**, utilizando Amazon RDS como servicio gestionado.

Despliegue de RDS MySQL

- **Motor de base de datos:** MySQL 8.0 gestionado por Amazon RDS.
- **Instancia:** tipo db.t3.micro, adecuada para entornos de laboratorio y pruebas.
- **Almacenamiento:** 20 GB con tipo gp2 (General Purpose SSD). Sería mejor práctica usar las de tipo Memory Optimized, pero al estar en un entorno de práctica se hace uso de las General Purpose.
- **Configuración:**
    - Base de datos inicial: penguindb.
    - Usuario administrador definido mediante parámetros de CloudFormation.
    - Credenciales gestionadas como parámetros (NoEcho) para evitar exposición en la consola.

Subnet Group y despliegue en red privada

- Se define un **DB Subnet Group** que agrupa las subnets privadas de la capa de datos.
- La base de datos se despliega exclusivamente en estas subnets, garantizando:
    - Aislamiento completo de internet (PubliclyAccessible: false).
    - Separación clara respecto a la capa de aplicación.
- Esto asegura que RDS solo sea accesible desde recursos internos de la VPC.

Seguridad y control de acceso

- **Security Group de RDS:**
    - Permite únicamente tráfico entrante en puerto MySQL (3306) desde las instancias EC2 de la capa de aplicación.
    - Bloquea cualquier acceso externo o desde otras capas no autorizadas.
- **Buenas prácticas aplicadas:**
    - Principio de mínimo privilegio.

  - Aislamiento de la base de datos en subnets privadas.
  - No exposición pública de endpoints.

Backups y persistencia

- **BackupRetentionPeriod:** 7 días.
- RDS realiza automáticamente:
  - Backups diarios.
  - Snapshots automáticos.
- Esto permite recuperar la base de datos ante fallos o pérdida de datos.

Alta disponibilidad y resiliencia

En esta implementación, se ha habilitado la configuración **Multi-AZ (MultiAZ: true)**, lo que permite desplegar una instancia primaria junto a una réplica en una segunda Availability Zone.

Esta configuración proporciona:

- **Replicación síncrona** de los datos entre ambas instancias, garantizando consistencia.
- **Alta disponibilidad**, mediante un mecanismo de **failover automático** en caso de caída de la instancia principal.
- Transparencia para la aplicación, ya que el endpoint de conexión permanece inalterado durante el proceso de conmutación.

De esta forma, se asegura la continuidad del servicio y la resiliencia de la capa de datos ante fallos a nivel de infraestructura.

Integración con la capa de aplicación

- Las instancias EC2 se conectan a RDS mediante el **endpoint exportado por CloudFormation**.
- La aplicación utiliza **PyMySQL** para realizar operaciones de escritura en la tabla de tickets.
- Flujo de datos:
  1. El usuario envía un formulario desde la web.
  2. La aplicación procesa la solicitud en EC2.
  3. Se inserta un registro en la base de datos RDS.
  4. Se genera una notificación mediante SNS, el cual envía un mail.

Monitorización y mantenimiento

- **CloudWatch Metrics:** permite monitorizar CPU, conexiones, almacenamiento y latencia.
- **Logs:** se pueden habilitar logs de MySQL para auditoría y troubleshooting.
- **Escalabilidad:** posibilidad de aumentar el tamaño de la instancia o almacenamiento sin rediseñar la arquitectura.

Buenas prácticas aplicadas

- Uso de servicio gestionado (RDS) para reducir carga operativa.
- Aislamiento en subnets privadas.
- Control de acceso mediante Security Groups.
- Backups automáticos y retención de datos.
- Integración desacoplada con la capa de aplicación.

Notificaciones y automatización de eventos: integración mediante SNS

En esta capa se implementa un sistema de notificaciones basado en el servicio gestionado Amazon SNS (Simple Notification Service), cuyo objetivo es **desacoplar la lógica de la aplicación de la gestión de notificaciones**.

La aplicación, desplegada en instancias EC2, publica eventos en un topic de SNS cada vez que se registra un nuevo ticket. Estos eventos contienen la información del formulario enviado por el usuario y son procesados de forma asíncrona por el propio servicio.

Para la entrega de notificaciones, se configura una suscripción de tipo email asociada al topic, permitiendo que los administradores reciban automáticamente un correo electrónico con la información del ticket generado.

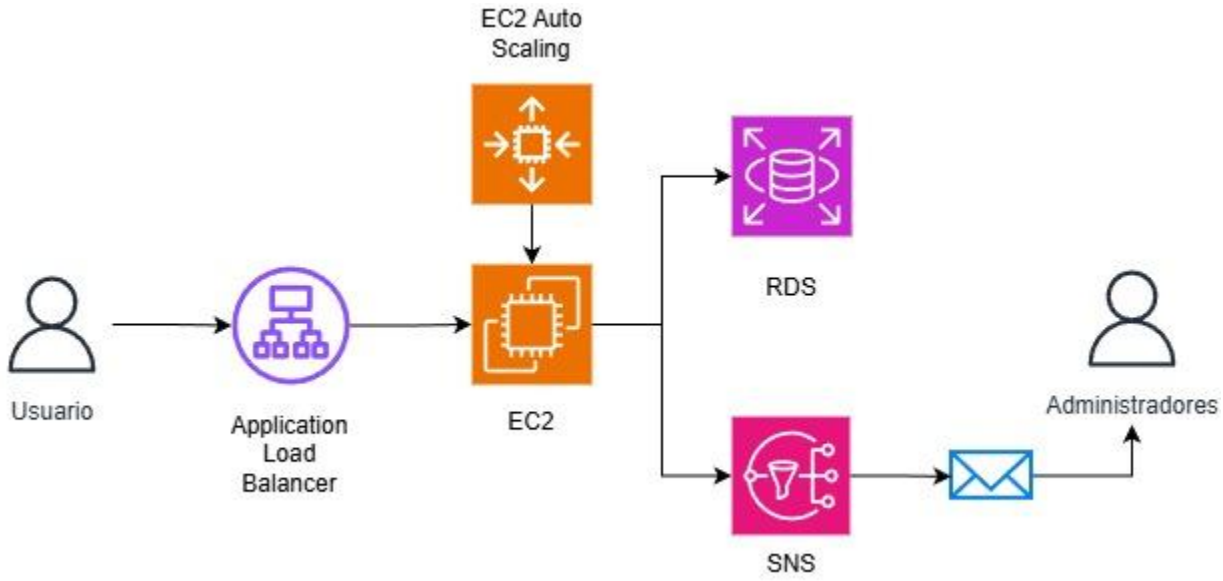
Funcionamiento del flujo de eventos

- El usuario envía un formulario desde la aplicación web.
- La aplicación almacena los datos en la base de datos RDS.
- Se publica un mensaje en el topic de SNS con la información del ticket.
- SNS distribuye el mensaje a los suscriptores configurados.
- El administrador recibe una notificación por correo electrónico.

Características clave de la solución

- **Desacoplamiento** entre la aplicación y el sistema de notificaciones.
- **Procesamiento asíncrono**, evitando bloquear la lógica principal de la aplicación.
- **Escalabilidad automática**, gestionada por SNS sin necesidad de aprovisionamiento.
- **Alta disponibilidad**, proporcionada por el servicio gestionado.
- **Simplicidad operativa**, al no requerir gestión de servidores ni lógica adicional (como funciones intermedias).

Diagrama conceptual de la infraestructura



Infraestructura como Código y despliegue

CloudFormation: stacks y parametrización

Flujo de despliegue y demostración

Aplicación de Penguin on the way! NGINX + Flask + Gunicorn

Infraestructura como Código y despliegue

En esta sección se detalla **cómo se ha definido y desplegado toda la infraestructura del proyecto** mediante Infraestructura como Código (IaC) utilizando AWS CloudFormation.

Esta metodología permite **crear, configurar y versionar los recursos de manera reproducible y automatizada**, reduciendo errores manuales y facilitando la escalabilidad y mantenimiento del entorno. Además, se explicará el funcionamiento de **Penguin On The Way**, abordando cómo se integra **Nginx, Flask y Gunicorn**, el despliegue del **frontend web**, el **backend** y la comunicación con la base de datos y servicios de AWS, proporcionando una visión completa del flujo de la aplicación y su infraestructura asociada.

CloudFormation: stacks y parametrización

En esta sección se explica cómo se ha utilizado **AWS CloudFormation** para definir, desplegar y gestionar los recursos de la infraestructura de manera automatizada.

Se detallará la creación de **stacks**, la organización de plantillas y la **parametrización de recursos**, lo que permite adaptar el despliegue a distintos entornos (por ejemplo, desarrollo, pruebas o producción) sin modificar la plantilla base.

Asimismo, se mostrará cómo los parámetros, outputs y dependencias entre stacks facilitan la integración entre distintos componentes de la aplicación, garantizando coherencia, trazabilidad y facilidad de mantenimiento.

Despliegue de la capa de red

Consta de **los siguientes archivos .YAML**: https://github.com/CarlosFdezSalazar/Penguin-On-the-Way- (network-stack.yaml, security-groups.yaml)

La capa de red constituye la base de la infraestructura de **Penguin On The Way!**, proporcionando aislamiento, segmentación y conectividad segura para todos los recursos desplegados.

A continuación se describen los recursos creados y su propósito:

VPC

- Se ha creado una **VPC** con CIDR parametrizable (VpcCIDR), que actúa como contenedor de todos los recursos de red del proyecto.
- Se habilitan **DNS support y DNS hostnames** para permitir la resolución de nombres dentro de la VPC.
- La VPC recibe un **tag con el nombre del entorno**, lo que facilita su identificación en la consola.

Internet Gateway

- Se crea un **Internet Gateway (IGW)** y se adjunta a la VPC, permitiendo conectividad hacia y desde internet para los recursos de las subredes públicas.
- Se asigna un **tag con el nombre del entorno**.

Subnets

- **Subnets públicas**: se crean dos subredes públicas, una por cada zona de disponibilidad (AZ1 y AZ2), donde se desplegarán los recursos accesibles desde internet (por ejemplo, el ALB) y las NAT Gateways. Se habilita **MapPublicIpOnLaunch** para asignar IP públicas automáticamente a las instancias lanzadas.
- **Subnets privadas de aplicación**: dos subredes privadas para instancias EC2 de aplicación, sin acceso directo a internet, sino a través de las NAT Gateway.
- **Subnets privadas de datos**: dos subredes privadas destinadas a la base de datos RDS, aisladas del tráfico público.

NAT Gateways

- Se crean **dos NAT Gateways**, uno por cada zona de disponibilidad, junto con sus Elastic IPs.
- Permiten a las subnets privadas de aplicación acceder a internet para actualizaciones o descargas sin exponerlas públicamente.

Tablas de rutas

- **Tablas de rutas públicas**: se crea una tabla de rutas que dirige el tráfico 0.0.0.0/0 hacia el Internet Gateway. Las subnets públicas se asocian a esta tabla.

- **Tablas de rutas privadas de aplicación**: cada subnet privada de aplicación tiene su propia tabla de rutas, que dirige el tráfico saliente hacia su NAT Gateway correspondiente.
- **Tablas de rutas privadas de datos**: las subnets de datos no tienen ruta a internet; únicamente permiten comunicación interna dentro de la VPC, garantizando aislamiento de los datos sensibles.

Network ACLs

Se han definido **Network ACLs (NACLs)** para añadir una capa extra de seguridad a nivel de subred:

- **Capa pública**: permite tráfico HTTP/HTTPS entrante, tráfico efímero para la comunicación de retorno y todo el tráfico saliente.
- **Capa de aplicación**: permite tráfico HTTP desde la VPC, tráfico efímero necesario para el handshake entre EC2 y ALB, y tráfico de retorno hacia la VPC o internet.
- **Capa de datos**: permite tráfico MySQL entre EC2 y RDS, tráfico efímero para handshakes internos y todo el tráfico saliente (aunque RDS no tiene ruta hacia internet).

Cada NACL se asocia a las subnets correspondientes mediante **SubnetNetworkAclAssociation**.

VPC Flow Logs

- Se habilitan **VPC Flow Logs** para registrar todo el tráfico dentro de la VPC (TrafficType: ALL) y enviarlo a **CloudWatch Logs**.
- Esto permite auditoría, monitorización de tráfico y análisis de patrones de comunicación entre los recursos.

Outputs

- Se exportan el **ID de la VPC** y su **CIDR** para que puedan ser reutilizados por otros stacks o plantillas de CloudFormation.

Security Groups

Para garantizar la seguridad y segmentación del tráfico dentro de la infraestructura de **Penguin On The Way!**, se han definido **Security Groups (SG)** específicos para cada capa de la arquitectura: **Application Load Balancer (ALB)**, **EC2** y **RDS**.

Estos SG actúan como **firewalls virtuales a nivel de instancia**, controlando el tráfico entrante y saliente y complementando las Network ACLs definidas previamente.

Security Group del ALB

- **Nombre:** ${EnvironmentName}-ALB-SG
- **Descripción:** Permite tráfico HTTP público hacia el ALB.
- **Ingress:**
  - Se permite TCP puerto 80 desde cualquier dirección (0.0.0.0/0) para que la aplicación sea accesible desde internet.
- **Egress:**
  - Se permite todo el tráfico saliente (0.0.0.0/0) para que el ALB pueda comunicarse con las instancias EC2 de la capa de aplicación.
- **Justificación:** El ALB necesita recibir tráfico externo y reenviarlo a las instancias EC2. Limitar la entrada únicamente al puerto 80 asegura que solo se acepte tráfico web.

Security Group de las EC2 (Auto Scaling)

- **Nombre:** ${EnvironmentName}-EC2-SG
- **Descripción:** Permite tráfico entrante desde el ALB y salida hacia RDS, SNS y servicios de internet.
- **Ingress:**
  - Se permite TCP puerto 80 **solo desde el Security Group del ALB**, garantizando que las instancias EC2 solo reciban tráfico que provenga del balanceador de carga.
- **Egress:**
  - Se permite todo el tráfico saliente (0.0.0.0/0) para que las EC2 puedan comunicarse con RDS, SNS, internet y otros servicios AWS que necesiten acceso saliente.
- **Justificación:** Esta capa asegura que las instancias de aplicación no estén expuestas directamente a internet, recibiendo tráfico únicamente a través del ALB.

Security Group de RDS

- **Nombre:** ${EnvironmentName}-RDS-SG
- **Descripción:** Permite tráfico solo desde las EC2 que ejecutan la aplicación.
- **Ingress:**

    - Se permite TCP puerto 3306 solo desde el Security Group de las EC2, restringiendo el acceso a la base de datos únicamente a las instancias de aplicación autorizadas.
- **Egress:**
    - Se permite todo el tráfico saliente (0.0.0.0/0) para cubrir necesidades de replicación, backups o comunicaciones internas de RDS con otros servicios AWS si fueran necesarias.
- **Justificación:** Garantiza que la base de datos no sea accesible desde internet y solo pueda ser consumida por la capa de aplicación

Outputs

- Se exportan los **IDs de los Security Groups** para poder reutilizarlos en otros stacks de CloudFormation (por ejemplo, al crear EC2, ALB o RDS) y mantener consistencia entre entornos.
    - ${EnvironmentName}-ALBSecurityGroupId
    - ${EnvironmentName}-EC2SecurityGroupId
    - ${EnvironmentName}-RDSSecurityGroupId

VPC Endpoint

Consta del **siguiente archivo .YAML**:
https://github.com/CarlosFdezSalazar/Penguin-On-the-Way- (vpc-endpoint.yaml)

Para garantizar que las instancias EC2 en subnets privadas puedan comunicarse de forma **segura y privada** con **Amazon SNS**, se ha creado un **VPC Endpoint de tipo Interface**. Esto evita que el tráfico salga a internet, cumpliendo con los principios de **privacidad y aislamiento de la red**.

Security Group del VPC Endpoint

- **Nombre:** ${EnvironmentName}-SNS-VPCE-SG
- **Descripción:** Controla el tráfico hacia el VPC Endpoint de SNS.
- **Ingress:**
    - Se permite todo tipo de tráfico (IpProtocol: -1) únicamente desde el **Security Group de EC2**, garantizando que solo las instancias de aplicación autorizadas puedan acceder al endpoint.
- **Egress:**
    - Se permite todo el tráfico saliente (0.0.0.0/0) para que el VPC Endpoint pueda comunicarse con el servicio SNS y otros recursos necesarios dentro de AWS.

- **Justificación:** Aísla el acceso al endpoint únicamente a la capa de aplicación, evitando exposición innecesaria a otras subnets o a internet.

VPC Endpoint para SNS

- **Tipo:** Interface
- **Subnets asociadas:**
  - Subnet privada de aplicación AZ1
  - Subnet privada de aplicación AZ2
    Esto asegura alta disponibilidad, distribuyendo la conectividad del endpoint entre dos zonas de disponibilidad.
- **Seguridad:**
  - Se asocia al Security Group previamente definido (VpcEndpointSG).
- **DNS privado habilitado:**
  - PrivateDnsEnabled: true permite que las instancias puedan resolver sns.<region>.amazonaws.com a la IP privada dentro de la VPC, evitando tráfico público.
- **Justificación:** El VPC Endpoint permite que la comunicación con SNS sea **privada, segura y redundante**, cumpliendo con buenas prácticas de arquitectura sin depender de NAT Gateways ni de tráfico público.

Outputs

- **SnsVpcEndpointId:** ID del endpoint, exportado para poder usarlo en otros stacks que necesiten conectividad con SNS.
- **VpcEndpointSGId:** ID del Security Group del endpoint, exportado para que otros recursos puedan referenciarlo si es necesario.

Despliegue de la capa de datos

Consta **del siguiente archivo .YAML:**
https://github.com/CarlosFdezSalazar/Penguin-On-the-Way- (data-stack.yaml)

La capa de datos de Penguin On The Way se implementa mediante **Amazon RDS MySQL**, proporcionando una base de datos **gestionada, escalable y altamente disponible**. Esta sección describe cómo se han definido los recursos de CloudFormation para desplegar RDS de forma segura dentro de la VPC.

Parámetros

- **EnvironmentName:** Prefijo para identificar los recursos en distintos entornos (desarrollo, pruebas, producción).
- **VpcId:** VPC donde se desplegará la base de datos, garantizando que RDS se ubique en la red privada.
- **PrivateDataSubnets:** Subnets privadas donde se desplegará RDS, distribuidas en múltiples zonas de disponibilidad para **alta disponibilidad (Multi-AZ)**.
- **RDSSecurityGroup:** Security Group que controla el tráfico hacia RDS, importado desde el stack de Security Groups, asegurando que solo las EC2 autorizadas puedan conectarse.
- **DBName, DBUsername, DBPassword:** Configuración de la base de datos. El password se marca como NoEcho para que no se exponga en la consola ni en logs.
- **DBInstanceClass, AllocatedStorage:** Especifican el tamaño y la capacidad de la instancia RDS.

Recursos

RDS Subnet Group

- **Tipo:** AWS::RDS::DBSubnetGroup
- **Propósito:** Agrupa las subnets privadas donde RDS puede desplegar instancias, permitiendo que la base de datos tenga **alta disponibilidad** en caso de fallo de una AZ.
- **Justificación:** RDS no puede estar expuesto públicamente, y su comunicación se mantiene dentro de la VPC.

Instancia RDS MySQL

- **Tipo:** AWS::RDS::DBInstance
- **Propiedades clave:**
    - Engine: mysql y EngineVersion: 8.0
    - VPCSecurityGroups: asociado al SG que permite tráfico solo desde EC2
    - DBSubnetGroupName: subnets privadas
    - PubliclyAccessible: false: asegura que la base de datos no tenga acceso público
    - MultiAZ: true: activa **replicación en una segunda AZ** para resiliencia
    - StorageType: gp2, AllocatedStorage: almacenamiento gestionado
    - BackupRetentionPeriod: 7: permite restauración ante fallos o borrado accidental

- **Justificación:** Se configura para cumplir con buenas prácticas de seguridad, disponibilidad y continuidad de negocio. La base de datos permanece aislada de internet y solo accesible desde la capa de aplicación.

Outputs

- **RDSEndpoint:** Dirección para conectar aplicaciones a la base de datos.
- **RDSSecurityGroupId:** ID del Security Group asociado, exportado para ser referenciado por otros stacks o recursos que necesiten conectividad con RDS.

Despliegue de SNS

Consta **del siguiente archivo .YAML**:
https://github.com/CarlosFdezSalazar/Penguin-On-the-Way- (sns.yaml)

Para gestionar el envío de notificaciones dentro de la arquitectura de **Penguin On The Way!**, se ha implementado **Amazon Simple Notification Service (SNS)**. Este servicio permite desacoplar componentes del sistema y enviar alertas o mensajes de forma asíncrona a distintos destinos, como correo electrónico.

Recursos

SNS Topic

- **Tipo:** AWS::SNS::Topic
- **Nombre:** POTW-Notifications
- **Propósito:** Actúa como canal centralizado de mensajería donde los distintos componentes de la aplicación pueden publicar eventos.
- **Justificación:** Permite desacoplar la lógica de la aplicación del sistema de notificaciones, facilitando la escalabilidad y la integración con otros servicios en el futuro.

Suscripción por Email

- **Tipo:** AWS::SNS::Subscription
- **Protocolo:** email
- **Endpoint:** penguinontheway.test@gmail.com
- **Propósito:** Permite recibir notificaciones directamente en un correo electrónico cada vez que se publica un mensaje en el topic.
- **Funcionamiento:**
    - Tras la creación, SNS envía un correo de confirmación al destinatario.
    - Es necesario aceptar la suscripción para empezar a recibir notificaciones.

- **Justificación:** Proporciona un mecanismo sencillo y directo para recibir alertas, útil tanto para monitorización como para pruebas funcionales de la aplicación.

Outputs

- **SNSTopicArn:** ARN del topic SNS, exportado para que pueda ser utilizado por otros stacks (por ejemplo, las instancias EC2 que envían notificaciones desde la aplicación backend).

Integración en la arquitectura

- La aplicación backend (Flask) publica mensajes en este topic cada vez que se registra un nuevo ticket.
- Gracias al uso de un **VPC Endpoint para SNS**, estas comunicaciones se realizan de forma privada dentro de la VPC, sin necesidad de salir a internet.
- Este enfoque permite ampliar fácilmente el sistema de notificaciones en el futuro (por ejemplo, añadiendo SMS, Lambda o integraciones con otros servicios).

Despliegue de la capa de aplicación

Consta **del siguiente archivo .YAML:**
https://github.com/CarlosFdezSalazar/Penguin-On-the-Way- (app-stack.yaml)

La capa de aplicación de Penguin On The Way constituye el núcleo del sistema, encargándose de procesar las peticiones de los usuarios, ejecutar la lógica de negocio y garantizar la escalabilidad y alta disponibilidad de la aplicación.

Esta capa está compuesta por un **Application Load Balancer (ALB)**, un **Auto Scaling Group (ASG)** de instancias EC2 y los recursos necesarios para gestionar el tráfico de forma eficiente.

Parámetros

- **EnvironmentName:** Prefijo para identificar recursos.
- **VpcId:** VPC donde se despliega la aplicación.
- **PublicSubnets:** Subnets públicas donde se despliega el ALB.
- **PrivateAppSubnets:** Subnets privadas donde se ejecutan las EC2.
- **ALBSecurityGroup / EC2SecurityGroup:** Controlan el tráfico entre capas.
- **RDSEndpoint / RDSDatabase / DBUsername / DBPassword:** Configuración de conexión a la base de datos.
- **AMIId:** AMI personalizada con la aplicación preinstalada.

Recursos

Application Load Balancer (ALB)

- **Tipo:** AWS::ElasticLoadBalancingV2::LoadBalancer
- **Esquema:** internet-facing
- **Ubicación:** Subnets públicas
- **Función:**
  - Recibe tráfico HTTP desde internet.
  - Distribuye las peticiones entre las instancias EC2.
- **Justificación:** Permite desacoplar el punto de entrada de la aplicación de las instancias backend, mejorando disponibilidad y escalabilidad.

Target Group

- **Tipo:** AWS::ElasticLoadBalancingV2::TargetGroup
- **Puerto:** 80 (HTTP)
- **Health Check:**
  - Path: /
  - Intervalo: 30 segundos
- **Función:**
  - Define el conjunto de instancias EC2 que reciben tráfico.
  - Supervisa su estado mediante health checks.
- **Justificación:** Garantiza que solo instancias sanas reciban tráfico.

Listener del ALB

- **Puerto:** 80
- **Protocolo:** HTTP
- **Acción:**
  - Redirige tráfico al Target Group
- **Justificación:** Punto de entrada del tráfico web hacia la aplicación.

Launch Template

- **Tipo:** AWS::EC2::LaunchTemplate
- **Configuración:**
  - Tipo de instancia: t3.micro
  - Security Group: EC2
  - IAM Role: LabInstanceProfile
  - AMI personalizada
- **UserData:** Script básico de arranque

- **Justificación:** Define cómo deben lanzarse las instancias EC2 de forma consistente.

Auto Scaling Group (ASG)

- **Tipo:** AWS::AutoScaling::AutoScalingGroup
- **Subnets:** Privadas (capa de aplicación)
- **Capacidad:**
  - Mínimo: 2
  - Deseado: 2
  - Máximo: 4
- **Integración:** Asociado al Target Group
- **Función:**
  - Mantiene el número de instancias necesarias.
  - Reemplaza instancias fallidas automáticamente.
- **Justificación:** Garantiza alta disponibilidad y tolerancia a fallos.

Política de Auto Scaling basada en tráfico

- **Tipo:** TargetTrackingScaling
- **Métrica:** ALBRequestCountPerTarget
- **Objetivo:** 50 requests por instancia
- **Cooldown:** 300 segundos
- **Función:**
  - Escala automáticamente el número de instancias en función de la carga real.
- **Justificación:** Permite adaptar dinámicamente la infraestructura al tráfico de la aplicación.

Flujo de funcionamiento

1. El usuario accede a la aplicación a través del **ALB**.
2. El ALB distribuye la petición a una instancia EC2 disponible.
3. La EC2 ejecuta la aplicación (Flask + Gunicorn + Nginx).
4. La aplicación interactúa con la base de datos RDS.
5. Si la carga aumenta, el ASG lanza nuevas instancias automáticamente.

Outputs

- **ALBArn:** Identificador del Load Balancer
- **AutoScalingGroupName:** Nombre del grupo de autoescalado

Flujo de despliegue y demostración

El despliegue de la infraestructura de **Penguin On The Way!** se ha diseñado siguiendo un enfoque estructurado por capas, respetando las dependencias entre los distintos componentes.

Este orden garantiza que cada recurso se cree únicamente cuando los elementos de los que depende ya están disponibles, evitando errores y asegurando la coherencia del entorno.

El flujo de despliegue se realiza en el siguiente orden:

1. **Capa de red**: creación de la VPC, subnets, tablas de rutas, NAT Gateways y configuración de conectividad.
2. **Security Groups**: definición de las reglas de acceso entre los distintos componentes de la arquitectura.
3. **VPC Endpoint**: habilitación de la comunicación privada con servicios de AWS (SNS) desde las subnets privadas.
4. **Capa de datos**: despliegue de la base de datos RDS en subnets privadas con alta disponibilidad (Multi-AZ).
5. **SNS**: creación del sistema de notificaciones y suscripciones por correo electrónico.
6. **Capa de aplicación**: despliegue del Application Load Balancer, Auto Scaling Group y las instancias EC2 que ejecutan la aplicación.

Este enfoque modular no solo facilita el despliegue inicial, sino que también permite reutilizar y mantener cada componente de forma independiente, siguiendo buenas prácticas de **Infraestructura como Código (IaC)**.

A continuación, se facilita un video comentado donde se hace el despliegue en Cloud Formation paso a paso: https://youtu.be/x_02zRjsbXw

Aplicación de Penguin on the way! NGINX + Flask + Gunicorn

La aplicación desplegada en las instancias EC2 sigue una arquitectura clásica de aplicaciones web en producción basada en la combinación de **Nginx, Gunicorn y Flask**, separando responsabilidades y optimizando el rendimiento y la escalabilidad.

Esta configuración se incluye dentro de una **AMI personalizada**, que posteriormente es utilizada por el Launch Template del Auto Scaling Group, garantizando que todas las instancias se desplieguen de forma homogénea.

Componentes de la aplicación

Flask (Backend)

Flask actúa como el núcleo de la lógica de negocio de la aplicación.

- Implementa una API sencilla con dos endpoints:
  - / → renderiza el formulario web
  - /submit → recibe los datos del formulario vía JSON
- Funcionalidades principales:
  - Validación de datos de entrada
  - Inserción de tickets en la base de datos RDS (MySQL) → Video demostrativo: https://youtu.be/pYG5wb1C6_I
  - Envío de notificaciones a través de SNS

Además, en el arranque:

- Se establece la conexión con RDS
- Se crea automáticamente la tabla formulario si no existe

Esto permite que la aplicación sea **autosuficiente** al desplegarse en nuevas instancias.

Gunicorn (Application Server)

Gunicorn actúa como servidor WSGI encargado de ejecutar la aplicación Flask en producción.

- Configuración:
  - 4 workers (-w 4)
  - Bind en 127.0.0.1:8000
- Se ejecuta como servicio systemd (penguin.service), lo que permite:
  - Arranque automático
  - Reinicio en caso de fallo
- Justificación:
  - Mejora el rendimiento frente al servidor de desarrollo de Flask
  - Permite manejar múltiples peticiones concurrentes

Nginx (Reverse Proxy)

Nginx se encarga de gestionar las conexiones HTTP entrantes y actuar como proxy inverso hacia Gunicorn.

- Escucha en el puerto 80
- Redirige tráfico a 127.0.0.1:8000
- Añade cabeceras necesarias (X-Forwarded-*)
- Justificación:
    - Mejora el rendimiento en serving de HTTP
    - Permite desacoplar la capa web del backend
    - Es el estándar en arquitecturas productivas

Flujo de la aplicación

1. El usuario accede a la web a través del ALB
2. El tráfico llega a Nginx en la instancia EC2
3. Nginx redirige la petición a Gunicorn
4. Gunicorn ejecuta la aplicación Flask
5. Flask:
    - Inserta los datos en RDS
    - Envía una notificación a SNS
6. Se devuelve la respuesta al usuario

Gestión de configuración

La aplicación utiliza un archivo .env para gestionar variables sensibles:

- Endpoint de RDS
- Credenciales de base de datos
- ARN del topic SNS
- Región de AWS

Esto permite separar configuración de código, siguiendo buenas prácticas.

Frontend

El frontend consiste en una página HTML sencilla que:

- Presenta un formulario de creación de tickets
- Envía los datos mediante fetch en formato JSON
- Muestra una respuesta al usuario

Aunque es simple, cumple perfectamente su función dentro del flujo de la aplicación.

Automatización y AMI

Todo el proceso de instalación se automatiza mediante un script que:

- Instala dependencias
- Configura la aplicación
- Levanta servicios (Gunicorn + Nginx)

Posteriormente, se genera una **AMI personalizada**, que es utilizada por el Auto Scaling Group.

Esto garantiza que:

- Todas las instancias tengan exactamente la misma configuración
- El escalado sea rápido y consistente
- Se siga un enfoque realista de despliegue en entornos productivos

Resumen

La aplicación sigue una arquitectura robusta y estándar en entornos cloud:

- **Flask** → lógica de negocio
- **Gunicorn** → ejecución del backend
- **Nginx** → gestión de tráfico HTTP
- **RDS** → persistencia de datos
- **SNS** → notificaciones

Este diseño permite que la aplicación sea:

- Escalable
- Reproducible
- Fácil de mantener
- Alineada con buenas prácticas DevOps

Script a partir del cual se ha creado la AMI del Launch Template

El siguiente enlace a GitHub contiene el **script completo** utilizado para la creación de la AMI empleada en el Launch Template del Auto Scaling Group.

En este script se define de forma automatizada toda la configuración de la instancia, incluyendo la instalación de dependencias, la **configuración del servidor web con Nginx**, la ejecución de la aplicación mediante **Gunicorn** y el despliegue tanto del **backend en Flask** como de**l frontend** de la aplicación.

Esto permit**e reproducir de manera consistente el entorno de ejecución de Penguin On The Way**, garantizando homogeneidad entre instancias y facilitando el escalado automático de la infraestructura.

En el enlace se puede encontrar tanto el script listo para ejecutarse como el script comentado sección a sección, para todo aquel interesado en comprender a fondo su funcionamiento:

https://github.com/CarlosFdezSalazar/Penguin-On-the-Way- (script-bash-ami-potw y script-bash-ami-potw-explicado)

Seguridad y control de accesos

IAM Roles y políticas de mínimo privilegio

Security Groups y NACLs

HTTPS y certificados en ALB

Seguridad y control de accesos

La seguridad es un componente esencial en cualquier arquitectura cloud. Esta sección describe las estrategias implementadas para proteger la infraestructura y los datos de la aplicación “Penguin On The Way!”, asegurando que los recursos sean accesibles únicamente por los agentes y servicios autorizados.

Se abordan tres aspectos clave:

1. **IAM Roles y políticas de mínimo privilegio**, garantizando que cada recurso y servicio solo tenga los permisos necesarios para su operación.
2. **Security Groups y NACLs**, que controlan de manera granular el tráfico entrante y saliente a nivel de recurso y subnet, asegurando aislamiento entre capas.
3. **HTTPS y certificados en ALB**, indicando la importante de mantener comunicaciones seguras entre los usuarios finales y la capa de aplicación.

El objetivo de esta sección es documentar cómo se ha implementado la seguridad de manera integral, considerando buenas prácticas y limitaciones del entorno de laboratorio (AWS Academy Learner Lab), con el fin de asegurar la confidencialidad, integridad y disponibilidad de los recursos.

IAM Roles y políticas de mínimo privilegio

La gestión de identidades y accesos en AWS es un pilar fundamental de la seguridad en la nube. Por ello, es importante tener siempre presente el principio de **mínimo privilegio**, es decir, que cada recurso y usuario solo tenga los permisos estrictamente necesarios para realizar sus tareas, reduciendo el riesgo de accesos no autorizados o acciones accidentales sobre la infraestructura.

Uso de IAM Roles

Debido a que el proyecto se desarrolla dentro de **AWS Academy Learner Lab**, el entorno está limitado y solo permite el uso de un rol predefinido denominado “Lab Role”. Este rol proporciona permisos amplios para crear y gestionar recursos dentro del laboratorio, lo que simplifica la implementación pero **no representa una práctica óptima de seguridad**.

En un entorno de producción, lo recomendable sería:

- **Definir roles específicos para cada tipo de recurso**, por ejemplo:

    - Rol para instancias EC2 con permisos mínimos necesarios para conectarse a RDS y SNS.
    - Rol para Lambda o servicios de automatización con permisos limitados a los recursos que necesita manipular.
    - Rol para CloudFormation con permisos solo para crear y modificar los stacks correspondientes.
- **Asignar políticas de IAM granuladas**, utilizando políticas gestionadas por AWS (idealmente) o personalizadas, limitando acciones, recursos y condiciones según la función del rol.
- **Aplicar el principio de separación de responsabilidades**, de modo que ningún rol tenga permisos innecesarios que puedan comprometer la seguridad de otros recursos.

Limitaciones del entorno

Aunque en este proyecto solo se utiliza “Lab Role”, la documentación y la estructura de los YAML de CloudFormation están diseñados para **facilitar la migración a un escenario real**, donde se podrían aplicar roles individuales para cada recurso, respetando el principio de mínimo privilegio y aumentando la seguridad global de la infraestructura.

Security Groups y NACLs

La protección de la infraestructura de “Penguin On The Way!” se ha implementado mediante **Security Groups (SG)** y **Network ACLs (NACLs)**, asegurando un control de acceso granular tanto a nivel de recurso como a nivel de subnet. Estas herramientas permiten definir qué tráfico está permitido hacia y desde cada capa de la arquitectura, reforzando la seguridad y el aislamiento entre componentes.

Security Groups

Los Security Groups actúan como **firewalls virtuales a nivel de recurso**, regulando el tráfico entrante (ingress) y saliente (egress) de instancias EC2, bases de datos RDS, el Application Load Balancer y el VPC Endpoint de SNS.

La configuración aplicada es la siguiente:

- **ALB Security Group**: permite tráfico HTTP (en un entorno ideal debería ser tráfico HTTPS) desde internet (puerto 80), garantizando que los usuarios puedan acceder a la aplicación. Todo tráfico saliente se permite hacia cualquier destino, asegurando conectividad con EC2 y otros servicios.

- **EC2 Security Group**: permite únicamente tráfico HTTP desde el ALB, bloqueando cualquier acceso directo desde internet, y permite tráfico saliente hacia RDS, SNS y otros servicios necesarios. Esto asegura que la capa de aplicación solo reciba solicitudes autorizadas.
- **RDS Security Group**: restringe el acceso únicamente a las EC2 de la capa de aplicación, garantizando que la base de datos no sea accesible desde internet ni desde otros recursos no autorizados.
- **VPC Endpoint Security Group (SNS)**: configurado para permitir todo el tráfico proveniente del Security Group de EC2, garantizando que la aplicación pueda publicar eventos en SNS sin exponer el endpoint a tráfico externo.

Este enfoque permite mantener una defensa en profundidad, donde cada recurso valida y filtra el tráfico de forma independiente.

Network ACLs (NACLs)

Los NACLs proporcionan **filtrado adicional a nivel de subnet**, funcionando como barreras que controlan el tráfico entrante y saliente antes de llegar a los recursos.

En esta arquitectura se han definido tres NACLs principales:

1. **Capa pública**: permite tráfico HTTP hacia el ALB y respuestas de tráfico efímero (ephemeral ports) para mantener las conexiones TCP.
2. **Capa de aplicación**: permite tráfico HTTP desde la VPC (ALB) hacia las EC2 y permite el tráfico de respuesta a puertos efímeros, asegurando que las instancias puedan comunicarse entre ellas y con la base de datos.
3. **Capa de datos**: permite únicamente tráfico MySQL desde la capa de aplicación y el tráfico de retorno efímero, protegiendo la base de datos de accesos no autorizados.

Justificación y buenas prácticas

- **Segregación de capas**: la combinación de SG y NACL garantiza que cada capa (pública, aplicación, datos) esté aislada y solo se comunique con las capas necesarias.
- **Tráfico mínimo necesario**: las reglas están definidas siguiendo el principio de **mínimo privilegio**, permitiendo únicamente el tráfico indispensable para la operación de la aplicación.
- **Auditoría y trazabilidad**: el diseño facilita el análisis de flujos de tráfico mediante VPC Flow Logs, lo que permite identificar intentos de acceso no autorizados y monitorizar la actividad de la red.

En entornos productivos, esta estrategia de control de acceso se puede complementar con **roles de IAM más granulares**, AWS WAF, y servicios de inspección de tráfico para incrementar la seguridad de la plataforma.

HTTPS y certificados en ALB

En un entorno de producción, la comunicación entre los clientes y el Application Load Balancer (ALB) debería estar protegida mediante HTTPS, utilizando certificados TLS/SSL para cifrar los datos en tránsito. Esto garantiza confidencialidad e integridad de la información, evitando que sea interceptada o modificada.

En este proyecto de laboratorio no se ha configurado un certificado HTTPS en el ALB por motivos prácticos y de coste, ya que la gestión de certificados válidos y el tráfico cifrado no son estrictamente necesarios para el objetivo didáctico. No obstante, la arquitectura está diseñada de manera que incorporar HTTPS en el ALB sería directo: bastaría con asociar un certificado gestionado por AWS Certificate Manager (ACM) al listener HTTPS del ALB.

El uso de HTTPS con certificados válidos sigue siendo la **mejor práctica** en cualquier despliegue real.

Observabilidad y monitorización

CloudWatch: métricas y dashboards

Logs centralizados: ALB, EC2 y RDS

Alarmas y notificaciones con SNS

Registro y auditoría de accesos (CloudTrail y VPC Flow Logs)

Observabilidad y monitorización

La observabilidad y monitorización son pilares esenciales para mantener la disponibilidad, el rendimiento y la seguridad de cualquier sistema en la nube. Estas prácticas permiten detectar problemas de manera temprana, comprender el comportamiento de los servicios y reaccionar proactivamente ante incidencias.

En este proyecto, la estrategia de observabilidad se estructura en varias capas complementarias:

1. **CloudWatch: métricas y dashboards** – seguimiento centralizado del estado y rendimiento de instancias EC2, RDS y otros recursos de infraestructura.
2. **Logs centralizados: ALB, EC2 y RDS** – recolección de registros de aplicaciones y servicios, facilitando análisis y auditoría.
3. **Alarmas y notificaciones con SNS** – alertas automáticas ante eventos críticos para permitir una respuesta rápida.
4. **Métricas de la aplicación y alertas personalizadas** – seguimiento de indicadores clave de negocio y de la salud de la aplicación.
5. **Registro y auditoría de accesos (CloudTrail y VPC Flow Logs)** – trazabilidad completa de las acciones sobre los recursos, mejorando la seguridad y cumpliendo buenas prácticas de cumplimiento.

Aunque el alcance del laboratorio es limitado, la arquitectura está diseñada para escalar hacia un entorno de observabilidad completo, incorporando dashboards más avanzados, métricas de negocio y alertas personalizadas según se necesite.

CloudWatch: métricas y dashboards

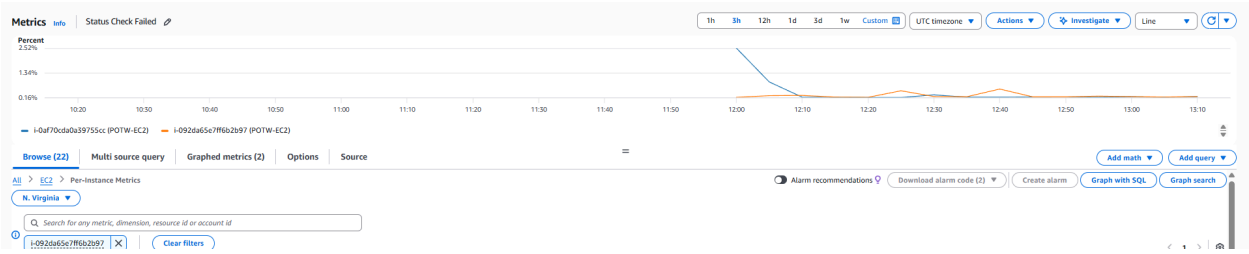
Con el objetivo de garantizar la observabilidad de la infraestructura, se ha implementado el uso de CloudWatch como servicio centralizado para la recopilación y análisis de métricas en tiempo real.

A través de CloudWatch se monitorizan los principales componentes del sistema, permitiendo evaluar el estado, rendimiento y comportamiento de cada capa de la arquitectura.

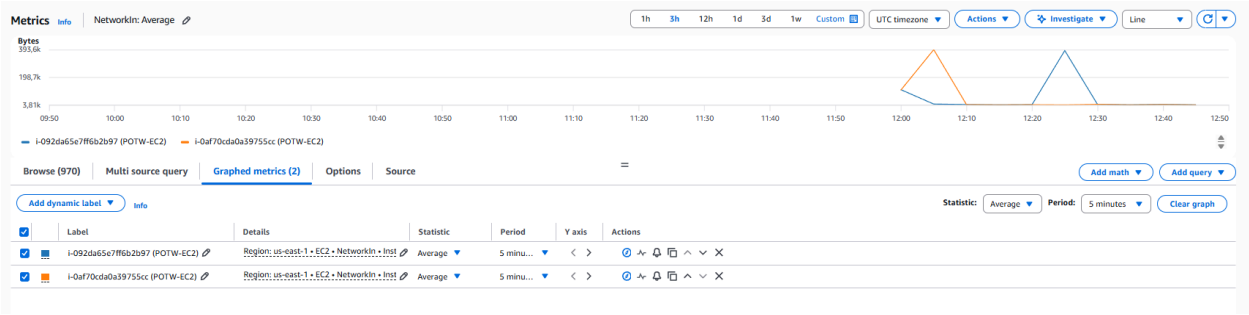
Métricas de las instancias (EC2)

Se monitorizan métricas clave de las instancias EC2, entre las que destacan:

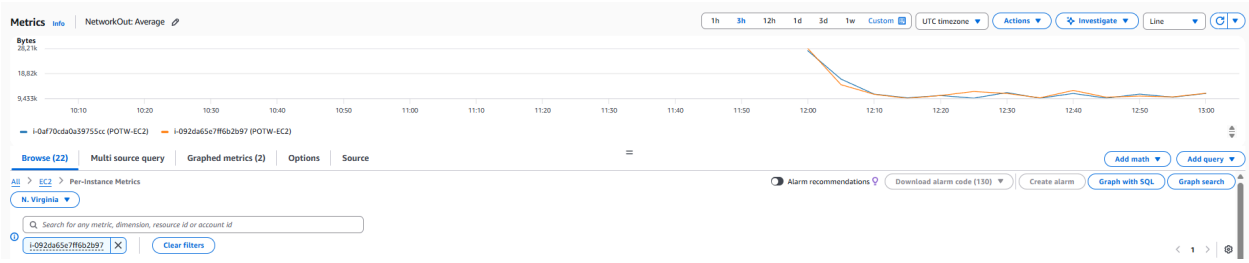
CPUUtilization, para medir la carga de procesamiento de la aplicación.



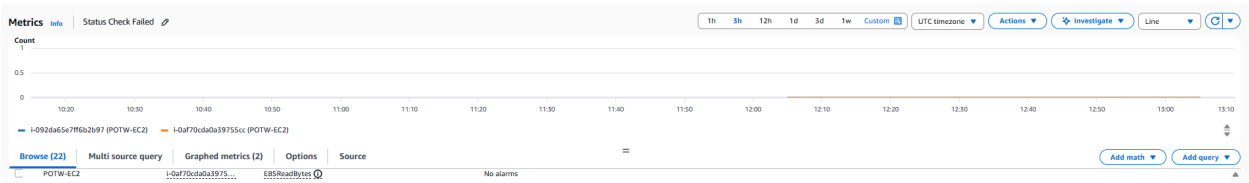
NetworkIn, representa el volumen de datos entrante hacia las instancias, principalmente desde el Application Load Balance.



NetworkOut, refleja el tráfico saliente desde las instancias hacia otros servicios, como clientes, base de datos o servicios de AWS.



StatusCheckFailed, para detectar problemas de salud en las instancias.

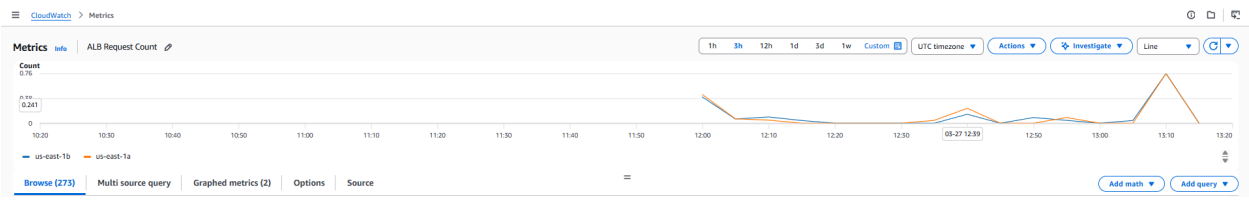


Estas métricas permiten validar el correcto funcionamiento del Auto Scaling y detectar posibles cuellos de botella en la capa de aplicación.

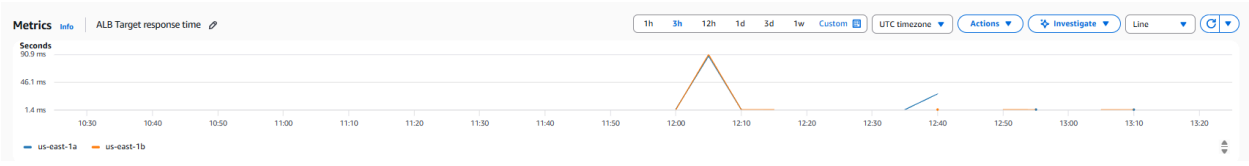
Métricas del balanceador de carga (ALB)

El Application Load Balancer proporciona métricas esenciales para entender el tráfico entrante y el comportamiento de la aplicación:

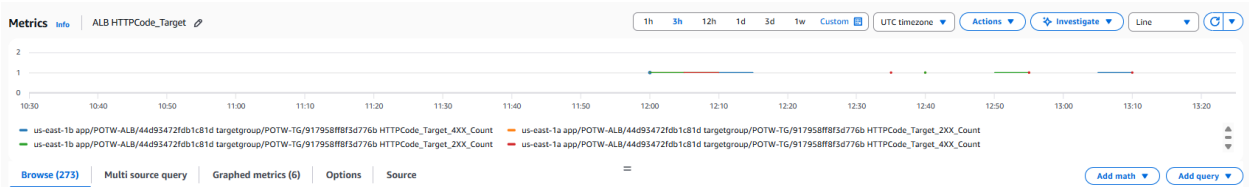
RequestCount, que refleja el número de peticiones recibidas.



TargetResponseTime, que mide la latencia de respuesta del backend.



HTTPCode_Target (2XX, 4XX, 5XX), que permite identificar errores en la aplicación.

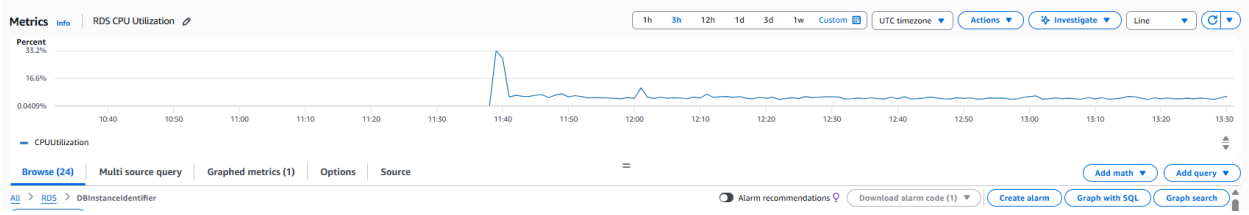


Estas métricas son fundamentales para evaluar la experiencia del usuario y detectar incidencias en el servicio.

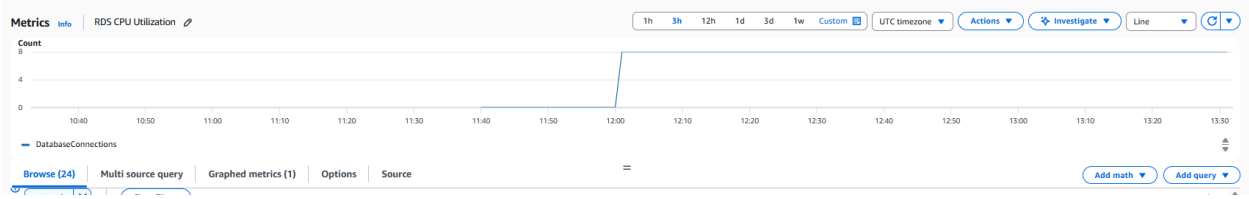
Métricas de la base de datos (RDS)

En la capa de datos, se monitoriza el rendimiento de la base de datos mediante:

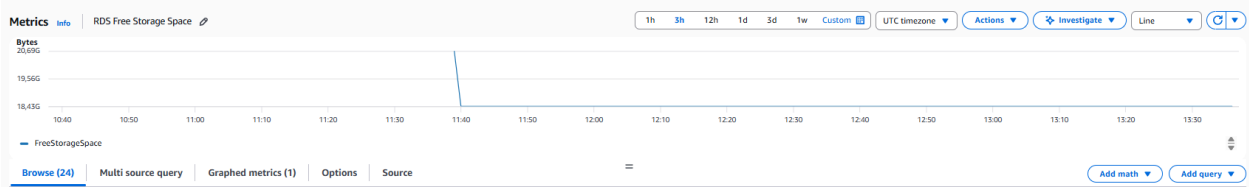
CPUUtilization, para analizar la carga del motor de base de datos.



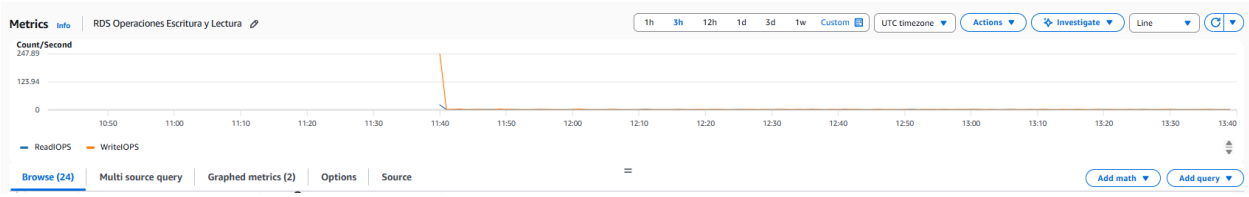
DatabaseConnections, que indica el número de conexiones activas.



FreeStorageSpace, para controlar la capacidad disponible.



ReadIOPS y WriteIOPS, que reflejan la actividad de lectura y escritura.

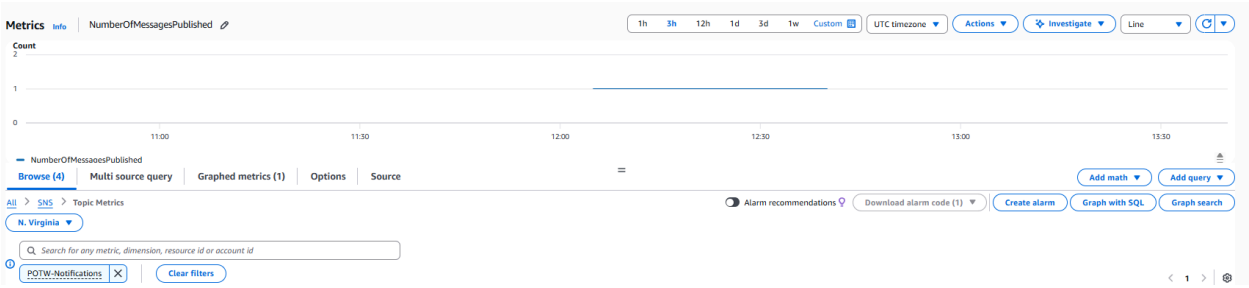


Estas métricas permiten garantizar la estabilidad y disponibilidad de los datos, así como anticipar necesidades de escalado.

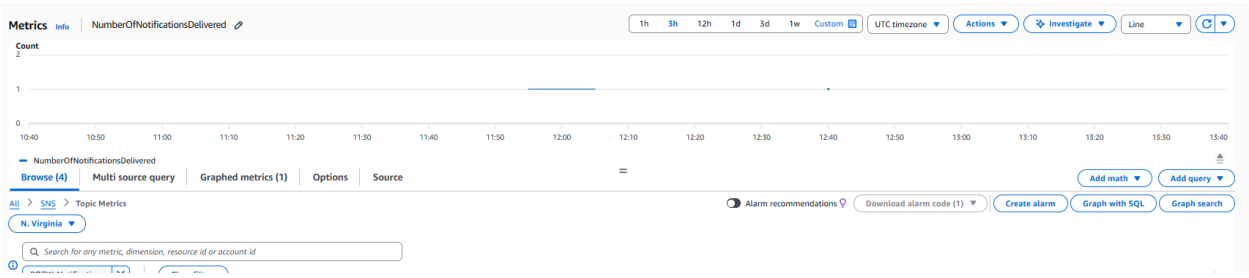
Métricas del sistema de notificaciones (SNS)

El servicio SNS se monitoriza para validar el correcto funcionamiento del sistema de notificaciones:

NumberOfMessagesPublished, que indica el número de mensajes enviados al tópico.



NumberOfNotificationsDelivered, que confirma la entrega de las notificaciones.



Esto permite verificar el flujo completo de eventos desde la aplicación hasta la notificación final a los administradores.

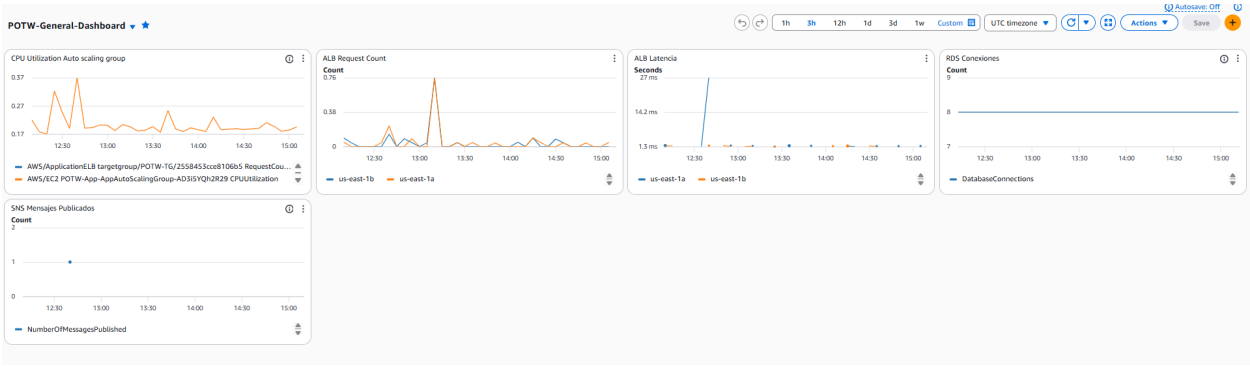
Dashboard centralizado

Con el objetivo de facilitar la monitorización en tiempo real, se ha creado un dashboard personalizado en CloudWatch que agrupa las métricas más relevantes de la infraestructura.

Este dashboard incluye información de:

- Uso de CPU de las instancias EC2.
- Número de peticiones y latencia del ALB.
- Conexiones activas en RDS.
- Mensajes publicados en SNS.

De esta manera, se dispone de una visión global del sistema que permite identificar rápidamente anomalías y tomar decisiones operativas de forma eficiente.



Logs centralizados: ALB, EC2 y RDS

La **centralización de logs** constituye un pilar fundamental dentro de la observabilidad, ya que permite analizar en detalle el comportamiento interno de cada componente de la infraestructura, facilitando tanto la monitorización como la resolución de incidencias.

En este proyecto, se ha implementado una estrategia de logging centralizado que abarca las principales capas de la arquitectura: el **Application Load Balancer (ALB)**, las **instancias EC2** y la **base de datos RDS**. Cada uno de estos componentes genera registros que aportan información específica sobre el flujo de peticiones, la ejecución de la aplicación y el estado de la base de datos.

La recopilación y almacenamiento de estos logs se realiza mediante servicios gestionados de AWS como **CloudWatch Logs y/o Amazon S3**, permitiendo su consulta, filtrado y análisis. Esto facilita la detección de errores, el análisis de patrones de uso y la auditoría del sistema, mejorando la capacidad de respuesta ante incidentes. Se muestra de esta manera la capacidad de gestionar logs en S3.

Para esta parte del proyecto, se hará uso de un bucket llamado “potw-bucket” que contiene la siguiente política para permitir escritura de los logs de ALB.

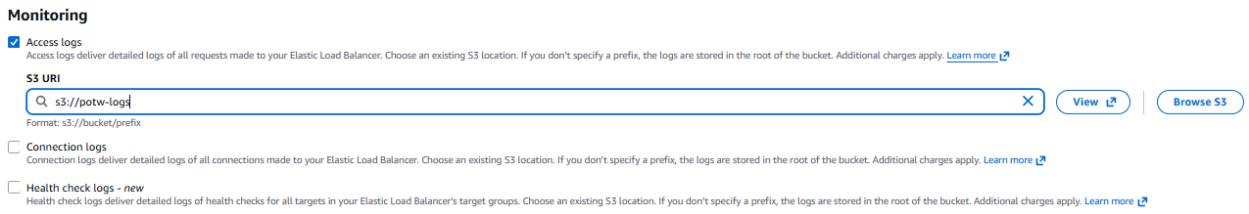
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logdelivery.elasticloadbalancing.amazonaws.com"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::potw-logs/ALB-Logs/*"
    }
  ]
}
```

Logs de ALB

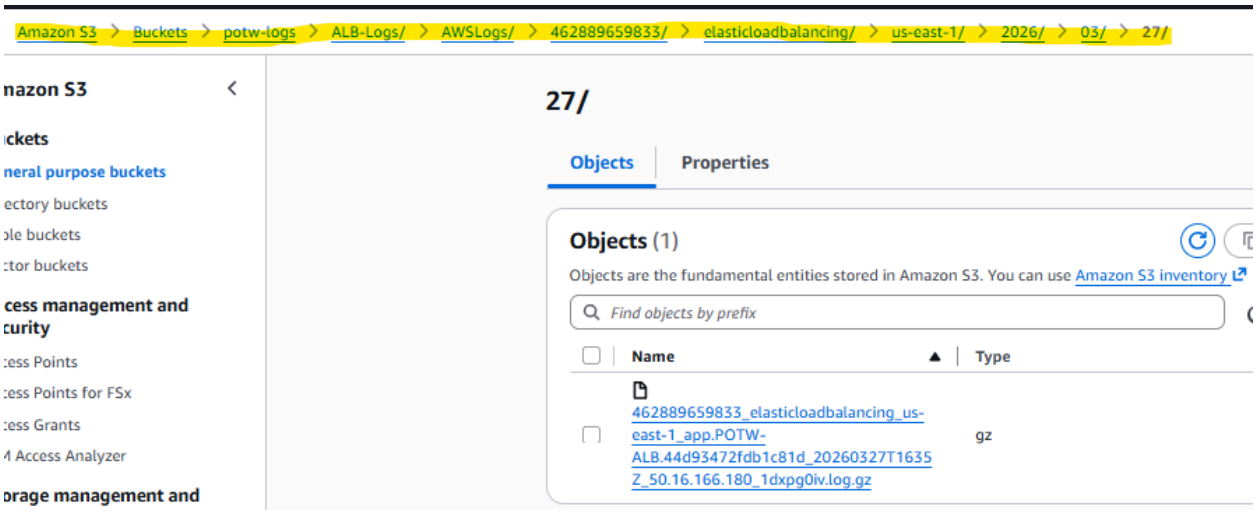
Primero se debe activar la monitorización en el recurso, para ello se pueden activar 3 tipos de log:

- **Access logs** → Permite analizar todas las peticiones HTTP/S, la IP de origen, el target, códigos de respuesta…
- **Connection logs** → Permite medir conexiones TCP/SSL a bajo nivel.
- **Health check logs** → Permite depurar problemas con el Target Group.

Por cuestiones prácticas y de necesidad en POTW solo se han configurado los “Access Logs”.



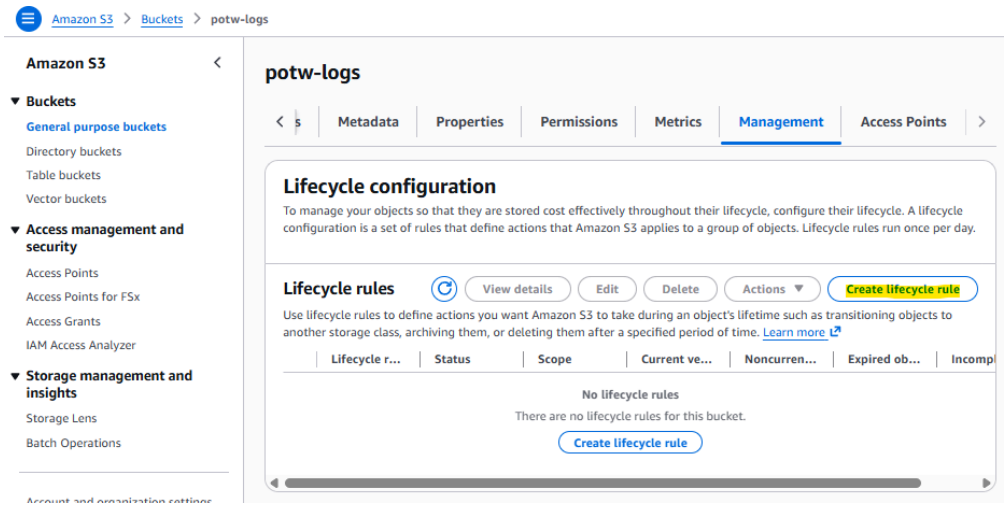
Se observa que tras aplicar la política al bucket y habilitar los Access Logs, se crea una carpeta en el bucket donde se almacenan los logs de ALB ordenados por año, mes y días:

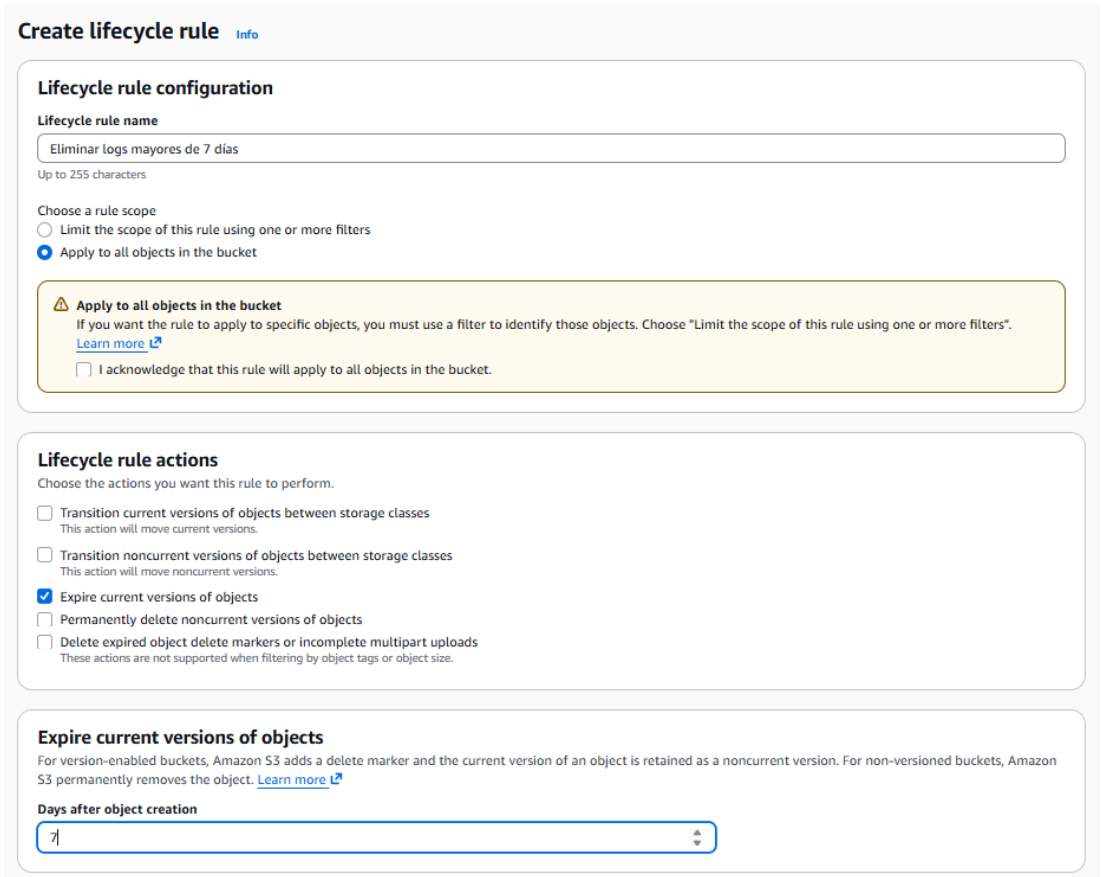


Ciclo de vida de logs en S3 para ALB

Se establecerá un ciclo de vida de tal manera que, tras 7 días, todos los logs almacenados se eliminen, así se evita gastar recursos en almacenar logs obsoletos.

Para ello se creará una “Lifecycle rule”:





Logs de RDS

En esta sección se describe cómo centralizar los logs generados por las instancias de RDS mediante **CloudWatch Logs**, facilitando la monitorización en tiempo real, la detección de errores y la observabilidad de la base de datos.

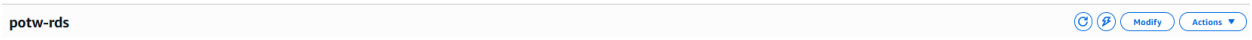
A diferencia de otros servicios como el ALB, que almacenan sus logs directamente en S3, RDS se integra de forma nativa con CloudWatch Logs, permitiendo consultar los registros de manera inmediata y configurar alertas sobre eventos relevantes.

En este proyecto se incluyen los logs más relevantes para la aplicación:

- **Error logs**: registro de errores y eventos del motor de base de datos, fundamentales para la detección de fallos..

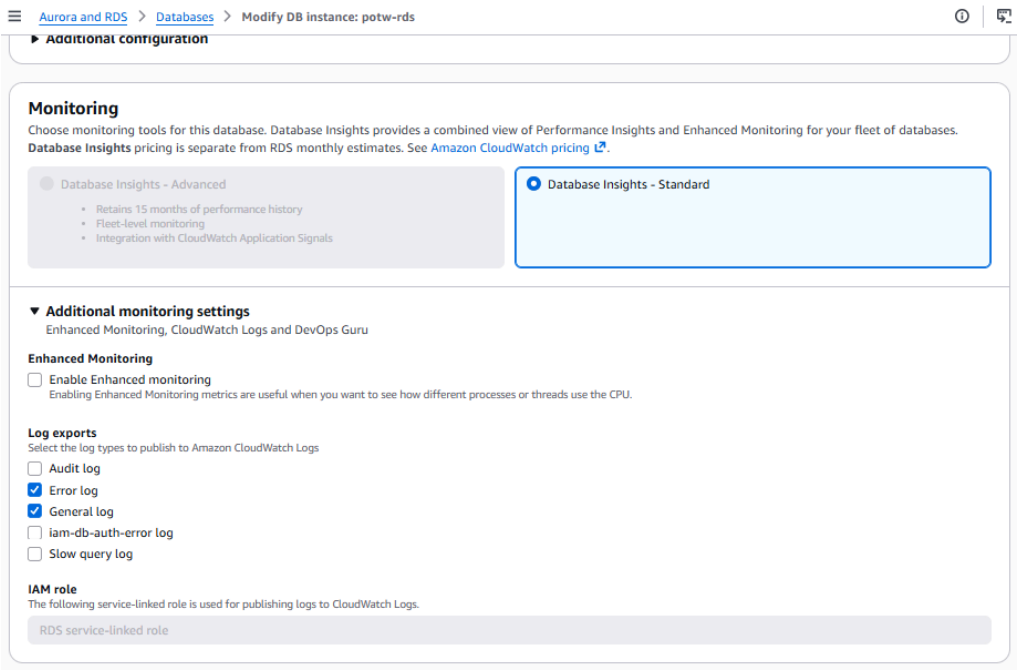
Siguiendo prácticas habituales en entornos de producción, estos logs se mantienen en CloudWatch Logs con una política de retención definida, priorizando la monitorización operativa y evitando la exportación a S3 al no existir requisitos de almacenamiento a largo plazo.

Para activar el envío de logs a Cloud Watch se debe ir al apartado “modify” en RDS

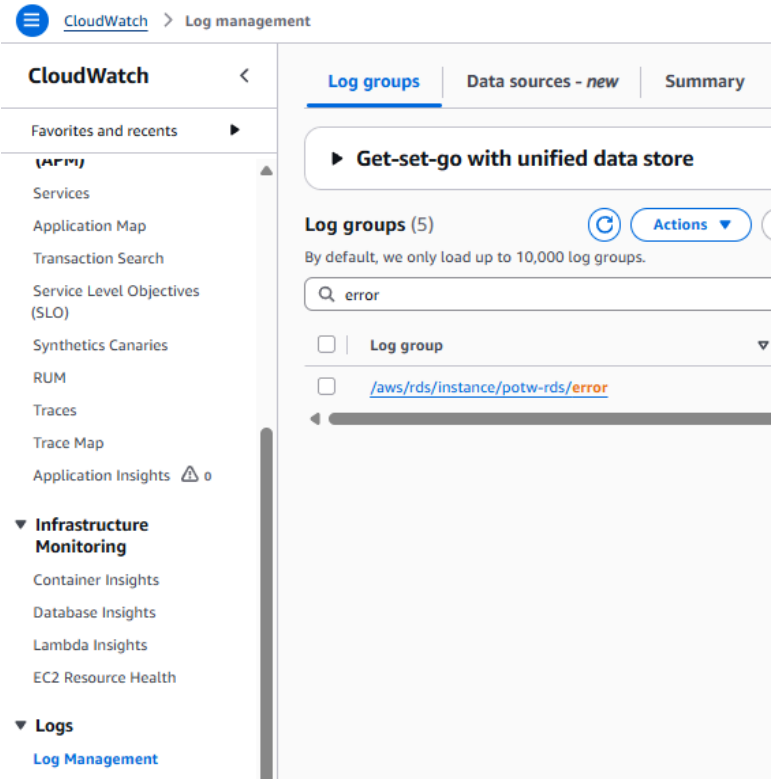


Una vez dentro, como en el caso de POTW se quiere auditar errores en la base de datos, se habilitan “Error logs”, en la sección de “Monitoring”.

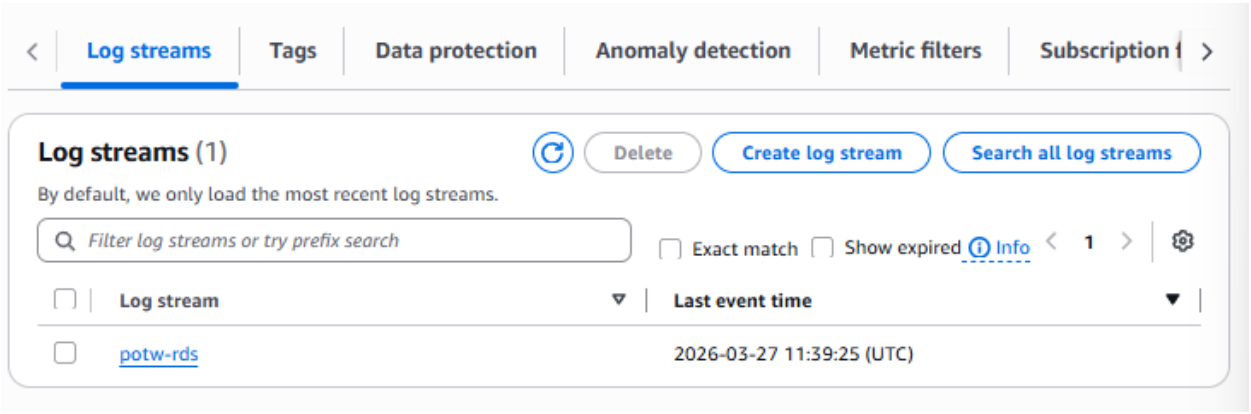
Opcionalmente se podrían habilitar los “general log” si se desea monitorizar las escrituras y otras operaciones, pero dada la simpleza del rol que cumple la base de datos en este proyecto, únicamente se almacenarán y analizarán los logs relativos a errores.

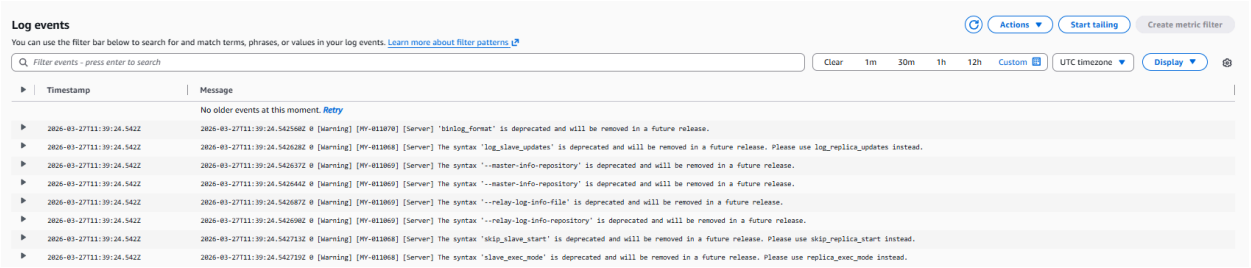


En Cloud Watch, en “Log Management → Log Groups” se observan los logs enviados por RDS a CLoud Watch, se muestra a continuación los logs de “error”:



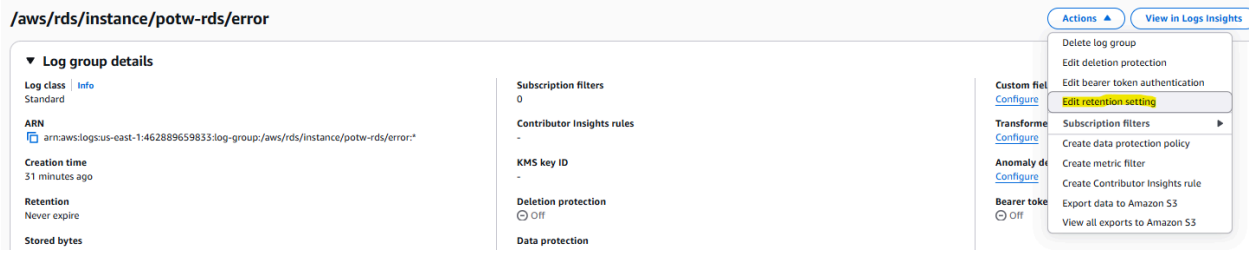
Si se accede al grupo, se observa en la sección de “Log streams” los logs de la base de datos relativos a errores:

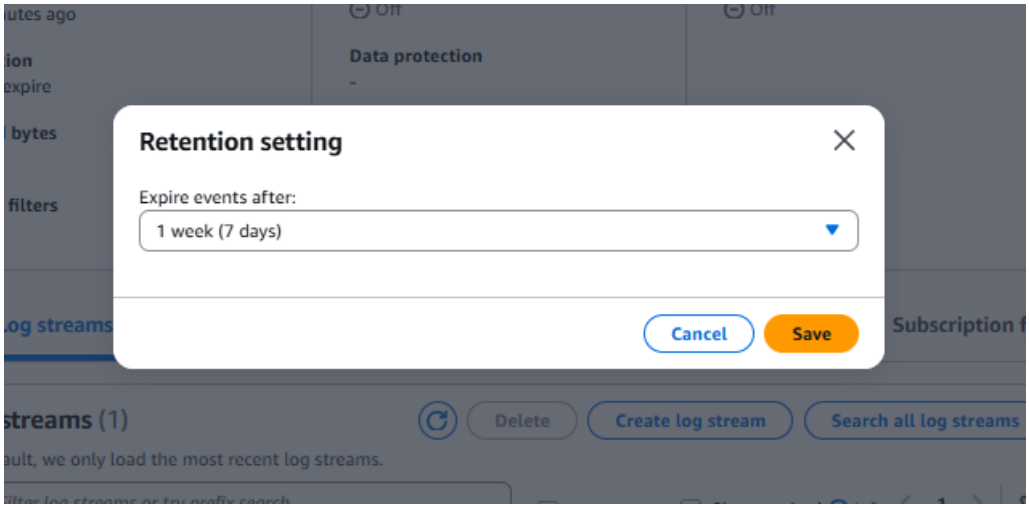




Política de retención de logs en Cloud Watch para RDS

Se establecerá una política de retención de los logs en Cloud Watch de 7 días siguiendo los siguientes pasos:





Logs de EC2

En este apartado se hace uso de Cloud Watch agent

Se instala en la máquina que se desea monitorizar CloudWatch Agent. Para ello se puede usar la siguiente documentación: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/manual-installation.html

Una vez instalado CloudWatch agent, se creará un JSON donde se indica que logs se desean enviar a CloudWatch.

```
 GNU nano 7.2                          /home/ubuntu/cloudwatch-agent-config.json
{
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/var/log/syslog",
            "log_group_name": "/potw/ec2/system",
            "log_stream_name": "{instance_id}"
          },
          {
            "file_path": "/var/log/auth.log",
            "log_group_name": "/potw/ec2/auth",
            "log_stream_name": "{instance_id}"
          }
        ]
      }
    }
  }
}
```

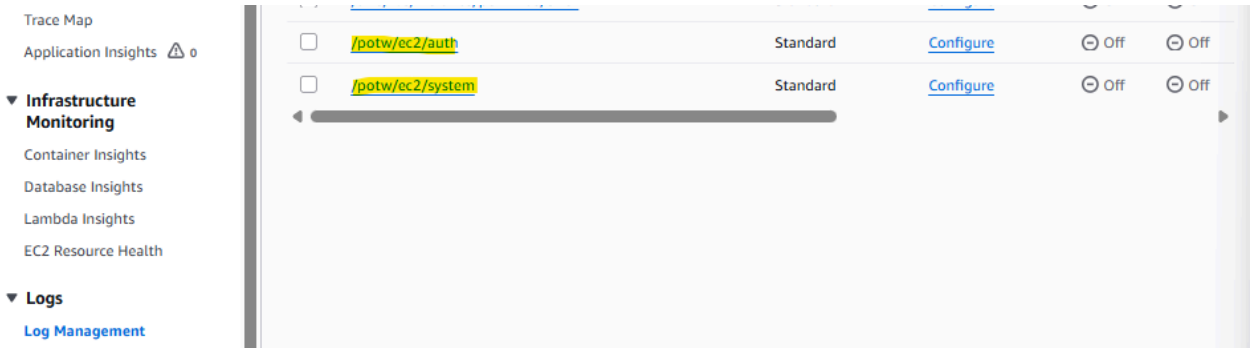
Se arranca el agente en la EC2:

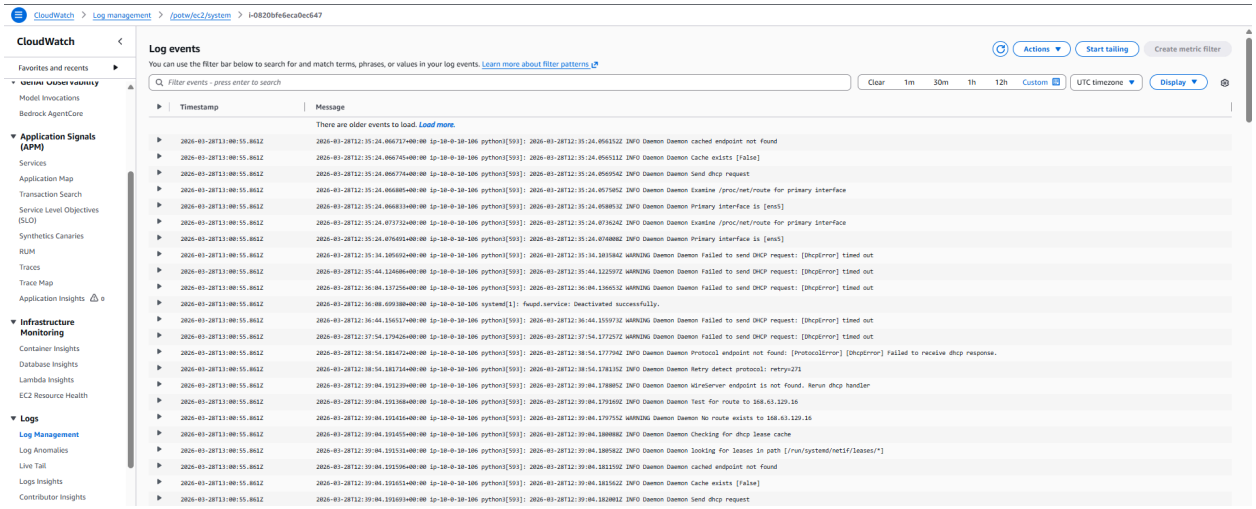
```
ubuntu@ip-10-0-10-106:~$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2
 -c file:/home/ubuntu/cloudwatch-agent-config.json
****** processing amazon-cloudwatch-agent ******
Starting config-downloader, this will map back to a call to amazon-cloudwatch-agent
Executing /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent with arguments: [config-downloader -download-s
ource file:/home/ubuntu/cloudwatch-agent-config.json -output-dir /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwat
ch-agent.d -config /opt/aws/amazon-cloudwatch-agent/etc/common-config.toml -multi-config default -mode ec2]I! Trying
to detect region from ec2
D! [EC2] Found active network interface
I! imds retry client will retry 1 times
Start configuration validation...
2026/03/28 12:55:08 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/fil
e_cloudwatch-agent-config.json.tmp ...
2026/03/28 12:55:08 I! Valid Json input schema.
2026/03/28 12:55:08 Configuration validation first phase succeeded
Starting config-translator, this will map back to a call to amazon-cloudwatch-agent
Executing /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent with arguments: [config-translator -input-dir
/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d -output /opt/aws/amazon-cloudwatch-agent/etc/amazon-cl
oudwatch-agent.toml -mode ec2 -config /opt/aws/amazon-cloudwatch-agent/etc/common-config.toml -multi-config default -
input /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json]I! Trying to detect region from ec2
D! [EC2] Found active network interface
I! imds retry client will retry 1 times
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -schematest -config /opt/aws/amazon-cloudwatch-agent/etc
/amazon-cloudwatch-agent.toml
Configuration validation second phase succeeded
Configuration validation succeeded
ubuntu@ip-10-0-10-106:~$
```

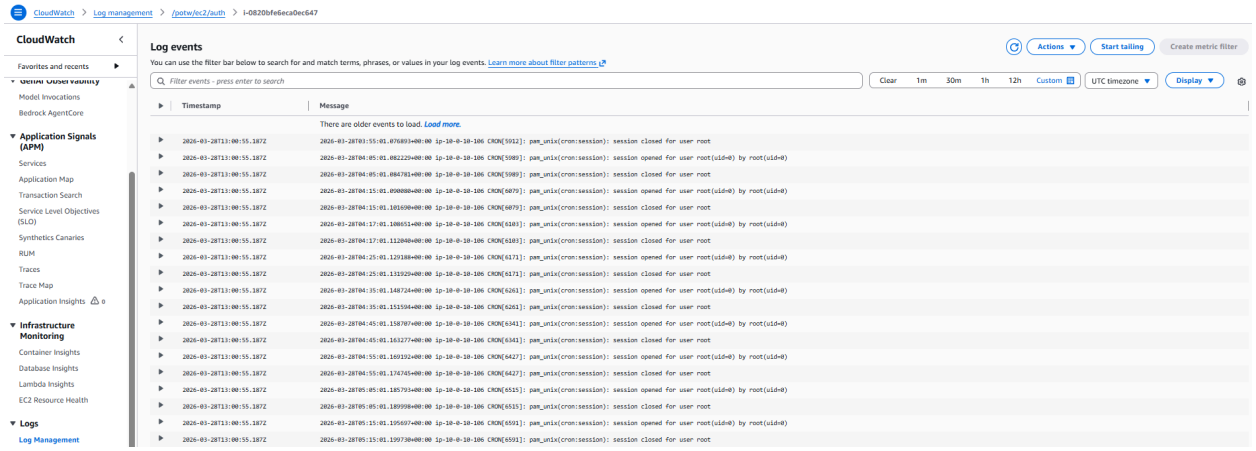
```
ubuntu@ip-10-0-10-106:~$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a start

****** processing amazon-cloudwatch-agent ******
Created symlink /etc/systemd/system/multi-user.target.wants/amazon-cloudwatch-agent.service → /etc/systemd/system/ama
zon-cloudwatch-agent.service.
ubuntu@ip-10-0-10-106:~$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
{
  "status": "running",
  "starttime": "2026-03-28T13:00:53+00:00",
  "configstatus": "configured",
  "version": "1.300064.1b1344"
}
ubuntu@ip-10-0-10-106:~$
```

A continuación, en CloudWatch se observa cómo se obtienen los logs de la EC2:







Se han seleccionado específicamente los logs de **syslog** y **auth** debido a su relevancia para la monitorización y auditoría de la infraestructura. Los **syslog** registran eventos del sistema y servicios críticos, permitiendo detectar errores, reinicios, caídas de servicios o problemas de configuración. Por su parte, los **auth logs** documentan intentos de acceso y autenticaciones, proporcionando visibilidad sobre la seguridad y posibles intentos de intrusión.

Alarmas y notificaciones con SNS

En esta sección se describe cómo se configuran **alarmas y notificaciones** para la infraestructura y la aplicación mediante **Amazon CloudWatch** y **SNS (Simple Notification Service)**.

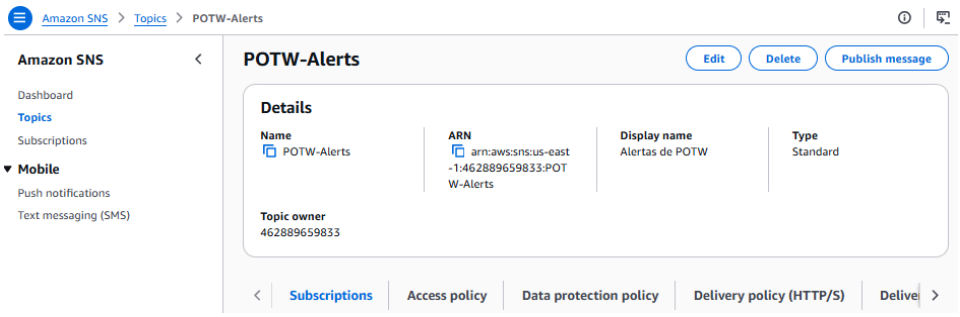
Las alarmas permiten detectar automáticamente condiciones críticas o anómalas, como un uso excesivo de CPU, memoria o conexiones en RDS, así como errores en la aplicación o el ALB.

Cuando se dispara una alarma, se envía una **notificación a un topic de SNS**, que puede distribuir mensajes vía correo electrónico (como será el caso en POTW), SMS o integraciones con otros sistemas de alerta.

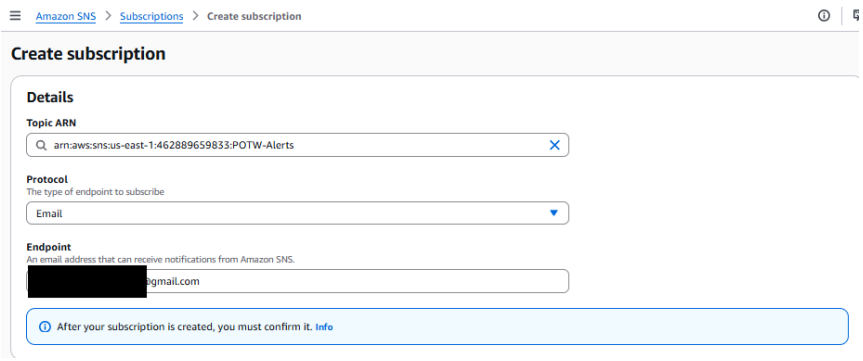
Este mecanismo garantiza que los administradores o responsables de la aplicación reciban avisos inmediatos, mejorando la capacidad de respuesta ante incidencias y la supervisión proactiva de la infraestructura.

Creación de un topic en SNS para alertas

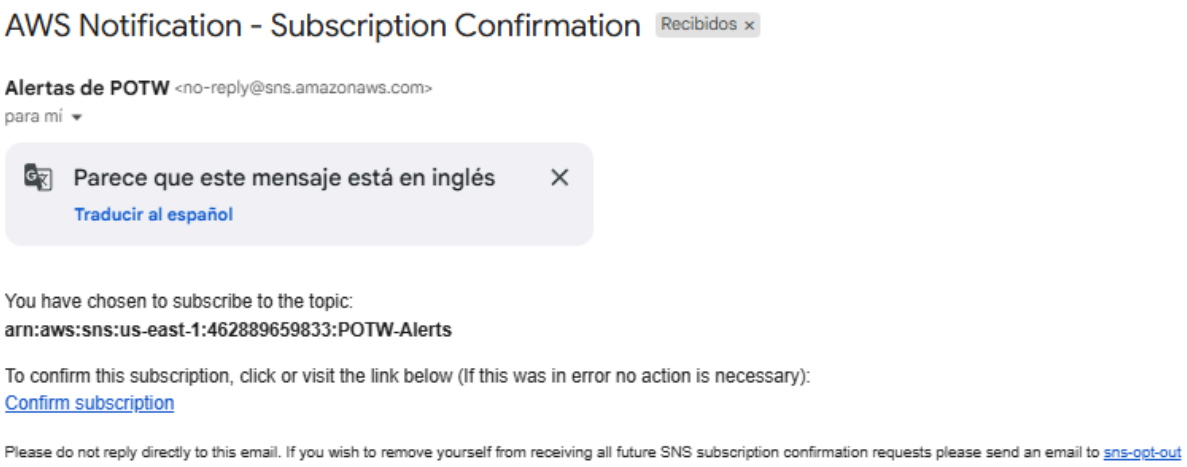
Como primer paso, se crea un topic en SNS, este topic llevará el nombre de “Alerts”, pues será el encargado de enviar las alertas a los administradores a través de correo electrónico.



Seguidamente, se crea una suscripción para el correo electrónico de los administradores:

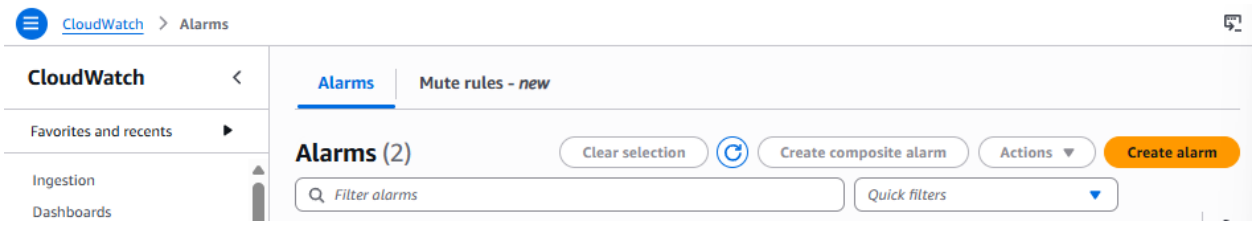


Se acepta la suscripción desde el correo electrónico:



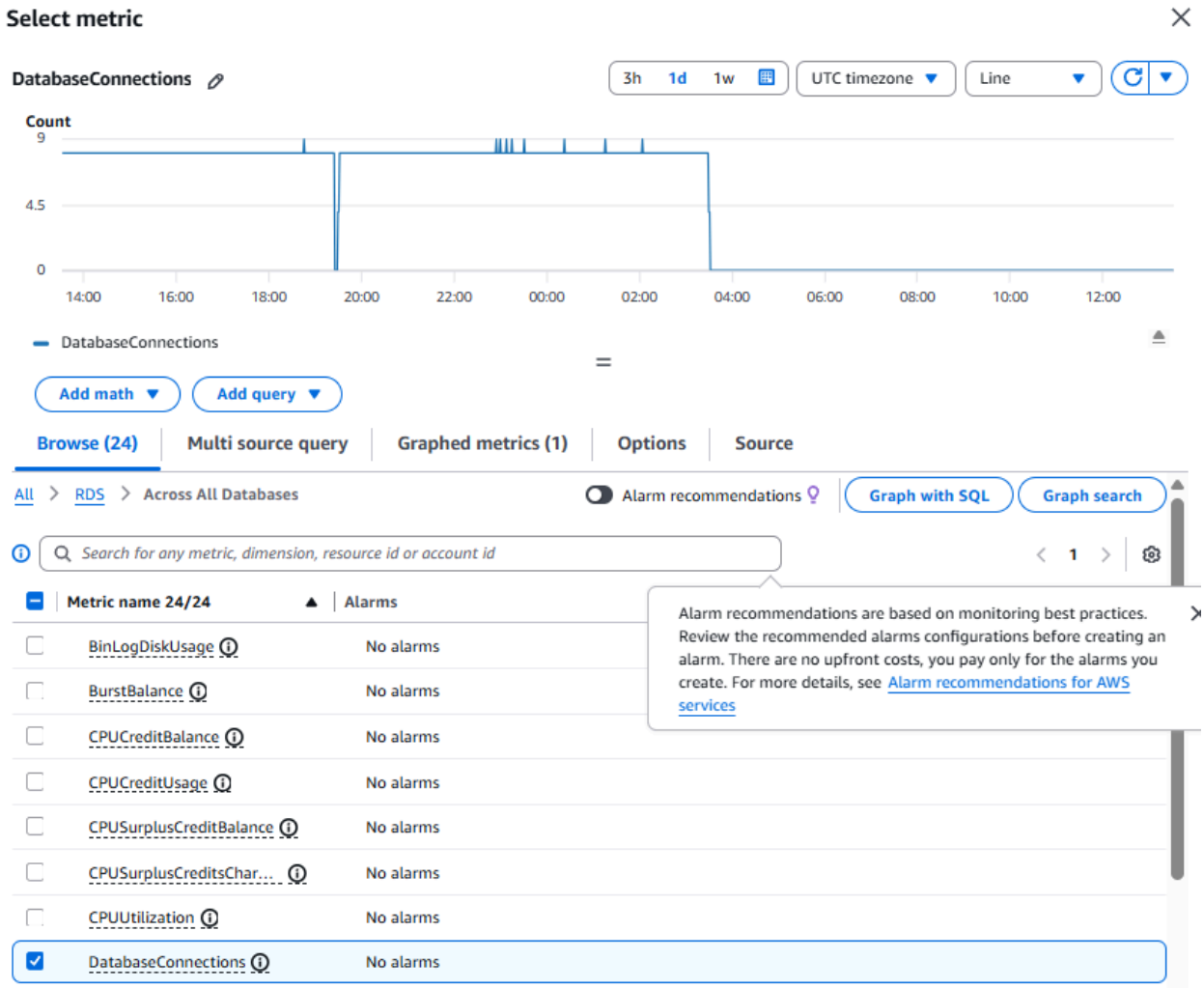
Creación de alarmas

Se crean desde CloudWatch:

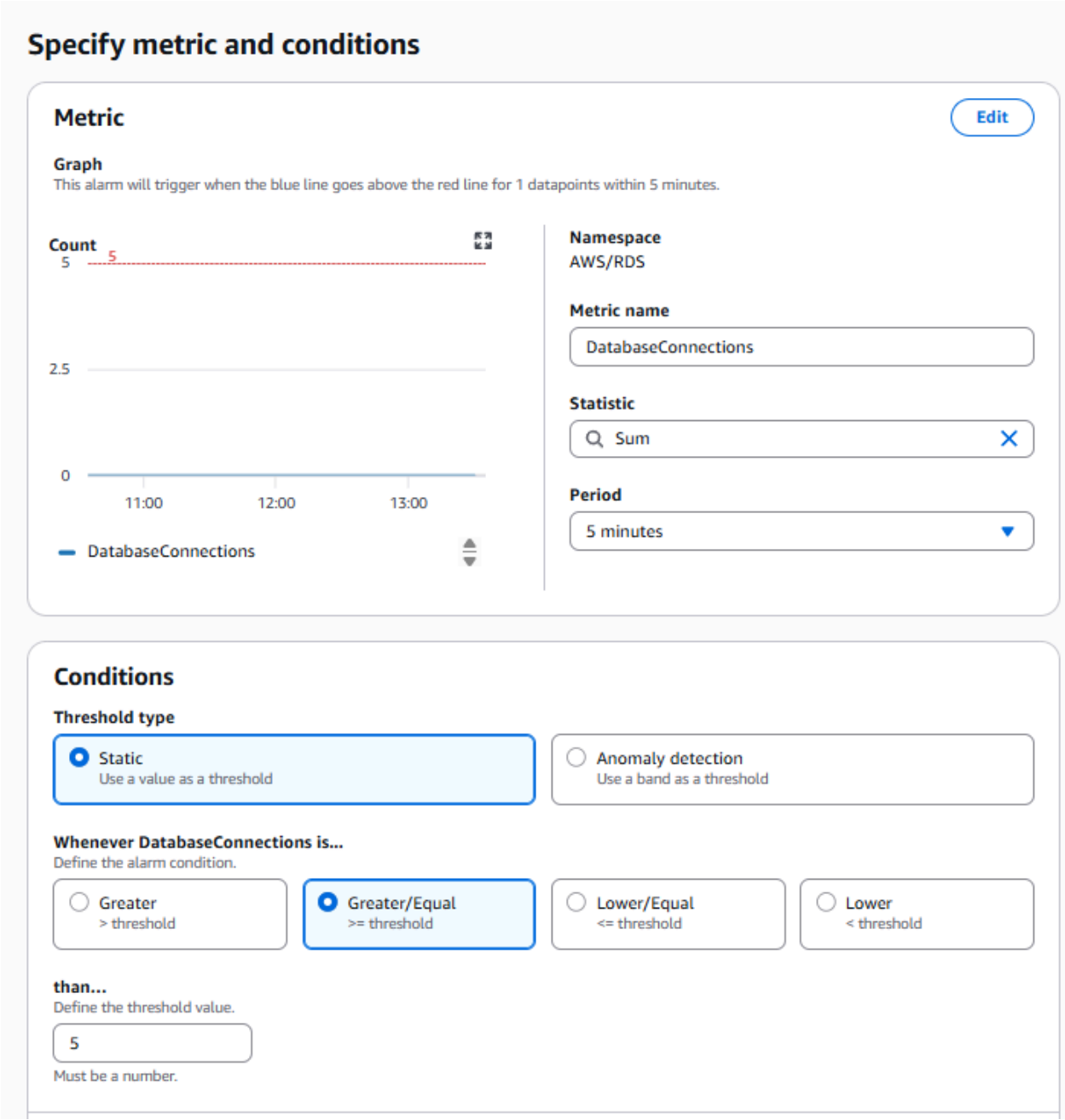


Alarma para RDS: Conexiones a BBDD

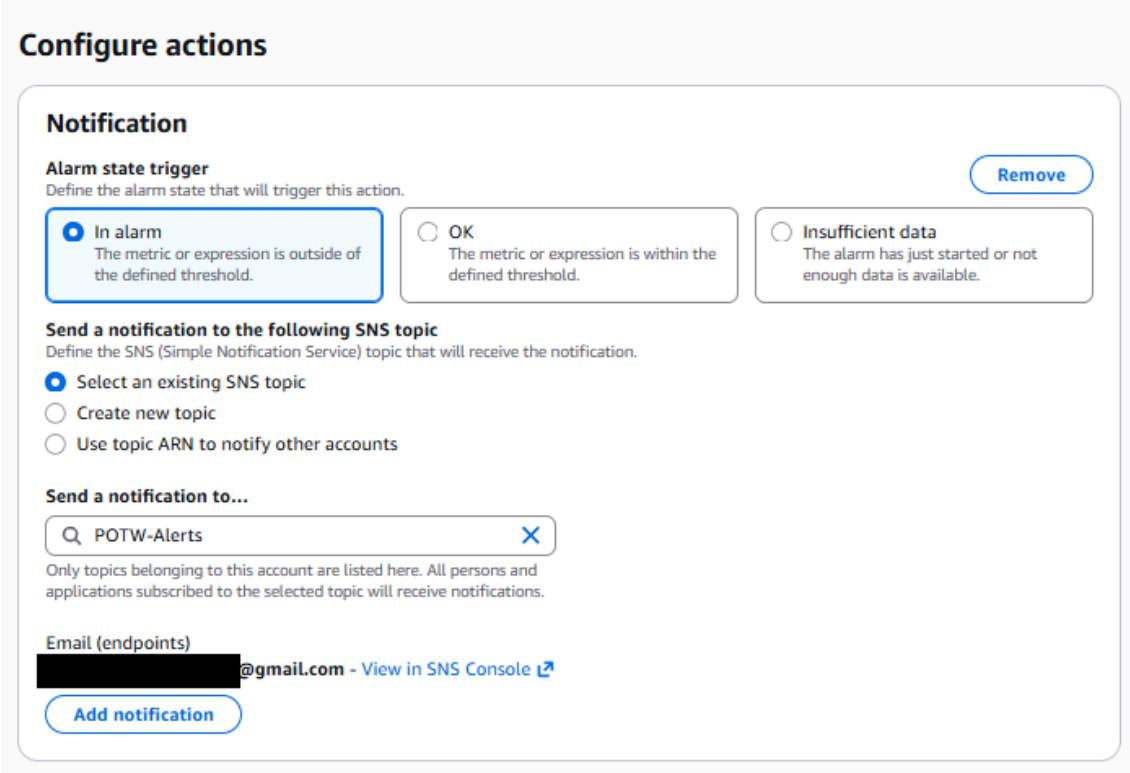
Para ello se crea la alarma sobre la métrica de “DatabaseConnections”



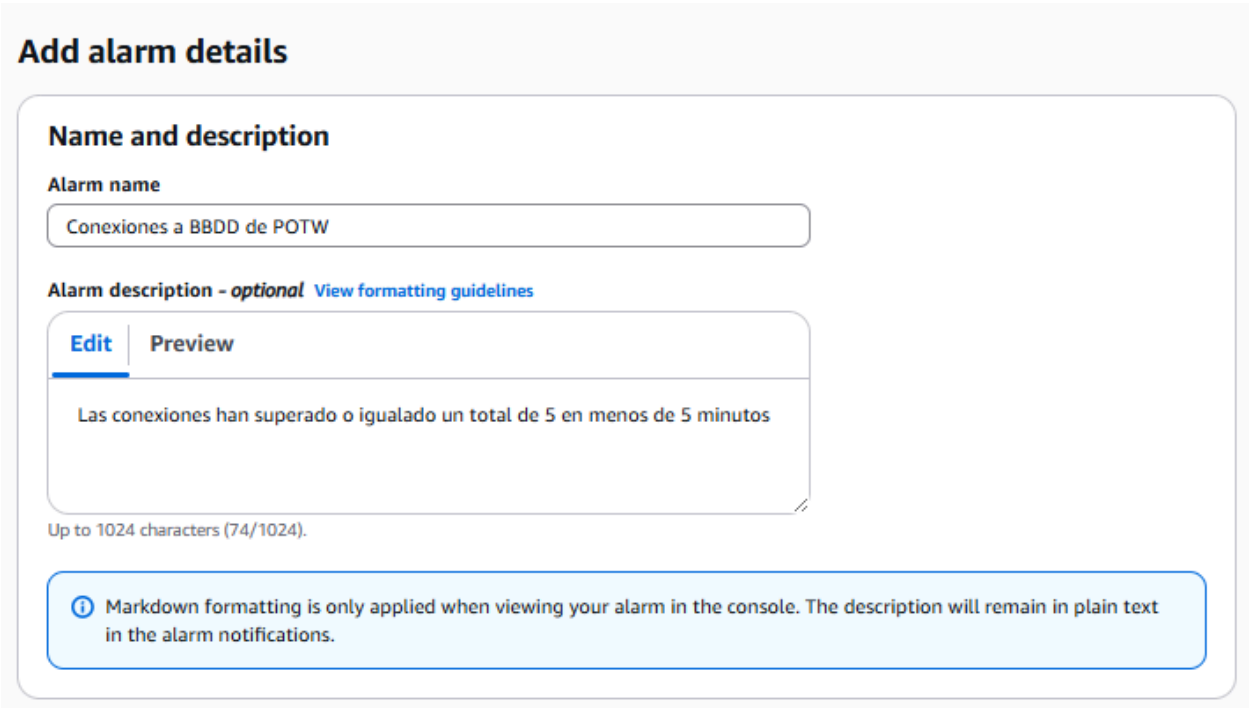
Se configura la alarma para que en caso de que las conexiones a la BBDD en un periodo de tiempo de 5 minutos superen o igualen a 5, se envíe una alarma..



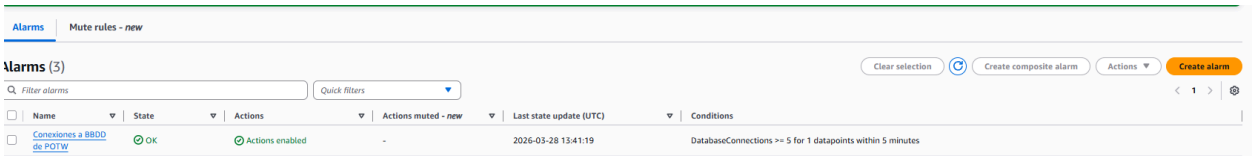
Se elige el topic creado anteriormente



Se añade un nombre a la alarma y una descripción:

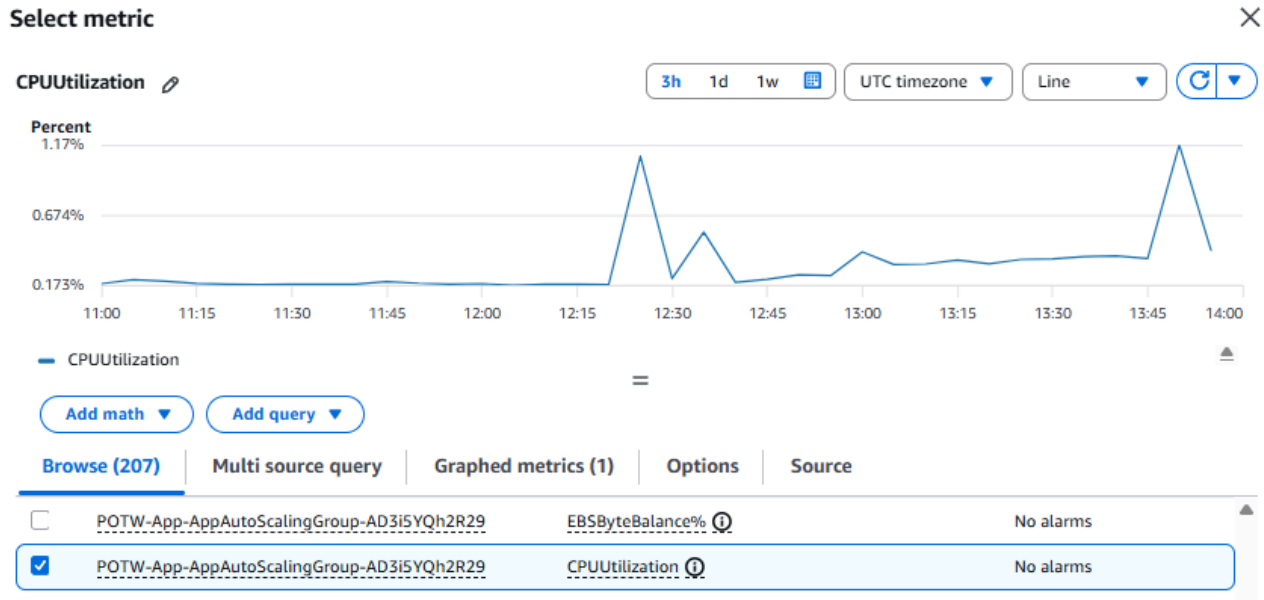


Se observa que la alarma se ha creado correctamente

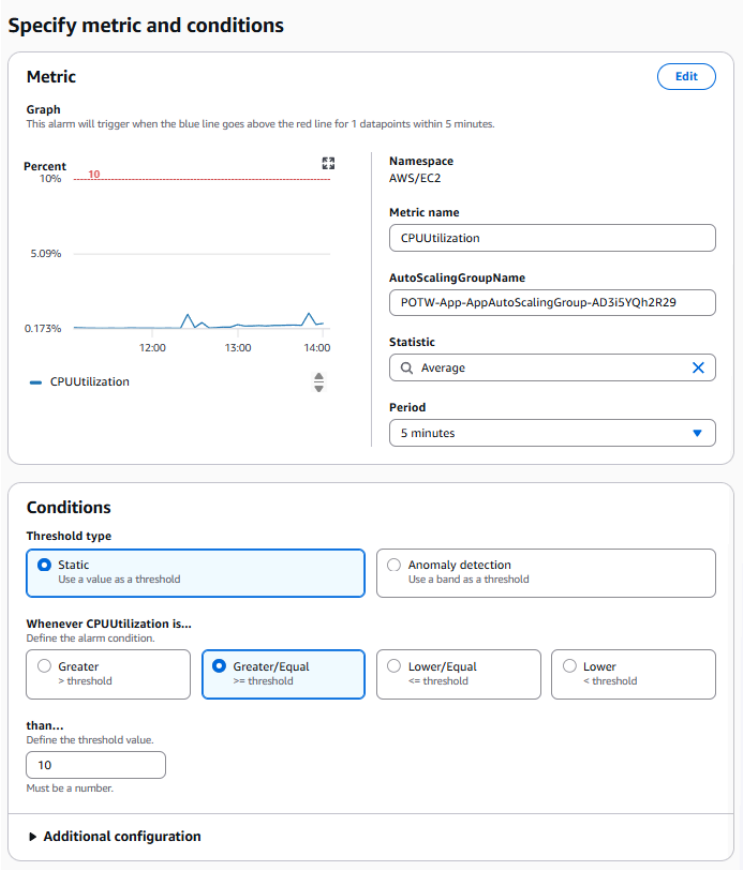


Alarma para EC2: Utilización de CPU

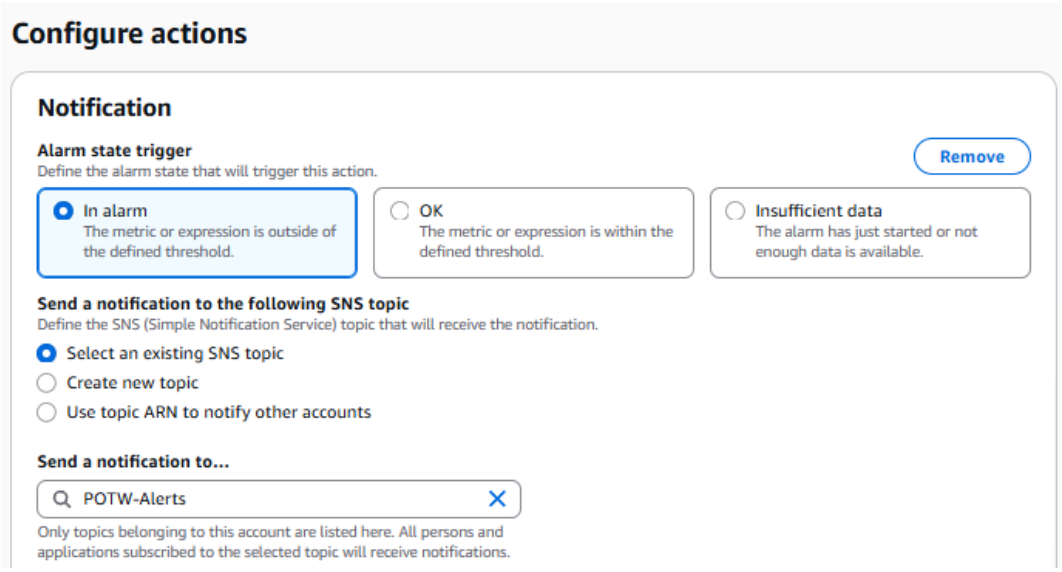
Se hace uso de la métrica CPUUtilization:



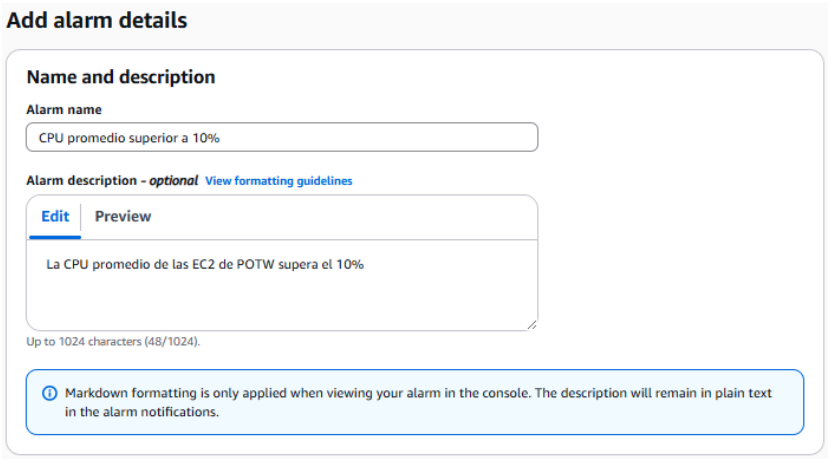
Con la siguiente configuración se consigue que si en algún momento las EC2 detras de Auto Scaling tienen un consumo de CPU medio superior al 10%, se envíe una alarma:



Se selecciona el topic de “Alerts”

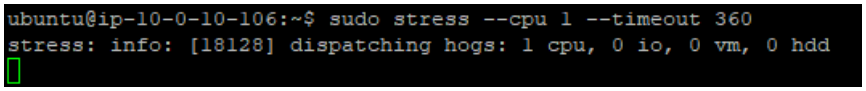


Se añade un nombre y descripción:



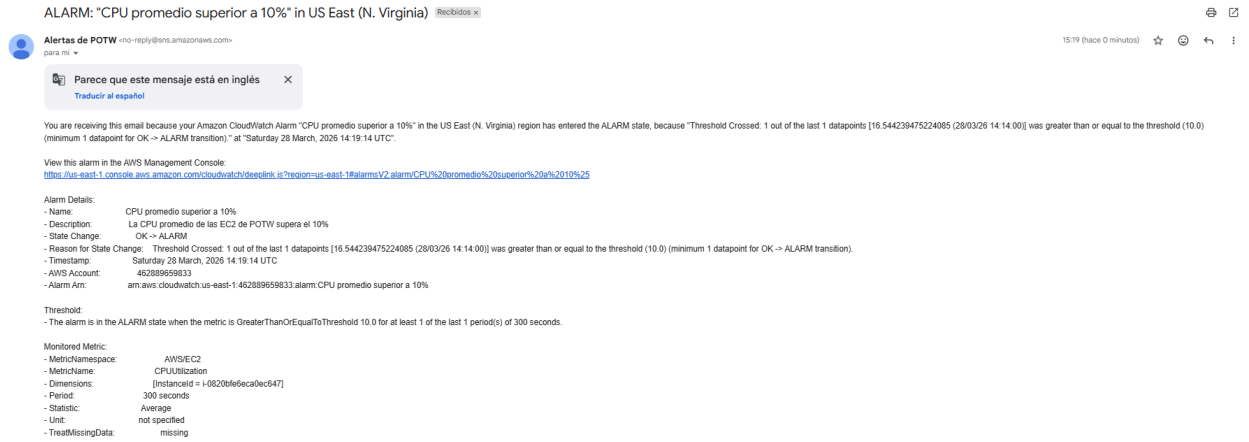
Se observa que la alarma se ha creado correctamente:

Para probar la alarma se usará el comando “stress”



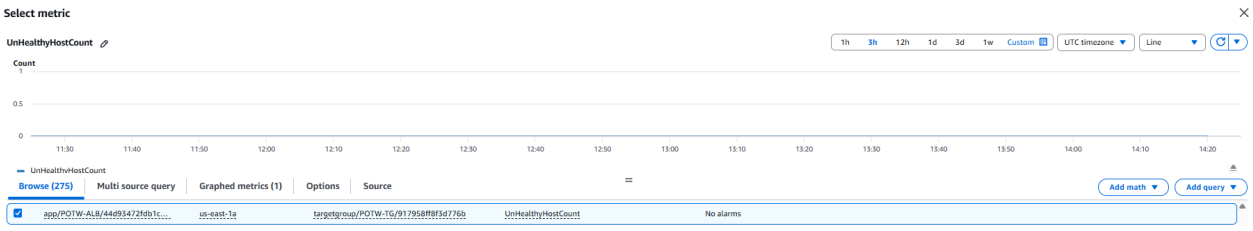
Podemos ver que la CPU ha superado su 10%

A su vez, se comprueba que se ha recibido un mail con la alerta

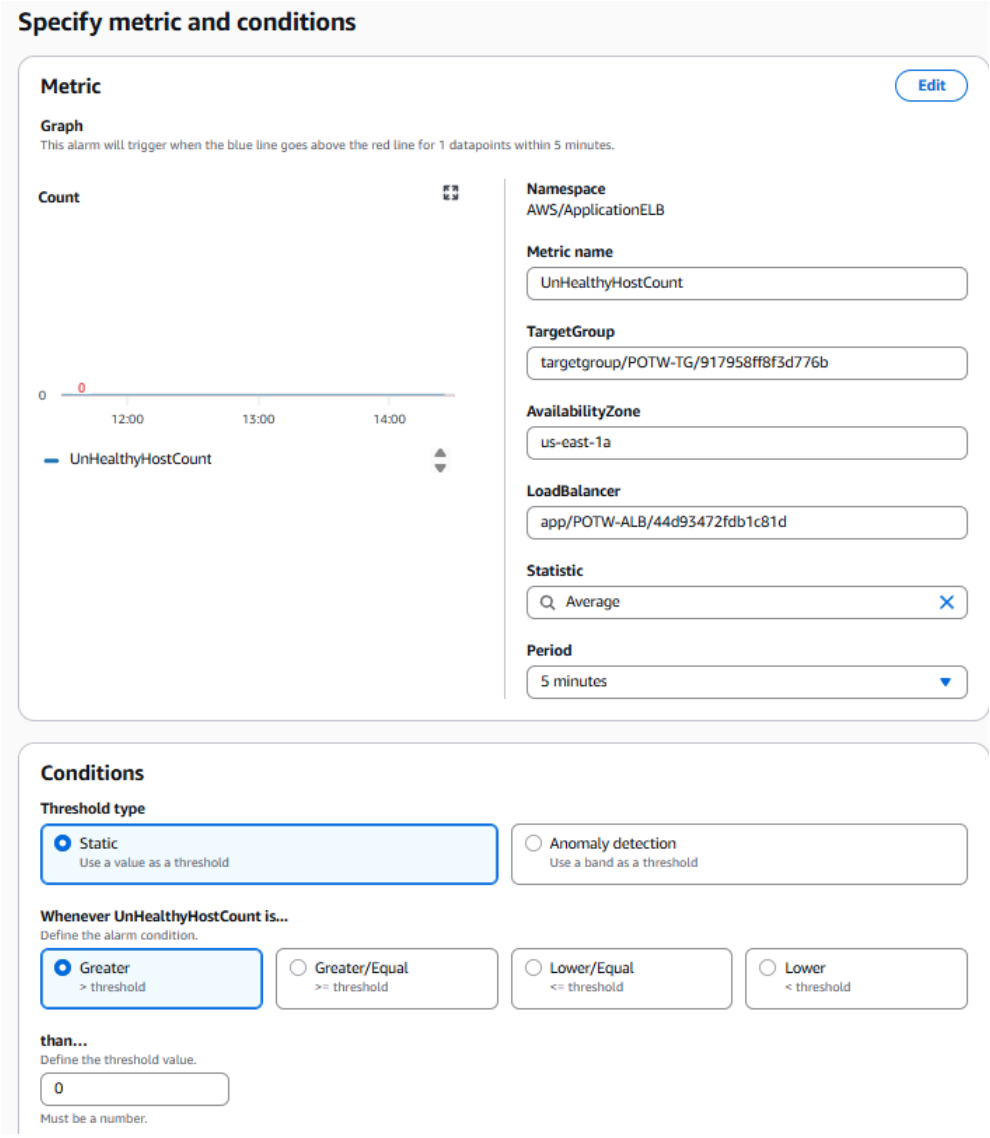


Alarma para ALB: Unhealthy host

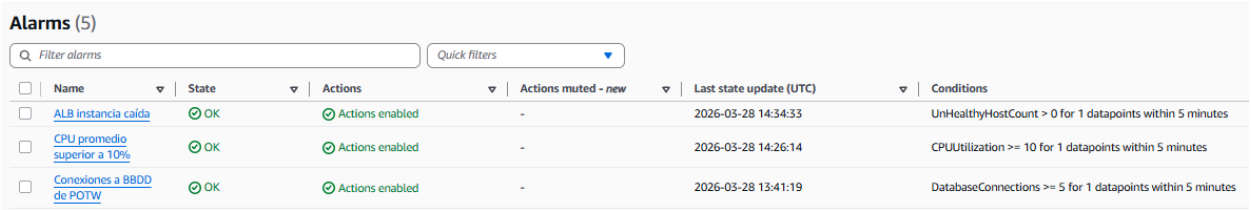
Se hace uso de la métrica de “UnhealthyHostCount”



Se indica que se envíe una alarma cuando el número de instancias en estado “Unhealthy” sea mayor a 0. Tras esto, igual que en los casos anteriores, se elige el topic de SNS adecuado y se guardan cambios.



Se comprueba que la alarma se haya creado correctamente:



Registro y auditoría de accesos (CloudTrail y VPC Flow Logs)

La auditoría de accesos y acciones realizadas sobre la infraestructura es un componente clave de la seguridad y la trazabilidad en entornos en la nube. En este proyecto, se implementa la captura de eventos y registros de red mediante **AWS CloudTrail** y **VPC Flow Logs**.

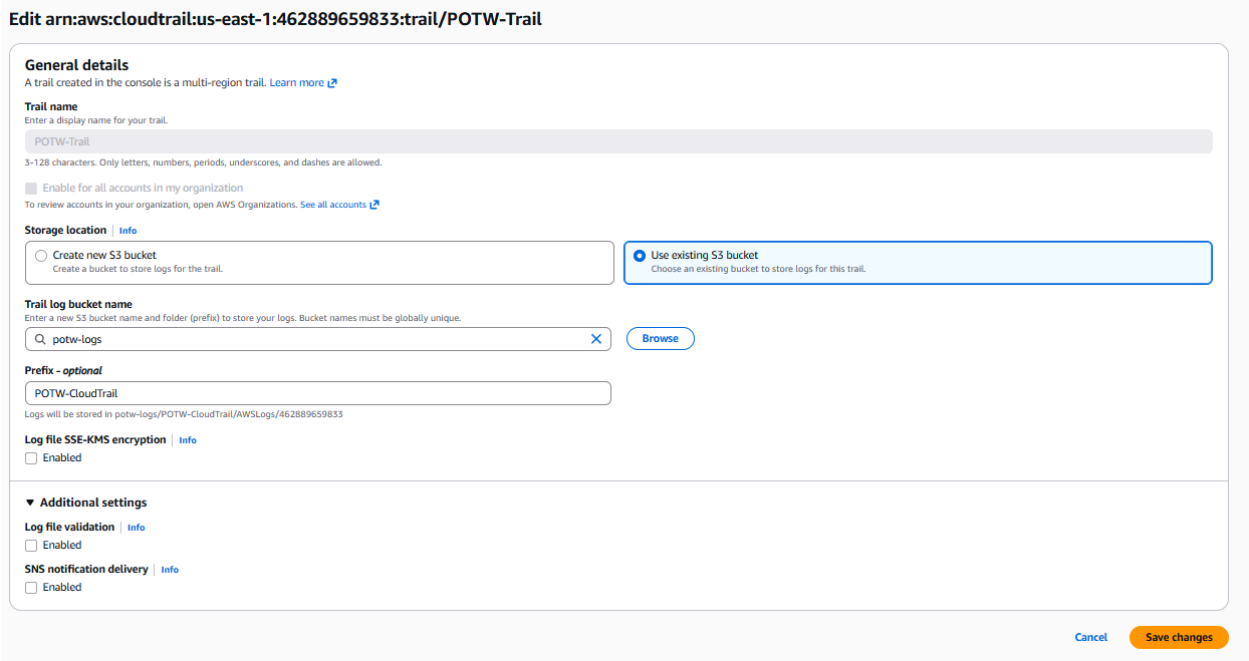
CloudTrail permite **registrar todas las llamadas a la API realizadas en la cuenta**, facilitando la detección de acciones no autorizadas o cambios en la configuración de los recursos, sintetizando cuenta quien hizo que, cuando y donde.

Por su parte, los VPC Flow Logs **capturan información de tráfico de red que atraviesa las interfaces de red de las instancias**, lo que ayuda a identificar patrones sospechosos, problemas de conectividad o intentos de acceso no autorizados.

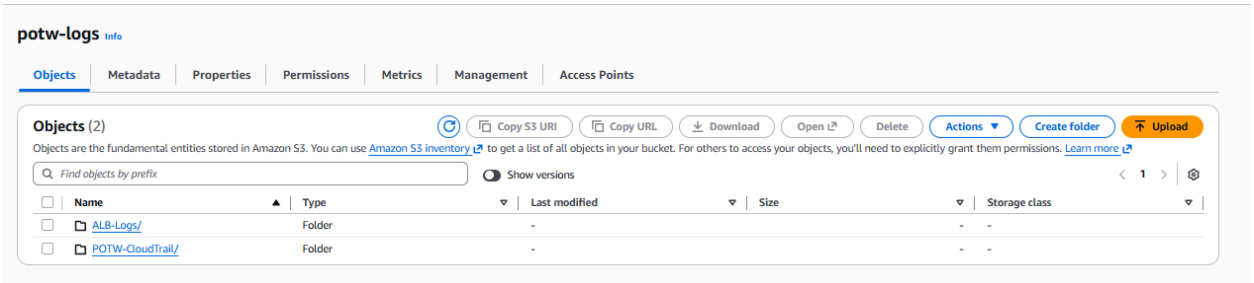
Juntos, estos servicios permiten mantener un historial completo de eventos y accesos en la infraestructura, garantizando la trazabilidad y el cumplimiento de buenas prácticas de seguridad.

Configuración de CloudTrail

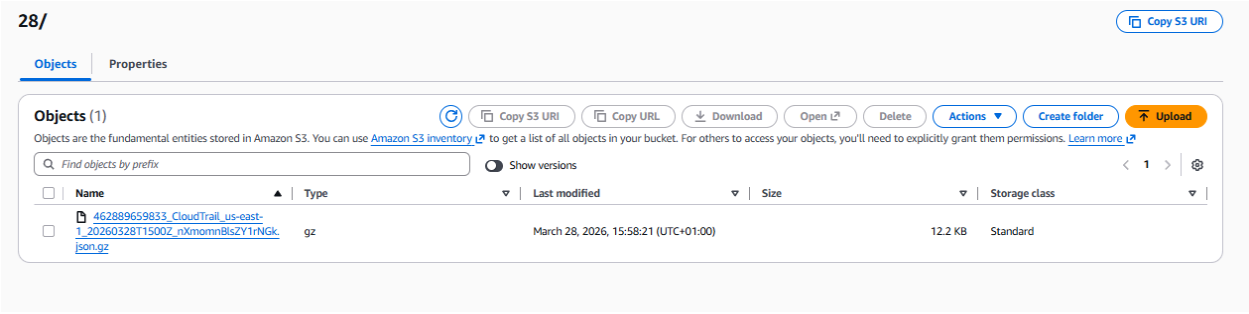
Una vez creado un nuevo trail desde CloudTrail (POTW-trail) se selecciona el bucket donde se desean almacenar los logs. Se hará uso del mismo bucket donde se almacenan los logs de SNS.



Desde el bucket, se observa la creación de una nueva carpeta para CloudTrail:



Para probar su funcionamiento, se va a eliminar una EC2 de la app de POTW, lo cual debería generar un log. Pasados unos segundos, se observa que se ha creado un archivo json en el día 28 de marzo de 2026:



Al descargarlo, se puede observar que desde mi usuario (Carlos Fernandez) se ha eliminado una EC2

*=**Carlos_Fernandez","accountId":"*",**"accessKeyId":"*","sessionContext":{"sessionIssuer":{"type":"Role","principalId":"*","arn":"arn:aws:iam::*:role/voclabs","accountId":"*","userName":"voclabs"},"attributes":{"creationDate":"2026-03-28T12:15:32Z","mfaAuthenticated":"false"}}},**"eventTime":"2026-03-28T14:57:21Z",**"eventSource":"ec2.amazonaws.com",**"eventName":"TerminateInstances",**"awsRegion":"us-east-1"*

Configuración de VPC Flow Logs

Una vez creado un nuevo trail desde CloudTrail (POTW) se selecciona el bucket donde se desean almacenar los logs. Se hará uso del mismo bucket donde se almacenan los logs de SNS.

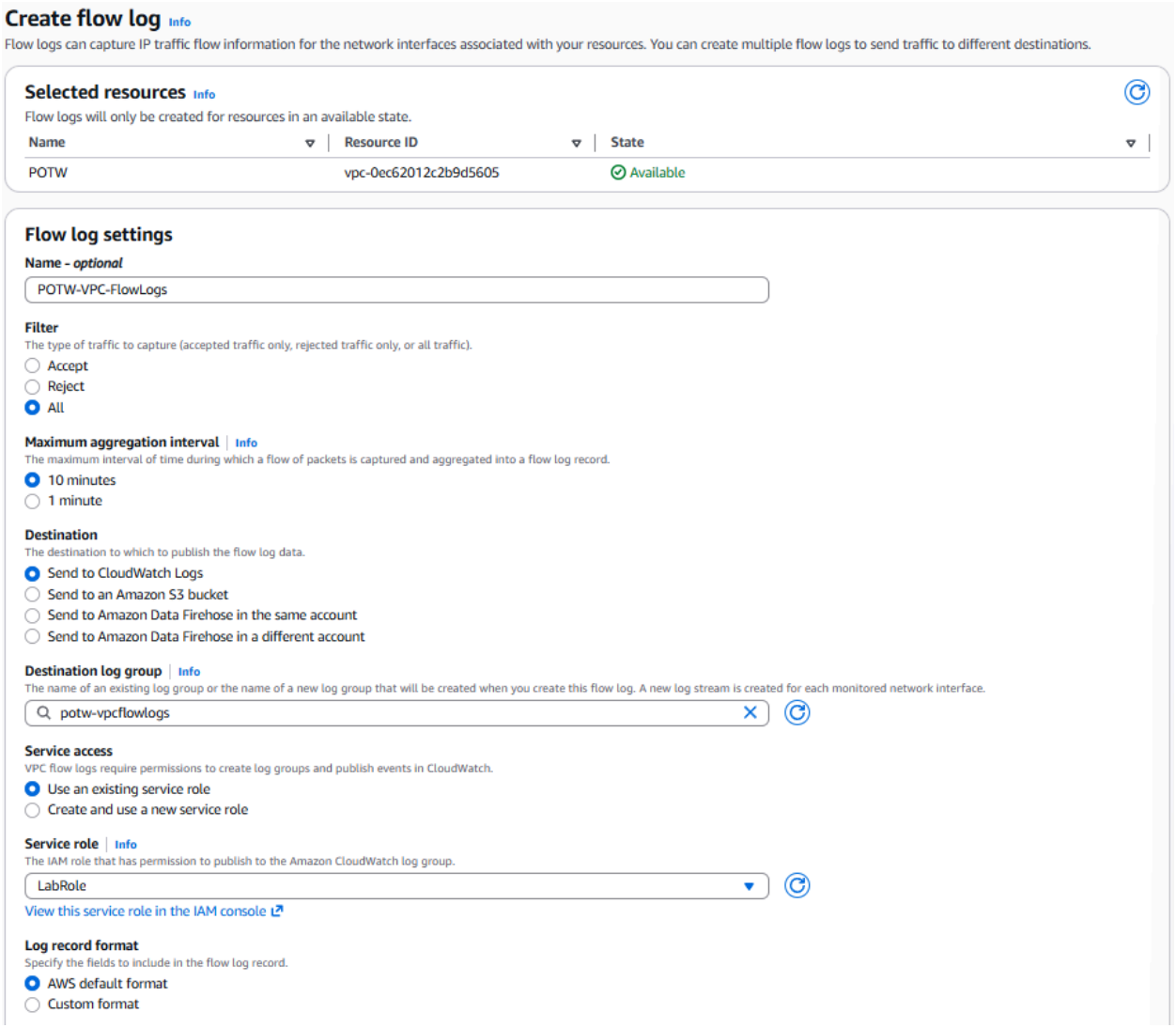
Los **VPC Flow Logs** registran información sobre el tráfico de red que entra y sale de tus interfaces de red (ENI) dentro de tu VPC. Esto incluye tráfico hacia y desde:

- Instancias EC2
- Load Balancers
- Bases de datos RDS
- Otros recursos dentro de la VPC

Qué registran:

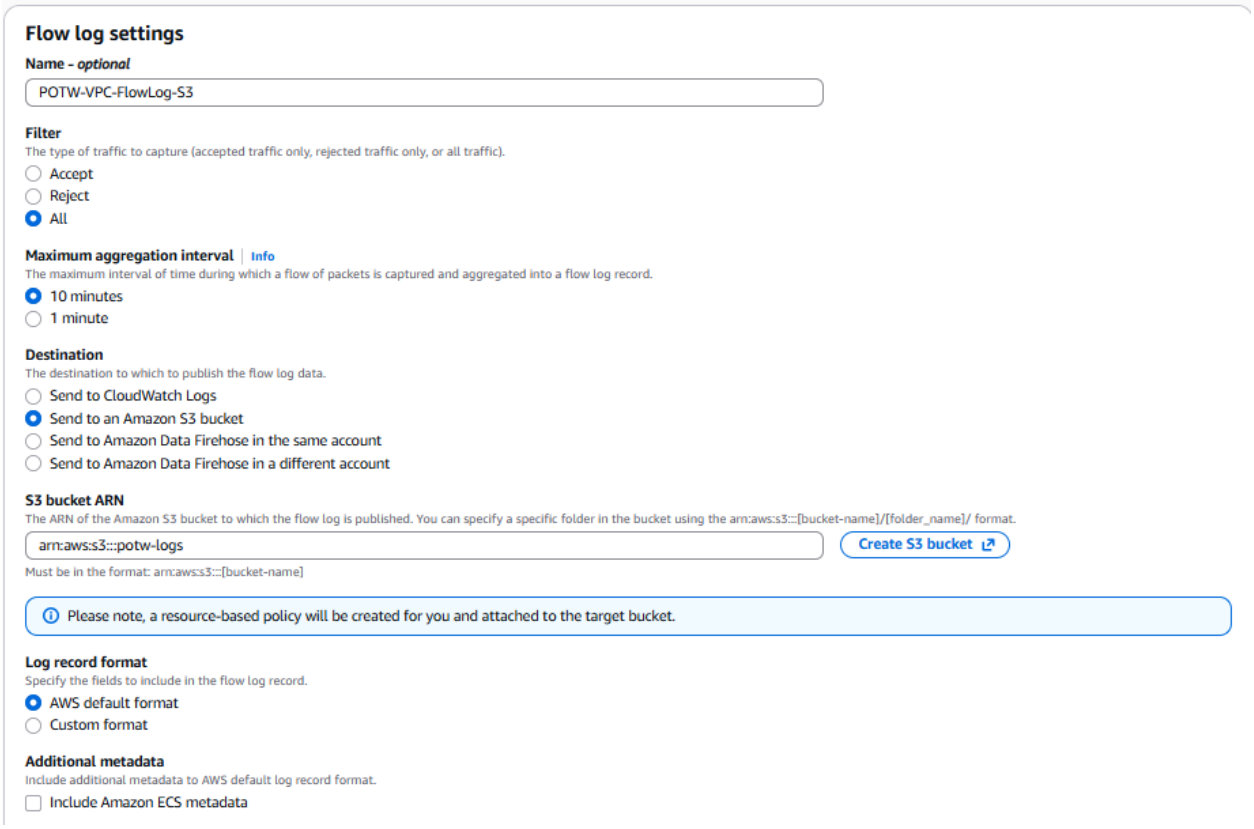
- Dirección IP de origen y destino
- Puerto de origen y destino
- Protocolo usado (TCP, UDP, ICMP…)
- Cantidad de bytes enviados/recibidos
- Si la conexión fue aceptada o rechazada

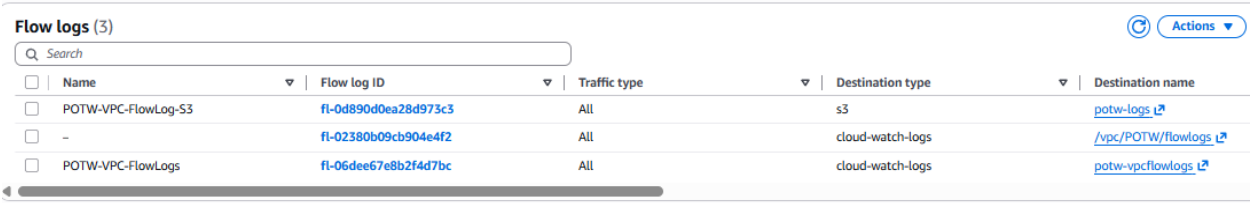
Desde la VPC de POTW, se crea el Flow Log



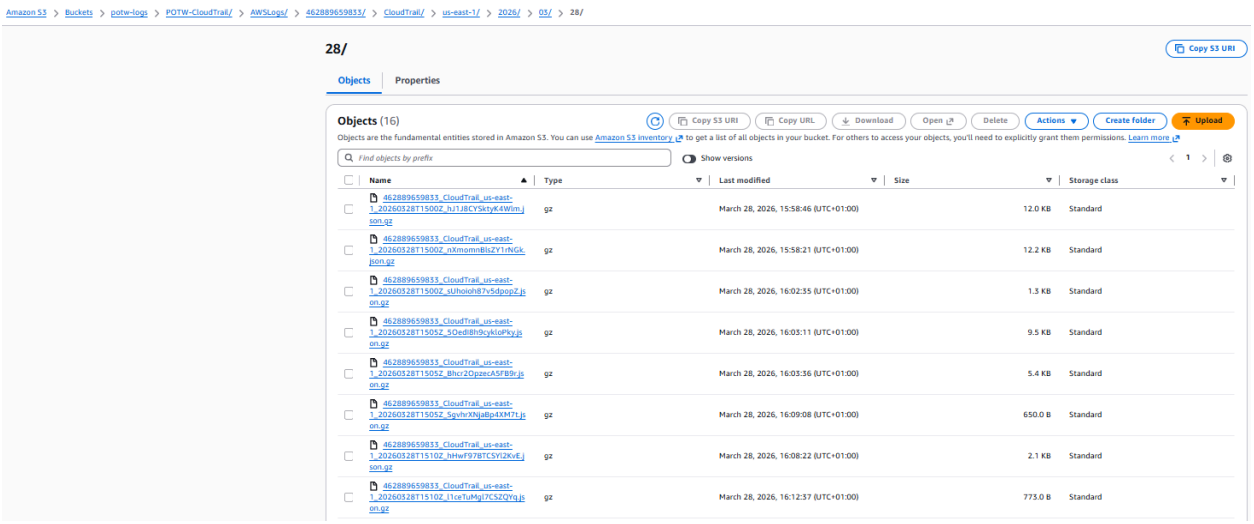
Como se observa, se mandan los logs a CloudWatch, lo cual permite monitorizar y configurar alertas.

Se crea un Flow Log adicional para mandar los logs al bucket donde se almacenan otros logs, de esta manera se mantienen durante el ciclo de vida configurado en el bucket (7 días) para una posible auditoría.





Se puede apreciar la creación de los logs:



Pruebas de resiliencia y continuidad

Validación de Auto Scaling

Failover de RDS

Simulación de caída de AZ

Pruebas de resiliencia y continuidad

En esta sección se realizan **simulaciones de fallos controlados sobre la infraestructura** desplegada para evaluar la capacidad del sistema de mantener la disponibilidad y el servicio ante incidencias.

El objetivo es **demostrar cómo los mecanismos de alta disponibilidad, balanceo de carga y tolerancia** a fallos permiten que la aplicación siga operativa incluso cuando uno o varios componentes presentan problemas, garantizando la continuidad del negocio y la experiencia del usuario.

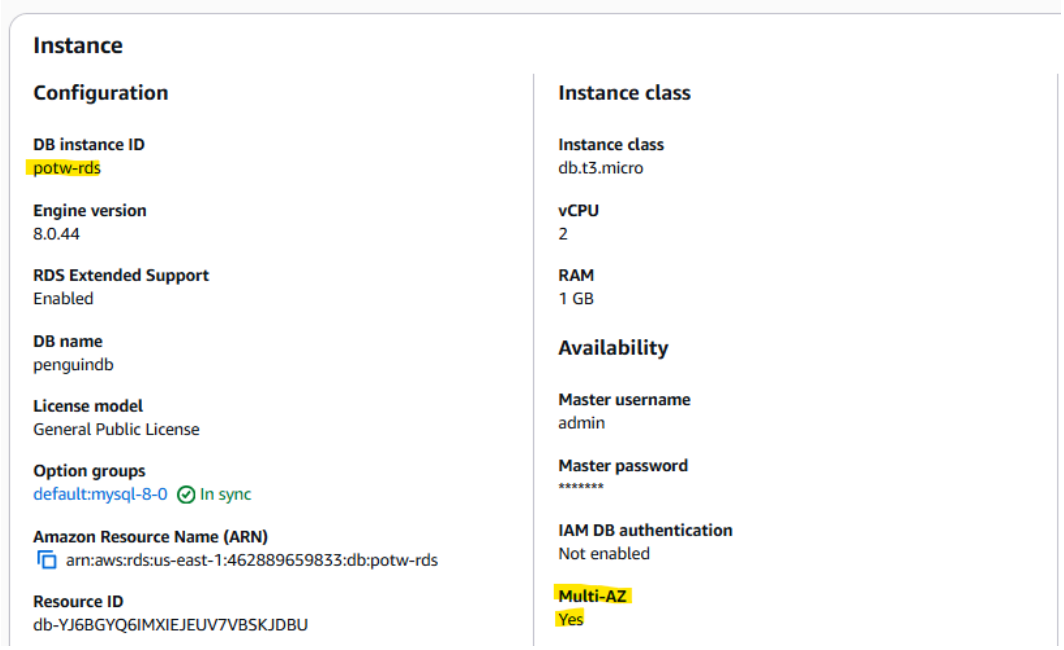
Validación de Auto Scaling

Para demostrar el correcto funcionamiento de Auto Scaling se ha realizado el siguiente video, donde se muestra que, tras terminar una instancia, otra se levanta cubriendo así el mínimo de instancias configuradas, que es de dos.

https://youtu.be/7D2WMmLskiE

Failover de RDS

Se ha configurado un Failover en RDS multi AZ para asegurar que una caída o destrucción del hardware físico donde se hostea la DB no suponga una interrupción en el funcionamiento de POTW



Para una demostración, se hará un reboot a la DB con failover activado en el siguiente video: https://youtu.be/244NvqrCE0c

Simulación de caída de AZ

Este escenario se ha simulado mediante la interrupción controlada de instancias EC2. Esto permite verificar que el Application Load Balancer redirige automáticamente el tráfico a instancias disponibles, garantizando la continuidad del servicio y validando la alta disponibilidad de la arquitectura.

Se hace una demostración en el siguiente video: https://youtu.be/Xpn_k0p4ioY

Backups y recuperación

Estrategia de snapshots automáticos

Procedimientos de recuperación ante fallos

Backups y recuperación

La estrategia de backups y recuperación es un elemento fundamental para garantizar la **continuidad del servicio y la integridad de los datos** ante posibles fallos, errores humanos o incidencias en la infraestructura.

En este proyecto, se han implementado mecanismos de respaldo aprovechando las capacidades nativas de AWS, especialmente en la **capa de datos mediante Amazon RDS**.

Estos backups **permiten restaurar la base de datos a un punto anterior en el tiempo**, minimizando la pérdida de información y reduciendo el impacto de posibles incidencias.

Estrategia de snapshots automáticos

La estrategia de snapshots automáticos se basa en la utilización de las capacidades nativas de respaldo de Amazon RDS, que permiten **generar copias de seguridad de forma periódica sin intervención manual**.

Estos snapshots **incluyen tanto los datos como la configuración de la base de datos**, posibilitando la restauración completa del sistema a un punto concreto en el tiempo.

En este proyecto, se ha configurado una política de retención que mantiene los backups durante un periodo definido, asegurando un equilibrio entre disponibilidad de históricos y optimización de costes.

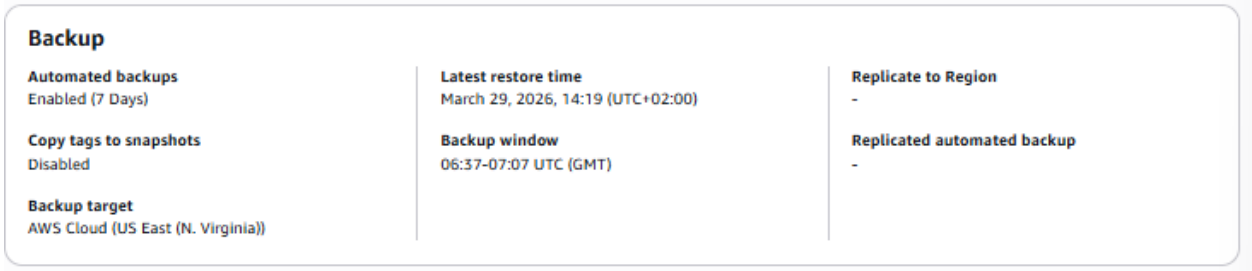
Este enfoque permite proteger la información crítica de la aplicación y garantiza una recuperación rápida y fiable ante posibles fallos o pérdidas de datos.

Backups

En la consola de AWS se puede comprobar que **la instancia de RDS tiene habilitados los backups automáticos**, con un periodo de retención configurado y una ventana de ejecución definida.

Esto garantiza que **las copias de seguridad se realizan de forma periódica sin intervención manual**, asegurando la disponibilidad de históricos recientes para su recuperación.

De este modo, se establece una política de respaldo consistente que protege los datos de la aplicación frente a posibles incidencias o pérdidas de información.

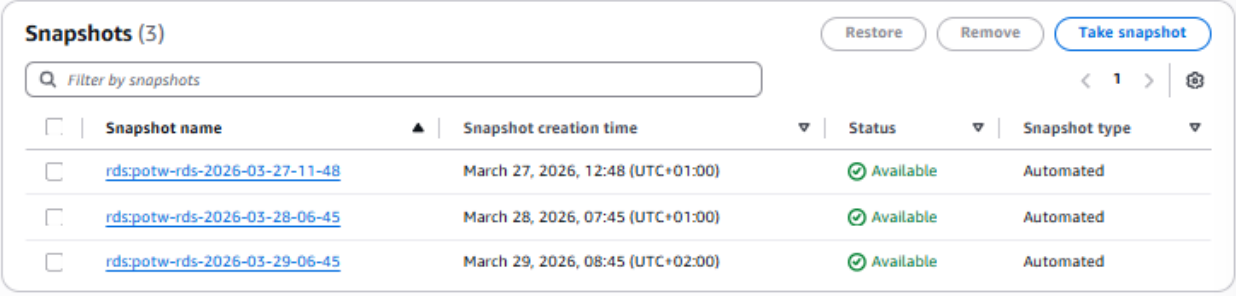


Snapshots

En la sección de snapshots de RDS se pueden visualizar las **copias generadas automáticamente por el sistema**, incluyendo su fecha y hora de creación.

Esto permite **verificar que los backups se están ejecutando correctamente con la frecuencia esperada**, así como disponer de un historial de puntos de restauración recientes.

De este modo, se **garantiza la trazabilidad de las copias de seguridad y la capacidad de recuperación** ante posibles fallos.



Procedimientos de recuperación ante fallos

Con el objetivo de garantizar la continuidad del servicio y la integridad de los datos, se muestra a continuación la **recuperación que permite restaurar la aplicación ante distintos tipos de incidencias**, especialmente aquellas relacionadas con la base de datos.

En el caso de fallo de la instancia de RDS o pérdida de información, **AWS permite realizar una restauración completa a partir de los backups automáticos**. Esta restauración puede hacerse tanto a partir de un **snapshot** concreto como mediante **Point-In-Time Recovery (PITR)**, lo que posibilita recuperar la base de datos a un momento exacto dentro del periodo de retención configurado.

El proceso de recuperación consiste en **lanzar una nueva instancia de RDS restaurada desde el punto deseado**.

Además, al tratarse de una arquitectura desacoplada basada en servicios gestionados (RDS, EC2 en Auto Scaling y ALB), **la recuperación de un componente no afecta al resto de la infraestructura**, facilitando una respuesta rápida y controlada ante fallos.

Se muestra como hacer una recuperación desde snapshot en el siguiente video: https://youtu.be/z5Yyacmn8Fs

Conclusiones y aprendizaje

Resumen de resultados

Competencias demostradas

Posibles mejoras y extensiones futuras

Resumen de resultados

El despliegue de **Penguin On The Way!** ha permitido validar de manera integral la arquitectura propuesta y cumplir con los objetivos del proyecto de práctica en un entorno AWS controlado. Los resultados más relevantes son:

- **Infraestructura completa y funcional:** Se ha desplegado con éxito la VPC, subnets públicas y privadas, Security Groups, VPC Endpoint para SNS, RDS MySQL en configuración Multi-AZ, Application Load Balancer (ALB), Auto Scaling Group y EC2 con la aplicación lista para un entorno de pruebas con características similares a producción. Esto demuestra la correcta implementación de una arquitectura en capas.
- **Seguridad y control de acceso:** Los Security Groups están configurados para permitir únicamente el tráfico necesario entre capas, y el VPC Endpoint asegura que las notificaciones SNS se transmitan de manera privada, evitando la exposición de tráfico sensible a Internet.
- **Disponibilidad y resiliencia:** La base de datos RDS cuenta con despliegue Multi-AZ y backups automáticos diarios con un periodo de retención de 7 días, lo que permite restaurar la instancia a un estado anterior en caso de fallo. Adicionalmente, se dispone de snapshots que facilitan recuperaciones puntuales. Esto asegura que la infraestructura pueda recuperarse de forma rápida y controlada ante incidentes.
- **Escalabilidad y balanceo de carga:** El ALB distribuye el tráfico HTTP entre las instancias EC2 del Auto Scaling Group. La política de escalado basada en el número de solicitudes por instancia garantiza que la capa de aplicación pueda adaptarse dinámicamente a la carga, optimizando recursos y costes.
- **Aplicación web desplegada y operativa:** La app Flask con NGINX y Gunicorn se ejecuta correctamente en EC2, interactúa con la base de datos RDS y envía notificaciones a SNS. El frontend es accesible a través del ALB, y las pruebas funcionales muestran que el flujo de registro y envío de tickets funciona según lo esperado.
- **Optimización de costes para entorno de pruebas:** Se seleccionaron tipos de instancia mínimos tanto para EC2 (t3.micro) como para RDS (db.t3.micro) que permiten realizar la práctica sin incurrir en gastos innecesarios, manteniendo un entorno seguro y operativo.

En conjunto, estos resultados confirman que la arquitectura implementada cumple con los principios de **seguridad, resiliencia y escalabilidad**, permite practicar competencias de DevOps y Cloud Architecture, y sirve como base para futuras mejoras

Competencias demostradas

El desarrollo de **Penguin On The Way!** ha permitido poner en práctica un conjunto amplio de competencias técnicas alineadas con el rol de **Cloud/DevOps Engineer**, abarcando desde el diseño de arquitectura hasta su despliegue y operación en AWS.

- **Diseño de arquitecturas en la nube:** Capacidad para diseñar una infraestructura en capas (red, aplicación y datos), aplicando principios de alta disponibilidad, aislamiento y buenas prácticas de arquitectura en AWS.
- **Infraestructura como Código (IaC):** Uso de AWS CloudFormation para definir, desplegar y gestionar todos los recursos de forma automatizada, incluyendo parametrización, reutilización de plantillas y organización por stacks.
- **Descomposición modular de la infraestructura:** Separación de la arquitectura en múltiples stacks (red, seguridad, datos, aplicación), facilitando la reutilización, mantenibilidad y escalabilidad del proyecto.
- **Administración de sistemas Linux y automatización:** Configuración de instancias EC2 mediante scripts Bash, instalación de dependencias y automatización del despliegue de la aplicación, reduciendo la intervención manual.
- **Despliegue de aplicaciones web:** Implementación completa de una aplicación basada en Flask, incluyendo frontend, backend, conexión a base de datos y uso de servicios AWS como SNS para notificaciones.
- **Networking en AWS:** Configuración avanzada de VPC, subnets públicas y privadas, tablas de rutas, NAT Gateways, Network ACLs y Security Groups, asegurando conectividad controlada entre los distintos componentes.
- **Seguridad en la nube:** Aplicación de principios de mínimo privilegio mediante Security Groups, uso de VPC Endpoint para tráfico privado y aislamiento de la base de datos en subnets privadas.
- **Alta disponibilidad y resiliencia:** Implementación de RDS en Multi-AZ, uso de Auto Scaling Group y Application Load Balancer, así como definición de estrategias de backup y recuperación ante fallos.

- **Escalabilidad automática:** Configuración de políticas de Auto Scaling basadas en métricas reales (peticiones por instancia), permitiendo adaptar dinámicamente la capacidad del sistema.
- **Observabilidad y monitorización:** Integración de CloudWatch para logs y métricas, configuración de alarmas con SNS y uso de CloudTrail y VPC Flow Logs para auditoría y trazabilidad del sistema.
- **Gestión de logs centralizados:** Recopilación y análisis de logs de ALB, EC2 y RDS mediante CloudWatch y S3, facilitando la detección de incidencias y el análisis del comportamiento del sistema.
- **Optimización de costes:** Selección de recursos adecuados para un entorno de pruebas, evitando sobredimensionamiento sin comprometer la funcionalidad del sistema.
- **Enfoque orientado a entornos reales:** Toma de decisiones técnicas justificadas (como el uso de EC2 frente a PaaS) con el objetivo de demostrar competencias prácticas alineadas con roles profesionales en cloud.

En conjunto, este proyecto pretende demostrar la **capacidad de diseñar, desplegar y operar una arquitectura completa en AWS**, integrando múltiples servicios y siguiendo un enfoque alineado con entornos reales de producción.

Posibles mejoras y extensiones futuras

Aunque la arquitectura de Penguin On The Way! cumple con los objetivos planteados para un entorno de pruebas, existen varias líneas de mejora que permitirían evolucionar el sistema hacia un entorno más cercano a producción, aumentando su robustez, seguridad y nivel de automatización.

- **Migración hacia servicios PaaS o serverless:** Evolucionar la capa de aplicación hacia soluciones como Elastic Beanstalk, App Runner o incluso una arquitectura serverless con AWS Lambda y API Gateway, reduciendo la carga operativa sobre las instancias EC2.
- **Mejora de la gestión de secretos:** Sustituir el uso de archivos .env por servicios gestionados como AWS Secrets Manager o Systems Manager Parameter Store, mejorando la seguridad y trazabilidad de las credenciales.
- **Pipeline de CI/CD:** Implementar un flujo de integración y despliegue continuo mediante herramientas como CodePipeline, CodeBuild o GitHub Actions, permitiendo automatizar la construcción, testeo y despliegue de la aplicación.
- **Contenerización de la aplicación:** Adaptar la aplicación para ejecutarse en contenedores Docker y desplegarla en servicios como ECS o EKS, facilitando la portabilidad y escalabilidad del sistema.
- **Monitorización avanzada:** Incorporar dashboards más completos en CloudWatch, así como herramientas adicionales de observabilidad (tracing distribuido, métricas personalizadas o integración con soluciones externas).
- **Mejoras en seguridad:** Implementar HTTPS mediante certificados gestionados (ACM), añadir AWS WAF para protección frente a ataques web y reforzar políticas IAM siguiendo el principio de mínimo privilegio de forma más granular.
- **Optimización de costes en producción:** Evaluar el uso de instancias reservadas o savings plans, así como ajustar dinámicamente los recursos en función del uso real del sistema.
- **Alta disponibilidad avanzada:** Extender la arquitectura a múltiples regiones (multi-region) para tolerancia a fallos a nivel geográfico, mejorando aún más la resiliencia del sistema.

- **Gestión avanzada de logs:** Implementar un sistema de centralización más completo (por ejemplo, integrando herramientas de análisis como OpenSearch) que permita búsquedas avanzadas y análisis en tiempo real.

En conjunto, estas mejoras permitirán evolucionar la solución desde un entorno de aprendizaje hacia una arquitectura más madura, alineada con estándares de producción y preparada para soportar cargas reales a gran escala.

Enlaces de interés

Enlaces de interés

GitHub del proyecto: https://github.com/CarlosFdezSalazar/Penguin-On-the-Way-

Despliegue mediante Cloud Formation: https://youtu.be/x_02zRjsbXw

Demostración del funcionamiento de Auto Scaling: https://youtu.be/7D2WMmLskiE

Demostración de Failover en RDS: https://youtu.be/244NvqrCE0c

Demostración de escritura en RDS: https://youtu.be/pYG5wb1C6_I

Recovery de RDS desde snapshot: https://youtu.be/z5Yyacmn8Fs

Demostración de funcionamiento tras caída de AZ simulada: https://youtu.be/Xpn_k0p4ioY