

# **PATITAS.PET: PLATAFORMA SERVERLESS EN AWS PARA GESTIÓN DE ADOPCIONES**

**Reto Final AWS**

María Perea

<b>Proyecto Patitas.Pet</b>	<b>2</b>
<b>Diagrama de arquitectura</b>	<b>2</b>
Relación entre servicios	3
<b>Justificación de diseño</b>	<b>4</b>
Objetivos	4
Estrategia tecnológica	4
Servicios elegidos y razones	4
Ventajas clave de esta estrategia	4
<b>Pasos de despliegue y configuración</b>	<b>5</b>
FRONTEND – Subir tu web como una PWA con S3	5
BACKEND – Funciones que procesan datos (con Lambda)	6
DYNAMODB	7
API GATEWAY – Conectar la web con Lambda	10
WAF – Aplicar reglas de seguridad a la API Gateway	10
S3 para Imágenes – Subida segura con URL prefirmadas	11
COGNITO – Registro y login de protectoras	12
INTEGRAR COGNITO + DYNAMODB	16
CLOUDWATCH – Logs y alertas	17
SES	17
<b>Consideraciones de buenas prácticas</b>	<b>19</b>
Seguridad	19
Gestión de credenciales	20
Escalabilidad	20
Costos	20
Extra: otras buenas prácticas	21

# Proyecto Patitas.Pet

Patitas.Pet es una aplicación web diseñada para que protectoras, casas de acogida y perreras puedan gestionar y mostrar los animales que tienen disponibles para adopción. La app permite subir contenido como fotos y datos de los animales, organizarlos por filtros como especie, edad o tamaño, y mostrar una galería pública para que posibles adoptantes puedan conocerlos fácilmente. Además, incluye un sistema de contacto directo en la ficha de cada animal para facilitar la comunicación entre adoptantes y protectoras.

El objetivo principal es dar visibilidad a los animales que buscan hogar, simplificando la gestión para las protectoras y facilitando la adopción responsable. Se busca implementar una plataforma segura, escalable y de uso sencillo que permita:

- Gestión de animales en adopción por parte de protectoras (alta, baja, modificación).
- Subida y almacenamiento seguro de fotos de los animales.
- Visualización pública y amigable para usuarios interesados en adoptar.
- Contacto directo y rápido con las protectoras desde la ficha de cada animal.
- Autenticación y control de acceso para diferentes protectoras o usuarios administradores.
- Monitorización y escalabilidad con costes mínimos en AWS.

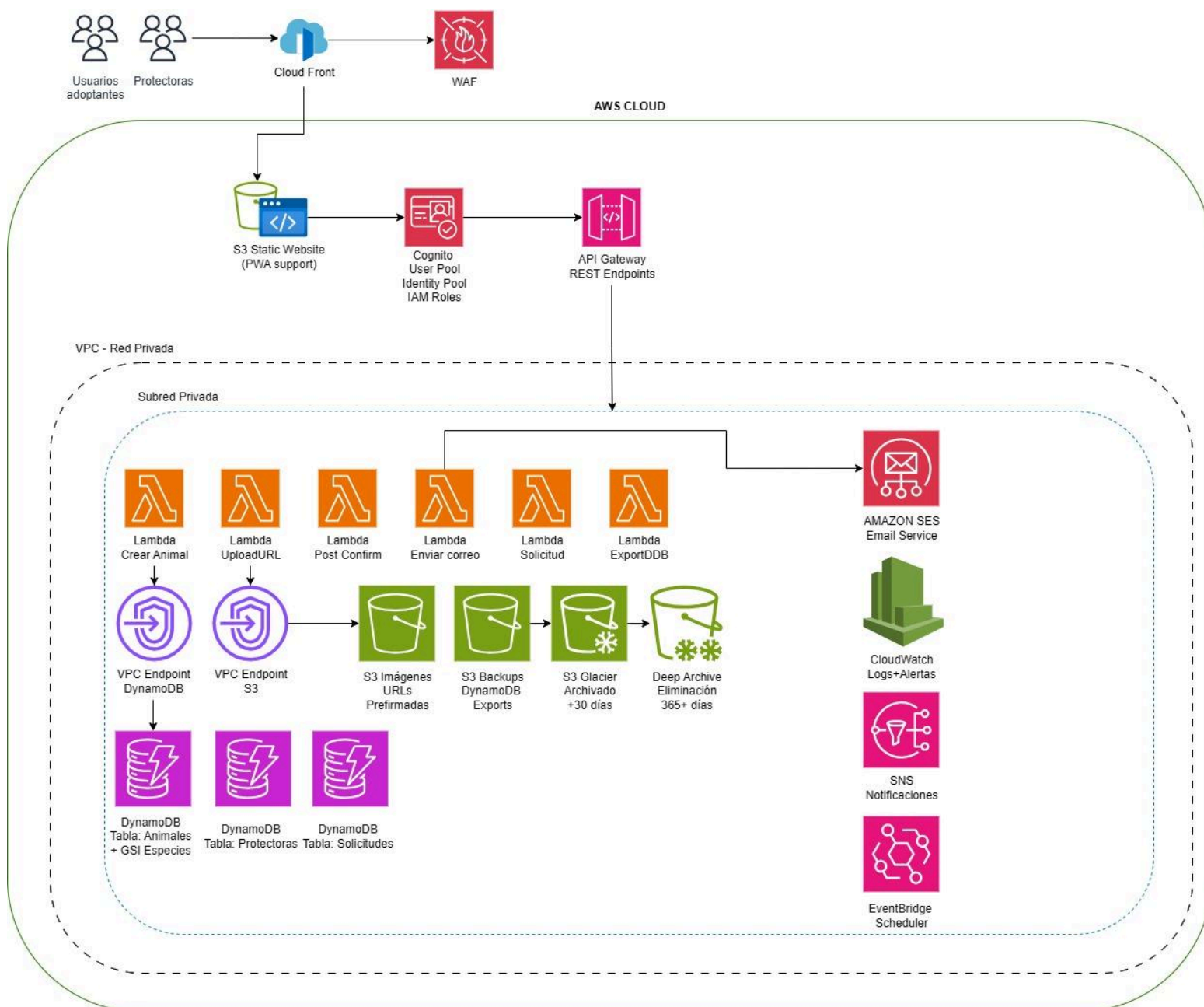
## Diagrama de arquitectura

La arquitectura de Patitas.Pet sigue un enfoque serverless dentro de una VPC segura. Incluye los siguientes componentes:

- **Amazon S3 (Frontend):** hosting de la aplicación web como sitio estático y soporte PWA.
- **Amazon API Gateway:** API REST que conecta el frontend con las funciones Lambda.
- **AWS Lambda:** funciones backend para CRUD de animales, generación de URLs prefirmadas y envío de emails.
- **Amazon DynamoDB:** base de datos NoSQL para almacenar animales, protectoras y usuarios.
- **Amazon Cognito** (User Pools e Identity Pools): autenticación, autorización y emisión de credenciales temporales.
- **Amazon S3 (Backups):** bucket dedicado a las exportaciones de DynamoDB con políticas de ciclo de vida.
- **Amazon SES:** envío de correos desde la ficha del animal.
- **Amazon CloudWatch:** métricas, logs y alarmas para Lambda y exportaciones.
- **AWS WAF:** protección de la API contra ataques comunes.
- **VPC + Subredes privadas + Endpoints:** aislamiento de las funciones Lambda y acceso seguro a DynamoDB y S3 sin tráfico público.

## Relación entre servicios

1. El usuario accede al frontend en S3.
2. Cognito gestiona autenticación y permisos.
3. El frontend consume la API de API Gateway.
4. Lambda ejecuta la lógica de negocio y accede a DynamoDB y S3.
5. Las imágenes se suben a S3 mediante URLs prefirmadas.
6. Backups de DynamoDB se exportan automáticamente a un bucket S3 seguro.
7. CloudWatch y WAF aseguran monitoreo y seguridad perimetral.



# Justificación de diseño

## Objetivos

- Facilitar la gestión de animales para protectoras pequeñas.
- Minimizar la administración de infraestructura.
- Garantizar una plataforma segura y escalable con costes bajos.
- Automatizar backups y recuperación de datos.

## Estrategia tecnológica

- **100% serverless:** Lambda, API Gateway, Cognito, DynamoDB y S3.
- **Seguridad en red:** Lambda en subred privada con VPC Endpoints.
- **Backups automáticos:** exportación programada de DynamoDB a S3.
- **Gestión de identidades robusta:** uso de Cognito con roles IAM vinculados a grupos.

## Servicios elegidos y razones

<b>S3 (Frontend e Imágenes)</b>	Bajo coste, escalabilidad ilimitada y soporte PWA.
<b>Lambda</b>	No requiere servidores, integración directa con DynamoDB y S3.
<b>DynamoDB</b>	Base NoSQL rápida y gestionada.
<b>Cognito</b>	Autenticación centralizada y federada con roles IAM.
<b>API Gateway</b>	Control de acceso, soporte CORS y fácil integración con Lambda
<b>CloudWatch</b>	Monitoreo, métricas y alarmas.
<b>WAF</b>	Protección frente a ataques comunes (SQLi, XSS, etc.).
<b>SES</b>	Envío de correos fiable y económico.
<b>VPC y Endpoints</b>	Aislamiento de red y eliminación de tráfico público innecesario.

## Ventajas clave de esta estrategia

- **Escalabilidad automática:** sin límites fijos ni configuraciones complejas
- **Alta disponibilidad:** tolerancia a fallos incluida en los servicios gestionados

- **Coste optimizado:** aprovechamiento del free-tier de AWS
- **Seguridad reforzada:** uso de VPC, IAM por función, URLs pre-firmadas
- **Multiusuario y modular:** cada protectora accede a su panel y catálogo

En resumen esta arquitectura está pensada para **maximizar impacto y simplicidad**. Combina lo mejor de AWS para un entorno moderno, socialmente útil, con bajo coste, alta seguridad y escalabilidad lista para crecer según las necesidades del proyecto.

## Pasos de despliegue y configuración

### FRONTEND – Subir tu web como una PWA con S3




1. **Ir a la consola de AWS** → <https://aws.amazon.com/console>
2. **Buscar “S3”** en el buscador superior
3. **Crear un bucket nuevo:**
  - Nombre: algo como **patitas-frontend**
  - Región: elige una cercana a ti (ej. EU West)
  - Propiedad de objetos: Desactivar listas de control de acceso (ACL) → Activar “Bucket owner enforced”
  - Desactiva “Bloquear acceso público”
  - Control de versiones (Opcional): Puedes dejarlo desactivado si no necesitas versiones de archivos
  - Etiquetas (Opcional): Puedes añadir etiquetas como “Proyecto”: “Patitas”
  - Cifrado predeterminado (Opcional): Puedes activar SSE-S3 para cifrado básico.

 [Resultado creación Bucket](#)


4. **Subir tu web:**
  - Una vez creado el bucket, entra en la configuración.
  - Ve a la pestaña “Propiedades”.
  - Activa “Hostear sitio web estático”.
    - Documento de inicio: **index.html**
    - Documento de error (opcional): **404.html**

 [404.html](#)

 [Activación Host Estático](#)

- Subir archivos
- Haz clic en “Subir” y selecciona:
  - index.html:  [index.html](#)
  - Archivos CSS, JS, etc.  [script.js](#)  [style.css](#)

 [Imagen carga de archivos](#)

- En la pestaña “Permisos” agrega una política JSON que permita la lectura pública:  [Política json solo lectura.json](#)

## [Editar Política Bucket](#)

- En la pestaña Permisos, asegúrate de que los archivos sean públicos.

A parte de la página web principal (index.html) se han añadido dos paginas conectadas a ella que permiten el inicio de sesión de los usuarios registrados ([login.html](#)) y el registro de los usuarios ([registro.html](#)). También se ha añadido una pagina donde ver el perfil de cada animal disponible y utilizando el formulario se envía un mensaje a la protectora de solicitud de adopción ([Adoptme.html](#)).

Como ejemplos también se ha creado la página que vería la protectora al iniciar sesión con su cuenta ([Protectoras.html](#)) y la página de perfil de la protectora donde puede modificar su información tanto en la web como en Cognito ([PerfilProtectoras.html](#)).

Más adelante, en la sección de Cognito, veremos qué debe incluir el archivo [registro.js](#) (referenciado en registro.html mediante una etiqueta <script src="">) para que la información de los usuarios se almacene correctamente y pueda utilizarse como método de autenticación.

### 5. Ver tu web:

- Accede a <http://patitas-frontend.s3-website-us-east-1.amazonaws.com>

### 6 . PWA (Progressive Web App, opcional):

- Añade un archivo manifest.json en la raíz del proyecto (define nombre, icono, colores, etc.).
- Crea un service-worker.js que almacene archivos en caché para funcionar sin conexión.
- Enlaza ambos archivos en el index.html: [PWA Enlaza archivos en el index.html](#)
- Asegúrate de subir también los iconos (ej: icon-192.png, icon-512.png) al bucket.
- Para que funcione el modo offline, necesitas servir la web por HTTPS (usa CloudFront si lo necesitas).

## BACKEND – Funciones que procesan datos (con Lambda)

### 1. Ve a **AWS Lambda**

### 2. Haz clic en "Crear función"

- Tipo: "Autor desde cero"
- Nombre: **crearAnimal**
- Runtime: elige Node.js o Python
- Rol: crea uno nuevo con permisos básicos

## [Función creada](#)

### 3. Dentro de la función:

- Escribe tu código o pega uno que ya tengas, por ejemplo: [🔗 Funcion Crear Animal.py](#)
- Pulsa el botón deploy para guardar los cambios.

## DYNAMODB

### Guardar datos de animales y protectoras

1. Ir a **DynamoDB**
2. “Crear nueva tabla”
  - Nombre de la tabla: **Animales**
  - Clave de partición (Partition key): **protectora\_id** (tipo: String)
  - Clave de ordenación (Sort key): **animal\_id** (tipo: String)
  - Esto permite identificar cada animal dentro de una protectora.
- Clase de tabla:
  - Clase estándar: selecciona **STANDARD** (por defecto).
  - Si tu acceso a la tabla será poco frecuente, puedes elegir **“STANDARD\_INFREQUENT\_ACCESS”** para ahorrar costos.
- Configuración de capacidad de lectura/escritura:
  - Modo de capacidad: selecciona **Sobredemanda** (On-demand)
  - Si eliges Provisionado, deberás definir:
    - Capacidad de lectura: ej. 5
    - Capacidad de escritura: ej. 5
- Rendimiento en caliente: Actívalo solo si tienes una clave que recibirá muchas lecturas/escrituras simultáneas.

### [Crear tabla](#)

- Índices secundarios (opcional): Si quieres hacer búsquedas por otros atributos (ej. especie, estado), puedes crear UN “Índice secundario global (GSI)”:
  - Nombre del índice: **porEspecie**
  - Clave de partición: **especie** (tipo String)
  - Clave de ordenación: (opcional, ej. estado)
  - Proyección: **All**
  - Modo de capacidad: **Sobredemanda** (On-demand)
  - Después de crear el índice: puedes hacer búsquedas usando Query sobre el GSI en tu código o en la consola.

### [Índice secundario](#)

- Cifrado en reposo: puedes dejar la opción por defecto: “Clave gestionada por AWS”. Si necesitas más control, selecciona una clave KMS personalizada.
  - Protección contra eliminaciones: actívalo si quieres evitar que la tabla se borre accidentalmente.
  - Política basada en recursos (opcional): puedes definir políticas para controlar quién puede acceder a la tabla (restringir el acceso por roles o servicios).
3. Las funciones Lambda usarán esta tabla para consultar o modificar datos.
  4. Ahora puedes hacer un test de la funcion “crearAnimal” que creamos anteriormente:



- Ve a Lambda, selecciona la función “crearAnimal”
- Pulsa “test” Si es la primera vez, te pedirá crear un evento de prueba.
- Crea un evento de prueba:
- Dale un nombre (ej. eventoCrearAnimal)
- En el cuerpo JSON, pon los datos que tu función espera:
- Haz clic en “Guardar” .
- Haz clic en “Probar”: Lambda ejecutará tu función con ese evento y verás el resultado en la parte inferior: salida, logs, duración, uso de memoria, etc.

 [Test Función](#)

 [Resultado Test](#)

Además de la tabla Animales, es necesario definir también una tabla “Protectoras” para almacenar los datos de cada entidad que participa en la plataforma. Esta tabla debería incluir como mínimo:

- protectora\_id (clave principal)
- nombre
- correo electrónico
- teléfono
- ubicación
- otros datos relevantes para su identificación.

Si se espera que cada protectora tenga varios usuarios (por ejemplo, empleados con distintos permisos o accesos), también se debería crear una tabla “Usuarios” que relacione a cada usuario con su protectora mediante un atributo como protectora\_id. Esto permitiría implementar una estructura multiusuario real y gestionar accesos de forma más granular.

Este diseño permite mantener una arquitectura organizada y escalable, facilitando tanto la administración de datos como el control de permisos en futuras fases del proyecto.

### Backups y recuperación en DynamoDB

Para asegurar la protección de datos, es recomendable implementar una política de backup y recuperación:

1. **Habilitar Point-in-Time Recovery (PITR):** En la consola DynamoDB, activar PITR en cada tabla para restaurar a cualquier punto de los últimos 35 días.

 [Configuración de recuperación a un momento dado](#)

2. **Exportaciones periódicas a S3:** Programar exportaciones automáticas de la tabla a un bucket S3 para conservar backups históricos y configurar ciclo de vida para archivar datos antiguos en Glacier y optimizar costes:
  - Crear bucket S3 para backups:
    - Configura un bucket dedicado para almacenar las exportaciones de DynamoDB.
    - Configura políticas de acceso restringido.

### [Politica PatitasDynamoDBBackupAccess.json](#)

- Configurar exportación manual o automática: AWS no ofrece exportación programada nativa para DynamoDB, pero se puede automatizar con AWS Lambda + EventBridge (CloudWatch Events):
  - Crea una función Lambda que use el API `ExportTableToPointInTime` para exportar la tabla a S3.

### [Exportar tabla a Amazon S3](#)

### [ExportTableToPointInTime](#)

- Programa EventBridge para ejecutar esa función Lambda según frecuencia deseada (diaria, semanal).

### [ExportDynamoDBRule](#)

- Configurar ciclo de vida del bucket S3
  - En la consola S3, define reglas de ciclo de vida para mover objetos antiguos a Glacier o Deep Archive (mover archivos a Glacier después de 30 días, borrarlos después de 365 días)

### [Crear la regla del ciclo de vida](#)

- Permisos necesarios: rol Lambda debe tener permisos:
    - `dynamodb:ExportTableToPointInTime`
    - `s3:PutObject` y gestión de objetos en el bucket S3.
  - Verificación y monitoreo: monitorear las exportaciones con CloudWatch y configurar alertas para fallos en exportación.
3. **Configurar permisos IAM:** Otorgar permisos a roles para crear backups y exportar tablas (`dynamodb:CreateBackup`, `dynamodb:ExportTableToPointInTime`) y permitir escritura en bucket S3 (`s3:PutObject`).
  4. **Pruebas de recuperación:** Realizar restauraciones periódicas a tablas temporales para validar integridad y funcionamiento.
  5. **Monitorización y documentación:** Documentar la política y configurar alertas en CloudWatch para detectar fallos en backups.

### Flujo desde la web hasta DynamoDB

- **Desde el panel web:** cada protectora accede a un formulario para añadir o modificar los datos de sus animales (nombre, especie, edad, estado, foto, etc.).
- **Envío a la función Lambda (`crearAnimal`)** el formulario envía los datos mediante una llamada API a la función Lambda `crearAnimal`, la cual valida y prepara la información.
- **Almacenamiento en DynamoDB:** la Lambda guarda los datos en la tabla Animales, usando las claves `protectora_id` y `animal_id`. Si ya existe el animal, se puede usar otra función (ej. `modificarAnimal`) para actualizar los datos.

- **Consulta desde la web:** las protectoras ven su lista de animales con una consulta a DynamoDB filtrando por su `protectora_id`. Si hay un GSI, también pueden buscar por especie o estado.

## API GATEWAY – Conectar la web con Lambda

1. Ve a **Amazon API Gateway**
2. Crear nueva API REST:
  - Haz clic en “Crear API” → selecciona “API REST” (no HTTP API).
  - Elige “Crear desde cero”.
  - Ponle un nombre, por ejemplo: `PatitasAPI`.
  - Deja el resto de opciones por defecto y haz clic en “Crear API”.

 [Creación API Rest](#)

3. Agrega recursos:
  - `/animales` → métodos: GET (listar), POST (crear)
  - `/animales/{id}` → métodos: PUT (editar), DELETE (borrar)
  - `/upload-url` → método: GET

 [Agregador de recursos 1](#)

 [Agregado de Recursos 2](#)

- Crear los métodos:
  - Elige “Lambda Function” como integración
  - Conecta a tu función correspondiente.

 [Creación de Métodos](#)



4. Habilita CORS:
  - Esto permite que tu frontend en otro dominio pueda comunicarse con tu API
5. Despliega la API:
  - Crea un “stage” llamado `prod`
  - Copia el endpoint final para usar en tu frontend

 [Despliegue](#)

6. Usa el endpoint en tu frontend:
  - Para crear un animal: POST `https://.../prod/animales`
  - Para listar: GET `https://.../prod/animales`
  - Para editar: PUT `https://.../prod/animales/{id}`
  - Para borrar: DELETE `https://.../prod/animales/{id}`
  - Para obtener URL de subida: GET `https://.../prod/upload-url`


## WAF – Aplicar reglas de seguridad a la API Gateway


1. Ve a WAF & Shield en la consola.
2. Recursos y protecciones > Agregar paquete de protección:


- Categoría: Sistemas de contenido y publicación
  - Fuente de tráfico: API y web
  - Seleccione los recursos que deben protegerse:
    - PatitasAPI
  - Elegir protecciones: Recomendado  
(Una línea base sólida de protección, con integración directa con tu API Gateway y posibilidad de crecer: luego puedes añadir detección de bots, límites de peticiones por IP, etc.)
  - Nombre del Web ACL: **PatitasPet-API-WAF**
  - Descripción (opcional): **Proteger la API de Patitas.Pet contra amenazas comunes.**
  -  [Agregar paquete de protección](#)
3. Finaliza la creación.
-  [Resultado WAF](#)


## S3 para Imágenes – Subida segura con URL prefirmadas

1. Crear bucket privado:
  - Nombre del bucket: **patitas-imagenes**
  - Región: La misma que tu Lambda
  - Bloqueo de acceso público: **Activado** (no permitir acceso público)
  - Propiedad de objetos: Activar **“Bucket owner enforced”**
  - Cifrado: Opcional (puedes usar SSE-S3)
  - Configura la policy “s3:PutObject” para el bucket

 [Política de bucket s3PutObject.py](#)
2. En tu función Lambda (**generarUploadURL**):
  - Usar **getSignedUrl()** para generar una URL segura
  - En el frontend, usar esa URL para subir la imagen con **fetch()** o **axios**

 [generarUploadURL](#)
3. Crea tu endpoint en API Gateway:
  - Método: GET /upload-url
  - Query param: nombreArchivo, contentType (opcional)
  - Integración: Lambda generarUploadURL
  - Autenticación (opcional): puedes proteger con Cognito si quieres restringir uso.

 [GET /upload-url](#)
4. Integra en el frontend: con el código de la web del formulario para añadir animales. Funciona de la siguiente manera:
  - Este fragmento de código es el encargado de subir la imagen directamente al bucket S3 desde el navegador del usuario, sin pasar por el backend.

 [subirImagenAS3\(event\)](#)

Usa una URL prefirmada generada por una función Lambda, que autoriza temporalmente el acceso para hacer una operación PUT en S3. El navegador aprovecha esa URL para mandar el archivo seleccionado, con su tipo MIME (Content-Type) y el contenido real (body), logrando así una subida segura y eficiente.

Este enfoque permite mantener el bucket privado y evita exponer credenciales, delegando la seguridad a AWS.

5. Guarda referencia del nombre en tu base de datos: Después de subir la imagen, guarda el nombre (foto123.jpg) en DynamoDB u otro lugar, para luego usarlo en las fichas del animal.
6. Mostrar la imagen:
  - Opción A: Habilita acceso público de lectura (más sencillo, menos seguro)
  - Opción B: Crea otra Lambda que genere una URL firmada de tipo GET y la uses para mostrar la imagen

#### ¿Cómo saber si todo funciona?

- Prueba subir una imagen de 1MB
- Asegúrate de ver el archivo en el bucket
- Comprueba que la URL prefirmada expira a los 15 min

La aplicación involucra a varios tipos de usuarios: las protectoras, los adoptantes y el equipo encargado de administrar la plataforma. El uso de URLs prefirmadas aporta beneficios clave:

- **Menor coste en transferencias:** los archivos se suben directamente a S3, sin necesidad de pasar por Lambda ni API Gateway.
- **Mayor velocidad de carga:** al eliminar intermediarios, las subidas son más rápidas y eficientes.
- **Seguridad mejorada:** las imágenes y el bucket de S3 están protegidos mediante permisos temporales y controlados.

## COGNITO – Registro y login de protectoras

### PASO 1: Crear un User Pool

1. Entra a la consola de AWS → Ve a Amazon Cognito.
2. En el menú lateral, selecciona User Pools → Clic en “Create user pool”.
  - Tipo de aplicación: selecciona “Aplicación web tradicional”
  - Nombre de la aplicación: **PatitasPet**
  - Opciones para los identificadores de inicio de sesión:
    - Correo electrónico
    - Nombre de usuario
  - Atributos necesarios para el inicio de sesión:
    - email
    - name

- En Agregar una URL de retorno, escribe la URL de tu frontend
- 3. Haz clic en “Crear grupo de usuarios”.

#### [Crear un User Pool](#)

- 4. Al finalizar, se creará el User Pool y verás una pantalla con:
  - User Pool ID
  - ARN del User Pool
  - URL de claves públicas

#### [Resultado User Pool](#)

- 5. En el menú izquierdo, ve a “Grupos”
  - Crea dos grupos:
    - admin
    - protectora

#### [Crear los grupos en el User Pool](#)

- 6. Crear roles de IAM para cada grupo
  - Ve a IAM > Roles > Crear Rol
  - En Seleccionar entidad de confianza, elige: "Identidad web"
  - En Proveedor de identidad web, selecciona: Cognito
    - Escribe o selecciona tu User Pool
    - Elige: Cognito User Pool (no Identity Pool)
    - En Identificador de audiencia, selecciona tu App client ID
  - Pulsa Siguiente
  - Para el grupo admin, puedes agregar por ejemplo:
    - AmazonDynamoDBFullAccess
    - AmazonAPIGatewayInvokeFullAccess
    - Luego haz clic en Siguiente.
  - Nombre del rol: **Patitas\_Admin\_Role**
  - Finaliza la creación.
- 7. Asignar los roles a los grupos en Cognito
  - Ve de nuevo a Cognito > Tu User Pool > Grupos.
  - En cada grupo, edita y selecciona el rol IAM correspondiente:
  - admin: **Patitas\_Admin\_Role**
  - protectora: **Patitas\_Protectora\_Role**

### PASO 2: Crear el Identity Pool

- 1. Entra a la consola de AWS → Ve a Amazon Cognito.
- 2. En el menú lateral, selecciona User Pools → Clic en “Crear grupo de identidades”
  - Acceso de usuarios:
    - Acceso autenticado: Activa esta opción (requerido).
    - Orígenes de identidad autenticados: elige "Grupo de usuarios de Amazon Cognito"
  - Configurar permisos:

- En “Rol autenticado” click en “Crear un nuevo rol de IAM”:  
[PatitasAuthRole](#)

(Cognito creará automáticamente una política mínima con confianza hacia Cognito)

- Conectar proveedores de identidades:
  - Verifica que el grupo de usuarios de Cognito ya esté seleccionado.
  - En “Selección de rol”, selecciona “Usar un rol autenticado predeterminado”.
  - Asignación de reclamaciones: Inactivo
- Configurar propiedades:
  - Nombre del grupo de identidades: [PatitasIdentityPool](#)
  - Autenticación básica (clásica): “Activar el flujo básico”, por compatibilidad.
  - Etiquetas (opcionales):
    - Clave: [Proyecto](#)
    - Valor: [Patitas](#)
- Revisar y crear:
  - Verifica que:
    - El User Pool y App Client están bien puestos
    - El rol IAM se llama [PatitasAuthRole](#)
  - Clic en “Crear grupo de identidades”

 [Crear el Identity Pool](#)

### PASO 3: Configurar el rol IAM (PatitasAuthRole)

1. Abre la consola de IAM
2. Busca el rol con nombre: [PatitasAuthRole](#)
3. Haz clic en él para editarlo:
  - Haz clic en la pestaña “Confianza” y asegúrate de que permite asumir el rol desde Cognito.


 [Política de confianza Rol IAM.json](#)

4. Haz clic en la pestaña “Permisos”, “Adjuntar políticas”:
  - Permitir acceso al bucket S3: [patitas-imagenes](#)
    - Clic en “Agregar permisos”
    - Adjuntar políticas
    - Crea una nueva.

 [Política acceso al bucket S3 rol IAM.json](#)

- Guarda y adjunta esta política al rol [PatitasAuthRole](#)

### ¿Cómo se guardan las protectoras registradas?

1. Código del formulario de registro  [registro.js](#)

- Inicialización del User Pool: en el código JavaScript se define la conexión con Cognito, es decir, conecta el frontend con el User Pool de Cognito para poder registrar usuarios.

 [Inicialización del User Pool](#)

- Creación de atributos del usuario: prepara los datos (nombre y correo) como atributos de usuario que se almacenarán en Cognito junto con la contraseña.

 [Creación de atributos del usuario](#)


- Registro del usuario en Cognito: llama al método signUp del SDK de Cognito.

 [Registro del usuario en Cognito](#)

- Asignación automática al grupo (mediante Lambda): después de que el usuario confirma su cuenta, la Lambda lo añade automáticamente a un grupo dentro de Cognito.


 [Asignación automática al grupo \(mediante Lambda\)](#)

## 2. Crear función Lambda **PostConfirmation**

- Ir a AWS Lambda y Crear función.
- Nombre: **PostConfirmation**.
- Runtime: Node.js 18.x.
- Pegar el siguiente código:  [Funcion PostConfirmation.py](#)
- Crear la función

 [PostConfirmation](#)

## 3. Dar permisos a la Lambda

- En la función Lambda, ir a Configuration > Permisos
- Hacer clic en el rol de ejecución
- En IAM, crear una nueva política con este contenido JSON:  [Permiso Lambda a Cognito.json](#)
- Adjuntar esa política al rol de ejecución de la Lambda.

## 4. Crear el grupo de usuarios en Cognito:

- Ir a Cognito > User Pools
- Hacer clic en Crear grupo.
  - Nombre: **PatitasPetUsers**
  - Guardar

 [Grupo de usuarios PatitasPetUsers](#)

## 5. Configurar el trigger de PostConfirmation

- Ir a Cognito > User Pools > PatitasPetUsers > Extensiones > Desencadenadores de Lambda
- Agregar desencadenador de Lambda:
  - En "Registro", seleccionar Desencadenador de Después de la confirmación



- Elegir la función Lambda PostConfirmation.
- Guardar

### [Desencadenadores de Lambda](#)

#### 6. Probar el registro

- Abrir el formulario de registro de la web
- Crear un nuevo usuario
- Confirmar el email del usuario
- Verificar en Cognito > User Pools > Usuarios que el nuevo usuario aparece en el grupo.

Con esto, cualquier usuario registrado se agregará automáticamente al grupo configurado.

## INTEGRAR COGNITO + DYNAMODB

Ya tenemos la Lambda que asigna al grupo PatitasPetUsers. Ahora vamos a modificarla para que además guarde datos de la protectora en DynamoDB.

#### 1. Modificar código Lambda:

- Ve a AWS Lambda
- Busca tu función PostConfirmation y haz clic en ella.
- Sustituye el código por el siguiente: [PostConfirmationUpdate.py](#)
- Haz clic en el botón "Deploy" o "Guardar" en la parte superior derecha.

#### 2. Dar permisos a la Lambda para escribir en DynamoDB:

- Ve a AWS Lambda y haz clic en tu función PostConfirmation
- Dentro de tu función Lambda, ve a Configuración > Permisos
- Verás un bloque llamado Rol de ejecución con un enlace azul (nombre del rol)
- Haz clic en ese nombre → te lleva a IAM
- Crea una política personalizada:
  - En el panel izquierdo, haz clic en Políticas
  - Luego haz clic en Crear política
  - Ve a la pestaña JSON y pega esta política: [Politica permisos lambda para usar dynamo.json](#)
  - Guarda la política "PermisoPut\_Protectoras"
- Asigna la política al rol:
  - Vuelve al rol IAM
  - Haz clic en Attach policies (Adjuntar políticas)
  - Busca la que acabas de crear: PermisoPut\_Protectoras
  - Selecciónala y haz clic en Adjuntar

#### 3. Asegúrate de tener estos atributos en Cognito:

- Ve a Cognito > User Pools > tu pool > Atributos personalizados
- Confirma que están creados:
  - custom:nombre
  - custom:telefono
  - custom:ubicacion

#### 4. El formulario de registro debe capturar esos atributos

5. Probar el flujo completo:
  - Registra un nuevo usuario de tipo protectora
  - Verifica que se:
    - Añade al grupo PatitasPetUsers
    - Se inserta un registro en la tabla Protectoras con sus atributos personalizados

## CLOUDWATCH – Logs y alertas

Cada Lambda genera logs automáticamente

1. En CloudWatch, ve al menú "Alarms" y haz clic en "Create alarm"
2. Haz clic en "Seleccionar una métrica" y elige:
  - Buscar>Lambda>Function Name
  - Selecciona la función (ej: **crearAnimal**) con la métrica Errors
  - Haz clic en "Select metric"
3. Configura la condición:
  - Tipo de límite: Estático
  - "Cuando Errors sea...": Mayor que 1
4. Configura las acciones:
  - Notificación:
    - Activador de estado de alarma: En modo alarma
    - Enviar una notificación al siguiente tema de SNS: Crear un nuevo tema
    - Opción: "Crear un nuevo tema":
      - Nombre: **PatitasLambdaErrors**
      - Email: tu correo (ej. [maria@email.com](mailto:maria@email.com))
      - Te llegará un email para confirmar la suscripción. Tienes que hacer clic en el enlace para activar las alertas.
5. Agregar detalles de alarma:
  - Nombre de la alarma: **LambdaErroresPatitasAPI**
  - Descripción de la alarma (opcional)

 [Email suscripción](#)

 [Crear Alerta](#)

Puedes configurar este proceso de monitoreo y alertas en CloudWatch para cualquier función Lambda que necesites. También puedes automatizar la creación de alarmas en AWS CloudWatch para varias funciones Lambda usando PowerShell, para ello necesitas tener instalados los módulos de AWS Tools for PowerShell, configurar tus credenciales de AWS y confirmar el email de suscripción a SNS tras ejecutar el script.

 [Script de creación de alarmas en AWS CloudWatch para varias funciones Lambda.ps1](#)

## SES

## Enviar correos desde la ficha del animal

### **PASO 1:** Validar dirección de email en Amazon SES

1. Ve a la consola de Amazon SES.
2. En la barra lateral, haz clic en "Identidades".
3. Haz clic en "Crear identidad":
  - Tipo: Email address
  - Dirección: info@patitas.pet (o cualquier dirección que controles)
4. Amazon te enviará un email con un enlace de verificación.
5. Haz clic en ese enlace para verificar la dirección.

### **PASO 2:** Crear función Lambda para enviar correos con SES

1. Ve a AWS Lambda → "Crear una función".
2. Elige:
  - Nombre: **EnviarCorreo**
  - Tiempo de ejecución: Python 3.12
  - Crear un rol nuevo con permisos básicos de Lambda.
3. Una vez creada, introduce el código: [🔗 Funcion Enviar Correo.py](#)

### **PASO 3:** Dar permisos SES a la Lambda

1. Ve a IAM > Roles.
2. Busca el rol que usa tu función Lambda (EnviarCorreo).
3. En ese rol, haz clic en "Agregar permisos" > "Crear política insertada".
4. Selecciona JSON en Editor de Políticas
5. Añade la política: [🔗 Política permisos SES a la Lambda.json](#)
6. Guarda la política con nombre: **EnviarEmailSES**
7. Vuelve al rol de la Lambda y adjunta esa política.

## Flujo extendido: envío de solicitudes con registro en base de datos



Además del envío de correos mediante SES, se ha implementado un flujo completo que permite registrar cada solicitud de adopción en una base de datos, asegurando trazabilidad y seguimiento por parte de la protectora.

1. Crear la función Lambda EnviarSolicitudAdopcion
  - Ir a AWS Lambda → "Crear función".
  - Nombre: EnviarSolicitudAdopcion.
  - Runtime: Node.js 18.x.
  - Crear un nuevo rol IAM básico (lo ajustaremos más adelante).
  - Pegar el código que:
  - Extrae datos del event.body.
  - Consulta Cognito para obtener el email.
  - Envía el correo con SES.
  - Guarda los datos en DynamoDB.

 [EnviarSolicitudAdopcion.py](#)

2. Dar permisos al rol de ejecución
  - Ir a IAM > Roles > buscar el rol de tu función EnviarSolicitudAdopcion y adjuntar estas políticas:
  - cognito-idp:AdminGetUser → para consultar el email de la protectora.
  - ses:SendEmail → para enviar el correo con los datos del adoptante.
  - dynamodb:PutItem → para guardar la solicitud en la tabla SolicitudesAdopcion.
3. Crear la tabla DynamoDB **SolicitudesAdopcion**
  - Ir a DynamoDB → “Crear tabla”
  - Nombre: SolicitudesAdopcion.
  - Clave de partición: adopcion (String).
  - Clave de ordenación (opcional): fechaEnvio (String).
  - Modo de capacidad: Sobredemanda (on-demand).
  - Activar PITR (Point-In-Time Recovery) para restauración.

 [Tabla solicitudes adopción](#)

4. Validar la dirección de envío en SES
    - Ir a Amazon SES → “Identidades”.
    - Crear identidad → tipo: dirección de correo (noreply@patitas.pet).
    - Confirmar el email desde el enlace enviado por AWS.
  5. Exponer la función vía API Gateway
    - Ir a API Gateway → Crear nueva API REST.
    - Recurso: /solicitud-adopcion.
    - Método: POST.
    - Integración: función Lambda EnviarSolicitudAdopcion.
    - Habilitar CORS para aceptar peticiones desde el frontend.
    - Crear Stage prod y copiar el endpoint para usar en el formulario web.
-  [API Rest Solicitud de Adopción](#)
6. Probar el flujo completo
    - Usa Postman o el formulario web para enviar un JSON como:  
 [Función flujo completo envío de solicitudes con registro en base de datos.json](#)
    - Verifica:
      - Que el correo llega a la protectora.
      - Que el registro aparece en la tabla DynamoDB.
      - Que CloudWatch muestra logs sin errores.

## Consideraciones de buenas prácticas

### Seguridad

- **Principio de privilegio mínimo (Least Privilege):** Cada función Lambda debe tener un rol IAM con solo los permisos estrictamente necesarios (por ejemplo, solo acceso a `PutItem` en una tabla específica).
- **Control de acceso a datos:** Las imágenes se suben mediante URLs prefirmadas, evitando el acceso directo y público al bucket S3 de fotos.
- **Autenticación y autorización:** Se usa Amazon Cognito para gestionar usuarios (protectora/admin) y controlar el acceso a funciones sensibles del backend.
- **Separación de entornos:** Crear diferentes “stages” o entornos para desarrollo, pruebas y producción usando API Gateway y separando recursos.
- **Aislamiento de red:** Las funciones Lambda se ejecutan dentro de una VPC con subred privada y acceso restringido a recursos internos mediante VPC Endpoints.
- **WAF para mitigar amenazas comunes en la API:** estas políticas configuran reglas en AWS WAF para proteger la API frente a ataques comunes como inyección SQL, cross-site scripting y tráfico malicioso, mejorando la seguridad y disponibilidad del servicio.

## Gestión de credenciales

- **Uso de AWS IAM Roles:** Lambda, API Gateway, Cognito y otros servicios acceden entre sí usando roles que definen sus permisos explícitamente.
- **Almacenamiento seguro de configuraciones:** Variables de entorno en Lambda (nunca directamente en el código). Para valores sensibles, usar AWS Secrets Manager (opcional).

## Escalabilidad

- **Arquitectura serverless:**
  - Lambda escala automáticamente según la carga.
  - DynamoDB maneja millones de consultas sin necesidad de ajustar servidores.
  - S3 puede almacenar ilimitadas imágenes sin impacto en rendimiento.
- **Separación de responsabilidades:** Frontend independiente del backend que facilita actualizaciones sin interrupciones.
- **Próxima evolución fácil:** Se puede añadir CloudFront, RDS, Elasticsearch u otros servicios según crezcan las necesidades.

## Costos

- **Uso eficiente del Free Tier:** S3, Lambda, DynamoDB, Cognito y SES tienen niveles gratuitos que cubren ampliamente las necesidades de una protectora pequeña.
- **Evitar desperdicios:** CloudWatch configurado para almacenar solo lo necesario y uso de funciones optimizadas para evitar ejecuciones innecesarias.
- **Monitoreo de uso:** Ver métricas de consumo en Cost Explorer para detectar gastos inesperados.
- **Políticas de ciclo de vida en S3 para backups antiguos:** automatizan la migración de backups antiguos desde S3 estándar a Glacier, optimizando costes de

almacenamiento. Permiten mantener los datos seguros y disponibles a largo plazo con mínimo mantenimiento manual. .

### Extra: otras buenas prácticas

- **Registro de actividad (auditoría):** CloudWatch guarda logs de cada invocación para seguimiento y análisis.
- **Manejo de errores:** Todas las funciones Lambda deben tener bloques `try/catch` y retornar mensajes claros de error.
- **Multilenguaje y accesibilidad:** Frontend preparado para varios idiomas con librerías como `i18next` y diseño amigable tanto en escritorio como en móvil (PWA).
- **Backups automáticos exportados a S3** y validados periódicamente.