Servicios AWS Utilizados en el Proyecto

Índice

1. Resumen General
2. Servicios Utilizados
3. Mapa de Arquitectura AWS
4. Flujo de Conexiones
5. Tabla Resumen

Resumen General

Este proyecto utiliza **7 servicios principales de AWS** para desplegar una aplicación web full-stack:

- Frontend en **AWS Amplify**
- Backend en **EC2** con **Elastic IP**
- Base de datos en **RDS SQL Server**
- Red privada con **VPC**
- Seguridad con **Security Groups**
- Autenticación opcional con **Cognito**
- Distribución de contenido con **CloudFront** (incluido en Amplify)

Servicios Utilizados

1. AWS Amplify

¿Qué es?
Servicio de hosting y CI/CD (Integración Continua / Deploy Continuo) para aplicaciones frontend.

¿Por qué lo usamos?

- **Deploy automático**: Cada push a GitHub activa una nueva construcción y despliegue
- **HTTPS gratuito**: Certificado SSL válido sin configuración manual
- **CDN incluido**: CloudFront distribuye los archivos desde servidores cercanos al usuario
- **Compilación en la nube**: No necesitamos servidor local para compilar Angular
- **Rollback fácil**: Si algo falla, podemos volver a versiones anteriores

¿Qué hace en el proyecto?

1. Detecta cambios en GitHub (rama `main`)
2. Clona el repositorio
3. Lee `amplify.yml` para saber cómo construir
4. Ejecuta `npm ci` y `npm run build`
5. Toma los archivos de `dist/frontend/browser`
6. Los almacena en S3 y los distribuye con CloudFront
7. Genera URL: `https://main.d2dxwi2afljxg5.amplifyapp.com`

Componentes internos (gestionados automáticamente):

- **S3**: Almacena los archivos HTML, CSS, JS compilados
- **CloudFront**: CDN que entrega los archivos rápidamente desde servidores edge
- **Certificate Manager**: Proporciona el certificado SSL/TLS

---

2. VPC (Virtual Private Cloud)

¿Qué es?
Red virtual privada y aislada donde viven todos los recursos (EC2, RDS).

¿Por qué lo usamos?

- **Seguridad por capas**: Separa recursos públicos (EC2) de privados (RDS)
- **Control total del tráfico**: Definimos qué puede comunicarse con qué
- **Subredes públicas y privadas**: EC2 tiene acceso a internet, RDS no
- **Aislamiento**: Otros usuarios de AWS no pueden acceder a nuestra red

¿Qué hace en el proyecto?

- Crea una red privada con rango de IPs (ejemplo: 10.0.0.0/16)
- Divide la red en subredes:
    - **Subred pública**: EC2 (con acceso a internet vía Internet Gateway)
    - **Subred privada**: RDS (sin acceso directo desde internet)
- Configura tablas de ruteo para controlar el tráfico

Estructura:

```
VPC (10.0.0.0/16)
├── Subred Pública (10.0.1.0/24)
│   └── EC2 + Elastic IP
├── Subred Privada (10.0.2.0/24)
│   └── RDS SQL Server
└── Internet Gateway (puerta de salida/entrada)
```

---

3. EC2 (Elastic Compute Cloud)

¿Qué es?
Máquina virtual (servidor) en la nube donde ejecutamos software personalizado.

¿Por qué lo usamos?

- **Control total**: Instalamos Node.js, NGINX, PM2, lo que necesitemos
- **Flexibilidad**: Acceso SSH para editar archivos, ver logs, gestionar servicios
- **Backend personalizado**: Express con lógica de negocio específica
- **Persistencia**: Con PM2 y startup scripts, el backend sobrevive a reinicios

¿Qué hace en el proyecto?

- Instancia: **Amazon Linux 2023** (sistema operativo)
- Software instalado:
  - **Node.js 18**: Runtime para ejecutar el backend Express
  - **NGINX**: Servidor web proxy inverso con SSL
  - **PM2**: Administrador de procesos Node.js
- Procesos corriendo:
  - `pm2: asgp-api` (puerto 3000) - Backend Express
  - `nginx` (puertos 80, 443) - Proxy HTTPS
- Ubicación: Subred pública de la VPC
- Security Group: Permite puerto 443 (HTTPS) y 22 (SSH)

ID de la instancia: `i-09fddc8815bd707a8`

---

4. Elastic IP

¿Qué es?
Dirección IP pública fija que reservamos y asociamos a EC2.

¿Por qué lo usamos?

- **Dirección permanente**: 98.95.235.51 no cambia aunque reiniciemos EC2
- **Sin DNS necesario**: No necesitamos comprar un dominio para tener IP fija
- **Frontend estable**: El código tiene hardcoded esta IP, si cambiara dejaría de funcionar
- **Presentación confiable**: Podemos apagar y reiniciar el lab sin perder conectividad

¿Qué hace en el proyecto?

- IP reservada: **98.95.235.51**
- Asociada a EC2: `i-09fddc8815bd707a8`
- El frontend hace peticiones a: `https://98.95.235.51/api/...`
- NGINX escucha en esta IP en el puerto 443

Costo: Gratis mientras esté asociada y en uso (puede generar cargo si se libera sin usar)

---

5. NGINX (en EC2)

¿Qué es?
Servidor web de alto rendimiento que actúa como proxy inverso.

¿Por qué lo usamos?

- **Terminación SSL/TLS**: Maneja el certificado HTTPS, cifra/descifra el tráfico
- **Proxy inverso**: Recibe peticiones en puerto 443 → las reenvía a Node.js en puerto 3000
- **Seguridad**: El backend Node.js no está expuesto directamente a internet
- **Puerto estándar**: Los navegadores esperan HTTPS en 443, Node.js corre en 3000
- **Performance**: Extremadamente eficiente manejando miles de conexiones concurrentes

¿Qué hace en el proyecto?

- Escucha en:
  - Puerto 80 (HTTP) → redirige a HTTPS
  - Puerto 443 (HTTPS) → proxy a `http://localhost:3000`
- Certificado SSL: `/etc/nginx/ssl/nginx.crt` (auto-firmado, 365 días)
- Configuración: `/etc/nginx/conf.d/asgp.conf`
- Headers agregados:
  - `X-Real-IP`: IP del cliente
  - `X-Forwarded-For`: Para saber origen real
  - `X-Forwarded-Proto`: Indica que originalmente era HTTPS

Flujo:

```
Internet (443) → NGINX (443) → Node.js (3000) → Respuesta
```

---

6. RDS (Relational Database Service) - SQL Server

¿Qué es?
Base de datos SQL Server administrada por AWS.

¿Por qué lo usamos?

- **Administración automática**: AWS gestiona backups, parches de seguridad, actualizaciones
- **Alta disponibilidad**: Podemos configurar réplicas en múltiples zonas (no lo hicimos pero es posible)
- **Seguridad**: Está en subred privada, solo accesible desde EC2
- **Sin mantenimiento manual**: No necesitamos instalar SQL Server en EC2 ni preocuparnos por corrupción de datos
- **SQL Server familiar**: El proyecto ya estaba diseñado con SQL Server

¿Qué hace en el proyecto?

- Motor: **SQL Server Express Edition**
- Endpoint: `asgp-db.cpmk4o00wr2f.us-east-1.rds.amazonaws.com:1433`
- Base de datos: `ASGP_DB`
- Tablas: Usuarios, Productos, Ventas, Stock, CierreCaja, Roles
- Ubicación: Subred privada de la VPC
- Security Group: Solo permite puerto 1433 desde el security group de EC2
- Credenciales: Configuradas en `backend/.env`

Conexión desde backend:

```
// backend/config/database.js
server: 'asgp-db.cpmk4o00wr2f.us-east-1.rds.amazonaws.com',
database: 'ASGP_DB',
user: process.env.DB_USER,
password: process.env.DB_PASSWORD,
port: 1433
```

---

7. Security Groups

¿Qué es?
Firewall virtual a nivel de instancia que controla tráfico entrante y saliente.

¿Por qué lo usamos?

- **Principio de mínimo privilegio**: Solo abrimos puertos estrictamente necesarios
- **Protección contra ataques**: Bloquea intentos de conexión a puertos no autorizados
- **Control granular**: Podemos especificar IPs, rangos, o referenciar otros security groups
- **Sin costo adicional**: Viene incluido con VPC

¿Qué hace en el proyecto?

Security Group de EC2:

- **Inbound (entrada):**
  - Puerto 443 (HTTPS): Desde cualquier IP (0.0.0.0/0) - para que el frontend pueda llamar a la API
  - Puerto 22 (SSH): Desde cualquier IP (0.0.0.0/0) - para acceso administrativo
- **Outbound (salida):**
  - Todo el tráfico permitido (necesario para conectar a RDS, descargar paquetes npm, etc.)

Security Group de RDS:

- **Inbound (entrada):**
  - Puerto 1433 (SQL Server): Solo desde el Security Group de EC2 - nadie más puede conectarse
- **Outbound (salida):**
  - No necesita salir a internet

Ejemplo de regla:

```
RDS Security Group:
  Inbound: Puerto 1433, Source: sg-ec2 (solo EC2 puede conectarse)

EC2 Security Group:
  Inbound: Puerto 443 (0.0.0.0/0), Puerto 22 (0.0.0.0/0)
  Outbound: Todo permitido
```

---

8. AWS Cognito

¿Qué es?
Servicio de autenticación y gestión de usuarios que proporciona login OAuth 2.0.

¿Por qué lo usamos?

- **Opción alternativa de login**: El proyecto tiene doble autenticación (Cognito o usuario/contraseña en RDS)
- **Seguridad delegada**: Cognito maneja hash de contraseñas, recuperación de cuenta, MFA
- **Escalabilidad**: Puede manejar millones de usuarios sin cambios en el backend
- **Flujo OAuth estándar**: Redirección segura, el frontend nunca ve la contraseña directamente
- **Tokens JWT**: Genera tokens estándar que el backend puede validar

¿Qué hace en el proyecto?

- User Pool ID: Gestiona usuarios Cognito (registro, login, recuperación)
- App Client ID: `59ef8tvgiv9ud49khfpi0tpeuj`
- Callback URL: `https://main.d2dxwi2afljxg5.amplifyapp.com/oauth-callback`
- Flujo:

    1. Usuario hace clic en "Iniciar con Cognito"
    2. Frontend redirige al dominio de Cognito
    3. Usuario se autentica en la pantalla de AWS
    4. Cognito redirige de vuelta con un token en la URL
    5. Frontend lee el token y crea sesión local

Configurado en:

- `frontend/src/environments/environment.ts` (config Cognito)
- `frontend/src/app/components/login/oauth-callback.component.ts` (maneja el callback)

---

9. PM2 (Process Manager)

¿Qué es?
Administrador de procesos Node.js (no es un servicio AWS, se instala en EC2).

¿Por qué lo usamos?

- **Auto-restart**: Si el backend crashea, PM2 lo reinicia automáticamente
- **Startup script**: Configurado para iniciar el backend cuando EC2 se reinicia
- **Logs centralizados**: `pm2 logs` muestra todos los console.log del backend
- **Control fácil**: `pm2 restart asgp-api` reinicia sin perder configuración
- **Disponibilidad 24/7**: Mantiene el proceso vivo incluso si cerramos SSH

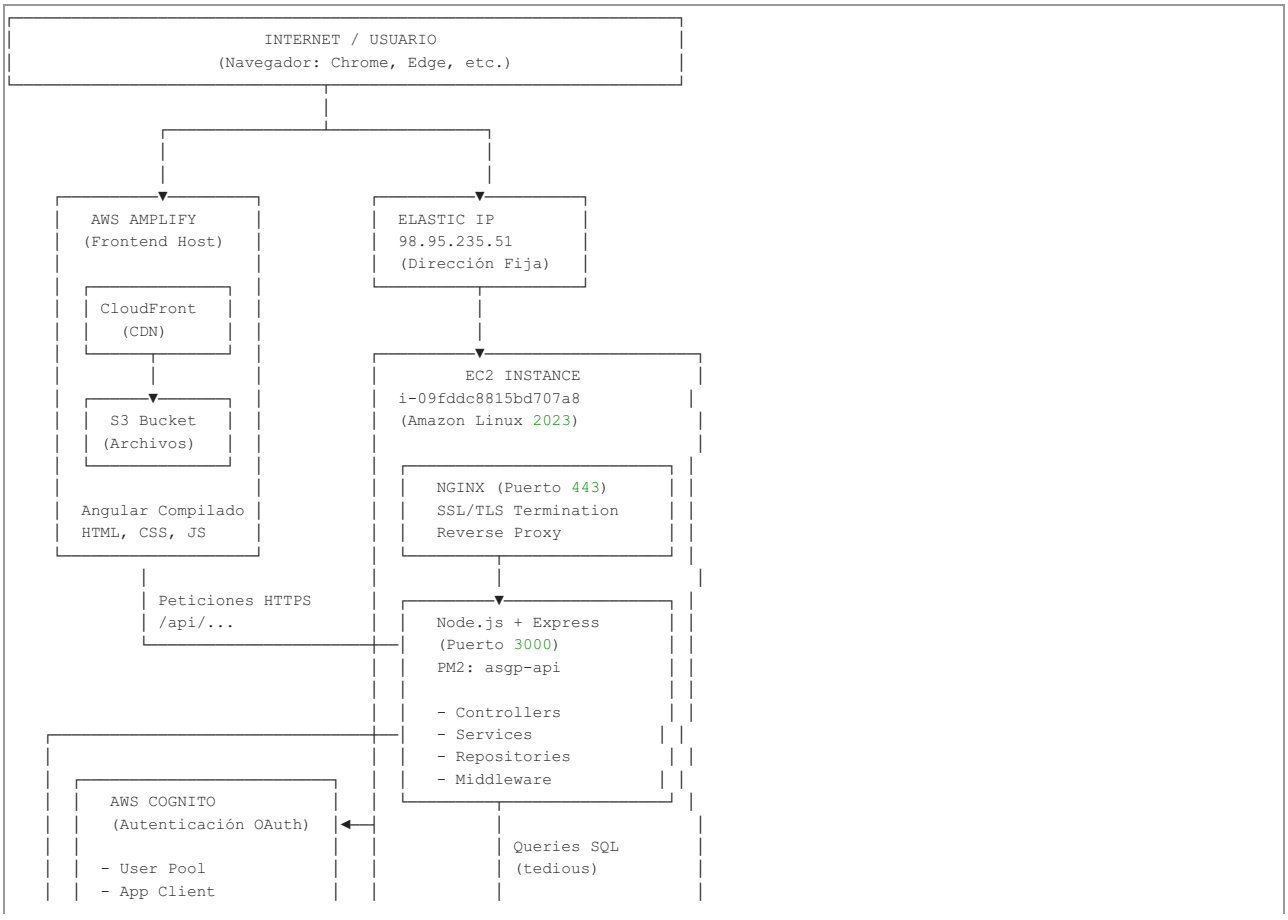
¿Qué hace en el proyecto?

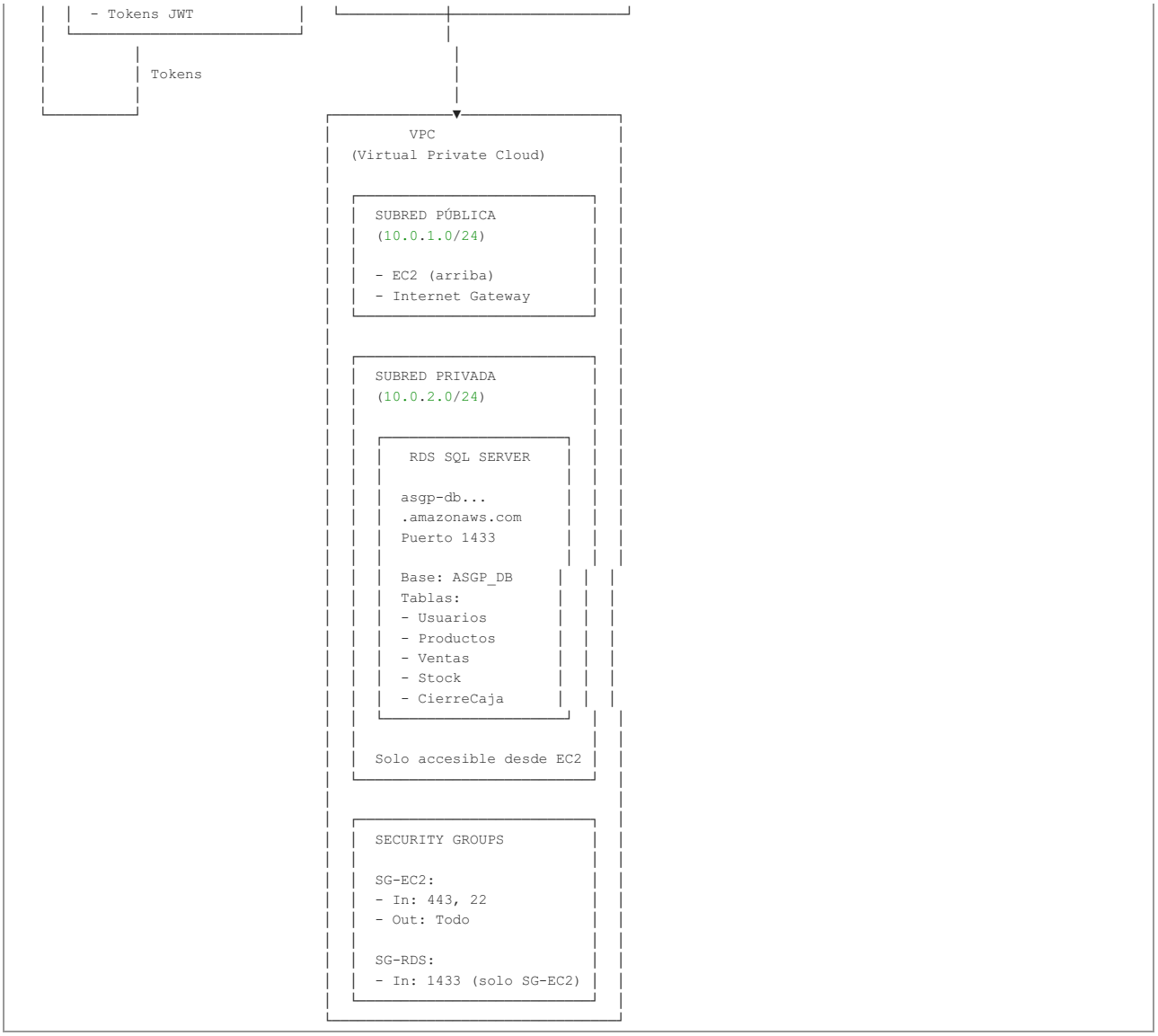
- Proceso: `asgp-api` (id: 0)
- Comando: `pm2 start server.js --name asgp-api`
- Startup script: `/etc/systemd/system/pm2-ec2-user.service`
- Archivo de estado: `/home/ec2-user/.pm2/dump.pm2`
- Logs: `/home/ec2-user/.pm2/logs/asgp-api-out.log` y `asgp-api-error.log`

Comandos útiles:

```
pm2 ls                  # Lista procesos
pm2 logs asgp-api       # Ver logs en tiempo real
pm2 restart asgp-api    # Reiniciar backend
pm2 save                # Guardar configuración actual
```

Mapa de Arquitectura AWS





Flujo de Conexiones

Flujo 1: Usuario accede al frontend

```
1. Usuario escribe: https://main.d2dxwi2afljxg5.amplifyapp.com
2. DNS resuelve → CloudFront (CDN de AWS)
3. CloudFront busca archivos en S3 (cache)
4. Si no está en cache, S3 entrega los archivos (HTML, CSS, JS)
5. CloudFront sirve al usuario desde servidor edge cercano
6. Navegador carga Angular (Single Page Application)
```

Flujo 2: Login con usuario/contraseña

```
1. Usuario completa formulario (email + contraseña)
2. Frontend envía: POST https://98.95.235.51/api/auth/login
3. Petición llega a Elastic IP (98.95.235.51)
4. NGINX recibe en puerto 443 (HTTPS)
5. NGINX descifra SSL y envía a: http://localhost:3000/api/auth/login
6. Express (Node.js) recibe la petición
7. Controller → Service → Repository
8. Repository consulta RDS: SELECT * FROM Usuarios WHERE mail = ?
9. RDS devuelve registro del usuario
10. Service valida contraseña con bcrypt
11. Si es válida, genera JWT con jsonwebtoken
12. Express responde: {success: true, token: "eyJ...", usuario: {...}}
13. NGINX cifra la respuesta con SSL
14. Frontend recibe respuesta, guarda token en localStorage
15. Redirige a /dashboard según rol (ADMIN o EMPLEADO)
```

Flujo 3: Login con Cognito (OAuth)

```
1. Usuario hace clic en "Iniciar con Cognito"
2. Frontend redirige a: https://[cognito-domain].auth.us-east-1.amazoncognito.com/login?...
3. Usuario ve pantalla de login de AWS Cognito
4. Usuario ingresa credenciales Cognito
5. Cognito valida y genera tokens
6. Cognito redirige a: https://main.d2dxwi2afljxg5.amplifyapp.com/oauth-callback?code=...
7. Frontend (oauth-callback.component.ts) extrae el código
8. Frontend intercambia código por tokens (id_token, access_token)
9. Frontend guarda tokens en localStorage
10. Redirige a /dashboard
```

Flujo 4: Consulta de productos (ruta protegida)

```
1. Usuario en /dashboard hace clic en "Productos"
2. Frontend envía: GET https://98.95.235.51/api/productos
   Headers: Authorization: Bearer eyJ...
3. Petición llega a NGINX → Express
4. Middleware (auth.js) intercepta la petición
5. Valida JWT con jsonwebtoken.verify(token, secret)
6. Si es válido, extrae usuario y continúa
7. Controller → Service → Repository
8. Repository consulta RDS: SELECT * FROM Productos
9. RDS devuelve lista de productos
10. Express responde: {success: true, data: [{id:1, nombre:...}, ...]}
11. Frontend recibe y muestra tabla de productos
```

Flujo 5: Registro de venta

```
1. Empleado completa formulario de venta
2. Frontend envía: POST https://98.95.235.51/api/ventas
   Body: {producto_id: 1, cantidad: 5, ...}
   Headers: Authorization: Bearer eyJ...
3. NGINX → Express → Middleware valida JWT
4. Controller → Service
5. Service inicia transacción SQL
6. Repository 1: INSERT INTO Ventas (...)
7. Repository 2: UPDATE Stock SET cantidad = cantidad - 5 WHERE producto_id = 1
8. Si ambas queries tienen éxito, COMMIT
9. Si alguna falla, ROLLBACK
10. Express responde: {success: true, message: "Venta registrada"}
11. Frontend actualiza tabla de ventas
```

Flujo 6: Reinicio de EC2 (persistencia)

```
1. Usuario cierra AWS Lab o reinicia EC2
2. EC2 se apaga → procesos terminan
3. EC2 vuelve a iniciar
4. Elastic IP 98.95.235.51 se mantiene asociada (no cambia)
5. systemd inicia nginx.service (habilitado con systemctl enable)
6. systemd inicia pm2-ec2-user.service (configurado con pm2 startup)
7. PM2 lee /home/ec2-user/.pm2/dump.pm2
8. PM2 ejecuta: pm2 start server.js --name asgp-api
9. Node.js conecta a RDS: asgp-db.cpmk4o00wr2f.us-east-1.rds.amazonaws.com:1433
10. Backend responde: http://localhost:3000 (listo para recibir peticiones)
11. NGINX proxy funciona: 443 → 3000
12. Sistema completamente operativo sin intervención manual
```

Tabla Resumen

Servicio	Función	Puerto/Endpoint	Ubicación	Por Qué Se Usa
Amplify	Hosting frontend	HTTPS (443)	Región AWS	Deploy automático desde GitHub con SSL incluido
CloudFront	CDN (dentro de Amplify)	N/A	Global (edge locations)	Distribución rápida de archivos estáticos
S3	Almacenamiento (dentro de Amplify)	N/A	us-east-1	Guardar archivos HTML, CSS, JS compilados
VPC	Red privada virtual	N/A	us-east-1	Aislar y proteger recursos, subredes públicas/privadas
EC2	Servidor backend	22 (SSH), 3000 (Node)	Subred pública VPC	Ejecutar Node.js + Express + NGINX con control total
Elastic IP	IP pública fija	N/A	Asociada a EC2	Evitar cambio de IP al reiniciar EC2
NGINX	Proxy + SSL	80, 443	Dentro de EC2	Terminación SSL, mapeo 443→3000, seguridad
Node.js	Backend API	3000	Dentro de EC2	Lógica de negocio, endpoints REST, autenticación
PM2	Process manager	N/A	Dentro de EC2	Auto-restart, startup automático, logs
RDS	Base de datos	1433 (SQL Server)	Subred privada VPC	Almacenar datos, AWS gestiona backups y parches
Security Groups	Firewall virtual	N/A	VPC	Control de puertos, permitir solo tráfico necesario
Cognito	Autenticación OAuth	N/A	us-east-1	Login alternativo seguro sin código de autenticación

---

Costos Estimados (Proyecto Académico)

Servicio	Costo en AWS Academy Lab
Amplify	Incluido (generalmente gratis en tier gratuito)
EC2 (t2.micro/t3.micro)	Incluido en créditos del lab
Elastic IP	Gratis mientras esté asociada
RDS (db.t3.micro)	Incluido en créditos del lab

VPC	Gratis (sin NAT Gateway)
Security Groups	Gratis
CloudFront	Incluido con Amplify
S3	Incluido con Amplify (pocos MB de archivos)
Cognito	Gratis hasta 50,000 usuarios activos mensuales

Total para este proyecto: $0 (cubierto por AWS Academy Lab)

Producción real (sin créditos académicos):

- EC2 t3.micro: ~$10-15/mes
- RDS db.t3.micro: ~$20-30/mes
- Amplify: ~$5-10/mes (incluye CloudFront + S3)
- Elastic IP: $0 (asociada), $3.60/mes (no asociada)
- Datos de transferencia: variable según tráfico
- **Total estimado:** $35-60/mes para proyecto pequeño

Mejoras Futuras (Post-Presentación)

1. **Certificado SSL válido**:
    - Comprar dominio ($10-15/año)
    - Usar Let's Encrypt con certbot (gratis)
    - O usar AWS Certificate Manager + Application Load Balancer
2. **Auto-escalado**:
    - Application Load Balancer
    - Auto Scaling Group para EC2
    - Múltiples instancias según tráfico
3. **Base de datos**:
    - RDS Multi-AZ (alta disponibilidad)
    - Read Replicas para consultas intensivas
    - Backups automatizados cada 6 horas
4. **Monitoreo**:
    - CloudWatch Logs para logs centralizados
    - CloudWatch Alarms para alertas (CPU >80%, errores HTTP 500)
    - AWS X-Ray para tracing de peticiones
5. **CI/CD backend**:
    - CodePipeline para deploy automático de backend
    - CodeDeploy para actualizar EC2 sin downtime
6. **Seguridad adicional**:
    - AWS WAF (Web Application Firewall)
    - AWS Shield (protección DDoS)
    - Secrets Manager para credenciales (en lugar de .env)

Documento generado el: 20 de noviembre de 2025
Proyecto: ASGP Full Stack
Propósito: Presentación académica AWS Academy
Estado: Producción (Lab AWS)