
Documento de Arquitectura – Célula 9

Proyecto: Single Page Application Serverless para ciclo CRUD de gadgets tipo vehículo

Empresa ficticia: *Acme Inc.*
Célula: 9
Integrantes: Sebastián Soto, Elías Ordoñez
Fecha: 2025
Versión: 1.0

Índice

10. Estimación de Costos (Guía)
11. Control de Cambios

---

1. Introducción

Este documento describe la solución asignada a la **Célula 9**, consistente en una **Single Page Application (SPA) serverless** para gestionar el ciclo **CRUD** (Crear, Leer, Actualizar, Eliminar) de gadgets de tipo **vehículo** de *Acme Inc.*

La solución debe implementarse en AWS utilizando servicios serverless, desplegarse mediante CloudFormation, protegerse con Cognito y KMS, residir en una VPC con VPC Endpoints, y contar con un pipeline CI/CD completo de GitHub a producción.

---

2. Escenario Acme Inc.

Acme Inc. es una empresa que diseña gadgets avanzados, en particular **vehículos de fantasía y alta tecnología** (por ejemplo:

- Monopatines voladores
- Vehículos cohete
- Motocicletas antigravedad
- Mini naves personales

El área de diseño y el área comercial necesitan un sistema web sencillo pero robusto para:

- Registrar nuevos gadgets de tipo vehículo
- Consultar el catálogo completo
- Modificar especificaciones (velocidad máxima, tipo de propulsión, número de plazas, estatus de producción)
- Eliminar gadgets obsoletos

Se requiere una **SPA** moderna, servida desde S3/CloudFront, con un backend serverless en AWS que exponga APIs protegidas.

---

3. Objetivo de la Arquitectura

Diseñar y desplegar una arquitectura serverless que permita:

- Operar el ciclo **CRUD** completo sobre gadgets de tipo vehículo.
- Proveer una SPA estática con comunicación vía APIs.
- Autenticar y autorizar usuarios mediante Amazon Cognito.
- Persistir información de gadgets en DynamoDB.
- Cifrar datos y comunicaciones usando KMS y buenas prácticas de seguridad.
- Desplegarse vía pipeline CI/CD en sandbox, pre-producción y producción.
- Exponer pruebas de los endpoints mediante `curl`.

---

4. Alcance

Incluye

- SPA estática (frontend) hospedada en S3 y distribuida con CloudFront.
- APIs REST/HTTP en Amazon API Gateway.
- Funciones Lambda (Python) para CRUD de gadgets.
- Tabla DynamoDB para gadgets de tipo vehículo.
- Integración con Cognito para autenticación.
- Cifrado con KMS en datos en reposo y sensibles.
- Pipeline CI/CD: GitHub → CodePipeline → CodeBuild → CloudFormation.

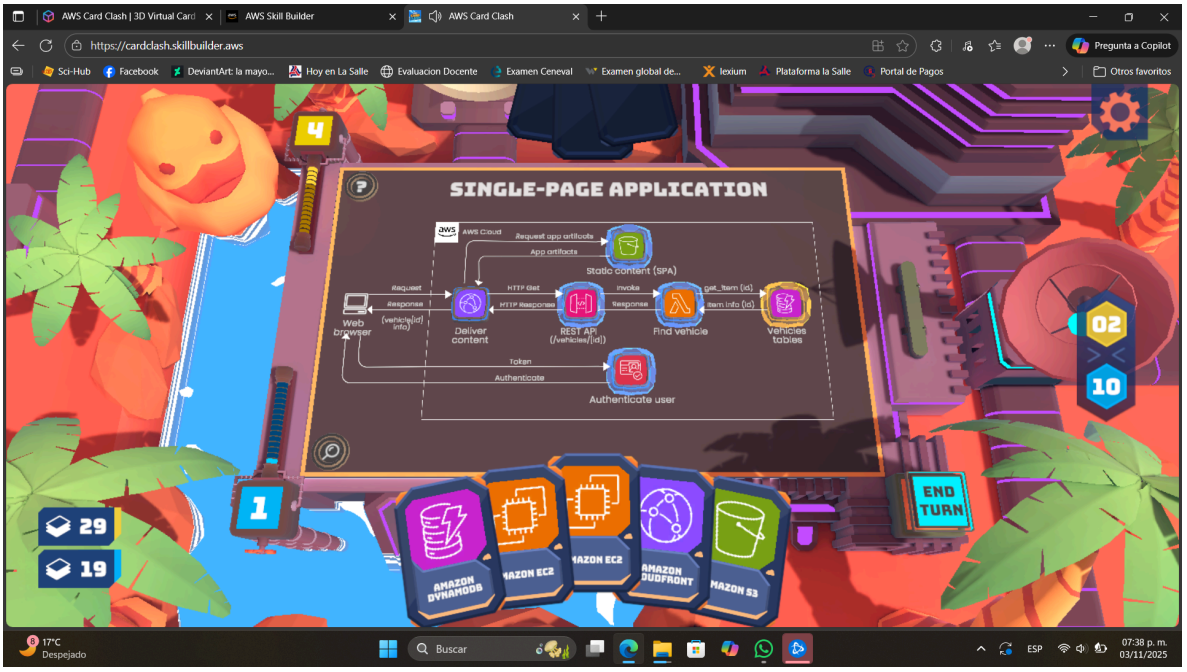
- Generación de al menos 50 gadgets de prueba con GenAI.
- Diagrama de arquitectura en draw.io.
- Pruebas de endpoints con `curl`.

No incluye

- Frontend avanzado con diseño gráfico profesional (basta SPA funcional).
- Procesos complejos de negocio más allá de CRUD básico.
- Integración con sistemas externos (ERP real, pagos, etc.).

---

5. Arquitectura Propuesta



5.1 Componentes Principales

- **Amazon S3**
    - Bucket para contenido estático (HTML, JS, CSS) de la SPA.

    - Cifrado SSE-KMS obligatorio.

- **Amazon CloudFront**
    - Distribución de la SPA con baja latencia.
    - Integración con el bucket S3 origen.
- **Amazon API Gateway (HTTP o REST API)**
    - Expone endpoints para el backend CRUD de gadgets de vehículo.
    - Integrado con Lambda.
    - Protegido por Cognito Authorizer.
- **AWS Lambda (Python)**
    - Función o funciones para:
        - Crear gadget (`CreateVehicleGadget`)
        - Obtener gadgets (`GetVehicleGadgets`, `GetVehicleGadgetById`)
        - Actualizar gadget (`UpdateVehicleGadget`)
        - Borrar gadget (`DeleteVehicleGadget`)
- **Amazon DynamoDB**
    - Tabla `VehicleGadgets` con llaves sugeridas:
        - `GadgetId` (PK)
    - Atributos de ejemplo:
        - `Name`, `Category`, `MaxSpeed`, `PropulsionType`, `Seats`, `Status`, `CreatedAt`
- **Amazon Cognito**
    - User Pool para usuarios de la SPA.
    - Autenticación (username/password u otros flujos que defina el equipo).

- **AWS KMS**
    - En S3 (SSE-KMS).
    - En DynamoDB.
    - En secretos y variables de entorno sensibles.
- **VPC + VPC Endpoints + Zona de inspección**
    - Lambdas en subredes privadas.
    - Endpoints de VPC para DynamoDB, S3, Logs, Secrets Manager, etc.
- **AWS CodePipeline + CodeBuild + CloudFormation**
    - Pipeline de despliegue:
        - Código SPA y backend desde GitHub.
        - Build (empaquetado y validación).
        - Deploy a cuentas sandbox → pre-producción → producción.

5.2 Flujo de Datos

1. El usuario accede a la SPA vía **CloudFront**, que sirve contenido estático desde S3.
2. El usuario inicia sesión mediante Cognito (desde la SPA).
3. La SPA obtiene un **token JWT** de Cognito.
4. La SPA invoca APIs en API Gateway, incluyendo el JWT en el header `Authorization`.
5. API Gateway valida el token con Cognito (Cognito Authorizer).
6. API Gateway invoca la Lambda correspondiente (operación CRUD deseada).
7. Lambda realiza operaciones en DynamoDB (a través de VPC Endpoint), con cifrado habilitado vía KMS.
8. La respuesta viaja de regreso: Lambda → API Gateway → SPA.

9. La SPA actualiza la interfaz con los resultados (lista, detalle, confirmación de cambios, etc.).

---

6. Lineamientos Técnicos Obligatorios

6.1 Infraestructura como Código

- Todos los recursos (S3, CloudFront, API Gateway, Lambdas, DynamoDB, Cognito, VPC Endpoints, etc.) se definen en **CloudFormation**.
- No se permiten cambios manuales en cuentas objetivo ni de producción.

6.2 Python y Generación de Datos

- Todo el backend (Lambdas) debe implementarse en **Python**.
- Deben generarse al menos **50 gadgets de tipo vehículo** usando GenAI (ej. Kiro), con datos razonables (nombre, tipo de vehículo, velocidad, etc.).
- Se recomienda almacenar datos de ejemplo en `data/` y/o scripts de carga.

6.3 API Gateway + Cognito

- Los endpoints de CRUD estarán detrás de API Gateway.
- Cognito se utiliza para:
    - Registro/autenticación de usuarios.
    - Emisión de JWT.
- API Gateway debe usar un **Cognito Authorizer** para validar tokens en cada operación.

6.4 Seguridad (VPC, KMS, VPC Endpoints, Inspección)

- Lambdas en subredes privadas dentro de una VPC.
- VPC Endpoints para:
    - DynamoDB
    - S3
    - Logs
    - Secrets Manager, etc.
- Cifrado con **AWS KMS** en:
    - S3 (bucket de la SPA, y cualquier bucket adicional).
    - DynamoDB (tabla `VehicleGadgets`).
    - Secrets y variables de entorno.
- El tráfico debe pasar por la **zona de inspección** definida por Ciberseguridad.

6.5 Pipeline CI/CD

- Repositorio central en **GitHub** con:
    - `src/` (Lambda backend)
    - `frontend/` o similar (SPA)
    - `iac/` (CloudFormation)
    - `buildspec.yml`
    - `README.md`
- Pipeline:
    - **Source**: GitHub
    - **Build**: CodeBuild (build del frontend, empaquetado backend, validación de plantillas).
    - **Deploy**: CloudFormation (sandbox → pre-producción → producción).

---

7. Entregables

1. Repositorio GitHub con la estructura mencionada.
2. Plantillas de CloudFormation para todos los recursos.
3. SPA funcional (CRUD vehículos) desplegada en S3 + CloudFront.
4. Backend serverless en Lambda + API Gateway.
5. Integración con Cognito (login y uso de JWT).
6. Datos de ejemplo generados con GenAI (mínimo 50 gadgets).
7. Pipeline CI/CD funcionando en sandbox y cuenta objetivo, y habilitado para producción.
8. Diagrama de arquitectura en draw.io.
9. Historias de usuario y backlog básico.
10. Estimación de costos con AWS Pricing Calculator.
11. Documento de pruebas con `curl`.

---

8. Pruebas con Curl

Ejemplos de pruebas que deben documentar (los nombres de rutas son ilustrativos):

1. Obtener token JWT (flujo simplificado con Cognito)

```
curl -X POST https://<cognito-domain>/oauth2/token \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d
"grant_type=password&client_id=<client_id>&username=<user>&password=<password>"
```

2. Crear un gadget de tipo vehículo

```
curl -X POST https://<api-id>.execute-api.<region>.amazonaws.com/prod/vehicles \
  -H "Authorization: Bearer <JWT>" \
```

```
  -H "Content-Type: application/json" \
  -d '{
        "name": "Hoverbike X1",
        "category": "vehicle",
        "maxSpeed": 280,
        "propulsionType": "antigravity",
        "seats": 1,
        "status": "prototype"
      }'
```

3. Listar todos los gadgets

```
curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/vehicles \
  -H "Authorization: Bearer <JWT>"
```

4. Actualizar un gadget

```
curl -X PUT
https://<api-id>.execute-api.<region>.amazonaws.com/prod/vehicles/<GadgetId> \
  -H "Authorization: Bearer <JWT>" \
  -H "Content-Type: application/json" \
  -d '{
        "maxSpeed": 300,
        "status": "production"
      }'
```

5. Eliminar un gadget

```
curl -X DELETE
https://<api-id>.execute-api.<region>.amazonaws.com/prod/vehicles/<GadgetId> \
  -H "Authorization: Bearer <JWT>"
```

---

9. Historias de Usuario

HU-01 – Registrar un nuevo gadget vehículo

Como diseñador de productos en Acme Inc.
Quiero registrar un nuevo gadget vehículo en el sistema
Para mantener actualizado el catálogo de gadgets disponibles.

HU-02 – Consultar catálogo de gadgets

Como miembro del equipo comercial
Quiero ver la lista de gadgets vehículo disponibles
Para preparar propuestas para los clientes.

HU-03 – Actualizar especificaciones

Como ingeniero de producto
Quiero modificar velocidad máxima y estado de producción
Para reflejar cambios técnicos en el sistema.

HU-04 – Eliminar gadgets obsoletos

Como administrador del catálogo
Quiero eliminar gadgets que ya no se fabrican
Para evitar confusiones en el equipo comercial.

---

10. Estimación de Costos (Guía)

La célula debe usar **AWS Pricing Calculator** considerando:

- S3 (almacenamiento de SPA y datos).
- CloudFront (transferencia de salida aproximada).
- Lambda (invocaciones + duración).
- API Gateway (número de llamadas).
- DynamoDB (lecturas/escrituras, almacenamiento).
- Cognito (MAUs estimados).
- CodePipeline / CodeBuild.
- KMS (uso de claves).
- VPC Endpoints.

---

11. Control de Cambios

Versión	Fecha	Cambios	Autor
1.0	2025	Primera versión formal del Documento de Célula 9	Arquitecto del Proyecto

---