Curso: Cloud Infrastructure

Catedráticos: Joseph Arreola y Alma Orrego

Justificación Arquitectónica

Mayco Castellanos

Isaac Cyrman

Christian Barrios

Decisiones Arquitectónicas GuatePASS

1. Arquitectura Serverless

Servicios utilizados: Lambda, API Gateway, DynamoDB, Step Functions, EventBridge, S3 y CloudWatch.

Alternativas descartadas:

- EC2 + RDS: Mayor control, pero con alta carga operativa, costos fijos y sin el enfoque serverless requerido.
- ECS/Fargate: Portabilidad, pero añade complejidad operativa innecesaria para un flujo basado en eventos.

Justificación:

- Escalamiento automático sin gestión de servidores.
- Modelo costo-efectivo basado en uso real.
- Reducción significativa de carga operativa.

2. Infrastructure as Code con AWS SAM

Infraestructura definida en template.yaml (17 Lambdas, 4 tablas, API, Step Function, Event Bus, Dashboard).

Alternativas descartadas:

- CloudFormation puro: Excesivamente verboso y difícil de mantener.
- Terraform: Requiere gestión de estado y complejidad innecesaria.
- Serverless Framework: Viable, pero SAM es la opción nativa optimizada para AWS.

Justificación:

- Sintaxis simplificada y orientada a aplicaciones serverless.
- Herramientas de desarrollo local rápidas (sam local start-api, sam logs).
- Menos código y mayor mantenibilidad.
- Mejores prácticas integradas (IAM mínimo necesario, logs, etc.).

3. **Arquitectura Híbrida (Síncrona + Asíncrona)**

    a. **Flujo implementado:**

        i. `POST /webhook/toll` (Síncrono)

        ii. Lambda `IngestToll` (valida, publica en EventBridge y responde 200/400)

        iii. `Step Function` (procesamiento asíncrono).

    b. **Alternativas descartadas:**

        i. Síncrona Pura: Inviable. Forzaría al cliente a esperar 5-10 segundos por una respuesta.

        ii. Asíncrona Pura: No permite validación síncrona. El cliente no sabría si envió un *payload* erróneo.

    c. **Justificación:**

        i. **Tiempo de Respuesta:** El cliente recibe una confirmación de validación inmediata (<300ms).

        ii. **Resiliencia:** El procesamiento pesado (cobro, factura) se delega a la Step Function asíncrona, que maneja sus propios reintentos sin afectar al cliente.

        iii. **Experiencia del Cliente API:** Combina la velocidad de un sistema asíncrono con la inteligencia (validación) de uno síncrono.

        iv. **Separación de Responsabilidades:** Separa la "ingesta" (que debe ser rápida) de la "lógica de negocio" (que es compleja).

4. **Orquestación con Step Functions**

Utilizada para orquestar los pasos del flujo de cobro.

Alternativas descartadas:

- Lambda monolítica: Baja mantenibilidad y difícil depuración.
- SQS con Lambdas encadenadas: Difícil trazabilidad del flujo completo.

Justificación:

- Orquestación visual del proceso.

- Manejo de errores robusto con reintentos.
- Separación clara de responsabilidades por función.
- Auditoría completa de cada ejecución.
- Costo bajo frente al valor obtenido.

5. Enrutamiento de Eventos con EventBridge

Bus de eventos para desacoplar ingestión y procesamiento.

Alternativas descartadas:

- SNS: No soporta enrutamiento basado en contenido.
- SQS: No es un bus de eventos y complica múltiples consumidores.

Justificación:

- Enrutamiento avanzado por contenido del evento.
- Desacoplamiento total entre productor y consumidores.
- Escalabilidad automática.
- Facilidad para agregar nuevos consumidores sin modificar el flujo existente.

6. Diseño de Base de Datos (DynamoDB)

Tablas separadas: Users, Tolls, Transactions, Invoices.

Justificación:

- Patrones de acceso distintos entre entidades.
- Mayor simplicidad de claves e IAM.
- Mejor rendimiento para cargas diferentes (consultas vs transacciones).

Diseño por tabla:

- GuatepassUsers: PK = placa, GSI para búsqueda por tag_id.
- Transactions e Invoices: PK por ID, GSI por placa con ordenamiento por fecha.

Uso de GSI:

- Permite consultas eficientes por placa sin escanear la tabla completa.
- Costo adicional mínimo comparado con el beneficio en rendimiento.