
Documento Específico de la Célula 6

Servicio de Procesamiento Paralelo “Ubertoons” – Acme Inc.

Proyecto: Arquitecturas Serverless en AWS
Célula: 6
Integrantes: Edgar Espinosa, Adrián Leyva
Versión: 1.0
Organización: Universidad La Salle – Facultad de Ingeniería
Arquitectura Asignada: Parallel Data Processing
Caso de uso: Servicio Ubertoons, sistema de render, fusión y transformación masiva de datos multimedia y telemétricos en Acme Inc.

Índice

---

1. Introducción

La Célula 6 es responsable del diseño e implementación del sistema **Ubertoons**, un motor serverless de Acme Inc. especializado en realizar **procesamiento paralelo, fan-out y fan-in** de grandes cantidades de datos.

Ubertoons constituye la columna vertebral de varios subsistemas de Acme:

- análisis de telemetría en tiempo real
- procesamientos masivos de imágenes
- integración con pipelines de ML
- flujos de validación y limpieza de datos

El diseño debe ser completamente **serverless**, resiliente, seguro y altamente escalable.

---

2. Objetivo del Módulo

Construir un servicio que permita ejecutar **procesamiento paralelo masivo** de eventos, archivos o mensajes provenientes de distintos sistemas de Acme Inc., utilizando técnicas de:

- fan-out (distribución)
- procesamiento concurrente
- procesamiento orquestado (fan-in opcional)

- almacenamiento de resultados
- exposición de APIs seguras para interactuar con Ubertoons

---

3. Alcance

Incluye:

- Procesamiento paralelo de datos (S3, SQS o Kinesis como entrada)
- Distribución automática de tareas mediante SNS/SQS
- Proceso principal mediante lambdas
- Agregación opcional con Step Functions
- Registro de metadatos y resultados
- Exposición de APIs REST seguras para solicitud y consulta del procesamiento
- Pipeline CI/CD completo
- Pruebas con curl

No incluye:

- Pipelines de ML
- Procesamientos GPU intensivos fuera del scope de Lambda
- Integración con sistemas heredados no serverless

---

4. Descripción del Servicio Ubertoons

Ubertoons es un subsistema que procesa “toons”:

unidades de datos multimedia, telemétricos o informativos que deben procesarse en paralelo.

Ejemplos:

- fragmentos de video o audio
- imágenes
- registros del gameplay (por ejemplo Pinky & Cerebro)
- telemetría de gadgets IoT
- datos crudos generados por otros sistemas

Las características principales del sistema son:

✔ Alta concurrencia

Miles de “toons” pueden ingresar en un intervalo breve.

✔ Procesamiento distribuido

Cada toon debe enviarse a un proceso Lambda independiente.

✔ Agregación opcional

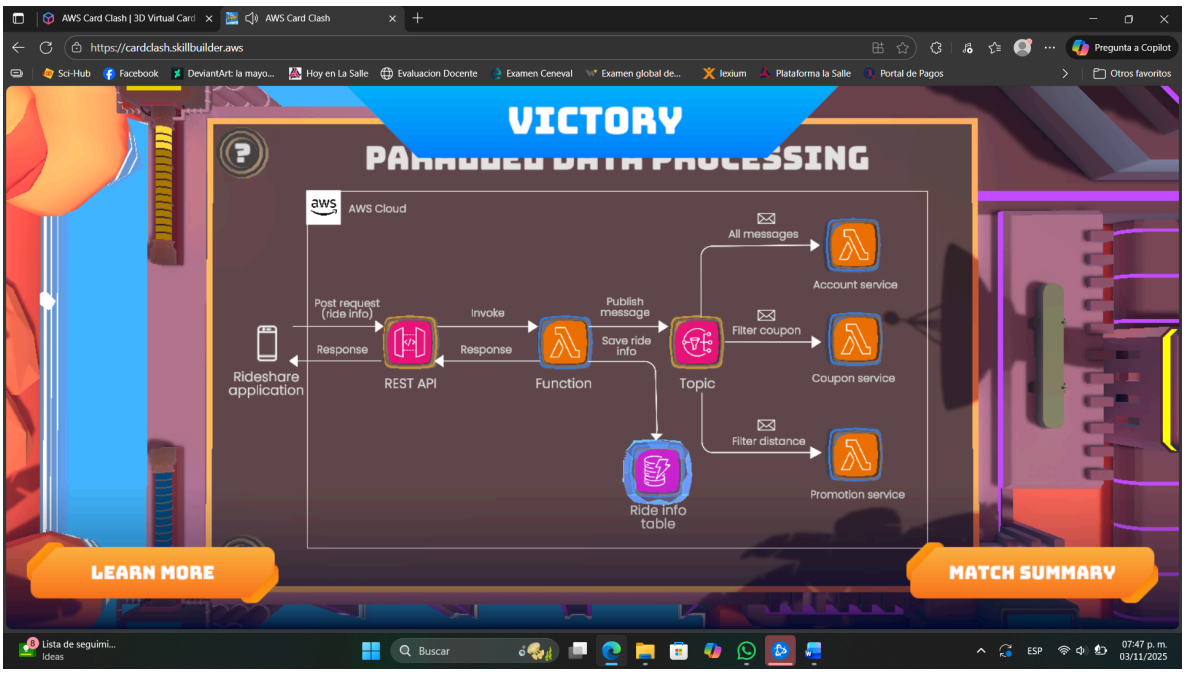
Los resultados pueden agregarse mediante Step Functions.

✔ Elasticidad total

El sistema debe poder escalar automáticamente a la carga.

---

5. Arquitectura General



El flujo sugerido:

1. Un cliente o sistema externo invoca una API Gateway solicitando procesamiento para un lote de toons.
2. La API inserta un lote de mensajes en SQS **o** coloca archivos en S3 **o** publica eventos en EventBridge.
3. Se dispara un mecanismo de **fan-out** mediante SNS o SQS.
4. Cada toon es procesado por una **Lambda Worker** en paralelo.
5. Los resultados se almacenan en:
    - DynamoDB
    - Amazon S3
    - EventBridge (para notificar otros sistemas)
6. Opcionalmente, Step Functions agrega los resultados finales.
7. El cliente consulta su estatus mediante API Gateway.

---

6. Componentes AWS Utilizados

- **S3**, **SQS** o **Kinesis** para ingestión
- **SNS** para fan-out
- **AWS Lambda (Python)** como workers
- **Step Functions** para orquestación (fan-in)
- **API Gateway** para endpoints REST
- **DynamoDB** para resultados/estatus
- **EventBridge** para eventos internos
- **Cognito** para autenticación
- **KMS** para cifrado end-to-end
- **VPC endpoints** + zona de inspección
- **CloudWatch Logs** para telemetría
- **CodePipeline / CodeBuild** para despliegue

---

7. Lineamientos Técnicos Específicos

Procesamiento paralelo

- Debe soportar decenas o cientos de invocaciones simultáneas.
- Lambda worker debe ser idempotente.

Lotes de trabajo

- Un job puede incluir desde 1 hasta miles de “toons”.
- Ubertoons divide automáticamente el trabajo en subunidades.

Estado del job

La salida debe incluir:

- jobId
- número de toons procesados
- estado (Pending / Processing / Completed / Failed)
- metadatos relevantes
- timestamp

Estándares del procesamiento

Cada worker Lambda debe extraer y registrar:

- Identificador del toon
- Tipo de toon (imagen, audio, telemetría, etc.)
- Resultado del procesamiento
- Hash o fingerprint del archivo
- Duración del procesamiento

---

8. IaC: Requerimientos CloudFormation

La plantilla deberá crear:

- SQS queues (input, dead-letter)
- SNS topic para fan-out
- Step Functions (opcional)
- API Gateway
- Lambda Workers en VPC
- DynamoDB
- VPC Endpoints

- Roles IAM (dependientes de la plantilla de Ciberseguridad)
- Variables cifradas
- KMS keys usadas por la célula (si Ciberseguridad no provee una global)

Además:

- Outputs obligatorios
- Parámetros para:
    - User Pool ID
    - Subnets privadas
    - VPC ID
    - Endpoints
    - DynamoDB table name

---

9. Pipeline CI/CD

El pipeline deberá incluir:

1. Source – GitHub

- Código Python
- Templates CloudFormation
- buildspec
- Documentación

2. Build – CodeBuild

- Validar IaC
- Instalar dependencias Python

- Empaquetar Lambdas
- Ejecutar pruebas (opcional)

3. Deploy – CodePipeline + CloudFormation

- Despliegue automático en:
    - sandbox
    - pre-producción
    - producción

---

10. Seguridad, IAM y Cifrado

- Identidad y acceso mediante **Amazon Cognito**
- Todas las Lambdas en **subredes privadas**
- Acceso a AWS mediante **VPC Endpoints**
- Buckets, tablas y logs cifrados con **KMS**
- Roles IAM mínimos, provistos por Ciberseguridad
- Validación estricta de payloads
- Control de trazabilidad
- Cumplimiento del pilar Security del Well-Architected Framework

---

11. Datos de Prueba (GenAI)

La célula deberá generar **al menos 50 toons sintéticos**, cada uno representando un elemento de procesamiento paralelo:

Ejemplos:

- 20 imágenes
- 10 archivos de telemetría
- 10 mensajes simulados
- 10 fragmentos tipo binario

Cada toon debe incluir metadatos generados por GenAI.

---

12. Pruebas funcionales (curl)

Enviar lote a Ubertoons

```
curl -H "Authorization: Bearer <jwt>" \
  -d '{"jobId":"JOB001","toons":[...n items...]}' \
  https://<api>/prod/submit-job
```

Consultar estatus

```
curl -H "Authorization: Bearer <jwt>" \
  https://<api>/prod/jobs/JOB001
```

Consultar resultados

```
curl -H "Authorization: Bearer <jwt>" \
  https://<api>/prod/results?jobId=JOB001
```

---

13. Diagrama de Arquitectura

El diagrama (draw.io) deberá incluir:

- API Gateway
- Cognito
- SQS / SNS
- Lambda Worker Farm

- Step Functions (fan-in opcional)
- DynamoDB
- S3 (opcional según tipo de toon)
- VPC endpoints
- Zona de inspección
- KMS

---

14. Entregables de la Célula

1. Repositorio GitHub completo
2. Plantillas CloudFormation
3. Código Python
4. Pipeline CI/CD
5. Datos sintéticos
6. Pruebas curl
7. Diagrama draw.io
8. Backlog + historias de usuario
9. Estimación de costos

---

15. Criterios de Evaluación

- Correctitud del modelo de fan-out/fan-in
- Calidad del IaC

- Seguridad y cumplimiento
- Comportamiento bajo carga
- Documentación
- Diagrama claro
- Pruebas curl funcionando
- Buen uso de patrones serverless

---

16. Control de Cambios

Versión	Fecha	Descripción	Autor
1.0	2025	Primera versión del documento de la Célula 6	Equipo de Arquitectura