Práctica 2025-26: Cloud Computing (AWS): Dashboard serverless programático para CSV (Web + 3 Lambdas)

Descripción general y objetivos

Debes automizar el **despliegue** de una arquitectura serverless que ingiere **archivos CSV** subidos a **Amazon S3**, almacena los registros en **Amazon DynamoDB** y expone una **UI web muy simple** para ver los datos. Todos los recursos de AWS deben crearse **programáticamente** (por ejemplo, con Python + boto3 o AWS CDK). No se permiten clics manuales en la consola para la configuración principal. Sí se recomienda uso de AWS Consola para comprobar que los recursos AWS son creados y usados correctamente.

Al finalizar, deberías demostrar que puedes:

- Aprovisionar recursos serverless por código (S3, DynamoDB, Lambda, IAM, API Gateway/CloudFront y SNS opcional).
- Invocar **Lambda A** con un evento PutObject de S3 para **parsear CSV** y escribir en DynamoDB.
- Exponer datos a una **web estática** mediante **Lambda B** + **API Gateway**. Usar DynamoDB Streams → Lambda + SNS como función adicional.
- Empaquetar y desplegar código de Lambda desde tu repositorio.

Arquitectura objetivo

- **S3 (bucket de ingesta)**: `inventory-uploads-<sufijo>`
  Los alumnos suben CSV como `inventory-berlin.csv`.
- **Lambda A:** **`load_inventory`** (disparada por S3)
  - Trigger: "All object create events" del bucket de ingesta
  - Parsea CSV e inserta en **DynamoDB** (`Inventory; PK: Store (S), SK: Item (S), atributo: Count (N)`)
- **DynamoDB**: tabla Inventory (on-demand; habilitar Streams *para el notificador*)
- **Lambda B:** **`get_inventory_api`** (endpoint HTTP)
  - Expuesta vía **Amazon API Gateway (HTTP API)** en `/items` y `/items/{store}`
  - Devuelve JSON para la web

- **S3 (bucket web)**: `inventory-web-<sufijo>` con un sitio estático mínimo (`index.html`) que llama a la API y renderiza una tabla
- **IAM**: Roles/políticas de mínimo privilegio para ambas Lambdas
- **Lambda C: notify_low_stock** con **DynamoDB Streams** → **SNS** (NoStock) → suscripción por email. Esta parte puede haceres directamente en AWS console.

La esencia de está práctica es demostrar la combinación de (S3→Lambda→DDB + alertas) y el flujo de despliegue programático.

Entregables

1. **Repositorio de código** con:
    - `/infra/` scripts (Python + boto3) o app CDK que cree **todos** los recursos de extremo a extremo
      (buckets, tabla, roles/policies, Lambdas, API Gateway, hosting S3 del sitio, notificaciones/streams si aplica)
    - `/lambdas/load_inventory/` y `/lambdas/get_inventory_api/` (y `/lambdas/notify_low_stock/`)
    - `/web/index.html` (JS simple que hace `fetch` a la API)
    - `.env.sample` con placeholders (región, emails, etc.)
    - `README.md` con comandos de despliegue/teardown en una sola instrucción
2. **Evidencia de ejecución**:
    - Captura de la **web** mostrando inventario cargado
    - Copia/pega de una **respuesta de la API** para /items
    - Captura del **email de SNS** por stock bajo
3. **Teardown** que elimine todos los recursos creados.

Restricciones y reglas

- **Solo despliegue programático** para los recursos principales (puedes confirmar manualmente la URL del sitio o suscribirte a SNS si es necesario).

- Usar **Python 3.11+** para los runtimes de Lambda; fija versiones de dependencias.
- IAM de **mínimo privilegio**; evita permisos excesivos o * innecesarios.
- **Idempotencia** en scripts de setup (re-ejecutar no debe fallar).
- Habilitar **CORS** en API Gateway para que la web en S3 pueda hacer fetch

Guide to practical labs to execute

- Guided Lab: Challenge (Cafe) lab: Creating a Static Website for the Café: https://awsacademy.instructure.com/courses/136876/assignments/1582079?module_item_id=13168195

You must edit the bucket policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::cafe-website-deusto/*"
    }
  ]
}
```

- Guided lab: Creating a Highly Available Environment: https://awsacademy.instructure.com/courses/136876/assignments/1582117?module_item_id=13168481

- Guided Lab: Implementing a Serverless Architecture on AWS: https://awsacademy.instructure.com/courses/136876/assignments/1582135?module_item_id=13168630

Objetivo general

Desplegar una aplicación web **FastAPI TODO** en la infraestructura de AWS utilizando servicios completamente administrados.

El estudiante deberá empaquetar la aplicación en un contenedor Docker, publicarlo en **Amazon ECR**, ejecutar los contenedores en **AWS Fargate** mediante **Amazon ECS**, y automatizar toda la infraestructura con **AWS CloudFormation**.

Descripción del escenario

La empresa ficticia **CloudTasks** necesita una aplicación web para gestionar tareas (TODOs).
Esta aplicación se ha desarrollado en **FastAPI** y utiliza **MySQL** como base de datos.
El equipo de DevOps quiere que la aplicación se despliegue sin preocuparse por servidores, de forma repetible y totalmente gestionada.

Tu misión es preparar e implementar esta solución siguiendo las prácticas de infraestructura como código.

Requerimientos técnicos

1. **Amazon ECR (Elastic Container Registry)**
    - Crea un repositorio privado en Amazon ECR para almacenar la imagen Docker del servicio FastAPI.
    - Explicación: *Amazon ECR es un servicio de registro de contenedores totalmente administrado que permite almacenar, gestionar y desplegar imágenes Docker de manera segura*.
2. **AWS ECS (Elastic Container Service) con Fargate**
    - Crea una definición de tarea con dos contenedores:
        - app: aplicación FastAPI (imagen desde ECR).
        - db: base de datos MySQL (imagen pública mysql:8).
    - Define las variables de entorno necesarias (DB_HOST, DB_USER, DB_PASSWORD, DB_NAME).
    - Explicación: *AWS Fargate permite ejecutar contenedores sin tener que aprovisionar ni administrar servidores, asignando automáticamente los recursos necesarios*.
3. **Application Load Balancer (ALB)**
    - Implementa un balanceador de carga para exponer el endpoint HTTP de la aplicación (puerto 80).

- El ALB deberá distribuir el tráfico a las tareas de ECS y verificar la salud del servicio en la ruta /docs.

4. **AWS CloudFormation**
    - Automatiza la creación de toda la infraestructura anterior mediante una plantilla YAML o JSON.
    - Incluye los siguientes recursos:
        - Cluster ECS
        - Task Definition (con 2 contenedores)
        - Service ECS (con ALB y Target Group)
        - Roles e IAM necesarios
    - Explicación: *AWS CloudFormation permite describir y desplegar todos los recursos de AWS mediante código, garantizando consistencia, trazabilidad y fácil recreación del entorno*.
5. **Validación del despliegue**
    - Una vez implementado el stack, abre el DNS público del Load Balancer en el navegador y verifica que la interfaz de Swagger de FastAPI aparece en la ruta /docs.

Entregables

- Archivo Dockerfile funcional.
- Imagen subida correctamente a ECR.
- Archivo de plantilla fastapi-todo.yaml (CloudFormation).
- Evidencia del despliegue exitoso:
    - Captura del servicio ECS en estado “RUNNING”.
    - Captura del acceso a http://<ALB-DNS>/docs.

Extra (opcional)

- Sustituir la base de datos MySQL en contenedor por un **Amazon RDS MySQL** persistente y actualizar el parámetro DB_HOST en la plantilla.
- Añadir el uso de **AWS Secrets Manager** para proteger las credenciales de base de datos.