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

Descripción general y objetivos

Debes automatizar 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 hacerse 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