Creación de un Pipeline de Datos y API con Caché Inteligente

Objetivo General

Desarrollar una solución integral de backend que demuestre la capacidad de migrar datos a gran escala, exponerlos a través de una API segura y optimizada, y desplegarla en un entorno de nube productivo. El proyecto culmina con la implementación de un sistema de monitoreo y una estrategia de caché con invalidación automática.

Fase 1: Preparación y Migración de Datos

El primer paso es construir la base de datos en la nube que alimentará nuestra aplicación.

1. **Selección del Dataset:**
    - Explora la plataforma **Kaggle** y selecciona un dataset que contenga una cantidad considerable de registros (se recomiendan más de 10,000). El tema es libre, pero busca datos que puedan ser catalogados o filtrados fácilmente.
2. **Migración de Datos con Azure Data Factory:**
    - Utiliza **Azure Data Factory** para crear un pipeline que extraiga los datos del dataset seleccionado (previamente subido a un Azure Blob Storage como un archivo .csv).
    - Transforma y carga (ETL) estos datos en una tabla específica dentro de una base de datos provisionada en **Azure** (puede ser **Azure SQL** o **Azure Database for PostgreSQL**, como se vio en los videos).

Fase 2: Desarrollo de API y Autenticación

Con los datos en la nube, crearás una API para interactuar con ellos, aplicando buenas prácticas de desarrollo y seguridad.

1. **Tecnología:**
    - Se recomienda desarrollar la API utilizando **FastAPI** (Python). Sin embargo, tienes la libertad de usar otro lenguaje o framework con el que te sientas más cómodo.
2. **Autenticación y Autorización con Firebase:**
    - Implementa **Firebase Authentication** para gestionar el registro e inicio de sesión de usuarios.
    - La API debe estar protegida, y el acceso a ciertos endpoints dependerá de un token JWT válido generado por Firebase.
    - La lógica de autorización es flexible, pero se sugiere replicar el modelo visto en clase:
        - En tu tabla de `Users` en la base de datos, incluye campos booleanos como `is_active` y `is_admin` para determinar los permisos de un usuario.
3. **Endpoints Requeridos:**
    - `POST /signup`: Para el registro de nuevos usuarios en Firebase y en tu base de datos.
    - `POST /login`: Para autenticar a un usuario y obtener su token de acceso.
    - `GET /catalog`: Este endpoint tendrá un comportamiento dual:
        - Si se llama **sin query strings**, debe retornar todos los registros del catálogo.
        - Si se llama **con un query string** (ej: `/catalog?category=A`), debe retornar solo la sección de datos que coincida con el filtro.
        - El diseño exacto del catálogo y si requiere autenticación de usuario o de administrador queda a tu discreción.

Fase 3: Monitoreo de Rendimiento

Para entender cómo se comporta tu API bajo carga, la integrarás con los servicios de monitoreo de Azure.

1. **Configuración de Application Insights:**
    - Integra tu API con **Azure Application Insights** para capturar telemetría, trazas de solicitudes, tiempos de respuesta y posibles errores.
2. **Pruebas de Carga:**
    - Realiza múltiples llamadas (requests) consecutivas a tu endpoint `GET /catalog`.
    - Analiza el panel de Application Insights para observar el comportamiento de la API: identifica los tiempos promedio de respuesta, el número de solicitudes por segundo y cualquier cuello de botella.

Fase 4: Implementación de Caché con Redis (El Reto)

Para mejorar drásticamente los tiempos de respuesta, implementarás una capa de caché.

1. **Integración con Redis:**
    - Configura y conecta una base de datos de **Redis** a tu API.
2. **Persistencia en Caché y Reto de Llaves Dinámicas:**
    - Modifica el endpoint `GET /catalog` para que, antes de consultar la base de datos, primero verifique si la respuesta ya existe en Redis.
    - La **llave (key)** utilizada para guardar cada respuesta en Redis debe generarse dinámicamente a partir de los **query strings** de la petición. Por ejemplo, una petición a `/catalog?category=A` generará una llave como `catalog:category=A`.
    - Si la llave existe en Redis, la respuesta se devuelve inmediatamente (cache hit).
    - Si no existe, se consulta la base de datos, se almacena la respuesta en Redis usando la llave dinámica y luego se devuelve al usuario (cache miss).

Fase 5: Invalidación de Caché

El caché debe ser inteligente. Si los datos originales cambian, el caché obsoleto debe ser eliminado.

1. **Endpoint de Creación:**
    - Implementa un endpoint `POST` (ej: `/catalog`) que permita agregar un nuevo registro a la tabla principal. Este endpoint debe estar protegido (por ejemplo, solo para administradores).
2. **Lógica de Invalidación:**
    - Después de agregar el nuevo registro a la base de datos, el sistema debe identificar a qué categoría o filtro pertenece este nuevo dato.
    - A continuación, debes **eliminar la llave específica en Redis** que corresponde a esa categoría.
3. **Verificación del Flujo:**
    - **Paso 1:** Realiza una petición `GET` con un query string (ej: `?category=C`). La primera vez será lenta, la segunda será rápida.
    - **Paso 2:** Usa el endpoint `POST` para agregar un nuevo ítem que pertenezca a la `category=C`. La lógica de invalidación deberá borrar la llave `catalog:category=C` de Redis.
    - **Paso 3:** Vuelve a realizar la misma petición `GET` del Paso 1. Esta vez, **deberá ser lenta de nuevo**, ya que el caché fue invalidado.

Fase 6: Despliegue en la Nube con Docker 🐋

Finalmente, toda la aplicación debe ser empaquetada y desplegada para funcionar en un entorno de nube real.

1. **Containerización:**
    - Crea un `Dockerfile` para tu aplicación de API, asegurándote de que incluya todas las dependencias y configuraciones necesarias para ejecutarse de forma aislada.
2. **Release en la Nube:**

- Publica la imagen de Docker en un registro de contenedores (como Azure Container Registry o Docker Hub).
- Despliega el contenedor en un servicio de Azure como **Azure App Service** o **Azure Container Apps** para que la API sea accesible públicamente.

Entregables

A continuación se detallan los elementos que deberás entregar como parte de la evaluación final del proyecto. Asegúrate de que cada uno cumpla con los criterios establecidos para demostrar tu comprensión y aplicación de los conceptos aprendidos durante el curso.

1. Entrada en el portafolio

- Explicación detallada y completa del proyecto, abarcando desde la implementación inicial del proceso ETL, pasando por el desarrollo de la API con autenticación, hasta las pruebas de rendimiento realizadas y documentadas en Application Insights.
- Incluye diagramas explicativos del flujo de datos y la arquitectura implementada.
- Links a todos los repositorios utilizados, asegurándote de que el código esté debidamente documentado con comentarios claros y un README que explique la estructura del proyecto y las instrucciones para su ejecución local.

2. Presentación

- Una presentación completa de PowerPoint con capturas de pantalla y evidencias detalladas del despliegue de todos los recursos en Azure.
- Documentación visual de cómo se ejecutó el pipeline de datos, métricas de rendimiento y los tiempos registrados en App Insights antes y después de implementar la estrategia de caché.

3. Acceso a la API

- Links a los endpoints productivos desplegados en la nube.
- Colección de Postman dentro del repositorio que contenga ejemplos de todas las peticiones posibles, que pueda ser descargada e importada fácilmente por cualquier evaluador para probar la funcionalidad del sistema.

4. Demostración en vivo

- Estar completamente preparado para realizar una demostración en tiempo real de tu proyecto, mostrando todas las funcionalidades implementadas y explicando las decisiones técnicas tomadas.
- Si por algún motivo no puedes traer tu computadora personal, es fundamental que coordines con tu profesor con suficiente anticipación para poder presentar desde su máquina sin contratiempos.

Rúbrica de evaluación:

Rúbrica de Evaluación

La siguiente rúbrica detalla los criterios que se utilizarán para evaluar tu proyecto final. Cada componente tiene un peso específico en la calificación total, reflejando la importancia de las diferentes habilidades y conocimientos demostrados.

Criterio	Observation
Migración de Datos (20%)	Pipeline ETL básico que migra datos a Azure SQL/PostgreSQL correctamente
API y Autenticación (25%)	API funcional con endpoints básicos y autenticación Firebase implementada
Monitoreo (15%)	Integración básica con Application Insights
Caché con Redis (25%)	Implementación básica de caché para consultas simples
Despliegue (15%)	Aplicación desplegada en la nube con Docker