Arquitectura del Proyecto de Procesamiento de Datos de Rick and Morty

Descripción General de la Arquitectura

Visión Global

Este proyecto simula un entorno de Big Data en la nube para procesar información de la API de Rick and Morty, siguiendo un flujo de trabajo ETL (Extract, Transform, Load) con tres fases principales:

- Ingesta**:** Extracción de datos de la API y almacenamiento inicial en SQLite
- Preprocesamiento: Limpieza y normalización de los datos
- Enriquecimiento: Integración con datos adicionales (scripts de episodios)

Este pipeline se ejecuta de manera automatizada mediante GitHub Actions, asegurando la eficiencia en la ingesta y transformación de datos. En un entorno en la nube real, se utilizaría Databricks para la escalabilidad y procesamiento distribuido.

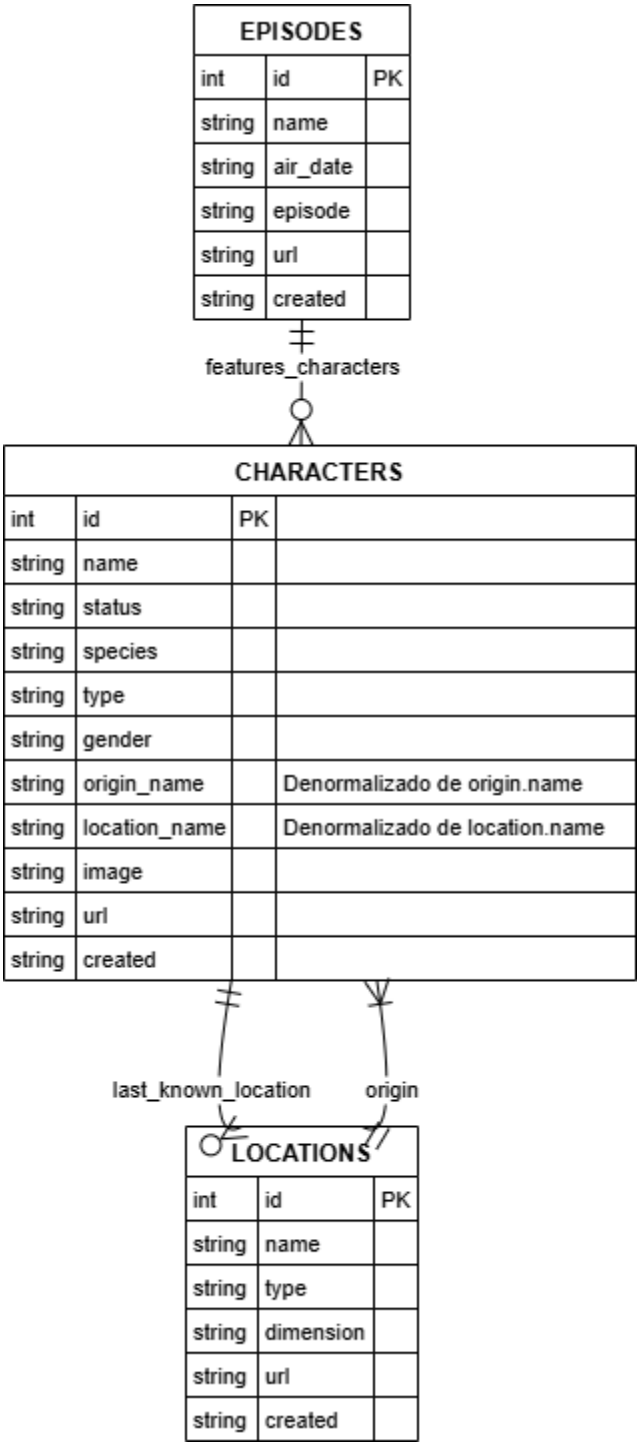
Componentes Principales

Base de datos analítica (SQLite): SQLite almacena los datos extraídos y transformados. Contiene tablas para personajes, ubicaciones y episodios, proporcionando una estructura relacional adecuada para el análisis.

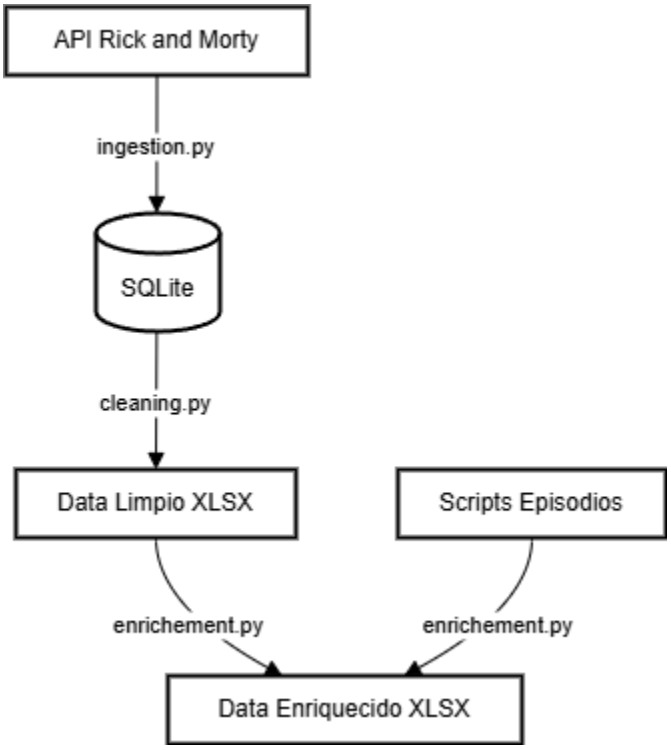
- **Scripts de procesamiento:**
    - Ingesta (ingestion.py): Obtiene datos desde la API, los almacena en SQLite y genera archivos de auditoría.
    - Preprocesamiento (cleaning.py): Normaliza y limpia los datos para eliminar duplicados y valores inconsistentes.
    - Enriquecimiento (enrichement.py): Combina los datos limpios con información externa para mejorar el análisis.
- **Mecanismo de automatización:** Se emplea GitHub Actions para ejecutar el pipeline de manera programada, garantizando la actualización constante del dataset.

Diagramas de Arquitectura

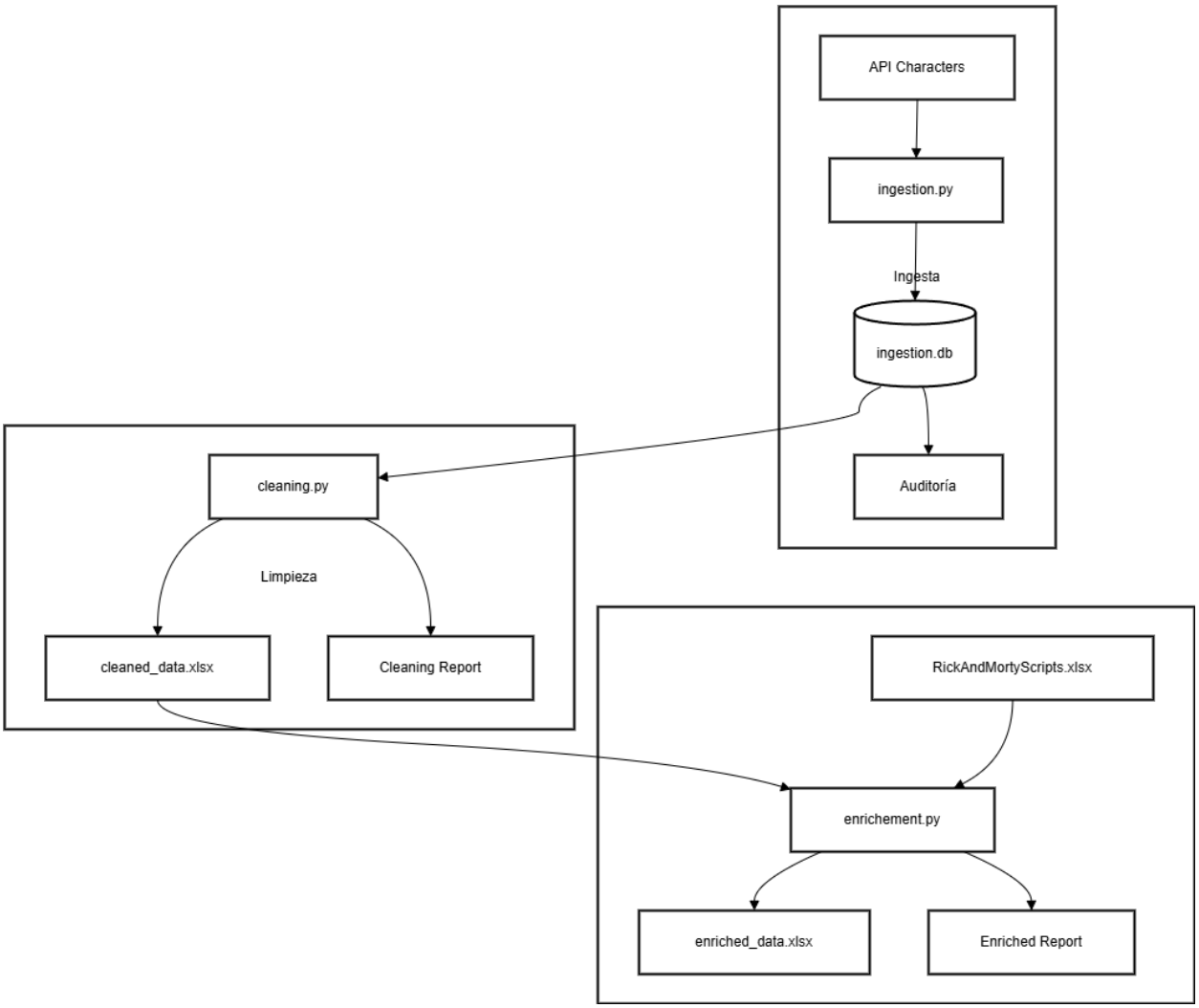
Entidad-relación (ER)



Flujo General de Datos



Proceso Detallado



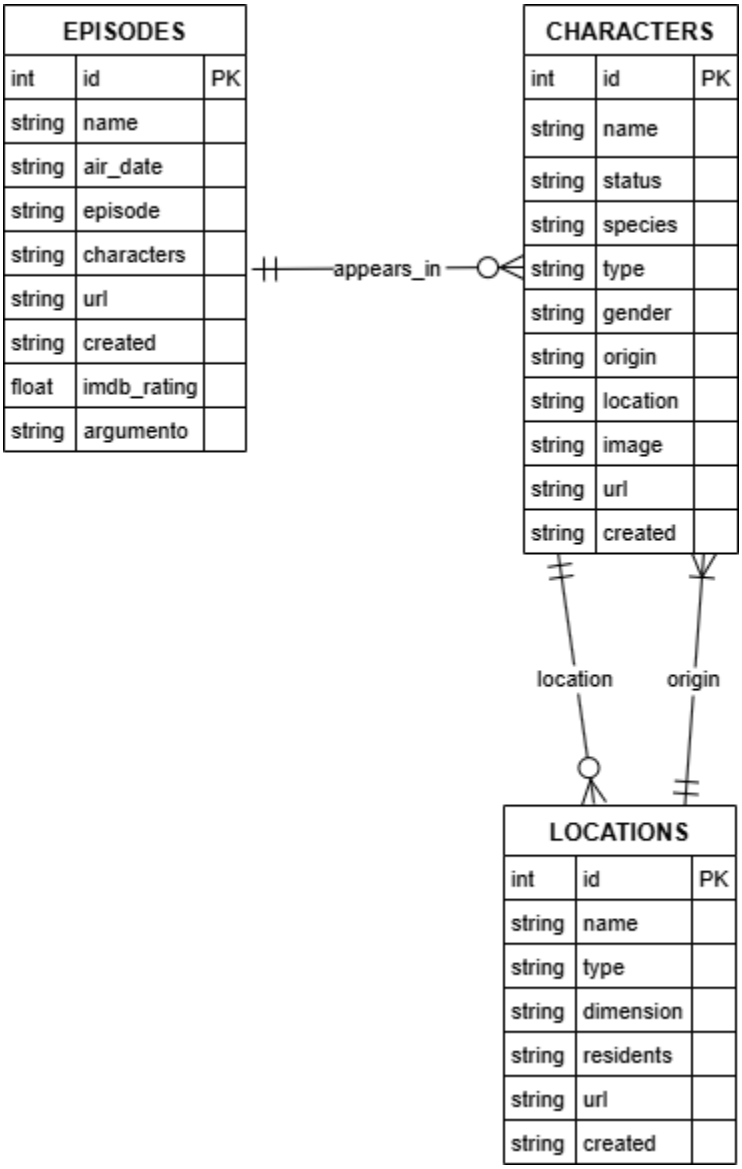
Modelo de Datos

Esquema Resultante

El modelo final consiste en tres tablas relacionadas:

- characters:
    - Campos: id (PK), name, status, species, type, gender, origin, location, image, url, created.
    - Relaciones: Aparece en múltiples episodios (relación muchos-a-muchos implícita)
- locations:
    - Campos: id (PK), name, type, dimension, residents, url, created.
    - Relaciones: Un personaje tiene una ubicación (relación uno-a-muchos).
- episodes (enriquecida con datos de scripts):
    - Campos originales: id (PK), name, air_date, episode, characters, url, created.
    - Campos añadidos: argumento, imdb rating (del dataset de scripts)

Diagrama de Datos



Justificación del Modelo

El modelo diseñado:

- Preserva las relaciones naturales de la API (personajes-ubicaciones-episodios)
- Mantiene la estructura original mientras añade campos enriquecidos
- Facilita consultas analíticas como:
    - Personajes por ubicación.
    - Episodios mejor valorados.
    - Relación entre escritores/directores y rating.

Justificación de Herramientas y Tecnologías

- SQLite:
    - Ideal para prototipado rápido.
    - Requiere mínima configuración.
- Pandas:
    - Procesamiento eficiente en memoria.
    - Amplio soporte para transformaciones de datos.
    - Buen equilibrio entre rendimiento y facilidad de uso
- PySpark (no usado actualmente):
    - Escalaría el procesamiento para grandes volúmenes.
    - Permitiría distribuir el trabajo en clusters.
- GitHub Actions:
    - Automatizaría la ejecución periódica del pipeline.
    - Facilitaría el despliegue continuo

Simulación del Entorno Cloud

El proyecto actual simula un entorno cloud mediante:

- Base de datos centralizada (ingestion.db).
- Procesamiento por etapas con salidas intermedias.
- Auditoría y reporting automatizado.
- Estructura de directorios que imita buckets cloud (src/static, src/xlsx)

Flujo de Datos y Automatización

- Ingesta:
    - Extracción de datos de la API.
    - Almacenamiento en SQLite.
    - Generación de informe de auditoría
- Limpieza:
    - Eliminación de duplicados
    - Tratamiento de valores nulos
    - Normalización de tipos de datos
    - Generación de reporte detallado
- Enriquecimiento:
    - Unión con dataset de scripts
    - Validación de coincidencias
    - Reporte de métricas de enriquecimiento

Beneficios:

- Arquitectura modular fácil de mantener
- Documentación automática mediante reports
- Procesamiento reproducible

Limitaciones:

- SQLite no escala para grandes volúmenes
- Falta de procesamiento incremental
- No hay manejo de errores robusto

Recomendaciones para la implementación en un entorno real de nube.

- Migración a Delta Lake:
  Almacenar los datos en Delta Lake (formato abierto basado en Parquet con ACID transactions).
  Ventajas:
  - Soporta lecturas/escrituras concurrentes.
  - Time Travel (histórico de versiones).
  - Optimización automática con Z-Order y bin-packing.

- Pipeline con Databricks Workflows (Jobs)
  Automatizar el flujo ETL con Jobs de Databricks:
  - Notebook 1**:** Ingesta desde API → Delta Lake.
  - Notebook 2**:** Limpieza y transformación (Spark SQL/PySpark).
  - Notebook 3**:** Enriquecimiento con JOINs optimizados.

- Optimización con Spark SQL y Delta Lake
  - Usar PySpark para procesamiento distribuido:
  - Particionar tablas por campos frecuentemente filtrados (species, status).

- Aplicar Delta Lake OPTIMIZE para mejorar el rendimiento.

- Monitoreo con Databricks Lakehouse Monitoring.

  Monitorear calidad de datos en tiempo real:

  - Alertas automáticas (ej: NULL en campos críticos como character_id).
  - Dashboard de métricas (ej: % de valores "unknown" en status).

- Integración con Unity Catalog (Governanza de Datos).

  Centralizar metadatos en Unity Catalog:

  - Rastrear el origen de los datos (API vs. scripts externos).
  - Definir permisos por equipo (ej: analysts solo lectura).

- Escalabilidad con AutoLoader (Streaming)**.**

  Usar AutoLoader para ingestas incrementales:

  - Ideal si la API expone nuevos episodios/personajes frecuentemente.

Conclusiones y Recomendaciones

El modelo propuesto permite estructurar los datos de manera eficiente para su análisis. Se recomienda migrar a una base de datos distribuida (como Delta Lake en Databricks) para escalar el procesamiento. Además, la implementación de PySpark mejoraría la velocidad y eficiencia en el tratamiento de grandes volúmenes de datos.