Procesos ETL

Table of contents

1 Capas de datos

- `data/raw/`: fuentes originales (CSV, XLSX, JSON, API dumps).
- `data/bronze/`: normalización mínima, tipos corregidos.
- `data/silver/`: tablas integradas con referencias (`data/ref/`).
- `data/gold/`: modelos listos para consumo analítico / API.
- `data/meta/`: diccionarios, linaje, contratapas, definiciones.
- `data/ref/`: catálogos maestros (dimensiones, dominios).

2 Pipelines

Los pipelines viven en `etl/pipelines/<tema>/<dominio>/`. Cada pipeline expone:

2.1 Arquitectura modular

- Cada paquete sigue la convención `extract.py`, `transform.py`, `load.py` (o submódulos equivalentes) más `settings.py` y `pipeline.py` como orquestador.
- Las dependencias compartidas residen en `etl/metadata/`, `etl/metadata_store.py` y `etl/core/` (abstracciones de `Pipeline`, `Task`, `Dataset`).

- Los pipelines mantienen archivos 400 líneas; helpers extensos se ubican en submódulos dedicados.
- Las pruebas viven en `tests/unit/` (parsers, procesadores) y `tests/data/` (integración con fixtures sintéticas).

```python
from etl.pipelines.base import Pipeline

class EncuestaEnuscBronze(Pipeline):
    source = "data/raw/enusc/*.csv"
    target_table = "bronze.enusc_respuestas"

    def extract(self):
        ...

    def transform(self, df):
        ...

    def load(self, df):
        ...
```

3 Pruebas de datos

- `tests/unit/` contiene suites específicas para loaders/procesadores (por ejemplo `tests/unit/test_gendarme` y `tests/unit/fiscalia_persecucion_penal/*`).
- `tests/data/` agrupa pruebas de integración que ejecutan pipelines sobre fixtures empaquetadas (`tests/data/test_dipres_bronze_pipeline.py`, `tests/data/test_ine_estadisticas_judiciale` etc.).
- `data/tests/` (pendiente) recogerá validaciones de integridad y constraints SQL directamente sobre la base de datos.

4 Semillas mínimas

`etl/seeds/minimum.py` crea dimensiones base (ej. `dim_fecha`, `dim_region`). Se ejecuta automáticamente en cada `make seed_min`.

5 Pipelines disponibles

- `etl.pipelines.dipres_budget.DipresBudgetBronze`: normaliza los binarios DIPRES seleccionando el formato preferente (CSV > XLS/XLSX > XML/HTML) y genera los CSV de la capa bronze (`data/bronze/dipres_presupuestos_totales.csv` y `data/bronze/dipres_presupuestos_nodos.csv`).
  - Convierte automáticamente los PDF en texto (`*.md`) para referencia documental.

  - Reconoce archivos `.xls` que en realidad son HTML exportado desde DIPRES y los procesa con `pandas.read_html`.
  - Multiplica los montos publicados en miles por `1000`, aplica el tipo de cambio promedio anual (`data/ref/dipres_exchange_rates.csv`) y normaliza los trimestres acumulados (`primer/segundo/tercer/cuarto`) a los meses equivalentes (`marzo`, `junio`, `septiembre`, `diciembre`).
  - Descarta la Partida 50 (Tesoro Público) para evitar doble conteo en los agregados nacionales.
  - Construye y persiste el glosario `data/meta/diccionarios/dipres_programas.csv` para enriquecer los nombres de partidas/capítulos/programas en todas las corridas.
  - Genera el puente COFOG (`data/meta/diccionarios/dipres_cofog_programas.csv`) a partir del Anexo 4 para habilitar agregaciones funcionales.
  - Registra archivos que no pudieron normalizarse (lista `skipped` en la ejecución del pipeline) y los deja etiquetados en el manifest.
  - Alimenta la API `/api/{version}/datasets/{slug}/raw-assets`, por lo que cada corrida deja disponibles los binarios con permalink descargable.
  - Registra la ejecución en `jobs_runs` y actualiza `meta_datasets/meta_files/meta_tables/meta_linea` mediante `etl.metadata_store.MetadataStore` (ver `tests/data/test_metadata_instrumentation.`
  - El script de carga `scripts/load_dipres_bronze_to_db.py` detecta los últimos periodos presentes en Postgres, inserta únicamente los meses nuevos en jerarquía y `presupuesto_ejecuciones_mensual`, y anota la corrida en `etl_log` con el periodo cubierto y su variant.

- `scripts/build_cofog_series.py`: agrega los nodos de nivel programa por función/subfunción COFOG, aplica deflactores IPC base 2018, suma totales de presupuesto y PIB real, y persiste las series anuales con montos nominales/reales y participaciones en `data/analytics/cofog_*_annual.csv`.
- `etl.pipelines.ine_estadisticas_judiciales.IneEstadisticasJudicialesPipeline`: recorre el manifest de INE estadísticas judiciales, procesa las 41 hojas (anuales y semestrales), genera archivos tidy por cuadro en `data/bronze/ine_estadisticas_judiciales/`, consolida los registros en `data/silver/ine_estadisticas_judiciales/records.parquet`, deriva automáticamente el segundo semestre, emulsiona chequeos de QA (totales, duplicados, valores negativos) y publica agregados en `data/gold/ine_estadisticas_judiciales/aggregated.parquet`.
- `etl.pipelines.enusc.ingest_enusc`: pipeline modular de ENUSC que:
  - Resume todo el procesamiento anual (lectura `.sav`, identificación de columnas, QA interanual) y delega la exportación en las clases `PersonHouseholdExporter` y `ModuleExporter`.
  - `PersonHouseholdExporter` construye `enusc_{year}_personas.parquet`, `enusc_{year}_hogares.par` devuelve `household_db` listo para cargar a bronze y recopila estadísticas (`hogares_registrados`, columnas retenidas).
  - `ModuleExporter` genera los parquet por módulo (`data/silver/enusc/modulos/<modulo>/<modulo>_{` limpia salidas anteriores si no se ejecuta en modo append y acumula métricas en `ModuleSummary`.
  - Las clases anteriores escriben en DB a través de `TableWriter`, por lo que el modo `ENUSC_DB_APPEND_ONLY=1` se respeta de manera consistente.
  - Se ejecutan validaciones ligeras con Pydantic antes de escribir `enusc_personas`, `enusc_hogares` y el resumen, permitiendo modo no estricto (descarta filas inválidas) o estrictamente (`SchemaValidator(strict=True)`).

  - El pipeline produce además `data/silver/enusc/reportes/enusc_personas_hogares_resumen.csv`, `data/silver/enusc/reportes/modulos_manifest.json`, indicadores regionales/nacionales (`data/gold/enusc/*.csv`) y el parquet interanual 2008-2024.

- `etl.pipelines.gendarmeria_reportes.GendarmeriaReportLoader`: orquesta los subsistemas Abierto, Cerrado y Postpenitenciario a través de `process_abierto`, `process_cerrado` y `process_postpenitenciario`; persiste CSV normalizados (`gendarmeria_abierto.csv`, `gendarmeria_cerrado_poblacion.csv`, `gendarmeria_postpenitenciario.csv`) y registra mediciones via mapas declarativos.
- `etl.pipelines.seguridad_fuentes.BronzeSeguridadFuentes`: integra estadísticas de DIPRES, Fiscalía, Gendarmería e INE reutilizando adaptadores por fuente (`etl/pipelines/seguridad_fuentes/*.py`) y utilidades territoriales comunes.

6 Series BCCh para deflactores / ratios

- Script: `scripts/fetch_bcch_series.py`
  - Consume el servicio SOAP `SieteWS` (`GetSeries`) con credenciales `BCCH_USER` / `BCCH_PASS` (definidas en `.env`).
  - Descarga dos series públicas:
    * `G073.IPC.IND.2018.M`: IPC general histórico (base 2018=100) → `data/ref/bcch/ipc_mensual.cs`
    * `F032.PIB.FLU.R.CLP.EP18.Z.Z.0.T`: PIB real trimestral encadenado, base 2018=100 (CLP de 2018) → `data/ref/bcch/pib_trimestral_real_2018.csv`.
  - Actualiza el catálogo `data/meta/catalog/ref_bcch.yaml` para documentar origen, esquema y términos de uso.

Ejecución manual:

```
BCCH_USER=... BCCH_PASS=... PYTHONPATH=. python scripts/fetch_bcch_series.py
```

Los CSV resultantes sirven como referencia para deflactar montos nominales y calcular porcentajes respecto del PIB en la capa analítica o dashboards.