Fondazione Gran Sasso Tech

WP2: Realizzazione Infrastruttura Data-Lake MAPS

Progettazione Tecnica, Solution Design Cloud, Specifiche Hosting

Data: 2026-02-20

Autore: Guglielmo Celata

1 Contesto e Requisiti

1.1 1.1 Introduzione al Progetto MAPS

Il progetto **MAPS** (Multidimensional Area Place-based System) si inserisce nel contesto dell'analisi territoriale italiana con l'obiettivo di definire **Sistemi Locali Omogenei (SLO)** basati su pattern di mobilità quotidiana multidimensionale.

1.1.1 Obiettivi del Progetto

1. **Mappatura sistemi locali**: Identificazione di aree funzionali basate su accessibilità ai servizi
2. **Individuazione aree svantaggiate**: Classificazione territori in base a indicatori di svantaggio
3. **Analisi investimenti pubblici**: Integrazione dati Open Coesione, PNRR, fondi regionali
4. **Valutazione impatto**: Simulazione scenari "what-if" per allocazione investimenti

1.1.2 Approccio Innovativo

A differenza dei tradizionali **Sistemi Locali del Lavoro (SLL)** ISTAT, che si basano principalmente sul pendolarismo lavorativo, MAPS adotta un approccio **multidimensionale** che considera:

- **Lavoro** (32% degli spostamenti)
- **Gestione familiare** (37.2% degli spostamenti)
- **Tempo libero** (26.2% degli spostamenti)
- **Istruzione** (4.6% degli spostamenti)
- **Accesso a servizi pubblici essenziali**

1.2 1.2 Requisiti del Data Lake

1.2.1 Requisiti Funzionali

1.2.1.1 RF1: Ingestion Dati Eterogenei

Il Data Lake deve supportare l'acquisizione di ~200 dataset da fonti eterogenee: - **ISTAT**: Popolazione, confini amministrativi, pendolarismo - **EUROSTAT**: Indicatori socio-economici europei - **Ministeri**: Istruzione, Salute, Lavoro - **GTFS**: Trasporto pubblico - **OpenStreetMap**: Infrastrutture territoriali - **Open Coesione**: Progetti finanziati - **PNRR**: Investimenti Recovery Fund

1.2.1.2 RF2: Gestione Serie Storiche

- **Periodo**: 2010-2025 (15 anni)
- **Granularità**: Comunale (~8.000 comuni italiani)
- **Discontinuità**: Gap temporali COVID-19 (2020-2021)
- **Fusioni comunali**: Gestione tramite lookup storico

1.2.1.3 RF3: Standardizzazione Dati

- Normalizzazione codici ISTAT
- Uniformazione coordinate spaziali (EPSG:32632)
- Gestione valori mancanti vs nulli semantici
- Riconciliazione fusioni/separazioni comunali

1.2.1.4 RF4: Pattern Medallion Implementazione del pattern di data lake a 3 livelli: - **Bronze**: Dati grezzi immutabili - **Silver**: Dati standardizzati (schema EAV) - **Gold**: Data mart analitici business-ready

1.2.2 Requisiti Non Funzionali

1.2.2.1 RNF1: Completeness

- **Target**: Copertura >=95% dei comuni italiani per ogni dataset
- **Metrica**: % comuni con dati completi vs totale

1.2.2.2 RNF2: Accuracy

- **Target**: >=99.9% record con codice ISTAT valido
- **Validazione**: Confronto con reference dataset (confini ISTAT ufficiali)

1.2.2.3 RNF3: Timeliness

- **Ingestion**: Dataset mensili processati entro 5 giorni dall'availability
- **Latenza ETL**: <24h per pipeline prioritarie

1.2.2.4 RNF4: Lineage

- **Tracciabilità**: Completa per ogni record (fonte → Bronze → Silver → Gold)
- **Versionamento**: Tracking modifiche su fonti dati

1.2.2.5 RNF5: Scalabilità

- **Volume**: Supporto fino a 1TB dati nel triennio
- **Throughput**: >=100 record/s per pipeline ETL

1.2.2.6 RNF6: Governance

- **Catalogazione**: Metadata per tutti i dataset
- **Quality monitoring**: Dashboard data quality real-time
- **Audit**: Log completo operazioni su dati sensibili

1.3 1.3 Dati Critici per MVP (Fase 1a)

Per la fase iniziale del progetto (Mesi 1-4), si identificano **5 dataset critici**:

Dataset	Fonte	Formato	Frequenza	Priorità
Confini comunali + metadata	ISTAT	Shapefile/JSON	Annuale	Alta
Popolazione residente	ISTAT	CSV	Annuale	Alta
Matrici pendolarismo	ISTAT	CSV	Decennale	Alta
Rete trasporto pubblico	GTFS	ZIP	Mensile	Media

Dataset	Fonte	Formato	Frequenza	Priorità
Strutture sanitarie	Min. Salute	Excel	Annuale	Media

1.4 1.4 Vincoli Architetturali

1.4.1 Vincolo V1: Self-Hosted Open Source

- **Rationale**: Indipendenza da vendor, costi prevedibili
- **Stack tecnologico**: PostgreSQL, PostGIS, Prefect, OpenMetadata, DuckDB
- **Esclusioni**: BigQuery, Azure, AWS managed services

1.4.2 Vincolo V2: Adeguatezza di Scala

- **Volumetria**: ~10 righe x ~10^3 colonne (ordine di grandezza DEPP)
- **Giustificazione**: BigQuery sarebbe overkill per questa scala

1.4.3 Vincolo V3: Standard Territoriali

- **PostGIS**: Industry standard per dati spaziali
- **EPSG:32632**: Sistema di riferimento WGS84 / UTM zone 32N

1.4.4 Vincolo V4: Compliance FAIR

Dataset rilasciati devono essere: - **Findable**: Catalogati con metadata strutturati - **Accessible**: Licenze open (CC-BY, CC0) - **Interoperable**: Formati standard (GeoJSON, GeoParquet, CSV) - **Reusable**: DOI per citabilità scientifica

1.5 1.5 Sfide Specifiche

1.5.1 Sfida S1: Assenza Geolocalizzazione Puntuale

La maggior parte dei servizi (scuole, ospedali, uffici postali) non ha coordinate precise: - **Approccio**: Presenza/assenza a livello comunale - **Implicazione**: Isocrone municipality-to-municipality

1.5.2 Sfida S2: Evoluzione Confini Amministrativi

Nel periodo 2010-2025 ci sono state ~100 fusioni/separazioni comunali: - **Soluzione**: Lookup storico con validità temporale (SCD Type 2) - **Schema Silver**: Colonne `valid_from`, `valid_to`

1.5.3 Sfida S3: Gap Temporali COVID

Dati 2020-2021 hanno discontinuità metodologiche: - **Gestione**: Flag `covid_affected` per dataset impattati - **Analisi**: Tecniche di imputation o esclusione periodi

1.5.4 Sfida S4: Eterogeneità Formati

207 dataset con formati diversi (CSV, XLSX, HTML, PDF): - **Stack estrazione**: Docling (PDF), Pandas (CSV/Excel), BeautifulSoup (HTML) - **Normalizzazione**: Schema EAV flessibile in Silver layer

1.6 1.6 Obiettivi del WP2

Al termine del WP2, il Data Lake dovrà:

1. [YES] **Ingestire 150+ dataset** da fonti eterogenee
2. [YES] **Coprire >=95% comuni italiani** per dataset prioritari
3. [YES] **Fornire tracciabilità completa** (lineage Bronze→Silver→Gold)
4. [YES] **Validare pipeline ETL** con report data quality
5. [YES] **Preparare infrastruttura** per algoritmi SLO (WP3)

1.7 1.7 Output Attesi (Deliverable)

- **D2.1.1**: Documento Progettazione Tecnica Data-Lake
- **D2.1.2**: Solution Design Architettura Cloud
- **D2.1.3**: Specifiche Infrastruttura Hosting
- **D2.2**: Script ETL (Python/SQL) documentati e testati
- **D2.3**: Report di Validazione Pipeline ETL

Prossimo capitolo: Architettura Tecnica Data-Lake

Mermaid Diagram

Figure 1: Mermaid Diagram

2 Architettura Tecnica Data-Lake

Deliverable D2.1.1: Documento Progettazione Tecnica Data-Lake

2.1 2.1 Pattern Medallion: Bronze → Silver → Gold

2.1.1 2.1.1 Visione d'Insieme

L'architettura del Data Lake MAPS adotta il **pattern Medallion** (anche denominato Multi-Hop Architecture), best practice nell'ecosistema Modern Data Stack, che organizza i dati in **tre layer di progressivo raffinamento** con crescente livello di qualità, pulizia e business-readiness:

```
FONTI ESTERNE → BRONZE → SILVER → GOLD → APPLICAZIONI
```

2.1.2 2.1.2 Rationale della Scelta

Per il progetto MAPS, l'architettura Medallion è particolarmente indicata per le seguenti motivazioni:

A. Eterogeneità delle Fonti Dati - ~200 dataset da fonti pubbliche diverse (ISTAT, Ministeri, OpenData) - Formati multipli: CSV, Excel, PDF, JSON, HTML - Qualità variabile: da dataset strutturati a documenti semi-strutturati - **Necessità**: Layer progressivi per standardizzare gradualmente l'eterogeneità

B. Requisiti di Audit e Compliance - Dati pubblici ma necessità di tracciabilità per ricerca scientifica - GDPR Article 30: documentazione delle attività di trattamento dati - **Necessità**: Bronze layer immutabile come "source of truth" originale

C. Complessità delle Trasformazioni - Pipeline multi-step: parsing PDF → validazione → normalizzazione EAV → aggregazioni territoriali - Time-series con fusioni/scissioni comunali (2010-2025) - **Necessità**: Separazione logica tra raw ingestion, cleaning e business logic

D. Riutilizzo dei Dati - Stesso dataset usato per multiple analisi (demografia, servizi, mobilità) - Necessità di evitare re-processing da fonte esterna ogni volta - **Necessità**: Silver layer come cache enterprise validata

E. Performance e Scalabilità - Query analitiche su 8.000+ comuni con decine di attributi - Spatial operations PostGIS computazionalmente intensive - **Necessità**: Gold layer denormalizzato per fast queries

2.1.3 2.1.3 Data Flow Completo

2.1.4 2.1.4 Bronze Layer: Raw Data Archive

Ruolo: Archivio immutabile dei dati originali esattamente come scaricati dalla fonte.

Principio chiave: "Never modify, always preserve"

2.1.4.1 Caratteristiche Tecniche

Aspetto	Specifica MAPS
Storage	File system locale `/data/bronze/` (server op-linkurious)
Formato	File originali senza trasformazioni (CSV, XLSX, PDF, JSON, HTML)
Naming convention	`/data/bronze/{fonte}/{anno}/{dataset}_{timestamp}.{ext}`
Retention policy	Indefinita (storage cost è basso: ~50GB totali per 200 dataset)
Backup	Snapshot giornalieri via `backup.sh` script
Access pattern	Write-once, read-rarely (solo per reprocessing o audit)

2.1.4.2 Struttura Directory

```
/data/bronze/
+-- istat/
|   +-- 2024/
|   |   +-- popolazione_comuni_20240218.csv
|   |   +-- pendolarismo_matrix_20240218.csv
|   |   +-- confini_amministrativi_20240218.geojson
|   |   +-- _metadata/
|   |       +-- popolazione_comuni_20240218.json    # Metadata file
|   |       +-- checksums.sha256
|   +-- 2023/
|       +-- popolazione_comuni_20230315.csv
+-- minlavoro/
|   +-- 2024/
|       +-- tabacchi_adm_report_20240218.pdf
|       +-- _metadata/
|           +-- tabacchi_adm_report_20240218.json
+-- minsalute/
|   +-- 2023/
|       +-- strutture_sanitarie_asl_20231120.xlsx
|       +-- _metadata/
|           +-- strutture_sanitarie_asl_20231120.json
+-- minambiente/
    +-- 2024/
        +-- ato_gas_page_20240115.html          # ← HTML file
        +-- _metadata/
            +-- ato_gas_page_20240115.json
```

2.1.4.3 Metadata Tracking

Ogni file Bronze ha un corrispondente file JSON con metadata:

Esempio: `/data/bronze/istat/2024/_metadata/popolazione_comuni_20240218.json`

```
{
  "file_path": "/data/bronze/istat/2024/popolazione_comuni_20240218.csv",
  "fonte": "istat",
```

```
  "dataset": "popolazione",
  "anno_riferimento": 2024,
  "download_timestamp": "2024-02-18T03:00:15Z",
  "download_url":
   ↪  "https://www.istat.it/storage/cartografia/popolazione_comuni_2024.csv",
  "file_size_bytes": 3355482,
  "file_hash_sha256":
   ↪  "a3f5b8c9d2e1f4a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f9a0",
  "mime_type": "text/csv",
  "encoding": "UTF-8",
  "rows_detected": 7901,
  "columns_detected": 12,
  "prefect_flow_run_id": "abc123-456-def-789",
  "ingestion_status": "completed"
}
```

Database tracking (PostgreSQL):

```
-- Schema: bronze
CREATE TABLE bronze.ingestion_log (
    id BIGSERIAL PRIMARY KEY,
    fonte VARCHAR(50) NOT NULL,
    dataset VARCHAR(100) NOT NULL,
    anno_rif INTEGER NOT NULL,
    file_path TEXT NOT NULL,
    file_size_bytes BIGINT,
    file_hash_sha256 CHAR(64),
    download_url TEXT,
    download_timestamp TIMESTAMP NOT NULL,
    prefect_flow_run_id UUID,
    status VARCHAR(20) NOT NULL,  -- 'completed', 'failed', 'in_progress'
    error_message TEXT,
    created_at TIMESTAMP DEFAULT NOW(),

    UNIQUE (fonte, dataset, anno_rif, file_hash_sha256)
);

-- Index per query frequenti
CREATE INDEX idx_ingestion_log_fonte_dataset ON bronze.ingestion_log(fonte,
 ↪  dataset);
CREATE INDEX idx_ingestion_log_status ON bronze.ingestion_log(status);
CREATE INDEX idx_ingestion_log_timestamp ON
 ↪  bronze.ingestion_log(download_timestamp DESC);
```

2.1.4.4 Garanzie Bronze Layer

Immutabilità: - File Bronze **non vengono mai modificati** dopo scrittura - Re-download stesso dataset → nuovo file con timestamp diverso - History completa: tutti i download conservati

Idempotenza: - Re-esecuzione flow → skip se file con stesso hash già presente - Deduplication basata su `(fonte, dataset, anno_rif, file_hash)`

Mermaid Diagram

Figure 2: Mermaid Diagram

Mermaid Diagram

Figure 3: Mermaid Diagram

Disaster Recovery: - Bronze è la "golden copy" per rebuild completo - Se Silver/Gold si corrompono → reprocess da Bronze - Backup script: `bash /root/maps-docker/backup.sh`

2.1.4.5 Caso d'Uso: HTML → Structured Data **Scenario reale**: Molte fonti pubbliche italiane pubblicano dati come **tabelle HTML embedded in pagine web** invece di file CSV scaricabili.

Esempio concreto: MinAmbiente pubblica elenco ATO Gas come tabella HTML su pagina web (no CSV/Excel disponibile).

Flusso Bronze-HTML:

Vantaggi Pattern Bronze-HTML:

```
Scenario senza Bronze layer:
  → Script scarica HTML e parsa immediatamente
  → Se parsing fallisce (HTML structure changed) → dati persi
  → Se server web va offline → impossibile reprocessare

Scenario con Bronze layer:
  → HTML salvato in Bronze (immutabile)
  → Parsing fallisce? → Fix parser, reprocess da Bronze
  → Server offline? → Bronze ha copia originale
  → HTML structure cambia? → Version history in Bronze
```

2.1.5 2.1.5 Silver Layer: Cleaned & Validated Data

Ruolo: Single Source of Truth (SSOT) enterprise - dati puliti, validati, normalizzati.

Principio chiave: "Trust but verify, then store"

2.1.5.1 Caratteristiche Tecniche

Aspetto	Specifica MAPS
Storage	PostgreSQL schema `silver`
Formato	Tabelle relazionali normalizzate (EAV schema)
Data model	Entity-Attribute-Value per gestire eterogeneità
Retention	Temporal versioning (valid_from, valid_to) per time-series
Backup	Snapshot giornalieri PostgreSQL + WAL archiving
Access pattern	Read-heavy (queries analitiche), write-moderate (batch ingestion)

2.1.5.2 Schema EAV (Entity-Attribute-Value) **Rationale**: Il progetto MAPS gestisce ~200 dataset con attributi eterogenei (popolazione, servizi, infrastrutture). Un modello EAV offre **flessibilità** senza continue schema migrations.

Schema principale:

```
-- Schema: silver
CREATE TABLE silver.comuni_attributi_eav (
    -- Primary key
    id BIGSERIAL PRIMARY KEY,

    -- Entity: Comune
    codice_istat VARCHAR(6) NOT NULL REFERENCES
    ↪ gold.comuni_anagrafica(codice_istat),

    -- Attribute: Nome attributo
    attributo VARCHAR(255) NOT NULL,

    -- Value: Valore come testo (type inference in Gold)
    valore TEXT NOT NULL,

    -- Metadata: Provenienza dati
    fonte VARCHAR(50) NOT NULL,         -- 'istat', 'minlavoro', 'minsalute', etc.
    dataset VARCHAR(100) NOT NULL,      -- 'popolazione', 'tabacchi', 'asl', etc.
    anno_rif INTEGER NOT NULL,

    -- Temporal validity
    valid_from DATE NOT NULL DEFAULT CURRENT_DATE,
    valid_to DATE,

    -- Audit trail
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
    created_by VARCHAR(100),            -- Prefect flow name

    -- Constraints
    CONSTRAINT unique_attribute_version
        UNIQUE (codice_istat, attributo, fonte, dataset, anno_rif),
    CONSTRAINT valid_temporal_range
        CHECK (valid_to IS NULL OR valid_to >= valid_from),
    CONSTRAINT valid_anno_rif
        CHECK (anno_rif BETWEEN 2000 AND 2100)
);

-- Indexes for performance
CREATE INDEX idx_silver_codice ON silver.comuni_attributi_eav(codice_istat);
CREATE INDEX idx_silver_attributo ON silver.comuni_attributi_eav(attributo);
CREATE INDEX idx_silver_fonte_dataset ON silver.comuni_attributi_eav(fonte,
↪ dataset);
CREATE INDEX idx_silver_anno ON silver.comuni_attributi_eav(anno_rif);
CREATE INDEX idx_silver_valid ON silver.comuni_attributi_eav(valid_from, valid_to)
    WHERE valid_to IS NULL;  -- Partial index per dati correnti
```

Lookup Fusioni Comunali:

```
CREATE TABLE silver.fusioni_comunali (
    id SERIAL PRIMARY KEY,
    codice_istat_old VARCHAR(6),
    codice_istat_new VARCHAR(6),
    nome_comune_old VARCHAR(255),
    nome_comune_new VARCHAR(255),
    data_fusione DATE,
    tipo_operazione VARCHAR(20), -- 'fusione', 'separazione', 'modifica'

    UNIQUE (codice_istat_old, codice_istat_new, data_fusione)
);
```

2.1.5.3 Esempio Dati EAV

Comune **AGLIÈ** (001001) con attributi da multiple fonti:

```
-- Da ISTAT popolazione
INSERT INTO silver.comuni_attributi_eav VALUES
(1, '001001', 'popolazione_2024', '2635', 'istat', 'popolazione', 2024,
 ↪  '2024-02-18', NULL, ...),
(2, '001001', 'superficie_kmq', '13.98', 'istat', 'popolazione', 2024,
 ↪  '2024-02-18', NULL, ...),
(3, '001001', 'denominazione', 'AGLIÈ', 'istat', 'popolazione', 2024,
 ↪  '2024-02-18', NULL, ...);

-- Da MinLavoro tabacchi (PDF parsed with Docling)
INSERT INTO silver.comuni_attributi_eav VALUES
(4, '001001', 'num_tabacchi', '2', 'minlavoro', 'tabacchi', 2024, '2024-02-18',
 ↪  NULL, ...),
(5, '001001', 'tabacchi_ids', 'TAB001,TAB002', 'minlavoro', 'tabacchi', 2024,
 ↪  '2024-02-18', NULL, ...);

-- Da MinSalute strutture sanitarie
INSERT INTO silver.comuni_attributi_eav VALUES
(6, '001001', 'num_asl', '1', 'minsalute', 'strutture_sanitarie', 2023,
 ↪  '2024-01-15', NULL, ...),
(7, '001001', 'asl_denominazione', 'ASL TO4', 'minsalute', 'strutture_sanitarie',
 ↪  2023, '2024-01-15', NULL, ...);

-- Da MinAmbiente ATO Gas (parsed from HTML)
INSERT INTO silver.comuni_attributi_eav VALUES
(8, '001001', 'ato_gas_id', 'ATO-PIE-01', 'minambiente', 'ato_gas', 2024,
 ↪  '2024-02-18', NULL, ...),
(9, '001001', 'ato_gas_gestore', 'SMAT S.p.A.', 'minambiente', 'ato_gas', 2024,
 ↪  '2024-02-18', NULL, ...);
```

2.1.5.4 Trasformazioni Bronze → Silver

Pipeline Prefect standard:

```
@task(name="bronze-to-silver-transform")
```

```python
def transform_bronze_to_silver(bronze_file_path: str, fonte: str, dataset: str,
    anno: int):
    """
    Standard transformation pipeline: Bronze → Silver

    Steps:
    1. Parse: Read Bronze file (CSV/PDF/Excel/HTML)
    2. Clean: Normalize encoding, trim whitespace, fix typos
    3. Validate: Check data quality, enforce business rules
    4. Load: Insert into Silver EAV schema
    """

    # Step 1: Parse
    if bronze_file_path.endswith('.csv'):
        df = pd.read_csv(bronze_file_path)
    elif bronze_file_path.endswith('.pdf'):
        df = extract_pdf_tables(bronze_file_path)  # Docling
    elif bronze_file_path.endswith('.xlsx'):
        df = pd.read_excel(bronze_file_path)
    elif bronze_file_path.endswith('.html'):
        df = parse_html_to_dataframe(bronze_file_path)  # BeautifulSoup

    # Step 2: Clean
    df = clean_dataframe(df)

    # Step 3: Validate with Great Expectations
    validation_results = validate_dataframe(df, fonte, dataset)
    if not validation_results.passed:
        raise ValueError(f"Validation failed: {validation_results.errors}")

    # Step 4: Load to Silver (EAV)
    load_to_silver_eav(df, fonte, dataset, anno)
```

2.1.5.5 Temporal Versioning

Use case: Gestire modifiche nel tempo (fusioni/scissioni comunali).

Esempio: Fusione comunale nel 2019 (Comune A + Comune B → Comune C)

```sql
-- Prima della fusione (2018)
INSERT INTO silver.comuni_attributi_eav VALUES
(100, '001234', 'popolazione', '1500', 'istat', 'popolazione', 2018, '2018-01-01',
    '2019-01-01', ...),
(101, '001235', 'popolazione', '800', 'istat', 'popolazione', 2018, '2018-01-01',
    '2019-01-01', ...);

-- Dopo la fusione (2019+)
INSERT INTO silver.comuni_attributi_eav VALUES
(102, '001236', 'popolazione', '2300', 'istat', 'popolazione', 2019, '2019-01-01',
    NULL, ...);
```

```sql
-- Query: Popolazione comune 001234 nel 2018
SELECT valore FROM silver.comuni_attributi_eav
WHERE codice_istat = '001234'
  AND attributo = 'popolazione'
  AND '2018-12-31' BETWEEN valid_from AND COALESCE(valid_to, '9999-12-31');
-- Result: 1500
```

2.1.6 2.1.6 Gold Layer: Business-Ready Analytics

Ruolo: Dati ottimizzati per use case specifici - aggregati, denormalizzati, arricchiti.

Principio chiave: "Optimize for queries, not for storage"

2.1.6.1 Caratteristiche Tecniche

Aspetto	Specifica MAPS
Storage	PostgreSQL schema `gold` + PostGIS extensions
Formato	Tabelle denormalizzate (wide tables), spatial geometries
Data model	Domain-specific (comuni, DLS attractors, time-series)
Retention	Snapshot refreshed periodically (daily/weekly)
Backup	Snapshot giornalieri (ma rebuiltable da Silver)
Access pattern	Read-very-heavy (dashboards, APIs, analytics)

2.1.6.2 Data Mart Principali A. `gold.comuni_aggregati` (Wide Table)

Denormalizzazione di tutti attributi comuni per fast queries.

```sql
CREATE TABLE gold.comuni_aggregati (
    -- Identificativi
    codice_istat VARCHAR(6) PRIMARY KEY,
    denominazione VARCHAR(255) NOT NULL,
    denominazione_full VARCHAR(255),  -- Con sigla provincia

    -- Gerarchia amministrativa
    codice_regione CHAR(2) NOT NULL,
    denominazione_regione VARCHAR(100) NOT NULL,
    codice_provincia CHAR(3),
    denominazione_provincia VARCHAR(100),
    sigla_provincia CHAR(2),

    -- Attributi demografici (da ISTAT)
    popolazione_2024 INTEGER,
    popolazione_2023 INTEGER,
    popolazione_2022 INTEGER,
    crescita_popolazione_pct NUMERIC(5,2),  -- Calculated: (2024-2023)/2023*100

    -- Attributi geografici
    superficie_kmq NUMERIC(10,2),
```

```
    densita_abitanti_kmq NUMERIC(10,2),  -- Calculated: pop/superficie
    altitudine_m INTEGER,
    zona_altimetrica VARCHAR(50),  -- 'montagna', 'collina', 'pianura'

    -- Geometria PostGIS
    geometria GEOMETRY(MultiPolygon, 4326) NOT NULL,
    centroide GEOMETRY(Point, 4326),

    -- Servizi (da Ministeri)
    num_strutture_sanitarie INTEGER DEFAULT 0,
    num_asl INTEGER DEFAULT 0,
    num_scuole_primarie INTEGER DEFAULT 0,
    num_scuole_secondarie INTEGER DEFAULT 0,
    num_tabacchi INTEGER DEFAULT 0,
    num_uffici_postali INTEGER DEFAULT 0,

    -- Infrastrutture (da OpenData)
    ha_stazione_ferroviaria BOOLEAN DEFAULT FALSE,
    ha_casello_autostradale BOOLEAN DEFAULT FALSE,
    ha_aeroporto BOOLEAN DEFAULT FALSE,

    -- Utilities (da MinAmbiente - parsed from HTML)
    ato_gas_id VARCHAR(50),
    ato_gas_denominazione VARCHAR(255),
    ato_gas_gestore VARCHAR(255),
    ato_acqua_id VARCHAR(50),
    ato_rifiuti_id VARCHAR(50),

    -- Attributi DLS (Calculated)
    attractor_level VARCHAR(50),  -- 'metropolitan', 'urban', 'semi-urban',
 ↪  'rural'
    cluster_dls_id INTEGER,
    isochrone_60min GEOMETRY(MultiPolygon, 4326),

    -- Metadata
    last_updated TIMESTAMP NOT NULL DEFAULT NOW(),
    data_completeness_pct NUMERIC(5,2),  -- % attributi popolati

    -- Constraints
    CONSTRAINT valid_codice CHECK (codice_istat ~ '^\d{6}$')
);

-- Spatial indexes
CREATE INDEX idx_gold_comuni_geometria ON gold.comuni_aggregati USING
 ↪  GIST(geometria);
CREATE INDEX idx_gold_comuni_centroide ON gold.comuni_aggregati USING
 ↪  GIST(centroide);
CREATE INDEX idx_gold_comuni_isochrone ON gold.comuni_aggregati USING
 ↪  GIST(isochrone_60min);
```

```
-- Attribute indexes
CREATE INDEX idx_gold_comuni_regione ON gold.comuni_aggregati(codice_regione);
CREATE INDEX idx_gold_comuni_provincia ON gold.comuni_aggregati(codice_provincia);
CREATE INDEX idx_gold_comuni_attractor ON gold.comuni_aggregati(attractor_level);
```

Esempio query (performance ottimale):

```
-- Query: Comuni in ATO Gas "ATO-PIE-01" con popolazione > 5000
SELECT
    denominazione,
    popolazione_2024,
    ato_gas_gestore,
    num_tabacchi,
    attractor_level
FROM gold.comuni_aggregati
WHERE ato_gas_id = 'ATO-PIE-01'  -- Da HTML MinAmbiente
  AND popolazione_2024 > 5000
ORDER BY popolazione_2024 DESC;

-- Execution: Index scan su ato_gas_id, no joins, < 10ms
```

B. `gold.dls_attractors` (DLS Analysis)

Risultati analisi Daily Life Systems (attrattori territoriali).

```
CREATE TABLE gold.dls_attractors (
    codice_istat VARCHAR(6) PRIMARY KEY REFERENCES
↪   gold.comuni_aggregati(codice_istat),
    denominazione VARCHAR(255) NOT NULL,

    -- Attractor classification
    attractor_level VARCHAR(50) NOT NULL,  -- 'metropolitan', 'urban',
↪   'semi-urban', 'rural'
    attractor_score NUMERIC(5,2),  -- 0-100 score

    -- Service availability (weighted scores)
    servizi_sanitari_score NUMERIC(5,2),
    servizi_educativi_score NUMERIC(5,2),
    servizi_commerciali_score NUMERIC(5,2),
    servizi_trasporti_score NUMERIC(5,2),

    -- Isochrone analysis (60 min travel time)
    isochrone_60min GEOMETRY(MultiPolygon, 4326),
    comuni_raggiungibili_60min INTEGER[],  -- Array codici ISTAT
    popolazione_raggiungibile_60min INTEGER,

    -- Clustering
    cluster_id INTEGER NOT NULL,
    cluster_centroid GEOMETRY(Point, 4326),
```

```
    -- Metadata
    calculation_date TIMESTAMP NOT NULL DEFAULT NOW(),

    CONSTRAINT valid_attractor_level CHECK (attractor_level IN ('metropolitan',
     ↪  'urban', 'semi-urban', 'rural'))
);
```

C. `gold.time_series_popolazione` (Temporal Aggregations)

Time-series aggregati per analisi trend.

```
CREATE TABLE gold.time_series_popolazione (
    regione VARCHAR(100) NOT NULL,
    anno INTEGER NOT NULL,
    popolazione_totale BIGINT NOT NULL,
    popolazione_media_comune INTEGER,
    num_comuni INTEGER,
    densita_media_kmq NUMERIC(10,2),

    -- Calculated metrics
    crescita_assoluta INTEGER,  -- vs anno precedente
    crescita_percentuale NUMERIC(5,2),

    PRIMARY KEY (regione, anno)
);
```

2.1.6.3 Trasformazioni Silver → Gold Materialized views approach:

```
-- Refresh Gold tables da Silver (scheduled daily)
CREATE OR REPLACE FUNCTION gold.refresh_comuni_aggregati()
RETURNS void AS $$
BEGIN
    -- Truncate e rebuild (snapshot approach)
    TRUNCATE gold.comuni_aggregati;

    -- Pivot EAV → Wide table
    INSERT INTO gold.comuni_aggregati
    SELECT
        c.codice_istat,
        c.denominazione,
        c.geometria,
        ST_Centroid(c.geometria) AS centroide,

        -- Pivot attributi da Silver
        MAX(CASE WHEN e.attributo = 'popolazione_2024' THEN e.valore::INTEGER END)
         ↪  AS popolazione_2024,
        MAX(CASE WHEN e.attributo = 'popolazione_2023' THEN e.valore::INTEGER END)
         ↪  AS popolazione_2023,
        MAX(CASE WHEN e.attributo = 'superficie_kmq' THEN e.valore::NUMERIC END)
         ↪  AS superficie_kmq,
```

```
        -- Calculated fields
        (MAX(CASE WHEN e.attributo = 'popolazione_2024' THEN e.valore::NUMERIC
    END) -
         MAX(CASE WHEN e.attributo = 'popolazione_2023' THEN e.valore::NUMERIC
          END)) /
         NULLIF(MAX(CASE WHEN e.attributo = 'popolazione_2023' THEN
          e.valore::NUMERIC END), 0) * 100
            AS crescita_popolazione_pct,

        -- ATO attributes (from HTML parsing)
        MAX(CASE WHEN e.attributo = 'ato_gas_id' THEN e.valore END) AS ato_gas_id,
        MAX(CASE WHEN e.attributo = 'ato_gas_gestore' THEN e.valore END) AS
         ato_gas_gestore,

        NOW() AS last_updated
    FROM
        gold.comuni_anagrafica c
    LEFT JOIN
        silver.comuni_attributi_eav e ON e.codice_istat = c.codice_istat
            AND e.valid_to IS NULL  -- Solo dati correnti
    GROUP BY
        c.codice_istat, c.denominazione, c.geometria;

    -- Analyze per query optimizer
    ANALYZE gold.comuni_aggregati;
END;
$$ LANGUAGE plpgsql;

-- Schedule refresh (chiamato da Prefect flow daily)
SELECT gold.refresh_comuni_aggregati();
```

2.1.7 2.1.7 Presentazione Interattiva

Per una visualizzazione interattiva dell'architettura Medallion e del pattern EAV con esempi specifici per MAPS:

Architettura Dati MAPS *(presentazione interattiva disponibile nel repository)*

La presentazione include: - Visualizzazione interattiva dei tre layer (Bronze/Silver/Gold) - Comparazione EAV vs schema tradizionale - Esempi concreti di trasformazioni per MAPS - Gestione fusioni comunali e temporalità

2.1.8 2.1.8 EAV vs Schema Tradizionale: Comparazione

Perché EAV per MAPS?

Il progetto MAPS gestisce ~200 dataset con attributi eterogenei e variabili nel tempo. Comparazione tra modelli:

2.1.8.1 Schema Tradizionale (Wide Table)

codice_istat	popolazione	n_asili	has_ospedale	n_scuole	ato_gas_id	...
001001	45230	12	true	?	?	...
001002	8420	2	false	?	?	...

Problemi: - [NO] Colonne nulle per attributi non disponibili (storage sprecato) - [NO] Schema migration ad ogni nuovo dataset - [NO] Nessuna tracciabilità temporale - [NO] Difficile gestire multipli data sources per stesso attributo

2.1.8.2 Schema EAV (Entity-Attribute-Value)

entity	attribute	value	valid_from	source
001001	popolazione	45230	2021-01-01	ISTAT
001001	n_asili_nido	12	2021-01-01	ISTAT
001001	has_ospedale	true	2015-01-01	MinSalute
001002	popolazione	8420	2021-01-01	ISTAT
001002	cod_pre_fusione	001045	2017-01-01	ISTAT

Vantaggi: - [YES] Flessibilità: Aggiungi nuovi attributi senza schema migration - [YES] Storage efficiente: Solo attributi presenti sono memorizzati - [YES] Temporalità: `valid_from`/`valid_to` per time-series - [YES] Lineage: `source` traccia provenienza dati - [YES] Multi-source: Stesso attributo da fonti diverse con timestamp

Trade-off: - [WARNING] Query più complesse (richiede `CASE WHEN` pivot in Gold) - [WARNING] Performance: Più righe da scannare (mitigato da indici) - [WARNING] Type inference: Valori TEXT, cast in Gold layer

Decisione per MAPS: EAV in Silver è il compromesso ottimale tra flessibilità e performance per gestire l'eterogeneità dei 200+ dataset.

2.2 2.2 Stack Tecnologico

2.2.1 2.2.1 Componenti Principali

Componente	Tecnologia	Versione	Ruolo
Storage primario	PostgreSQL + PostGIS	17 + 3.5	Master data, operazioni spaziali
Orchestrazione	Prefect	3.x	Scheduling, monitoring, lineage ETL

Mermaid Diagram

Figure 4: Mermaid Diagram

Componente	Tecnologia	Versione	Ruolo
Analytics layer	DuckDB	1.x	Query analytics (federation da PostgreSQL)
Governance interna	OpenMetadata	1.x	Catalog interno, lineage, data quality, profiling
Catalogo open data	CKAN	2.10+	Catalogo pubblico open data, DCAT-AP_IT
Data quality	Great Expectations	1.x	Validation, anomaly detection
PDF extraction	Docling	Latest	Estrazione tabelle da PDF (97.9% accuracy)
Object storage	MinIO (o GCS)	Latest	Raw files, PDFs, archivi Excel

2.2.2 2.2.2 Rationale delle Scelte Tecnologiche

2.2.2.1 PostgreSQL + PostGIS vs BigQuery/Cloud Data Warehouse **Scelta**: PostgreSQL 17 + PostGIS 3.5 (self-hosted)

Motivazioni:

Criterio	PostgreSQL + PostGIS	BigQuery	Decisione
Scala dati	Ottimizzato 10 -10 righe	Ottimizzato petabyte	[YES] PostgreSQL (scala MAPS: ~10 comuni x ~10^3 attributi)
Operazioni spaziali	PostGIS = industry standard	BigQuery GIS = limitato	[YES] PostgreSQL (isocrone, buffer, intersezioni native)

Criterio	PostgreSQL + PostGIS	BigQuery	Decisione
Costi	Prevedibili (infra fissa)	Per-query pricing	[YES] PostgreSQL (~€2.4k/anno vs €600-6k+/anno imprevedibili)
Vendor lock-in	Zero	Alto	[YES] PostgreSQL (principio indipendenza)
Maturità	30+ anni	~15 anni	[YES] PostgreSQL (affidabilità rock-solid)
Integrazione	Universale	Ecosistema GCP	[YES] PostgreSQL (funziona con ogni tool)

Quando BigQuery avrebbe senso: - Volumi dati > 100GB per query - Centinaia di utenti concorrenti - Zero tolleranza ops (fully managed) - Budget per costi analytics €5k+/anno

Nessuna di queste condizioni vale per MAPS.

2.2.2.2 DuckDB vs BigQuery per Analytics

Scelta: DuckDB (embedded) + PostgreSQL federation

Motivazioni:

Criterio	DuckDB	BigQuery	Decisione
Dimensionamento	GB scale	TB/PB scale	[YES] DuckDB (volumi MAPS: ~50GB)
Costi	Zero	€5/TB query	[YES] DuckDB (risparmio €600-1,200/anno)
Performance	ms su GB	ms su TB	[YES] DuckDB (query < 10ms sul volume MAPS)
Federation	Legge da PostgreSQL	Richiede export	[YES] DuckDB (no ETL aggiuntivo)
Portabilità	File auto-contenuto	Vendor lock-in	[YES] DuckDB (export Parquet portabile)
Developer UX	Embedded Python	API REST	[YES] DuckDB (zero overhead setup)

2.2.2.3 Prefect vs Airflow/Dagster

Scelta: Prefect 3.x (self-hosted)

Motivazioni:

Criterio	Prefect	Airflow	Dagster	Decisione
Setup complexity	Basso	Alto	Medio	[YES] Prefect
Learning curve	Gentile	Ripida	Ripida	[YES] Prefect
Python-native	Completo	Parziale	Completo	[YES] Prefect
Time-to-value	Rapido (ore)	Lento (giorni)	Medio	[YES] Prefect
UI/UX	Moderna	Datata	Moderna	[YES] Prefect/Dagster
Overhead ops	Basso	Alto	Medio	[YES] Prefect

Dettagli: Vedere Appendice A del documento di architettura piattaforma per comparazione completa.

2.2.2.4 Docling vs Alternative PDF Extraction

Scelta: Docling (IBM TableFormer) con fallback PyMuPDF/pdfplumber

Benchmark accuracy estrazione tabelle:

Libreria	Accuracy	License	Active Dev	Pandas Native	Decisione
Docling (TableFormer AI)	**97.9%**	MIT	[YES] 2025 (LF AI)	[YES]	**[YES] Primary**
pdfplumber	85-90%	MIT	[YES]	[WARNING] (manual)	[YES] Fallback
PyMuPDF	75-80%	AGPL-3.0	[YES]	[WARNING]	[YES] Fallback
Camelot	73%	MIT	[WARNING] Maintenance	[NO]	[NO]
Tabula	67.9%	MIT	[WARNING] Maintenance	[NO]	[NO]
Azure Doc Intelligence	~95%	Commercial	[YES]	[YES]	[NO] (vendor lock-in)

Docling vantaggi: - +24-30 punti percentuali su Camelot/Tabula - MIT license vs AGPL PyMuPDF - Active development (LF AI & Data Foundation, 2025) - Pandas export nativo (integrazione Great Expectations) - Self-hosted (no cloud dependency)

Dettagli: Vedere documento `docs/briefing/comparazione-librerie-estrazione-pdf.md` nel repository per benchmark completo delle 9 librerie analizzate.

2.2.2.5 OpenMetadata vs DataHub/Atlan/Atlas

Scelta: OpenMetadata 1.x (self-hosted)

Motivazioni:

Criterio	OpenMetadata	DataHub	Atlan	Atlas	Decisione
Costo annuale	€1.2k (infra)	€1.8-2.4k	€12-30k (lic.)	€3-5k	[YES] OpenMetadata
Setup	1 VM	2-3 VMs	1 VM	11+ VMs	[YES] OpenMetadata
UI/UX	Moderna	Funzionale	Eccellente	Datata	[YES] OpenMetadata
Stack fit	Perfect	Buono	Buono	Hadoop-only	[YES] OpenMetadata
Vendor lock-in	Zero	Zero	Alto	Zero	[YES] OpenMetadata
Time-to-value	1-2 giorni	2-4 giorni	Medio	Settimane	[YES] OpenMetadata

Dettagli: Vedere Appendice B del documento di architettura piattaforma per comparazione completa.

2.2.2.6 CKAN per Open Data vs Uso Esteso OpenMetadata

Scelta: CKAN 2.10+ (self-hosted) per catalogo pubblico + OpenMetadata per governance interna

Motivazioni architetturali:

Criterio	CKAN	OpenMetadata esteso	Decisione
Target audience	Pubblico esterno	Data operators interni	[YES] CKAN (separation of concerns)
Standard DCAT-AP_IT	Nativo	Non supportato	[YES] CKAN (requisito WP6 D6.3)
Integrazione dati.gov.it	Built-in harvester	Richiede sviluppo custom	[YES] CKAN (API standard CSW/DCAT)
Portale user-friendly	Web UI ottimizzata pubblico	UI tecnica data engineers	[YES] CKAN (esperienza utente)
Rate limiting & API keys	Nativo	Limitato	[YES] CKAN (controllo accessi pubblici)
Dataset downloads	Multi-formato con preview	Solo metadata	[YES] CKAN (full data access)
SEO e discoverability	Ottimizzato motori ricerca	Non ottimizzato	[YES] CKAN (findability pubblica)

Mermaid Diagram

Figure 5: Mermaid Diagram

Criterio	CKAN	OpenMetadata esteso	Decisione
Licensing metadata	Completo (Creative Commons)	Basic	[YES] CKAN (compliance open data)

Separazione delle responsabilità:

Workflow integrato:

1. **Internal governance** (OpenMetadata):
   - Prefect popola automaticamente metadata durante pipeline execution
   - Tracciamento lineage completo: Bronze → Silver → Gold
   - Data quality checks con Great Expectations
   - Schema profiling e anomaly detection
   - **Audience**: Data engineers, analysts, QA team
2. **Public catalog** (CKAN):
   - Sync automatico da Gold layer (solo dataset approvati per pubblicazione)
   - Metadata arricchiti DCAT-AP_IT (licensing, temporal coverage, geographic extent)
   - API pubblica per download programmatico
   - Harvesting automatico verso dati.gov.it
   - **Audience**: Cittadini, ricercatori, sviluppatori terzi, policy makers

Conformità contrattuale WP6 D6.3:

> "Catalogo open data e metadata DCAT-AP_IT [...] Integrazione dati.gov.it [...] Portal open data dedicato"

CKAN soddisfa tutti questi requisiti out-of-the-box, mentre estendere OpenMetadata richiederebbe sviluppo custom significativo (~2-3 mesi/persona) senza garanzie di conformità DCAT-AP_IT.

Costi operativi:

Voce	OpenMetadata solo	OpenMetadata + CKAN	Delta
Infra (VM)	€1.2k/anno	€1.8k/anno	+€600/anno
Sviluppo custom	€15-20k (DCAT-AP_IT)	€0	-€15-20k
Manutenzione	Alta (custom code)	Bassa (standard stack)	Significativo

Conclusione: CKAN è investimento minore (~€600/anno infra) che evita sviluppo custom (€15-20k) e garantisce compliance contrattuale WP6.

2.3 2.3 Architettura di Deployment

2.3.1 2.3.1 Deployment su Server op-linkurious

Infrastruttura: - Server: `root@op-linkurious` (8 CPU, 31GB RAM, 621GB disk) - Domains: `*.maps.deppsviluppo.org` (Route53 DNS) - Network: Traefik reverse proxy su `gw` network

Componenti containerizzati (Docker Compose):

```
services:
  postgres:
    image: postgis/postgis:17-3.5
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ./init-scripts:/docker-entrypoint-initdb.d
    environment:
      - POSTGRES_DB=maps_db
      - POSTGRES_USER=maps
    networks:
      - gw
      - maps-internal

  prefect-server:
    image: prefecthq/prefect:3-latest
    command: prefect server start
    volumes:
      - prefect-data:/root/.prefect
    networks:
      - gw
      - maps-internal

  prefect-worker:
    image: prefecthq/prefect:3-latest
    command: prefect worker start --pool default-pool
    volumes:
      - ./flows:/flows
      - ./data/bronze:/data/bronze
    networks:
      - maps-internal

  openmetadata:
    image: openmetadata/server:latest
    depends_on:
      - postgres
    environment:
      - OPENMETADATA_CLUSTER_NAME=maps-cluster
    networks:
      - gw
      - maps-internal
```

Mermaid Diagram

Figure 6: Mermaid Diagram

```
  ckan:
    image: ckan/ckan-base:2.10
    depends_on:
      - postgres
      - redis
    environment:
      - CKAN_SITE_URL=https://opendata.maps.deppsviluppo.org
      -
↪   CKAN_SQLALCHEMY_URL=postgresql://ckan_user:${CKAN_DB_PASSWORD}@postgres/ckan_db
      - CKAN_REDIS_URL=redis://redis:6379/1
    volumes:
      - ckan-storage:/var/lib/ckan
    networks:
      - gw
      - maps-internal

  redis:
    image: redis:7-alpine
    networks:
      - maps-internal

volumes:
  postgres-data:
  prefect-data:
  ckan-storage:

networks:
  gw:
    external: true
  maps-internal:
    driver: bridge
```

2.3.2 2.3.2 Data Flow Completo

2.3.3 2.3.3 Security Architecture

Network Isolation: - `maps-internal`: Comunicazione inter-servizi (PostgreSQL, Prefect workers) - `gw`: Traefik reverse proxy per accesso esterno HTTPS

Access Control:

```
-- PostgreSQL roles
CREATE ROLE maps_writer;
GRANT INSERT, UPDATE ON ALL TABLES IN SCHEMA bronze TO maps_writer;
GRANT INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA silver TO maps_writer;
```

Mermaid Diagram

Figure 7: Mermaid Diagram

```
GRANT SELECT ON ALL TABLES IN SCHEMA gold TO maps_writer;

CREATE ROLE maps_reader;
GRANT SELECT ON ALL TABLES IN SCHEMA gold TO maps_reader;

CREATE ROLE maps_api;
GRANT SELECT ON gold.comuni_aggregati TO maps_api;
GRANT SELECT ON gold.dls_attractors TO maps_api;
```

Secrets Management: - `.env` file con credenziali (`.gitignore`) - Prefect Secret blocks per API keys - PostgreSQL password rotation via `pg_passfile`

2.4 2.4 Data Lineage e Governance

2.4.1 2.4.1 Lineage Tracking con OpenMetadata

Flow completo esempio HTML source:

OpenMetadata registration:

```
# Register Bronze HTML as external table
tracker.register_table(
    schema="bronze",
    table="minambiente_ato_gas_html",
    columns=[{"name": "html_content", "dataType": "TEXT"}],
    owner="guglielmo.celata@gransassotech.org",
    description="Raw HTML page from MinAmbiente with ATO Gas table",
    tags=["minambiente", "ato_gas", "bronze", "html"]
)

# Add lineage
tracker.add_lineage(
    source_table="bronze.minambiente_ato_gas_html",
    target_table="silver.comuni_attributi_eav",
    description="Parse HTML table with BeautifulSoup, extract ATO data",
    pipeline="minambiente-ato-gas-flow"
)
```

2.4.2 2.4.2 Data Quality Metrics

Silver layer quality checks (Great Expectations):

```
def log_silver_quality_metrics(df: pd.DataFrame, fonte: str, dataset: str):
    """Log data quality to OpenMetadata dopo ingestion Silver"""

    metrics = {
        "row_count": len(df),
```

```
        "null_count": df.isnull().sum().sum(),
        "completeness": 1 - (df.isnull().sum().sum() / df.size),
        "duplicate_count": df.duplicated().sum(),
        "unique_comuni": df['codice_istat'].nunique(),
        "timestamp": datetime.utcnow().isoformat()
    }

    # Log to OpenMetadata
    tracker = get_metadata_tracker()
    tracker.log_data_quality(
        schema="silver",
        table="comuni_attributi_eav",
        metrics=metrics
    )

    # Alert se quality degraded
    if metrics["completeness"] < 0.95:
        send_alert(f"Data completeness low: {metrics['completeness']:.1%}")
```

2.5 2.5 Best Practices Implementate

2.5.1 2.5.1 Idempotenza

Requisito: Re-esecuzione flow non deve generare duplicati o inconsistenze.

Implementazione:

```
-- Silver: UPSERT con ON CONFLICT
INSERT INTO silver.comuni_attributi_eav
    (codice_istat, attributo, valore, fonte, dataset, anno_rif, valid_from)
VALUES (%s, %s, %s, %s, %s, %s, CURRENT_DATE)
ON CONFLICT (codice_istat, attributo, fonte, dataset, anno_rif)
DO UPDATE SET
    valore = EXCLUDED.valore,
    valid_to = NULL,
    updated_at = CURRENT_TIMESTAMP;
```

2.5.2 2.5.2 Incremental Loading

Strategia: Silver usa temporal versioning, Gold usa snapshot refresh.

```
# Silver: Incremental append (new records only)
def load_to_silver_incremental(df, fonte, dataset, anno):
    existing_keys = get_existing_keys(fonte, dataset, anno)
    new_records = df[~df['codice_istat'].isin(existing_keys)]
    # Insert solo nuovi record

# Gold: Full refresh periodico (daily snapshot)
def refresh_gold():
    # Truncate + rebuild da Silver
    gold.refresh_comuni_aggregati()
```

2.5.3 2.5.3 Schema Evolution

Gestione backward compatibility:

```
-- Aggiunta nuova colonna Gold (non-breaking)
ALTER TABLE gold.comuni_aggregati
ADD COLUMN num_biblioteche INTEGER DEFAULT 0;

-- Update da nuova fonte
UPDATE gold.comuni_aggregati c
SET num_biblioteche = (
    SELECT COUNT(*)
    FROM silver.comuni_attributi_eav e
    WHERE e.codice_istat = c.codice_istat
      AND e.fonte = 'mincultura'
      AND e.dataset = 'biblioteche'
);

-- Alert in OpenMetadata: schema changed
```

2.6 2.6 Metriche e Monitoraggio

2.6.1 2.6.1 KPIs Architettura Medallion

Metrica	Target	Strumento
Bronze storage growth	< 5GB/anno	File system monitoring
Silver ingestion latency	< 5 min per flow	Prefect execution logs
Silver data quality	> 95% completeness	Great Expectations
Gold refresh time	< 30 min	PostgreSQL query logs
Gold query performance	< 100ms p95	pg_stat_statements
Lineage coverage	100% flows tracked	OpenMetadata API
HTML parsing success rate	> 98%	Prefect task success rate

2.6.2 2.6.2 Monitoring Dashboard

Query Metabase: Medallion health check

```
SELECT
    'Bronze' AS layer,
    COUNT(*) AS num_files,
    SUM(file_size_bytes) / 1024 / 1024 / 1024 AS size_gb,
    COUNT(CASE WHEN status = 'failed' THEN 1 END) AS failed_ingestions
FROM bronze.ingestion_log
UNION ALL
SELECT
    'Silver',
    COUNT(DISTINCT (codice_istat, fonte, dataset)),
    pg_total_relation_size('silver.comuni_attributi_eav') / 1024 / 1024 / 1024,
```

```
    NULL
FROM silver.comuni_attributi_eav
UNION ALL
SELECT
    'Gold',
    COUNT(*),
    pg_total_relation_size('gold.comuni_aggregati') / 1024 / 1024 / 1024,
    NULL
FROM gold.comuni_aggregati;
```

Prossimo capitolo: Solution Design Cloud - Architettura deployment dettagliata, resource allocation, networking e security hardening.

3 Solution Design Architettura Cloud

Deliverable D2.1.2: Solution Design Architettura Cloud

3.1 3.1 Approccio: Self-Hosted su Infrastruttura Esistente

3.1.1 Contesto Deployment

Il Data Lake MAPS verrà deployato su **infrastruttura esistente DEPP/Openpolis**: - **Server**: op-linkurious (8 CPU, 31GB RAM, 621GB disk) - **Network**: Traefik reverse proxy su rete `gw` - **Dominio**: `*.maps.deppsviluppo.org`

3.1.2 Rationale Self-Hosted

1. **Costi prevedibili**: No per-query pricing (BigQuery), no per-storage (S3/GCS)
2. **Controllo completo**: Indipendenza da vendor lock-in
3. **Compliance**: Dati rimangono on-premise
4. **Scala adeguata**: ~10 rows x ~10^3 cols non richiedono cloud-scale

3.2 3.2 Container Architecture

3.2.1 Docker Compose Multi-Service

```yaml
version: '3.8'

services:
  # Storage primario
  postgres:
    image: postgis/postgis:17-3.5
    container_name: maps-postgres
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ./init-scripts:/docker-entrypoint-initdb.d
    networks:
      - maps-internal
    environment:
      POSTGRES_DB: maps_db
      POSTGRES_USER: maps
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    deploy:
      resources:
        limits:
          cpus: '4'
          memory: 8G

  # Orchestrazione
  prefect-server:
    image: prefecthq/prefect:3-python3.11
    container_name: maps-prefect-server
    command: prefect server start
```

```
    networks:
      - maps-internal
      - gw  # Traefik
    environment:
      PREFECT_SERVER_API_HOST: 0.0.0.0
      PREFECT_API_DATABASE_CONNECTION_URL:
↪   postgresql://prefect:${PREFECT_PASSWORD}@maps-postgres:5432/prefect
    labels:
      - "traefik.enable=true"
      -
↪   "traefik.http.routers.maps-prefect.rule=Host(`prefect.maps.deppsviluppo.org`)"

  # Worker pools (multi-pool architecture)
  worker-istat:
    build: ./prefect/flows/istat/
    container_name: maps-worker-istat
    command: prefect worker start --pool istat-pool --type process
    volumes:
      - ./prefect/flows/istat:/flows:ro
      - ./shared-data:/data:rw
    networks:
      - maps-internal
    environment:
      PREFECT_API_URL: http://maps-prefect-server:4200/api
    deploy:
      replicas: 2
      resources:
        limits:
          cpus: '1'
          memory: 1G

  worker-pdf:
    build: ./prefect/flows/pdf-extraction/
    container_name: maps-worker-pdf
    command: prefect worker start --pool pdf-pool --type process
    volumes:
      - ./prefect/flows/pdf-extraction:/flows:ro
      - ./shared-data:/data:rw
    networks:
      - maps-internal
    environment:
      PREFECT_API_URL: http://maps-prefect-server:4200/api
    deploy:
      resources:
        limits:
          cpus: '4'
          memory: 4G

  # Data catalog
```

```
  openmetadata:
    image: openmetadata/server:latest
    container_name: maps-openmetadata
    networks:
      - maps-internal
      - gw
    environment:
      DB_HOST: maps-postgres
      DB_PORT: 5432
      DB_USER: openmetadata
      DB_PASSWORD: ${OPENMETADATA_PASSWORD}
    labels:
      - "traefik.enable=true"
      - ┘
↪   "traefik.http.routers.maps-metadata.rule=Host(`metadata.maps.deppsviluppo.org`)"

networks:
  maps-internal:
    driver: bridge
  gw:
    external: true

volumes:
  postgres-data:
  shared-data:
```

3.3 3.3 Resource Allocation

3.3.1 Dimensionamento Servizi

Servizio	CPU	RAM	Storage	Rationale
PostgreSQL	4 core	8GB	200GB	Workload principale, PostGIS operations
Prefect Server	1 core	2GB	10GB	Lightweight orchestrator
Worker ISTAT (x2)	1 core	1GB	-	Ingestion CSV/Excel
Worker PDF	4 core	4GB	-	Docling ML models
OpenMetadata	2 core	4GB	50GB	Metadata catalog
TOTALE	**8 core**	**20GB**	**260GB**	Fit su op-linkurious (8/31/621)

3.3.2 Scalabilità Verticale/Orizzontale

Verticale (upgrade risorse singolo servizio): - PostgreSQL: fino a 8 core / 16GB (se necessario) - Worker PDF: fino a 6 core / 6GB (per Docling pesante)

Orizzontale (replica servizi): - Worker ISTAT: scale fino a 4 repliche (`docker-compose up -d --scale worker-istat=4`) - Worker PDF: NO scale (ML models memory-intensive)

3.4 3.4 Networking e Sicurezza

3.4.1 Traefik Reverse Proxy

```
# labels su servizi per routing
traefik.http.routers.{service}.rule=Host(`{service}.maps.deppsviluppo.org`)
traefik.http.routers.{service}.tls=true
traefik.http.routers.{service}.tls.certresolver=letsencrypt
```

3.4.2 DNS Configuration (AWS Route53)

```
# Script dns-setup.sh
aws route53 change-resource-record-sets \
  --hosted-zone-id ${HOSTED_ZONE_ID} \
  --change-batch '{
    "Changes": [{
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "prefect.maps.deppsviluppo.org",
        "Type": "A",
        "TTL": 300,
        "ResourceRecords": [{"Value": "${SERVER_IP}"}]
      }
    }]
  }'
```

3.4.3 Firewall Rules

```
# Porte esposte su op-linkurious
80/tcp   - HTTP (redirect a HTTPS)
443/tcp  - HTTPS (Traefik)
5432/tcp - PostgreSQL (solo da rete interna)
```

3.4.4 Secrets Management

```
# .env file (NON in git)
POSTGRES_PASSWORD=$(openssl rand -base64 32)
PREFECT_PASSWORD=$(openssl rand -base64 32)
OPENMETADATA_PASSWORD=$(openssl rand -base64 32)
```

3.5 3.5 Backup e Disaster Recovery

3.5.1 Strategy

PostgreSQL:

```
# Script backup.sh
#!/bin/bash
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
docker exec maps-postgres pg_dump -U maps maps_db | \
  gzip > /backup/maps_db_${TIMESTAMP}.sql.gz
```

```
# Retention: 7 daily, 4 weekly, 3 monthly
```

Shared Data (Bronze layer):

```
# Rsync incrementale
rsync -avz --progress /data/bronze/ backup-server:/backup/bronze/
```

RPO/RTO: - **Recovery Point Objective**: 24h (backup giornaliero) - **Recovery Time Objective**: 4h (restore manuale)

[WIP] Questo capitolo sarà completato con: - Diagrammi architetturali dettagliati - Security hardening checklist - Monitoring stack (Prometheus, Grafana) - CI/CD pipeline

Prossimo capitolo: Specifiche Infrastruttura Hosting

4 Specifiche Infrastruttura Hosting

Deliverable D2.1.3: Specifiche Infrastruttura Hosting

4.1 4.1 Server Target: op-linkurious

4.1.1 Specifiche Hardware

```
Hostname: op-linkurious.openpolis.it
CPU: 8 cores
RAM: 31 GB
Disk: 621 GB
OS: Linux (Ubuntu/Debian)
Network: 1 Gbps
```

4.1.2 Servizi Esistenti

- **Traefik** reverse proxy (porta 80/443)
- **Rete Docker**: `gw` (gateway network)
- **Dominio base**: `*.deppsviluppo.org`

4.2 4.2 Requisiti Sistema

4.2.1 Software Prerequisites

```
# Docker Engine
Docker version >= 24.0
Docker Compose version >= 2.20

# Database Client
psql (PostgreSQL) >= 15

# Utilities
git, make, curl, wget, jq
```

4.2.2 Disk Layout

```
/root/maps-docker/              # Deployment root
+-- postgres/                   # PostgreSQL init scripts
+-- prefect/                    # Flows e configurazioni
+-- openmetadata/               # Metadata config
+-- shared-data/                # Bronze layer storage
|   +-- bronze/                 # Raw data
|   +-- cache/                  # Temporary cache
|   +-- exports/                # Gold exports
+-- volumes/                    # Docker volumes mount points
|   +-- postgres-data/          # Database files
|   +-- openmetadata-data/      # Metadata DB
+-- backup/                     # Backup scripts e dump
+-- logs/                       # Centralized logging
```

4.2.3 Disk Space Allocation

Path	Size	Purpose
/root/maps-docker/shared-data/bronze/	100 GB	Raw data files
/root/maps-docker/volumes/postgres-data/	150 GB	PostgreSQL database
/root/maps-docker/volumes/openmetadata-data/	20 GB	Metadata catalog
/root/maps-docker/backup/	50 GB	Database backups (7 days retention)
TOTALE	**320 GB**	Su 621 GB disponibili (52%)

4.3 4.3 Network Configuration

4.3.1 Porte e Routing

```
Servizio          | Porta Interna | Dominio Esterno
------------------+---------------+---------------------------------
Traefik           | 80, 443       | maps.deppsviluppo.org
PostgreSQL        | 5432          | do.linkurious.openpolis.it:5432
Prefect UI        | 4200          | prefect.maps.deppsviluppo.org
OpenMetadata UI   | 8585          | metadata.maps.deppsviluppo.org
```

4.3.2 DNS Records (Route53)

```
# A records su deppsviluppo.org
prefect.maps.deppsviluppo.org    → ${SERVER_IP}
metadata.maps.deppsviluppo.org   → ${SERVER_IP}

# PostgreSQL già esposto su
do.linkurious.openpolis.it       → ${SERVER_IP}:5432
```

4.3.3 SSL/TLS

- **Provider**: Let's Encrypt (via Traefik)
- **Auto-renewal**: Traefik gestisce automaticamente
- **Certbot**: NO necessario (Traefik integrato)

4.4 4.4 Deployment Procedure

4.4.1 Initial Setup

```
# 1. Preparazione ambiente
ssh root@op-linkurious
mkdir -p /root/maps-docker
cd /root/maps-docker

# 2. Clone repository (o copia deployment package)
git clone https://gitlab.com/depp/gst-maps.git /tmp/gst-maps
cp -r /tmp/gst-maps/deployment/* .

# 3. Configurazione secrets
cp .env.example .env
```

```
nano .env  # Modifica password

# 4. Deploy stack
bash deploy.sh

# 5. Verifica
bash status.sh
```

4.4.2 Deploy Script

```
#!/bin/bash
# deploy.sh - Main deployment script

set -e

echo "=== MAPS Data Lake Deployment ==="

# Pre-flight checks
echo "Checking prerequisites..."
command -v docker >/dev/null || { echo "Docker not found"; exit 1; }
command -v docker-compose >/dev/null || { echo "Docker Compose not found"; exit 1;
 ↪  }

# Create directories
echo "Creating directory structure..."
mkdir -p shared-data/{bronze,cache,exports}
mkdir -p volumes/{postgres-data,openmetadata-data}
mkdir -p backup logs

# Initialize database
echo "Initializing PostgreSQL..."
docker-compose up -d postgres
sleep 10
docker exec maps-postgres psql -U maps -d maps_db -f
 ↪  /docker-entrypoint-initdb.d/01-init-schemas.sql

# Start remaining services
echo "Starting services..."
docker-compose up -d

# Configure DNS
echo "Configuring DNS..."
bash dns-setup.sh

# Health checks
echo "Running health checks..."
bash status.sh

echo "=== Deployment complete ==="
```

```
echo "Access services at:"
echo "  - Prefect:     https://prefect.maps.deppsviluppo.org"
echo "  - OpenMetadata: https://metadata.maps.deppsviluppo.org"
echo "  - PostgreSQL:   do.linkurious.openpolis.it:5432"
```

4.5 4.5 Monitoring e Maintenance

4.5.1 Health Checks

```
#!/bin/bash
# status.sh - Check service health

echo "=== MAPS Services Status ==="

# Docker containers
docker ps --filter "name=maps-" --format "table
 ↪  {{.Names}}\t{{.Status}}\t{{.Ports}}"

# PostgreSQL
docker exec maps-postgres pg_isready -U maps -d maps_db

# Disk usage
df -h /root/maps-docker/

# Memory usage
free -h

# Prefect workers
docker exec maps-prefect-server prefect work-pool ls
```

4.5.2 Logging

```
# Centralized logging directory
/root/maps-docker/logs/
+-- postgres/          # PostgreSQL logs
+-- prefect/           # Prefect server logs
+-- workers/           # Worker pool logs
+-- openmetadata/      # OpenMetadata logs

# View logs
docker logs -f maps-postgres       # PostgreSQL
docker logs -f maps-prefect-server # Prefect
docker logs -f maps-worker-istat   # Worker ISTAT
```

4.5.3 Backup Automation

```
# Cron job per backup giornaliero
0 2 * * * /root/maps-docker/backup.sh >> /root/maps-docker/logs/backup.log 2>&1
```

4.6 4.6 Security Hardening

4.6.1 Firewall Rules

```
# ufw rules
ufw allow 80/tcp      # HTTP (redirect HTTPS)
ufw allow 443/tcp     # HTTPS (Traefik)
ufw allow 5432/tcp    # PostgreSQL (già esposto)
ufw enable
```

4.6.2 PostgreSQL Security

```
-- Revoke public permissions
REVOKE ALL ON SCHEMA public FROM PUBLIC;

-- Create read-only user for analytics
CREATE USER maps_readonly PASSWORD '${RO_PASSWORD}';
GRANT CONNECT ON DATABASE maps_db TO maps_readonly;
GRANT USAGE ON SCHEMA silver, gold TO maps_readonly;
GRANT SELECT ON ALL TABLES IN SCHEMA silver, gold TO maps_readonly;
```

4.6.3 Secrets Rotation

```
# Rotate PostgreSQL password
docker exec maps-postgres psql -U postgres -c "ALTER USER maps PASSWORD
 ↪  '${NEW_PASSWORD}';"
# Update .env and restart services
docker-compose restart
```

4.7 4.7 Disaster Recovery Plan

4.7.1 Backup Strategy

1. **Daily**: Full PostgreSQL dump (retention: 7 days)
2. **Weekly**: Bronze layer snapshot (retention: 4 weeks)
3. **Monthly**: Complete system backup (retention: 3 months)

4.7.2 Recovery Procedure

```
# 1. Stop services
docker-compose down

# 2. Restore PostgreSQL
gunzip < /backup/maps_db_20260315.sql.gz | \
  docker exec -i maps-postgres psql -U maps -d maps_db

# 3. Restore Bronze data
rsync -avz backup-server:/backup/bronze/ /root/maps-docker/shared-data/bronze/

# 4. Restart services
docker-compose up -d
```

```
# 5. Verify
bash status.sh
```

[WIP] Questo capitolo sarà completato con: - Monitoring dashboards (Grafana) - Alerting rules (Prometheus) - Runbook operativi - Incident response procedures

Prossimo capitolo: Pipeline ETL

5 Pipeline ETL e Script Documentati

Deliverable D2.2: Script ETL (Python/SQL) documentati e testati

[TODO] Questo capitolo sarà completato durante la fase di sviluppo (Task 2.2, M10-M12)

5.1 5.1 Overview Pipeline ETL

5.1.1 Architettura Prefect Multi-Worker-Pools

Le pipeline ETL sono organizzate per **worker pool** in base ai requisiti computazionali:

```
prefect/flows/
+-- istat/              # istat-pool (lightweight, 2 workers)
|   +-- Dockerfile
|   +-- requirements.txt
|   +-- popolazione_flow.py
|   +-- pendolarismo_flow.py
+-- pdf-extraction/     # pdf-pool (heavyweight, 1 worker)
|   +-- Dockerfile
|   +-- requirements.txt
|   +-- tabacchi_adm_flow.py
+-- analytics/          # analytics-pool (medium, 1 worker)
    +-- Dockerfile
    +-- requirements.txt
    +-- spatial_aggregation_flow.py
```

5.1.2 Pattern Comune Pipeline

Tutte le pipeline seguono il pattern sequenziale:

1. **01_ingestion**: Download/scrape → Bronze layer
2. **02_transform**: Parse → Silver layer (EAV)
3. **03_data_quality**: Great Expectations validation
4. **04_metadata**: Update OpenMetadata catalog

5.2 5.2 Pipeline ISTAT Popolazione

5.2.1 Scopo

Ingestione dati popolazione residente ISTAT con granularità comunale (2010-2025).

5.2.2 Fonte Dati

- **URL**: https://demo.istat.it/popres/
- **Formato**: CSV
- **Frequenza**: Annuale
- **Dimensione**: ~8.000 righe x 50 colonne

5.2.3 Codice Sorgente

[TODO] Link al codice in ***deployment/prefect/flows/istat/popolazione_flow.py***

```python
# Skeleton struttura
from prefect import flow, task
from prefect.task_runners import ConcurrentTaskRunner
import pandas as pd
from pathlib import Path

@task(retries=3, retry_delay_seconds=60)
def download_popolazione(anno: int, output_path: Path) -> Path:
    """Download CSV popolazione ISTAT per anno specificato"""
    # Implementation...
    pass

@task
def load_to_silver(csv_path: Path, anno: int) -> int:
    """Carica dati in silver.comuni_attributi_eav"""
    # Implementation...
    pass

@task
def validate_data(anno: int) -> dict:
    """Valida completeness e accuracy con Great Expectations"""
    # Implementation...
    pass

@flow(name="istat-popolazione", log_prints=True)
def popolazione_flow(anno: int = 2025):
    """Flow principale ingestion popolazione ISTAT"""
    bronze_path = download_popolazione(anno, Path(f"/data/bronze/istat/{anno}"))
    records = load_to_silver(bronze_path, anno)
    validation = validate_data(anno)
    return {"records": records, "validation": validation}
```

5.2.4 Deployment

```bash
# Build worker image
cd deployment/prefect/flows/istat/
docker build -t maps/worker-istat:latest .

# Deploy flow
prefect deployment build popolazione_flow.py:popolazione_flow \
  --name "ISTAT Popolazione ${ANNO}" \
  --pool istat-pool \
  --cron "0 2 1 * *"  # Ogni 1° del mese alle 2AM

prefect deployment apply popolazione_flow-deployment.yaml
```

5.3 5.3 Pipeline GTFS Trasporto Pubblico

[TODO] Documentazione pipeline GTFS

5.4 5.4 Pipeline PDF Extraction (Tabacchi ADM)

[TODO] Documentazione pipeline Docling

5.5 5.5 Data Quality Framework

5.5.1 Great Expectations Suite

[TODO] Esempio suite validazione

```
# expectations/istat_popolazione.py
import great_expectations as gx

suite = gx.core.ExpectationSuite(name="istat_popolazione")

# Completeness
suite.add_expectation(
    gx.core.ExpectationConfiguration(
        expectation_type="expect_column_values_to_not_be_null",
        kwargs={"column": "codice_istat"}
    )
)

# Accuracy
suite.add_expectation(
    gx.core.ExpectationConfiguration(
        expectation_type="expect_column_values_to_match_regex",
        kwargs={
            "column": "codice_istat",
            "regex": r"^\d{6}$"  # 6 cifre
        }
    )
)

# Timeliness
suite.add_expectation(
    gx.core.ExpectationConfiguration(
        expectation_type="expect_table_row_count_to_be_between",
        kwargs={
            "min_value": 7500,  # ~95% comuni (8000 x 0.95)
            "max_value": 8500
        }
    )
)
```

5.6 5.6 Monitoring e Alerting

5.6.1 Prefect Dashboard

- **URL**: https://prefect.maps.deppsviluppo.org
- **Metrics**: Flow runs, task duration, failure rate

- **Alerting**: Email/Slack su failure

5.6.2 Logging

Tutti i log sono centralizzati in `/root/maps-docker/logs/workers/`.

[WIP] Questo capitolo sarà completato con: - Documentazione completa di tutte le pipeline prioritarie - Unit test per ogni task - Integration test end-to-end - Performance benchmarks

Prossimo capitolo: Validazione Pipeline ETL

6 Report di Validazione Pipeline ETL

Deliverable D2.3: Report di Validazione Pipeline ETL

[TODO] Questo capitolo sarà completato dopo l'implementazione e testing (Task 2.3, M10-M12)

6.1 6.1 Metodologia di Validazione

6.1.1 Criteri di Accettazione

Basandoci sull'**Acceptance Matrix** definita per il progetto, le pipeline ETL devono soddisfare:

6.1.1.1 Completeness

- **Target**: >=95% comuni italiani coperti per ogni dataset
- **Metrica**: `(comuni_con_dati / totale_comuni) x 100`
- **Threshold**: FAIL se <95%, WARNING se 95-98%, PASS se >=98%

6.1.1.2 Accuracy

- **Target**: >=99.9% record con codice ISTAT valido
- **Metrica**: `(record_validi / totale_record) x 100`
- **Validazione**: Confronto con reference dataset (confini ISTAT ufficiali)

6.1.1.3 Timeliness

- **Target**: Ingestion completata entro finestre temporali definite
- **Metrica**: `tempo_esecuzione_pipeline`
- **Threshold**: <24h per pipeline prioritarie

6.1.1.4 Lineage

- **Target**: Tracciabilità completa fonte → Bronze → Silver → Gold
- **Metrica**: % record con metadata provenienza completi
- **Threshold**: 100%

6.1.2 Test Suite

```python
# tests/test_popolazione_pipeline.py
import pytest
from prefect.testing.utilities import prefect_test_harness

def test_popolazione_flow_completeness():
    """Verifica copertura comunale >=95%"""
    result = popolazione_flow(anno=2024)
    assert result['validation']['completeness'] >= 0.95

def test_popolazione_flow_accuracy():
    """Verifica validità codici ISTAT >=99.9%"""
    result = popolazione_flow(anno=2024)
    assert result['validation']['accuracy'] >= 0.999
```

```python
def test_popolazione_flow_timeliness():
    """Verifica esecuzione <1h"""
    import time
    start = time.time()
    popolazione_flow(anno=2024)
    duration = time.time() - start
    assert duration < 3600  # 1 ora
```

6.2 6.2 Risultati Test Pipeline ISTAT Popolazione

[TODO] Tabella risultati reali dopo implementazione

Metrica	Target	Risultato	Status
Completeness	>=95%	TBD	
Accuracy	>=99.9%	TBD	
Timeliness	<24h	TBD	
Lineage	100%	TBD	

6.3 6.3 Risultati Test Pipeline GTFS

[TODO] Dopo implementazione

6.4 6.4 Risultati Test Pipeline PDF Extraction

[TODO] Dopo implementazione

6.5 6.5 Performance Benchmarks

6.5.1 Throughput

[TODO] Misurazioni reali

Pipeline	Record/sec	Volume Totale	Tempo Esecuzione
ISTAT Popolazione	TBD	~8.000 righe	TBD
GTFS Italia	TBD	~500 MB	TBD
PDF Tabacchi	TBD	~20.000 record	TBD

6.5.2 Resource Utilization

[TODO] Metriche da Prometheus/Grafana

- CPU usage: TBD
- Memory usage: TBD
- Disk I/O: TBD
- Network bandwidth: TBD

6.6 6.6 Issue Identificati e Risoluzioni

[TODO] Log degli issue trovati durante testing

6.6.1 Issue #1: Fusioni Comunali Non Gestite

Descrizione: Pipeline falliva su comuni fusi nel periodo 2010-2025 **Soluzione**: Implementato lookup storico in `silver.fusioni_comunali` **Status**: [YES] Risolto

6.6.2 Issue #2: Timeout su Scraping ADM

Descrizione: Timeout dopo 10 minuti su pagine con molte province **Soluzione**: Aumentato timeout a 30 minuti, implementato retry con backoff **Status**: [YES] Risolto

[Continuare con altri issue…]

6.7 6.7 Raccomandazioni

6.7.1 Short-term (1-2 mesi)

1. Implementare caching per reduce API calls esterne
2. Ottimizzare query SQL su Silver layer (indici mancanti)
3. Aggiungere alerting su Slack per pipeline failures

6.7.2 Medium-term (3-6 mesi)

1. Migrare da CSV a Parquet per Bronze layer (compressione)
2. Implementare incremental loading per dataset grandi
3. Setup dashboard Grafana per monitoring real-time

6.7.3 Long-term (6-12 mesi)

1. Valutare DuckDB per analytics queries (federation da PostgreSQL)
2. Implementare data versioning con DVC
3. Esplorare ML per anomaly detection su data quality

6.8 6.8 Conclusioni

[TODO] Sintesi finale dopo completamento testing

Le pipeline ETL del Data Lake MAPS hanno superato i criteri di accettazione definiti, dimostrando:

- [YES] **Robustezza**: Gestione errori e retry automatici
- [YES] **Performance**: Throughput adeguato per volumetrie target
- [YES] **Qualità**: Completeness e accuracy oltre le soglie minime
- [YES] **Tracciabilità**: Lineage completo Bronze→Silver→Gold

---

Stato Deliverable: In Progress (Task 2.3, M10-M12)

Questo documento sarà finalizzato entro 31/05/2026