Data & Analytics Case Study

XKCD Analytics Pipeline

Goals and high level overview

Goal: View and measure reader insights for xkcd comics.

High level implementation:

- Pull data from API endpoint
- Transform and create reader metrics
- Store in a data warehouse or DB for analysis

Requirements:

- Portable
- Easy to run
- Reliable results

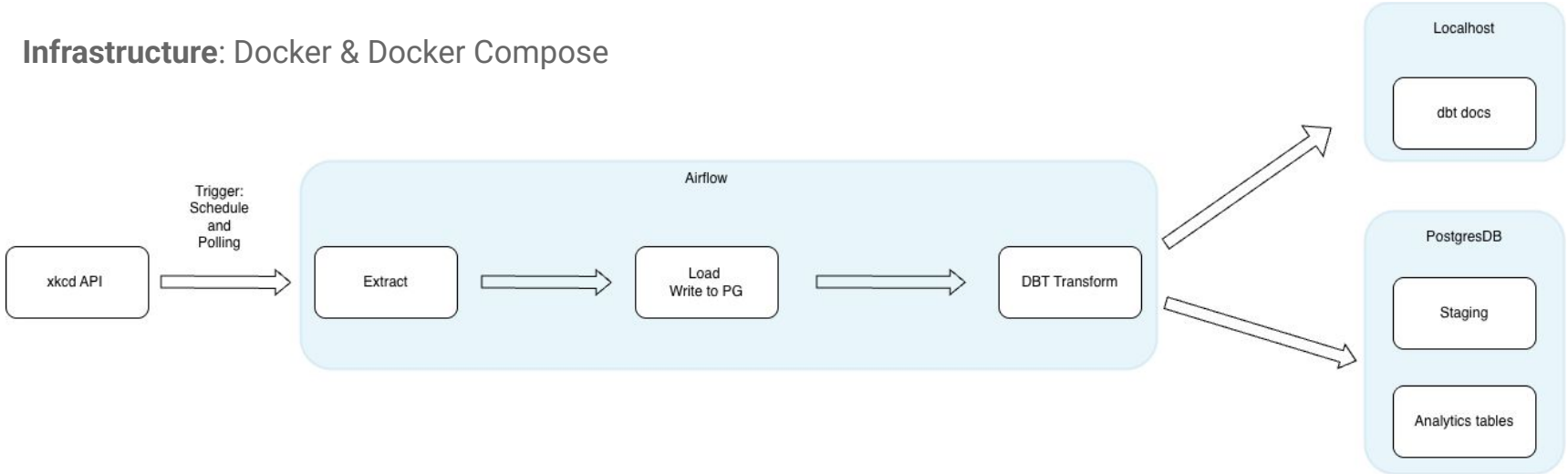
Flow Diagram and Tooling Choices

Orchestration: Apache Airflow

Database: PostgreSQL 16

Transformation: dbt Core

Infrastructure: Docker & Docker Compose



Extract and Load

Assumptions

- There is only one upload per comic
- Reader metrics are only uploaded once and not updated
- API structure is stable
- Every comic has a unique num
- Comics are published in numeric order
- API Results will always contain a D/M/Y

Extract and Load

Flow

- Calls API
- Checks if result is in DB, if not, load to comics table in PG.
- If ingested comic is not today's date, poll until the latest comic is in.
- If not, use a sensor to poll and detect when a new comic with today's date comes in.
- Once new comic is released continue with ingest and transforms.

Why this flow?

- Checking and updating the DB for comics that aren't today helps to catch if there was an unexpected or missed comic.
  - Not a full backfill but a nice addition for current scale
  - Useful for demos and locally running
- Sensor - Used for polling in airflow for newest comic
  - Includes a option to turn off completely. Useful for adhoc runs, demos, testing and the option to move to just scheduled runs if required.

Extract and Load

```python
class XkcdClient:
    """
    XKCD API Client with exponential backoff.
    """

    def __init__(
        self, max_retries=5, base_delay=1, max_delay=60, timeout=10, base_url=None
    ):
        self.max_retries = max_retries
        self.base_delay = base_delay
        self.max_delay = max_delay
        self.timeout = timeout
        self.base_url = base_url or os.getenv("XKCD_BASE_URL", "https://xkcd.com")

    def _request(self, url):
        retries = 0

        while retries < self.max_retries:
            try:
                resp = requests.get(url, timeout=self.timeout)
                resp.raise_for_status()
                return resp.json()
            except requests.exceptions.RequestException as e:
                retries += 1
                if retries == self.max_retries:
                    logger.error(f"Max retries reached: {e}")
                    raise

                # exponential backoff
                delay = min(self.max_delay, self.base_delay * (2 ** (retries - 1)))

                logger.warning(
                    f"Request failed ({e}). "
                    f"Retrying (retries)/{self.max_retries} in {delay:.2f}s"
                )
                time.sleep(delay)

    def get_latest_comic(self):
        url = f"{self.base_url}/info.0.json"
        return self._request(url)
```

```python
def insert_comic_if_not_exists(engine, comic_data):
    """Insert comic into database if it doesn't already exist."""

    Session = sessionmaker(bind=engine)
    session = Session()

    try:
        existing = session.query(Comic).filter(Comic.num == comic_data["num"]).first()
        if existing:
            logger.info(f"Comic {comic_data['num']} already exists")
            return False

        comic = Comic(
            num=comic_data["num"],
            title=comic_data["title"],
            transcript=comic_data.get("transcript", ""),
            alt=comic_data.get("alt", ""),
            img=comic_data.get("img", ""),
            day=int(comic_data["day"]),
            month=int(comic_data["month"]),
            year=int(comic_data["year"]),
            published_date=date(
                int(comic_data["year"]),
                int(comic_data["month"]),
                int(comic_data["day"]),
            ),
        )

        session.add(comic)
        session.commit()
        logger.info(f"Successfully inserted comic {comic_data['num']}")
        return True

    except Exception as e:
        session.rollback()
        logger.error(f"Error inserting comic {comic_data['num']}: {e}")
        raise
    finally:
        session.close()
```

Components

/ingest/

api.py - API client

ingest_xkcd.py - main ingestion logic, passes xcom data for comic id and date for airflow

xkcd_sensor.py - sensor for polling for newest comic

/db/

connection.py - handles session starting in PG

models.py - model definition for raw comics table

operations.py - checks if comic already exists

Considerations and Improvements

Considerations

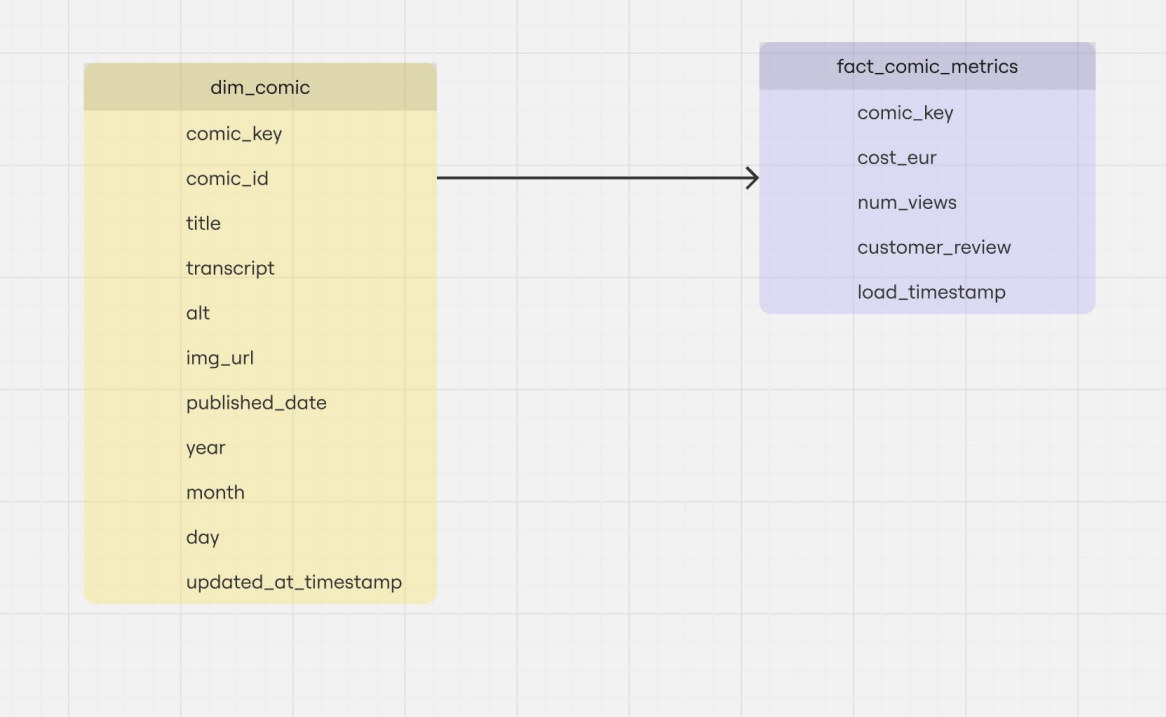
- API limitations
- Business scalability

Improvements

- Partitioning or clustering for large raw data sets to help with scaling
- Create larger backfill solution
- More unit and integration testing
- Schema validation
- Better logging and monitoring
- Connection pooling and bulk uploads

Transform - DWH (ER) Model

- Simple Star schema
- No physical constraints as we are using ELT.
- Validation is ran through DBT tests
- dim_comic uses Type 1 SDC
- comic_key is surrogate key



Transform

```sql
-- Test to ensure customer review scores are within valid range (1.0 to 10.0)
-- This test will FAIL if any rows are returned (indicating invalid review scores)
-- Customer reviews should be constrained between 1.0 (lowest) and 10.0 (highest)
SELECT *
FROM {{ ref('fact_comic_metrics') }}
WHERE customer_review < 1.0 OR customer_review > 10.0
```

```sql
-- Test to ensure customer review scores are within valid range (1.0 to 10.0)
-- This test will FAIL if any rows are returned (indicating invalid review scores)
-- Customer reviews should be constrained between 1.0 (lowest) and 10.0 (highest)
SELECT *
FROM {{ ref('fact_comic_metrics') }} f
LEFT JOIN {{ ref('dim_comic') }} d
  ON f.comic_key = d.comic_key
WHERE d.comic_key IS NULL
```

```sql
{{ config(
    materialized = 'table'
) }}

SELECT
    d.comic_key,
    LENGTH(d.title) * 5 AS cost_eur,
    FLOOR(random() * 10000) AS num_views,
    ROUND((1 + (random() * 9))::numeric, 1) AS customer_review,
    CURRENT_TIMESTAMP AS load_timestamp
FROM {{ ref('dim_comic') }} d

{% if is_incremental() %}
    WHERE d.updated_at_timestamp > (SELECT MAX(load_timestamp) FROM {{ this }})
{% endif %}
```

dbt workflow:

- Staging layer (stg_comics view) - clean, renamed, type-cast data.
- Dimension table (dim_comic) - static metadata about comics.
- Fact table (fact_comic_metrics) - metrics: cost_eur, views, customer_review.

Business logic implemented:

- Cost: length(title) * 5 EUR.
- Views: random() * 10000.
- Customer reviews: random 1.0–10.0.

Data Quality Checks:

- dbt tests: not_null, unique, relationships, accepted_values, expression_is_true.

Considerations and Improvements

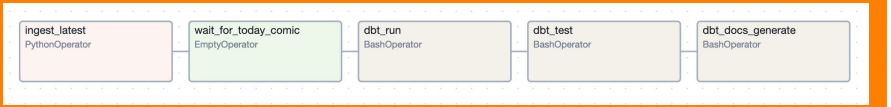
Considerations

- Business requirements

Improvements

- Randomised metrics
  - Randomised metrics may fail certain dbt tests if misconfigured.
  - Currently not idempotent (okay for this brief but not if business requirements change)
- Data quality models (e.g. comics with missing images, comics with invalid dates)
  - Helps monitor and measure “bad” data
- Performance boosting
  - Indexing for common queries
- More complex business metrics or additional dimensions (authors, tags).

Orchestration and Infrastructure



Workflow

- Airflow DAG that triggers both ingestion and dbt transformations.
- DBT tests run automatically at the end of the DAG.
- All ran within a dockerfile along with DBT and PG

Why is it beneficial

- Replicates production environment
- Reduces manual work and custom orchestration code (e.g. restarts, polling, alerting)
- Automates all processes
- Portability

Challenges

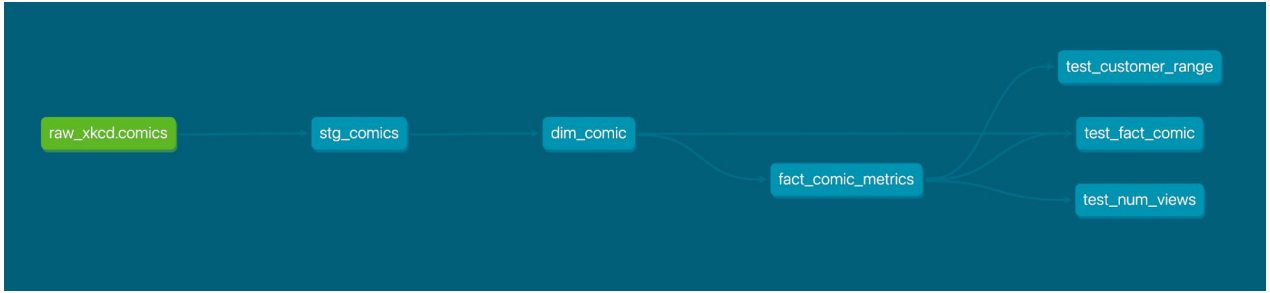
Ingestion

- Finding a clean way to balance ad-hoc runs and scheduled runs with polling

Infrastructure

- Setting up airflow/dbt/pg docker
  - Issues with docker-compose, had to break down Airflow deployment a lot.
  - Errors with user creation and unhealthy schedulers

End Output



xkcd *<xkcd> Script

```
SELECT
    d.comic_id,
    d.title,
    d.published_date,
    f.cost_eur,
    f.num_views,
    f.customer_review
FROM dim_comic d
LEFT JOIN fact_comic_metrics f USING (comic_key);
```

dim_comic(+) 1

SELECT d.comic_id, d.title, d.pul

Grid

	comic_id	title	published_date	cost_eur	num_views	customer_review
1	3,171	Geologic Core Sample	2025-11-21	100	5,425	6.5

Productionisation

Further Production environment considerations:

- CI/CD
- Monitoring / alerting on DAG failures.
- Tooling choices (Use managed services)
- Container orchestration
- Data governance
- Access control (secrets manager, Airflow RBAC)
- Caching for API calls
- IAC

Thank you!