Workflow Overview

1. **Ingestion (Bronze Layer)**
   - Pull Fingrid hydro power API data (JSON/CSV) using Spark notebook.
   - Store directly in Fabric Lakehouse (bronze tables).
2. **Cleaning & Transformation (Silver Layer)**
   - Normalize timestamps, units, missing values.
   - Join with **hydro plant metadata** (capacity, type).
3. **Analytics (Gold Layer)**
   - Aggregations: hourly/daily averages, regional summaries.
   - Calculate load factor = actual output / capacity.
   - Store as clean fact tables for dashboards.
4. **Streaming Simulation**
   - Use Fabric Eventstream (or Python generator) to push real-time power readings.
   - Spark Structured Streaming job consumes → stores into Lakehouse.
5. **CI/CD**
   - GitHub repo with all notebooks & SQL scripts.
   - GitHub Actions workflow runs unit tests (Spark job validation, SQL checks).
   - Deploy updated pipelines automatically into Fabric workspace.
6. **Visualization (Power BI)**
   - Simple dashboard with:
     - Hydro production trends (hourly, daily)
     - Regional breakdowns
     - Anomaly detection (e.g., sudden dips in output)

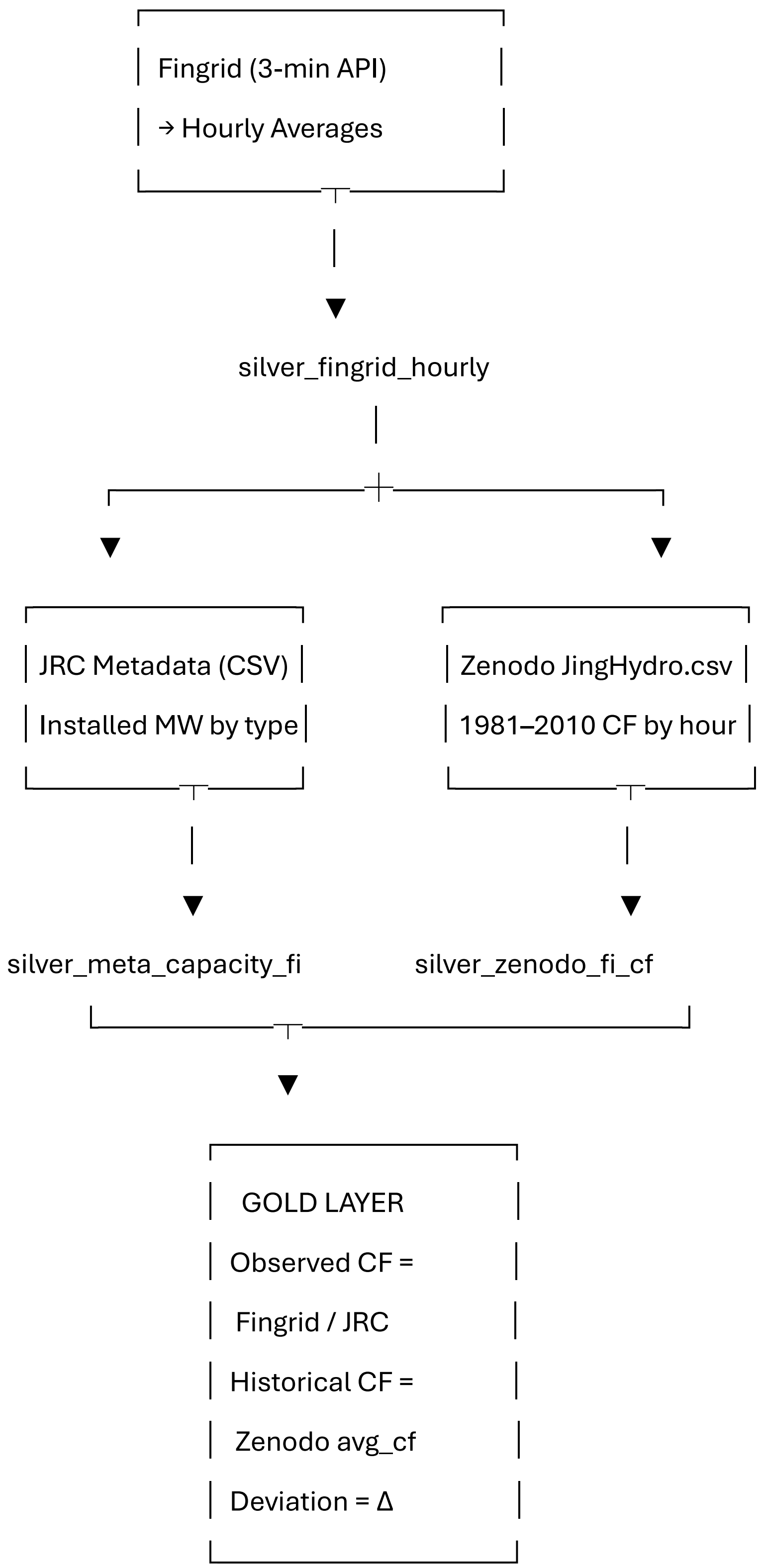
Dataset	What It Tells	Time Dimension	Granularity	Purpose
Fingrid API	Actual real-time generation (MW)	Minutes	National total	Observed performance
Zenodo	Modeled long-term capacity factors	Hourly (1981–2010)	Country/bidding zone	Historical baseline
Metadata	Installed plant capacity, type, location	Static	Plant-level	Reference and scaling factor

Layer	Dataset	Granularity	What It Brings	How It Connects
Silver	silver_fingrid_hourly	Hourly	Real generation (MW)	Used as numerator in Observed CF
Silver	silver_meta_capacity_fi	Static	Installed capacity (MW) by type	Denominator for Observed CF
Silver	silver_zenodo_fi_cf	Hourly (1981–2010)	Historical capacity factors by hour/month	Baseline to compare observed CF
Gold	—	Hourly (aligned)	Joins all three → Observed CF, Historical CF, and Deviation	Final KPI output

So:

$$\text{Observed CF} = \frac{\text{Fingrid Hourly MW}}{\text{Installed Capacity (MW)}}$$

$$\text{Deviation} = \text{Observed CF (2025)} - \text{Historical CF (1981–2010 average)}$$


Fingrid (3-min API)
→ Hourly Averages
silver_fingrid_hourly
JRC Metadata (CSV)
Installed MW by type
Zenodo JingHydro.csv
1981–2010 CF by hour
silver_meta_capacity_fi
silver_zenodo_fi_cf
GOLD LAYER
Observed CF =
Fingrid / JRC
Historical CF =
Zenodo avg_cf
Deviation = Δ

Hydropower Performance Analysis — From Data Pipeline to Power BI Dashboard

This project demonstrates the end-to-end development of a modern data-engineering pipeline and analytics model for assessing Finland's hydropower efficiency against long-term climatic baselines. The objective was to integrate real-time operational data with historical hydrological benchmarks, transforming raw feeds into actionable insights through Microsoft Fabric and Power BI.

The process began with data ingestion at the **Bronze layer**, where three complementary sources were collected.

1. **Fingrid API** provided real-time electricity generation from Finnish hydro power plants at 3-minute intervals. These records reflect the country's live production output in megawatts (MW).
2. **Zenodo's JingHydro dataset** offered nationally aggregated hourly capacity-factor data for 30 climatic years (1981-2010), serving as a long-term baseline of hydro potential under historical conditions.
3. **The JRC Hydro-Power-Plant Database** supplied metadata on installed capacities and plant typologies (run-of-river, storage, and pumped-storage), giving crucial context for calculating efficiency metrics.

At the **Silver layer**, each dataset was cleaned and standardized. Fingrid data were resampled from 3-minute intervals to hourly averages and stored as silver_fingrid_hourly. Zenodo's European-wide data were filtered for Finland and separated into run-of-river and storage components (silver_zenodo_fi_cf). Metadata were aggregated to produce Finland's total installed hydro capacity (silver_meta_capacity_fi).

The **Gold layer** combined these curated sources into analytics-ready tables. Observed capacity factors were computed by dividing actual generation (Fingrid) by total installed capacity (metadata). Historical capacity factors were drawn from Zenodo's long-term averages. A deviation metric captured how current operational performance diverged from climatological expectations. Monthly aggregates produced two key tables:

- gold_cf_deviation_monthly — observed CF, historical CF, and deviation.
- gold_zenodo_baseline_monthly — 30-year mean and standard deviation of historical CFs.

Finally, results were presented through **Power BI visualizations** connected directly to the Fabric Lakehouse in Direct Lake mode:

1. **Line Chart:** plotted avg_observed_cf, avg_historical_cf, and baseline_avg_cf to compare current October performance with the 30-year baseline.
2. **Column Chart:** visualized avg_deviation by month, highlighting whether production was above or below expectations (−3.2 % for October 2025).
3. **Area Band Chart:** used baseline_avg_cf ± baseline_std_cf to illustrate natural historical variability, forming a shaded confidence band around the baseline trend.

Although the pilot currently covers ten days of Fingrid data (October 2025), the architecture is fully scalable: as new months are ingested, Power BI visuals will automatically extend. The project showcases the integration of real-time cloud ingestion, PySpark transformations, and business-ready analytics in Microsoft Fabric. It bridges energy-sector domain data and modern engineering practices—turning raw hydropower measurements into intuitive, evidence-based operational intelligence.

From **ETL → analytics → AI augmentation:**

Component	Purpose
Data Engineering workspace (Lakehouse)	Stores all gold-level Delta tables.
Semantic Model (Power BI / Fabric)	Defines relationships and measures (makes it queryable).
Lakehouse AI services or Copilot in Fabric Notebooks	For retrieval + LLM generation.
Vector Index (OneLake AI / Azure Cognitive Search)	Stores embeddings of time-series chunks or metadata for fast retrieval.

Since this is just 1ECTS course, I did not go on refining the visuals in Power BI, however, I tried to demonstrate a fully working end-to-end analytical workflow following data fetching via a unified platform, data processing using medallion architecture and analytics in Power BI.

After this, I also wanted to try out generative AI capabilities, so I wanted to implement RAG in the time-series data that I had. I performed following things:

- **Build Feature Store:** Create a new Lakehouse table with time windows (hour/day) and derived features (CF mean, deviation, trend slope).
- **Generate Embeddings:** Could not find Fabric's *AI services that* can create *Text embeddings* on textualized summaries of each record. So proceeded with FAISS locally.
- **Create Vector Index:** Again, could not register the embeddings in Fabric's AI Catalog or Azure Cognitive Search. So created embeddings in the fabric notebook instead.
- **Create a Semantic Q&A Model:** Now, if I had succeeded in utilizing the Fabric's embeddings and vector index, I could have used Fabric Copilot or Azure OpenAI Service with retrieval plugin configured for the vector index. To substitute this, I wanted to implement this via code in the notebook.
- **Integrate into Dashboard:** Add a **chat visual** in Power BI (Copilot for Power BI) to query natural-language questions directly on top of our gold data. Now, this step was also unsuccessful.

Successful implementation could have had me an **intelligent hydropower assistant** inside Fabric that can:

- Answer questions like *“Show similar performance drops in past years.”*
- Explain anomalies in plain language, grounded in our dataset.
- Generate automated narrative summaries for monthly reports.

All built entirely within the Fabric workspace — using **existing gold tables** as the retrieval base, without moving data outside OneLake. Even though I did not succeeded, I have documented my learnings.

Phase 1 — RAG Implementation Plan

Step 1 – Prepare the Base Data (Lakehouse)

Goal: ensure our Gold tables are clean, timestamp-indexed, and semantically rich.

Actions

- Use existing:
    - gold_cf_deviation (hour-level)
    - gold_cf_deviation_monthly (month-level)
    - gold_zenodo_baseline_monthly (baseline reference)
- Add a descriptive text column for each record, e.g. in a PySpark notebook.

After Phase 1, I had a **prototype RAG system** inside Fabric that:

- Accepts natural-language questions about hydropower performance.
- Retrieves matching time-series slices from our own Lakehouse.
- Generates context-aware explanations and anomaly reports.

All computation stays within OneLake → no external storage or pipelines needed.

Step 2 – Create an AI Workspace / AI Catalog in Fabric

Table	Purpose	Include in RAG?
gold_cf_deviation	✅ **Main dataset** — observed, historical, deviation (hour-level)	**Yes** → use this as **df**
gold_cf_deviation_monthly	Aggregated by month	Optional — for summary or trend explanations
gold_zenodo_baseline_monthly	Long-term climatology baseline	Optional — for background context if we want to retrieve 30-year norms

Created `summary_text` column that was purely numeric. By bringing metadata, I was able to compute national level context for hydropower. Thereafter, I was also able to add seasonal / baseline context using appropriate data source and build enriched summary text. Finally, the **retriever** was able to find relevant time periods *and* understand seasonal and typological context.

Step 3 – Generate Embeddings for Each Record

After this, I converted each record's summary_text into a numerical vector so it can be semantically searched.

Step 4 – Register a Vector Index (keycolumn timestamp)

Step 5 – Create an LLM Endpoint for Retrieval + Generation

Step 6 – Integrate into Power BI / Copilot Experience

These can now be iterated on various use cases:

Scenario	Implementation Path
Q&A Bot for Specific Times	Query = timestamp filter → retrieve hour/day records → summarize.
Pattern Search & Explanation	Retrieve top-K similar embeddings → describe similar historical episodes.
Anomaly Reporting	Detect z-score > 3 → send to LLM for narrative report generation.

While I tried Steps 4 – 6 to accomplish various scenarios/use-cases, I could not succeed. However, I have summarized my learnings as below:

4️⃣ Attempted RAG Implementation in Fabric

The extended objective was to **embed descriptive summaries** of each observation for *semantic retrieval* and *AI-driven insights*.

Implemented steps:

1. **Generated textual summaries** per observation (combining capacity, deviation, and month context).
2. **Created vector embeddings** using Azure OpenAI's text-embedding-3-small model.
3. **Indexed vectors** with FAISS for similarity search on historical-like events.

Encountered challenges:

- Fabric's **AI Foundry deployment region restrictions** (NorwayEast unavailable) prevented Chat model deployment.
- **Schema mismatches and serialization issues** arose when saving embedding arrays to Delta format.
- Attempted joins between FAISS results and metadata failed due to **index-type conflicts (long vs int)**.

These steps nonetheless demonstrated the **blueprint for an intelligent retrieval system** inside Fabric — where a user could query, e.g.,

"Show recent hours when Finland's hydropower deviated most from baseline during autumn months,"
and the system would return semantically matched summaries from similar events.

5️⃣ What Would Have Followed

If the AI service were fully connected:

- A **chat interface** could summarize deviations and explain anomalies using retrieved context ("agentic" reasoning).
- The **Power BI dashboard** would include a natural-language Q&A visual for contextual search across summary_text and embeddings.

This would evolve the lakehouse from static analytics to a **self-explaining, data-aware assistant**.

6️⃣ Key Takeaways

- ✅ Hands-on understanding of **Microsoft Fabric Lakehouse architecture**.
- ✅ Experience integrating **real-time + historical** energy datasets.
- ✅ Implementation of **Spark-based ETL and Delta management**.
- ⚙️ Exposure to embedding generation, vector indexing, and semantic retrieval workflows.
- 💡 Learned practical constraints of **region-restricted AI deployment** and schema consistency for embedding persistence.

In essence:
I successfully established the foundation for a *hydropower intelligence platform* — bridging data engineering, analytics, and AI.
While the RAG portion couldn't be completed due to Fabric's AI region limits, my design demonstrates a scalable path to **agentic time-series analytics**, where natural-language exploration meets data-driven hydropower insights.