Data Ops Technical Challenge – ETL & Serving Pipeline

Benjamin Eduardo De La Torre Rojas
Medellín, Antioquia
(57) 321345549

Crypto Stock ETL

21 of July 2025

1. Executive Summary

This solution proposes a reliable and scalable ETL pipeline that automates the daily extraction, transformation, and loading of cryptocurrency and stock market data into a centralized PostgreSQL database. It uses Apache Airflow for orchestration, Docker for local development, and APIs such as CoinGecko and Alpha Vantage to collect the data. The pipeline is designed to be idempotent and fault tolerant, with retries and alerting in case of failures. Cleaned and curated data is stored in structured tables and can optionally be accessed through a REST API. This setup allows teams to work with accurate and up to date market data to support analytics and decision making.

2. Data Source Overview

2.1. Table Overview

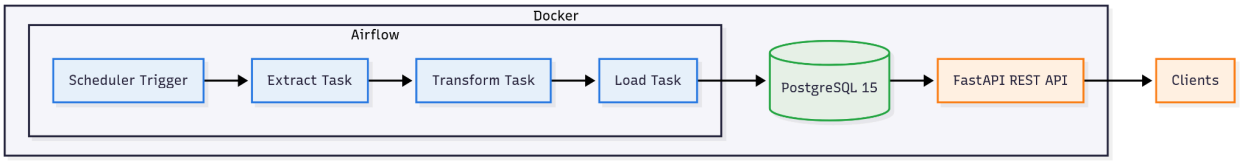
Data Source	Endpoint URL	Format	Frequency	Expected Volume / Rate Limits
CoinGecko (Cryptocurrencies)	https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&ids={coin}	JSON	Daily (0:00 UTC) with optional hourly updates for history	1–2 KB per coin per request, no API key, rate limit: 50 req/minute

Alpha Vantage (Stocks)	https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&symbol={symbol}&apikey=YOUR_API_KEY&datatype=json	JSON	Daily after market close (20:00 UTC)	30–50 KB per symbol per full history request, free tier: 5 req/min, 500 req/day
Yahoo Finance (Alternative Stocks)	https://query1.finance.yahoo.com/v7/finance/download/%7Bsymbol%7D?period1={start}&period2={end}&interval=1d&events=history	CSV	Daily (real time during market hours, final at 20:00 UTC)	10–20 KB per symbol per request, no formal limit.

2.2. Expected Volume & Rate Limits Detail

- **CoinGecko**: Each API call for a coin returns around 1 to 2 KB of data. If we check 4 coins, that's only 4 to 8 KB in total. If we also want to get hourly updates, we could make up to 24 extra calls, adding around 24 to 48 KB. CoinGecko allows up to 50 requests per minute, so our small number of calls per day is completely safe.
- **Alpha Vantage**: For stock data, it returns between 30 and 50 KB per stock. With 4 stocks, we download around 120 to 200 KB per day. The free plan allows 5 requests per minute and 500 per day, so we stay well within the limits.
- **Yahoo Finance**: Gives data as CSV files, with about 10 to 20 KB per stock. With 4 stocks, that's around 40 to 80 KB. They don't have a strict limit, but too many calls too quickly could get blocked. Since we only make a few calls per day, this shouldn't be a problem.

3. High-Level Architecture Diagram



3.1. Component Descriptions

- **Airflow Scheduler & Workers**: Airflow runs the pipeline once a day based on a set schedule. If any task fails, it tries again automatically up to three times. If it still fails, it sends an alert. Airflow also makes sure that each run is safe to repeat, by using parameters and keeping data organized by date.
- **Extract Task**: This task is in charge of getting the data from external APIs like CoinGecko, Alpha Vantage, and Yahoo Finance. For each symbol, it downloads the data and saves it in a raw area (as JSON files or staging tables). Keeping all API calls in one place makes it easier to handle errors, and we also add retry logic with waiting time in case the API has rate limits.
- **Transform Task**: This task takes the raw data and loads it into memory using pandas. It converts the data types, removes any invalid records, and calculates extra values if needed. Keeping this step separate helps us test and improve the data cleaning process without affecting the extract or load steps.
- **Load Task**: This task saves the cleaned data into a PostgreSQL table called curated.price_history. It uses a method that avoids duplicate records by updating existing rows if the same symbol and date already exist. This makes sure that running the pipeline again for the same day won't add the same data twice.
- **PostgreSQL 15+**: The main place where all the data is stored. We use two schemas: one called staging for the raw data, and another called curated for the cleaned and organized tables. We add indexes on the symbol and date columns to make queries faster. For larger datasets, we can also split the data by date to improve performance.
- **FastAPI (Optional)**: provides a simple REST API to access the data. You can list records, get data by symbol and date, or search within a date range. The API uses the path /v1/ so it's easier to add changes in the future without breaking existing code.
- **Docker Compose**: Encapsulates Airflow, PostgreSQL, and the FastAPI service in separate containers. This helps create a consistent local setup that's easy to start and test. It also makes CI/CD processes simpler. The services can talk to each other through a shared Docker network, and environment variables are used to pass settings securely.

3.2. Failure Recovery & Observability

Each task in the pipeline logs useful information such as execution time, task name, and symbol. These logs are automatically collected by Amazon CloudWatch Logs, making it easy to monitor and troubleshoot issues.

Airflow's built-in sensors track the health of key components like the scheduler, workers, and database connection. If any task fails, Airflow retries it up to three times using exponential backoff. Failures that persist trigger alerts via CloudWatch Alarms or notifications through Amazon SNS.

For the API, AWS X-Ray provides end-to-end tracing to detect bottlenecks or downstream failures. Combined with CloudWatch Dashboards, these tools offer full visibility into system performance and reliability—without needing external monitoring tools.

4. Airflow DAG Design

4.1. Dag Overview

We use one DAG called crypto_stock_etl that runs once a day. It processes a list of symbols (like BTC, ETH, AAPL, TSLA), which we can change when needed. The workflow is divided into three main parts for each symbol: Extract, Transform, and Load.

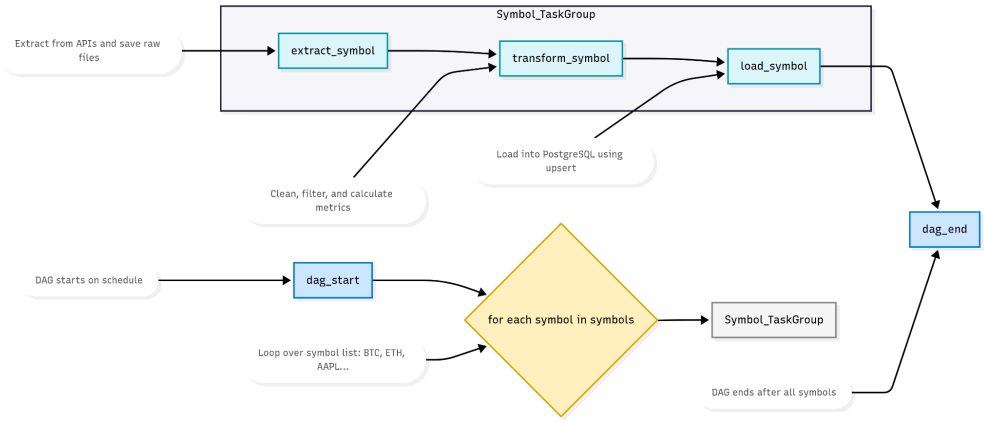
4.2. Schedule & Default Arguments

4.2.1. **Schedule & Default Settings:** The DAG runs every day at 00:00 UTC, which makes sure we get the latest market data after daily updates. If a task fails, it will retry up to 3 times, with a 10-minute wait between each try. If it still fails, the system sends an email (and optionally a Slack message).

4.2.2. **Parameters & Idempotency**: We can pass the list of symbols and optional dates (start and end) through the Airflow UI or configuration. To avoid duplicated data: The extract task saves files by date and symbol (e.g. raw/BTC/2025-07-21.json).
The load task updates existing records using the combination of symbol and date, so running the DAG again doesn't insert duplicates.

4.2.3. **Alerts & Monitoring:** Important tasks have alerts that send messages on failure. We define an SLA (Service Level Agreement) for the final load task, for example, it must finish within 2 hours. If it doesn't, it shows up in Airflow's UI. All tasks write structured logs (INFO level), which can be sent to a central logging system like Elasticsearch and Kibana for easier searching and monitoring.

4.2.4. **Task Graph:**



5. Data Modeling

5.1. Overview

The data model in PostgreSQL 15+ is organized into two schemas: staging and curated.

- The staging schema stores the raw data exactly as it comes from the APIs (in JSON or CSV format). This raw layer is useful for debugging and for reprocessing data if needed. Each record is uniquely identified by the combination of the symbol and the timestamp when the data was fetched.
- The curated schema stores the cleaned and structured data, ready to be used for analytics or reporting. The main table here contains key fields such as symbol, date, open, high, low, close, and volume. Each entry is uniquely identified by the combination of symbol and date, ensuring that there is only one record per asset per day.

5.2. Support Performance & Data Integrity:

- Primary keys are defined in both schemas. In the staging table, the primary key is the combination of symbol and fetch timestamp, which

avoids duplicate raw records. In the curated table, the key is symbol and date, ensuring uniqueness for daily price data.
- Secondary indexes are added on the symbol and date columns in the curated schema. These indexes speed up queries that filter by asset, date, or date ranges, especially useful for dashboards or API calls.
- The curated table is partitioned by date, using monthly ranges. This means that data is physically split into smaller tables under the hood. Partitioning improves query performance as the system only scans the relevant time slices. It also simplifies long-term storage and maintenance, for example, old partitions can be archived or dropped without affecting newer data.
- A step in the Airflow DAG is responsible for automatically creating new partitions at the start of each month. This avoids errors and keeps the system ready for new data.

6. Serving Layer

6.1. Overview

The serving layer exposes our curated price data over HTTPS via a simple RESTful API built with FastAPI. This allows downstream clients (dashboards, trading bots, BI tools) to fetch lists of records, individual records by key, and date-range queries.

6.2. Technology Choice

- **Framework**: FastAPI (Python 3.11 +)
- **Web Server**: Uvicorn/Gunicorn for production
- **Database Driver**: SQLAlchemy + asyncpg for PostgreSQL 15+
- **Containerization**: Included in Docker Compose as an optional api service

6.3. API Endpoints

Method	Path	Desciption	Parameters
GET	/v1/prices?symbol={symbol}&start={start}&end={end}&limit={limit}&offset={offset}	List price records for a symbol within a date range	symbol (required), start (YYYY-MM-DD), end (YYYY-MM-DD), limit, offset
GET	/v1/prices/{symbol}/{date}	Fetch a single price record by symbol and date	symbol, date (YYYY-MM-DD)
GET	/v1/symbols	List all available symbols in the database	–

6.4. Versioning & URL Design

- **Version Prefix**: /v1/ to allow future, non-breaking API evolutions.
- **Resource-Oriented**: Clear noun-based paths (/prices, /symbols).
- **Query Parameters**: Date filters as ISO strings (YYYY-MM-DD).

6.5. Authentication & Security

- Basic Auth or API Key through request headers.
- HTTPS enforced at the ingress/load-balancer layer.
- Rate Limiting can be added via a reverse proxy (e.g., NGINX, Kong) if needed.
- CORS configured to restrict domains in production.

6.6. Error Handling & Validation

- **400 Bad Request** for malformed dates or missing parameters.
- **404 Not Found** when a record doesn't exist.
- **422 Unprocessable Entity** for JSON schema validation errors.
- **500 Internal Server Error** logged centrally, with generic message to clients.

7. Observability & Alerting

To monitor the health and performance of the pipeline, we include several observability mechanisms that cover logging, metrics, SLAs, and system health checks.

7.1. Logging

All Python tasks involved in the ETL process, extraction, transformation, and loading, generate structured logs in JSON format. These logs include important context such as the asset symbol, task ID, execution time, and run date. Logs are stored in a centralized logging system, such as Elasticsearch and Kibana or a managed alternative like AWS CloudWatch Logs.

Airflow's built-in logging system is also configured to send logs from each task to the central log storage. Proper rotation and retention policies are applied to manage storage usage over time.

7.2. Metrics

We collect pipeline metrics using Amazon CloudWatch. Each Airflow task logs structured messages that can be filtered into custom metrics, such as records processed, task durations, and failure counts. For the API, AWS X-Ray tracks request

traces, response times, and errors across endpoints. Metrics are visualized in CloudWatch Dashboards to monitor trends and detect issues.

7.3. SLAs & Alerts

Critical tasks include SLAs to ensure timely completion (e.g., within 2 hours). If a task fails or misses its SLA, alerts are sent through CloudWatch Alarms via email or SNS notifications. Airflow also includes callbacks that can trigger alerts on failure. Repeated issues can be escalated using tools like AWS Chatbot or PagerDuty.

7.4. Health Checks

All services expose basic health checks:

- Airflow monitors its own components and database using built-in sensors.
- FastAPI includes a /health endpoint that verifies connectivity and readiness.
- Docker containers have internal health checks that report to CloudWatch.
- AWS X-Ray gives deeper insight into system health by tracing API calls and downstream services.

8. Infrastructure & CI/CD

8.1. Development Environment

The local development setup uses Docker Compose to quickly start all services: Airflow, PostgreSQL, and FastAPI, with a single command. This allows developers to test and iterate locally in a consistent environment. The codebase includes configuration for live code reloading and automatic dependency setup, using Python 3.11 and a virtual environment manager such as Poetry or Pipenv.

8.2. Environment Variables & Secrets

Environment variables are used to manage configurations across local and production environments:

- Local development uses a .env file based on a shared .env.example, containing non-sensitive defaults like database names and Airflow settings.
- In production, sensitive values such as database passwords and API keys are managed through AWS Secrets Manager or Airflow's Secrets Backend. This ensures secrets are never stored in code or Docker images.
- Configuration values follow a layered approach: defaults in code, overrides via environment variables, and runtime parameters through Airflow Variables and Connections.

8.3. Deployment Outline

The codebase follows a Git-based workflow, using feature branches and pull requests. CI is handled via GitHub Actions, which automatically run tests and linters on every commit. When changes are merged into the main branch, the pipeline builds and pushes Docker images to a container registry.

There are multiple options for deployment:

- **Cloud-native**: The system can be deployed to AWS using services like:
  - Amazon MWAA (Managed Workflows for Apache Airflow) for orchestration.
  - Amazon RDS or Aurora for PostgreSQL.
  - Amazon ECS or EKS for running the API if needed.
- **Alternative (self-managed)**: For teams running their own infrastructure, Docker Compose or Kubernetes with Helm can be used. CI/CD scripts can deploy new DAGs and restart services via SSH or automated helm upgrades.

9. Security & Compliance

9.1. Data Encryption (TLS)

All traffic between services and user interfaces (like Airflow and the API) is secured using TLS 1.2+. In development, we use self-signed certificates. In production, certificates are provisioned via AWS Certificate Manager, and HTTPS is enforced using Application Load Balancers (ALB) or API Gateway.

Database connections (PostgreSQL) also require encryption. Clients like Airflow and FastAPI connect using SSL with verified certificates.

9.2. Secrets Management

In local development, non-sensitive values are stored in .env files ignored by Git. For production, all secrets, including database credentials and API keys are managed securely with AWS Secrets Manager.

- Airflow uses its Secrets Backend to fetch values at runtime.
- FastAPI loads secrets from environment variables injected by the task runner or container platform.

Secrets are rotated regularly (e.g., every 90 days), and access logs are audited via CloudTrail or Secrets Manager logs.

9.3. Least-Privilege Access

Database roles are restricted based on purpose:

- Ingestion tasks can only insert into staging tables.
- Transformation and load tasks can read from staging and write to curated tables.
- The API can only read from curated data.
- Admin privileges are limited to CI/CD pipelines or DBAs for schema changes.

Airflow and FastAPI each connect using specific roles with the minimum required permissions. No service uses super user access.

9.4. Network Isolation

In cloud deployments, services like PostgreSQL and Airflow workers are placed in private subnets. Only public facing components like the Airflow web UI or API endpoints are accessible through a load balancer with firewall rules (via Security Groups) that restrict access by IP or CIDR.

Port access is limited:

- Database ports (5432) are only open to trusted internal services.
- API ports (8000) are only exposed through secure public endpoints.

9.5. Compliance & Audit Logging

Changes to curated data are logged by enabling PostgreSQL query logging, and all access to secrets or infrastructure is tracked using AWS CloudTrail. These logs can be sent to a centralized system (like Amazon S3) for long-term storage and monitoring.

The system follows security best practices (OWASP Top 10 for APIs), and all controls are documented as part of the project's internal review process.