Overzicht

Doel: een driedelige architectuur waarmee je lokaal of in de cloud het ophalen, opslaan en analyseren van Emply-jobdata kunt simuleren en automatiseren:

A. Dummy Emply-service (mock API)
B. Data-ingest applicatie (scheduler + opslag)
C. Analyselaag (BigQuery + dbt)

De drie delen communiceren uitsluitend via REST API's en databases.

A. Dummy Emply-service (Mock API)

Doel

Een Python-service die zich gedraagt als Emply's publieke API, maar eigen data uit een lokale database teruggeeft.
De endpoints en authenticatie moeten **identiek lijken aan** de echte Emply API.

Technische richting

Framework: FastAPI (asynchroon, genereert automatisch een OpenAPI-spec).
Databank: SQLite of PostgreSQL voor testdata.
Endpoints:
GET /v1/{customer}/postings/{mediaId} → retourneert vacatures.
GET /v1/{customer}/applications/find-by-date → retourneert sollicitaties binnen een datumbereik.
Beveiliging:
Header x-api-key moet aanwezig zijn, maar hoeft niet gevalideerd te worden.
Als de key ontbreekt, HTTP 401; anders doorgaan.
Dit bootst de "Authorize"-functionaliteit uit Emply's Swagger na.
Responsdata:
JSON-structuur gelijk aan die van de echte API (zoals beschreven op api.emply.com/index.html).
Data afkomstig uit een lokale tabel postings en applications.
postings bevatten velden als id, title, location, status, published_at.
applications bevatten id, posting_id, candidate_name, date_applied.

Gebruik

Start lokaal op http://localhost:8000.
Teruggegeven JSON bootst Emply exact na.
Kan later eenvoudig worden uitgebreid met extra endpoints.

B. Scheduler-applicatie (ETL-laag)

Doel

Een afzonderlijke Python-app die:

Elk uur data ophaalt van de dummy Emply API.
De data opslaat in zijn eigen relationele database.
Duplicaten voorkomt (idempotent gedrag).

Technische richting

Framework:
Backend: FastAPI (optioneel, alleen voor handmatige triggers).
Scheduler: APScheduler of Celery Beat (cronjobs in Python).
Data-opslag: PostgreSQL of MySQL.
Authenticatie:
Dummy Emply API aanroepen met x-api-key header.
Flow:
Scheduler roept GET /postings/{mediaId} op.
Scheduler roept GET /applications/find-by-date?from=X&to=Y op.
Nieuwe records worden vergeleken met bestaande IDs.
Nieuwe of gewijzigde data wordt geüpdatet.
Logging:
Iedere run logt starttijd, eindtijd, aantal records en eventuele fouten.
Configuratie:
API-base-URL, API-key, customer-id, mediaId.
Sync-interval (standaard elk uur).
Database-URL.

Resultaat

Een lokale opslaglaag met persistente data, gesynchroniseerd vanuit de mock Emply.

C. BigQuery + dbt Analyse

Doel

De verzamelde data analyseren en visualiseren.

Technische richting

BigQuery
Data uit de scheduler-database wordt dagelijks geëxporteerd of gesynchroniseerd.
Tabellen: job_posting, application, candidate.
Gebruik bq load of een Airflow/dbt job om nieuwe data bij te werken.
dbt (Data Build Tool)
Definieer modellen in /models:
stg_postings.sql → gestandaardiseerde staging views.
stg_applications.sql → normalisatie van application data.
fct_applications_per_posting.sql → aggregatie per vacature.
Gebruik dbt run voor transformatie en dbt test voor kwaliteitschecks.
Dashboards (optioneel)
Koppel Google Looker Studio of Metabase aan BigQuery.
Visualiseer o.a.:
Aantal sollicitaties per vacature.
Aantal sollicitaties per dag/week.
Gemiddelde sollicitatieduur.
Toplocaties of populairste vacatures.

Datastroom (End-to-End)

Mock Emply (A) → levert JSON-data bij elke GET-aanroep.
Scheduler-app (B) → haalt deze data elk uur op en slaat het lokaal op.
BigQuery + dbt (C) → leest en analyseert deze data periodiek.