VISIUM

Rayner Däppen

Data Engineer - Job Application

Covid-19 Use Case

Roadmap

1. Define project goals
2. Gather requirements
3. Create/Define processes and tasks (ETL, DHW, Documentation)
4. Design solution
5. Evaluate tools
6. Implementation
7. Testing

Project Goal

In the middle of the covid-19 crisis, the Swiss government asked the company you are working for to build for them a tool which would help them analyse the evolution of the pandemic and better understand the effect of the different measures they are taking.

As the Data Engineer of the company, you have receive as a task to build from scratch a data warehouse in order to daily store, for some cities in Switzerland, a set of information.

Requirements

- Spending time in requirements gathering will help us saving implementation time and avoid developing things that are not needed.

Covid Datawarehouse - Functional Requirements

Points of discussion not flashed out in initial descriptions

Data Governance		
Requirement Name	Description	Priority
Scope		
Roles & Responsabilities		
Data Catalog	The solution contains a metadata management tool to organize and describe the data contained wihin the system	Medium

Data Processes & Tools		
Requirement Name	Description	Priority
Querying	The solution allows users to request records, make updates to the records without maintaining a connection to the data source, and then send the record updates back to the data source at some other time. This process needs to take place daily	High
Data Sources	The solution can query and store data from multiple types of data sources (Traffic, Weather, Covid-19).	High
Packaging	The solution is packaged within docker containers and accessed through docker-compose	Low

Data architecture and quality		
Requirement Name	Description	Priority
Pipelines	The solution contains end-end data pipelines that allows data processing within regular intervals of time	High
Documentation	The solution presents a clear and defined documentation containing all the important processes (pipelines, DHW design, etc)	High

Data Security and Privacy		
Requirement Name	Description	Priority
Access Management	The solution allows users to define different access policies according to roles and reponsabilities	Low

Covid Datawarehouse - Non Functional Requirements

name	Description	Priority
Granularity	The solution stores the highest possible granularity that the different sources provide	High
Quality	The solution ensures the highest quality of the data posible in terms of design, harmonisation, quality, etc.	High
Scalability	The solution takes into consideration future expansions of the project	Medium
Cloud		Low

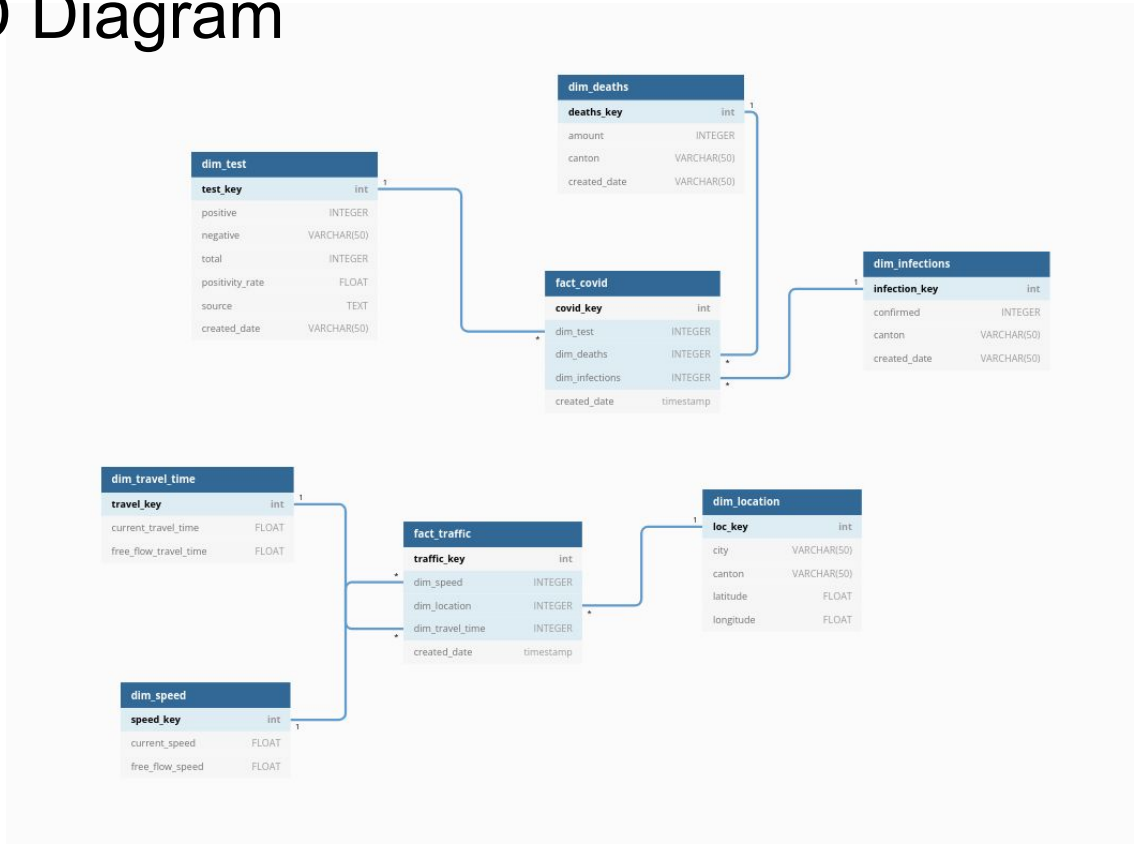
Process - Data Warehouse

1. Requirements Gathering
2. Environment Setup: make a proper environment setup for development, testing, and production.
    a. In this case I choosed Jupyter Notebooks, Airflow, Postgres and Metabase
3. Data Modelling: design how to connect the data source, process, and store in the data warehouse.
4. ETL Implementation

Source: educba

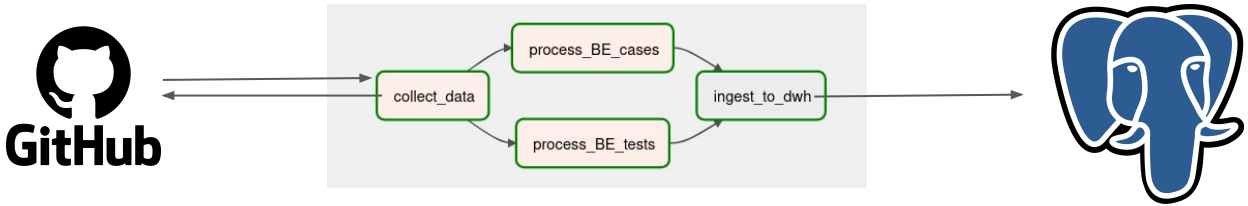
Design Process: ERD Diagram

- 2 Fact Tables: covid & traffic
- Unfortunately data is too heterogeneous and in OpenZH github's repo there is no city information.
- Tables can be related by the canton dimension
- A bigger effort would be needed to harmonise both stars for analysis.



Process - Data Pipeline Implementation

1. Draw a high-level plan: Design a draft of your installation
2. Choose an ETL tool: Depending on requirements
3. Develop default strategies: Applicable to all data sources, for all targets
4. Drill down by target table: Set up a detailed planning, per target.



Source: Berner Fachhochschule - Data Engineering Specialization

Tool Evaluation

- A comprehensive set of requirements help us to define what tool is better for the job.
- In this case:
  - DWH - Postgres
  - ETL - Airflow

Requirement	Data Warehousing			ETL & Automation	
	Hive	Druid	Postgres	Airflow	Dagster
Querying	x	x	x	-	-
Data Sources	x	x	x	-	-
Analysis		x	x	-	-
Packaging	x	x	x	-	-
Pipelines	-	-	-	x	x
Documentation	-	-	-	-	-
Access Management	x	x	x	x	x
Granularity	x	x	x	x	x
Scalability	x	x	~	x	x
Cloud	-	-	-	-	x

Implementation - DHW

- Postgres as a main tool
- DHW instantiated by a DAG task

```python
dag = airflow.DAG(
    'init_dwh_conn',
    schedule_interval="@once",
    default_args=args,
    max_active_runs=1)

t1 = PythonOperator(task_id='initialize_dwh_conn',
                    python_callable=initialize_dwh_conn,
                    dag=dag)

t2 = PostgresOperator(task_id='build-tables',
                      postgres_conn_id = 'postgres_dwh',
                      autocommit = True,
                      sql = 'sql/dwh_tables.sql',
                      dag = dag)

t1 >> t2
```

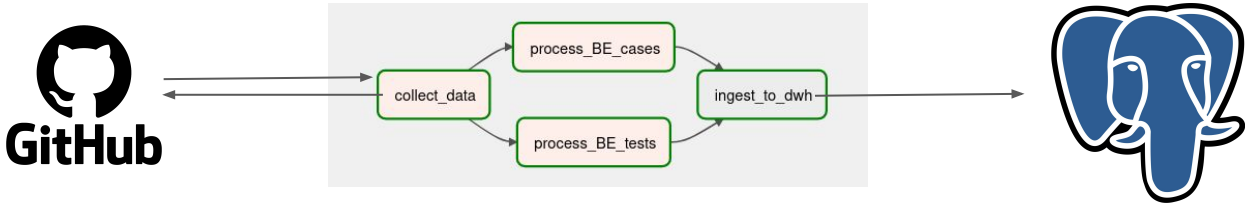
./dags/dag_dwh.py

```sql
-- Covid Star

DROP TABLE IF EXISTS fact_covid;
DROP TABLE IF EXISTS dim_test;
DROP TABLE IF EXISTS dim_deaths;
DROP TABLE IF EXISTS dim_infections;

CREATE TABLE dim_test (
  test_key    SERIAL PRIMARY KEY,
  positive    INTEGER,
  negative    VARCHAR(50),
  total       INTEGER,
  positivity_rate FLOAT,
  source          TEXT,
  created_date VARCHAR(50)
);

CREATE TABLE dim_deaths (
  deaths_key SERIAL PRIMARY KEY,
  amount INTEGER,
  canton VARCHAR(50),
  created_date VARCHAR(50)
);

CREATE TABLE dim_infections (
  infection_key SERIAL PRIMARY KEY,
  confirmed INTEGER,
  canton VARCHAR(50),
  created_date VARCHAR(50)
);


CREATE TABLE fact_covid (
    covid_key     SERIAL PRIMARY KEY,
    dim_test      INTEGER,
    dim_deaths    INTEGER,
    dim_infections INTEGER,
    created_date TIMESTAMP NOT NULL DEFAULT current_timestamp,
```

./dags/sql dwh_tables.sql

Implementation - Data Pipelines

- 2 DAGs for the data warehouse data pipelines



./dags/covid_ingestion.py

- 4 Tasks.
  - collect-data -> Queries the OpenZH github repository
  - [process_BE_cases, process_BE_tests] -> Prepare the data for ingestion
  - ingest_to_dwh -> connects to the warehouse and persists the Postgres database

Implementation - Data Pipelines - Collect Data

- Used requests to query all the addresses in the repo
- Gathering all the data from the repository would take too long. Solution: limited the example to Bern.
- Used xcom for cross task communication

```python
##### Main Python Data Pipeline Tasks

#Create requests to the different sites in the OpenZH github repository i
def collect_data(**op_kwargs):
    cases_BE = requests.get(covid_cases_BE)
    cases_ZH = requests.get(covid_cases_ZH)
    cases_VD = requests.get(covid_cases_VD)

    tests_BE = requests.get(covid_tests_BE)
    tests_ZH = requests.get(covid_tests_ZH)
    tests_VD = requests.get(covid_tests_VD)

    data = {}

    data['BE'] = {"cases": cases_BE.text, "tests": tests_BE.text}
    #data['ZH'] = (cases_ZH.text, tests_ZH.text)
    #data['VD'] = (cases_VD.text, cases_VD.text)



    return data
```

./dags/src/CovidHelper.py

Implementation - Data Pipelines - Collect Data

- Pulled data from xcom.
- Prepared the data in a dictionary
- Pushed xcom for next task

```python
def process_BE_cases(ti):
    data = ti.xcom_pull(task_ids="collect_data", key="return_value")

    bern_data = data["BE"]

    #Converting to StringIO so it can be read by pandas as dataframe
    str_cases = StringIO(bern_data["cases"])
    csv_cases = pd.read_csv(str_cases, header=0)

    #Sorting the dataframe

    csv_cases = csv_cases.sort_values(by=['date'], ascending=False)

    # In the case of the covid cases, the data only presents cumulative numbers. We are going to ca

    cases_yesterday = csv_cases.iloc[1]
    cases_today = csv_cases.iloc[0]

    cases_today['infections'] = cases_today['ncumul_conf'] - cases_yesterday['ncumul_conf']
    cases_today['deaths'] = cases_today['ncumul_deceased'] - cases_yesterday['ncumul_deceased']

    cases_today['infections'] = cases_today['ncumul_conf'] - cases_yesterday['ncumul_conf']
    cases_today['deaths'] = cases_today['ncumul_deceased'] - cases_yesterday['ncumul_deceased']

    #locating cases
    cases_today = cases_today.to_dict()

    #Prepating data to ingest the data warehouse

    export = {
    "date" : cases_today['date'],
    "deaths": cases_today['deaths'],
    "infections": cases_today['infections'],
    "canton": "BE",
    }

    # Pushing data for cross process communications
    ti.xcom_push(key='date', value = export['date'])
```

./dags/src/CovidHelper.py

Implementation - Data Pipelines - Ingest DWH

- Used xcom for reaching the data
- Jinja Templating allows to use xcom for ingesting directly using a sql script.

./dags/dag_covid.py

```python
t3 = PostgresOperator(
        task_id="ingest_to_dwh",
        postgres_conn_id='postgres_dwh',
        sql="sql/ingest_BE_COVID.sql",
        dag=dag
    )
```

```
9   INSERT INTO dim_deaths VALUES
10      (DEFAULT,
11      {{ task_instance.xcom_pull(task_ids='process_BE_cases', key='deaths')}},
12      '{{ task_instance.xcom_pull(task_ids='process_BE_cases', key='canton')}}',
13      '{{ task_instance.xcom_pull(task_ids='process_BE_cases', key='date')}}'
14    ) RETURNING "deaths_key" INTO dim_deaths;
15
16  INSERT INTO dim_infections VALUES
```

./dags/sql/ingest_BE_COVID.sql

Testing

- Airflow DAGs can be tested using any of the python test packages.

```python
import pytest
from airflow.models import DagBag

def test_no_import_errors():
  dag_bag = DagBag()
  assert len(dag_bag.import_errors) == 0, "No Import Failures"
```

- Different types of tests can be carried out: DAG validation testing, unit testing and data integrity testing.
- These tests can be integrated in CD/CI pipelines

Source: [astronomer.io]

Questions?

VISIUM

Backup

Process - Documentation

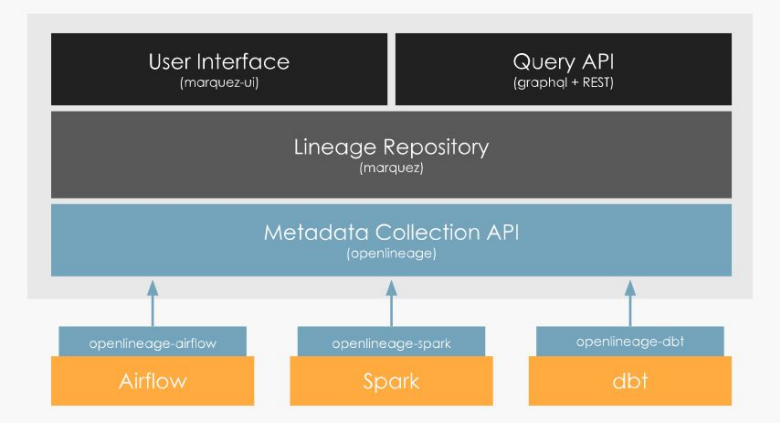
1. Group Common Dimensions
2. Group Facts into Processes
3. Provide narrative for each process
4. Document Table Relations
5. Visualize Data Model with ERD
6. Document primary and business keys
7. Describe tables
    a. Path to table
    b. Primary key
    c. Data Type
    d. Null Values
    e. Constraints
    f. Succinct description

Source: Kimball DWH Best Practices.

Data Lineage

- Understand how data is transformed across the ecosystem, achieve observability and monitor pipelines.
- Understand how the dependencies between different stages of the data pipelining process.
- What's the impact of a change in the downstream pipeline?

```
from marquez_airflow import DAG
```



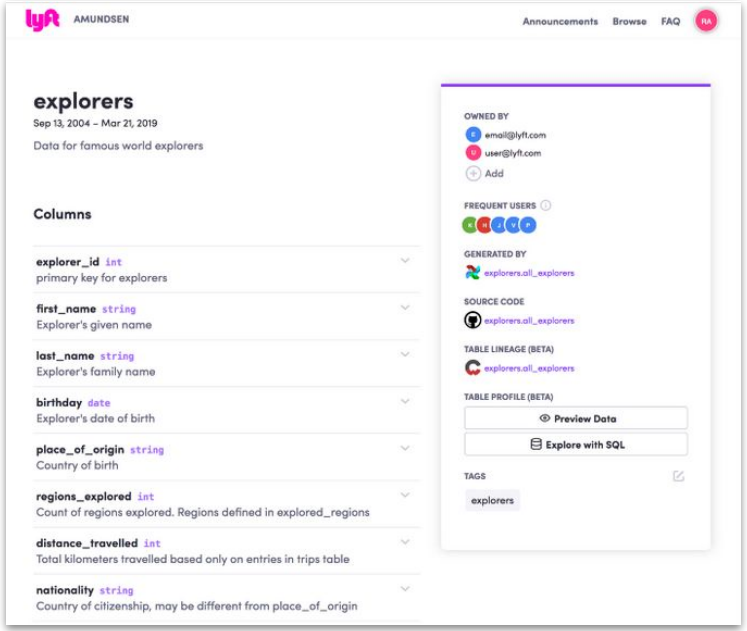
Tools: Marquez, Open Lineage.

Data Catalogue

Problem: how is data used within an organization? What data is being collected? How to break the silos?

Solution: single place where all of your organization's data can be catalogued, enriched, searched, tracked and prioritized.

Tools: Magda, Amundsen



Amundsen

Source: Amundsen — Lyft's data discovery & metadata engine