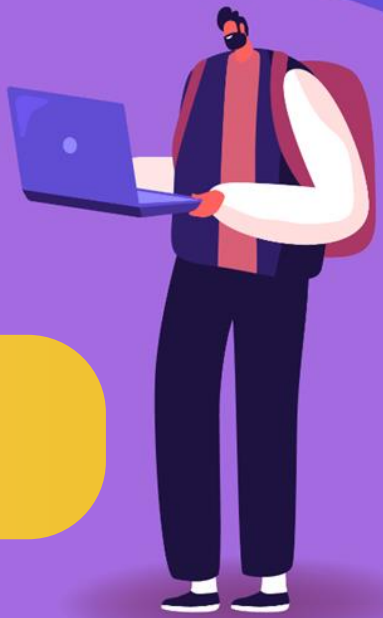
CERTIFIED

DATA ENGINEER

Narasio DATA



Bitcoin Price Index ETL

API Source (Case 2)

Agenda

- Problem Statement
- Tools & Stack
- Workflow & Considerations
- Configurations
- Bonus Content!

Problem Statement

CoinDesk BPI - ETL Pipeline

1. **Extract** data dari API BPI (yang diupdate setiap menit)

2. **Transform**, bersihkan, dan tambahkan informasi IDR dalam data

3. **Load** ke Postgres/BigQuery

Lakukan secara *hourly*

Problem Statement

CoinDesk BPI - ETL Pipeline

Menggunakan **Airflow Docker**...

1. **Extract** data dari API BPI (yang diupdate setiap menit) + **API OER USD-IDR**
2. **Transform**, bersihkan, dan tambahkan informasi IDR dalam data
3. **Load** ke **BigQuery**

Lakukan secara *hourly*

Secara lebih spesifik

Summary

Tools & Stack

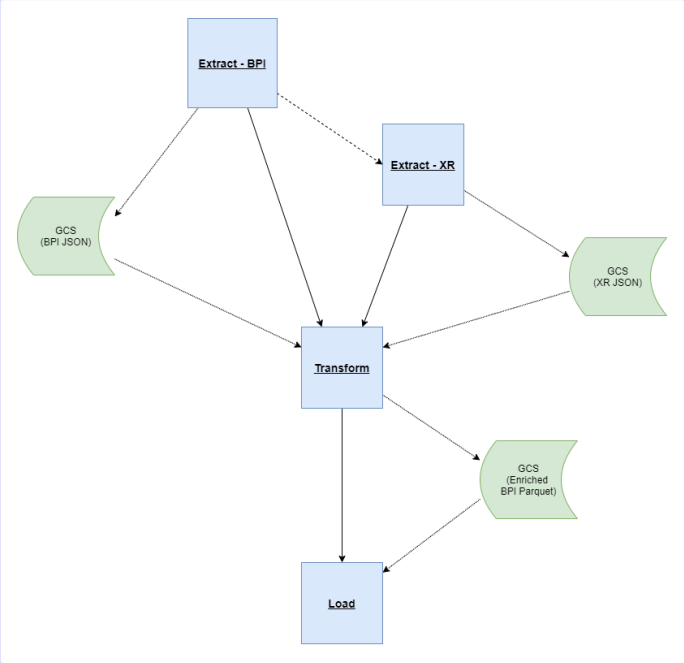
- **Airflow**
  Docker Desktop (Compose)
- **GCP services**
  Google Cloud Storage, BigQuery
- **GCP operations**
  apache-airflow-providers-google/Google Cloud Client
- **Other libraries***
  requests, pandas, fastparquet, pendulum

** Hanya menyebutkan library non-standard*

Airflow Steps

- **extract_bpi()**
  Untuk ekstraksi Bitcoin Price Index (BPI), berisi kurs konversi Bitcoin dari API CoinDesk.
- **extract_xr()**
  Untuk ekstraksi kurs *hourly* USD-IDR terbaru, dari API Open Exchange Rates.
- **transform_data()**
  Untuk membersihkan, mentransformasi, dan memperkaya data BPI dengan tambahan data kurs IDR.
- **load_data()**
  Untuk memasukkan data BPI yang telah ditransformasi ke BigQuery.

Workflow



Benefits

- **Mengisolasi setiap *step***

Jika salah satu *step* gagal, *workflow* didesain untuk *graceful recovery* dan *step* tersebut dapat menggunakan informasi dari step sebelumnya untuk *re-run* yang *smooth*.

- **Data tersimpan di setiap *step***

Eksekusi *workflow* bisa *distributed*, *step transform* dan *load* bisa dilakukan kapan saja, serta mudah untuk *tracing* jika ada *error* data.

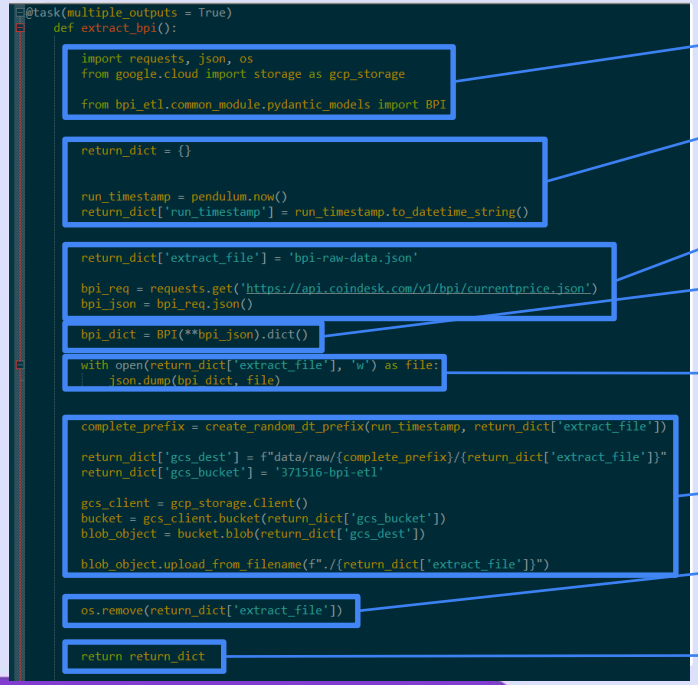
- ***Step extract* XR bisa menyesuaikan dari *step extract* BPI**

Aspek yang cukup penting karena kedua *step extract* ini bersifat cukup *time-sensitive*.

Note: *Jika dilihat workflownya, sesungguhnya mengeliminasi kemungkinan menggunakan operator bawaan Airflow*

Step *Extract*

Step 1: Extract BPI



```python
@task(multiple_outputs = True)
def extract_bpi():

    import requests, json, os
    from google.cloud import storage as gcp_storage

    from bpi_etl.common_module.pydantic_models import BPI

    return_dict = {}


    run_timestamp = pendulum.now()
    return_dict['run_timestamp'] = run_timestamp.to_datetime_string()

    return_dict['extract_file'] = 'bpi-raw-data.json'

    bpi_req = requests.get('https://api.coindesk.com/v1/bpi/currentprice.json')
    bpi_json = bpi_req.json()

    bpi_dict = BPI(**bpi_json).dict()

    with open(return_dict['extract_file'], 'w') as file:
        json.dump(bpi_dict, file)


    complete_prefix = create_random_dt_prefix(run_timestamp, return_dict['extract_file'])

    return_dict['gcs_dest'] = f"data/raw/{complete_prefix}/{return_dict['extract_file']}"
    return_dict['gcs_bucket'] = '371516-bpi-etl'

    gcs_client = gcp_storage.Client()
    bucket = gcs_client.bucket(return_dict['gcs_bucket'])
    blob_object = bucket.blob(return_dict['gcs_dest'])

    blob_object.upload_from_filename(f"./{return_dict['extract_file']}")

    os.remove(return_dict['extract_file'])

    return return_dict
```

1. **Import *library***
2. **Menyiapkan informasi** untuk *step* selanjutnya
3. **Mengambil data** dari API CoinDesk BPI
4. **Melakukan validasi dan konversi** dengan Pydantic
5. **Menyimpan data sebagai JSON** (sesuai format aslinya)
6. **Menyimpan data ke GCS**
7. **Menghapus data** yang sudah disimpan ke GCS
8. ***Return* informasi** untuk *step* selanjutnya

Step 1: Extract BPI

```
return_dict['extract_file'] = 'bpi-raw-data.json'

bpi_req = requests.get('https://api.coindesk.com/v1/bpi/currentprice.json')
bpi_json = bpi_req.json()

bpi_dict = BPI(**bpi_json).dict()

with open(return_dict['extract_file'], 'w') as file:
    json.dump(bpi_dict, file)
```

API BPI CoinDesk

- **API BPI CoinDesk memiliki dua jenis besar:**
  1. Versi *current price* yang dihitung setiap menit
  2. Versi *historical* yang mengembalikan data 31 hari terakhir
- **Digunakan API *current price***, sesuai dengan instruksi dalam *case*.
- Karena **data berubah setiap menit**, maka API ini bersifat ***time-sensitive*** dan riskan jika ada *error/re-run*.
- **Kurs yang disediakan** pada *endpoint* API ini adalah **USD, EUR, dan GBP**.

Step 2: Extract XR

```python
@task(multiple_outputs = True)
def extract_xr(fetch_timestamp):

    from airflow.models import Variable

    import requests, json, os
    from google.cloud import storage as gcp_storage

    from bpi_etl.common_module.pydantic_models import HistoricalXR

    fetch_datetime = pendulum.parse(fetch_timestamp)

    return_dict = {}

    return_dict['extract_file'] = 'rupiah-exchange-rate.json'

    xr_fetch_date = fetch_datetime.strftime('%Y-%m-%d')

    auth_params = {
        'app_id': Variable.get('oer_api_key'),
        'symbols': 'IDR'
    }

    IDR_xr_url = f'https://openexchangerates.org/api/historical/{xr_fetch_date}.json'
    IDR_xr_req = requests.get(IDR_xr_url, params = auth_params)

    IDR_xr_json = IDR_xr_req.json()

    IDR_xr_dict = HistoricalXR(**IDR_xr_json).dict()

    with open(return_dict['extract_file'], 'w') as file:
        json.dump(IDR_xr_dict, file)

    complete_prefix = create_random_dt_prefix(pendulum.now(), return_dict['extract_file'])

    return_dict['gcs_dest'] = f"data/raw/{complete_prefix}/{return_dict['extract_file']}"
    return_dict['gcs_bucket'] = '371516-bpi-etl'

    gcs_client = gcp_storage.Client()
    bucket = gcs_client.bucket(return_dict['gcs_bucket'])
    blob_object = bucket.blob(return_dict['gcs_dest'])

    blob_object.upload_from_filename(f"./{return_dict['extract_file']}")

    os.remove(return_dict['extract_file'])

    return return_dict
```

1. **Import *library***
2. **(NEW!)* Membaca informasi dari *step* sebelumnya**
3. **Menyiapkan informasi** untuk *step* selanjutnya
4. **Mengambil data** dari API Open Exchange Rates
5. **Melakukan validasi dan konversi** dengan Pydantic
6. **Menyimpan data sebagai JSON** (sesuai format aslinya)
7. **Menyimpan data ke GCS**
8. **Menghapus data** yang sudah disimpan ke GCS
9. ***Return* informasi** untuk *step* selanjutnya

** Berbeda dari step extract BPI*

Step 2: Extract XR

```
return_dict['extract_file'] = 'rupiah-exchange-rate.json'

xr_fetch_date = fetch_datetime.strftime('%Y-%m-%d')

auth_params = {
    'app_id': Variable.get('oer_api_key'),
    'symbols': 'IDR'
}

IDR_xr_url = f'https://openexchangerates.org/api/historical/{xr_fetch_date}.json'
IDR_xr_req = requests.get(IDR_xr_url, params = auth_params)

IDR_xr_json = IDR_xr_req.json()

IDR_xr_dict = HistoricalXR(**IDR_xr_json).dict()

with open(return_dict['extract_file'], 'w') as file:
    json.dump(IDR_xr_dict, file)
```

API Open Exchange Rates (OER)

- **API ini dipilih** karena merupakan **API yang sama** dengan API konversi kurs yang **dipakai CoinDesk** (API BPI)
- **Digunakan akses *free plan*** dari OER, yang memiliki **akses data *hourly*** sejumlah **1,000 *request* per bulan.**
- **API Open Exchange Rates juga memiliki dua jenis:**
  1. Versi *latest* yang memiliki kurs tiap jam
  2. Versi *historical*
- **Digunakan API *historical***, berfungsi **sama persis dengan API *latest*** jika di hari yang sama, **sekaligus berfungsi sebagai *fallback*** jika ada *re-run* di hari yang berbeda.
- **Bisa pilih kurs spesifik** dengan parameter GET (dalam kasus ini, hanya mengambil USD-IDR untuk penghematan *resource*).

Step *Transform*

Step 3: Transform Data

```python
@task(multiple_outputs = True)
def transform_data(bpi_data_loc, xr_data_loc):

    import json, os
    import pandas as pd

    from google.cloud import storage as gcp_storage

    gcs_client = gcp_storage.Client()

    return_dict = {}

    for file_loc in [bpi_data_loc, xr_data_loc]:
        bucket = gcs_client.bucket(file_loc['gcs_bucket'])
        blob_object = bucket.blob(file_loc['gcs_dest'])

        blob_object.download_to_filename(f"./{file_loc['extract_file']}")

    with open(bpi_data_loc['extract_file'], 'r') as file:
        bpi_json = json.load(file)

    with open(xr_data_loc['extract_file'], 'r') as file:
        xr_json = json.load(file)
```

1. **Import dan set-up *library***
2. **Menyiapkan informasi** untuk *step* selanjutnya
3. **Mengambil dan membaca data** dari langkah-langkah sebelumnya dari Google Cloud Storage

Step 3: Transform Data

4. **Mengubah nama dan urutan kolom** sesuai instruksi dalam *case study**
5. **Melengkapi data** dengan kurs **BTC-IDR**
6. **Mengubah format tanggal** sesuai dengan instruksi dalam *case study*
7. **Menambah kolom** `last_updated` pada data

** Comment detail ada pada source code asli*

```
final_meta_columns = ['disclaimer', 'chartName', 'time.updated', 'time.updatedISO']

final_metadata_regex = '|'.join(['code', 'rate_float', 'description'])
final_currency_filter = bpi_df.columns.\
    str.contains(f'^bpi\\.(?:.*).{final_metadata_regex}$', regex = True)
final_currency_columns = bpi_df.columns[final_currency_filter].tolist()

final_columns = final_meta_columns + final_currency_columns

bpi_df = bpi_df.loc[:, bpi_df.columns.isin(final_columns)]


bpi_df.columns = bpi_df.columns.\
str.replace('.', '_', regex = False).\
str.replace('([a-z])([A-Z])', '\\1_\\2', regex = True).\
str.lower()
```

```
USD_IDR_rate = xr_json['rates']['IDR']

bpi_df['bpi_idr_rate_float'] = bpi_df['bpi_usd_rate_float'] * USD_IDR_rate
```

```
date_format = '%Y-%m-%d %H:%M:%S'
```

```
bpi_df[['time_updated', 'time_updated_iso']] = bpi_df[['time_updated', 'time_updated_iso']].\
    apply(lambda timestamp: pd.to_datetime(timestamp).dt.strftime(date_format))

bpi_df['last_updated'] = pendulum.now().strftime(date_format)
```

Step 3: Transform Data

```
return_dict['extract_file'] = 'bpi-xr-data.parquet'

bpi_df.to_parquet(return_dict['extract_file'], index = False, engine = 'fastparquet', has_nulls
= ['disclaimer', 'chart_name'])

complete_prefix = create_random_dt_prefix(pendulum.now(), return_dict['extract_file'])

return_dict['gcs_dest'] = f"data/bigquery_load/{complete_prefix}/{return_dict['extract_file']}"
return_dict['gcs_bucket'] = '371516-bpi-etl'

blob_object = bucket.blob(return_dict['gcs_dest'])

blob_object.upload_from_filename(f"./{return_dict['extract_file']}")

files_to_remove = [return_dict['extract_file'], bpi_data_loc['extract_file'],
                   xr_data_loc['extract_file']]

for file in files_to_remove:
    os.remove(file)

return return_dict
```

8. **Menyimpan data** sebagai **parquet** menggunakan *library* `fastparquet`
9. **Menyimpan data ke GCS**
10. **Menghapus data** yang sudah tidak dipakai atau telah disimpan ke GCS
11. ***Return* informasi** untuk *step* selanjutnya

Step 3: Transform Data


```
return_dict['extract_file'] = 'bpi-xr-data.parquet'

bpi_df.to_parquet(return_dict['extract_file'], index = False, engine =
'fastparquet', has_nulls = ['disclaimer', 'chart_name'])
```

Mengapa `fastparquet` dan BigQuery?

- **Parquet dipilih** karena memiliki ***trade-off*** *data storage* yang **paling sesuai** untuk penggunaan kita.
- **Kita dapat menentukan kolom REQUIRED dan NULLABLE** dalam membuat skema tabel di **BigQuery**
- **Parquet** yang akan **diunggah** ke BigQuery **WAJIB** memiliki **informasi *nullable* yang sama** dengan skema **di BigQuery** (atau akan mendapatkan *error* menarik!).
- Agar dapat menspesifikkan **informasi nullable secara presisi**, maka **perlu digunakan** `fastparquet` dengan argument `has_nulls`.

Step *Load*

Step 4: Load Data

```python
@task()
def load_data(final_data_loc):

    from google.cloud import bigquery

    bq_client = bigquery.Client()

    job_config = bigquery.LoadJobConfig(
        write_disposition = bigquery.WriteDisposition.WRITE_APPEND,
        source_format = bigquery.SourceFormat.PARQUET
    )

    gcs_uri = f"gs://{final_data_loc['gcs_bucket']}/{final_data_loc['gcs_dest']}"
    bq_table_id = 'bitcoin_price_index.bpi_xr_hourly'

    load_job = bq_client.load_table_from_uri(
        gcs_uri, bq_table_id, job_config = job_config
    )

    load_job.result()
```

1. **Import *library***
2. **Mengatur konfigurasi *job* Airflow** untuk mengunggah data GCS ke BigQuery
3. **Mengunggah data parquet** BPI kita **ke tabel BigQuery**

All Together Now!

```
        load_job.result()

    bpi_data_info = extract_bpi()
    xr_data_loc = extract_xr(bpi_data_info['run_timestamp'])

    final_data_loc = transform_data(bpi_data_info, xr_data_loc)

    load_data(final_data_loc)


bpi_etl_bigquery()
```

Connecting with Taskflow API

- **API baru di Airflow 2.0**

Terutama sangat berguna jika kita banyak menggunakan `PythonOperator` dan XCom (seperti dalam kasus ini!).

- **Informasi disalurkan ke langkah-langkah selanjutnya**

Setelah ekstraksi data BPI awal, informasi disalurkan ke *step* ekstraksi data XR dan transformasi data, dan seterusnya seperti pada diagram *flow* awal secara otomatis.

- ***Pipeline* dipanggil di baris terakhir**

Pipeline yang telah didefinisikan lengkap dipanggil untuk dijalankan di baris terakhir.

Konfigurasi

GCP Configuration

Google Cloud Storage (GCS)

- **Pastikan atur *region* GCS == BigQuery**

Wajib agar *step load* berjalan mulus. Jangan lupa yang gratis! Ada 3 opsi, namun proyek ini memilih di *us-west-1* (low CO_2).

- **Gunakan *standard storage***

Selain sebagai satu-satunya tipe yang gratis, objek yang kita simpan juga langsung kita akses di *step* selanjutnya setelah dibuat.

- **Siapkan folder dan *prefix* yang dipakai untuk penyimpanan**

Siapkan folder yang sesuai dalam *bucket* dan skema untuk menghasilkan *prefix* nama file yang tidak urut (ada alasannya).

Google BigQuery

- **Pastikan atur *region* BigQuery == GCS**

Wajib agar *step load* berjalan mulus. Proyek ini memilih di *us-west-1* (low CO_2). Ingat, *region* != *multi-region*

- **Atur skema sesuai dengan data akhir yang kita inginkan**

Skema tabel pastinya wajib sesuai dengan data parquet kita nanti, termasuk tipe data dan *nullable*nya.

- **Tambahkan *partitioning* dan *clustering* yang sesuai**

Mempertimbangkan datanya, proyek ini memilih hanya menggunakan *clustering*.

- **Mulai isi-isi metadata untuk Data Catalog**

Setidaknya, isi bagian *description* dari skema kita!

GCP Configuration

IAM

Jika ada metode yang lebih baru, boleh diubah/diganti, tetapi untuk sekarang, *service account*.

- **Buat *service account* untuk Airflow kita**

Ambil *credential* yang diperlukan untuk *pipeline* Airflow nanti. Akan kita langsung gunakan di *slide* selanjutnya!

- **Berikan akses ke *bucket* GCS untuk *service account***

Untuk keperluan kita, cukup berikan *role* “Storage Object Creator” dan “Storage Object Viewer”.

- **Berikan akses ke *project* dan dataset untuk keperluan BigQuery**

Agar BigQuery kita siap dipakai, *service account* kita perlu setidaknya role “BigQuery Job User” pada *project* dan “BigQuery Data Editor” pada dataset.

Airflow/Docker Configuration

Airflow/Docker (1)

Berikut ini beberapa *pointer* berguna, tetapi selebihnya bisa disesuaikan dengan keadaan Docker/Airflow/deployment masing-masing.

- **Siapkan semua *package* Python yang dipakai**

Berikut ini semua *package* yang diperlukan: `apache-airflow-providers-google`, `requests`, `pandas`, `fastparquet`, `pydantic` (jika pakai `pydantic`) dan `typing-extensions` (jika pakai Python 3.7).

- **Atur Google *Application Default Credentials* (ADC)**

Agar *pipeline* kita dapat mengakses *resource* GCP yang akan dipakai. Jika menggunakan *service account*, maka pasang *environment variable* `GOOGLE_APPLICATION_CREDENTIALS` yang mengarah ke JSON *service account key* kita. Jika diperlukan, bisa juga mengatur *environment variable* `AIRFLOW_CONN_GOOGLE_CLOUD_DEFAULT`.

Airflow/Docker Configuration

Airflow/Docker (2)

```python
@dag(
    schedule = '@hourly',
    start_date = pendulum.datetime(2022, 12, 18, tz = 'Asia/Jakarta'),
    catchup = False
)
def bpi_etl_bigquery():
    """
```

Beberapa *point* yang lebih spesifik untuk *case* ini.

- **Atur setting pipeline DAG Airflow seperti disajikan di samping!**

Jadwal dapat ditentukan sebagai *hourly*, `start_date` adalah H-1 (!) kita ingin memulai *pipeline* secara regular, dan `catchup` adalah `False` karena data kita berubah setiap menit ;)

- **Buat Variable/Connection Airflow untuk *key* API OER**

Agar kita tidak perlu menuliskan *key* API kita pada *source code* langsung dan lebih aman!
Ada dua opsi untukmu:

1) Atur `airflow.cfg` agar `app_id` API OER juga bisa mendapatkan *masking* pada JSON *keys* ekstra dari Connection, atau (yang bisa jadi lebih simpel)
2) Simpan *key* sebagai Variable dengan nama yang pasti dapat *masking* (contoh di sini: `oer_api_key`)

Bonus
Data Studio/Looker Dashboard

THAT'S ALL,

THANK YOU!

Narasio DATA