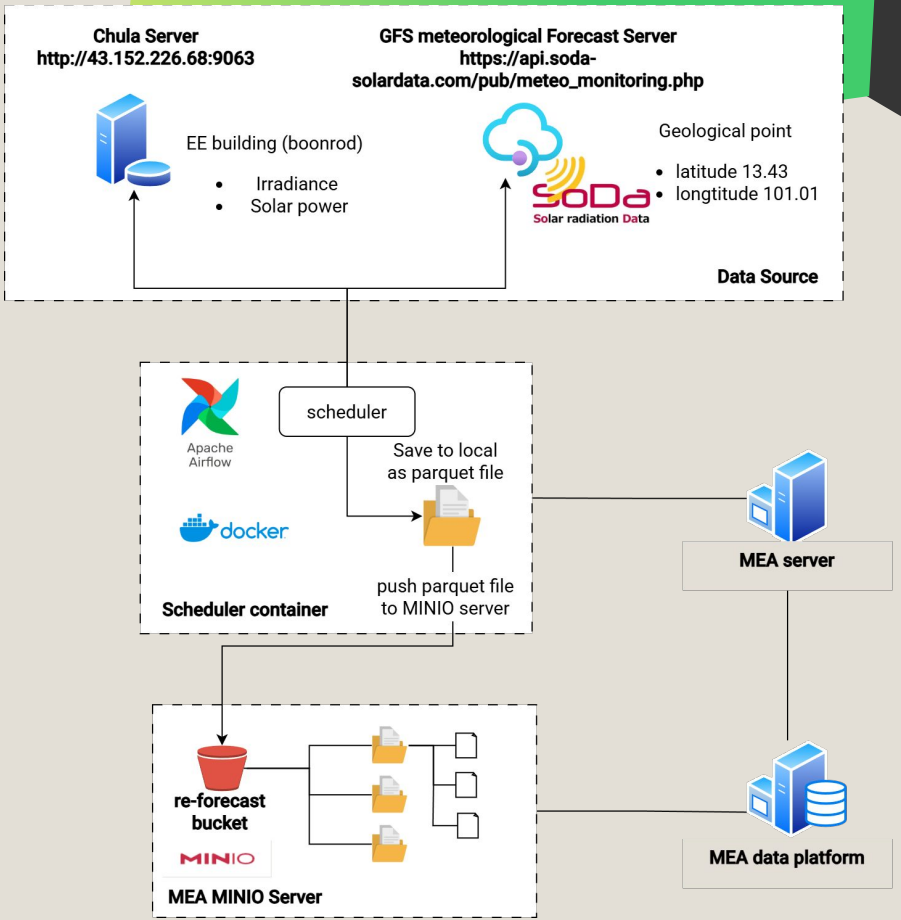
Re-Forecast
2025 / 06 / 23

RE-Forecast Data Pipeline

Data Pipeline Architecture



Flexible Data Access for the Web Application

🗂️ Web App Can Access Data From:

- ✅ API – RESTful API adapter (MEA meter API)
- ✅ CSV Files – On shared storage or uploaded manually
- ✅ MinIO – Object storage access via S3-compatible interface (MEA data platform)

🛠️ Extensible – Adapter-Based Design can add support for:

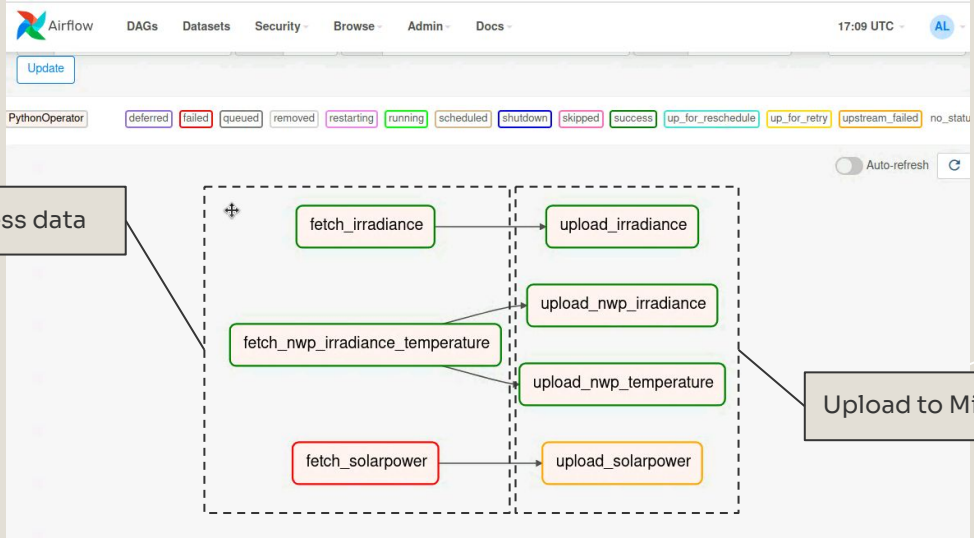
If new data types are needed, simply write a new adapter following the interface

- 🧮 SQL (PostgreSQL, MySQL, etc.)
- 🧪 Any other format via custom adapter

MinIO in Our Pipeline

- Compatible with MEA's internal data platform
- Secure and Private – Can be accessed by MEA's private server only

Fetch data from API and preprocess data



Upload to MinIO Bucket (In MEA Data Platform)

Sample MinIO pipeline in Airflow

Airflow DAG: Custom Python Operators for ETL

```python
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
}

with DAG(
    'data_pipeline',
    default_args=default_args,
    description='Fetches data and uploads to MinIO every minute',
    schedule_interval='*/5 * * * *',  # Every minute
    catchup=False,
    max_active_runs=1,  # Prevents overlapping runs
) as dag:

    fetch_nwp_task = PythonOperator(
        task_id='fetch_nwp_irradiance_temperature',
        python_callable=nwp_irradiance_temperature_fetch,
        op_args=[npw_irradiance_directory, npw_temperature_directory],
    )

    fetch_irradiance_task = PythonOperator(
```

Airflow DAG(Directed Acyclic Graph)

Python Operator Task

```python
    upload_solarpower_task = PythonOperator(
        task_id='upload_solarpower',
        python_callable=upload_to_minio,
        op_args=[bucket_name, "solarpower"],
    )

    upload_nwp_irradiance_task = PythonOperator(
        task_id='upload_nwp_irradiance',
        python_callable=upload_to_minio,
        op_args=[bucket_name, "nwp_irradiance"],
    )

    upload_nwp_temp_task = PythonOperator(
        task_id='upload_nwp_temperature',
        python_callable=upload_to_minio,
        op_args=[bucket_name, "nwp_temperature"],
    )

    # Set task dependencies
    fetch_nwp_task >> [upload_nwp_irradiance_task, upload_nwp_temp_task]
    fetch_irradiance_task >> upload_irradiance_task
    fetch_solarpower_task >> upload_solarpower_task
```

Set Task Dependencies

Fetch Irradiance Task

```python
def fetch_irradiance(start, end):
    """
    Fetch irradiance data from the API for the given time range.
    """
    datetime_index = pd.date_range(start, end, freq='1min')

    # Example data (replace this with actual fetch logic)
    site_names = ['EE Station 1']
    # point_I = df_sites[df_sites['site_name'] == site_names[0]]['points'].iloc[0][2]  # Irradiance
    point_I = 30
    print("point : {}".format(point_I))
    # Prepare the request body
    body_I = json.dumps({
        "point": point_I,
        "start": start,
        "end": end,
        "process": {"1": "raw"},
        "post_process": {"1": "empty;"}
    })

    # Download the data
    link_I = requests.post('https://p6.cusolarforecast.com/export/submit', data=body_I)
    status = json.loads(link_I.text)['status']

    try:
        url_I = url + json.loads(link_I.text)['link']

        # Read data from the URL
        data = requests.get(url_I)
        df_I = pd.read_csv(StringIO(data.text), header=0, names=['Datetime', 'Irradiance_3 (W/m2)'], parse_dates=['Datetime'])
        print(df_I)
        return df_I, True

    except Exception as e:
        error_msg = f"Failed to fetch or process data: {str(e)}"
        print(error_msg)
        raise Exception(error_msg)  # Re-raise as Airflow exception
```

Chula EE building API URL

Download data from URL and read to Dataframe

Update or Create new local file

```python
def update_parquet(directory, start_date=None, end_date=None):
    """
    Update a Parquet file with data for the current month.
    """
    if start_date is None:
        start_date = datetime.now()
    if end_date is None:
        end_date = datetime.now()

    ensure_directory_exists(directory)

    # Get the current year and month
    year = start_date.year
    month = start_date.month

    # Define the file path
    file_name = f"irradiance_{year}-{month:02d}.parquet"
    file_path = os.path.join(directory, file_name)

    # Open or create the Parquet file
    existing_data, file_path = open_or_create_parquet(directory, start_date)

    # Get the start date for fetching new data
    if existing_data.empty:
        start_date = datetime(year, month, 1, 0, 0, 0)  # Start of the month

    else:
        # Get the last datetime from the existing data and add 1 minute to it
        start_date = pd.to_datetime(existing_data['Datetime'].max()) + timedelta(minutes=1)
        start_date = start_date.to_pydatetime()
        start_date = start_date.replace(tzinfo=None)

    # Convert to ISO 8601 format with timezone (+07:00)
    tz = pytz.timezone('Asia/Bangkok')

    now = datetime.now()

    if end_date >= now:
        end_date = now

    start_date = tz.localize(start_date).strftime("%Y-%m-%d %H:%M:%S")
    end_date = tz.localize(end_date).strftime("%Y-%m-%d %H:%M:%S")

    if start_date > end_date:
        print("empty data fetching")
        return 0
    # Fetch new data

    irradiance, res = fetch_irradiance(start_date, end_date)
    if res:
        irradiance['Datetime'] = irradiance['Datetime'].dt.strftime("%Y-%m-%d %H:%M:%S")
        updated_data = pd.concat([existing_data, irradiance]).drop_duplicates(subset=['Datetime'])
        save_to_parquet(updated_data, file_path)
    else:
        return
```

Open or create new file

Get start date from existing file

Concat existing data with new data and save as file

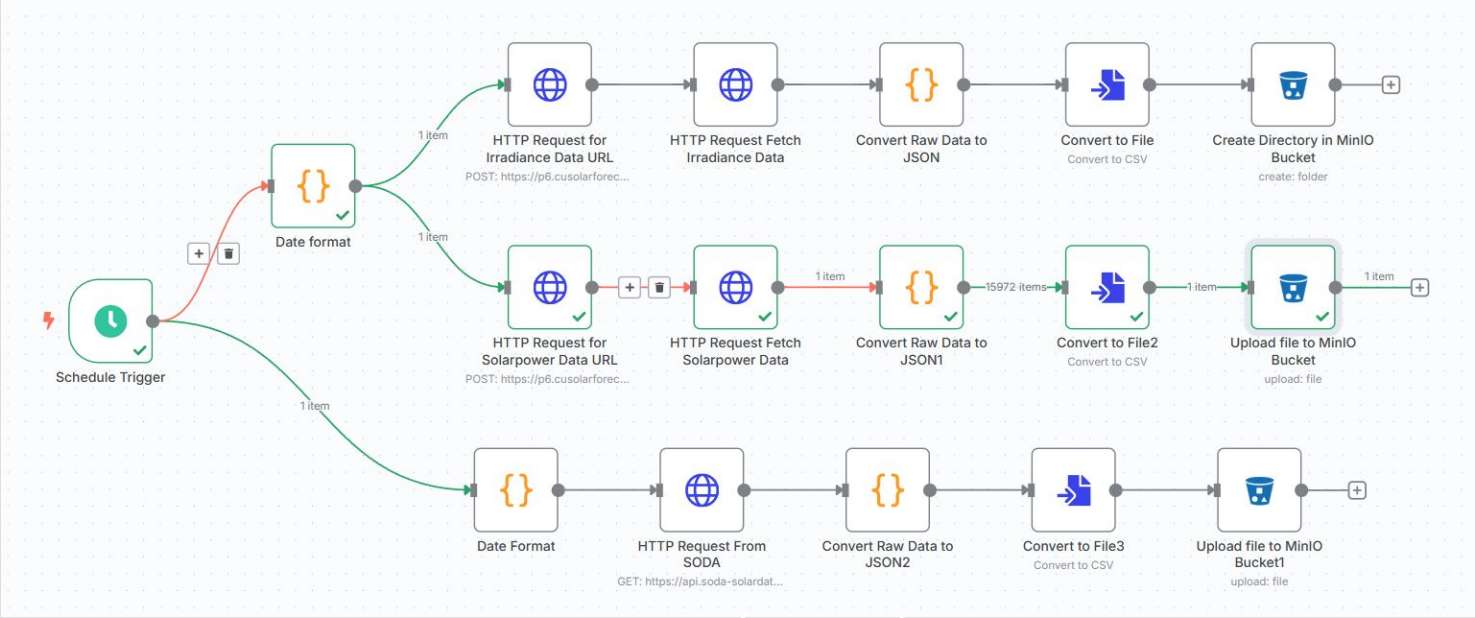
Upload file to MinIO

```python
import datetime
from minio import Minio
from minio.error import S3Error

# Assuming minio_client is already initialized
client = Minio(
    endpoint='172.17.113.251:9000',
    access_key='cu_reforecast2',
    secret_key='Ref0rec@st',
    secure=False
)

    # Replace with your bucket name

def upload_to_minio(bucket_name, directory, date=datetime.datetime.now()):
    """Uploads a file to MinIO with dynamic file name."""
    try:
        # Get current date and construct file name
        file_name = f"eebuilding1_raw_{directory}/{directory}_{date.strftime('%Y-%m')}.parquet"
        print(file_name)
        # Upload the file to MinIO
        client.fput_object(bucket_name, file_name, file_name)
        print(f"Uploaded {file_name} to {bucket_name} bucket as {file_name}")
    except S3Error as e:
        print(f"MinIO upload error: {e}")
```

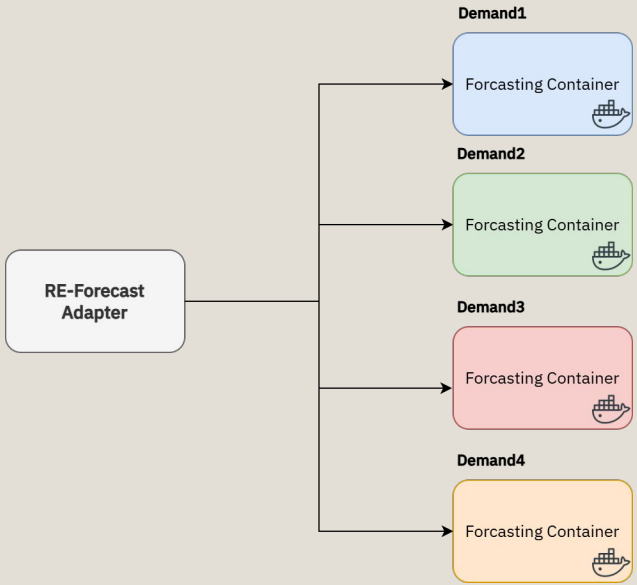
Initialize MinIO Client

Put Object to MinIO Bucket

Alternative Approach: Using n8n for No-Code Workflows



Forecasting



🏠 Demand Forecasting Container

🐳 Packaged as a Docker container running FastAPI

🌐 Provides a RESTful API to request electricity demand forecasts

📌 Site registration: When adding a new site (e.g., house), you specify:

⚙️ Model selection:

- demand1: trained for House1
- demand2: trained for House2

Demand Forecast API

```python
import pandas as pd
import requests

# CSV file path
csv_file_path = 'example_meter_data.csv'

# Read the CSV file into a Pandas DataFrame
df = pd.read_csv(csv_file_path)

# Convert the DataFrame to a JSON object that matches the format expected by FastAPI
data_json = df.to_dict(orient="index")

# Define the FastAPI endpoint URL
url = 'http://localhost:8000/demand/infer'

# Prepare the data payload
payload = {
    "infer_time": "2021-01-06 00:00:00",
    "target_site": "cu_bems",
    "target_col": "netload(kW)",
    "input_cols": ["netload(kW)"],
    "model": "neuralprophet",
    "data": data_json
}

# Send the POST request
response = requests.post(url, json=payload)

# Print the response from the server
print(response.json())
```

Inference endpoint

Parameter and data payload

Testing Inference

1. Navigate to the `client-test` directory:

```
cd client-test
```

2. Run the `client-infer.py` script:

```
python client-infer.py
```

This script will:

- Send a CSV file to the inference endpoint at `http://localhost:8000/demand/infer`.
- Display the server's response, which includes inference results.

API Parameters

Below are the key parameters used in the `/demand/infer` endpoints:

Parameters

Parameter	Description	Example Value
infer_time	Time for inference in the format	`YYYY-MM-DD HH:MM:SS`
target_site	Target column for the prediction.	`cubems`
target_col	Target column for the prediction	`netload(kW)`
input_cols	Input columns for the prediction	`pv(kW),Temperature(C),Short-wave_radiation(Wh/m2)`