FarmNav: An Open Source Database for Agriculture

Kui-Dong Riman[a]

[a]kimriman03@gmail.com

July 2, 2022

Contents

Acknowledgement

The author wants to share this triumph and express his appreciation and sincerest gratitude towards Thinking Machines, and the Eskwelabs for this great learning opportunity, as well as the instructors, mentors, and co-fellows, who patiently shared their knowledge and provided support throughout this learning journey, and the organizers who made the whole event possible.

1 About the Project

Climate change has a significant impact on agriculture. Farmers face difficulties as a result of changes in the frequency and severity of floods and droughts. Having limited information on agricultural areas, made it difficult to understand the causes of crop loss. Advancements in big data analytics, however, created an opportunity for the agriculture sector, thereby combating the issue of unpredictability. In order to mitigate the negative effects of climate change, we need useful information that we can utilize for analytics and strategic decision making. This project aims to provide an open-source database for everyone to use in designing effective strategies towards improving the current state of Agriculture in the Philippines.

2 Data Architecture

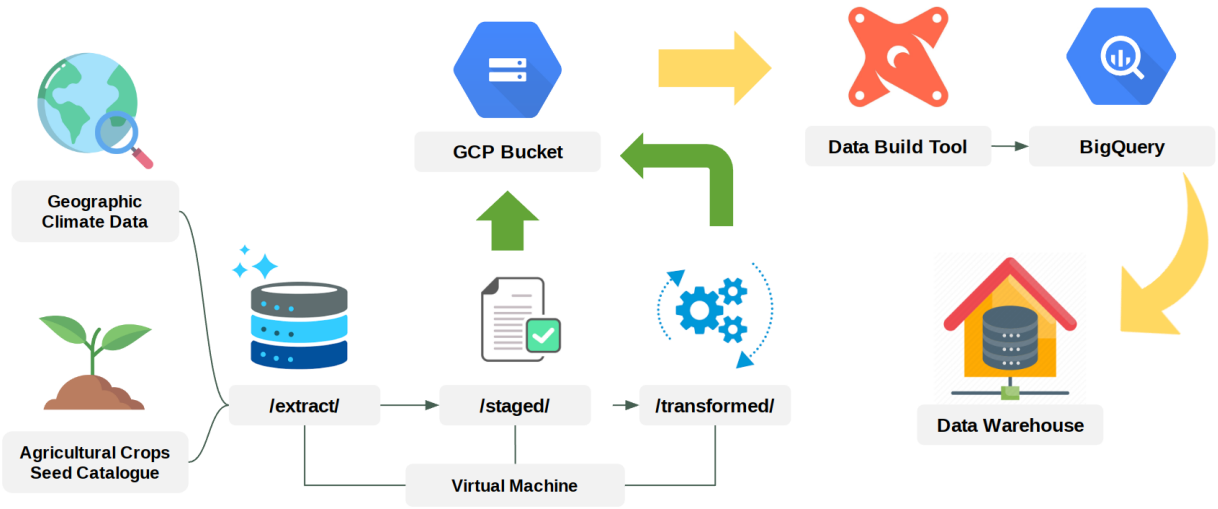

Figure 1. Data Architecture of FarmNav.

This example highlights the concept of using an *EL-T* approach. Ingested raw data are placed in the */extract/* directory. A staging step (*/staged/*) was placed in between extract and transform steps for validating ingested data prior to loading. Transformed data (*/transformed/*) was then validated and uploaded to the cloud storage bucket. Data integration was automated via Data Build Tool (DBT) for creating tables and views in BigQuery. These tables and views would constitute the data warehouse.

***Note**: This version cannot run DBT automation at the moment, this feature should be available in future releases, thank you!* :)

3 Getting Started

1. Install Docker Engine from: https://docs.docker.com/engine/

2. Obtain a CDS API key:

   (a) Register for a Copernicus account here: https://cds.climate.copernicus.eu/

   (b) Get your CDS API key here: https://cds.climate.copernicus.eu/api-how-to

3. Create a Google Cloud VM instance or Install the Google Cloud CLI.

   You need to login[3] and obtain access credentials[2] for your gcloud account (See Section 7.1)

4. Setup your secrets via Google Cloud Secret Manager[4]

   Create secrets for the following:

   (a) Cloud storage access:

   ```
   <access_key>:<token>
   ```

   (b) Copernicus API key:

   ```
   "url":<url>
   "token":<token>
   ```

   (c) Discord webhook URL:

   ```
   <discord_webhook_url>
   ```

5. Input these values to your configuration file in: **./dags/scripts/config.py**

6. Clone the github repository from: https://github.com/Kui-03/farm-nav

4 Installation

4.1 YAML file

The **docker-compose.yaml** was modified to include data and google cloud config directories:

```
image: ${AIRFLOW_IMAGE_NAME:-kui03/airflow-conda:latest}
volumes:
- ./data:/opt/airflow/data
- ${HOME}/.config/gcloud:/home/airflow/.config/gcloud
```

4.2 Running Airflow

Login to your VM, create and move to new directory:

```
mkdir airflow_project
cd airflow_project
```

Pull the docker image:

```
docker pull kui03/airflow-conda:latest
```

Execute the following commands:

```
mkdir -p ./dags ./logs ./plugins
echo -e "AIRFLOW_UID=$(id -u)" > .env
docker compose up
```

5 Tasks

5.1 check_request_queue

This checks local records whether an existing request has been submitted to the API.

Xcom_push

- queue: list, csv paths found
- status: str, the next function to return

Returns

- str, branch task name

5.2 create_request

This creates a new request and submits it to the API. Notifies discord that a request has been made.

Returns

- dict, containing request_id, variable, date

5.3 pull_request_data

This pulls a data from the API to check whether the request exists, then uses the information to monitor the status of a request.

Xcom_pull

check_request_queue()

- queue: list, list of csv paths found in request directory.

Returns

- dict, containing request_id, variable, date

5.4 wait_request

This task waits for the request's 'complete' state before proceeding to download.

Xcom_pull

check_request_queue

- status: str, returned task name

create_request / pull_request_data

- request: dict, containing request_id, variable, returned from either branch

Xcom_push

- request_id: str, unqiue id provided by the API
- variable: str, requested climate variable
- date: str, date request created

Returns

- return: bool, if request has completed

5.5 download_request

Downloads a completed request. Sends a message that a file has been downloaded.

Xcom_pull

wait_request

- request_id: str, unqiue id provided by the API
- variable: str, requested climate variable

Returns

- str, path to downloaded file

5.6 verify_download

Checks whether a download is corrupted. Sends a message on error.

Xcom_pull

download_request

- str, download path

Returns

- str, path to downloaded file

5.7 stage_download

Moves a downloaded file to staged directory, for transformation and uploading.

Xcom_pull

verify_download

- str, filepath

Returns

- str, path to staged file

5.8 upload_staged

Uploads the raw data (staged) to the cloud. Sends a message on completion.

Xcom_pull

stage_download

- str, path to staged file

5.9 transform_grib

Converts the downloaded *grib* file to a dataframe, applies the necessary transformations, and exports the data in a compressed *parquet* format.

Xcom_pull

stage_download

- str, path to staged file

Returns

- str, path to transformed file

5.10 check_transform

Perform validation on transformed data. Sends a message on error.

Xcom_pull

check_request_queue

- status: str, returned task name

create_request / pull_request_data

- request: dict, containing request_id, variable, returned from either branch

transform_grib

- str, path to transformed file

Returns

- str, path to transformed file

5.11 upload_transformed

Uploads transformed (validated) data to google cloud storage bucket. Performs a local directory clean-up. Sends a message on completion.

Xcom_pull

check_transform

- str, google storage path to file

Returns

- str, google cloud storage bucket - path to transformed file

6 Scripts

6.1 transform.py

Contains functions for transforming GRIB file:

- **transform_clean(df: pd.DataFrame)** - takes a pandas dataframe, drops null values, and renames columns. Returns a pandas dataframe.
- **create_ids(df: pd.DataFrame)** - takes a pandas dataframe, creates unique ids per row. Returns a pandas dataframe.
- **transform_pipeline(df: pd.DataFrame)** - contains the functions above for transformation, additional functions may also be placed here. Returns a pandas dataframe.
- **main_transform(filepath: str)** - reads the GRIB file, then applies the transformations from *transform_pipeline()*, and saves the data to PARQUET format.

6.2 validate.py

Contains functions for validating the transformed file, *raises an Exception* if the set conditions are not met:

- **check_nan_unique(df: pd.DataFrame, cols: list)** - takes a pandas dataframe, checks if there are null values in the columns (*cols*) provided. Also checks if values under *id* columns are unique. Returns True.
- **validate_pipeline(df: pd.DataFrame, variable: str)** - contains the function(s) above for validation, additional functions may also be placed here. Takes a pandas dataframe, and the requested climate variable. Returns True.

- **main_validate(filepath: str, request_dict: dict)** - Takes the transformed file path, *filepath*, and request information, *request_dict*. Reads the transformed PARQUET file, and applies the validations from the *validate_pipeline()* function.

7 Security

7.1 Setting up Google Cloud CLI

In order to access Google Cloud Secret Manager in the backend, the gcloud CLI should be initialized first[5].

This step is done by executing the following commands:

```
gcloud auth login --no-browser
gcloud auth application-default login
```

7.2 Google Cloud Secret Manager

Secrets can be accessed by providing the secret name and appropriate versions in the configuration file (./dags/scripts/config.py):

```
SECRET_COPERNICUS_API_NAME = [SECRET-NAME-FOR-COPERNICUS-API-KEYS]
SECRET_COPERNICUS_API_VERSION = [SECRET-VERSION]

SECRET_CLOUD_ACCESS_NAME = [SECRET-NAME-FOR-CLOUD-STORAGE-SECRETS]
SECRET_CLOUD_ACCESS_VERSION = [SECRET-VERSION]

SECRET_SLACK_WEBHOOK_NAME = [SECRET-NAME-FOR-DISCORD-WEBHOOK-URL]
SECRET_SLACK_WEBHOOK_VERSION = [SECRET-VERSION]
```

8 Monitoring

The URL for the webhook is stored via Secret Manager and can be accessed by providing the secret name in the configuration file (Section 7.2).

8.1 Update notifications

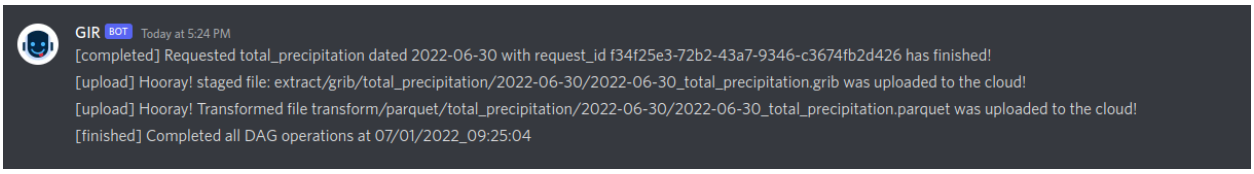

Figure 2. Discord update notifications.

Important task updates can be recieved via notifications to monitor the current status of the DAG (Fig. 2). Additional notifications can be created using the *send_message()* function from *message.py* script (./dags/scripts/message.py).

8.2 Error notifications

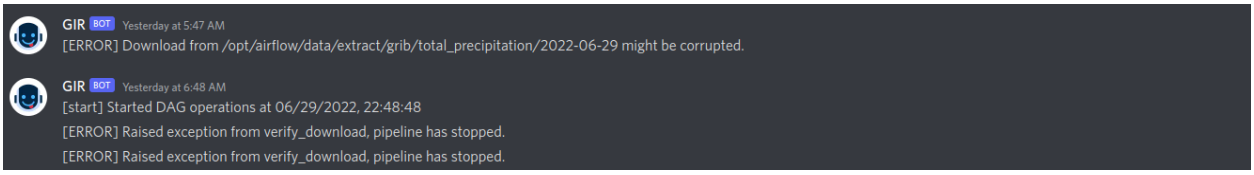

Figure 3. Discord error notifications.

In the event that there has been an error in the pipeline, the task sends an error notification via discord webhook. This includes whether an Exception has been raised, or under a specific condition, the pipeline has decided to stop (Fig. 3).

9 Data Source

This project used the ERA5-Land hourly data from 1950 to present (total_precipitation), from the Copernicus Copernicus Climate Data Store[1], and Ramgo seeds production guide[6] as primary data sources.

10 Future Improvements

The following can be done to improve the current project:

- *Data Build Tool (DBT)* for data integration, and building BigQuery tables.
- *MongoDB*. As ingested data grows bigger, storing request information can be difficult. In this regard, MongoDB might be more suited for this task in the future.
- *Additional data*. Including geo/climate variables (elevation, humidity, runoff, surface pressure, etc.), as well as area population and market prices, to target high value crops.

Updates for this project including a recent version of this documentation can be found here.

11 Web demo

The web demo of this project can be accessed using the txt file contained within this document (expiry: July 8, 2022).

References

[1] J. Muñoz Sabater. “ERA5-Land hourly data from 1981 to present”. In: *Copernicus Climate Change Service (C3S) Climate Data Store (CDS)* (2019). DOI: https://doi.org/10.24381/cds.e2161bac.

[2] Google Cloud Platform. *Application Default Credentials*. 2022. URL: https://cloud.google.com/sdk/gcloud/reference/auth/application-default/login.

[3] Google Cloud Platform. *Google Auth Login*. 2022. URL: https://cloud.google.com/sdk/gcloud/reference/auth/login.

[4] Google Cloud Platform. *Google Secret Manager*. 2022. URL: https://cloud.google.com/secret-manager.

[5] Google Cloud Platform. *Initializing the gcloud CLI*. 2022. URL: https://cloud.google.com/sdk/docs/initializing.

[6] Ramgo Seeds. *Ramgo seeds production guide*. URL: http://ramgoseeds.com/vegetables/.