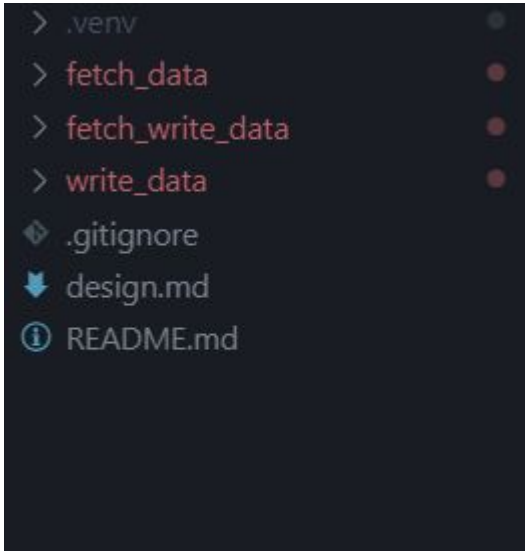
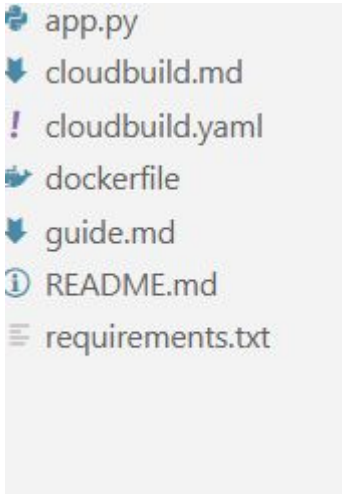
Genomgång

Small things that matter

- In exercise you deployed code for a single project, located in root
- Eventually your project will contain different folders,
- Each containing its own dockerfiles, cloudbuilds etc
- There are some things to consider





CICD pipeline

- You will have triggers (cloud build) for different parts of your pipeline, so each service can deploy on its own (unless you build one gigantic service)

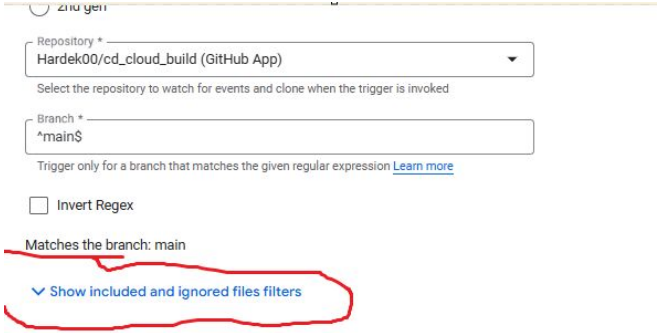
fetcher-writer	europe-north2	demo for fetcher-writer	Hardek00/demo_ingestion_pipeline	Push to branch	fetch_write_data/cloudbuild.yaml	Enabled	Run
ingeestion-pipeline	europe-north2	Full ingestion	Hardek00/demo_ingestion_pipeline	Push to branch	fetch_data/cloudbuild.yaml	Enabled	Run
writer	europe-north2	writer app	Hardek00/demo_ingestion_pipeline	Push to branch	write_data/cloudbuild.yaml	Enabled	Run

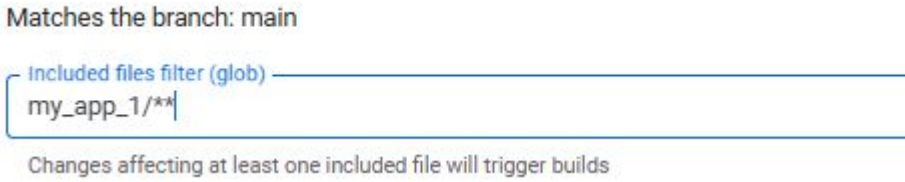
- Here is the catch, if we build all triggers from the same repo, and do a push…what will happen?

The solution:

- its kinda hidden
- Use <your_project_folder_you_want_to_track>
- followed by “/”
- followed by two stars **
- It will only trigger and deploy code if there have

  have been changes in code in that specific folder.



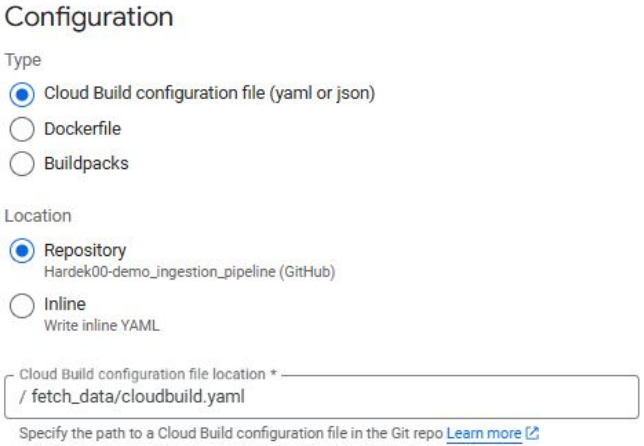


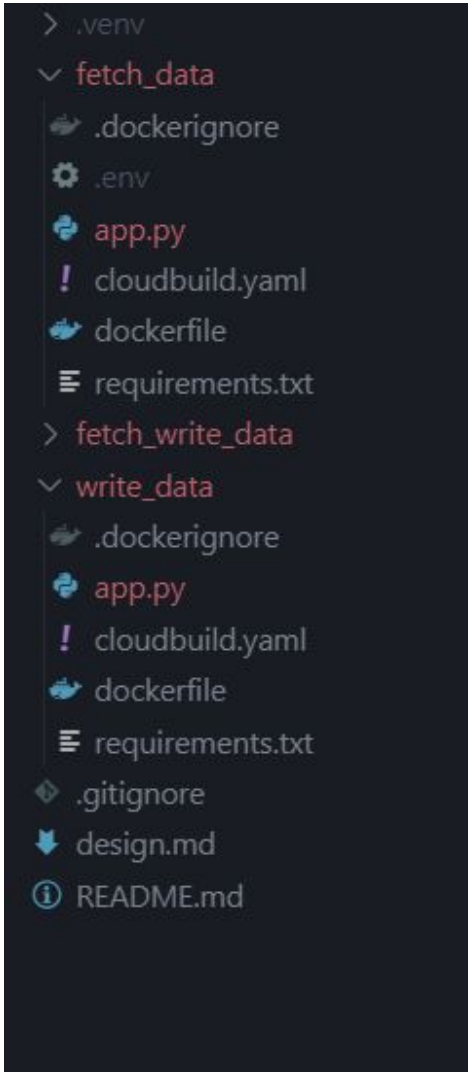
Cloudbuild.yaml pathing

- If you choose cloudbuild for your CICD,

  consider this:

- You will most likely have many different cloudbuild yamls
- You need to set pathing correctly in the triggers





Secrets

- You should be familiar with .env files by this point.
- Here's the catch, if you are doing things right, those files should never be deployed to github.
- Which means information in them never reaches the cloud.
- Does it mean we can't deploy sensitive information into cloud?

All cloud providers offer solution.

- In GCP we have cloud secrets. (Secret manager)
- It let use paste in sensitive information to be used in our environment.

- We still use API_KEY = os.environ["WEATHER_API_KEY"] to call it.

Non-sensitive env variables

- meaning: tables id, app names, regions etc.
- Can be handled in many different ways:
- Hardcoding(easiest, not the best practice)
- Manually deployed in cloud run (not effective)
- Substitution and env var in cloudbuild.yaml(most effective, complex)

Cloudbuild.yaml

- In real projects you usually have **one `cloudbuild.yaml`** and many **environments**.
- With hardcoding, you'd need separate files (`cloudbuild.dev.yaml`, `cloudbuild.prod.yaml`).
- With substitutions, you reuse one file, and the trigger/job defines the values

```yaml
substitutions:
  _BQ_DATASET: weather
  _BQ_TABLE: raw_events

steps:
- name: gcr.io/cloud-builders/gcloud
  args:
    - run
    - deploy
    - writer
    - --image=gcr.io/$PROJECT_ID/writer
    - --region=europe-north1
    - --set-env-vars=BQ_DATASET=${_BQ_DATASET},BQ_TABLE=${_BQ_TABLE}
```

How do we run a pipeline?

- Scheduled batch (most common)

  - **Orchestrator** (Workflows/Composer) runs steps on a schedule (via Cloud Scheduler).

  - Steps: fetch → validate → transform → write → notify (exempel)

  - Data exchanged via files (GCS) or tables (BigQuery).

What is an orchestrator?

An orchestrator is a tool or framework that **manages, schedules, and monitors data workflows** (pipelines). Instead of manually running jobs, the orchestrator ensures that tasks happen **in the right order, at the right time, with the right resources**.

Why Orchestration is Needed

- **Dependencies** → Job B starts only after Job A finishes successfully.
- **Automation** → Replace manual runs with automated, scheduled execution.
- **Reliability** → Built-in retries, alerts, and failure handling.
- **Scalability** → Coordinate hundreds/thousands of jobs across systems.
- **Observability** → Central place to track status, logs, and metrics.

Popular Orchestrators in DE

- **Apache Airflow** → widely used, DAG-based orchestration
- **Prefect** → Pythonic, cloud-friendly
- **Dagster** → focuses on data asset management
- **Cloud-native tools** → AWS Step Functions, GCP Workflows, Azure Data Factory

Cloud scheduler

What it is:

- Fully managed **cron-like service** on GCP.
- Triggers HTTP(S), Pub/Sub, or App Engine tasks.
- Used for **time-based automation** in cloud environments.

Key Benefits:

- No servers to manage.
- Highly reliable & scalable.
- Integrates with other GCP services.
- Supports retries & monitoring.

Use Cases:

- Kick off ETL/ELT pipelines daily.
- Schedule batch jobs or reports

Cron

The **Unix-cron format** is a simple **time expression language**.

It tells the system *when* to run a job (e.g., “every 5 minutes”, “midnight every day”).

Cloud Scheduler (and many orchestrators) reuse this standard format

A cron string has **5 fields**: minute, hour, day, month, day-of-week.

Each field can hold a number, range, list, or wildcard (`*`) to describe time.

`0 9 * * *` → “Run every day at 09:00.”

Just use https://crontab.cronhub.io/ or something