# A GenAI Pipeline for Content Generation with Apache Airflow

May 30th 2024

ASTRONOMER

## Agenda

- **Quick introduction:** Airflow and Astronomer

- **Challenges** of GenAI pipelines and how Airflow addresses them

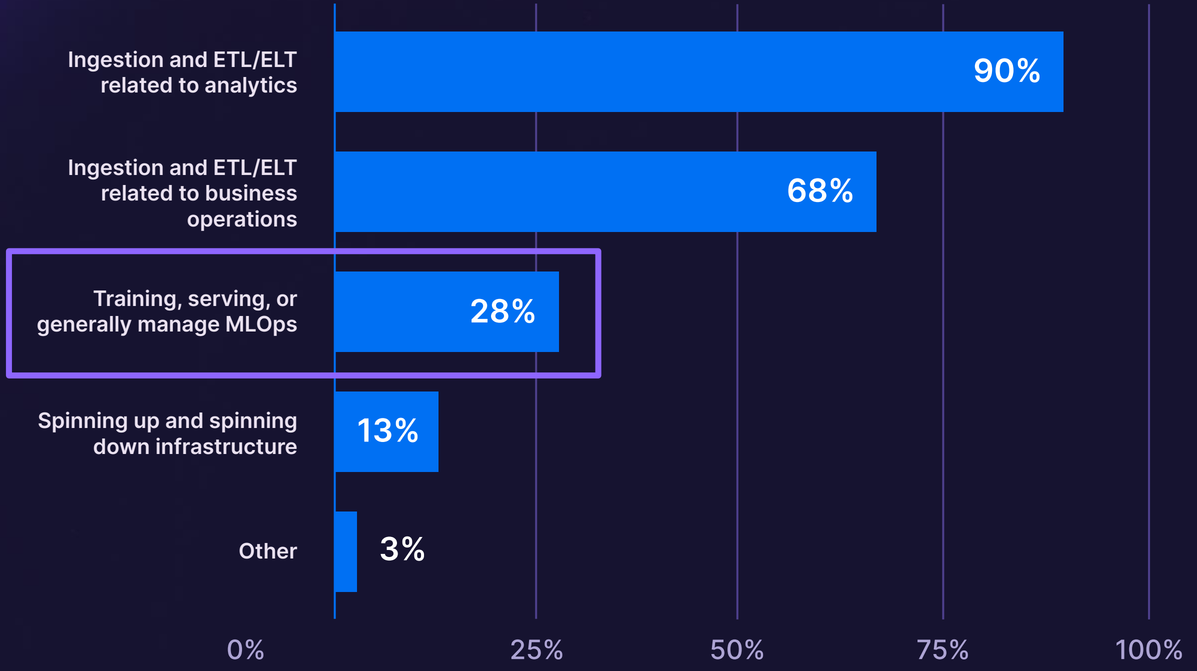
- **Key Airflow features** used in the demo
    - Focus on the latest 2.9 features:
        - Advanced Dataset Scheduling
        - Dynamic Task Mapping

- **Demo:** Fine-tune GPT for an RAG pipeline for content generation
  https://github.com/astronomer/gen-ai-fine-tune-rag-use-case

# **Airflow is** the open standard for **Workflow Management.**

| | | | |
|---|---|---|---|
| **22M+** monthly downloads | **46k+** Slack members | **2800+** contributors | **1600+** building blocks |
| GROWTH | COMMUNITY | INNOVATION | ECOSYSTEM |

## More and more people are using Airflow for ML/AI

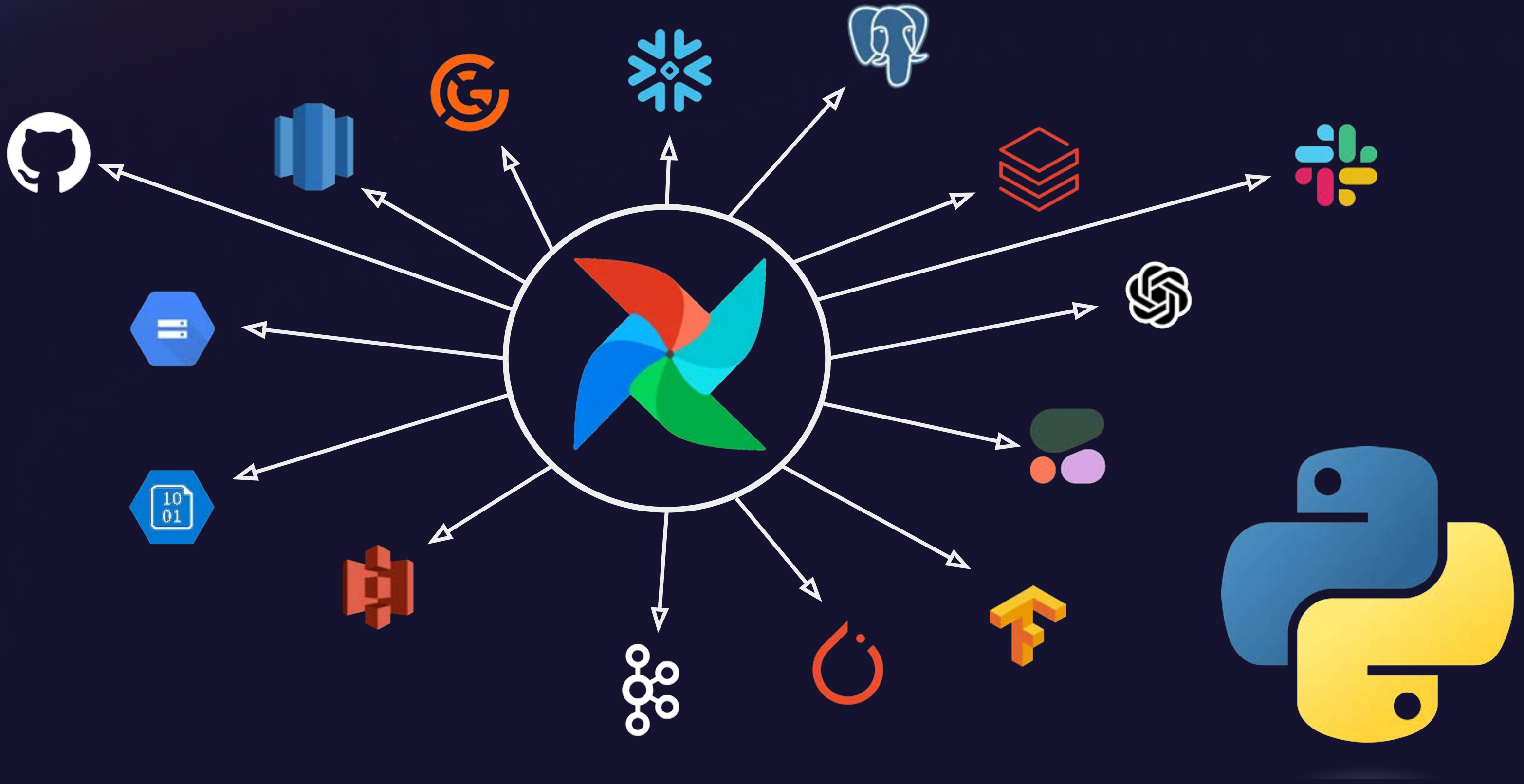


*Source: 2023 Apache Airflow Survey, n=797*

28%

Of Airflow survey respondents in 2023 said they use Airflow for at least one ML/AI related use case.

ASTRONOMER



ASTRONOMER

## The driving force behind Apache Airflow

24×7 worldwide support | Worlds Top Airflow Experts

**100%**

Driving 100% of Airflow releases

**55%**

Of Airflow code contributed

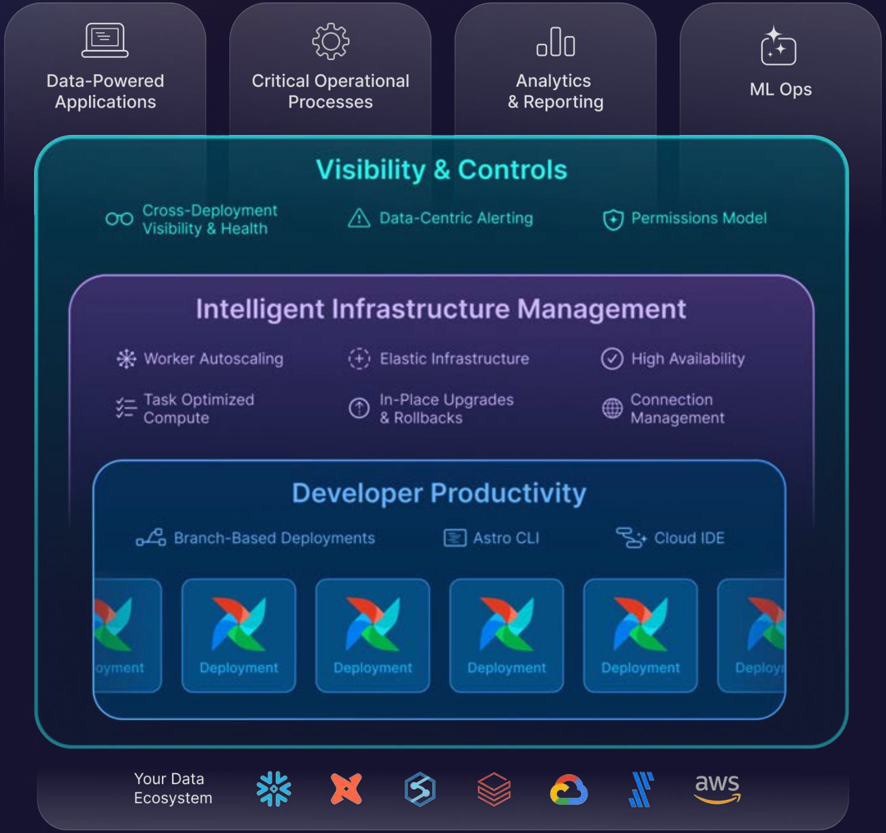
**18 of 25**

18 of the top 25 committers on board, 8 PMC members

**30K+**

30K+ Airflow students in Academy ecosystem

# Astro



## Supercharge Airflow with Astronomer

**Start Your 14-Day Free Trial**
(no credit card required)

New sign-ups receive $300 in credits *and* a complimentary Airflow Fundamentals Certification Exam (normally $150).

https://qrco.de/bf2ICP

ASTRONOMER

## Challenges when creating GenAI pipelines

The prototype works great - but production is a different beast

- API outages and rate limits
- Need to keep training data up to date
  - Your data is what sets you apart from competitors!
- Changing tools and APIs - new models coming out every day
- Complex pipeline structures
- Need the ability to determine which data went into training (compliance!)
- Scalability
- Reliability
- …

## Challenges when creating GenAI pipelines

The prototype works great - but production is a different beast

- API outages and rate limits -> Automatic retries
- Need to keep training data up to date -> Airflow already the standard
- Changing tools and APIs -> Airflow is tool agnostic, TaskFlow API
- Complex pipeline structures -> Datasets, dynamic task mapping, branching
- Need the ability to determine which data went into training (compliance!) -> Observability + OpenLineage integration
- Scalability -> Pluggable compute
- Reliability -> Battle tested + it is all code: CI/CD and DevOps best practices

Your data **+ your orchestration** is what sets you apart from competitors!

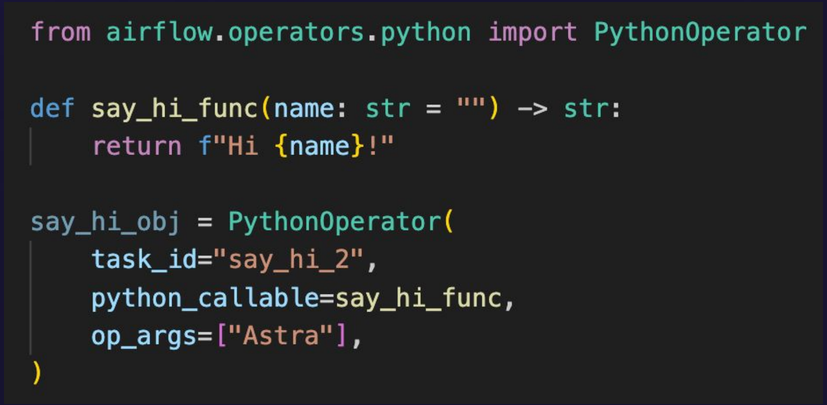
## Key Airflow features for GenAI

These features build a good foundation for best practice GenAI pipelines

- TaskFlow API
- Automatic retries
- Branching
- Deferrable operators
- Data-driven scheduling using Datasets
- Dynamic task mapping
- Alerts and notifications
- Setup and teardown tasks
- Backfills and reruns

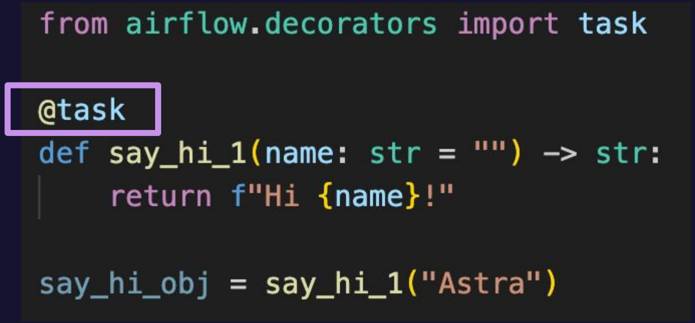
## TaskFlow API ⇒ Airflow decorators

The pythonic way to write Airflow DAGs



```python
from airflow.operators.python import PythonOperator

def say_hi_func(name: str = "") -> str:
    return f"Hi {name}!"

say_hi_obj = PythonOperator(
    task_id="say_hi_2",
    python_callable=say_hi_func,
    op_args=["Astra"],
)
```

⇒



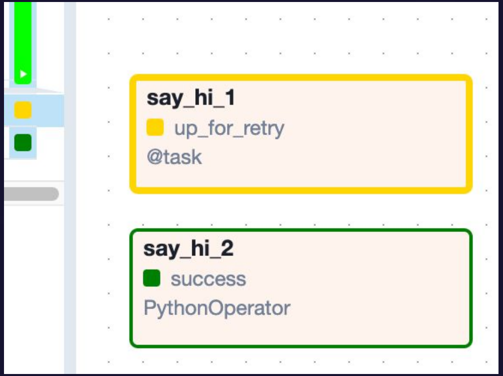
```python
from airflow.decorators import task

@task
def say_hi_1(name: str = "") -> str:
    return f"Hi {name}!"

say_hi_obj = say_hi_1("Astra")
```

You can mix traditional operators and Airflow decorators!

There are many decorators: @dag, @task.kubernetes, @task.branch, @task.bash etc… see: https://astronomer.io/docs/learn/airflow-decorators

## Automatic retries in Airflow

Protects pipelines against rate-limits and API failures



You can configure:

- Number of retries
- Delay between retries
- Exponential backoff
- Maximum delay

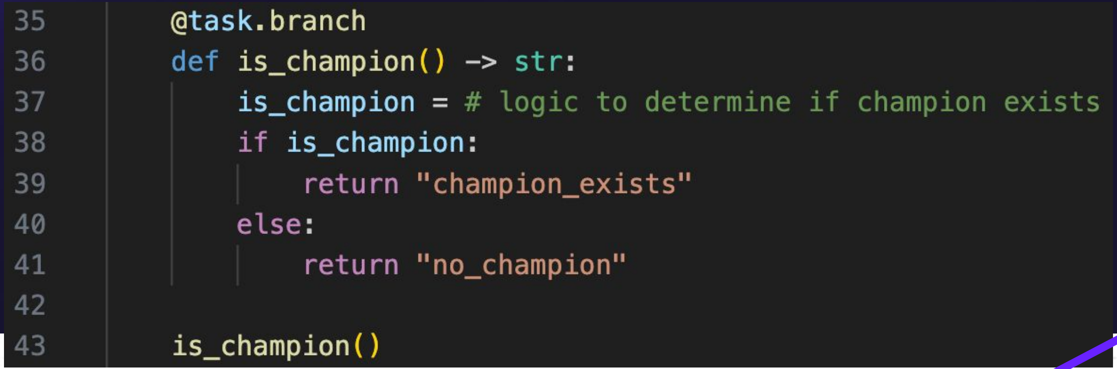
Ways to configure:

- Airflow config
- `default_args` in DAGs
- Individual tasks

**Best practice**: Always set retries in production, unless a task has a reason not to

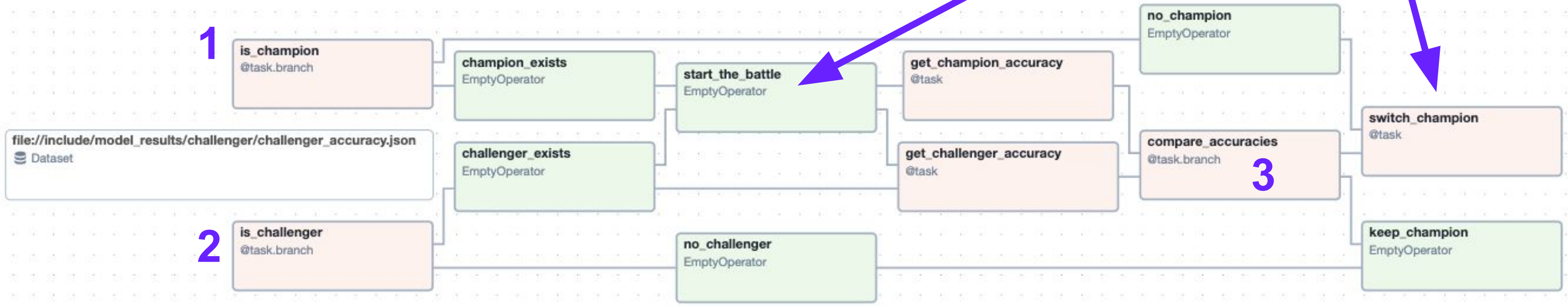
Details: https://astronomer.io/docs/learn/rerunning-dags#automatically-retry-tasks

# Branching in Airflow



```python
    @task.branch
    def is_champion() -> str:
        is_champion = # logic to determine if champion exists
        if is_champion:
            return "champion_exists"
        else:
            return "no_champion"

    is_champion()
```

Careful with downstream trigger rules!



Details: https://astronomer.io/docs/learn/airflow-branch-operator

# Deferrable operators

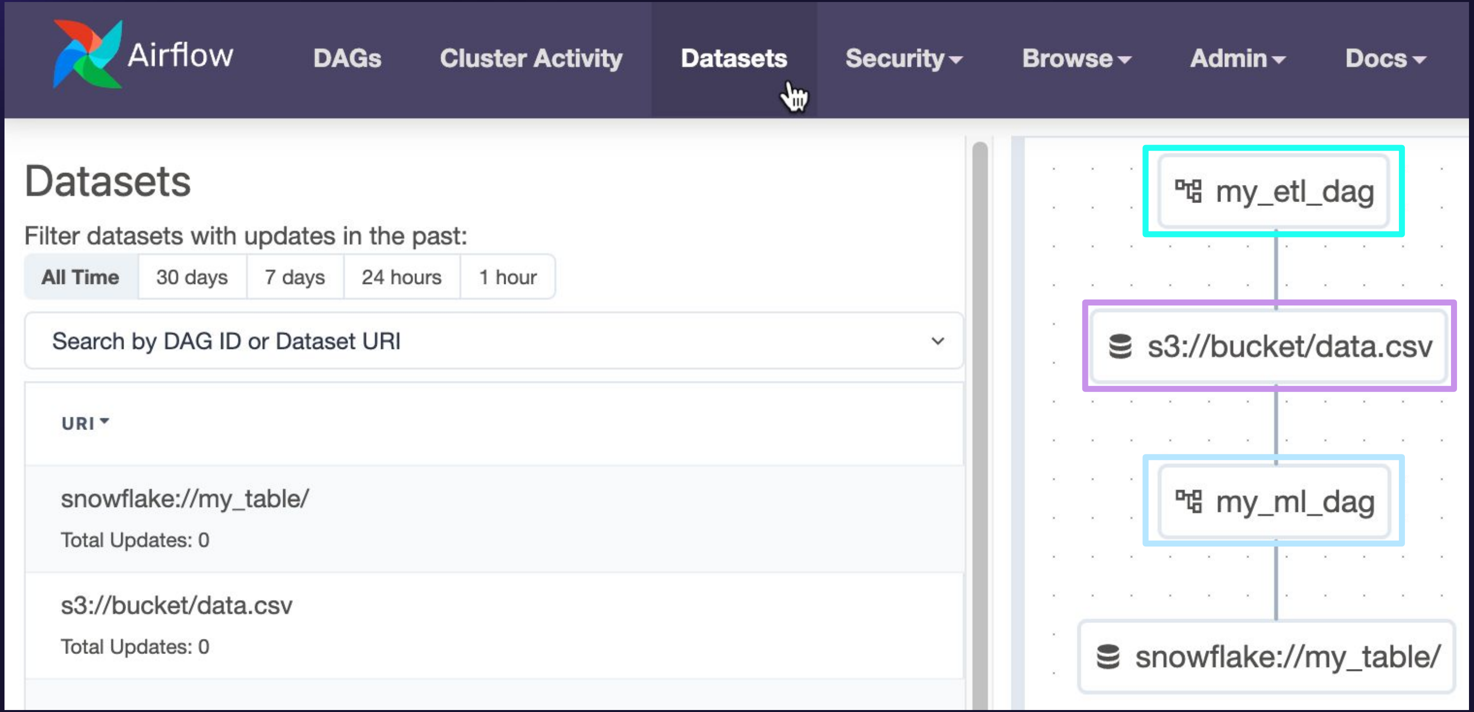
- Deferrable operators can start async processes in the **Triggerer** component.
- Use case:
    - Waiting for a long running process to finish (e.g. model training)
    - Waiting for an event to occur in an external system (like a sensor)
- Advantage:
    - The worker slot is released = resource use optimization

- **Best practice**: Use deferrable operators whenever possible for longer tasks.

00:00:12
00:00:06
00:00:00
async_task

Details: https://astronomer.io/docs/learn/deferrable-operators

# Dataset scheduling



Datasets

Filter datasets with updates in the past:

snowflake://my_table/
Total Updates: 0

s3://bucket/data.csv
Total Updates: 0

DAG with producer task

Dataset

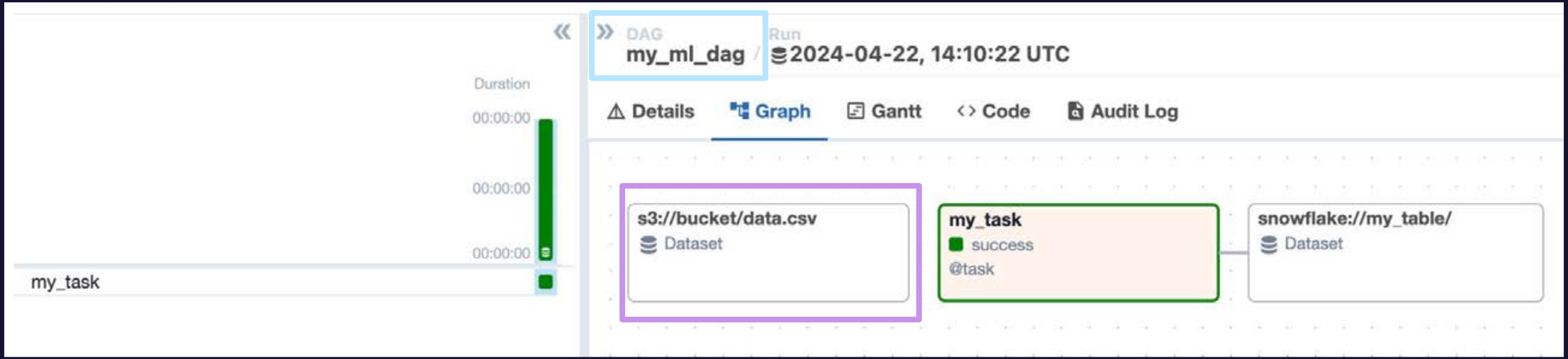
Consumer DAG + producer task

Next Dataset

Airflow 2.9

# Datasets in the Airflow 2.9 UI

Consumer
DAG



Dataset

Producer
task

Next
Dataset

Airflow 2.9

## Advanced Dataset scheduling

Airflow 2.9 additions:

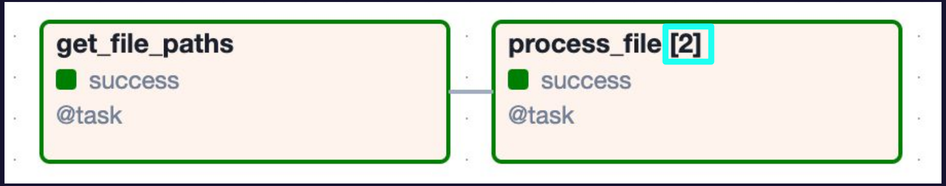
- Schedule on logical dataset expressions
    - Use AND (&) / OR ( | ) to create dataset logic
- Schedule on both time and datasets
    - `DatasetOrTimeSchedule` takes a `timetable` and a `dataset` argument
- REST API endpoint to update Datasets
    - Use for cross-deployment dependencies

Details: https://astronomer.io/docs/learn/airflow-datasets

# Dynamic Task Mapping

- Create a variable number of copies of the same task based on input at runtime!
- Define parameters that stay the same (`.partial()`) and parameters that change in between task instances (`.expand()` / `.expand_kwargs()`)

- **Best practice:**
  - Use dynamic tasks when possible over dynamic DAGs
  - Customize the map index (Airflow 2.9)



ASTRONOMER

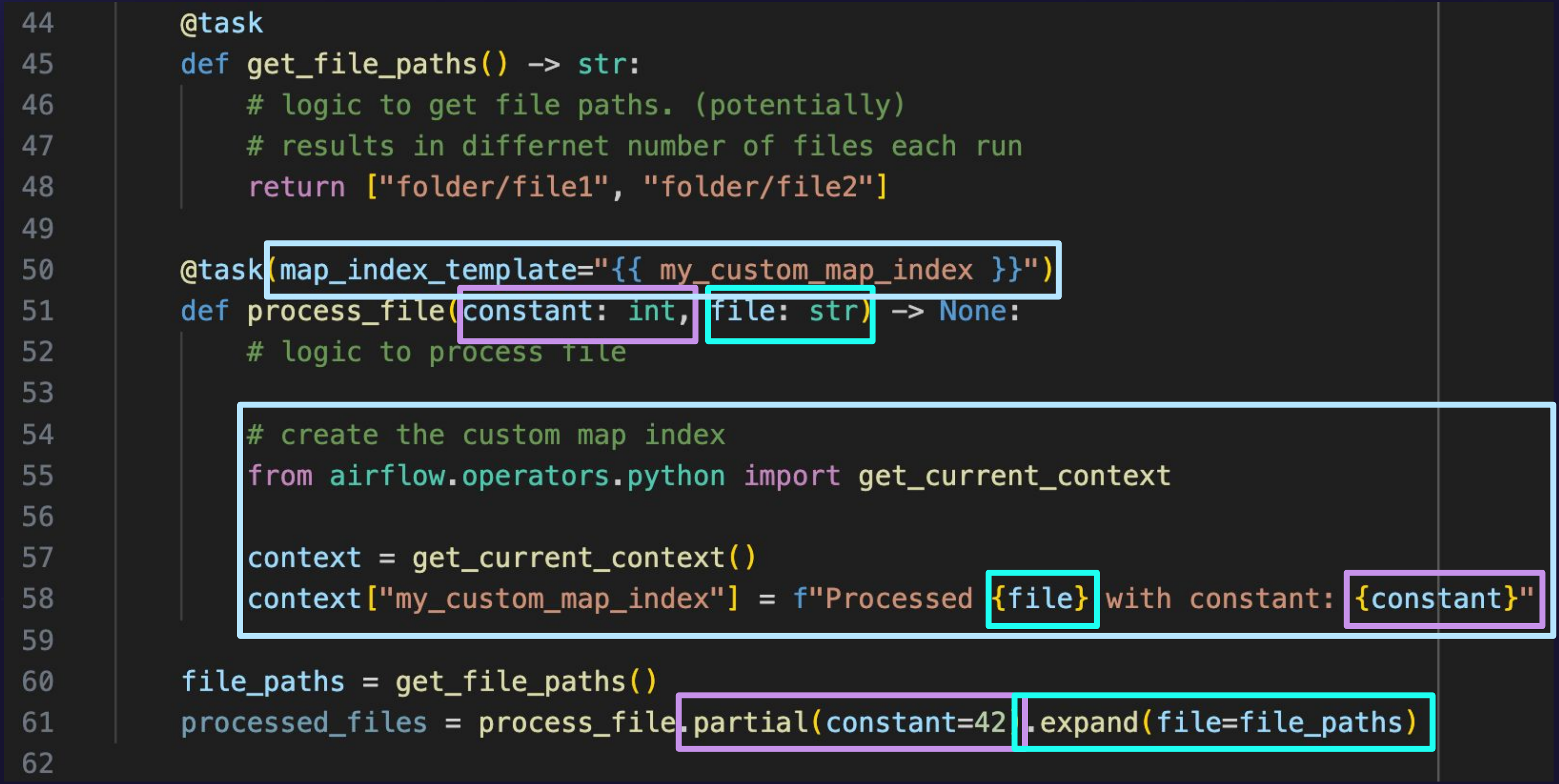
# Dynamic Task Mapping

| | |
|---|---|
| **Basic:** | |
| - `.partial(a=2)` | → all parameters that stay the **same** for each mapped instance |
| - **`.expand(b=[0,1])`** | → the parameter that changes as a list. Naming the kwarg is mandatory! |
| - `map_index_template` | → customize the map index displayed in the **UI** (2.9) |
| **Advanced:** | |
| - .expand_kwargs([{“a”:1}]) | → map over sets of keyword arguments |
| - .map(lambda x: x) | → transform the output of an upstream task before mapping over it |

Details: https://astronomer.io/docs/learn/dynamic-tasks

Airflow 2.9

# Dynamic Task Mapping - Simple example

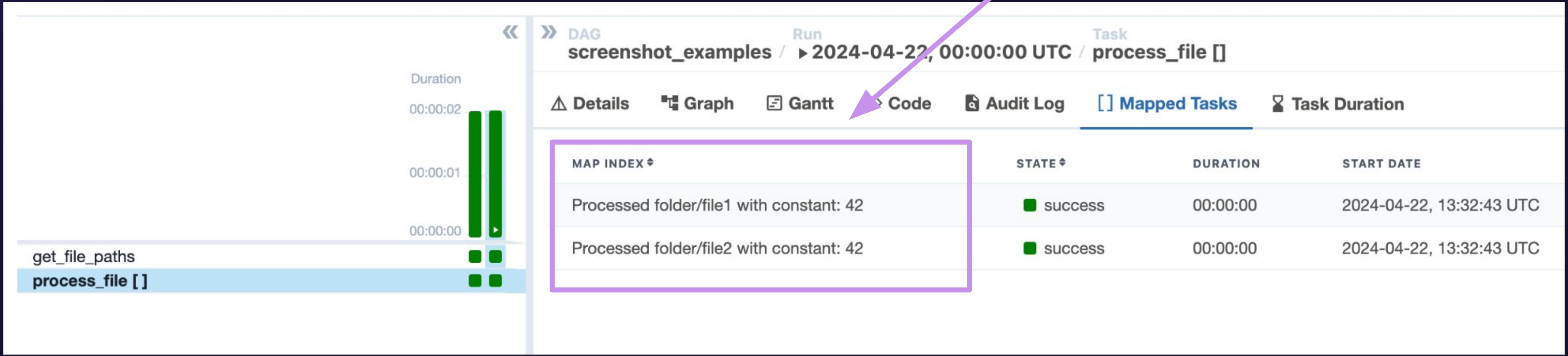


```python
    @task
    def get_file_paths() -> str:
        # logic to get file paths. (potentially)
        # results in differnet number of files each run
        return ["folder/file1", "folder/file2"]

    @task(map_index_template="{{ my_custom_map_index }}")
    def process_file(constant: int, file: str) -> None:
        # logic to process file

        # create the custom map index
        from airflow.operators.python import get_current_context

        context = get_current_context()
        context["my_custom_map_index"] = f"Processed {file} with constant: {constant}"

    file_paths = get_file_paths()
    processed_files = process_file.partial(constant=42).expand(file=file_paths)

```

Airflow 2.9

# Dynamic Task mapping custom index

Custom map index (2.9)

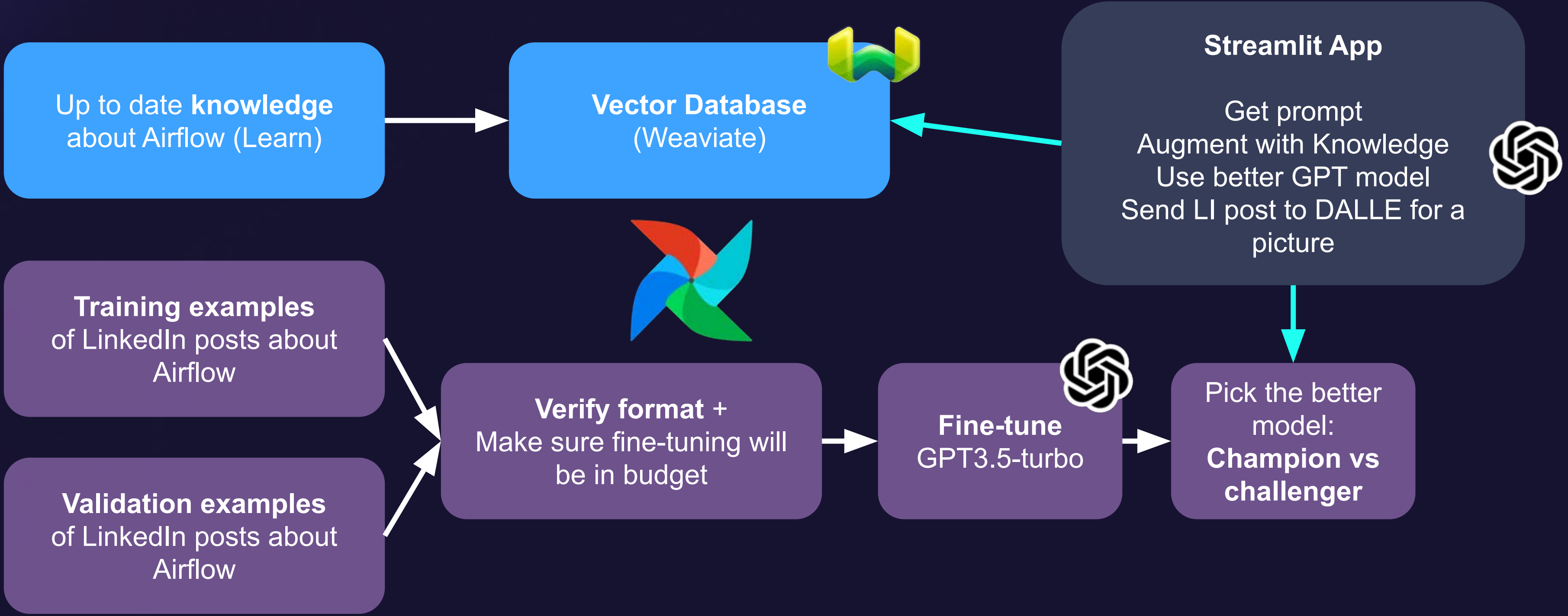


| MAP INDEX | STATE | DURATION | START DATE |
|---|---|---|---|
| Processed folder/file1 with constant: 42 | 🟩 success | 00:00:00 | 2024-04-22, 13:32:43 UTC |
| Processed folder/file2 with constant: 42 | 🟩 success | 00:00:00 | 2024-04-22, 13:32:43 UTC |

Details: https://astronomer.io/docs/learn/dynamic-tasks

# Demo repository: Content Generation

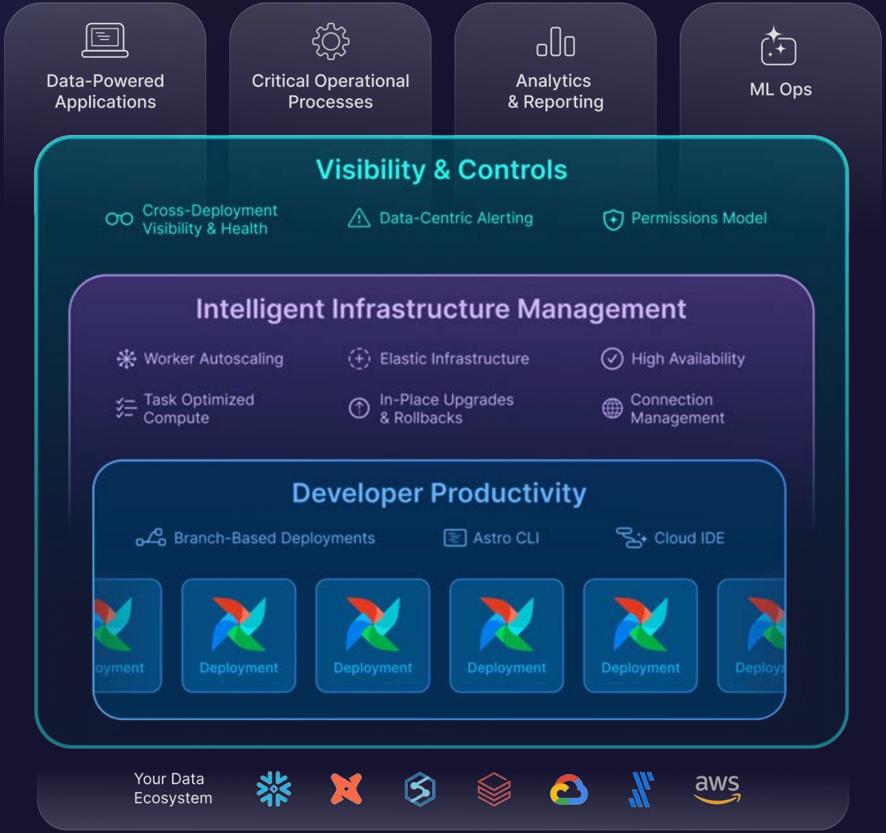
https://github.com/astronomer/gen-ai-fine-tune-rag-use-case


Up to date **knowledge** about Airflow (Learn)
**Vector Database** (Weaviate)
**Streamlit App**
Get prompt
Augment with Knowledge
Use better GPT model
Send LI post to DALLE for a picture
**Training examples** of LinkedIn posts about Airflow
**Validation examples** of LinkedIn posts about Airflow
**Verify format** + Make sure fine-tuning will be in budget
**Fine-tune** GPT3.5-turbo
Pick the better model: **Champion vs challenger**

## Demo

https://github.com/astronomer/gen-ai-fine-tune-rag-use-case

# Astro



## Supercharge Airflow with Astronomer

**Start Your 14-Day Free Trial**

(no credit card required)

New sign-ups receive $300 in credits *and* a complimentary Airflow Fundamentals Certification Exam (normally $150).

https://qrco.de/bf2ICP

ASTRONOMER

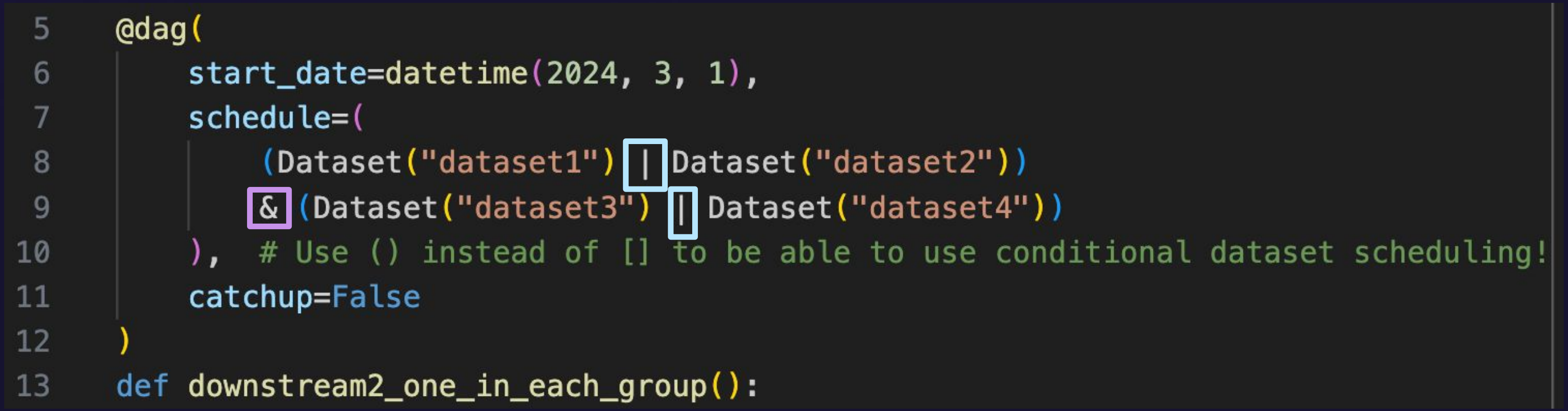
## Take Home Message:

Your data + your orchestration with Airflow is what sets you apart from competitors when creating GenAI applications!

## Appendix

## Feature focus: Advanced Dataset scheduling (2.9)

### Conditional Dataset Scheduling

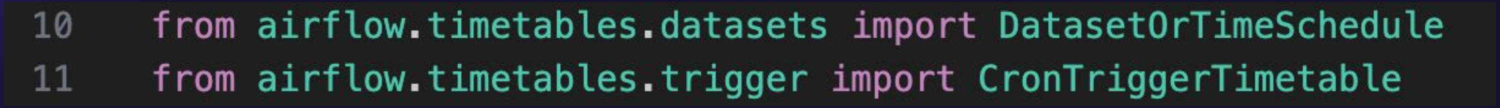


```
@dag(
    start_date=datetime(2024, 3, 1),
    schedule=(
        (Dataset("dataset1") | Dataset("dataset2"))
        & (Dataset("dataset3") | Dataset("dataset4"))
    ),  # Use () instead of [] to be able to use conditional dataset scheduling!
    catchup=False
)
def downstream2_one_in_each_group():
```

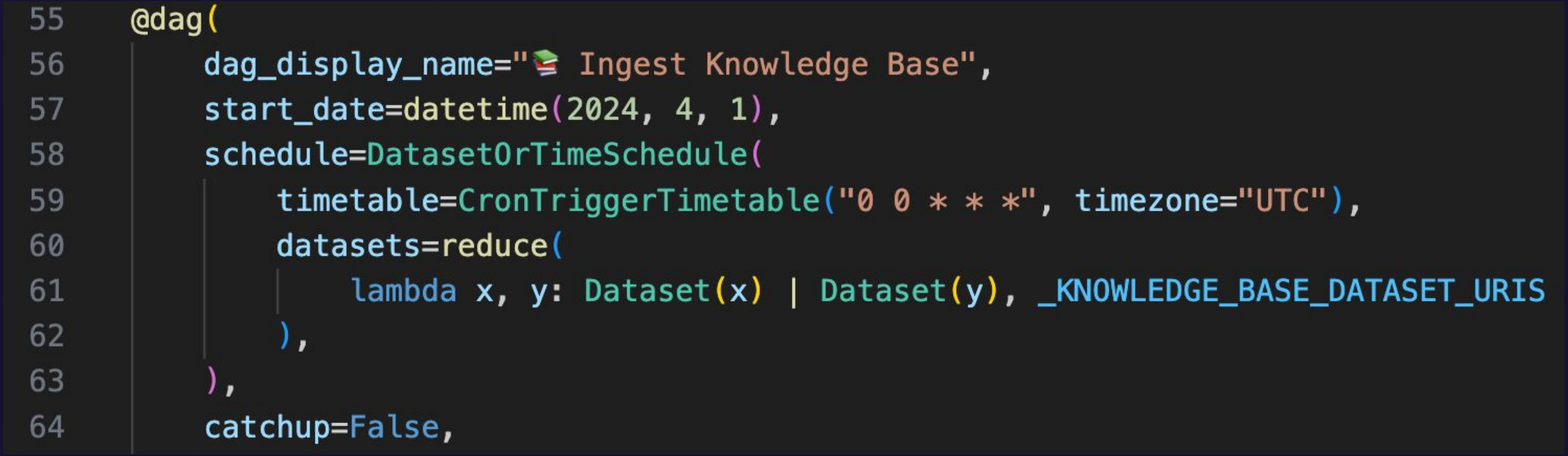
(Dataset 1 **OR** Dataset 2) **AND** (Dataset 3 **OR** Dataset 4)

## Feature focus: Advanced Dataset scheduling (2.9)

### Time + Dataset Scheduling



```python
from airflow.timetables.datasets import DatasetOrTimeSchedule
from airflow.timetables.trigger import CronTriggerTimetable
```



```python
@dag(
    dag_display_name="📚 Ingest Knowledge Base",
    start_date=datetime(2024, 4, 1),
    schedule=DatasetOrTimeSchedule(
        timetable=CronTriggerTimetable("0 0 * * *", timezone="UTC"),
        datasets=reduce(
            lambda x, y: Dataset(x) | Dataset(y), _KNOWLEDGE_BASE_DATASET_URIS
        ),
    ),
    catchup=False,
```