Chương 8

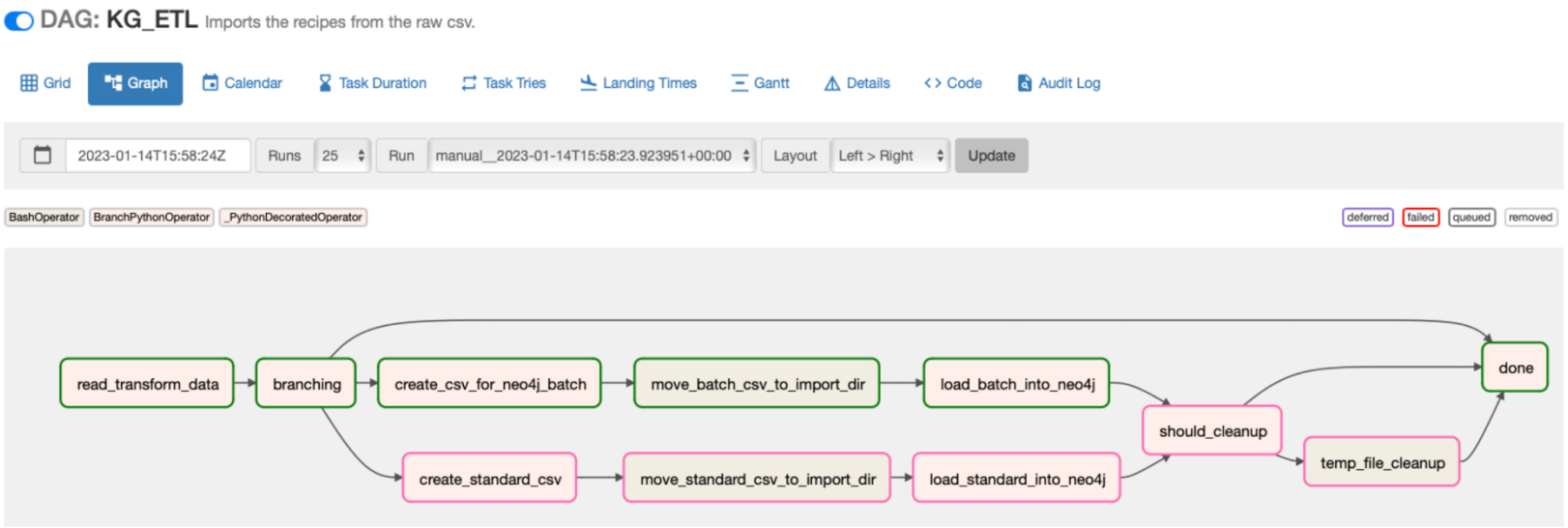
Apache Airflow: Orchestrating Data Workflows

What is Apache Airflow?

- An open-source platform for orchestrating complex computational workflows and data processing pipelines
- Pain points in data engineering
  - Dependency management between tasks
  - Scheduling and retries
  - Monitoring and error handling
  - Scalability of data pipelines

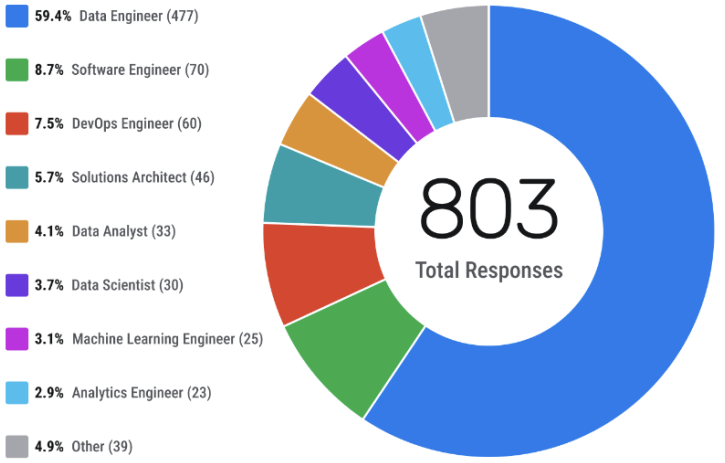
Airflow's Approach to Workflow Management

- Introduces the concept of Directed Acyclic Graphs (DAGs) for workflow representation
- Benefits
  - Clear visualization of task dependencies
  - Easier management of complex workflows
  - Improved reliability and error handling



https://airflow.apache.org/survey/

What best describes your current occupation?



According to the survey, more than half of Airflow users are Data Engineers (59.4%). The 2022 results are similar to those from 2019 and 2020 with a slight increase in the representation of Solutions Architect roles.

How important is Airflow to your business?

46%

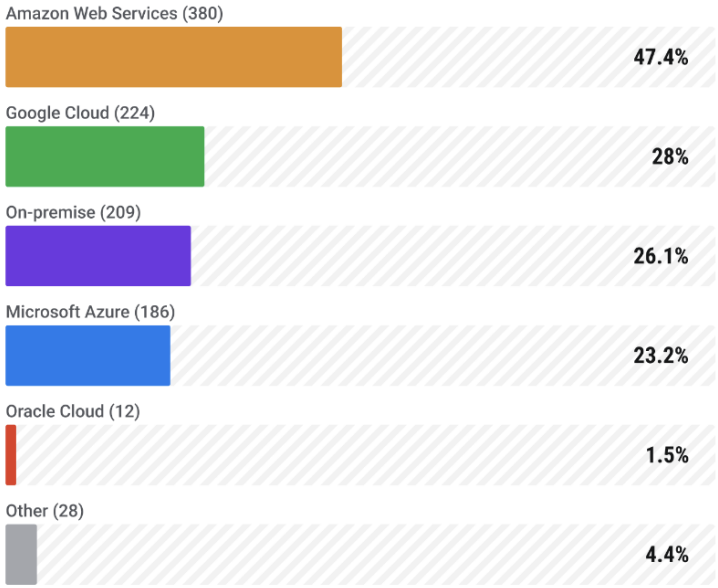
of Airflow users say that Airflow is very important to their business.

How often do you interact with Airflow?

55%

of users who responded use it on a daily basis; 26% on a weekly basis.

What is your company's primary cloud provider for Airflow?



How many people work at your company?

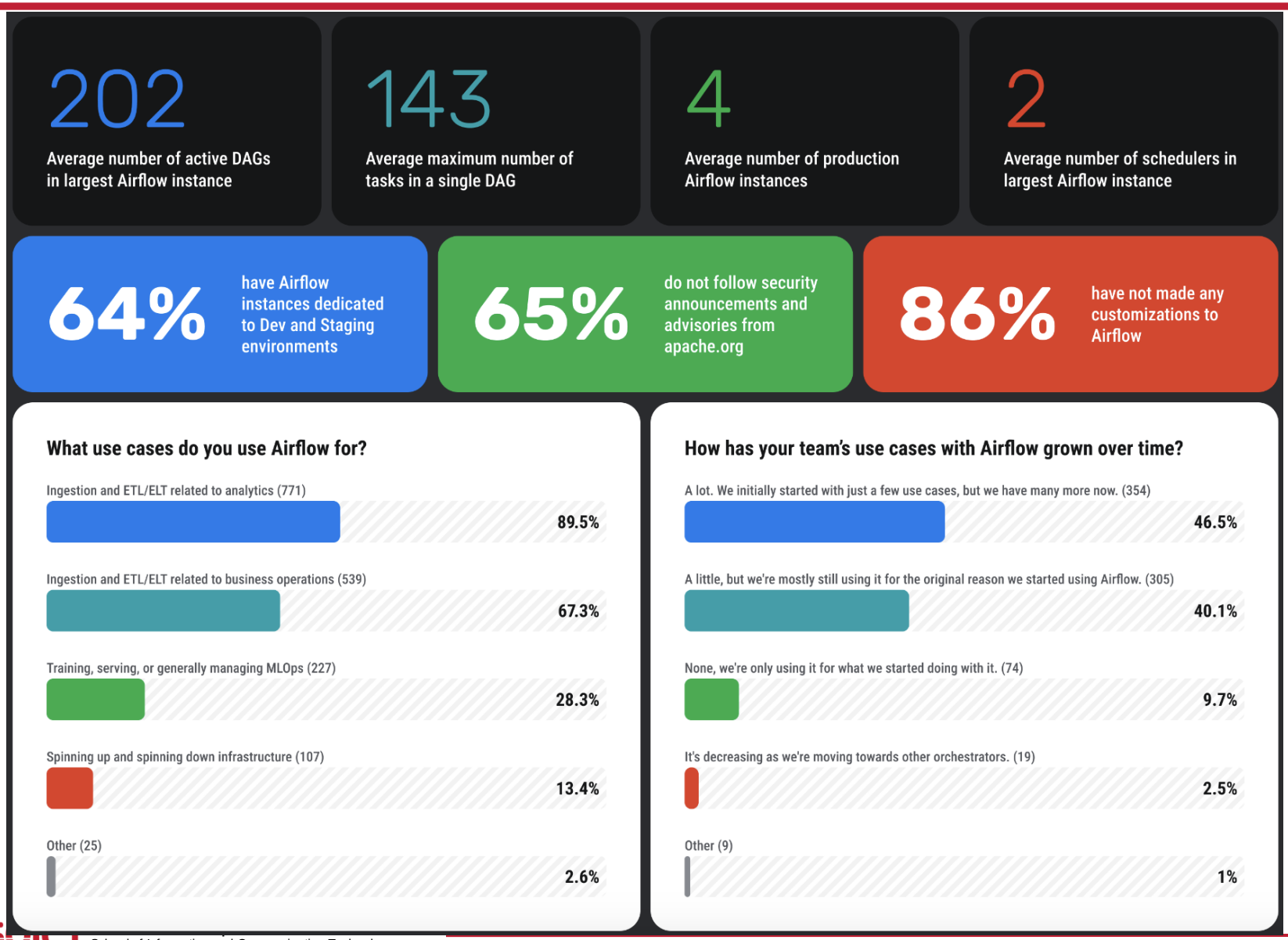
60%

of Airflow users work at companies with more than 200 employees.

How many people at your company use Airflow?

67%

of respondents have more than six Airflow users at their company.


202
Average number of active DAGs in largest Airflow instance
143
Average maximum number of tasks in a single DAG
4
Average number of production Airflow instances
2
Average number of schedulers in largest Airflow instance
64%
have Airflow instances dedicated to Dev and Staging environments
65%
do not follow security announcements and advisories from apache.org
86%
have not made any customizations to Airflow
What use cases do you use Airflow for?
Ingestion and ETL/ELT related to analytics (771)
89.5%
Ingestion and ETL/ELT related to business operations (539)
67.3%
Training, serving, or generally managing MLOps (227)
28.3%
Spinning up and spinning down infrastructure (107)
13.4%
Other (25)
2.6%
How has your team's use cases with Airflow grown over time?
A lot. We initially started with just a few use cases, but we have many more now. (354)
46.5%
A little, but we're mostly still using it for the original reason we started using Airflow. (305)
40.1%
None, we're only using it for what we started doing with it. (74)
9.7%
It's decreasing as we're moving towards other orchestrators. (19)
2.5%
Other (9)
1%

Airflow's Workflow Management Processes

- DAG Definition
- Scheduling
- Execution
- Monitoring
- Error Handling

DAG Definition: Workflows as Python Code

- DAGs are defined using Python, allowing for dynamic and programmatic creation
- Each DAG is a separate Python file in the DAGs folder
- DAGs consist of a series of tasks and their dependencies

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta

def task_1():
    print("Executing task 1")

def task_2():
    print("Executing task 2")

with DAG('example_dag',
         start_date=datetime(2024, 1, 1),
         schedule_interval=timedelta(days=1)) as dag:

    t1 = PythonOperator(task_id='task_1', python_callable=task_1)
    t2 = PythonOperator(task_id='task_2', python_callable=task_2)

    t1 >> t2  # Set task dependency
```

DAG Scheduling

- Cron-based Scheduling
  - Uses cron expressions for precise time-based scheduling
  - Example: 0 0 * * * (run daily at midnight)
- Interval-based Scheduling
  - Defines intervals between DAG runs
  - Example: timedelta(days=1) for daily runs
- External Triggers
  - Manual triggers via UI or API
  - Triggered by external systems or events

```
# Cron-based
dag = DAG('cron_dag', schedule_interval='0 0 * * *')
# Interval-based
dag = DAG('interval_dag', schedule_interval=timedelta(hours=6))
# Externally triggered
dag = DAG('on_demand_dag', schedule_interval=None)
```

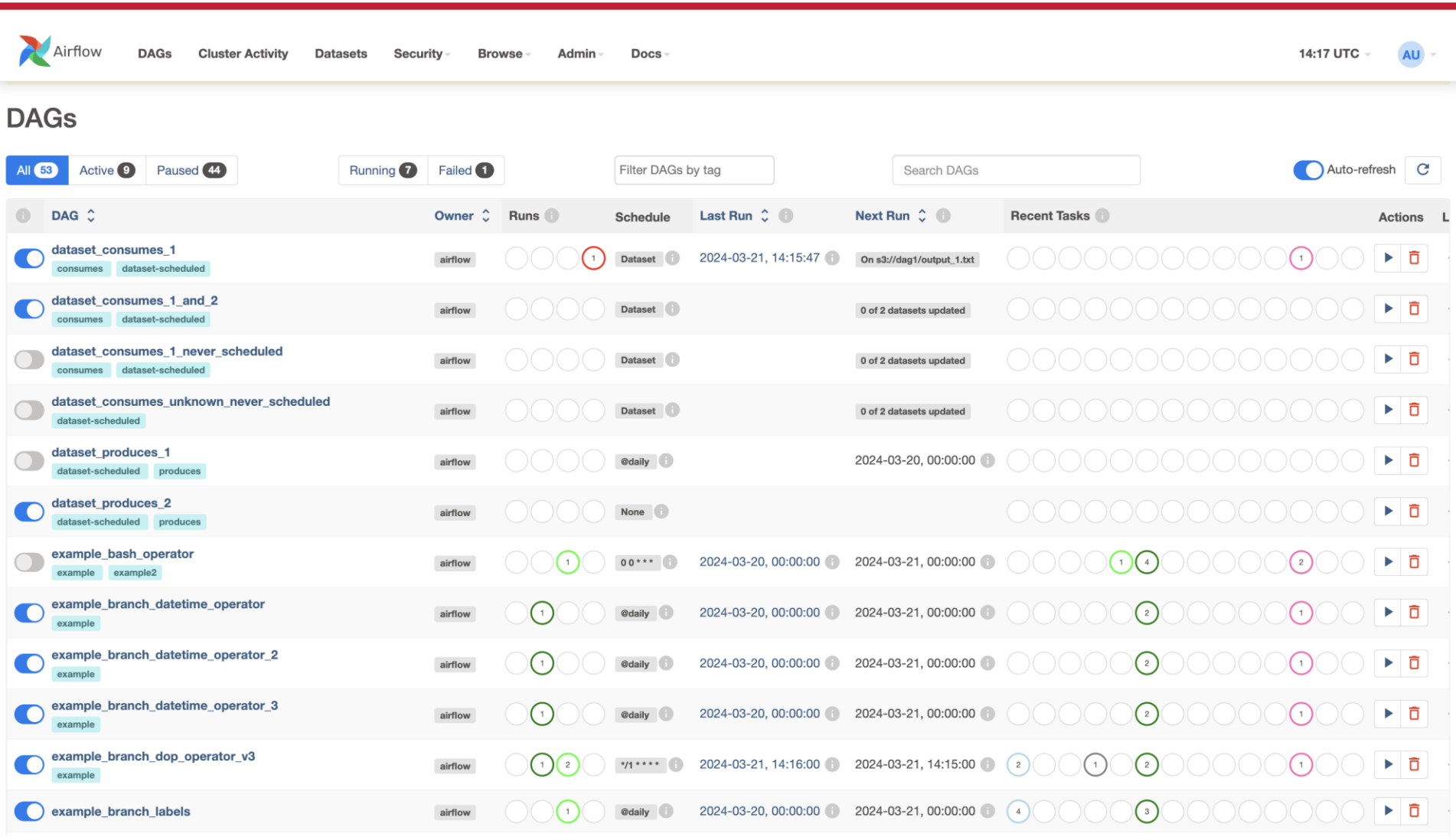
Task Execution Based on Dependencies

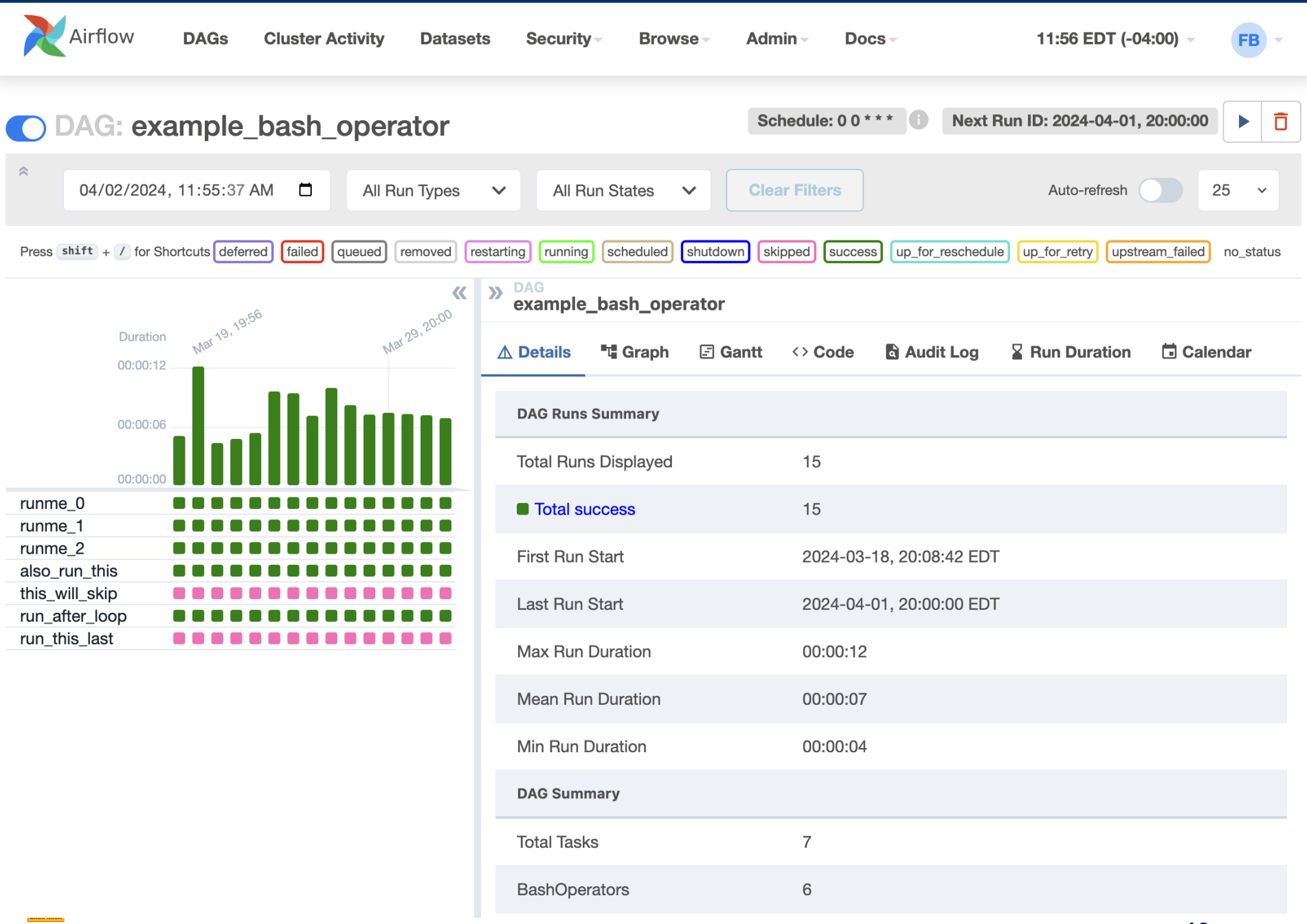
- Execution Process
  - DAG parsing and task instantiation
  - Dependency resolution
  - Task queuing
  - Worker assignment and execution
- Types of Dependencies
  - Temporal: Based on execution time or data intervals
  - Logical: Based on task relationships within the DAG
- Execution Modes
  - Sequential: One task at a time
  - Parallel: Multiple tasks simultaneously (based on dependencies)

Example

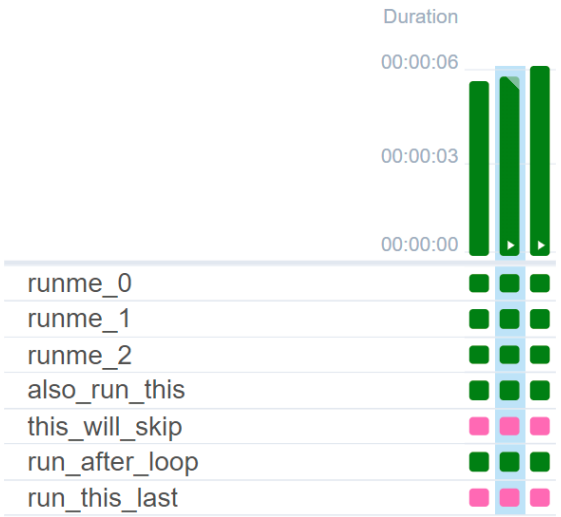
```
# Option 1: Using bitshift operators
task_1 >> task_2 >> task_3
# Option 2: Using set_upstream/set_downstream
task_3.set_upstream(task_2)
task_2.set_upstream(task_1)
# Option 3: Using list notation [task_1, task_2] >>
task_3
```

Real-time Monitoring through Web UI and Logging




Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
11:56 EDT (-04:00)
FB
DAG: example_bash_operator
Schedule: 0 0 * * *
Next Run ID: 2024-04-01, 20:00:00
04/02/2024, 11:55:37 AM
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
Mar 19, 19:56
Mar 29, 20:00
00:00:12
00:00:06
00:00:00
runme_0
runme_1
runme_2
also_run_this
this_will_skip
run_after_loop
run_this_last
DAG
example_bash_operator
Details
Graph
Gantt
Code
Audit Log
Run Duration
Calendar
DAG Runs Summary
Total Runs Displayed 15
Total success 15
First Run Start 2024-03-18, 20:08:42 EDT
Last Run Start 2024-04-01, 20:00:00 EDT
Max Run Duration 00:00:12
Mean Run Duration 00:00:07
Min Run Duration 00:00:04
DAG Summary
Total Tasks 7
BashOperators 6

deferred | failed | queued | removed | restarting | running | scheduled | shutdown | skipped | success | up_for_reschedule | up_for_retry | upstream_failed | no_status



DAG
example_bash_operator / Run
2023-05-27, 00:00:00 UTC

Clear | Mark state as...

Details | Graph | Code

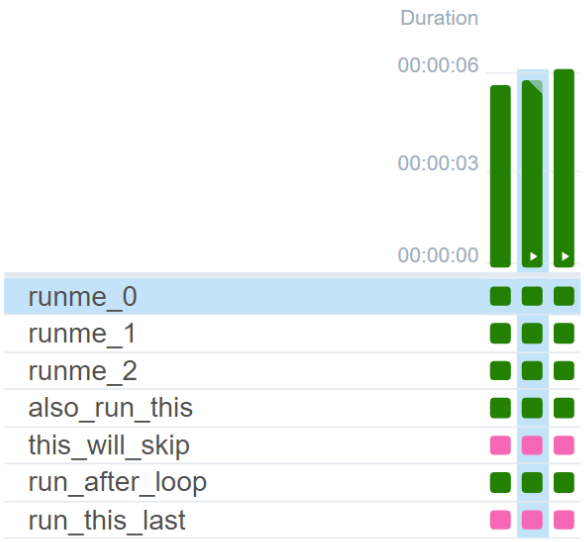
DAG Run Notes:

This is DagRun notes!

Edit Note

Status	success
Run ID	manual__2023-05-27T07:54:05.676224+00:00
Run type	manual
Run duration	00:00:05
Last scheduling decision	2023-05-27, 07:54:11 UTC
Queued at	2023-05-27, 07:54:05 UTC
Started	2023-05-27, 07:54:06 UTC
Ended	2023-05-27, 07:54:11 UTC
Data interval start	2023-05-26, 00:00:00 UTC
Data interval end	2023-05-27, 00:00:00 UTC
Externally triggered	True
Run config	None

deferred | failed | queued | removed | restarting | running | scheduled | shutdown | skipped | success | up_for_reschedule | up_for_retry | upstream_failed | no_status



DAG
example_bash_operator / Run ▶ 2023-05-27, 00:00:00 UTC / Task runme_0

Clear task | Mark state as... | Filter Tasks

Details | Graph | Logs

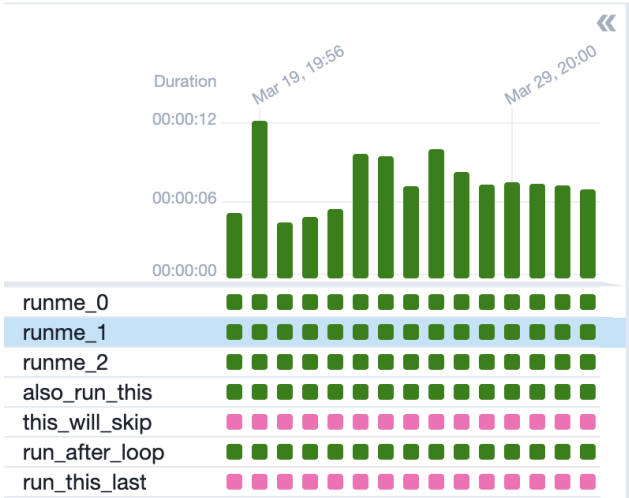
More Details | Rendered Template | XCom | List Instances, all runs

Task Instance Notes:

Add Note

Task Instance Details

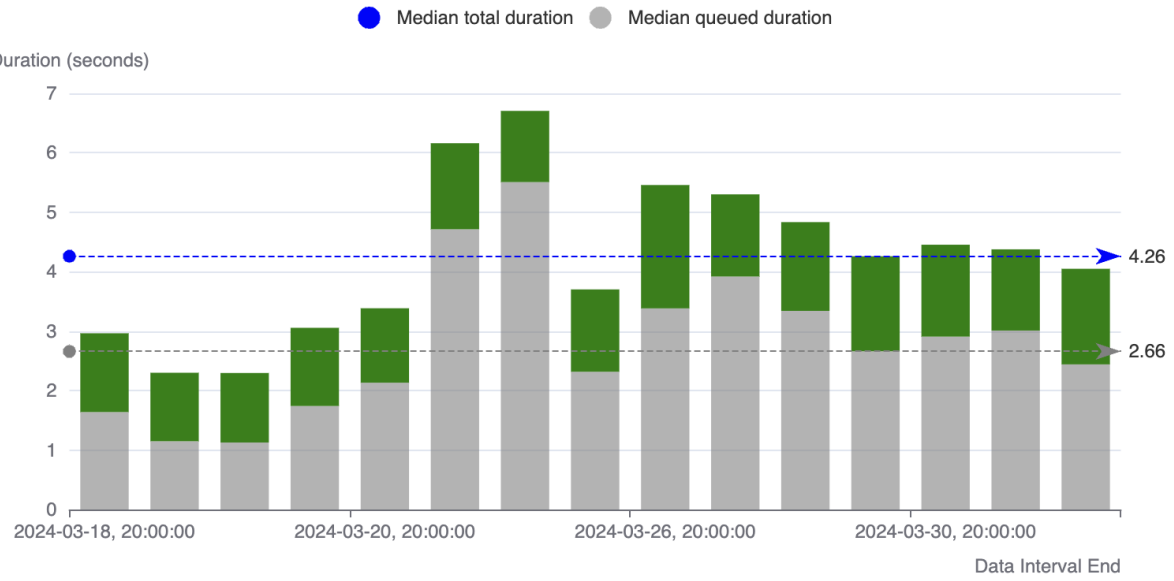
Status	success
Task ID	runme_0
Run ID	manual__2023-05-27T07:54:05.676224+00:00
Operator	BashOperator
Trigger Rule	all_success
Duration	00:00:01
Started	2023-05-27, 07:54:07 UTC
Ended	2023-05-27, 07:54:09 UTC



DAG
example_bash_operator / Task runme_1

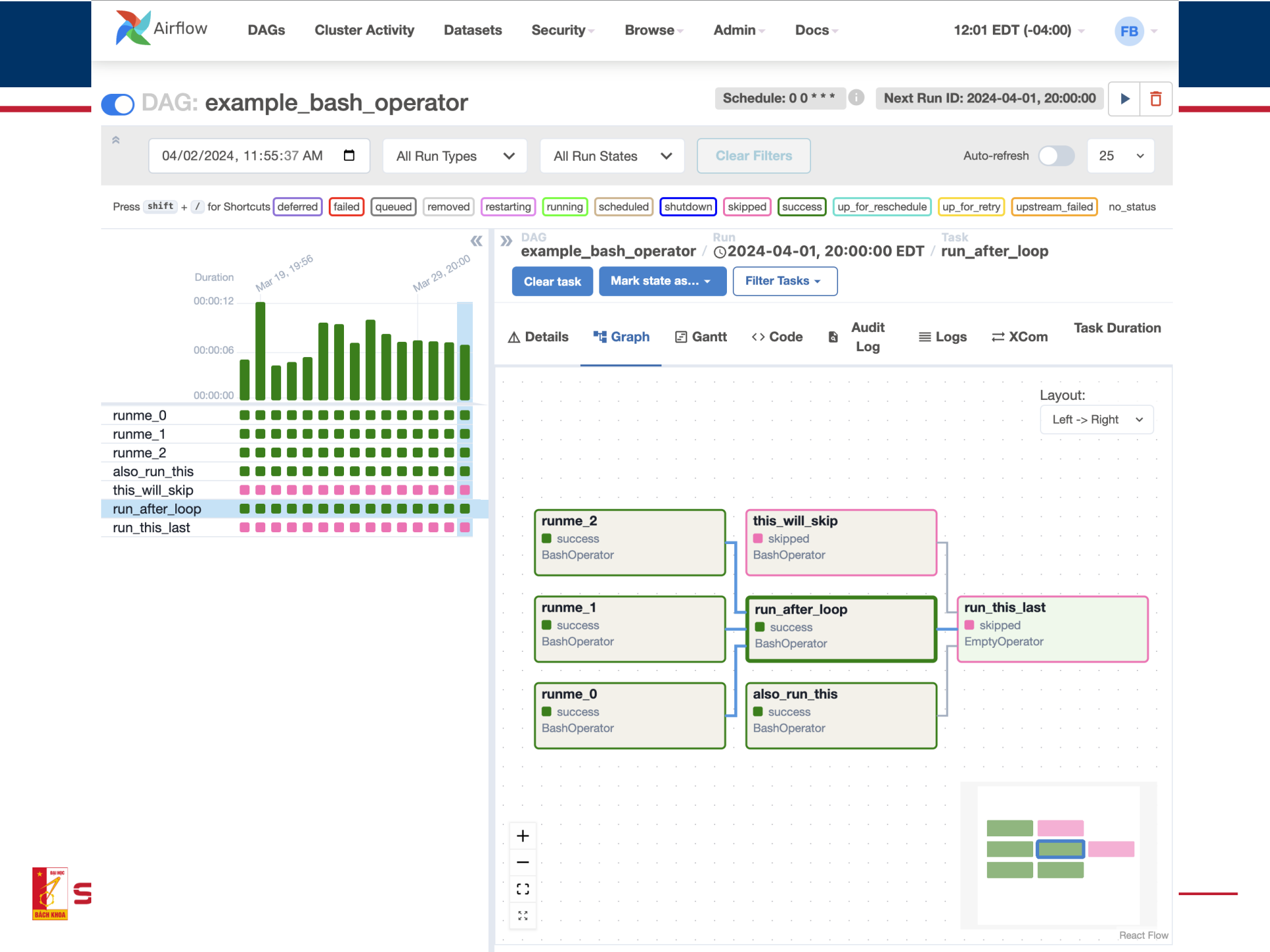
Details | Graph | Gantt | Code | Audit Log

Task Instance Duration

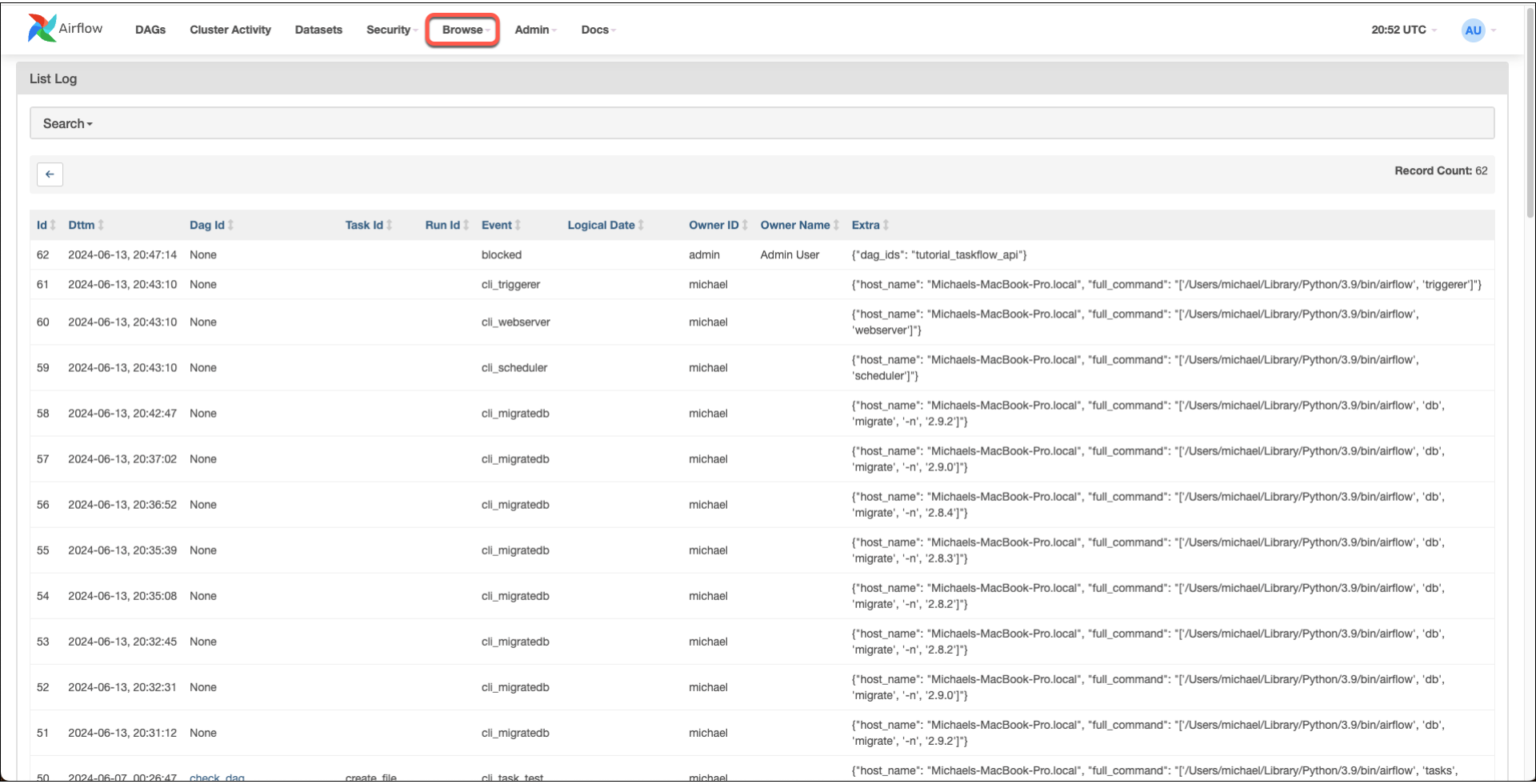


Task Details

Operator	BashOperator
Trigger Rule	all_success


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
12:01 EDT (-04:00)
FB
DAG: example_bash_operator
Schedule: 0 0 * * *
Next Run ID: 2024-04-01, 20:00:00
04/02/2024, 11:55:37 AM
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
Mar 19, 19:56
Mar 29, 20:00
00:00:12
00:00:06
00:00:00
runme_0
runme_1
runme_2
also_run_this
this_will_skip
run_after_loop
run_this_last
DAG
example_bash_operator
Run
2024-04-01, 20:00:00 EDT
Task
run_after_loop
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Code
Audit Log
Logs
XCom
Task Duration
Layout:
Left -> Right
runme_2
success
BashOperator
this_will_skip
skipped
BashOperator
runme_1
success
BashOperator
run_after_loop
success
BashOperator
run_this_last
skipped
EmptyOperator
runme_0
success
BashOperator
also_run_this
success
BashOperator
React Flow

Airflow logging



Id	Dttm	Dag Id	Task Id	Run Id	Event	Logical Date	Owner ID	Owner Name	Extra
62	2024-06-13, 20:47:14	None			blocked		admin	Admin User	{"dag_ids": "tutorial_taskflow_api"}
61	2024-06-13, 20:43:10	None			cli_triggerer		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'triggerer']"}
60	2024-06-13, 20:43:10	None			cli_webserver		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'webserver']"}
59	2024-06-13, 20:43:10	None			cli_scheduler		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'scheduler']"}
58	2024-06-13, 20:42:47	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.9.2']"}
57	2024-06-13, 20:37:02	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.9.0']"}
56	2024-06-13, 20:36:52	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.8.4']"}
55	2024-06-13, 20:35:39	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.8.3']"}
54	2024-06-13, 20:35:08	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.8.2']"}
53	2024-06-13, 20:32:45	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.8.2']"}
52	2024-06-13, 20:32:31	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.9.0']"}
51	2024-06-13, 20:31:12	None			cli_migratedb		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'db', 'migrate', '-n', '2.9.2']"}
50	2024-06-07, 00:26:47	check_dag	create_file		cli_task_test		michael		{"host_name": "Michaels-MacBook-Pro.local", "full_command": "['/Users/michael/Library/Python/3.9/bin/airflow', 'tasks',

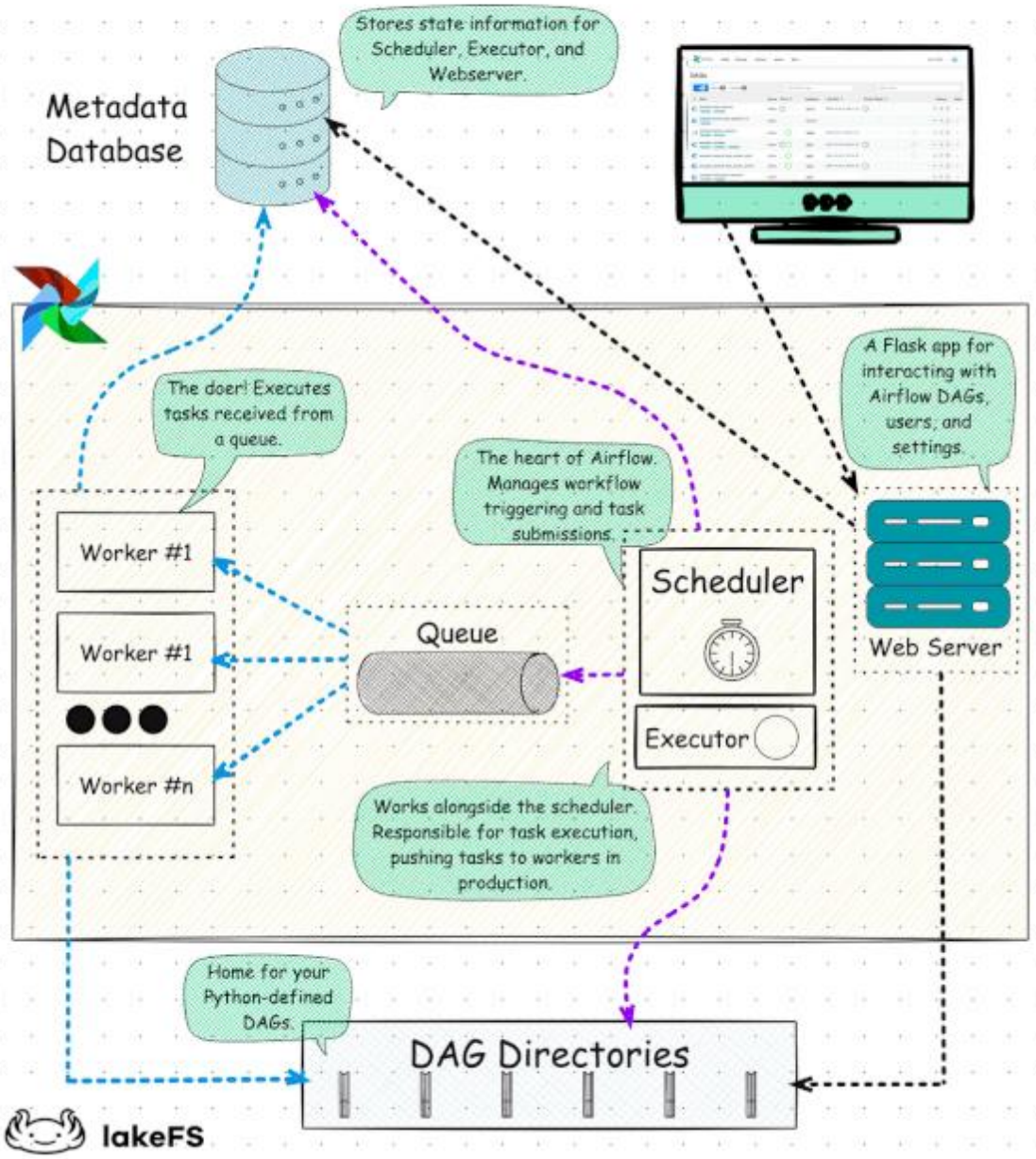
Airflow DAG notifications

```
@task(
        email=["noreply@astronomer.io",  "noreply2@astronomer.io"],
        email_on_failure=True,
        email_on_retry=True
)
def t1():
        return "hello”


from airflow.utils.email import send_email

def success_email_function(context):
        dag_run = context.get("dag_run")

        msg = "DAG ran successfully"
        subject = f"DAG {dag_run} has completed"
        send_email(to=your_emails, subject=subject, html_content=msg)

@dag(
        start_date=datetime(2023, 4, 26),
        schedule="@daily",
        catchup=False,
        on_success_callback=success_email_function
)
```

Robust Error Handling and Retry Mechanisms

- Automatic Retries
  - Configurable retry attempts and intervals
  - Exponential backoff support
- Failure Notifications
  - Email alerts on task failures
  - Customizable notification rules
- Conditional Execution
  - Trigger specific tasks on failure of others

Airflow- Architecture



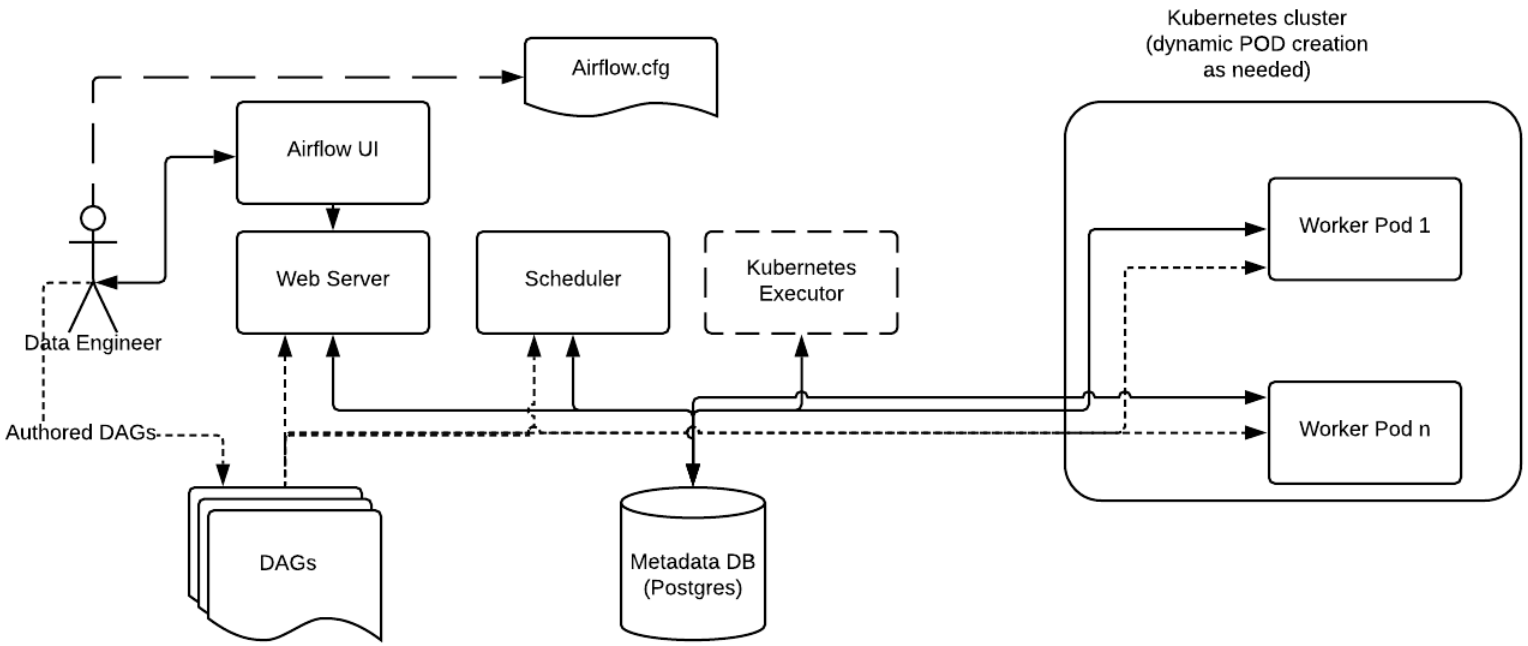
Airflow Executors

- SequentialExecutor
  - Default executor
  - Runs one task at a time
  - Uses SQLite as the Metadata DB
  - Pros: Simple, easy to set up
  - Cons: Not suitable for production or parallel execution
  - Use case: Development and testing
- LocalExecutor
  - Runs tasks on the same machine as the Scheduler
  - Can run tasks in parallel using multiple processes
  - Pros: Easy to set up, good for medium workloads
  - Cons: Limited by single machine resources
  - Use case: Small to medium-scale deployments

Airflow Executors (2)

- CeleryExecutor
  - Uses Celery to run distributed tasks
  - Allows for horizontal scaling across multiple worker machines
  - Pros: Highly scalable, good for large workloads
  - Cons: More complex setup, requires additional components (Redis/RabbitMQ)
  - Use case: Large-scale production deployments
- KubernetesExecutor
  - Runs each task in a separate Kubernetes pod
  - Allows for dynamic allocation of resources
  - Pros: Highly scalable, efficient resource utilization
  - Cons: Requires Kubernetes cluster, complex setup
  - Use case: Cloud-native environments, need for isolation between tasks

KubernetesExecutor



HUST

hust.edu.vn fb.com/dhbkhn

Thank you for your attention!

Q&A