OSA CON 23

`New` Workflow Orchestrator in town:

"Apache Airflow 2.x"

Jarek Potiuk

Apache Airflow Committer

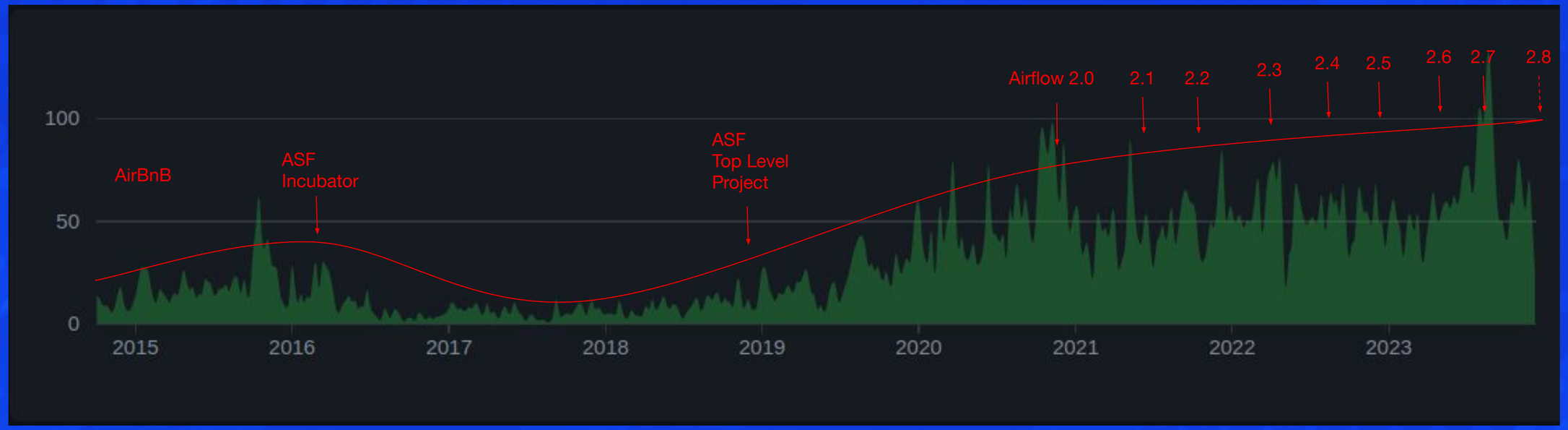
https://github.com/potiuk

December 12-14 2023

Airflow as ..

- Modern UI
- With great DAG Authoring capabilities
- Being extensible platform
- Being True Open Source with as strong community you can get
- Solid infrastructure
- Shortly - modern orchestrator of your choice :)

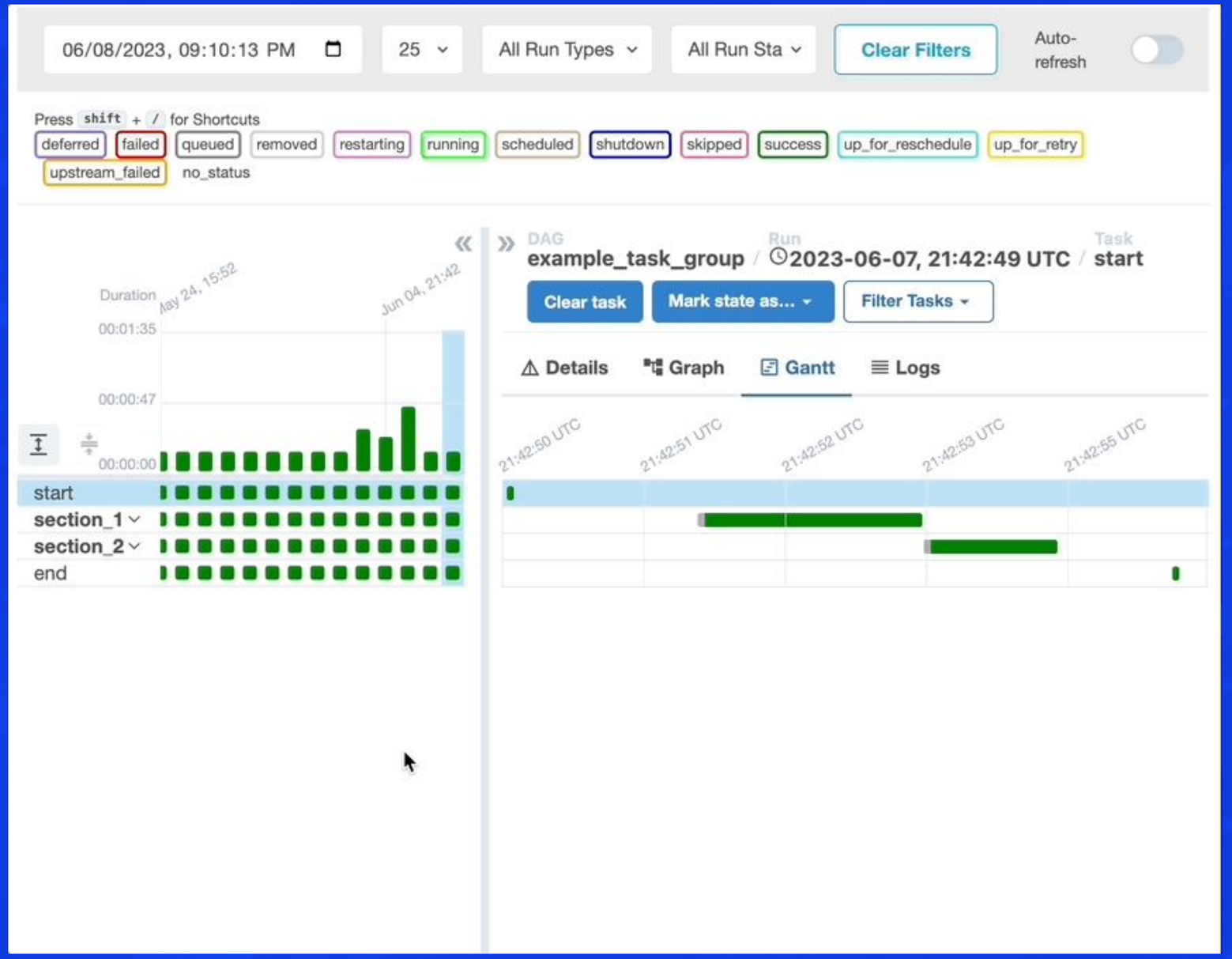
Airflow 2 timeline



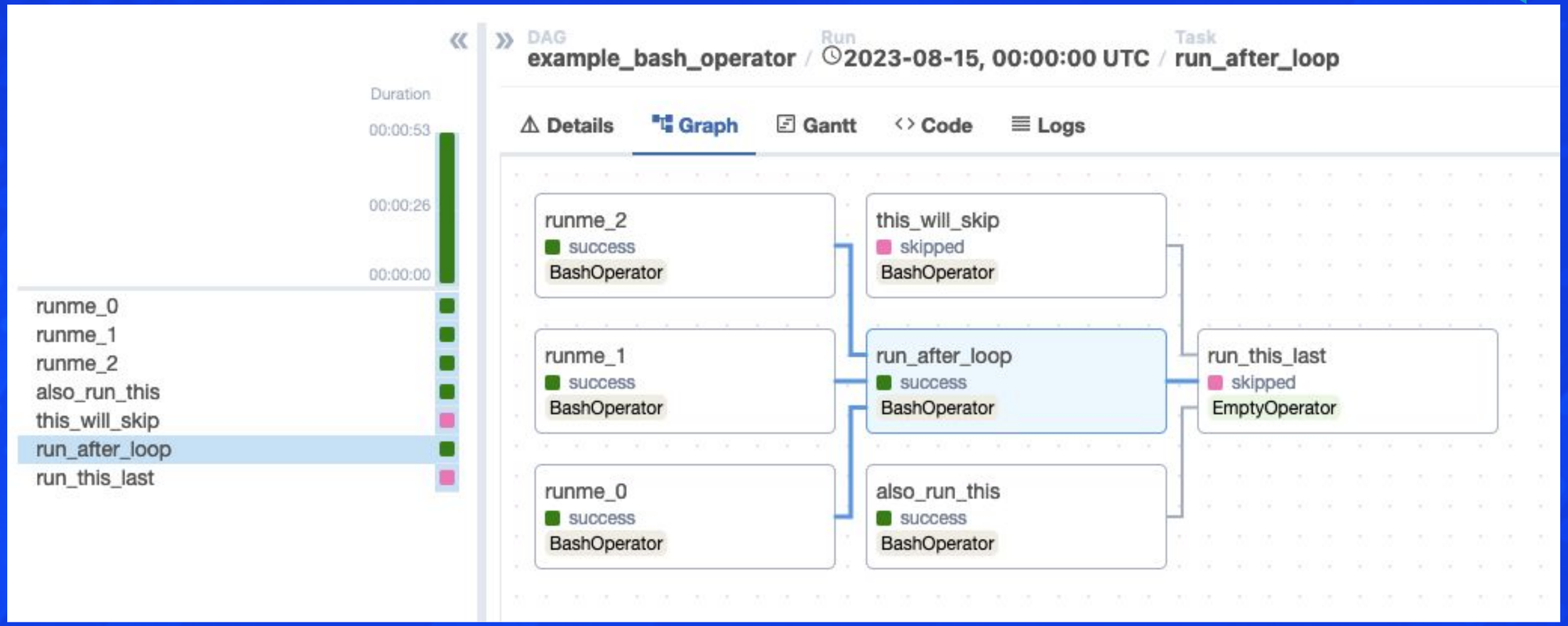
10th Anniversary next year:
Unofficial
Airflow Summit 2024, September, Bay Area, 1000+ attendees

Modern UI

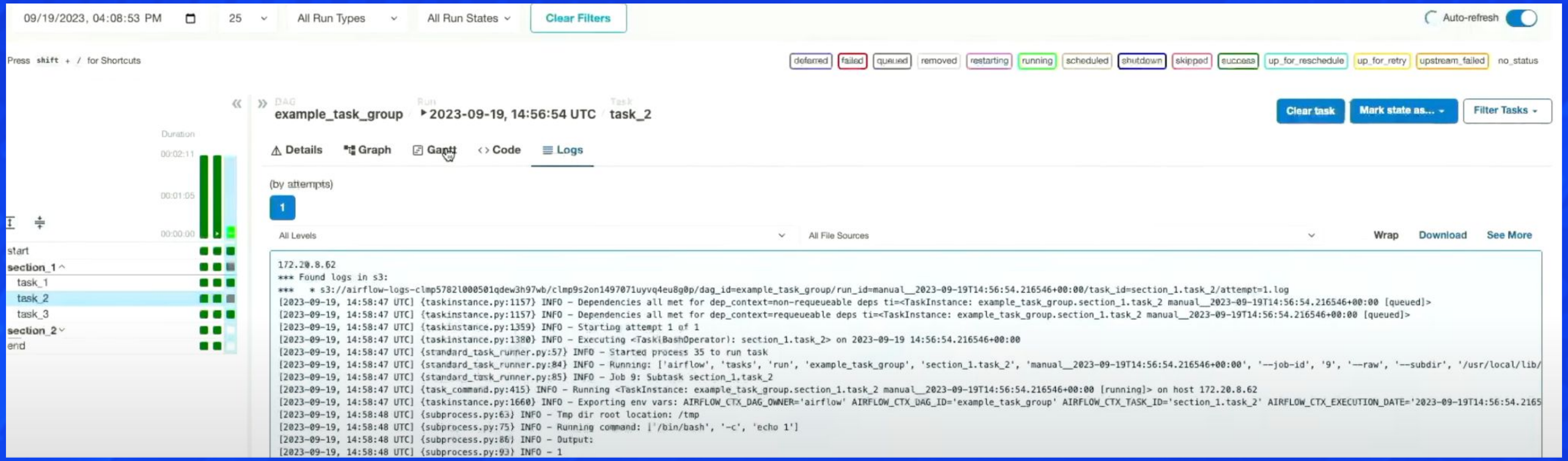
Grid View


06/08/2023, 09:10:13 PM
25
All Run Types
All Run Sta
Clear Filters
Auto-refresh
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
May 24, 15:52
Jun 04, 21:42
00:01:35
00:00:47
00:00:00
start
section_1
section_2
end
DAG
example_task_group
Run
2023-06-07, 21:42:49 UTC
Task
start
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Logs
21:42:50 UTC
21:42:51 UTC
21:42:52 UTC
21:42:53 UTC
21:42:55 UTC

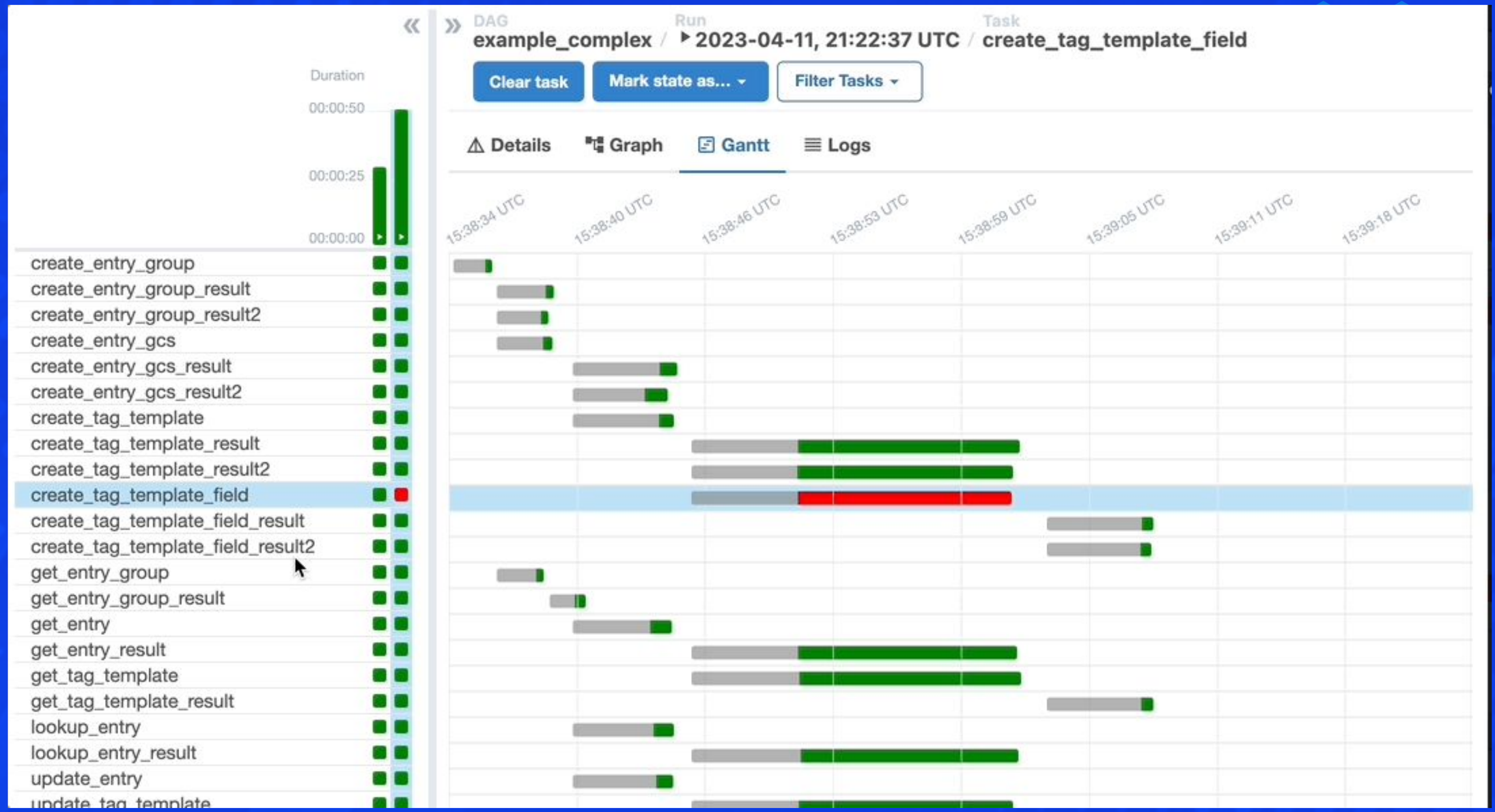
Graph View


DAG
example_bash_operator
Run
2023-08-15, 00:00:00 UTC
Task
run_after_loop
Details
Graph
Gantt
Code
Logs
Duration
00:00:53
00:00:26
00:00:00
runme_0
runme_1
runme_2
also_run_this
this_will_skip
run_after_loop
run_this_last
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

Log view


09/19/2023, 04:08:53 PM
25
All Run Types
All Run States
Clear Filters
Auto-refresh
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
DAG
example_task_group
Run
2023-09-19, 14:56:54 UTC
Task
task_2
Clear task
Mark state as...
Filter Tasks
Duration
00:02:11
00:01:05
00:00:00
start
section_1
task_1
task_2
task_3
section_2
end
Details
Graph
Gantt
Code
Logs
(by attempts)
1
All Levels
All File Sources
Wrap
Download
See More
172.20.8.62
*** Found logs in s3:
*** * s3://airflow-logs-clmp5782l000501qdew3h97wb/clmp9s2on1497071uyvq4eu8g0p/dag_id=example_task_group/run_id=manual__2023-09-19T14:56:54.216546+00:00/task_id=section_1.task_2/attempt=1.log
[2023-09-19, 14:58:47 UTC] {taskinstance.py:1157} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: example_task_group.section_1.task_2 manual__2023-09-19T14:56:54.216546+00:00 [queued]>
[2023-09-19, 14:58:47 UTC] {taskinstance.py:1157} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: example_task_group.section_1.task_2 manual__2023-09-19T14:56:54.216546+00:00 [queued]>
[2023-09-19, 14:58:47 UTC] {taskinstance.py:1359} INFO - Starting attempt 1 of 1
[2023-09-19, 14:58:47 UTC] {taskinstance.py:1380} INFO - Executing <Task(BashOperator): section_1.task_2> on 2023-09-19 14:56:54.216546+00:00
[2023-09-19, 14:58:47 UTC] {standard_task_runner.py:57} INFO - Started process 35 to run task
[2023-09-19, 14:58:47 UTC] {standard_task_runner.py:84} INFO - Running: ['airflow', 'tasks', 'run', 'example_task_group', 'section_1.task_2', 'manual__2023-09-19T14:56:54.216546+00:00', '--job-id', '9', '--raw', '--subdir', '/usr/local/lib/
[2023-09-19, 14:58:47 UTC] {standard_task_runner.py:85} INFO - Job 9: Subtask section_1.task_2
[2023-09-19, 14:58:47 UTC] {task_command.py:415} INFO - Running <TaskInstance: example_task_group.section_1.task_2 manual__2023-09-19T14:56:54.216546+00:00 [running]> on host 172.20.8.62
[2023-09-19, 14:58:47 UTC] {taskinstance.py:1660} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='airflow' AIRFLOW_CTX_DAG_ID='example_task_group' AIRFLOW_CTX_TASK_ID='section_1.task_2' AIRFLOW_CTX_EXECUTION_DATE='2023-09-19T14:56:54.2165
[2023-09-19, 14:58:48 UTC] {subprocess.py:63} INFO - Tmp dir root location: /tmp
[2023-09-19, 14:58:48 UTC] {subprocess.py:75} INFO - Running command: ['/bin/bash', '-c', 'echo 1']
[2023-09-19, 14:58:48 UTC] {subprocess.py:86} INFO - Output:
[2023-09-19, 14:58:48 UTC] {subprocess.py:93} INFO - 1

Gantt view


DAG
example_complex
Run
2023-04-11, 21:22:37 UTC
Task
create_tag_template_field
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Logs
Duration
00:00:50
00:00:25
00:00:00
15:38:34 UTC
15:38:40 UTC
15:38:46 UTC
15:38:53 UTC
15:38:59 UTC
15:39:05 UTC
15:39:11 UTC
15:39:18 UTC
create_entry_group
create_entry_group_result
create_entry_group_result2
create_entry_gcs
create_entry_gcs_result
create_entry_gcs_result2
create_tag_template
create_tag_template_result
create_tag_template_result2
create_tag_template_field
create_tag_template_field_result
create_tag_template_field_result2
get_entry_group
get_entry_group_result
get_entry
get_entry_result
get_tag_template
get_tag_template_result
lookup_entry
lookup_entry_result
update_entry

Cluster Activity

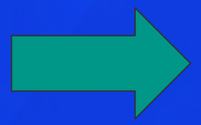
DAG Authoring

Handling dependencies

- An issue in early Airflow 2.0 days - much less nowadays
- Multiple options to handle it
  - Python Virtualenv Operator, External Python Operator, Docker Operator, Kubernetes Pod Operator, Multiple Docker Images + Celery Queues
  - Coming soon -> Multi-tenancy setup with per-tenant dependencies
- Mastering dependencies: The Airflow Way talk from Airflow Summit 2023
- Plays super-well with Task Flow

TaskFlow

```python
def choose_mode():
  accuracy = 6
  if accuracy > 5:
    return "accurate"
  return 'inaccurate'

choose_best_model = BranchPythonOPerator(
  task_id = 'choose_best_model',
  python_callable = choose_best_model
)
```



```python
@task.branch
def choose_best_model():
  accuracy = 6
  if accuracy > 5:
    return 'accurate'
  return 'inaccurate'
```

Task Flow cases

Core:

- @dag
- @task.python
- @task.virtualenv
- @task.external_python
- @task.sensor
- @task.branch
- @task.short_circuit
- @task.bash (coming)

Providers:

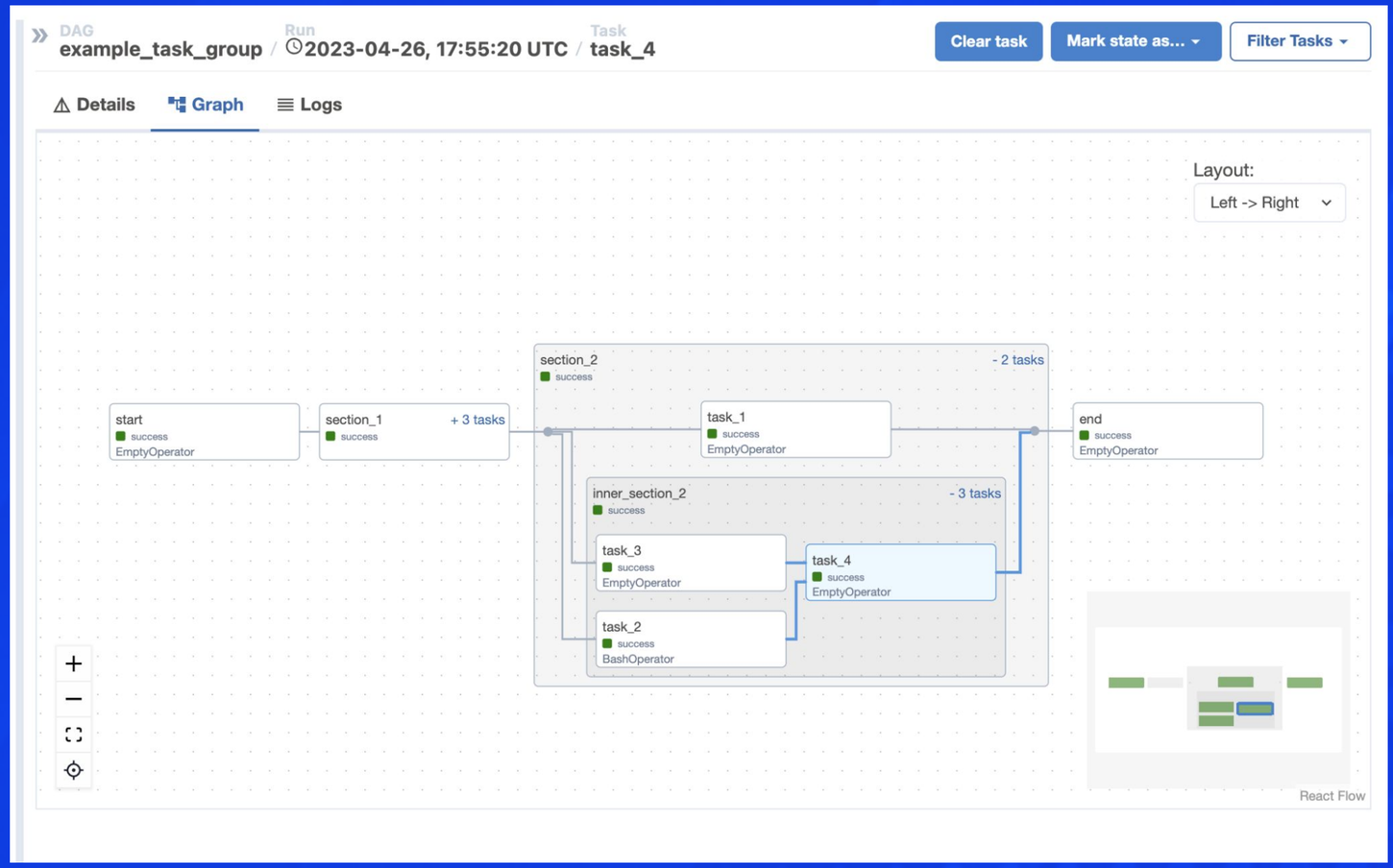
- @task.docker
- @task.kubernetes
- @task.sftp_sensor
- …
- Providers can provide their own

… and

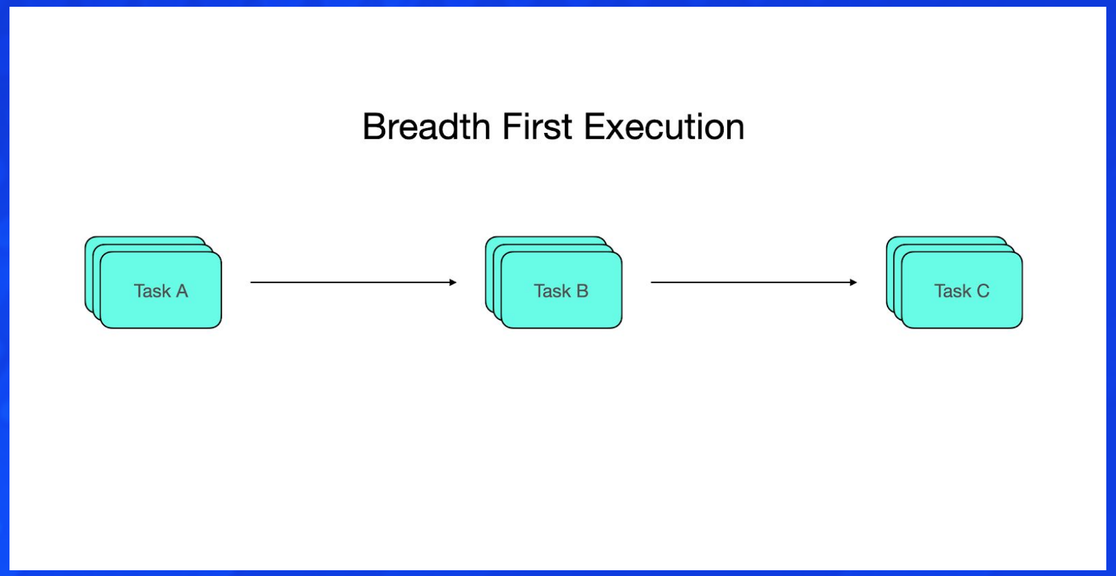
- @task_group

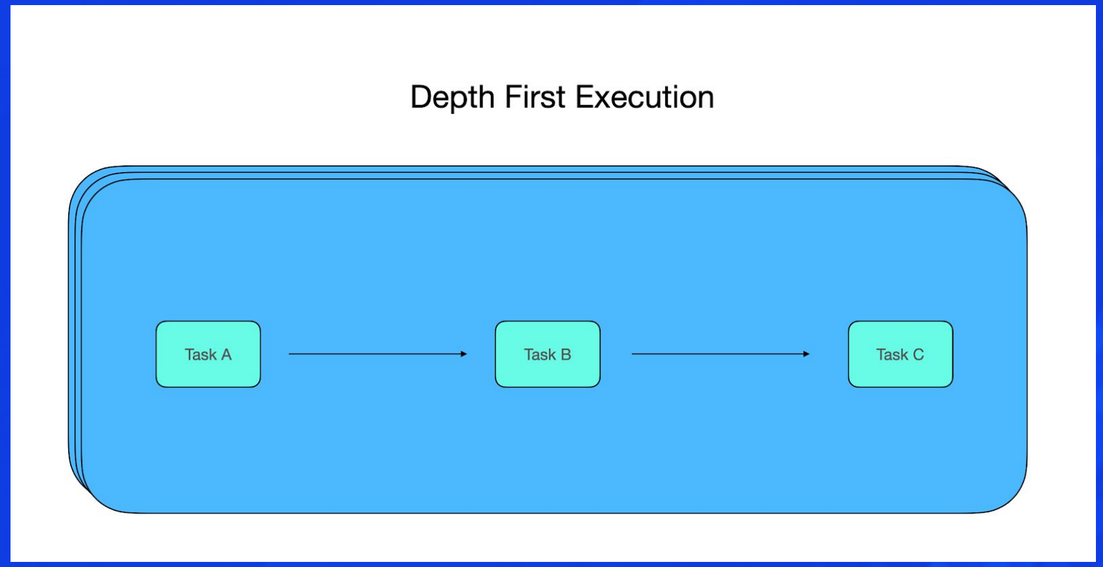
Task Groups

```python
@task
def task_1(value: int) -> str:
    """Empty Task1"""
    return f"[ Task1 {value} ]"


@task
def task_2(value: str) -> str:
    """Empty Task2"""
    return f"[ Task2 {value} ]"

@task
def task_3(value: str) -> None:
    """Empty Task3"""
    print(f"[ Task3 {value} ]")

@task_group
def task_group_function(value: int) -> None:

  task_3(task_2(task_1(value)))
```



Dynamic Task and Group mapping

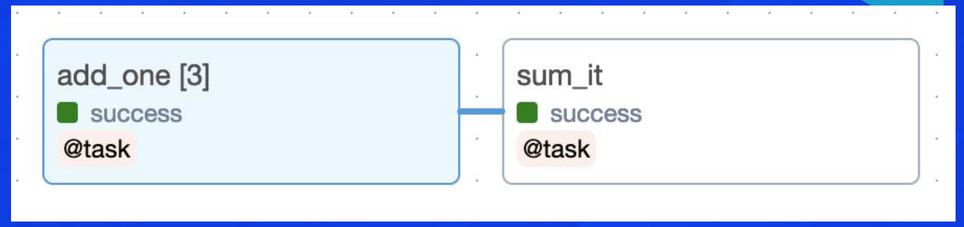


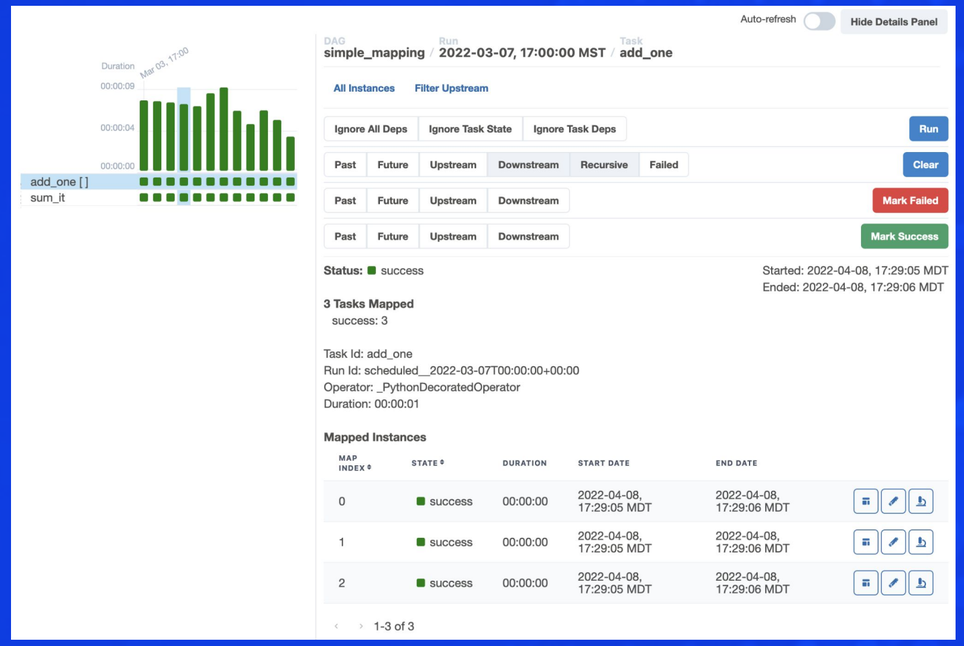


- Map Reduce - kind of workflows if you want Airflow to also “do stuff”
- You can parallelise even complex workflows

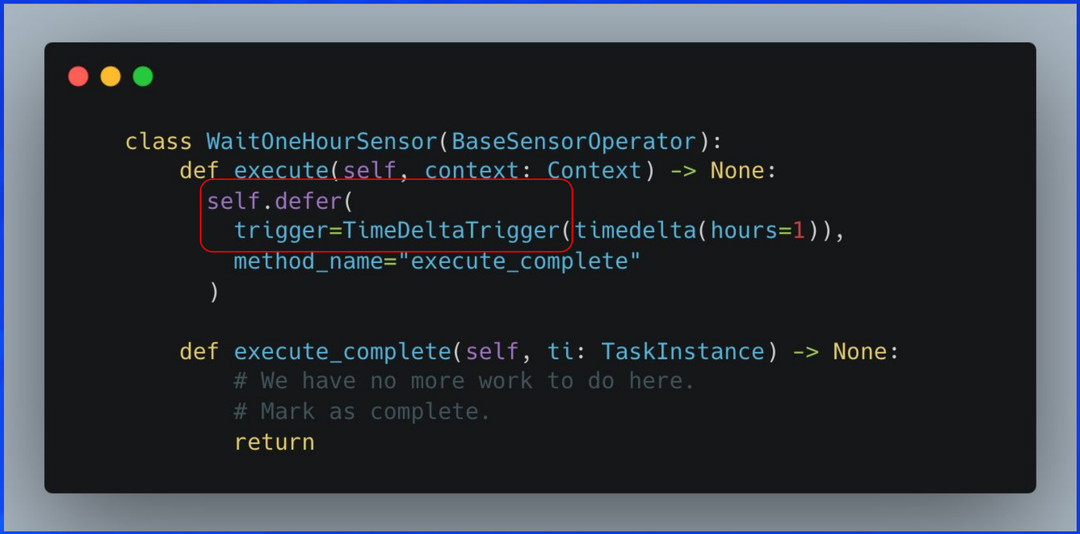
Dynamic Task mapping

```python
@task
def add_one(x: int):
    return x + 1

@task
def sum_it(values):
    total = sum(values)
    print(f"Total was {total}")

added_values = add_one.expand(x=[1, 2, 3])
sum_it(added_values)
```





Deferrable (AsyncIO) operators

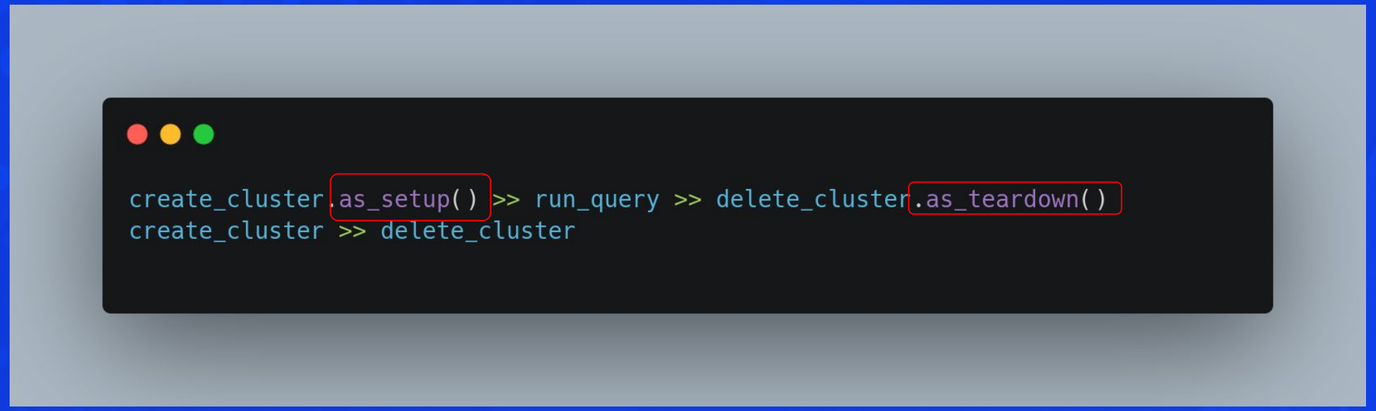


```python
class WaitOneHourSensor(BaseSensorOperator):
    def execute(self, context: Context) -> None:
      self.defer(
        trigger=TimeDeltaTrigger(timedelta(hours=1)),
        method_name="execute_complete"
      )

    def execute_complete(self, ti: TaskInstance) -> None:
        # We have no more work to do here.
        # Mark as complete.
        return
```

80% - 90% performance improvements

- no worker slots while waiting (other jobs can run)
- multiple 100s of Deferrable Operators out-of-the-box
- 10s of Triggers available
- you can roll your own Trigger

Setup/Teardown



```
create_cluster.as_setup() >> run_query >> delete_cluster.as_teardown()
create_cluster >> delete_cluster
```

Notifiers

```python
from airflow.notifications.basenotifier import BaseNotifier
from my_provider import send_message


class MyNotifier(BaseNotifier):
    template_fields = ("message",)

    def __init__(self, message):
        self.message = message

    def notify(self, context):
        # Send notification here, below is an example
        title = f"Task {context['task_instance'].task_id} failed"
        send_message(title, self.message)
```

```python
from datetime import datetime

from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator

from myprovider.notifier import MyNotifier

with DAG(
    dag_id="example_notifier",
    start_date=datetime(2022, 1, 1),
    schedule_interval=None,
    on_success_callback=MyNotifier(message="Success!"),
    on_failure_callback=MyNotifier(message="Failure!"),
):
    task = BashOperator(
        task_id="example_task",
        bash_command="exit 1",
        on_success_callback=MyNotifier(message="Task Succeeded!"),
    )
```

- easily reusable notifiers when your task fails (or not)

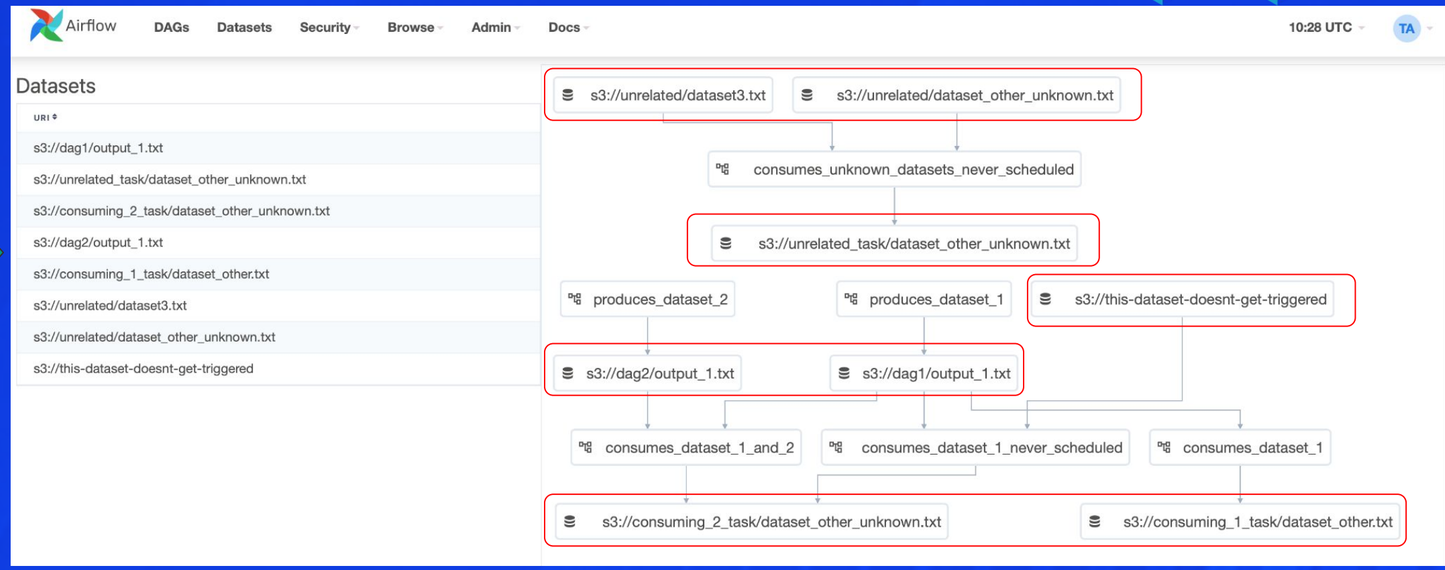
Object storage - FsSpec (Coming in Airflow 2.8)

- Open standard
- Integrates with all object storages
- Modern Pythonic way of interacting
  - Pathlib
- Supported by:
  - Pandas, Polars, Parquet, DuckDB, Iceberg, PyArrow
- One way to rule them all

```python
fs_base = ObjectStoragePath("s3://airflow-tutorial-data/", conn_id="aws_default")

# ensure the bucket exists
fs_base.mkdir(exist_ok=True)

formatted_date = execution_date.format("YYYYMMDD")
path = fs_base / f"air_quality_{formatted_date}.parquet"

df = pd.DataFrame(response.json()).astype(foelds)
with path.open("wb") as file:
    df.to_parquet(file)

return path
```

Data-aware scheduling

```python
example_dataset = Dataset("s3://dataset/example.csv")

with DAG(dag_id="producer", ...):
    BashOperator(task_id="producer", outlets=[example_dataset], ...)

with DAG(dag_id="consumer", schedule=[example_dataset], ...):
    ...
```



- Micropipelines concept
- Still early days
- But mind-boggling things are coming (Object storage integration, Partial Datasets, Data aware triggering, Open Lineage)

LLM Operators

Donated by Astronomer (yay!)

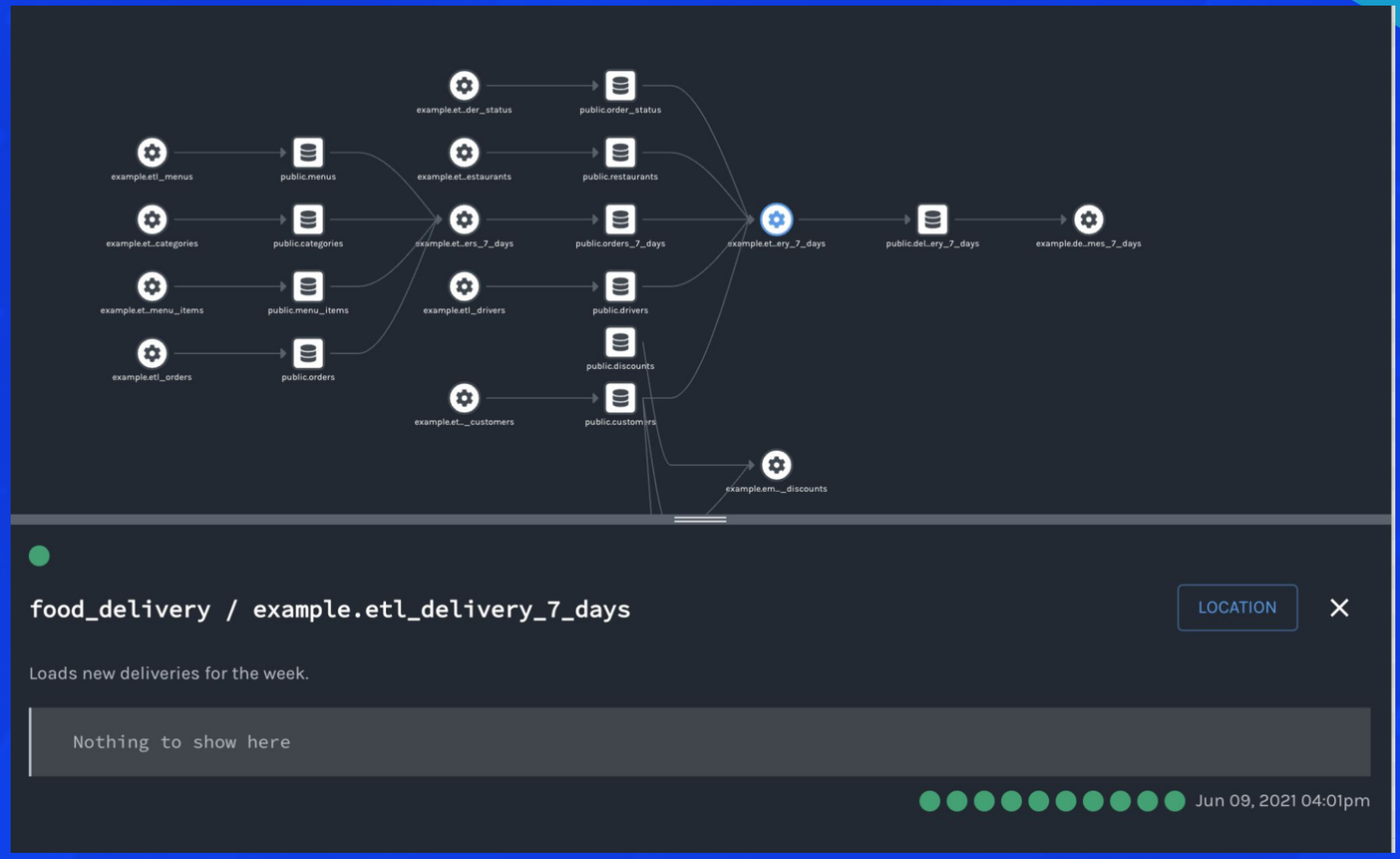
- Open AI
- Cohere
- Weviate
- pgvector
- Pinecone
- OpenSearch

Powering @AskAstro:
https://ask.astronomer.io/

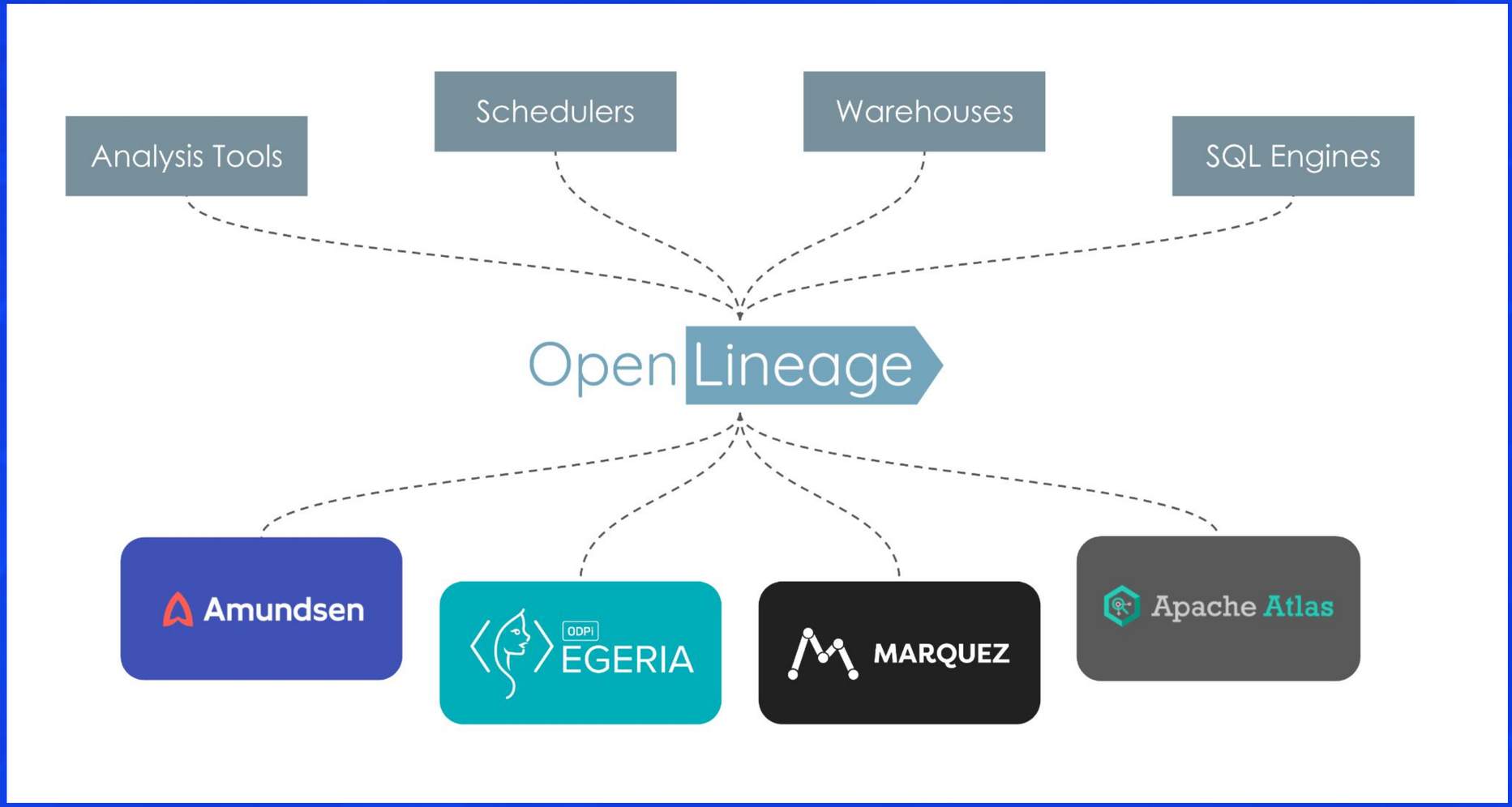
Airflow as a platform

Open Lineage

- Integrated in Airflow
- Column level lineage
- Better TaskFlow support in works
- Great adoption as open standard

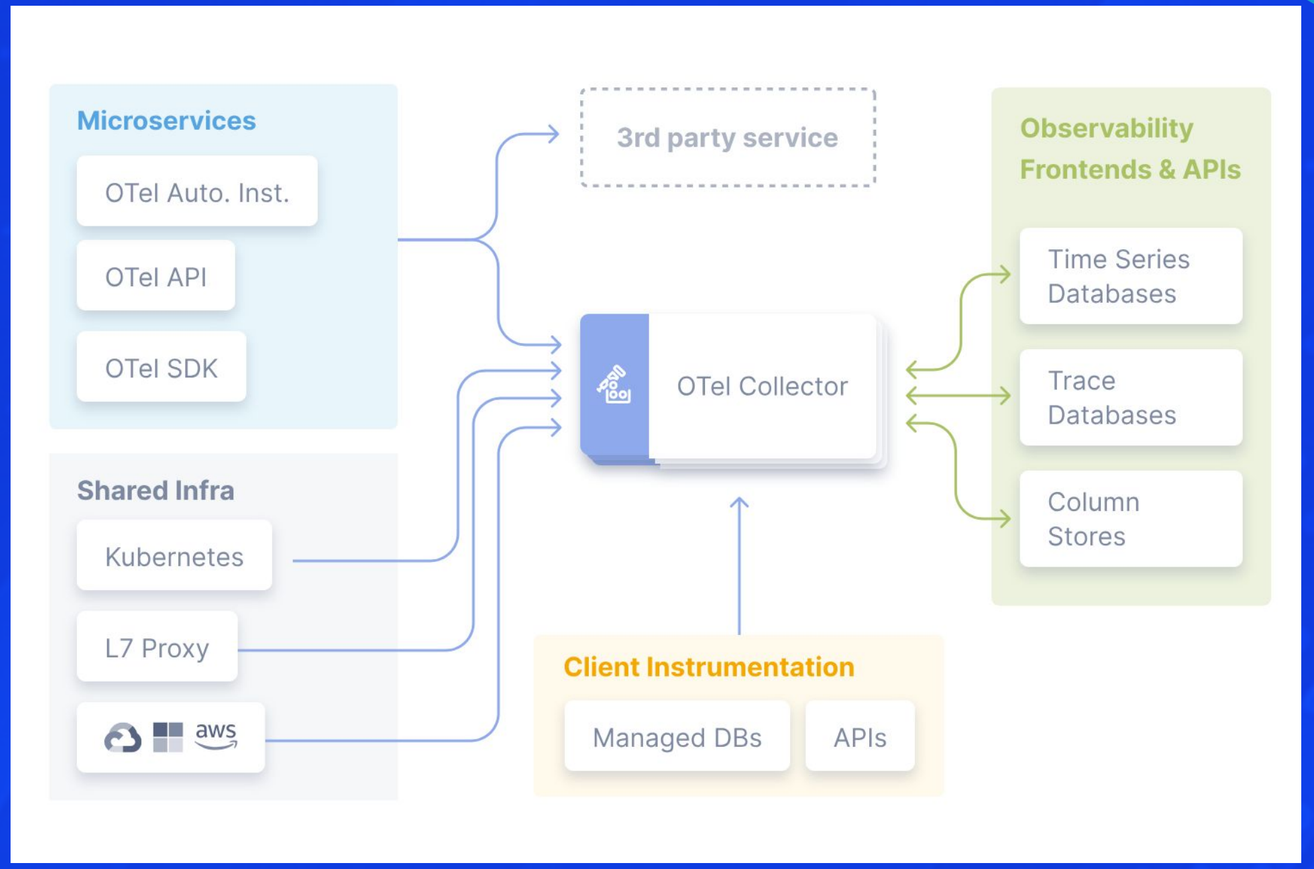


Open Lineage



Open Telemetry

- Integrated in Airflow
- Adopted by everyone
- Still early days
- Traces, Log support in the works

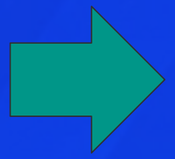


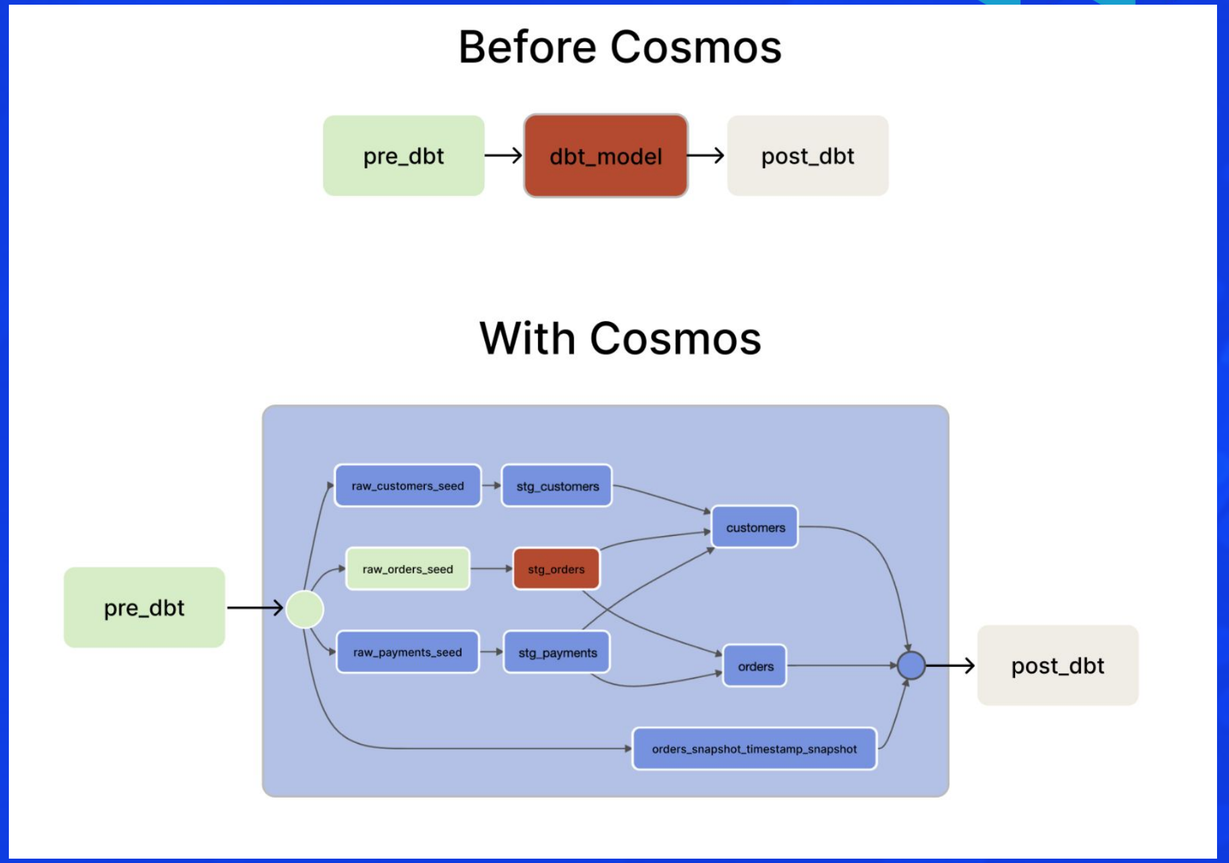
Astronomer's Cosmos

dbt Core

+

Apache Airflow





Fully fledged REST API

Engineering friendliness

- Workflow as a code front and center
- Tests
  - airflow task test
  - airflow dag test
  - unit test guidelines
  - system tests support
- Running Airlow locally
  - airflow standalone
  - docker compose
  - airflowctl - by Kaxil, Airflow PMC member

```
❯ airflowctl --help

Usage: airflowctl [OPTIONS] COMMAND [ARGS]...

Streamline getting started with Apache Airflow™ and managing multiple Airflow projects.

╭─ Options ──────────────────────────────────────────────────────────────────────────────────────╮
│ --install-completion          Install completion for the current shell.                        │
│ --show-completion             Show completion for the current shell, to copy it or customize the installation. │
│ --help                        Show this message and exit.                                      │
╰────────────────────────────────────────────────────────────────────────────────────────────────╯
╭─ Commands ─────────────────────────────────────────────────────────────────────────────────────╮
│ airflow   Forward commands to Airflow CLI.                                                     │
│ build     Build an Airflow project. This command sets up the project environment, installs Apache Airflow and its │
│           dependencies.                                                                        │
│ info      Display information about the current Airflow project.                               │
│ init      Initialize a new Airflow project.                                                    │
│ list      List all Airflow projects created using this CLI.                                    │
│ logs      Continuously display live logs of the background Airflow processes.                  │
│ start     Start Airflow.                                                                       │
│ stop      Stop a running background Airflow process and its entire process tree.               │
╰────────────────────────────────────────────────────────────────────────────────────────────────╯
```

Airflow IS Open Source

(and always will)

Community - part of Apache Software Foundation

- The largest project in ASF (for contributors count) >2700
- Licencing ASF, permissive licence (that will NEVER change)
- Well established, strong governance
- 61 committers, 32 PMC members
- Stakeholders/Managed services/Vendor neutrality
  - Astronomer, Amazon, Google, Microsoft, …
- Security / Release process / Maintenance certainty

Tools integrating with Airflow

- DAG visual editors
- Declarative DAG authoring
- IDE integration
- CLIs to manage Airflow
- Debugging aids
- UI extensions
- …

Tools integrating with Airflow

ADA - A microservice created to retrieve analytics metrics from an Airflow database instance.

as-scraper - An integration with Selenium to build & mantain web scrapers inside Airflow.

afctl - A CLI tool that includes everything required to create, manage and deploy airflow projects faster and smoother.

airflint - Enforce Best Practices for all your Airflow DAGs.

airflow-aws-executors - Run Airflow Tasks directly on AWS Batch, AWS Fargate, or AWS ECS; provisioning less infra is more.

airflow-code-editor - A tool for Apache Airflow that allows you to edit DAGs in browser.

airflow-diagrams - Auto-generated Diagrams from Airflow DAGs

airflow-maintenance-dags - Clairvoyant has a repo of Airflow DAGs that operator on Airflow itself, clearing out various bits of the backing metadata store.

AirflowK8sDebugger - A library for generate k8s pod yaml templates from an Airflow dag using the KubernetesPodOperator.

Airflow Ditto - An extensible framework to do transformations to an Airflow DAG and convert it into another DAG which is flow-isomorphic with the original DAG, to be able to run it on different environments (e.g. on different clouds, or even different container frameworks - Apache Spark on YARN vs Kubernetes). Comes with out-of-the-box support for EMR-to-HDInsight-DAG transforms.

Amundsen - Amundsen is a data discovery and metadata platform for improving the productivity of data analysts, data scientists and engineers when interacting with data. It can surface which Airflow task generates a given table.

Apache-Liminal-Incubating - Liminal provides a domain-specific-language (DSL) to build ML/AI workflows on top of Apache Airflow. Its goal is to operationalise the machine learning process, allowing data scientists to quickly transition from a successful experiment to an automated pipeline of model training, validation, deployment and inference in production.

Astro CLI - The Astro CLI is the easiest way to get a local Airflow server for prototyping and development.

Astro SDK - Astro SDK allows rapid and clean development of Extract, Load, Transform workflows using Python and SQL, powered by Apache Airflow and maintained by Astronomer.

Chartis - Python package to convert Common Workflow Language (CWL) into Airflow DAG.

CWL-Airflow - Python package to extend Apache-Airflow 1.10.11 functionality with CWL v1.2 support.

dag-factory - A library for dynamically generating Apache Airflow DAGs from YAML configuration files.

Dag Dependencies viewer - A tool which creates a view to visualize dependencies between the Airflow DAGs

data-dag - A library for building factories to dynamically generate DAGs from data (such as YAML files)

Databand - Observability platform built on top of Airflow.

DataHub - A metadata platform for the modern data stack. It can automatically collect lineage and other metadata from Airflow.

dbt (data build tool) - Data transformation tool, dbt jobs can be scheduled using Airflow.

Domino - Domino is an open source Graphical User Interface platform for creating data and Machine Learning workflows (DAGs) with no-code, visually intuitive drag-and-drop actions. It is also a standard for publishing and sharing your Python code so it can be automatically used by anyone, directly in the GUI.

Elyra - Elyra provides a visual editor that enables data scientists to create AI pipelines in a low-code/no-code fashion.

GeniumCloud - One-Stop-Shop Platform for rapid build, scheduling and control Airflow workflows via completely new UI. Out of the box comprehensive Airflow infrastructure monitoring, integration with alerting systems and service adoption from small to enterprise organizations. The easiest way to manage complex workflows.

gusty - Create a DAG using any number of YAML, Python, Jupyter Notebook, or R Markdown files that represent individual tasks in the DAG. gusty also configures dependencies, DAGs, and TaskGroups, features support for your local operators, and more. A fully containerized demo is available here.

Marquez - Marquez is an open source metadata service that maintains data provenance, shows how datasets are consumed and produced and centralizes dataset lifecycle management. Marquez can be used with Apache Airflow as an OpenLineage backend.

Meltano - Open source, self-hosted, CLI-first, debuggable, and extensible ELT tool that embraces Singer for extraction and loading, leverages dbt for transformation, and integrates with Airflow for orchestration.

Nexla - Build, transform, and manage data flows to and from databases, APIs, streams, SaaS services, events, and even emails. Use Nexla's Airflow Operator to trigger flows to start in other Operators when your Nexla flow finishes running.

Oozie to Airflow - A tool to easily convert between Apache Oozie workflows and Apache Airflow workflows.

OpenLineage - An open standard for the collection of data lineage, which can be used to trace the path of datasets as they traverse multiple systems including Apache Airflow.

Panda Patrol - Test and profile your data right within your Airflow DAGs. With dashboards and alerts already pre-built.

Pylint-Airflow - A Pylint plugin for static code analysis on Airflow code.

Redactics - A managed appliance (built on Airflow) installed next to your databases that powers a growing collection of data management workflows.

simple-dag-editor - Zero configuration Airflow tool that let you manage your DAG files.

Viewflow - An Airflow-based framework that allows data scientists to create data models without writing Airflow code.

whirl - Fast iterative local development and testing of Apache Airflow workflows.

ZenML - Run your machine learning specific pipelines on Airflow, easily integrating with your existing data science tools and workflows.

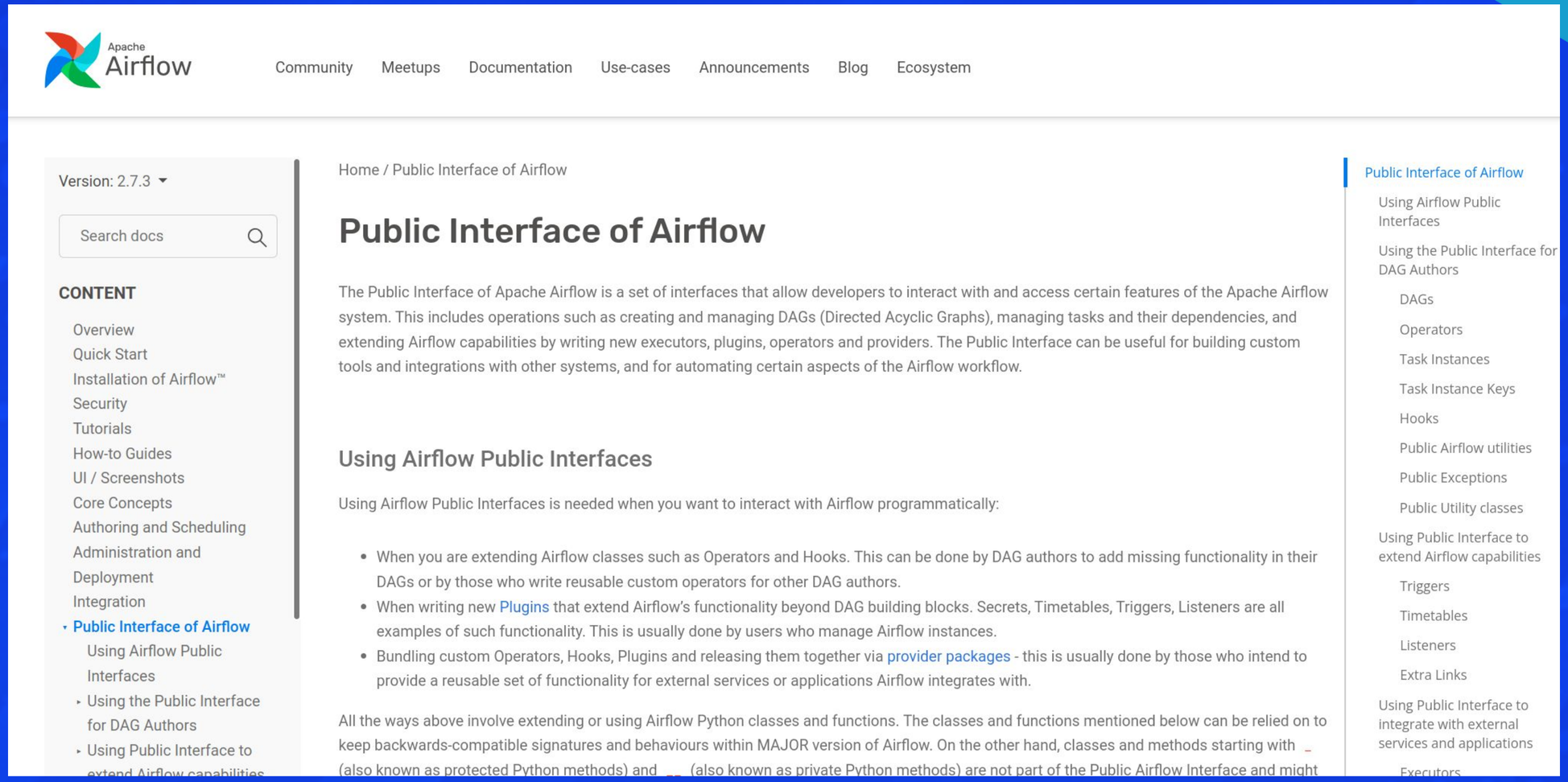
Airflow Vscode Extension This is a VSCode extension for Apache Airflow 2+. You can trigger your DAGs, pause/unpause DAGs, view execution logs, explore source code and do much more.

Airflow Provider Template - Template and commands for creating and testing airflow provider packages.

Airflow Template - Template and commands for creating minimal airflow environments for rapid testing and prototyping.

Solid Infrastructure

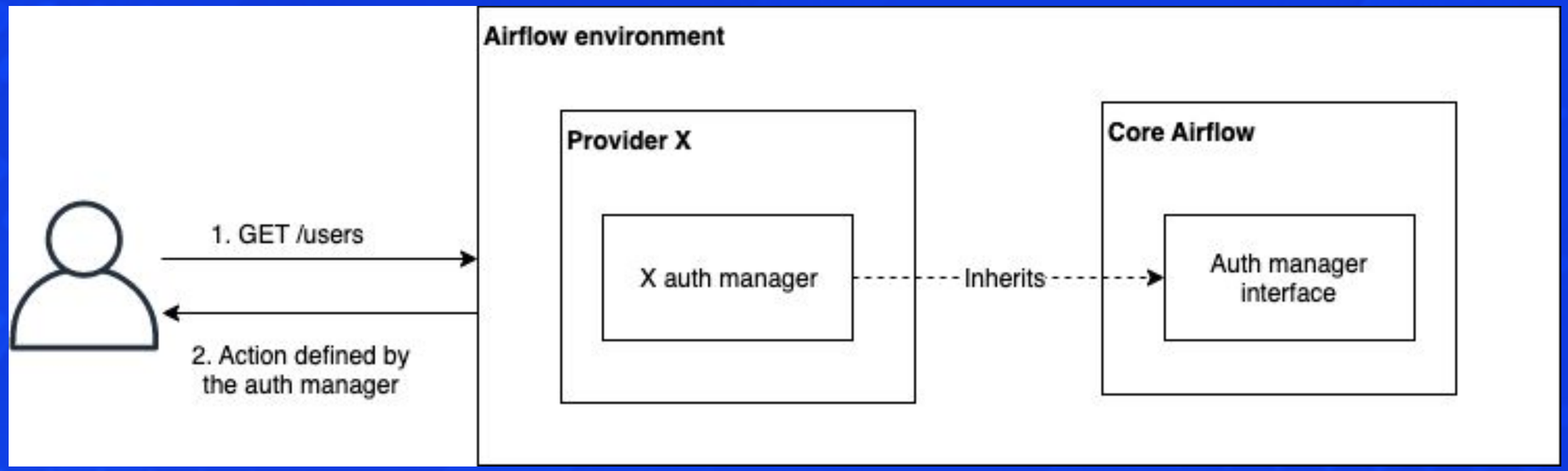
Public Interface of Airflow



Providers

- Can upgrade/downgrade separately
- Can provide:
  - Hooks/Operators/Sensors,Extra-links, Connection types
  - Secret Backends, Triggers, Log Handlers,
  - Executors, Notifications, Configuration, Decorators
  - Filesystems (2.8)
- Full lifecycle of providers defined
  - Approval by community (or not)
  - Support lifecycle for multiple Airflow versions
  - Suspension/Resuming/Removal
- 3rd-party providers and registries

Extensible user management



Security - coming soon for everyone

- Regulations are coming (CRA act just agreed in EU Trilogue)
- Airflow is part of the HackerOne OSS Bounty
- Highly functional Security Team ~50 reports handled
- 4 Airflow contributors: Sovereign Tech Fund funding for security
  - Security Model and Security Policy
  - SBOM generated
  - Securing release process (reproducible builds)
  - Component Isolation (Multi-tenancy in progress)

Summary

- Airflow is a modern, solid orchestrator with strong foundations
- New, slick ways to interact with the Modern Data Stack
- True Open Source
- Community is huge, strong and supportive
- More, exciting things are coming. Fast.

Q&A

https://github.com/potiuk

https://www.linkedin.com/in/jarekpotiuk