Benchmarking the Performance of

Dynamically Generated DAGs

Rahul Vats
Engineering Manager
@ Astronomer

Tati Al-Chueyr
Principal Software Engineer
@ Astronomer

Airflow Summit, Seattle, USA - 7 October 2025

ASTRONOMER

Agenda

1. Motivation
2. Benchmark Principles
3. Representative Workflows
4. Metrics
5. Implementation
6. Results
7. Next steps

1. Motivation

Dynamic DAGs in Airflow

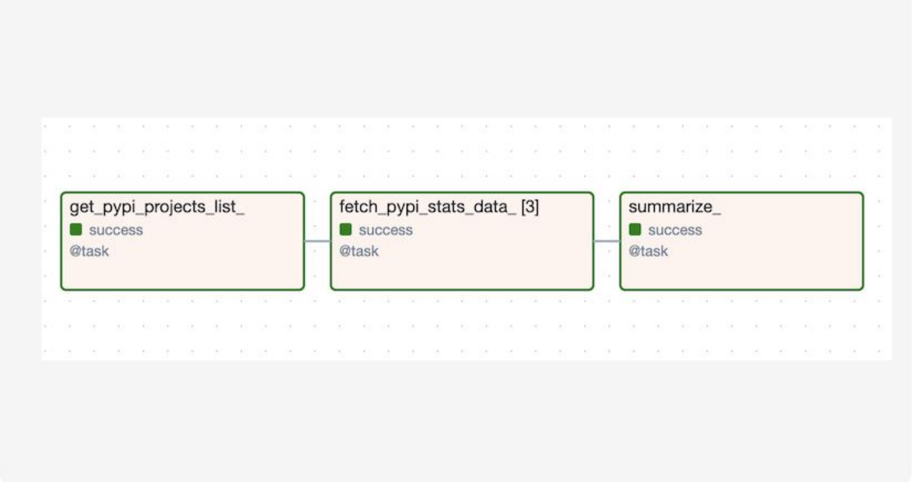
Because everything in Airflow is code, we can generate DAGs dynamically

```
from datetime import datetime
from airflow.sdk import DAG
from airflow.providers.standard.operators.bash import BashOperator

dag_configs = [
    {"dag_id": "dynamic_dag_1", "command": "echo DAG 1"},
    {"dag_id": "dynamic_dag_2", "command": "echo DAG 2"},
]

def create_dag(dag_id: str, command: str) -> DAG:
    with DAG(dag_id=dag_id, start_date=datetime(2025, 10, 7), schedule="@daily") as dag:
        BashOperator(task_id="run_cmd", bash_command=command)
    return dag

for cfg in dag_configs:
    globals()[cfg["dag_id"]] = create_dag(cfg["dag_id"], cfg["command"])
```

Dynamic DAGs in Airflow

The DAG Factory library, for example, builds Airflow DAGs out of YAML files

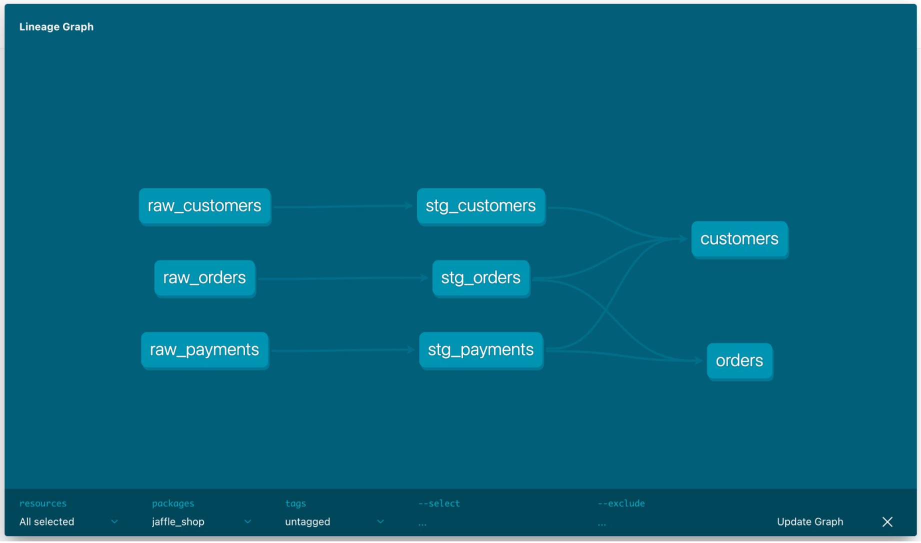
```yaml
example_pypi_stats_dagfactory:
  default_args:
    start_date: 2025-10-07
  tasks:
    - task_id: "get_pypi_projects_list"
      decorator: airflow.sdk.task
      python_callable: pypi_stats.get_pypi_projects_list
    - task_id: "fetch_pypi_stats_data"
      decorator: airflow.sdk.task
      python_callable: pypi_stats.fetch_pypi_stats_data
      expand:
        package_name: +get_pypi_projects_list
    - task_id: "summarize"
      decorator: airflow.sdk.task
      python_callable: pypi_stats.summarize
      values: +fetch_pypi_stats_data
```

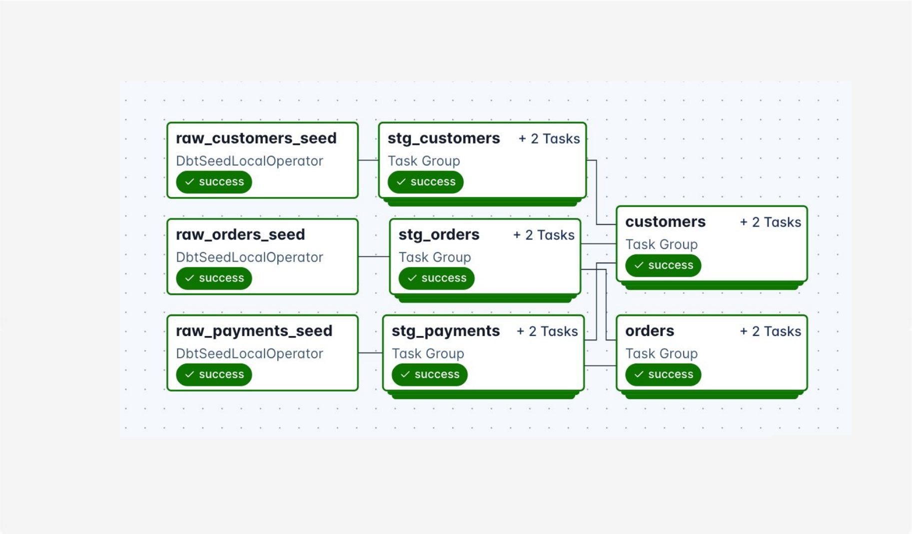



```
$ pip install dag-factory
```

Dynamic DAGs in Airflow

The Cosmos package dynamically translates dbt pipelines into Airflow DAGs

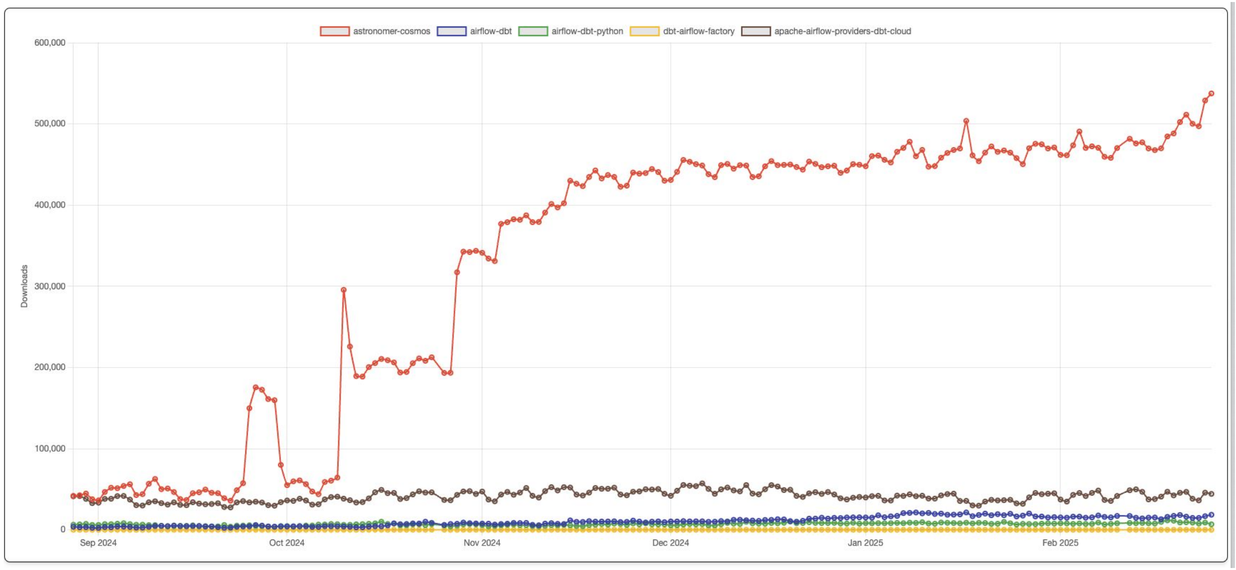





```
$ pip install astronomer-cosmos
```

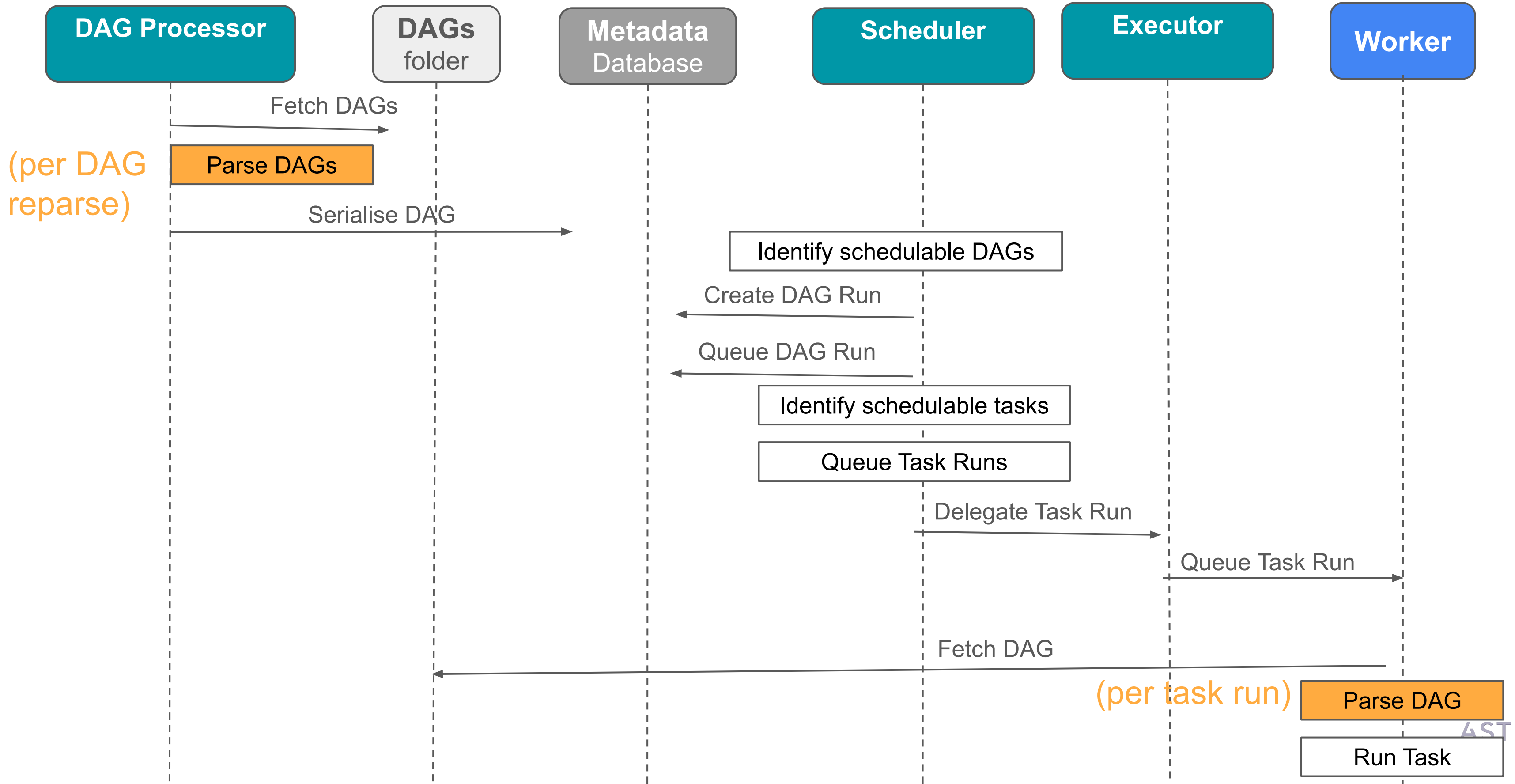
High adoption of Dynamic DAGs tools

- Over 20M+ monthly downloads in PyPI (just Cosmos & DAG Factory)
- Millions of dynamically build DAGs run every month in Astro



ASTRONOMER

How often Airflow reparse(Dynamic) DAGs



Dynamic DAGs needs

- DAGs are parsed both by the DAG Processor and **every** Worker node

- Compared to non-dynamic DAGs, dynamic DAGs will likely:
  - Consume more CPU
  - Consume more memory
  - Take longer to be parsed

- Users can be surprised by:
  - DAG Timeout issues
  - Long task queue times
  - Resource consumption not only on the DAG Processor, but also on worker nodes

Dynamic DAGs issues

Dags taking a long time to appear in the UI and staying in a queued state...

Via web form

Annie Friedman Internal • Aug 06 10:43

They said Dags were talking sometimes upwards of 30 minutes to reflect changes and tasks are being queued for upwards of 10min. They are a bit high on dag processor and worker cpu but not horribly so. From my cursory glance, I suspect this is exacerbated from their high rate of dag only deploys. Is there anything on the backend we can do to alleviate some of the pressure from the deploys? They aren't interested in using ephemeral deployments and their ci/cd runs a dag only deploy with every developer's commits. They also aren't interested in changing their ci/cd process at this time. This is effecting both dev and prod and is new since they have moved to hosted.

https://astronomer.zendesk.com/agent/tickets/80415

Dynamic DAGs issues

cosmos task taking too long

Via API

[redacted] • Jun 26 01:43

To: [redacted] Show more

Hi, I have an issue with a dag "ingestion_dbt_starfish_retail_orders", where the dag is scheduled to run every 10min but each run is taking more than 10min after I moved to cosmos. This dbt job is running several dbt models based on a dbt tag. After moving to cosmos, the length of each run has increased because the dbt compile takes place for each individual model instead of just once when running with the tag. Do you have any suggestions on reducing the time taken when using cosmos and running dbt models using dbt tags
Workspace: Data Team
Deployment: [redacted]-data-prod-astro-deployment

https://astronomer.zendesk.com/agent/tickets/78119

Stakeholders who cares about performance?

- **End-users:** can lose money due to misbehaving workflows
- **Airflow Developers**: want to improve - and not degrade - performance over versions
- **Sales**: so they communicate metrics to prospective customers with confidence
- **Product Marketing**: wants to compare against competitors

Lack of benchmark standardization

- **No clear** standard for running **performance benchmarks** on Apache Airflow

- Users and companies very often rely on **ad-hoc benchmarking**

- There is **lack of consistency** and **manual overhead**

- **Lack of history**, results are usually presented in one-off spreadsheets, docs and slides

2. Benchmark Principles

Clear objectives

- Define **what** you want to **measure** (throughput, latency, resource usage, etc)

- Tie benchmarks to **real-world use patterns** (peak loads, typical queries, business workflows)

- Motivation:
    - Find bottlenecks
    - Comparing against a baseline

Workload Design

- **Representativeness**: Use realistic workloads, not just synthetic stress tests.
- **Variability**: Include different query types, request patterns, and concurrency levels.
- **Scaling**: Test both typical and extreme workloads (steady state + stress testing).

Environment consistency

- Ensure test environments are **isolated and reproducible** (same hardware, cloud instance type, config).
- Minimize **external noise**: background jobs, network contention, autoscaling effects.
- Use **version control** for test configs, datasets, and scripts.

Benchmark experiment life cycle

Pre-process

🔅 setup isolated infrastructure
🔅 install necessary software

→

Run experiment

🔅 run desired command

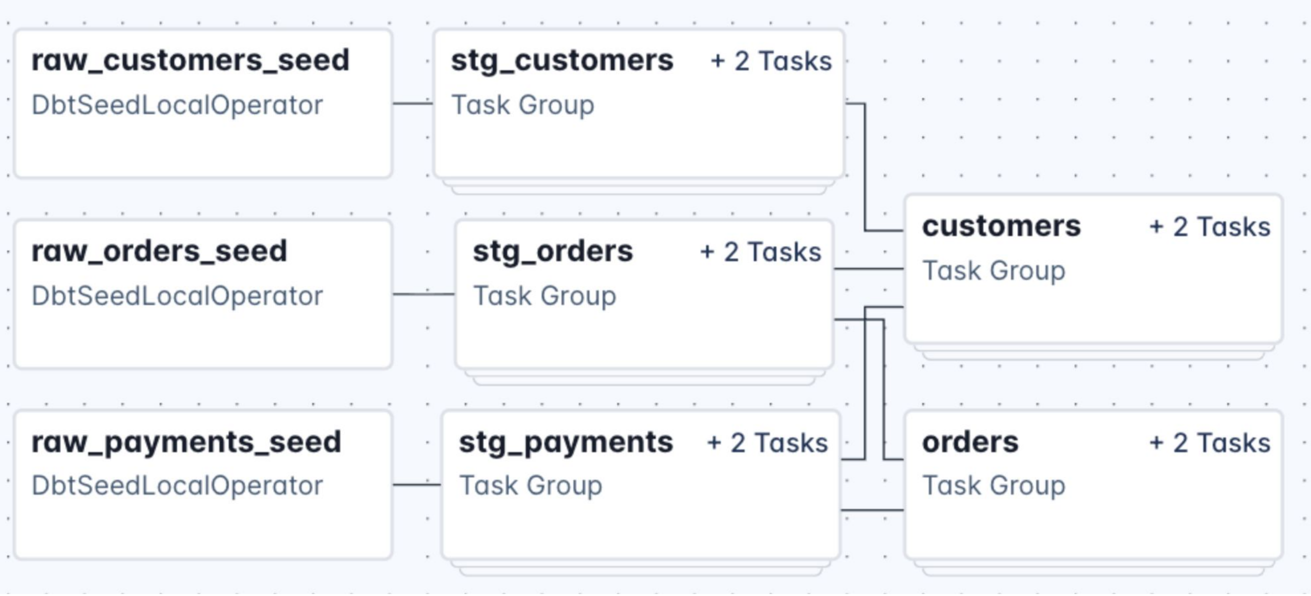
→

Post-process

🔅 collect metrics
🔅 store metrics
🔅 tear down infrastructure

3. Representative Workflows

Some workflows are too small



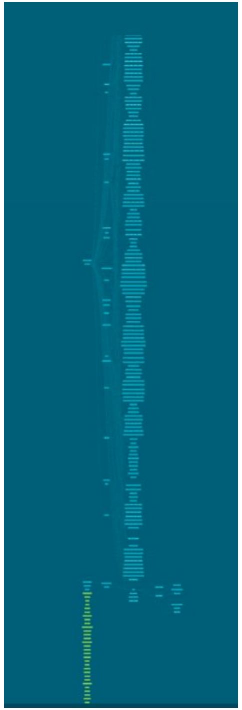
https://github.com/dbt-labs/jaffle-shop-classic

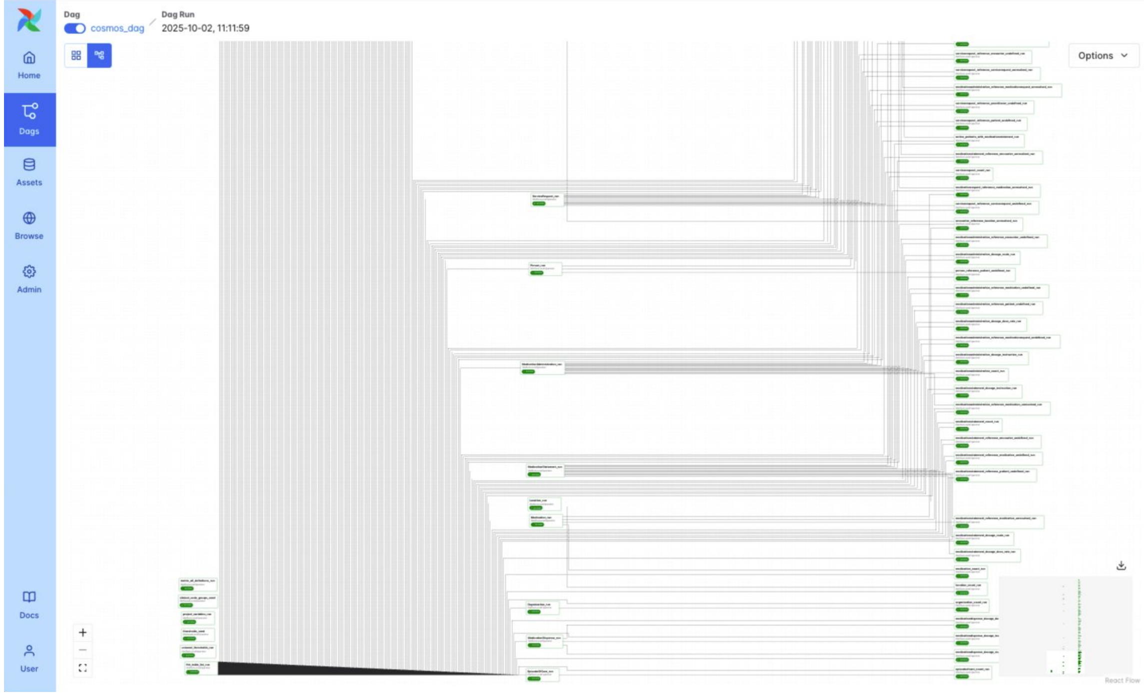
Synthetic workflows can not be representative

https://github.com/astronomer/astronomer-cosmos/pull/827

ASTRONOMER

Real (open source) dbt project





https://github.com/google/fhir-dbt-analytics

ASTRONOMER

4. Measurement & Metrics

Measurement & Metrics

- Core metrics:
    - DAG Run
    - Task Throughput
    - Error rate
    - Resource utilization
    - Memory
- Secondary metrics:
    - Startup time
    - Queue time
- Monitor system health
    - Logs, GC, caching, throttling

Statistical Significance

- Run **multiple iterations**, don't rely on single runs.
- **Be aware of variance** (especially in cloud environments)
- Use **statistical techniques** (confidence intervals, standard deviation) to confirm results are stable:
  - Standard deviation
  - Percentiles (p50, p95, p99)

5. Implementation

Experiment goal

Understand Cosmos 1.10 performance compared to dbt Core and dbt Cloud, when splitting the execution of a dbt pipeline in one or multiple commands, using a representative dbt project.

Experiment goal

cmd: 1

```
$ dbt build
```

cmd: 3

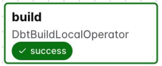
```
$ dbt seed
$ dbt run
$ dbt test
```

#cmd: 13

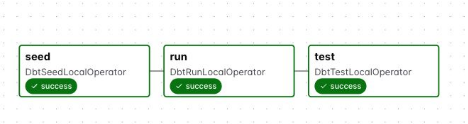
```
$ dbt seed --select raw_customers
$ dbt seed --select raw_orders
$ dbt seed --select raw_payments

$ dbt run --select stg_customers
$ dbt run --select stg_orders
$ dbt run --select stg_payments
$ dbt run --select customers
$ dbt run --select orders

$ dbt test --select stg_customers
$ dbt test --select stg_orders
$ dbt test --select stg_payments
$ dbt test --select customers
$ dbt test --select orders
```

Experiment goal

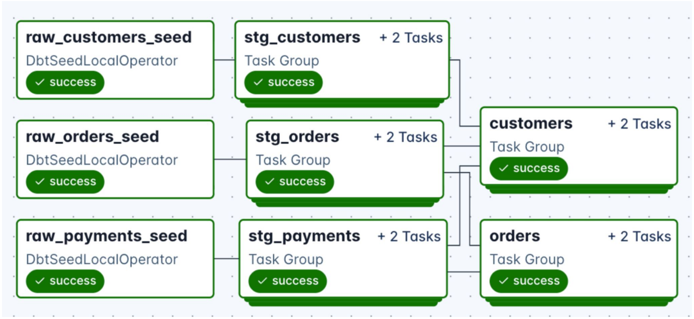
cmd: 1



cmd: 3



#cmd: 13



Metrics considered

- Pipeline execution time
- Memory consumption (average and standard deviation)
- CPU (average and standard deviation)

Benchmark experiment life cycle dbt Cloud

Pre-process

- create trial account
- setup project

→

Run experiment

- trigger dbt Cloud job

→

Post-process

- collect time
- store metrics

Benchmark experiment life cycle dbt Core

Pre-process

- 🔘 create K8s cluster
- 🔘 install Prometheus

→

Run experiment

- 🔘 run dbt command

→

Post-process

- 🔘 check command status
- 🔘 collect metrics
- 🔘 store metrics
- 🔘 tear down infrastructure

ASTRONOMER

Benchmark experiment life cycle Airflow

Pre-process

- create K8s cluster
- install Prometheus
- create Airflow Deployment

→

Run experiment

- trigger DAG to run

→

Post-process

- check command status
- collect metrics
- store metrics
- tear down infrastructure

Repository

We strongly believe that benchmarks should be public and reproducible by anyone in the community, and for this reason we've open-sourced this repository:
https://github.com/astronomer/cosmos-benchmark

6. Results

Results

Platform	Airflow Command	Airflow DAG	dbt Command	Granuarity	Durati on	Max CPU Utilization	Stddev CPU Utilization	Max Memory Usage
dbt Cloud	N/A	N/A	dbt build	single command	0:05:10	N/A	N/A	N/A

Results

Platform	Airflow Command	Airflow DAG	dbt Command	Granuarity	Duration	Max CPU Utilization	Stddev CPU Utilization	Max Memory Usage
dbt Cloud	N/A	N/A	dbt build	single command	0:05:10	N/A	N/A	N/A
dbt Core	N/A	N/A	dbt run	single command	0:05:05	0.39	0.06	306 MiB
dbt Core	N/A	N/A	dbt run	multi command (one per model)	0:31:50	0.39	0.06	306 MiB

https://github.com/astronomer/cosmos-benchmark/pull/4
https://github.com/astronomer/cosmos-benchmark/pull/5

Results

Platform	Airflow Command	Airflow DAG	dbt Command	Granuarity	Duration	Max CPU Utilization	Stddev CPU Utilization	Max Memory Usage
dbt Cloud	N/A	N/A	dbt build	single command	0:05:10	N/A	N/A	N/A
dbt Core	N/A	N/A	dbt run	single command	0:05:05	0.39	0.06	306 MiB
dbt Core	N/A	N/A	dbt run	multi command (one per model)	0:31:50	0.39	0.06	306 MiB
Airflow OSS	airflow dags test	DbtBuildLocalOperator	dbt build	single command	0:05:59	0.18	0.03	537 MiB
Airflow OSS	airflow dags test	DbtDag	dbt run	multi command (one per model)	0:27:26	0.19	0.25	1 GiB

https://github.com/astronomer/cosmos-benchmark/pull/6

Results

Platform	Airflow Command	Airflow DAG	dbt Command	Granuarity	Duration	Max CPU Utilization	Stddev CPU Utilization	Max Memory Usage
dbt Cloud	N/A	N/A	dbt build	single command	0:05:10	N/A	N/A	N/A
dbt Core	N/A	N/A	dbt run	single command	0:05:05	0.39	0.06	306 MiB
dbt Core	N/A	N/A	dbt run	multi command (one per model)	0:31:50	0.39	0.06	306 MiB
Airflow OSS	airflow dags test	DbtBuildLocalOperator	dbt build	single command	0:05:59	0.18	0.03	537 MiB
Airflow OSS	airflow dags test	DbtDag	dbt run	multi command (one per model)	0:27:26	0.19	0.25	1 GiB
Airflow OSS	airflow dags trigger	DbtBuildLocalOperator	dbt build	single command	0:05:50	1.3	0.09	1.6 GB
Airflow OSS	airflow dags trigger	DbtDag	dbt run	multi command (one per model)	0:15:13	3	0.15	2.5 GB

https://github.com/astronomer/cosmos-benchmark/pull/7

AIRFLOW SUMMIT

Boosting dbt Core Workflows Performance

With Airflow's Deferrable Capabilities

12:00 PT • Wednesday, October 8, 2025

Pankaj Koti
Software Engineer
@ Astronomer
Apache Airflow Committer

Tatiana Al-Chueyr
Staff Software Engineer
@ Astronomer
Cosmos Tech Lead

Pankaj Singh
Senior Software Engineer
@ Astronomer
Apache Airflow Committer

7. Next steps

Next steps

https://github.com/astronomer/cosmos-benchmark

- Have a **configuration-driven** approach to run the tests - and track those over time
- Leverage **Airflow 3 APIs** to trigger and monitor the status of Airflow jobs
- Store **results consistently** in a way we can track experiments over time - publically
- Automate tests via the **CI** based on changes
- Collect **more metrics**
- Extend benchmark to run in **Astro**

ASTRONOMER

8. Take away

Performance/benchmark testing isn't just about **running stress tools**—it's about designing **fair**, **reproducible**, and **meaningful** experiments that **guide decision-making**.

ASTRONOMER

We need to have a **Open Source standard** to run **benchmarks** on Airflow to allow the project to continue being a **leader** among **orchestration tools**

Learn more about how to run dbt with Apache Airflow and Cosmos

Thank you!

Questions?

Rahul Vats

Tati Al-Chueyr

#airflow-dbt Slack channel

ASTRONOMER