# RSP From Scratch

## Summary

Standard demo emphasizing Astro's ability to make Airflow more reliable, more scalable, and more productive!

## Creator

- Name: Tony Huinker
- GitHub: tonyhuinker
- Slack: @Tony Huinker

## Video Link

Link to Video

## Audience Tags

- Data Engineer
- DevOps/IT
- Data Engineering Manager

## Type Tags

- Standard

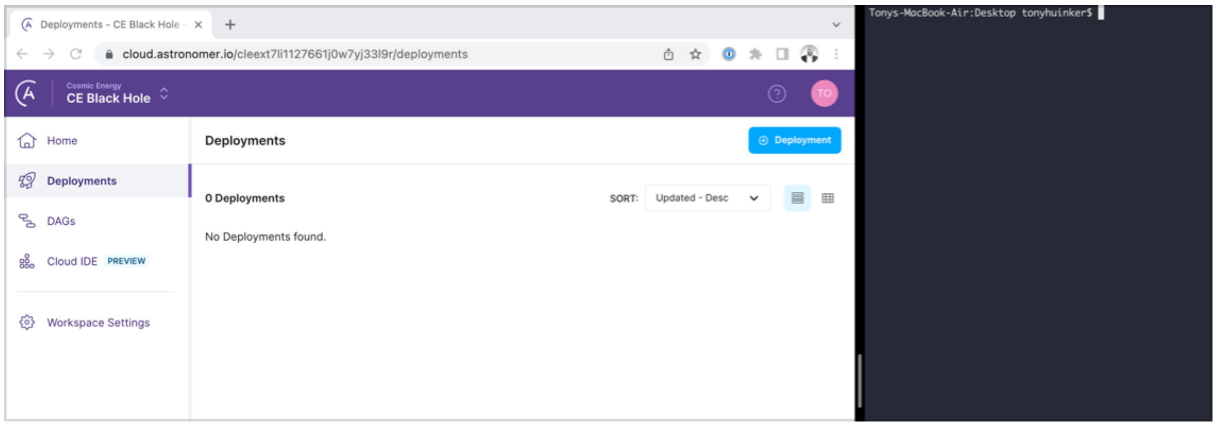
## Available Tags

- Data Engineer
- Security
- Data Scientist
- DevOps/IT
- Data Engineering Manager

## Demo Steps

1. **Demo Pre-Reqs**
   1. Your own, empty workspace in Cosmic Energy
   2. Astro CLI installed and functional (ensure astro dev start works, etc)

   3. Access to the Demo Companion Deck, opened up in a browser tab
2. **Show**
   1. Assuming the audience has seen nothing but slides so far, let's provide them some slide relief by having the Astro UI open and viewable as soon as you start sharing your screen.  Start from your own empty (no deployments) workspace, as well as having a terminal handy.  One way to present that is via split screen mode on MacOS, like below.



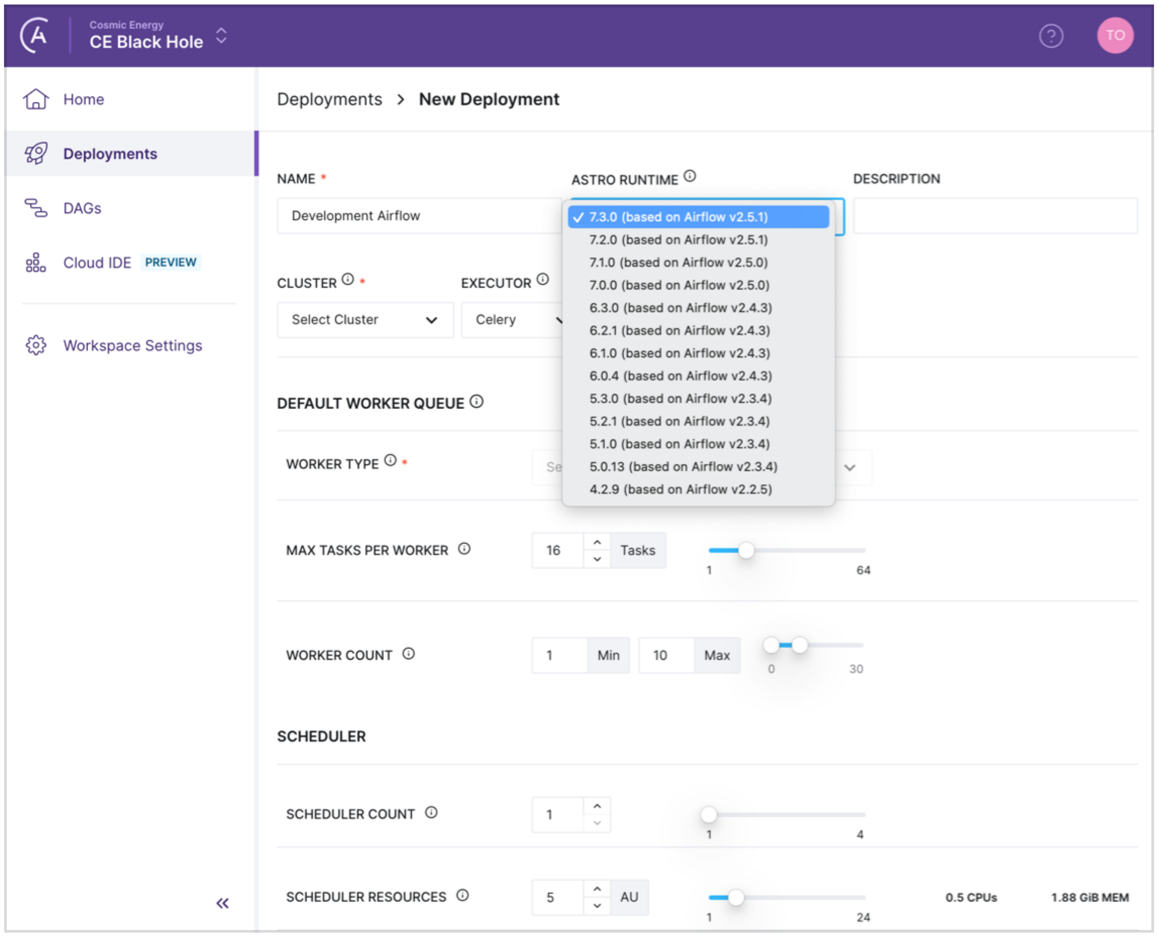
Deployments

0 Deployments

No Deployments found.

```
Tonys-MacBook-Air:Desktop tonyhuinker$
```

3. **Tell**
   1. "Today I'll be demoing Astro, a Fully Managed Service for running Airflow in the Cloud.   To help you get familiar for what it's like to begin using Astro, I'm going to walk through the process of building out an Airflow environment from Scratch."
   2. "I'm currently logged into a mock Organization named Cosmic Energy, but you could imagine your own company name here instead.  In addition, I've been invited to a "workspace", which I've named CE Black Hole (notice the space theme?  Cuz we're named Astronomer? Get it?)   You can think of a workspace more or less as a place to provision Airflow deployments, as well as a way to invite and control who can access, view, or deploy to them.
   3. To get started, let's create an Airflow that will be used for our development environment.
4. **Action**
   1. Click Create Airflow
5. **Show**
   1. The New Deployment Creation Screen

2. 
Cosmic Energy
CE Black Hole
TO
Home
Deployments
DAGs
Cloud IDE PREVIEW
Workspace Settings
«
Deployments > New Deployment
✓ 7.3.0 (based on Airflow v2.5.1)
7.2.0 (based on Airflow v2.5.1)
7.1.0 (based on Airflow v2.5.0)
7.0.0 (based on Airflow v2.5.0)
6.3.0 (based on Airflow v2.4.3)
6.2.1 (based on Airflow v2.4.3)
6.1.0 (based on Airflow v2.4.3)
6.0.4 (based on Airflow v2.4.3)
5.3.0 (based on Airflow v2.3.4)
5.2.1 (based on Airflow v2.3.4)
5.1.0 (based on Airflow v2.3.4)
5.0.13 (based on Airflow v2.3.4)
4.2.9 (based on Airflow v2.2.5)

NAME * Development Airflow
ASTRO RUNTIME 7.3.0 (based on Airflow v2.5.1)
DESCRIPTION
CLUSTER * Select Cluster
EXECUTOR Celery

DEFAULT WORKER QUEUE
WORKER TYPE *
MAX TASKS PER WORKER 16 Tasks (1–64)
WORKER COUNT 1 Min 10 Max (0–30)

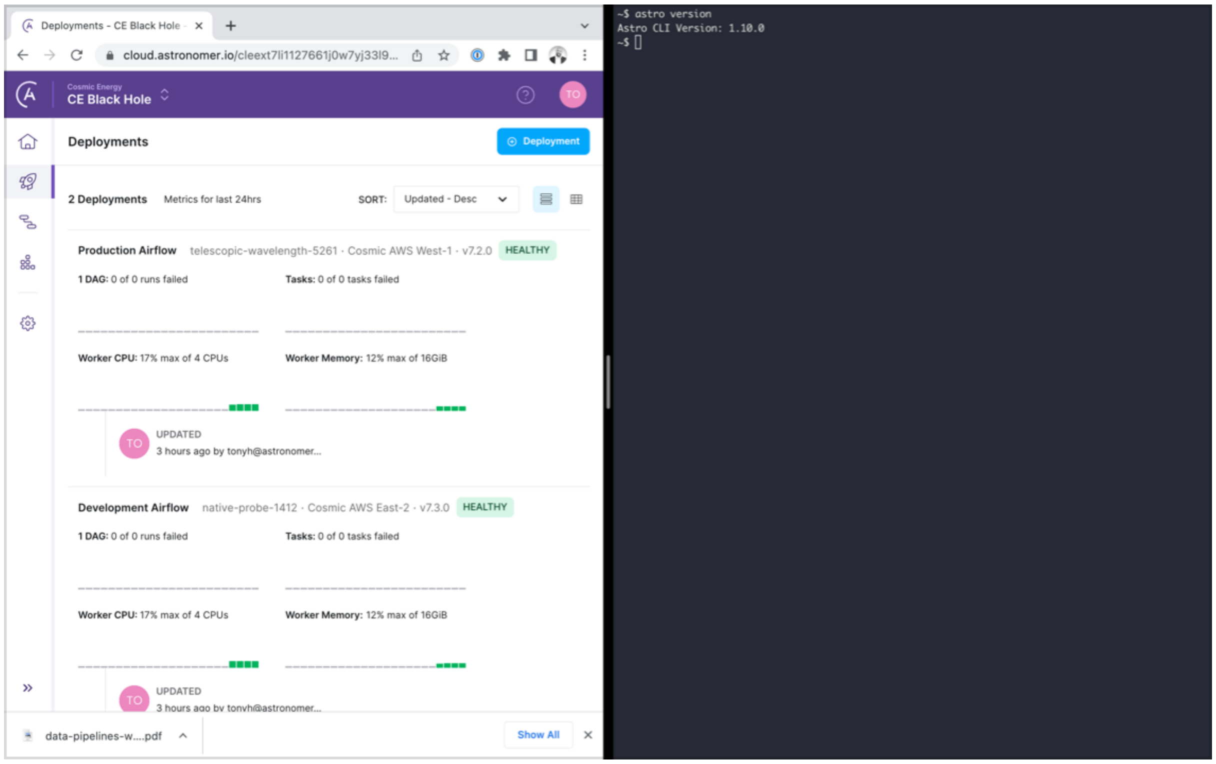
SCHEDULER
SCHEDULER COUNT 1 (1–4)
SCHEDULER RESOURCES 5 AU (1–24) 0.5 CPUs 1.88 GiB MEM

6. **Tell**
   1. The first decision we get to make is a target cluster.  Since y'all are predominantly [GCP|Azure|AWS], let's choose that.
   2. There are a few knobs and dials here but for now let's just go with the defaults.  Also, since it's dev, it makes sense to go with the latest and greatest Airflow Version, which will always be available to you on Astro the same day a new Airflow version gets released
   3. Then we just click Create, and that's it.  Under the hood, Astro is provisioning an Airflow environment with your specifications built around best practices.
7. **Action**
   1. Go back to deployments screen (will show your dev environment being created)
8. **Tell**
   1. Ok, we've created out Dev environment, now lets create an Environment for Production.
9. **Action**
   1. Click on the +Deployment Button Again
10. **Tell**
    1. Alright, for production, let's think through these settings a bit more.
    2. For the cluster, here we have a choice.  If we're looking to keep costs down, we can leverage the same infrastructure as Dev, and Astronomer will keep them isolated via

Kubernetes namespaces, so the chances of Dev impacted production is low. If you'd prefer zero chance , you can instead choose a completely different cluster, in this case, even a different region, if you'd prefer to keep your Dev and Prod environments completely isolated.

3. For Airflow version, lets go one version back from the latest and greatest.. no need to be bleeding edge on Production after all :-)
4. <as you work your way thorough the options, hover over the little info icons to let the product do some explanations of the options available>
5. Add an extra scheduler to the production Airflow
6. And now let's create!

11. **Action**
    1. Go back to the deployments screen, showing the Dev and Production Airflows, in various stages of being built.
12. **Tell**.
    1. Ok, now we have two airflow environments, one for Dev, One for Staging. Let's walk through how we're going to get Dags deployed to them. To do this, we'll leverage the Astro CLI
13. **Show**
    1. Either change what you're sharing to include a terminal, or make your terminal more prominent.
    2.



Production Airflow telescopic-wavelength-5261 · Cosmic AWS West-1 · v7.2.0 HEALTHY

**1 DAG:** 0 of 0 runs failed **Tasks:** 0 of 0 tasks failed

**Worker CPU:** 17% max of 4 CPUs **Worker Memory:** 12% max of 16GiB

UPDATED
3 hours ago by tonyh@astronomer...

Development Airflow native-probe-1412 · Cosmic AWS East-2 · v7.3.0 HEALTHY

**1 DAG:** 0 of 0 runs failed **Tasks:** 0 of 0 tasks failed

**Worker CPU:** 17% max of 4 CPUs **Worker Memory:** 12% max of 16GiB

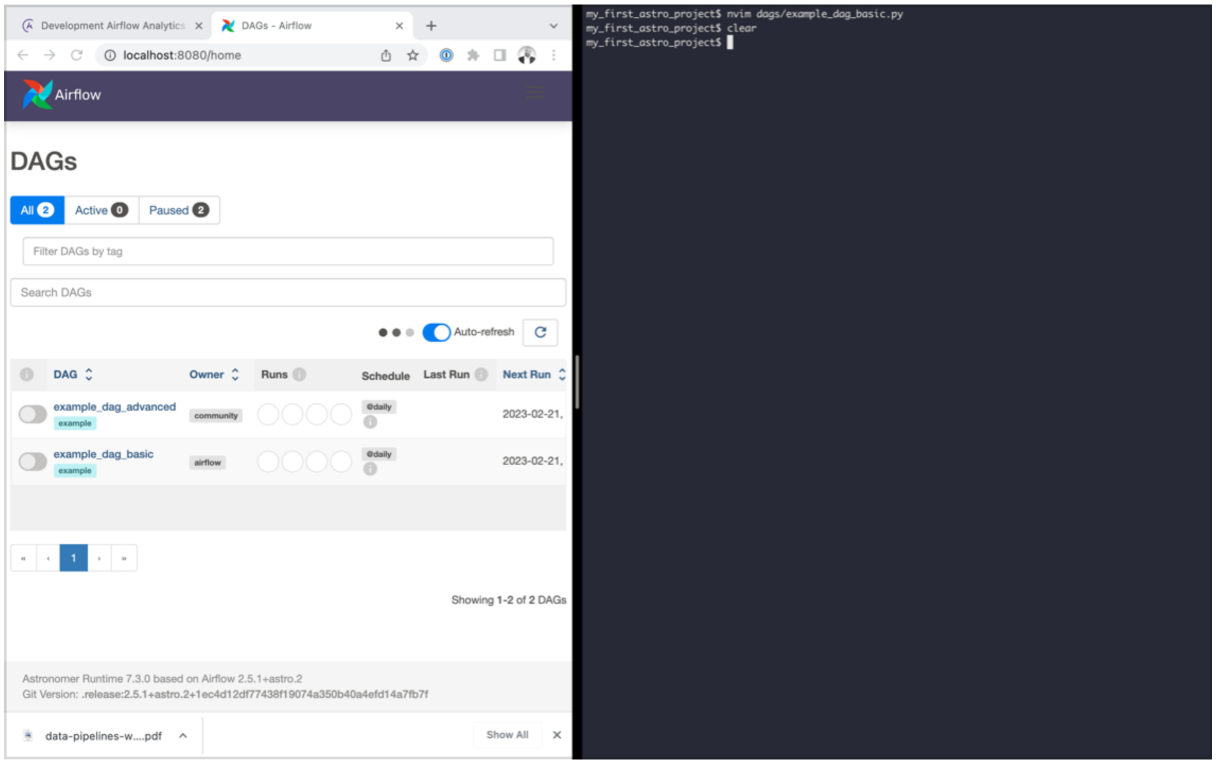
UPDATED
3 hours ago by tonyh@astronomer...

```
~$ astro version
Astro CLI Version: 1.10.0
~$
```

14. **Tell**
    1. We'll start by creating an empty directory for our Astro Project. We can then use the

Astro CLI to initialize our project

15. **Action**
    1. `astro version`
    2. `mkdir my_first_astro_project`
    3. `cd my_first_astro_project`
    4. `astro dev init`
    5. `ls`
16. **Tell**
    1. These files make up a simple Airflow project in Astro. In addition to the dags folder, you see that you have the ability to modify a `requirements.txt` file, an `include` directory, and even the Dockerfile itself!
    2. Let's say we wanted to begin development on some of the dags in our project directory. One thing you can do with the Astro CLI, is spin up a an airflow locally on your own machine.
17. Action
    1. `astro dev start`
18. Tell
    1. Here, you can see the Astro CLI building the components necessary to run Airflow on your local machine, including containers for the Webserver, Scheduler, Triggered, etc
    2. <wait to finish>
    3. Ok! Now we have a local airflow we can play with!
19. Show
    1. Ensure the Airflow UI is showing the main screen with the DAGs listed
    2.



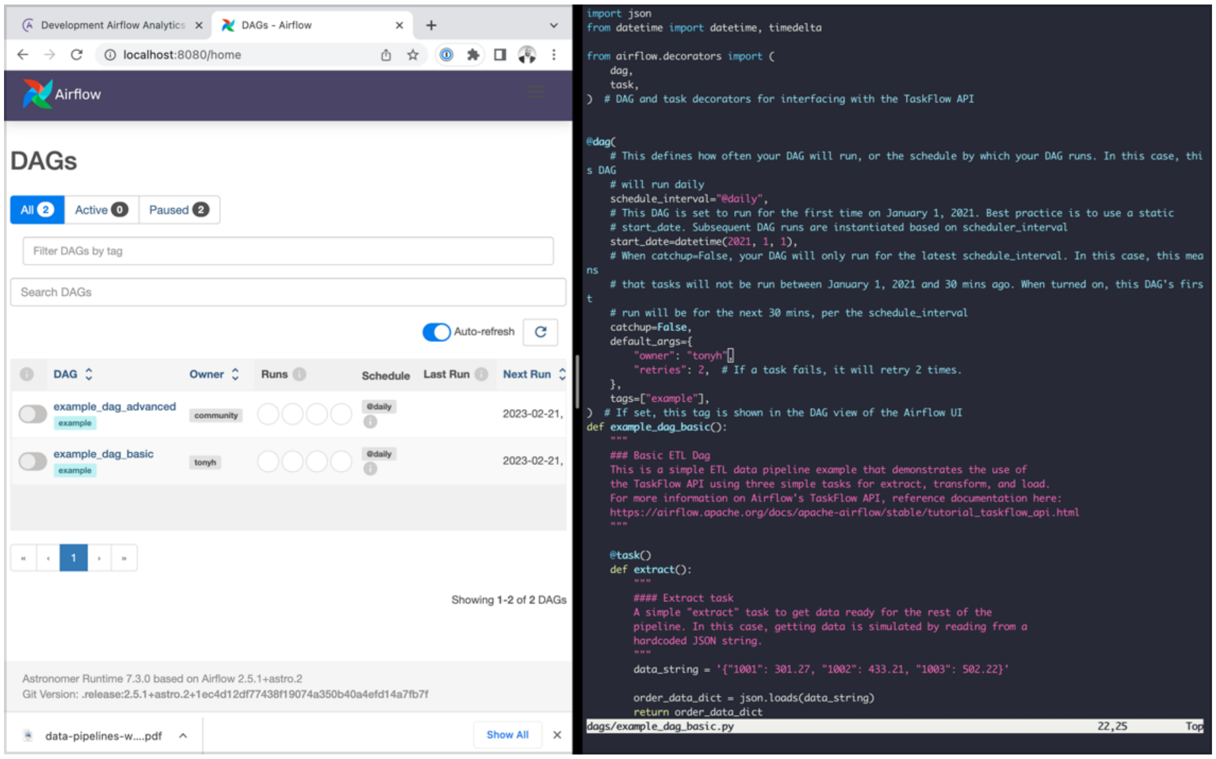
| DAG | Owner | Runs | Schedule | Last Run | Next Run |
|---|---|---|---|---|---|
| example_dag_advanced example | community | | @daily | | 2023-02-21, |
| example_dag_basic example | airflow | | @daily | | 2023-02-21, |

Showing 1-2 of 2 DAGs

Astronomer Runtime 7.3.0 based on Airflow 2.5.1+astro.2
Git Version: .release:2.5.1+astro.2+1ec4d12df77438f19074a350b40a4efd14a7fb7f

```
my_first_astro_project$ nvim dags/example_dag_basic.py
my_first_astro_project$ clear
my_first_astro_project$ 
```

20. Tell
    1. "Lets make a simple modification, so we can see the type of real time feedback you can get with local Airflow"
21. Action
    1. Using a text editor, modify `default_args` to include an owner argument, and sent it yourself
    2. After saving the updated dag, manually refresh the airflow ui
22. Show
    1. Show the updated owner!
    2.



DAGs

All 2 | Active 0 | Paused 2

Filter DAGs by tag

Search DAGs

Auto-refresh

| DAG | Owner | Runs | Schedule | Last Run | Next Run |
|---|---|---|---|---|---|
| example_dag_advanced example | community | | @daily | | 2023-02-21, |
| example_dag_basic example | tonyh | | @daily | | 2023-02-21, |

Showing 1-2 of 2 DAGs

Astronomer Runtime 7.3.0 based on Airflow 2.5.1+astro.2
Git Version: .release:2.5.1+astro.2+1ec4d12df77438f19074a350b40a4efd14a7fb7f

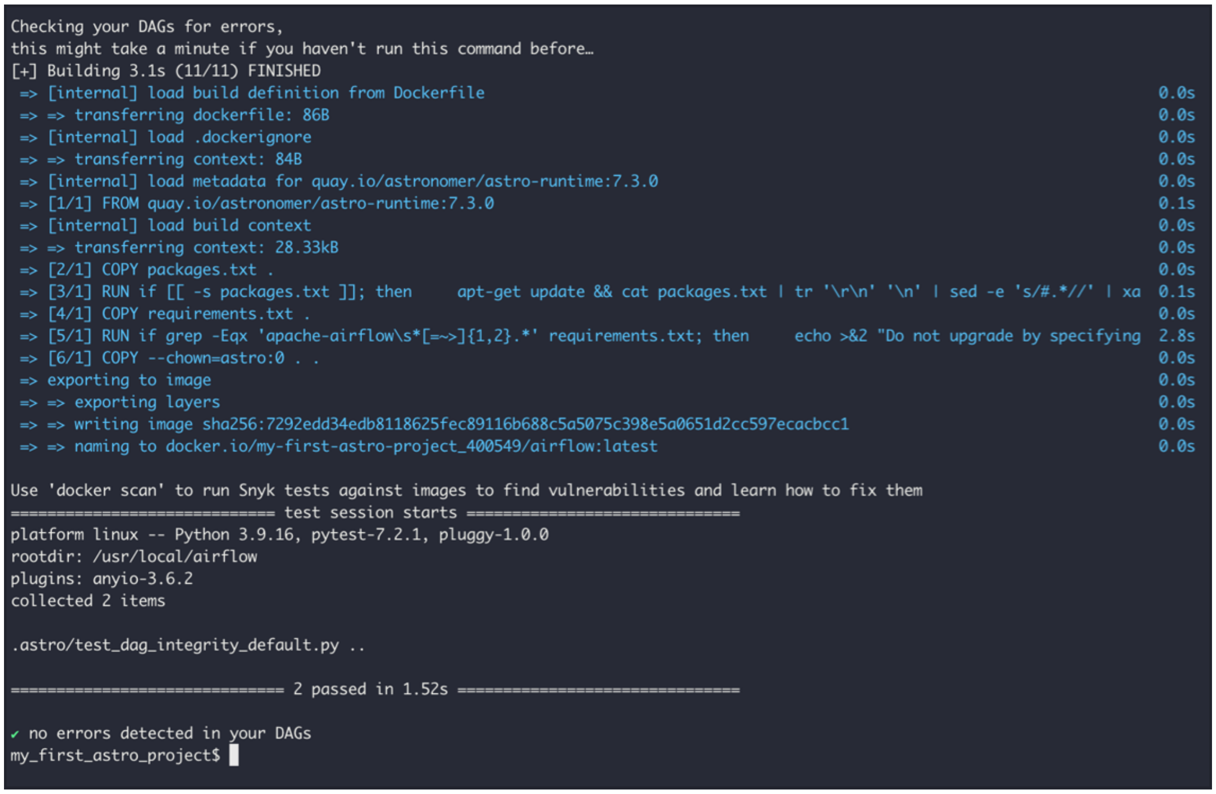
data-pipelines-w....pdf    Show All

```
import json
from datetime import datetime, timedelta

from airflow.decorators import (
    dag,
    task,
)  # DAG and task decorators for interfacing with the TaskFlow API


@dag(
    # This defines how often your DAG will run, or the schedule by which your DAG runs. In this case, thi
s DAG
    # will run daily
    schedule_interval="@daily",
    # This DAG is set to run for the first time on January 1, 2021. Best practice is to use a static
    # start_date. Subsequent DAG runs are instantiated based on scheduler_interval
    start_date=datetime(2021, 1, 1),
    # When catchup=False, your DAG will only run for the latest schedule_interval. In this case, this mea
ns
    # that tasks will not be run between January 1, 2021 and 30 mins ago. When turned on, this DAG's firs
t
    # run will be for the next 30 mins, per the schedule_interval
    catchup=False,
    default_args={
        "owner": "tonyh",
        "retries": 2,  # If a task fails, it will retry 2 times.
    },
    tags=["example"],
)  # If set, this tag is shown in the DAG view of the Airflow UI
def example_dag_basic():
    """
    ### Basic ETL Dag
    This is a simple ETL data pipeline example that demonstrates the use of
    the TaskFlow API using three simple tasks for extract, transform, and load.
    For more information on Airflow's TaskFlow API, reference documentation here:
    https://airflow.apache.org/docs/apache-airflow/stable/tutorial_taskflow_api.html
    """

    @task()
    def extract():
        """
        #### Extract task
        A simple "extract" task to get data ready for the rest of the
        pipeline. In this case, getting data is simulated by reading from a
        hardcoded JSON string.
        """
        data_string = '{"1001": 301.27, "1002": 433.21, "1003": 502.22}'

        order_data_dict = json.loads(data_string)
        return order_data_dict
dags/example_dag_basic.py                                          22,25         Top
```

23. Tell
    1. Ok, now that we've made a simple change, let's walk through how we can push this new dag with its fancy new owner to the Cloud.
    2. First, we need to ensure we've authenticated our Astro CLI with Astro Cloud by typing `astro login` (be sure to comment on fancy login method :-)
24. Action
    1. After running `astro login`
    2. Run `astro workspace list`, `astro workspace switch`, etc, to ensure you are in the right workspace
    3. Run `astro deployment list` to show the two Airflow deployments we made previously.
25. Tell

    1. Before pushing to the cloud, let's do one last check.  Another built in feature of the CLI is the ability to run test against your DAGs .
    2. Run `astro dev parse` to run syntax checks against your dags

26. Show
    1. Show the result of astro dev parse showing all clear
    2.



```
Checking your DAGs for errors,
this might take a minute if you haven't run this command before…
[+] Building 3.1s (11/11) FINISHED
 => [internal] load build definition from Dockerfile                                                                  0.0s
 => => transferring dockerfile: 86B                                                                                   0.0s
 => [internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 84B                                                                                      0.0s
 => [internal] load metadata for quay.io/astronomer/astro-runtime:7.3.0                                               0.0s
 => [1/1] FROM quay.io/astronomer/astro-runtime:7.3.0                                                                 0.1s
 => [internal] load build context                                                                                     0.0s
 => => transferring context: 28.33kB                                                                                  0.0s
 => [2/1] COPY packages.txt .                                                                                         0.0s
 => [3/1] RUN if [[ -s packages.txt ]]; then     apt-get update && cat packages.txt | tr '\r\n' '\n' | sed -e 's/#.*//' | xa  0.1s
 => [4/1] COPY requirements.txt .                                                                                     0.0s
 => [5/1] RUN if grep -Eqx 'apache-airflow\s*[=~>]{1,2}.*' requirements.txt; then     echo >&2 "Do not upgrade by specifying  2.8s
 => [6/1] COPY --chown=astro:0 . .                                                                                    0.0s
 => exporting to image                                                                                                0.0s
 => => exporting layers                                                                                               0.0s
 => => writing image sha256:7292edd34edb8118625fec89116b688c5a5075c398e5a0651d2cc597ecacbcc1                          0.0s
 => => naming to docker.io/my-first-astro-project_400549/airflow:latest                                               0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
============================= test session starts ==============================
platform linux -- Python 3.9.16, pytest-7.2.1, pluggy-1.0.0
rootdir: /usr/local/airflow
plugins: anyio-3.6.2
collected 2 items

.astro/test_dag_integrity_default.py ..

============================== 2 passed in 1.52s ===============================

✔ no errors detected in your DAGs
my_first_astro_project$ █
```

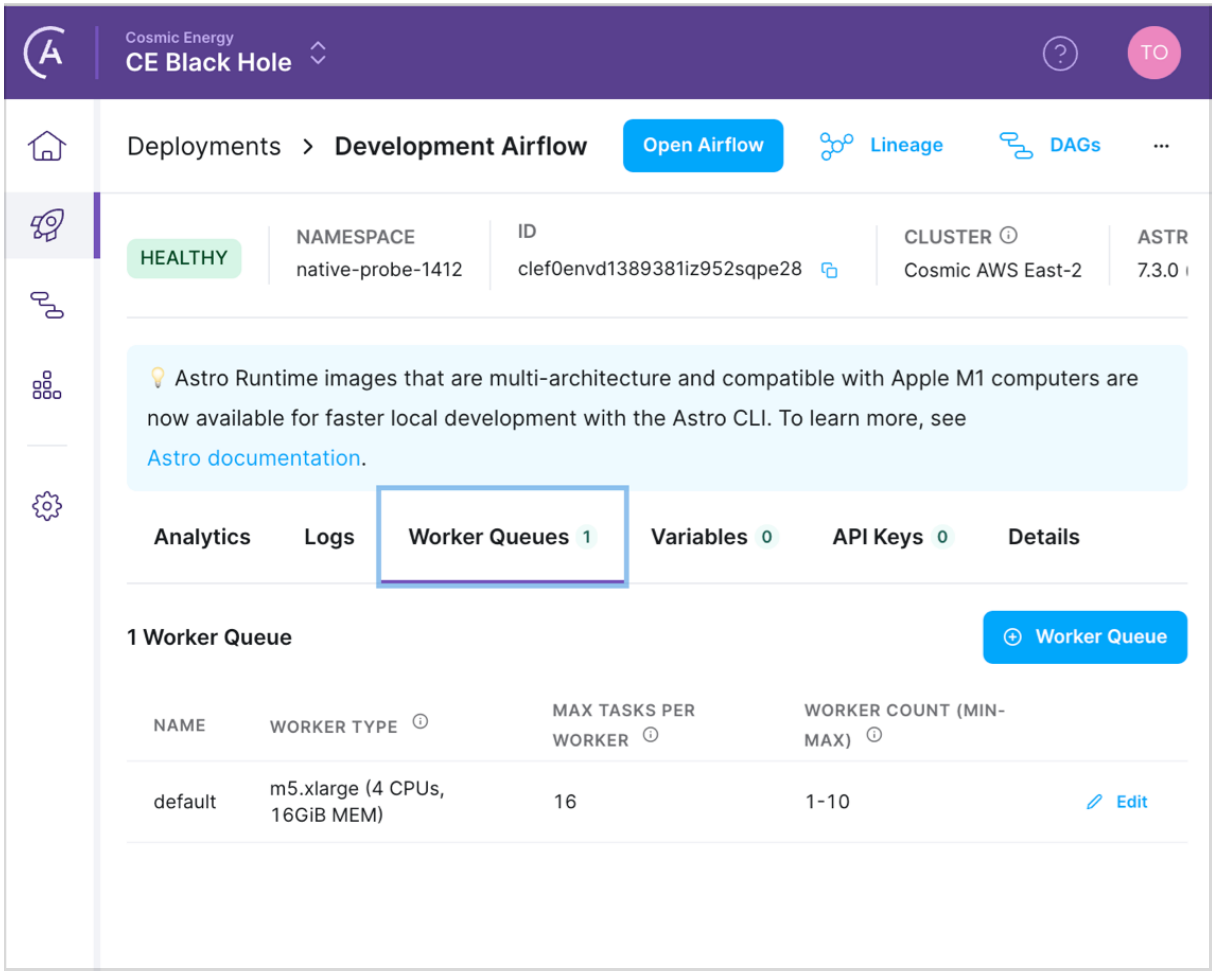
27. Action
    1. Run `astro deploy`, and choose the development environment.

28. Tell
    1. Now, as that's pushing to development, this is a good time to talk about in place upgrades.   Remember when we created the production environment to be one version back, on Astro 7.2? Well if we've finished our decided and have decided to upgrade, all we need to do is specify the newer version in the docker file, and deploy it production, and Astro will handle the upgrade for you.

29. Action
    1. Run `astro deploy` again, this time targeting the the production environment.

30. Tell
    1. Now.  If you happen to be looking at the syntax of the example dag we were looking at, you might have noticed it was a simple ETL tag, Extract, Transform, Load.  Let's say that that middle step, the Transform Task, requires significantly more resources than either the extract or load tasks.  In Astro, you can leverage a feature called Worker Queues, an designate that a specific task run on a node with specific hardware.  I'll

show you now how simple that is to setup.

31. Action
    1. Navigate to your development deployment page.
    2. Select the worker queues tab
32. Show
    1. 

Cosmic Energy
CE Black Hole

Deployments > Development Airflow

Open Airflow | Lineage | DAGs

| HEALTHY | NAMESPACE | ID | CLUSTER | ASTR |
|---|---|---|---|---|
| | native-probe-1412 | clef0envd1389381iz952sqpe28 | Cosmic AWS East-2 | 7.3.0 |

Astro Runtime images that are multi-architecture and compatible with Apple M1 computers are now available for faster local development with the Astro CLI. To learn more, see Astro documentation.

Analytics | Logs | Worker Queues 1 | Variables 0 | API Keys 0 | Details

1 Worker Queue

| NAME | WORKER TYPE | MAX TASKS PER WORKER | WORKER COUNT (MIN-MAX) | |
|---|---|---|---|---|
| default | m5.xlarge (4 CPUs, 16GiB MEM) | 16 | 1-10 | Edit |

33. Action
    1. Create a new worker queue, choose a bigger instance type, and name the queue "more-compute"
    2. Modify the transform task to include `@task( multiple_outputs=True, queue='more-compute')`, which is the same but has the added queue name for the task.
    3. Optional run `astro deployment update --deployment-name "Development Airflow" --dag-deploy enable`
    4. Run `astro deploy` or `astro deploy –dags`, and push your updated dag to astro
34. Action
    1. Navigate back to the deployments page, showing the two environments, each now on 7.3.
35. Tell

1. Ok, in just a small amount of time, we've gone from having no airflows, to having 2 Airflow environments, one for Prod, one for Dev, running in [AWS|GCP|Azure], isolated infrastructure, we've pushed dags to both Dev and Prod, and we modified a dag to tag advantage of Astro Worker Queue.