Improving the Airflow User Experience
About Astronomer

Astronomer is focused on helping organizations adopt Apache Airflow, the open-source standard for data pipeline orchestration.

100+
Enterprise customers around the world

4 of top 7
Airflow committers are Astronomer advisors or employees

Products
- Astronomer Enterprise
- Astronomer Cloud

Locations
- San Francisco
- London
- New York
- Cincinnati
- Hyderabad

Investors
- venrock
- SIERRA VENTURES
- BainCapital VENTURES
- Frontline
- Wireframe VENTURES
7 Stages of Airflow User Experience

Author → Build → Test → Deploy → Run → Monitor → Security / Governance
Current

LDAP authentication
Kerberos (w/ some operators)
Fernet key encryption
External secrets backend
CVE Mitigations

RBAC
- Astronomer has multi-tenant RBAC solution built in
astronomer-fab-securitymanager

A custom Flask-AppBuilder security manager for use with Apache Airflow inside the Astronomer Platform.
Current

- LDAP authentication
- Kerberos (with some operators)
- Fernet key encryption
- External secrets backend
- CVE Mitigations
- RBAC
  - Astronomer has multi-tenant RBAC solution built in

Future

- Data lineage
- Audit logs
- Integration with external identity providers (Auth0, Okta, Ping, SAML)
Current

Your Text Editor + Python environment

Astronomer CLI

Community Projects

- DagFactory (DevotedHealth)
- Airflow DAG Creation Manager Plugin
- Kedro
git pull

code .

```python
with DAG('covid_data_to_s3',
    start_date=datetime(2020, 3, 1),
    max_active_runs=1,
    schedule_interval='@daily',
    default_args=default_args,
    catchup=False # enable if you don't want historic
) as dag:

    t0 = DummyOperator(task_id='start')

    for endpoint in endpoints:
        generate_files = PythonOperator(
            task_id='generate_file_{0}'.format(endpoint),
            python_callable=upload_to_s3,
            op_kwargs={'endpoint': endpoint, 'date': date}
        )

        t0 >> generate_files
```
**dag-factory**

**dag-factory** is a library for dynamically generating **Apache Airflow** DAGs from YAML configuration files.

[https://github.com/ajbosco/dag-factory](https://github.com/ajbosco/dag-factory)
Define a DAG with YAML

eample_dag1:
  default_args:
    owner: 'example_owner'
    start_date: 2018-01-01  # or '2 days'
    end_date: 2018-01-05
    retries: 1
    retry_delay_sec: 300
    schedule_interval: '0 3 * * *'
    concurrency: 1
    max_active_runs: 1
    dagrun_timeout_sec: 60
    default_view: 'tree'  # or 'graph', 'duration', 'gantt', 'landing_times'
    orientation: 'LR'  # or 'TB', 'RL', 'BT'
    description: 'this is an example dag'
  on_success_callback_name: print_hello
  on_success_callback_file: /usr/local/airflow/dags/print_hello.py
  on_failure_callback_name: print_hello
  on_failure_callback_file: /usr/local/airflow/dags/print_hello.py
  tasks:
Define a DAG with YAML

```yaml
example_dag1:
  default_args:
    owner: 'example_owner'
    start_date: 2018-01-01  # or '2 days'
    end_date: 2018-01-05
    retries: 1
    retry_delay_sec: 300
    schedule_interval: '0 3 * * *'
    concurrency: 1
    max_active_runs: 1
    dagrun_timeout_sec: 60
    default_view: 'tree'  # or 'graph', 'duration', 'gantt', 'landing_times'
    orientation: 'LR'  # or 'TB', 'RL', 'BT'
    description: 'this is an example dag!
    on_success_callback_name: print_hello
    on_success_callback_file: /usr/local/airflow/dags/print_hello.py
    on_failure_callback_name: print_hello
    on_failure_callback_file: /usr/local/airflow/dags/print_hello.py
```

Parse the YAML

```python
from airflow import DAG
import dagfactory

dag_factory = dagfactory.DagFactory("/path/to/dags/config_file.yml")

dag_factory.clean_dags(globals())

dag_factory.generate_dags(globals())
```
….and you have a DAG!
Airflow DAG Creation Manager Plugin

Description

A plugin for Apache Airflow that create and manage your DAG with web UI.

https://github.com/lattebank/airflow-dag-creation-manager-plugin
Create and manage DAGS directly from the UI
Current

Your Text Editor + Python environment
Astronomer CLI
Community Projects
- DagFactory (DevotedHealth)
- Airflow DAG Creation Manager Plugin
- Kedro

Future

DAGs from Notebooks
Scheduling SQL query from UI
DAG Generator from standard templates
Current

Most users git-sync DAGs, add prod dependencies manually

Official Community Docker Image

Astronomer is Docker-centric

- Define dependencies (both (Python packages + system-level packages) directly in your code project
- Run the image locally with Docker
- Reduces devOps workload, since data engineers trial and error dependencies locally
- Can run the whole image through CVE testing
Current

No standardization around DAG unit testing

Adhoc testing for different data scenarios

Community Projects:

- Raybeam Status Plugin
- Great Expectations Pipeline Tutorial
Raybeam Status Plugin

Data confidence plugin for Airflow.

The Status Airflow plugin makes it easy to communicate confidence about your data system to managers, executives and other stakeholders in your organization. It improves trust in underlying data by increasing transparency.

https://github.com/Raybeam/rb_status_plugin
No reports have run yet!

Don't worry, here's some steps for creating a new report:

- Create a new report.
- Turn on the new report on the reports page.
- Run the new report manually or let it run naturally on the schedule you provided.
- Wait for the report to finish running.
- This status page will now be populated with a new report.
Schedule data quality tasks as reports
Keep stakeholders aware of data quality
Keep stakeholders aware of data quality

Hooks into existing Airflow functionality
Current

No standardization around DAG unit testing
Adhoc testing for different data scenarios

Community Projects:
- Raybeam Status Plugin
- Great Expectations Pipeline Tutorial

Future

Data awareness?
Standardized best practices for DAG unit testing
Additional automated testing of Hooks and Operators
Current

Most Airflow deployments are pets, not cattle — manually deployed

“Guess and check” for configurations

The Astronomer Way

- Use Kubernetes!
- Airflow now has an official Helm chart
- Astronomer platform makes it easy to CRUD Airflow deployments
Official Helm Chart for Apache Airflow

This chart will bootstrap an Airflow deployment on a Kubernetes cluster using the Helm package manager.

Prerequisites

- Kubernetes 1.12+ cluster
- Helm 2.11+ or Helm 3.0+
- PV provisioner support in the underlying infrastructure

```bash
# from the chart directory of the airflow repo
kubectl create namespace airflow
helm repo add stable https://kubernetes-charts.storage.googleapis.com
helm dep update
helm install airflow . --namespace airflow
```
uid
gid
nodeSelector
affinity
tolerations
labels
privateRegistry.enabled
privateRegistry.repository
networkPolicies.enabled
airflowHome
rbacEnabled
executor
allowPodLaunching
defaultAirflowRepository
defaultAirflowTag
images.airflow.repository
images.airflow.tag
images.airflow.pullPolicy
images.flower.repository
images.flower.tag
images.flower.pullPolicy
images.statsd.repository
images.statsd.tag
images.statsd.pullPolicy
images.redis.repository
images.redis.tag
images.pgbouncer.repository
images.pgbouncer.tag
images.pgbouncer.pullPolicy
images.pgbouncerExporter.repository
images.pgbouncerExporter.tag
images.pgbouncerExporter.pullPolicy
env
secret
data.metadataSecretName
data.resultBackendSecretName
data.metadataConnection
data.resultBackendConnection
fernetKey
fernetKeySecretName
workers.replicas
workers.keda.enabled
workers.keda.pollingInterval
workers.keda.cooldownPeriod
workers.keda.maxReplicaCount
workers.persistence.enabled
workers.persistence.size
workers.persistence.storageClassName
workers.resources.limits.cpu
workers.resources.limits.memory
workers.resources.requests.cpu
workers.resources.requests.memory
workers.terminateGracePeriodSeconds
workers.safeToEvict
scheduler.podDisruptionBudget.enabled
scheduler.podDisruptionBudget.config.maxUnavailable
scheduler.resources.limits.cpu
scheduler.resources.limits.memory
scheduler.resources.requests.cpu
scheduler.resources.requests.memory
scheduler.airflowLocalSettings
scheduler.safeToEvict
webserver.livenessProbe.initialDelaySeconds
webserver.livenessProbe.timeoutSeconds
webserver.livenessProbe.failureThreshold
webserver.readinessProbe.initialDelaySeconds
webserver.readinessProbe.timeoutSeconds
webserver.readinessProbe.failureThreshold
webserver.replicas
webserver.resources.limits.cpu
webserver.resources.limits.memory
webserver.resources.requests.cpu
webserver.resources.requests.memory
webserver.defaultUser
dags.persistence.*
dags.gitSync.*
helm install airflow-ry . --namespace airflow-ry

NAME: airflow-ry
LAST DEPLOYED: Wed Jul 8 20:10:29 2020
NAMESPACE: airflow-ry
STATUS: deployed
REVISION: 1

You can now access your dashboard(s) by executing the following command(s) and visiting the corresponding port at localhost in your browser:

Airflow dashboard: kubectl port-forward svc/airflow-ry-webserver 8080:8080 --namespace airflow

kubectl get pods --namespace airflow-ry

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>airflow-ry-postgresql-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>6m45s</td>
</tr>
<tr>
<td>airflow-ry-scheduler-78757cd557-t8zdn</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>6m45s</td>
</tr>
<tr>
<td>airflow-ry-statsd-5c889cc6b6-jxhw</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>6m45s</td>
</tr>
<tr>
<td>airflow-ry-webserver-59d79b9955-7sgp5</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>6m45s</td>
</tr>
</tbody>
</table>
astro deployment create test-deployment --executor celery

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEPLOYMENT NAME</th>
<th>ASTRO</th>
<th>DEPLOYMENT ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-deployment</td>
<td>theoretical-element-5806</td>
<td>0.15.2</td>
<td>ckce1ssco4uf90j16a5adke17</td>
</tr>
</tbody>
</table>

Successfully created deployment with Celery executor. Deployment can be accessed at the following URLs:

Airflow Dashboard: https://deployments.astronomer.io/theoretical-element-5806/airflow
Flower Dashboard: https://deployments.astronomer.io/theoretical-element-5806/flower

astro deployment delete ckce1ssco4uf90j16a5adke17

Successfully deleted deployment
## Execution Environment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Current Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executor</td>
<td>Kubernetes</td>
<td></td>
</tr>
<tr>
<td>Worker Count</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Worker Resources</td>
<td>20 AU</td>
<td>2 CPU, 7.5 GB memory, $200/mo</td>
</tr>
<tr>
<td>Worker Termination Grace Period</td>
<td>10 minutes</td>
<td></td>
</tr>
<tr>
<td>Extra Capacity</td>
<td>0 AU</td>
<td>0 CPU, 0 memory</td>
</tr>
</tbody>
</table>

Only necessary to run the KubernetesPodOperator (minimum 10AU).

## Core Resources

<table>
<thead>
<tr>
<th>Component</th>
<th>Current Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Webserver</td>
<td>9 AU</td>
<td>0.9 CPU, 3.38 GB memory, $90/mo</td>
</tr>
<tr>
<td>Scheduler</td>
<td>9 AU</td>
<td>0.9 CPU, 3.38 GB memory, $90/mo</td>
</tr>
</tbody>
</table>
### airflow.cfg name | Environment Variable | Default Value
---|---|---
parallelism | AIRFLOW__CORE__PARALLELISM | 32
dag_concurrency | AIRFLOW__CORE__DAG_CONCURRENCE | 16
worker_concurrency | AIRFLOW__CELERY__WORKER_CONCURRENCE | 16
max_threads | AIRFLOW__SCHEDULER__MAX_THREADS | 2

**parallelism** is the max number of task instances that can run concurrently on airflow. This means that across all running DAGs, no more than 32 tasks will run at one time.

**dag_concurrency** is the number of task instances allowed to run concurrently within a specific dag. In other words, you could have 2 DAGs running 16 tasks each in parallel, but a single DAG with 50 tasks would also only run 16 tasks - not 32.

These are the main two settings that can be tweaked to fix the common "Why are more tasks not running even after I add workers?"

**worker_concurrency** is related, but it determines how many tasks a single worker can process. So, if you have 4 workers running at a worker concurrency of 16, you could process up to 64 tasks at once. Configured with the defaults above, however, only 32 would actually run in parallel. (and only 16 if all tasks are in the same DAG)

**Pro tip:** If you increase worker_concurrency, make sure your worker has enough resources to handle the load. You may need to increase CPU and/or memory on your workers. Note: This setting only impacts the CeleryExecutor.
Current

Most Airflow deployments are pets, not cattle — manually deployed

“Guess and check” for configurations

The Astronomer Way

- Use Kubernetes!
- Airflow now has an official Helm chart
- Astronomer platform makes it easy to CRUD Airflow deployments

Future

Infrastructure and configuration recommendations to optimize performance and identify bottlenecks
**Current**

Most Airflow deployments running on virtual machines

Running in K8s enhances stability, observability, and ability to scale
## System Admin

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Users</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Redacted</strong></td>
<td><strong>205</strong></td>
<td><strong>1512</strong></td>
</tr>
</tbody>
</table>

- **Tag:** deploy-28
- **Updated:** 06/09/20
- **Created:** 09/25/19

- **Tag:** ci-fa3b117570ffadca4f07963a6ac96b089001d3c
- **Updated:** 06/09/20
- **Created:** 10/15/19

- **Tag:** ci-0.1.949
- **Updated:** 07/08/20
- **Created:** 10/16/19

- **Tag:** deploy-21
- **Updated:** 07/06/20
- **Created:** 10/21/19

- **Tag:** ci-6b00ab4
- **Updated:** 06/19/20
- **Created:** 10/21/19

- **Tag:** ci-6b00ab4
- **Updated:** 06/22/20
- **Created:** 10/21/19

← on a single k8s cluster!
← All this for one celery worker. But it's ready to scale.
The challenge w/ KubernetesExecutor

Airflow Scheduler → Kubernetes Scheduler → Airflow KubernetesExecutor Pods

Long-running tasks
The challenge w/ KubernetesExecutor

Airflow Scheduler → Kubernetes Scheduler → Airflow KubernetesExecutor Pods

Medium-running tasks
The challenge w/ KubernetesExecutor

Airflow Scheduler → Kubernetes Scheduler → Airflow KubernetesExecutor Pods

Short-running tasks
Celery with KEDA

CEIL((20 RUNNING + 20 QUEUED)/16) = 4 workers
**Current**

Most Airflow deployments running on virtual machines

Running in K8s enhances stability, observability, and ability to scale

**Future**

Highly Available Scheduler

“Fastfollow” task scheduling
HA Scheduler

Airflow Scheduler

Airflow Scheduler

...
Fast follow

**SchedulerJob**
- Run Scheduler Loop
- Get Available Dags
- Get Available Tasks
- Get Task Instances
- Task Instances to Scheduled
- Queue Task Instances

**Executor**
- Execution Magic

**TaskInstance**
- Run Task
- Set Downstream Task Instances to Scheduled

*Fast Follow*
Current

Airflow built-in dashboards based on task metadata

Airflow native statsd exporter offers deeper metrics
Counters
<job_name>_start
<job_name>_end
operator_failures_<operator_name>
operator_successes_<operator_name>
ti_failures
ti_successes
zombies_killed
scheduler_heartbeat
dag_processing.processes
dag_processing.failed_tasks
scheduler.tasks.killed_externally

gauges
dagbag_size
dag_processing.import_errors
dag_processing.total_parse_time
dag_processing.last_runtime.<dag_file>
dag_processing.last_run.seconds_ago.<dag_file>
dag_processing.processor_timeouts
dag_processing.failed_tasks
executor.open_slots
executor.queued_tasks
executor.running_tasks
pool.open_slots.<pool_name>
pool.used_slots.<pool_name>
pool.starving_tasks.<pool_name>

Timers
dagrun.dependency-check.<dag_id>
dag.<dag_id>.<task_id>.duration
dag_processing.last_duration.<dag_file>
dagrun.duration.success.<dag_id>
dagrun.duration.failed.<dag_id>
dagrun.schedule_delay.<dag_id>
<table>
<thead>
<tr>
<th>Service</th>
<th>Platform</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airflow Database Activity</td>
<td>airflow</td>
<td></td>
</tr>
<tr>
<td>Airflow Deployment Overview</td>
<td>airflow</td>
<td></td>
</tr>
<tr>
<td>Airflow Resource Utilization</td>
<td>airflow</td>
<td></td>
</tr>
<tr>
<td>Airflow Scheduler</td>
<td>airflow</td>
<td></td>
</tr>
<tr>
<td>Airflow State</td>
<td>airflow</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blackbox Exporter Overview</td>
<td>blackbox</td>
<td>prometheus</td>
</tr>
<tr>
<td>Docker Registry</td>
<td></td>
<td>registry</td>
</tr>
<tr>
<td>Elasticsearch</td>
<td></td>
<td>platform</td>
</tr>
<tr>
<td>Fluentd</td>
<td>fluentd</td>
<td></td>
</tr>
<tr>
<td>Istio Dashboard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Istio Performance Dashboard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kubernetes All Nodes</td>
<td></td>
<td>prometheus</td>
</tr>
<tr>
<td>Kubernetes Pods</td>
<td>airflow</td>
<td>platform</td>
</tr>
<tr>
<td>NGINX Ingress Controller</td>
<td></td>
<td>platform</td>
</tr>
<tr>
<td>Platform Overview</td>
<td></td>
<td>platform</td>
</tr>
<tr>
<td>Prometheus</td>
<td></td>
<td>platform</td>
</tr>
<tr>
<td>Velero</td>
<td></td>
<td>velero</td>
</tr>
</tbody>
</table>
Current

Airflow built-in dashboards based on task metadata

Airflow native statsd exporter offers deeper metrics

Future

Enhance integration options with third party services (Sumologic, Splunk, etc)

Task progress API
Airflow

Task Start

Task Progress
+ “subdag” view

Task Complete

DAG-Based Execution Engines

Apache Spark
dbt
Dagster
...

...
Thank You!

Now Q&A