Improving the Airflow User Experience

ASTRONOMER

Speakers

Ry Walker

Founder/CTO at Astronomer

@rywalker

Viraj Parekh

Head of Field Engineering at Astronomer

@vmpvmp94

Maxime Beauchemin

Founder/CEO of Preset, Creator of Apache Airflow and Apache Superset

@mistercrunch

About Astronomer

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

Products

Astronomer Enterprise

Astronomer Cloud

Locations

San Francisco | London | New York | Cincinnati | Hyderabad

100+

Enterprise customers around the world

4 of top 7

Airflow committers are Astronomer advisors or employees

Investors

venrock

SIERRA VENTURES

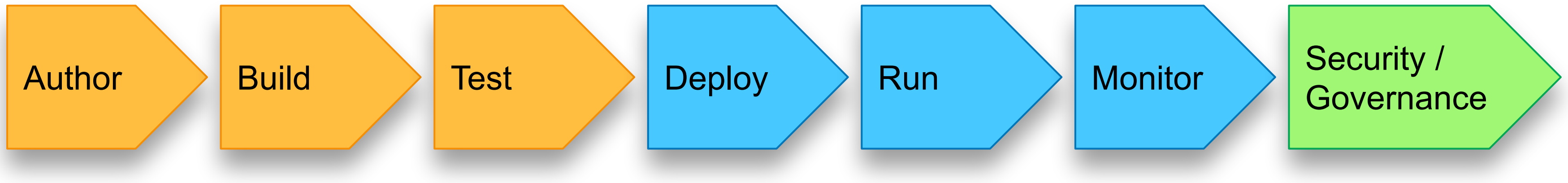
BainCapital VENTURES

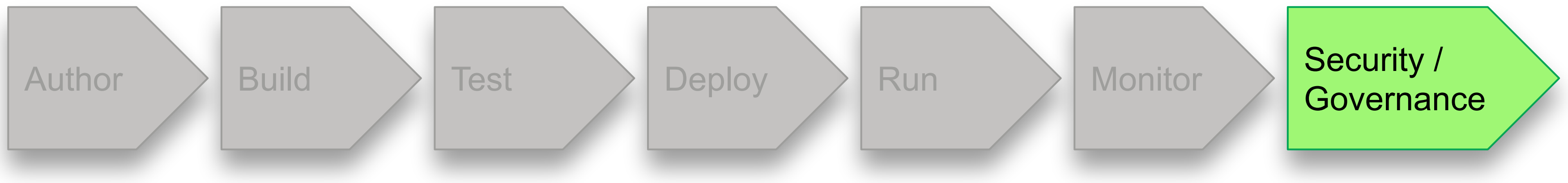
REFINERY VENTURES

Frontline

Wireframe VENTURES

7 Stages of Airflow User Experience





Current

LDAP authentication

Kerberos (w/ some operators)

Fernet key encryption

External secrets backend

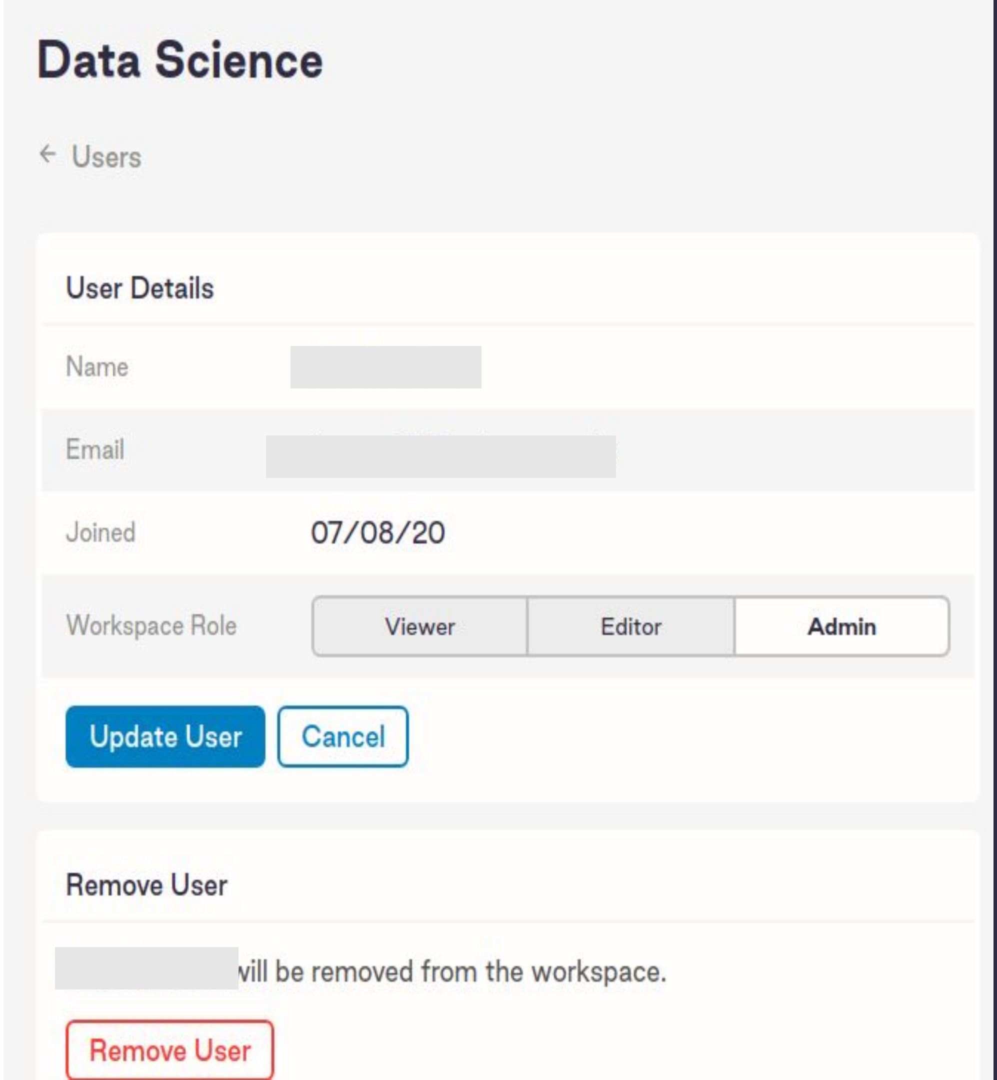
CVE Mitigations

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.



Data Science

← Users

User Details

Name	[redacted]
Email	[redacted]
Joined	07/08/20
Workspace Role	Viewer / Editor / [✓ Admin]

Update User | Cancel

Remove User

[redacted] will be removed from the workspace.

Remove User

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

Current

LDAP authentication

Kerberos (w/ some operators)

Fernet key encryption

External secrets backend

CVE Mitigations

RBAC

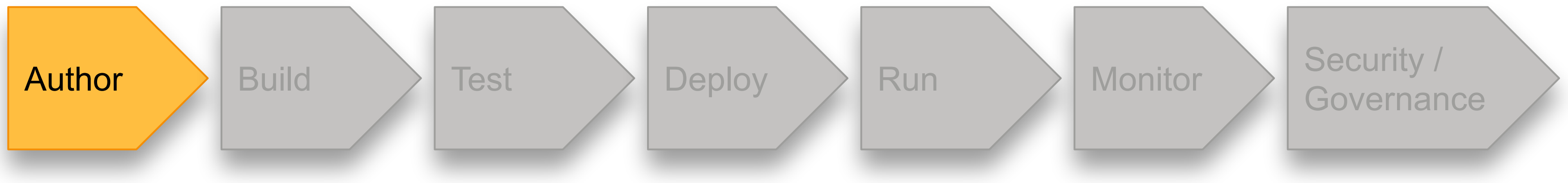
- Astronomer has multi-tenant RBAC solution built in

Future

Data lineage

Audit logs

Integration with external identity providers (Auth0, Okta, Ping, SAML)



Current

Your Text Editor + Python environment

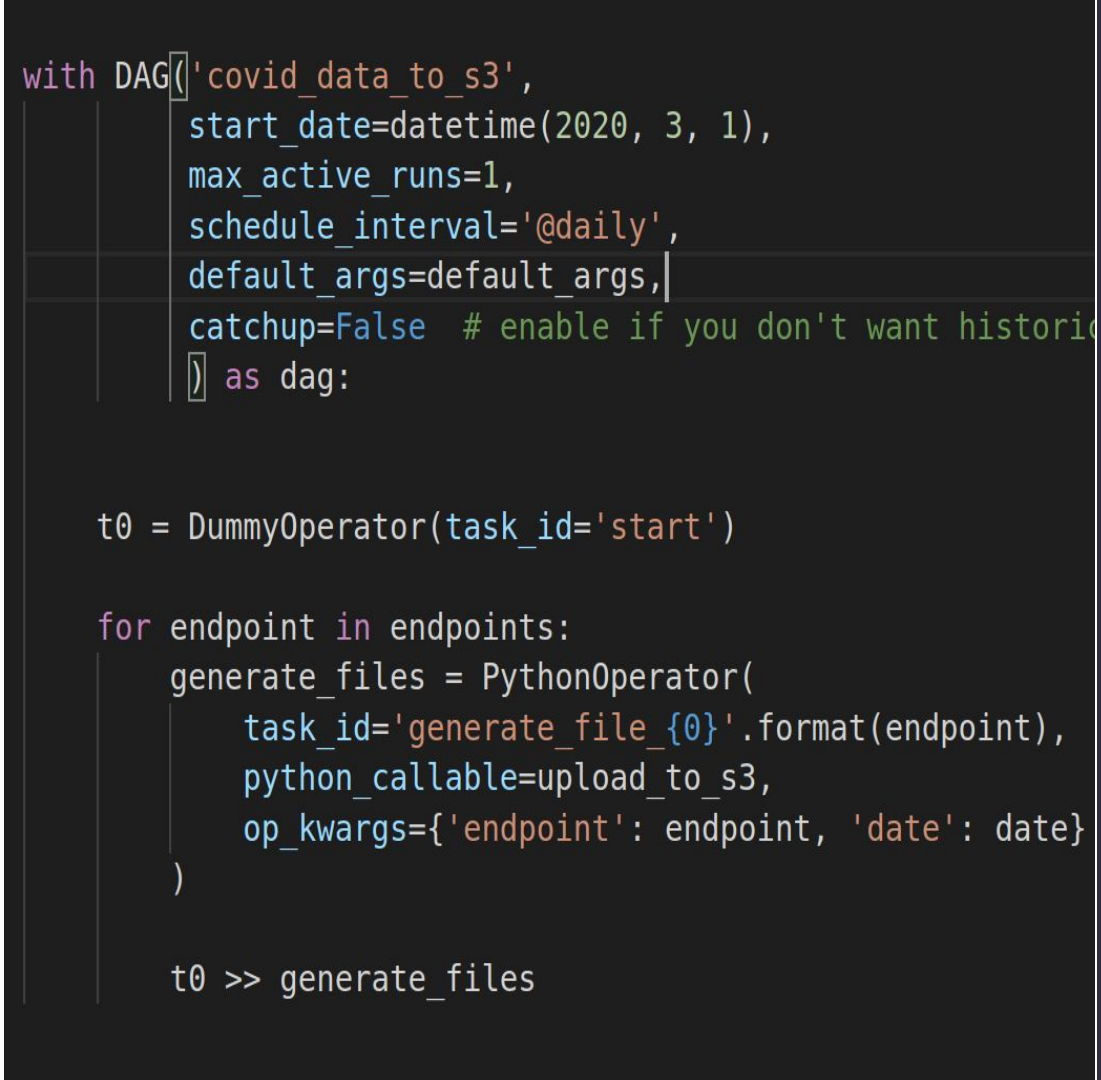
Astronomer CLI

Community Projects

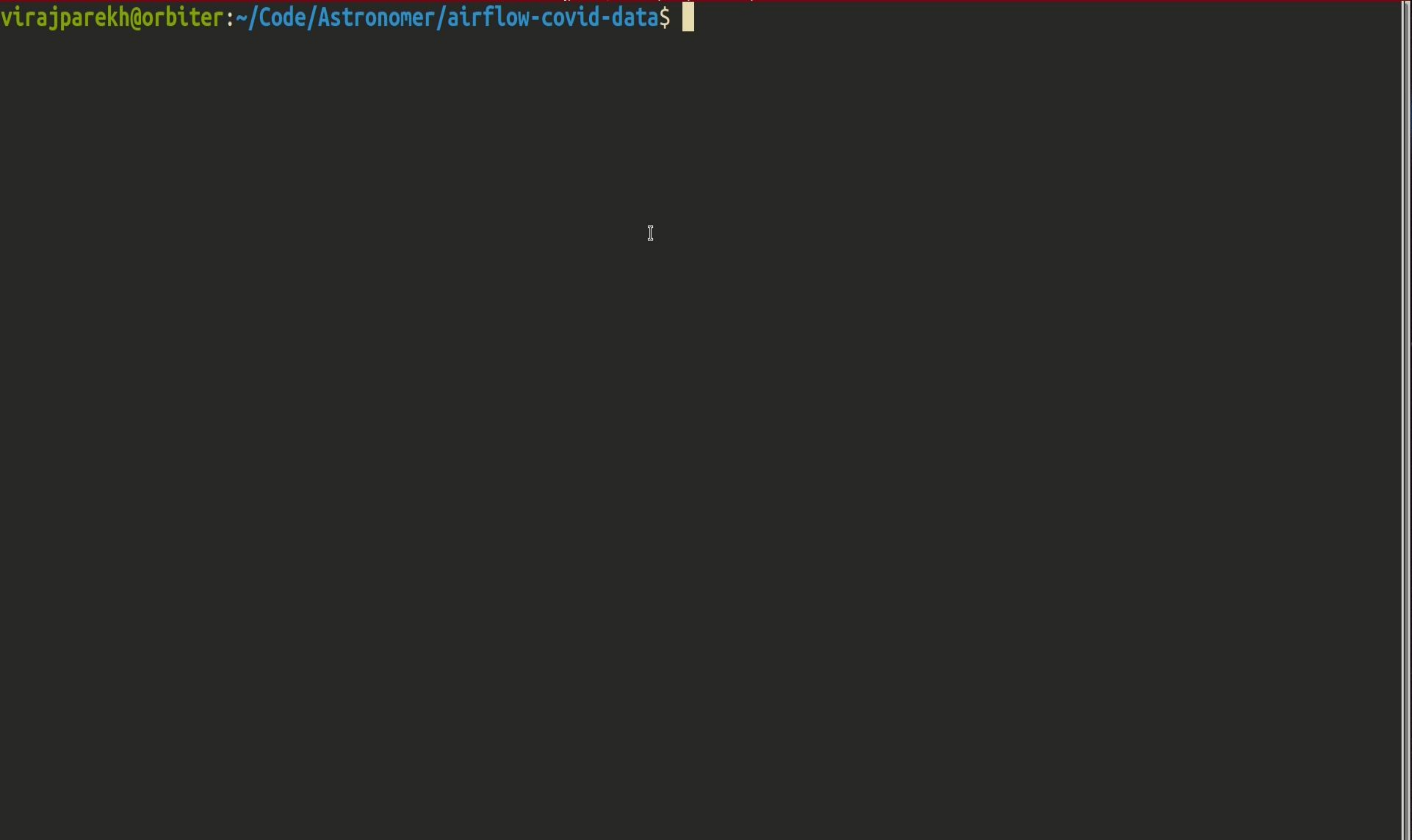
- DagFactory (DevotedHealth)
- Airflow DAG Creation Manager Plugin
- Kedro

git pull

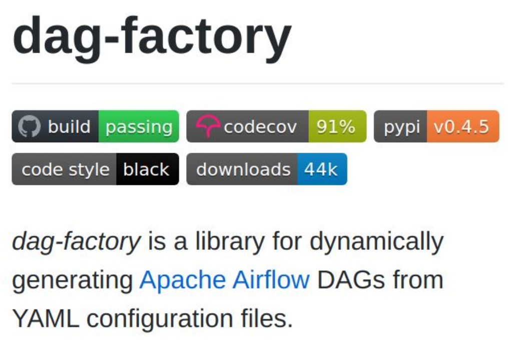
code .



```
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
```



```
virajparekh@orbiter:~/Code/Astronomer/airflow-covid-data$
```

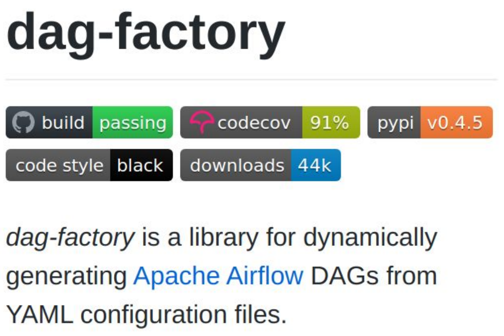


dag-factory

build passing | codecov 91% | pypi v0.4.5 | code style black | downloads 44k

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

https://github.com/ajbosco/dag-factory

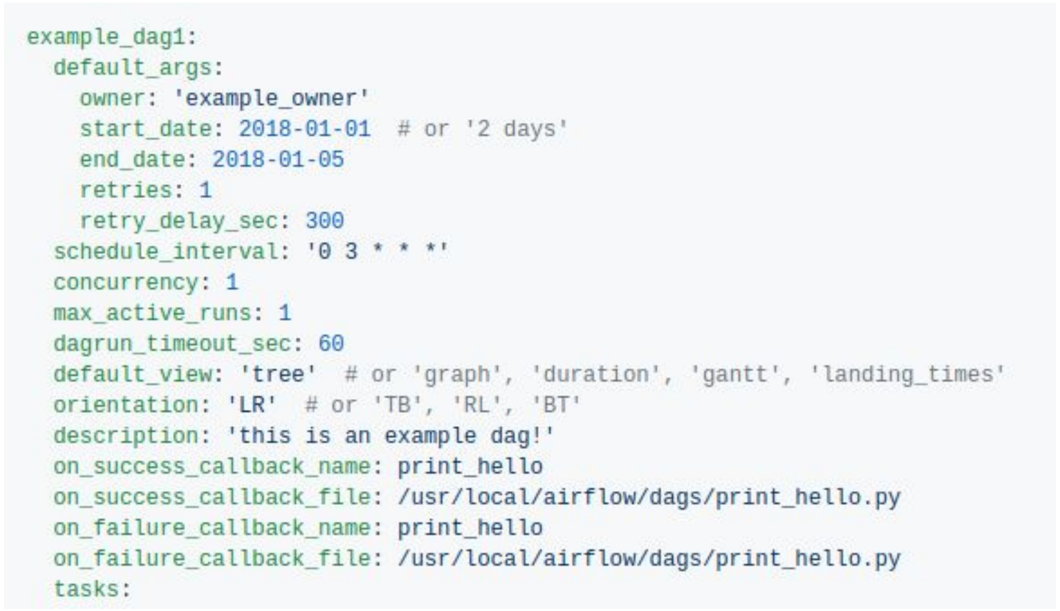


dag-factory

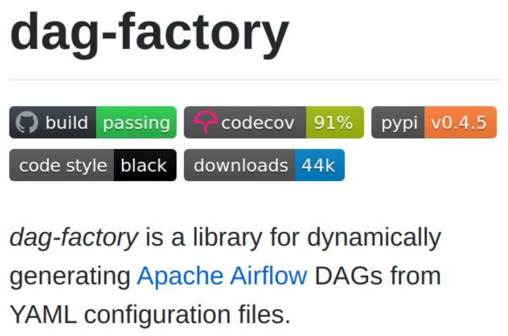
build passing | codecov 91% | pypi v0.4.5 | code style black | downloads 44k

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

Define a DAG with YAML



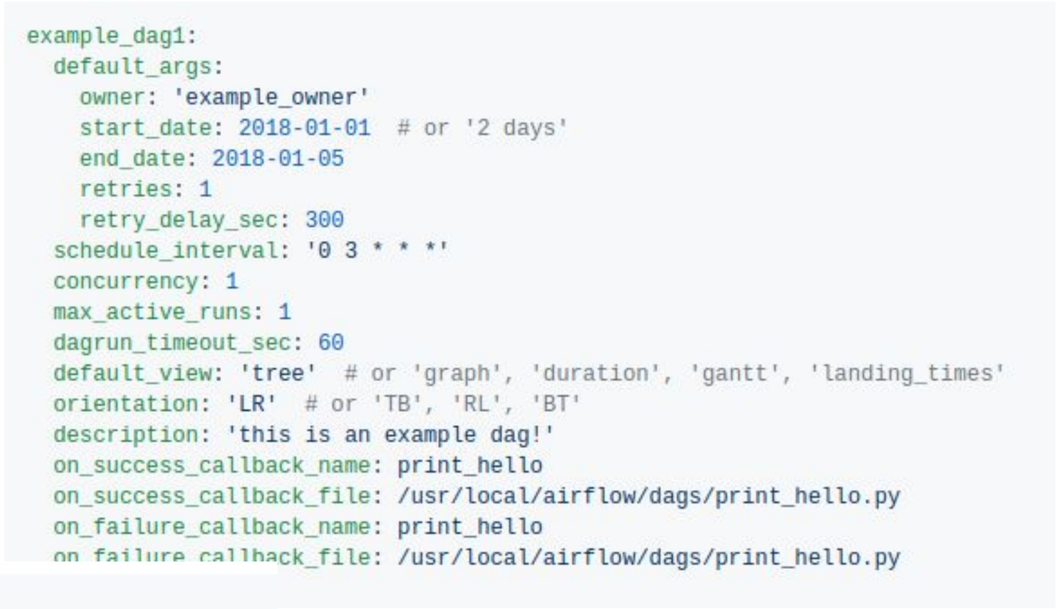
```yaml
example_dag1:
  default_args:
    owner: 'example_owner'
    start_date: 2018-01-01  # or '2 days'
    end_date: 2018-01-05
    retries: 1
    retry_delay_sec: 300
  schedule_interval: '0 3 * * *'
  concurrency: 1
  max_active_runs: 1
  dagrun_timeout_sec: 60
  default_view: 'tree'  # or 'graph', '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
  tasks:
```



dag-factory

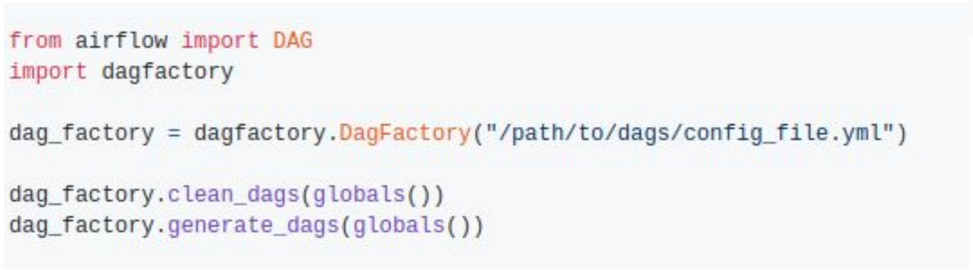
dag-factory is a library for dynamically generating Apache Airflow DAGs from YAML configuration files.

Define a DAG with YAML



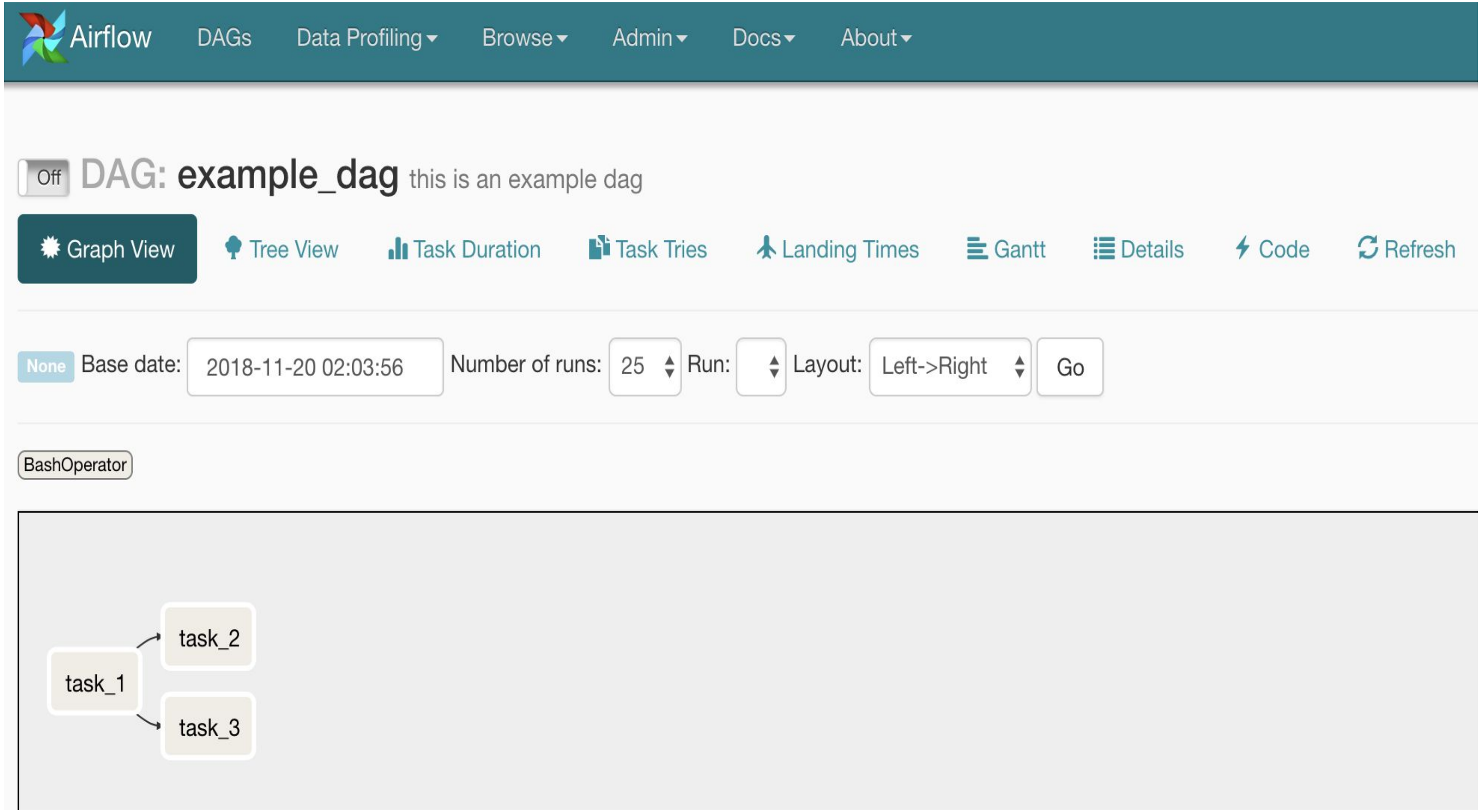
```yaml
example_dag1:
  default_args:
    owner: 'example_owner'
    start_date: 2018-01-01  # or '2 days'
    end_date: 2018-01-05
    retries: 1
    retry_delay_sec: 300
  schedule_interval: '0 3 * * *'
  concurrency: 1
  max_active_runs: 1
  dagrun_timeout_sec: 60
  default_view: 'tree'  # or 'graph', '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



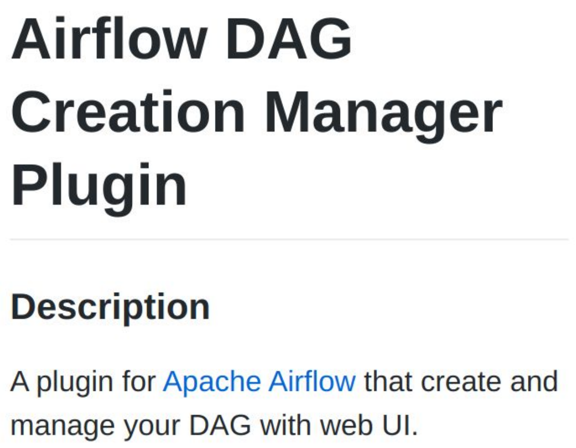
```python
from airflow import DAG
import dagfactory

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!



DAG: example_dag this is an example dag

Base date: 2018-11-20 02:03:56 Number of runs: 25 Run: Layout: Left->Right



Airflow DAG Creation Manager Plugin

Description

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

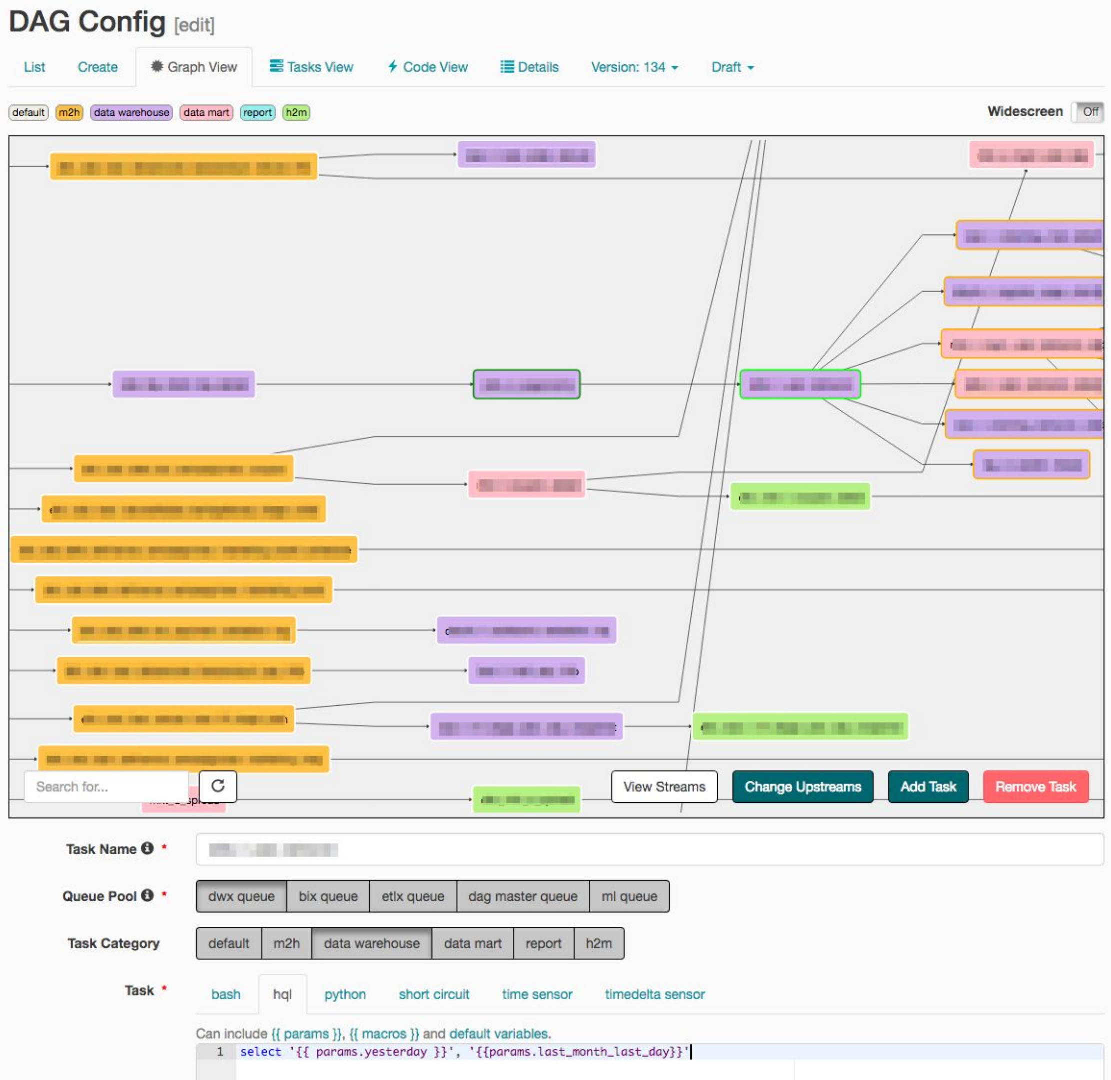
https://github.com/lattebank/airflow-dag-creation-manager-plugin

Airflow DAG Creation Manager Plugin

Description

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

Create and manage DAGS directly from the UI



Widescreen Off

Task Name * [redacted]

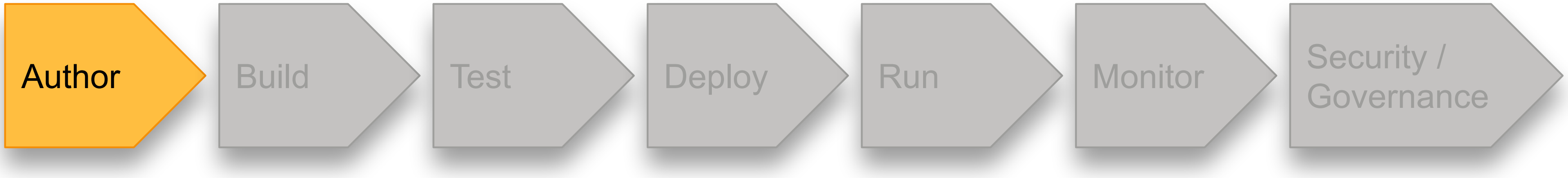
Queue Pool * [✓ dwx queue] bix queue etlx queue dag master queue ml queue

Task Category default m2h [✓ data warehouse] data mart report h2m

Task * bash [✓ hql] python short circuit time sensor timedelta sensor

Can include {{ params }}, {{ macros }} and default variables.

```
select '{{ params.yesterday }}', '{{params.last_month_last_day}}'
```



Current

Your Text Editor + Python environment

Astronomer CLI

Community Projects

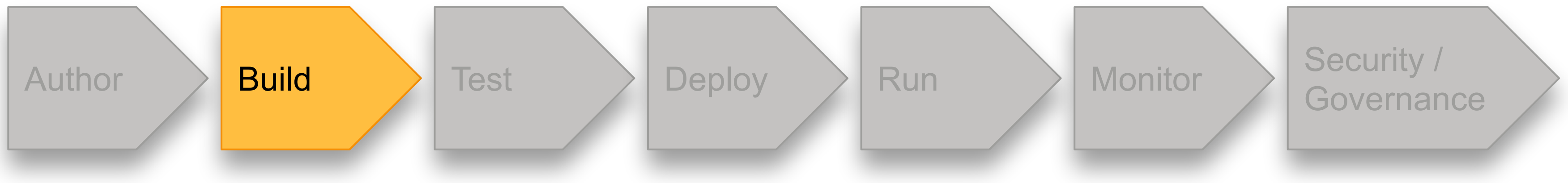
- DagFactory (DevotedHealth)
- Airflow DAG Creation Manager Plugin
- Kedro

Future

DAGs from Notebooks

Scheduling SQL query from UI

DAG Generator from standard templates



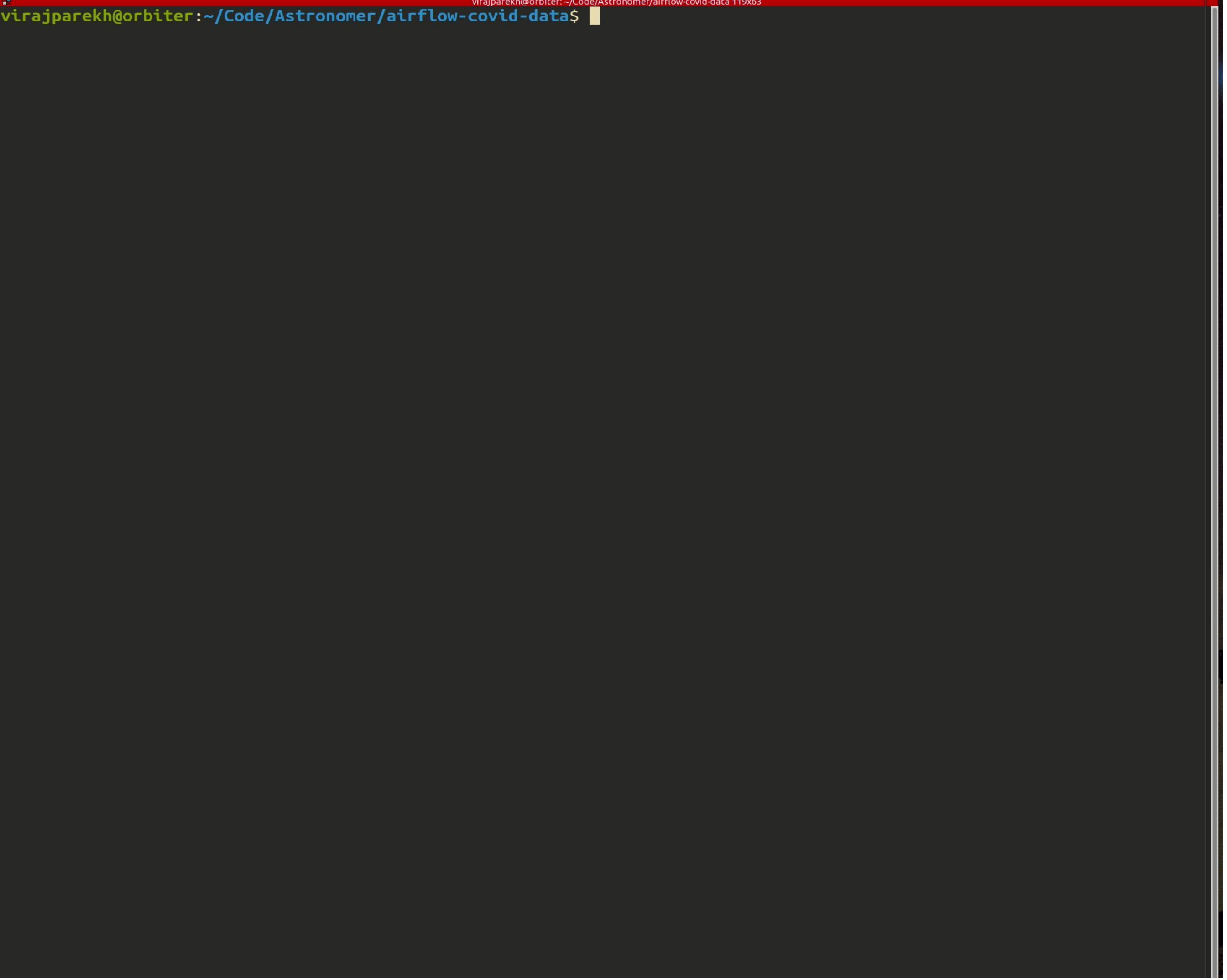
Current

Most users git-sync DAGs, add prod dependencies manually

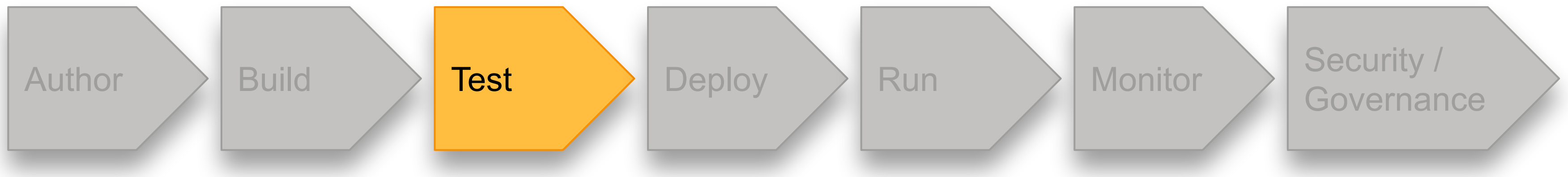
Official Community Docker Image

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



```
virajparekh@orbiter: ~/Code/Astronomer/airflow-covid-data 119x63
virajparekh@orbiter:~/Code/Astronomer/airflow-covid-data$ 
```



Current

No standardization around DAG unit testing

Adhoc testing for different data scenarios

Community Projects:

- Raybeam Status Plugin
- Great Expectations Pipeline Tutorial

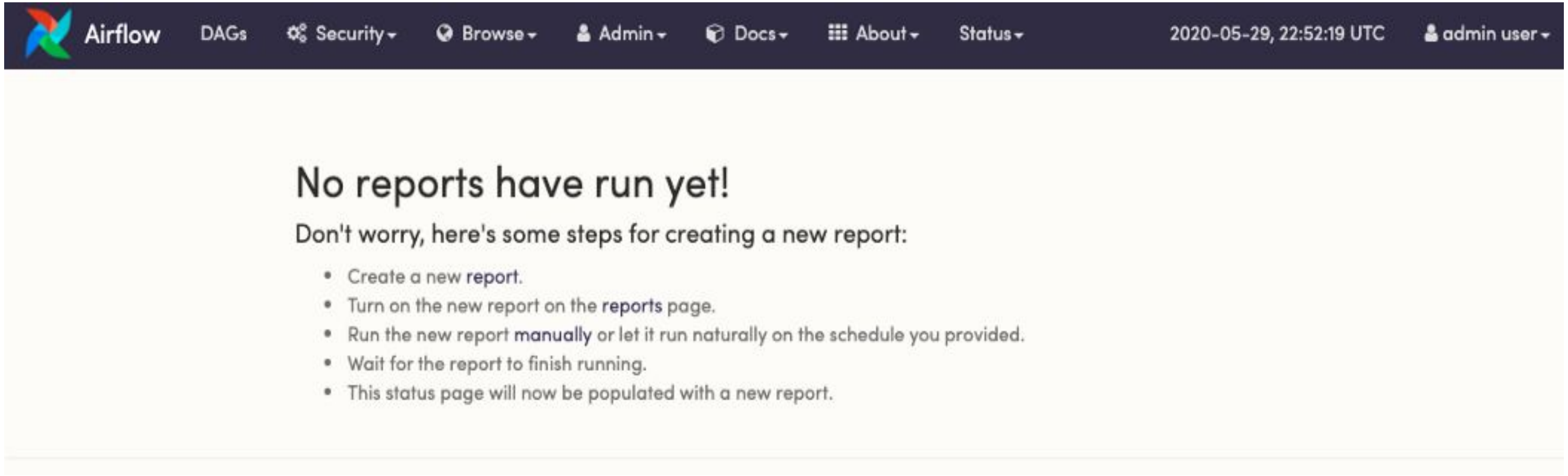
Raybeam Status Plugin

Data confidence plugin for Airflow.

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

https://github.com/Raybeam/rb_status_plugin

Is the data ready?

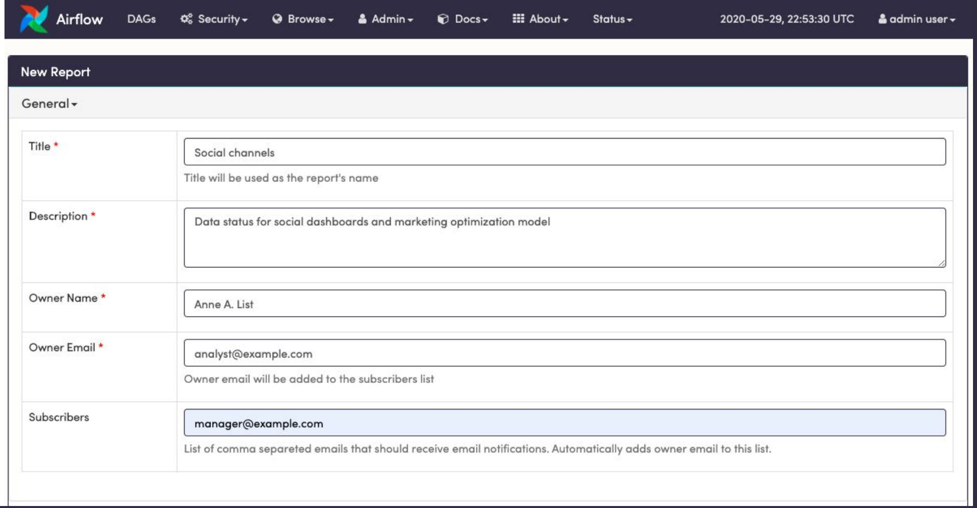


No reports have run yet!

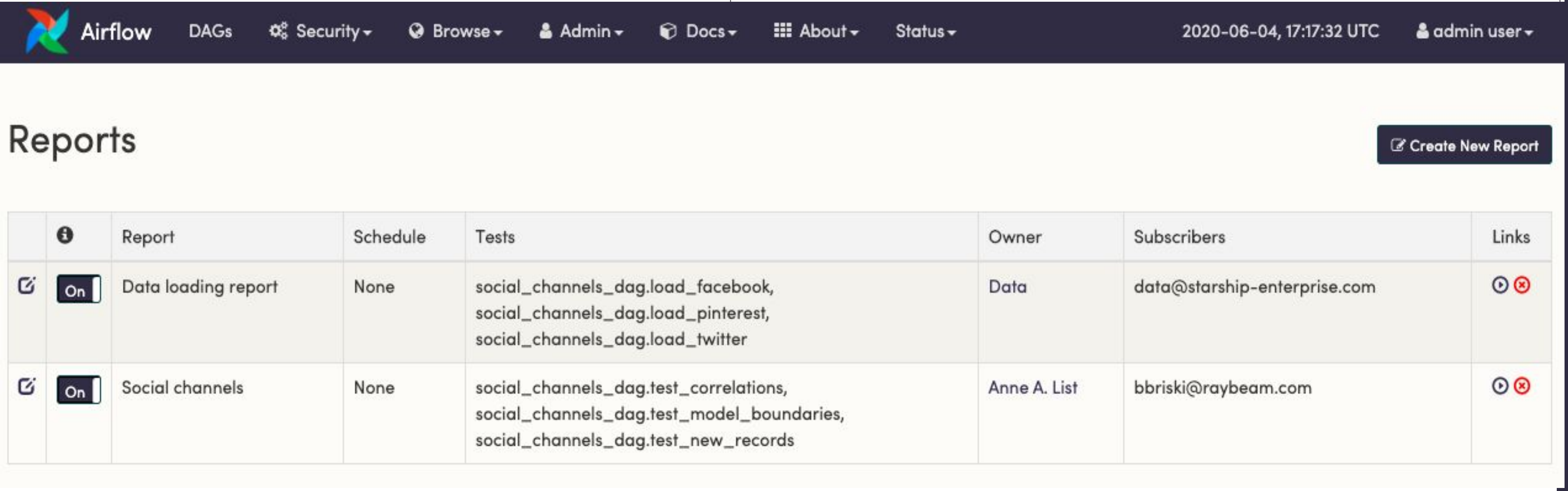
Don't worry, here's some steps for creating a new report:

- 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

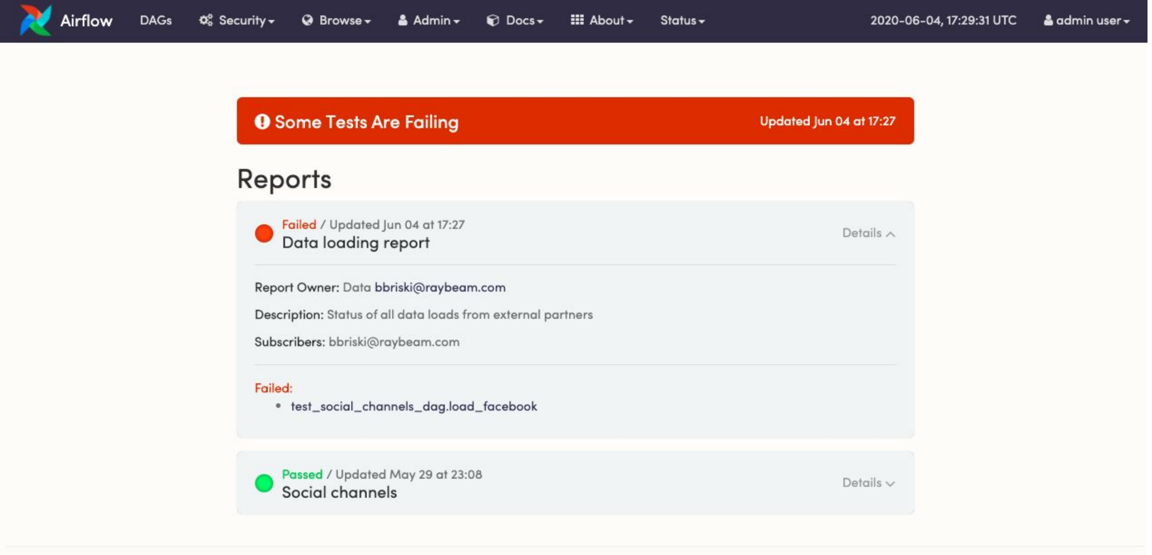


Title *	Social channels Title will be used as the report's name
Description *	Data status for social dashboards and marketing optimization model
Owner Name *	Anne A. List
Owner Email *	analyst@example.com Owner email will be added to the subscribers list
Subscribers	manager@example.com List of comma separeted emails that should receive email notifications. Automatically adds owner email to this list.



Reports

		Report	Schedule	Tests	Owner	Subscribers	Links
	On	Data loading report	None	social_channels_dag.load_facebook, social_channels_dag.load_pinterest, social_channels_dag.load_twitter	Data	data@starship-enterprise.com	
	On	Social channels	None	social_channels_dag.test_correlations, social_channels_dag.test_model_boundaries, social_channels_dag.test_new_records	Anne A. List	bbriski@raybeam.com	



Some Tests Are Failing Updated Jun 04 at 17:27

Reports

Failed / Updated Jun 04 at 17:27
Data loading report
Details

Report Owner: Data bbriski@raybeam.com

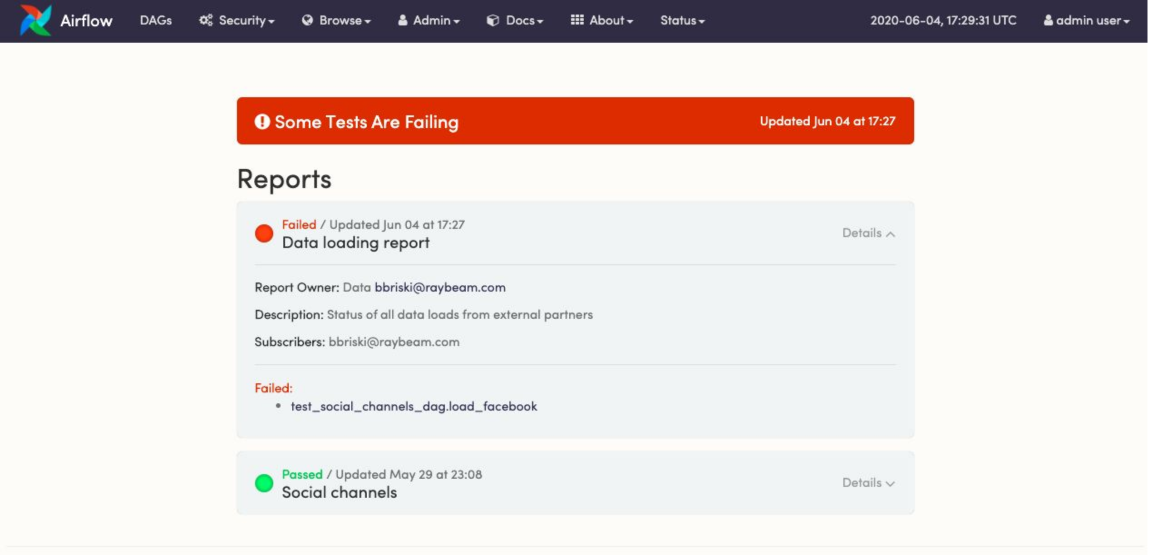
Description: Status of all data loads from external partners

Subscribers: bbriski@raybeam.com

Failed:
- test_social_channels_dag.load_facebook

Passed / Updated May 29 at 23:08
Social channels
Details

Keep stakeholders aware of data quality


Airflow DAGs Security Browse Admin Docs About Status 2020-06-04, 17:29:31 UTC admin user

Some Tests Are Failing Updated Jun 04 at 17:27

Reports

● Failed / Updated Jun 04 at 17:27
Data loading report Details

Report Owner: Data bbriski@raybeam.com

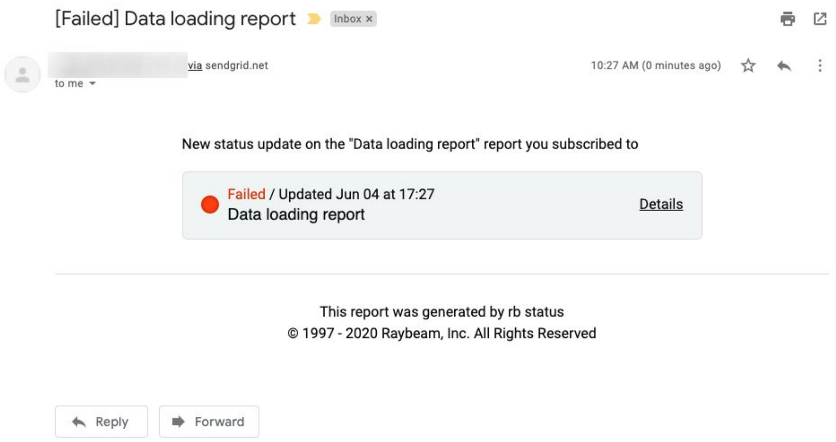
Description: Status of all data loads from external partners

Subscribers: bbriski@raybeam.com

Failed:
- test_social_channels_dag.load_facebook

● Passed / Updated May 29 at 23:08
Social channels Details

Keep stakeholders aware of data quality


[Failed] Data loading report Inbox ×

[redacted] via sendgrid.net 10:27 AM (0 minutes ago)
to me

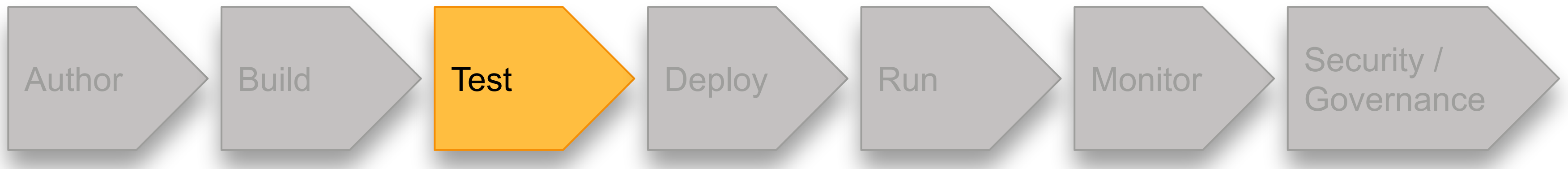
New status update on the "Data loading report" report you subscribed to

● Failed / Updated Jun 04 at 17:27
Data loading report Details

This report was generated by rb status
© 1997 - 2020 Raybeam, Inc. All Rights Reserved

Reply Forward

Hooks into existing Airflow functionality



Current

No standardization around DAG unit testing

Adhoc testing for different data scenarios

Community Projects:

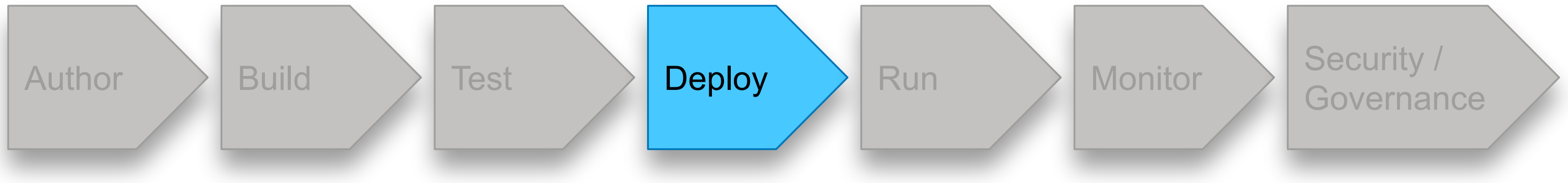
- Raybeam Status Plugin
- Great Expectations Pipeline Tutorial

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

github.com/apache/airflow/tree/master/chart

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

```
## 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.redis.pullPolicy
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.metadataConection
data.resultBackendConnection
fernetKey
fernetKeySecretName
workers.replicas
workers.keda.enabled
workers.keda.pollingInverval
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.terminationGracePeriodSeconds
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.livenessProbe.periodSeconds
webserver.readinessProbe.initialDelaySeconds
webserver.readinessProbe.timeoutSeconds
webserver.readinessProbe.failureThreshold
webserver.readinessProbe.periodSeconds
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
```

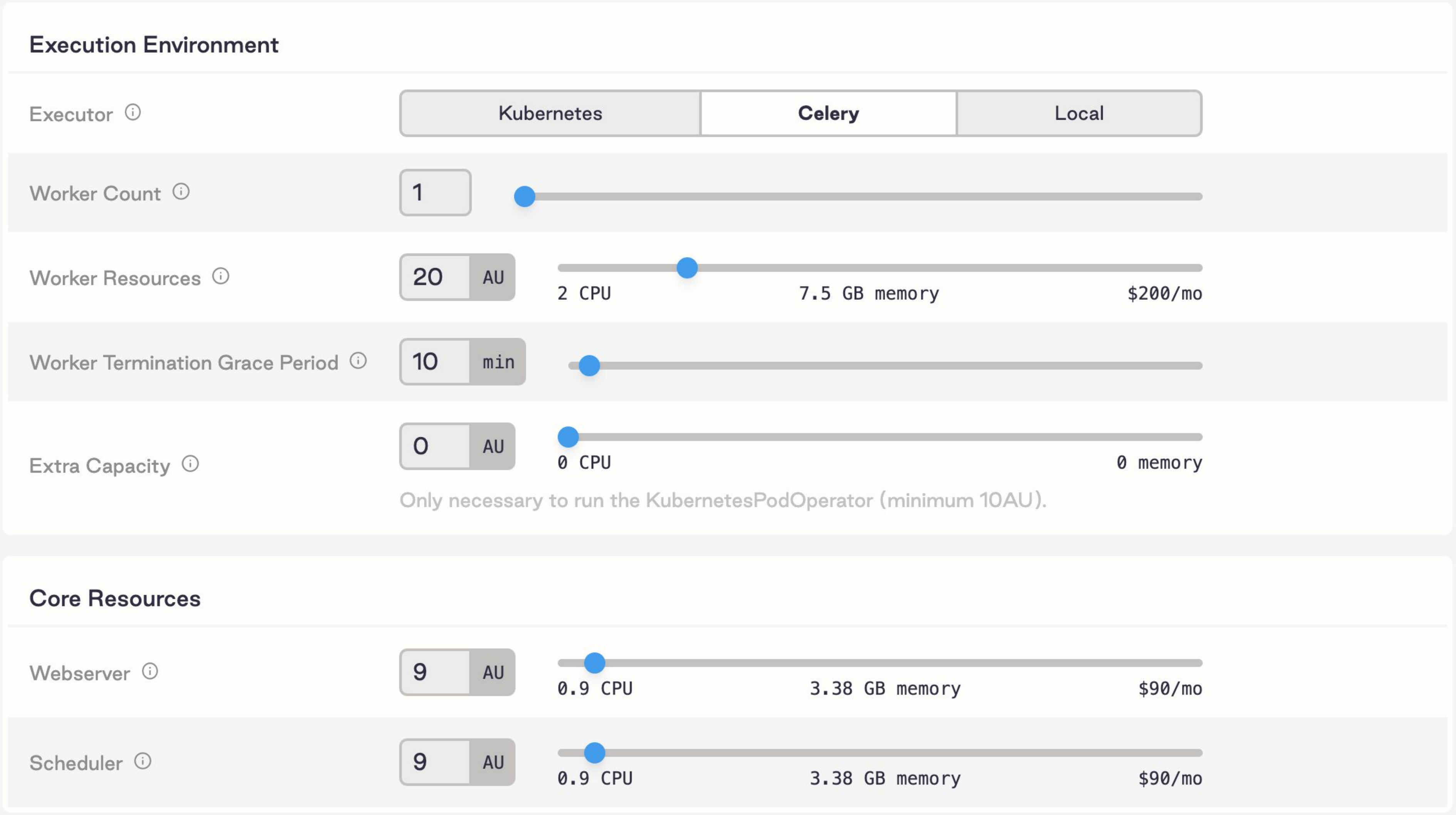
```
NAME                                    READY   STATUS    RESTARTS   AGE
airflow-ry-postgresql-0                 1/1     Running   0          6m45s
airflow-ry-scheduler-78757cd557-t8zdn   2/2     Running   0          6m45s
airflow-ry-statsd-5c889cc6b6-jxhzw      1/1     Running   0          6m45s
airflow-ry-webserver-59d79b9955-7sgp5   1/1     Running   0          6m45s
```

```
astro deployment create test-deployment --executor celery
```

```
NAME                 DEPLOYMENT NAME             ASTRO      DEPLOYMENT ID
test-deployment      theoretical-element-5806    0.15.2     ckce1ssco4uf90j16a5adkel7

 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 ckce1ssco4uf90j16a5adkel7
```

```
Successfully deleted deployment
```



Execution Environment

Executor	Kubernetes [✓ Celery] Local	
Worker Count	1	
Worker Resources	20 AU	2 CPU · 7.5 GB memory · $200/mo
Worker Termination Grace Period	10 min	
Extra Capacity	0 AU	0 CPU · 0 memory

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

Core Resources

Webserver	9 AU	0.9 CPU · 3.38 GB memory · $90/mo
Scheduler	9 AU	0.9 CPU · 3.38 GB memory · $90/mo

www.astronomer.io/guides/airflow-scaling-workers

airflow.cfg name	Environment Variable	Default Value
parallelism	AIRFLOW__CORE__PARALLELISM	32
dag_concurrency	AIRFLOW__CORE__DAG_CONCURRENCY	16
worker_concurrency	AIRFLOW__CELERY__WORKER_CONCURRENCY	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

Author | Build | Test | **Deploy** | Run | Monitor | Security / Governance

Current

Most Airflow deployments are pets, not cattle — manually deployed

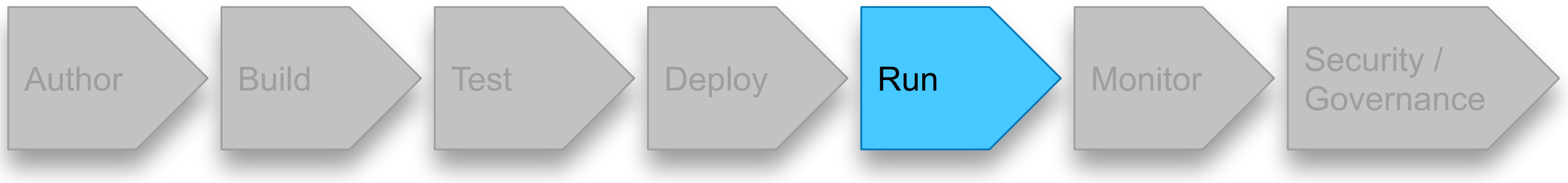
“Guess and check” for configurations

The Astronomer Way

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

Future

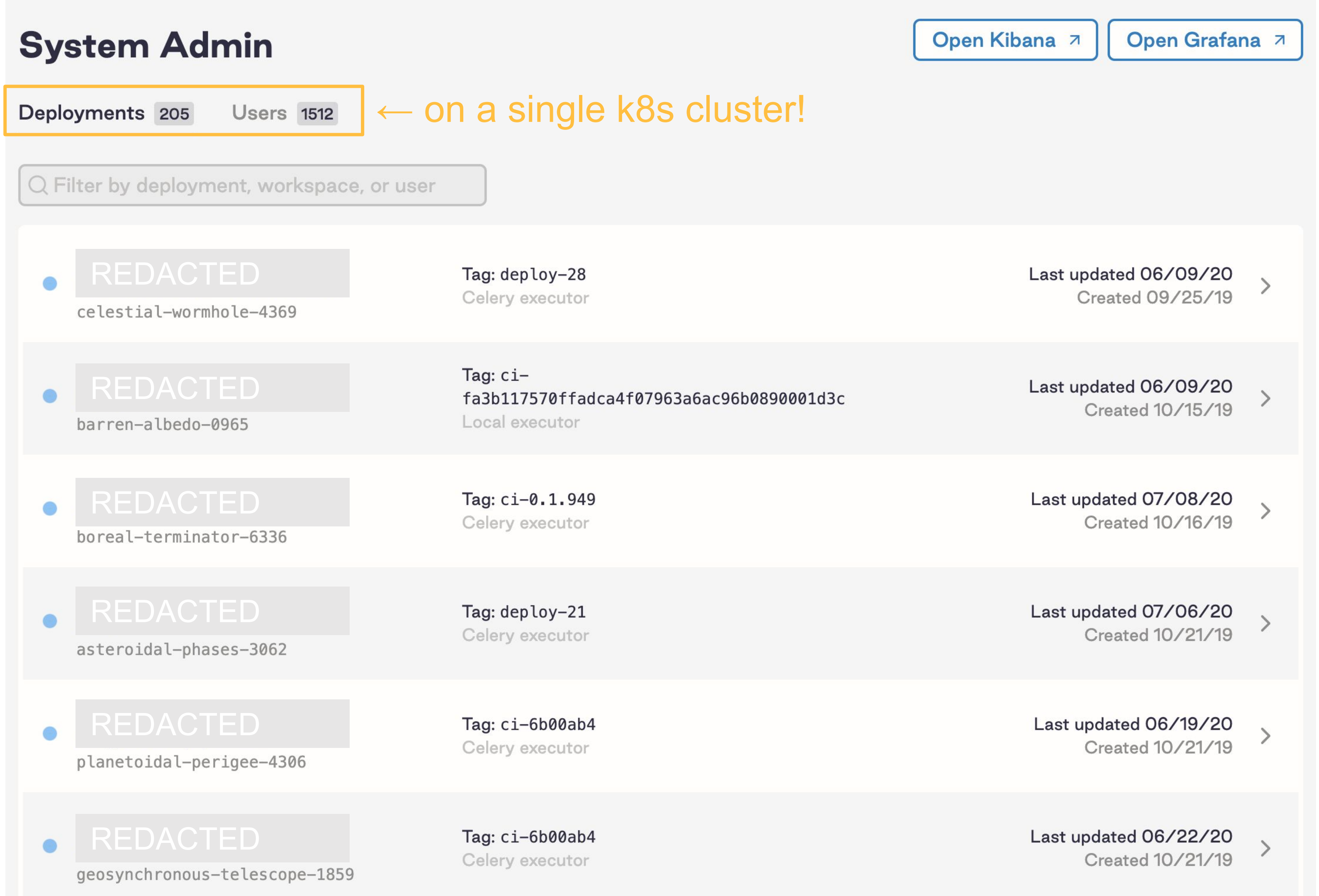
Infrastructure and configuration recommendations to optimize performance and identify bottlenecks



Current

Most Airflow deployments running on virtual machines

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



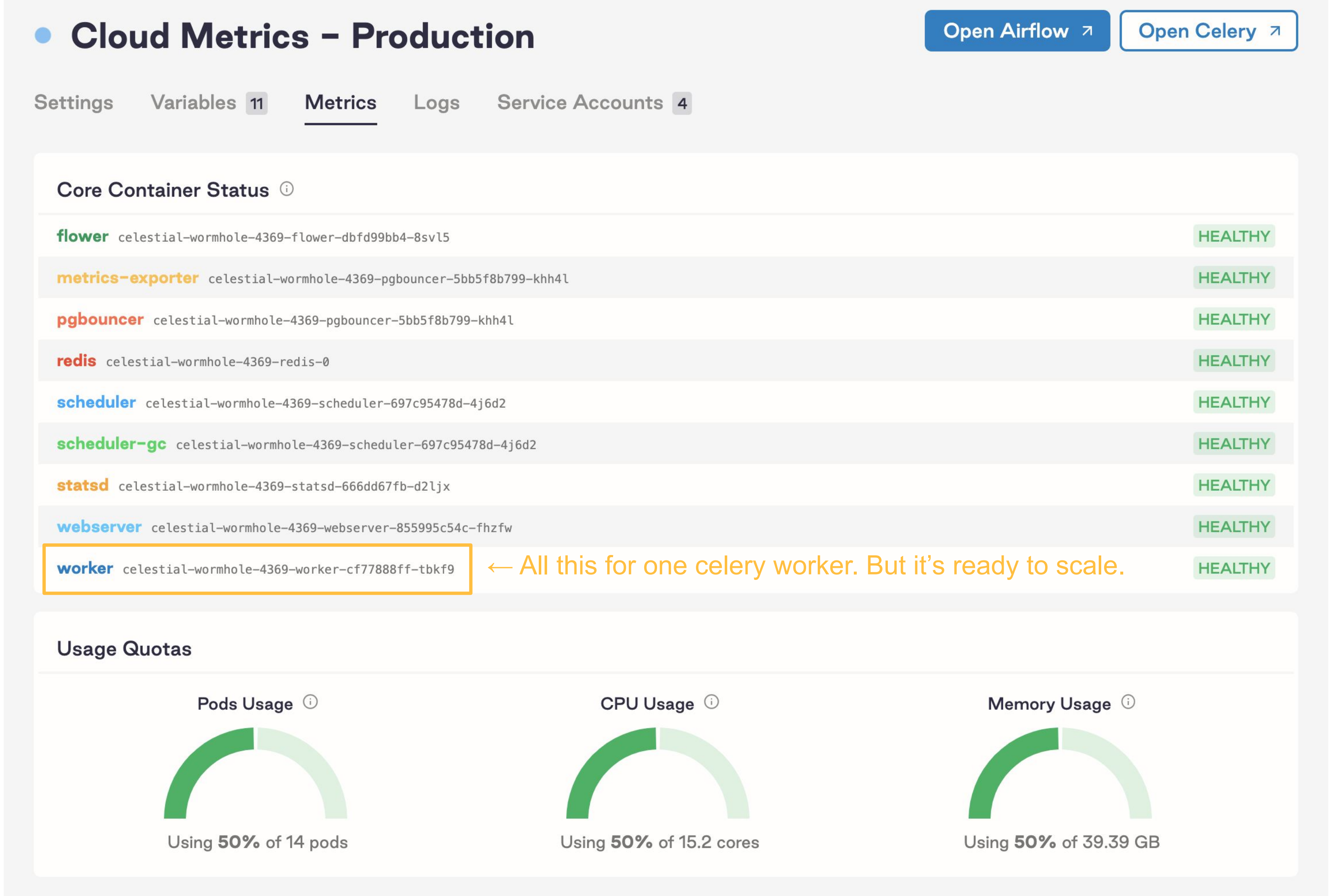
System Admin

Open Kibana ↗ Open Grafana ↗

Deployments 205 Users 1512 ← on a single k8s cluster!

Filter by deployment, workspace, or user

REDACTED celestial-wormhole-4369	Tag: deploy-28 Celery executor	Last updated 06/09/20 Created 09/25/19
REDACTED barren-albedo-0965	Tag: ci-fa3b117570ffadca4f07963a6ac96b0890001d3c Local executor	Last updated 06/09/20 Created 10/15/19
REDACTED boreal-terminator-6336	Tag: ci-0.1.949 Celery executor	Last updated 07/08/20 Created 10/16/19
REDACTED asteroidal-phases-3062	Tag: deploy-21 Celery executor	Last updated 07/06/20 Created 10/21/19
REDACTED planetoidal-perigee-4306	Tag: ci-6b00ab4 Celery executor	Last updated 06/19/20 Created 10/21/19
REDACTED geosynchronous-telescope-1859	Tag: ci-6b00ab4 Celery executor	Last updated 06/22/20 Created 10/21/19



Cloud Metrics - Production

Open Airflow ↗ Open Celery ↗

Settings | Variables 11 | Metrics | Logs | Service Accounts 4

Core Container Status ⓘ

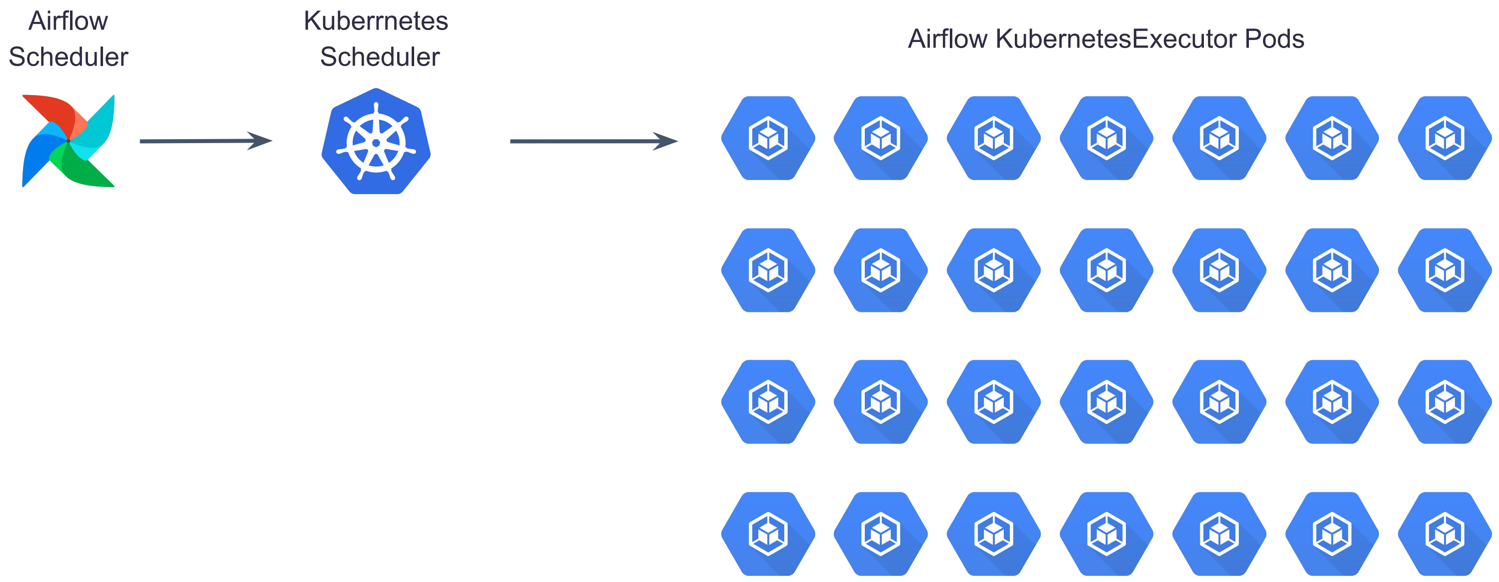
flower	celestial-wormhole-4369-flower-dbfd99bb4-8svl5	HEALTHY
metrics-exporter	celestial-wormhole-4369-pgbouncer-5bb5f8b799-khh4l	HEALTHY
pgbouncer	celestial-wormhole-4369-pgbouncer-5bb5f8b799-khh4l	HEALTHY
redis	celestial-wormhole-4369-redis-0	HEALTHY
scheduler	celestial-wormhole-4369-scheduler-697c95478d-4j6d2	HEALTHY
scheduler-gc	celestial-wormhole-4369-scheduler-697c95478d-4j6d2	HEALTHY
statsd	celestial-wormhole-4369-statsd-666dd67fb-d2ljx	HEALTHY
webserver	celestial-wormhole-4369-webserver-855995c54c-fhzfw	HEALTHY
worker	celestial-wormhole-4369-worker-cf77888ff-tbkf9	HEALTHY

← All this for one celery worker. But it's ready to scale.

Usage Quotas

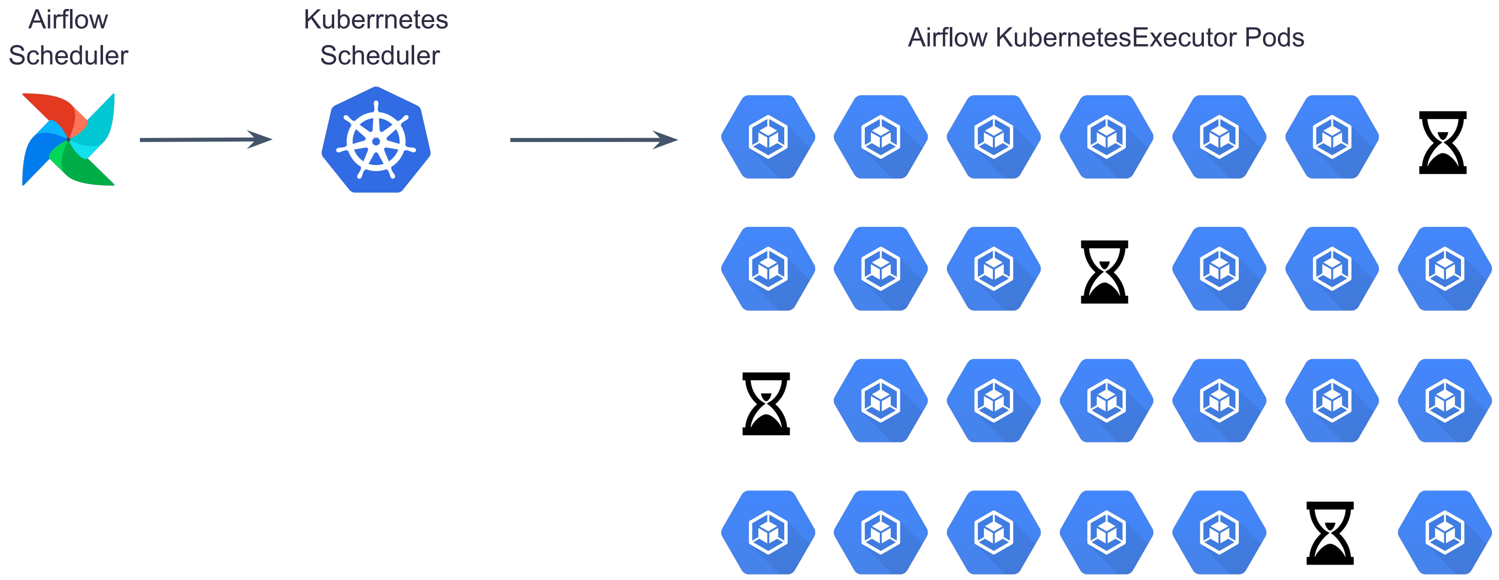
Pods Usage ⓘ	CPU Usage ⓘ	Memory Usage ⓘ
Using **50%** of 14 pods	Using **50%** of 15.2 cores	Using **50%** of 39.39 GB

The challenge w/ KubernetesExecutor



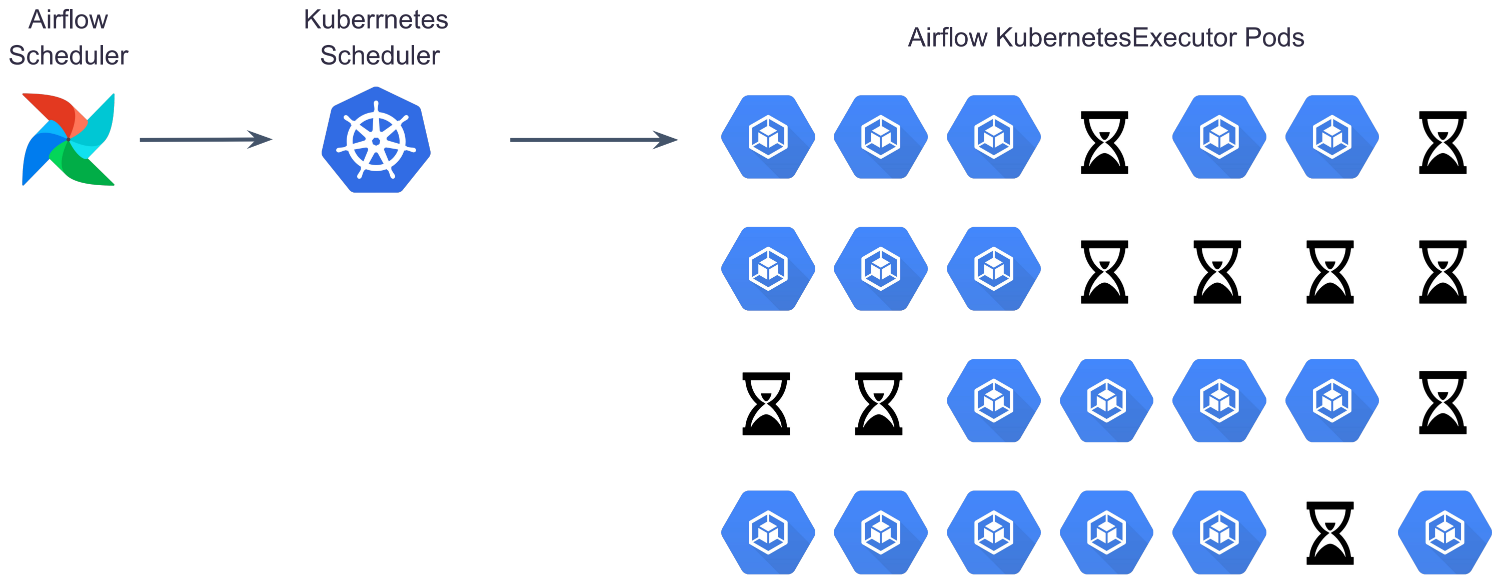
Long-running tasks

The challenge w/ KubernetesExecutor



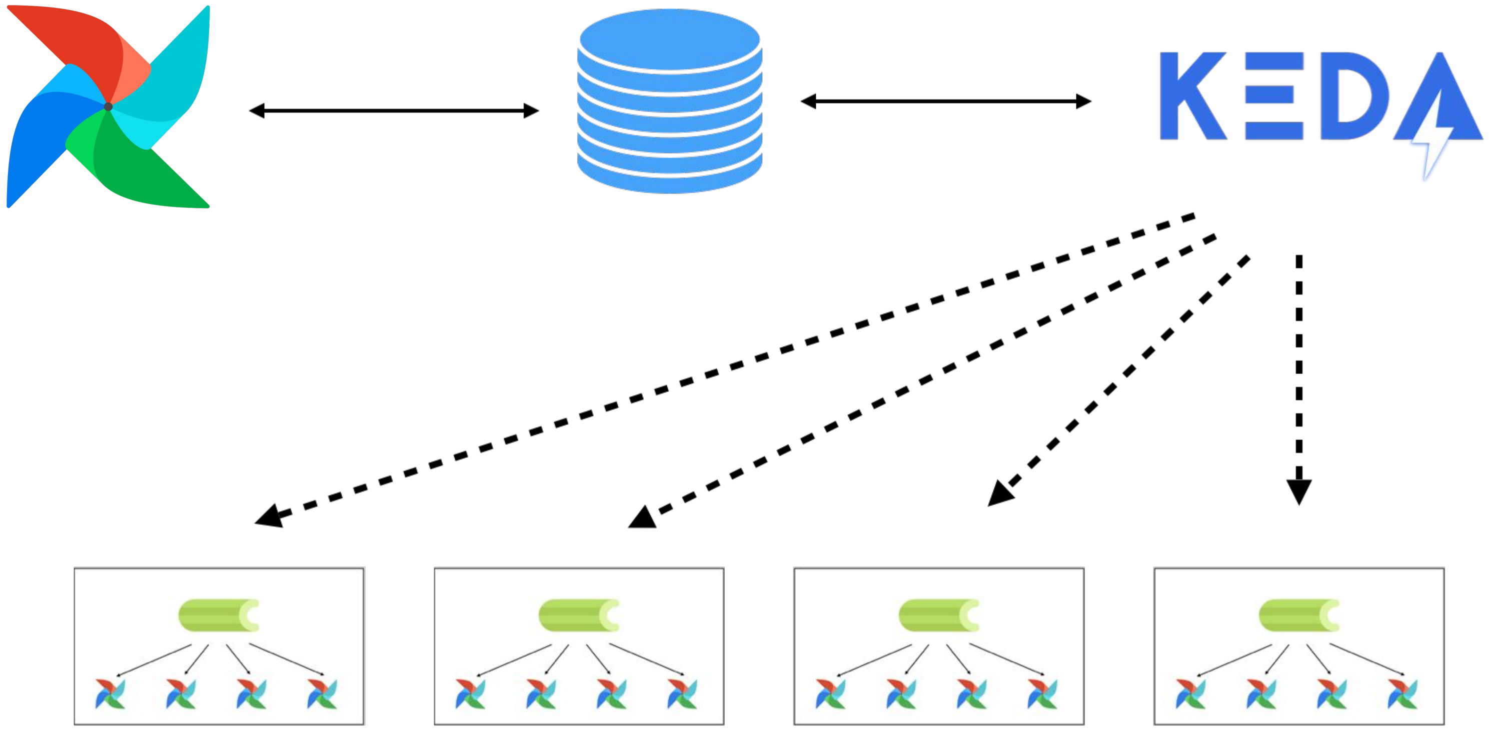
Medium-running tasks

The challenge w/ KubernetesExecutor



Short-running tasks

Celery with KEDA



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

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

Current

Most Airflow deployments running on virtual machines

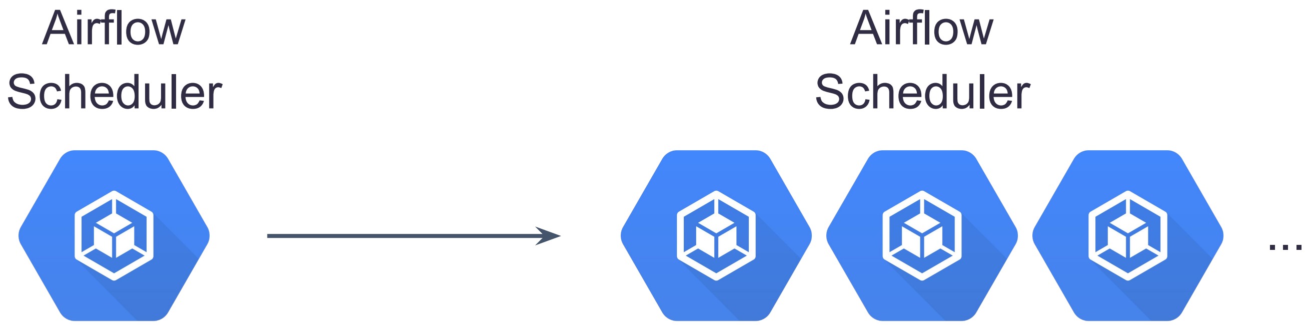
Running in K8s enhances stability, observability, and ability to scale

Future

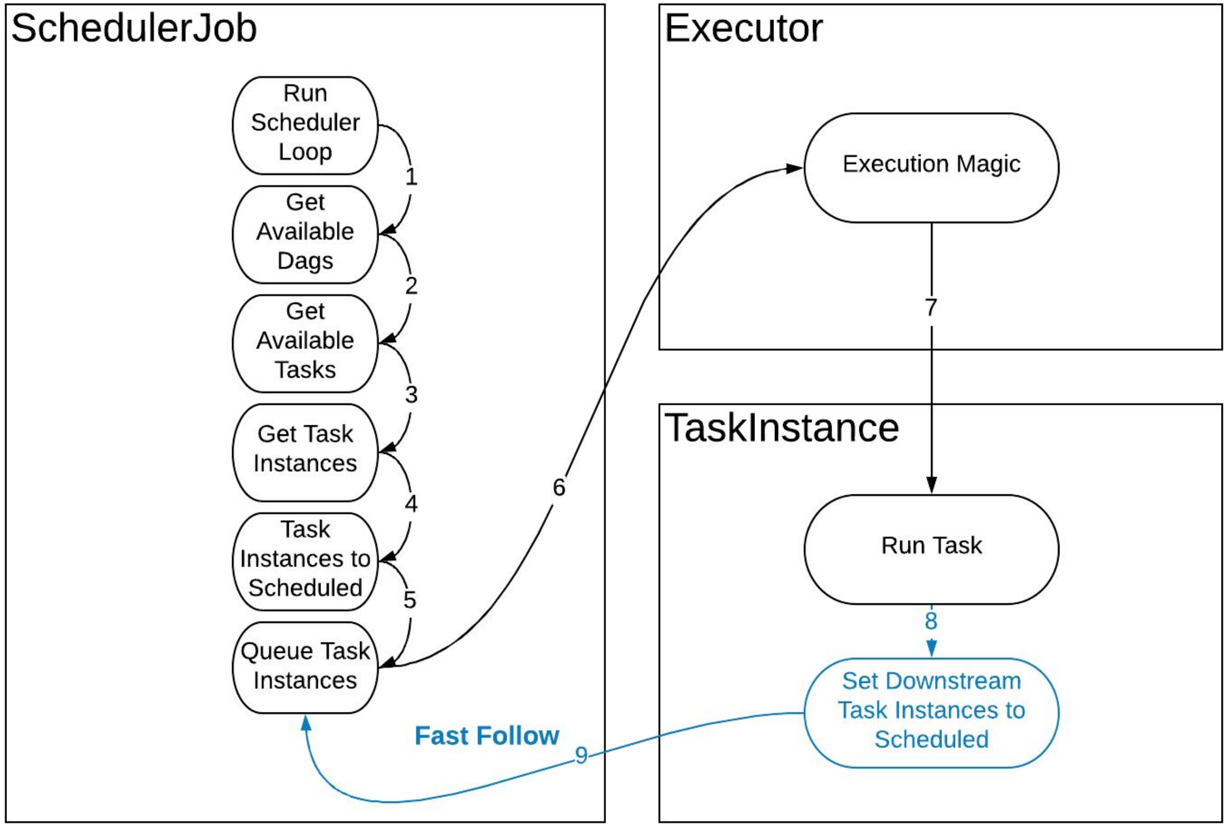
Highly Available Scheduler

“Fastfollow” task scheduling

HA Scheduler



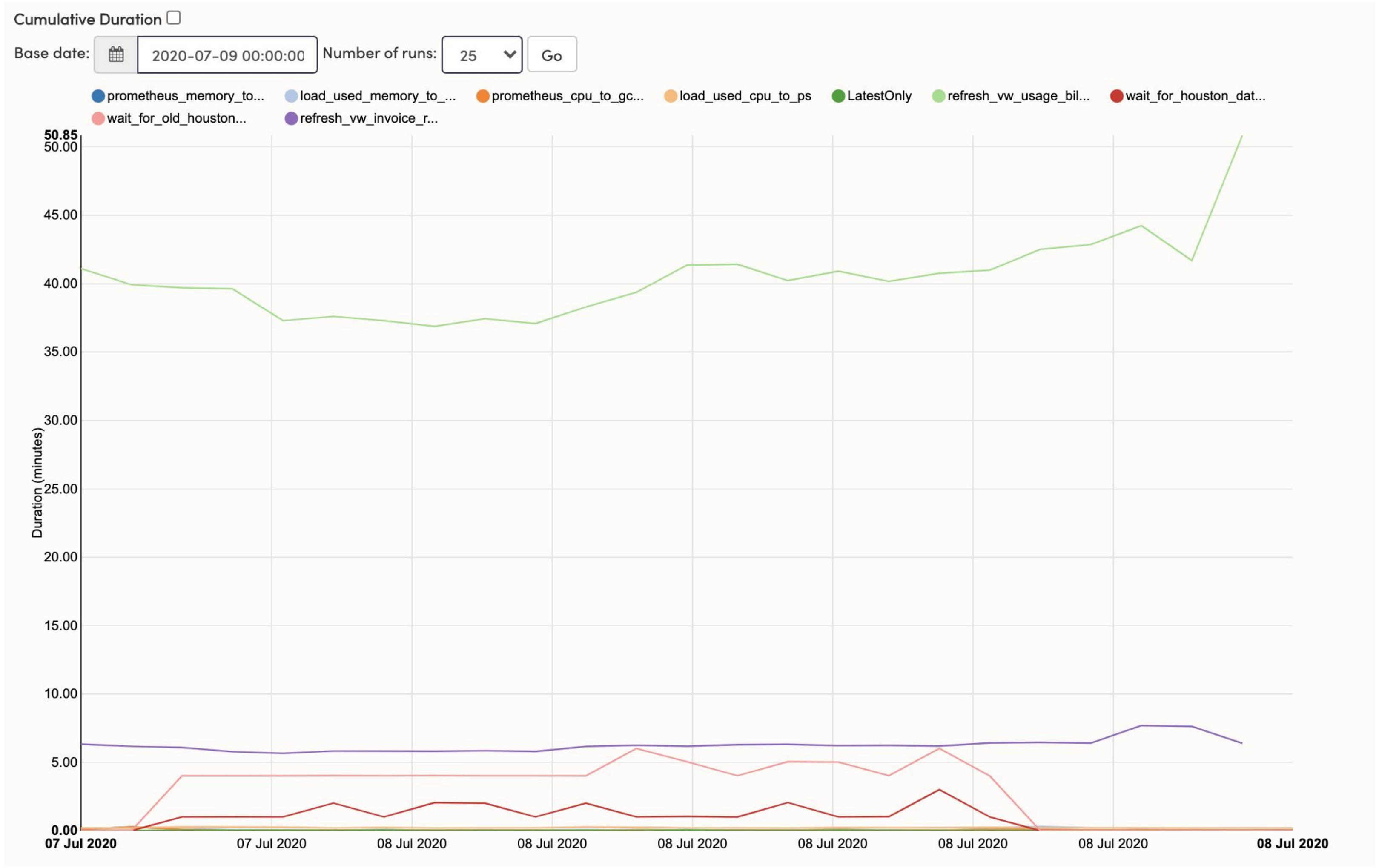
Fast follow


SchedulerJob
Run Scheduler Loop
1
Get Available Dags
2
Get Available Tasks
3
Get Task Instances
4
Task Instances to Scheduled
5
Queue Task Instances
6
Executor
Execution Magic
7
TaskInstance
Run Task
8
Set Downstream Task Instances to Scheduled
9
Fast Follow

Current

Airflow built-in dashboards based on task metadata

Airflow native statsd exporter offers deeper metrics



Cumulative Duration
Base date: 2020-07-09 00:00:00
Number of runs: 25
Go
● prometheus_memory_to...
● load_used_memory_to_...
● prometheus_cpu_to_gc...
● load_used_cpu_to_ps
● LatestOnly
● refresh_vw_usage_bil...
● wait_for_houston_dat...
● wait_for_old_houston...
● refresh_vw_invoice_r...
Duration (minutes)
50.85
50.00
45.00
40.00
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
07 Jul 2020
07 Jul 2020
08 Jul 2020
08 Jul 2020
08 Jul 2020
08 Jul 2020
08 Jul 2020
08 Jul 2020
08 Jul 2020

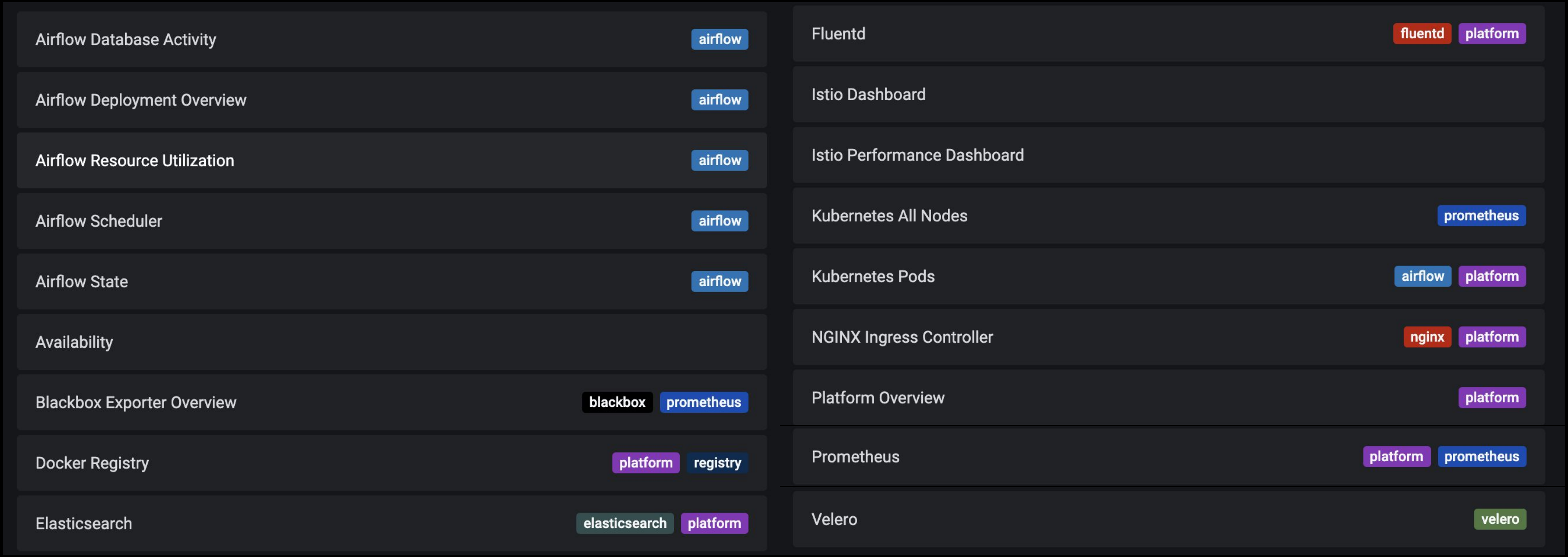
airflow.apache.org/docs/stable/metrics.html

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
scheduler.tasks.killed_externally

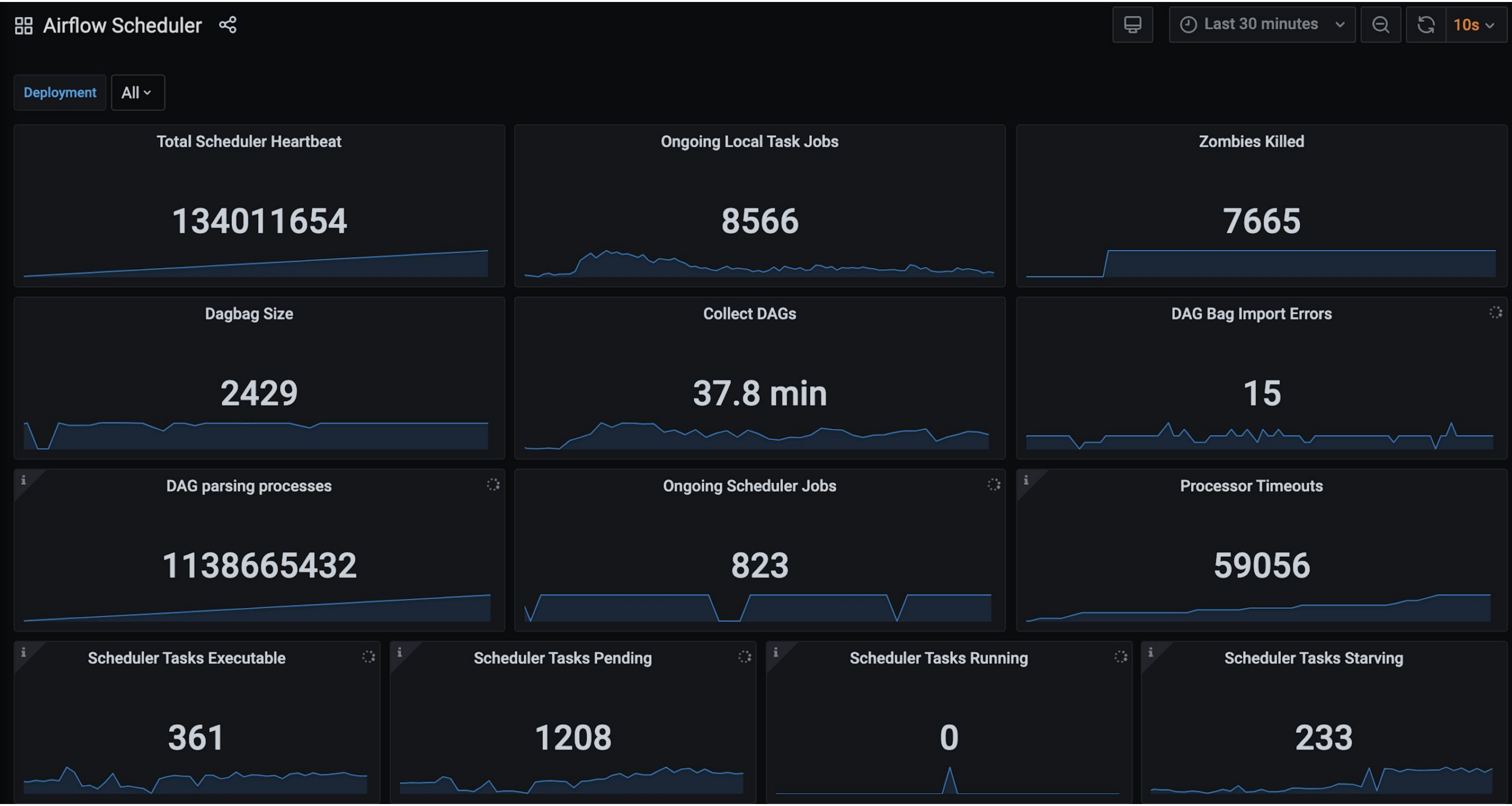
Timers
dagrun.dependency-check.<dag_id>
dag.<dag_id>.<task_id>.duration
dag_processing.last_duration.<dag_file>
dagrun.duration.success.<dag_id>
dagrun.duration.failed.<dag_id>
dagrun.schedule_delay.<dag_id>

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
executor.open_slots
executor.queued_tasks
executor.running_tasks
pool.open_slots.<pool_name>
pool.used_slots.<pool_name>
pool.starving_tasks.<pool_name>

STATSD



Airflow Database Activity	airflow
Airflow Deployment Overview	airflow
Airflow Resource Utilization	airflow
Airflow Scheduler	airflow
Airflow State	airflow
Availability	
Blackbox Exporter Overview	blackbox prometheus
Docker Registry	platform registry
Elasticsearch	elasticsearch platform
Fluentd	fluentd platform
Istio Dashboard	
Istio Performance Dashboard	
Kubernetes All Nodes	prometheus
Kubernetes Pods	airflow platform
NGINX Ingress Controller	nginx platform
Platform Overview	platform
Prometheus	platform prometheus
Velero	velero



Total Scheduler Heartbeat 134011654

Ongoing Local Task Jobs 8566

Zombies Killed 7665

Dagbag Size 2429

Collect DAGs 37.8 min

DAG Bag Import Errors 15

DAG parsing processes 1138665432

Ongoing Scheduler Jobs 823

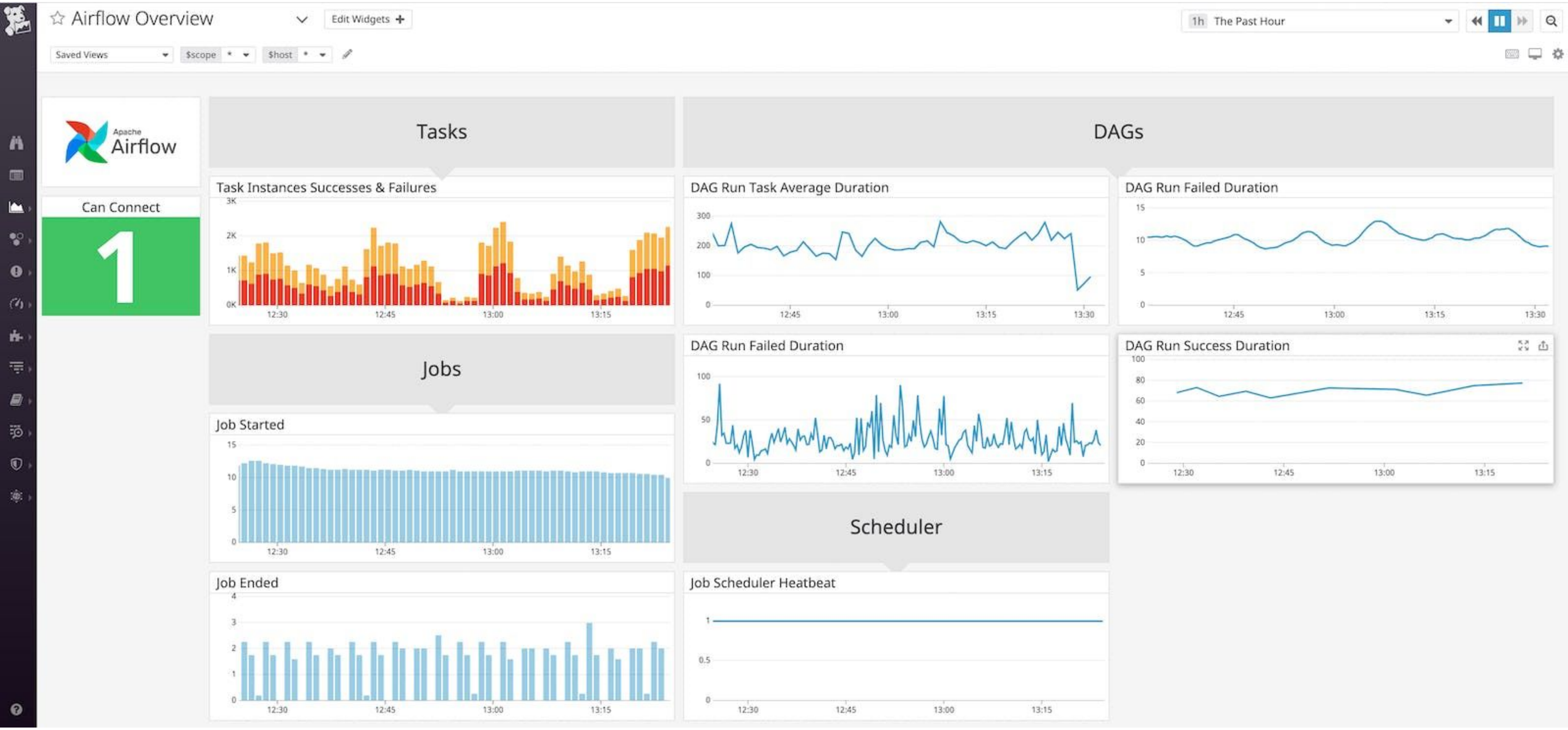
Processor Timeouts 59056

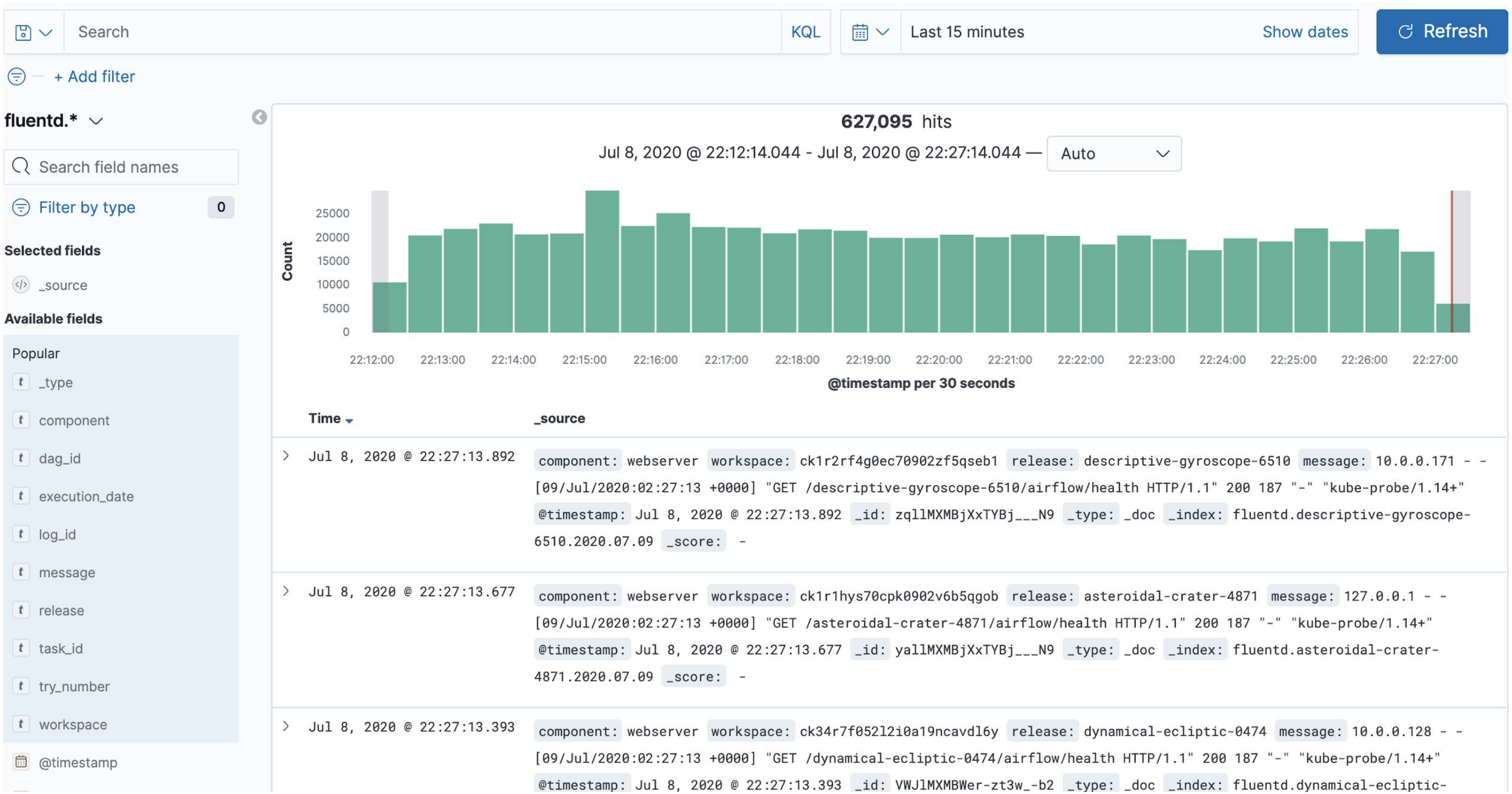
Scheduler Tasks Executable 361

Scheduler Tasks Pending 1208

Scheduler Tasks Running 0

Scheduler Tasks Starving 233

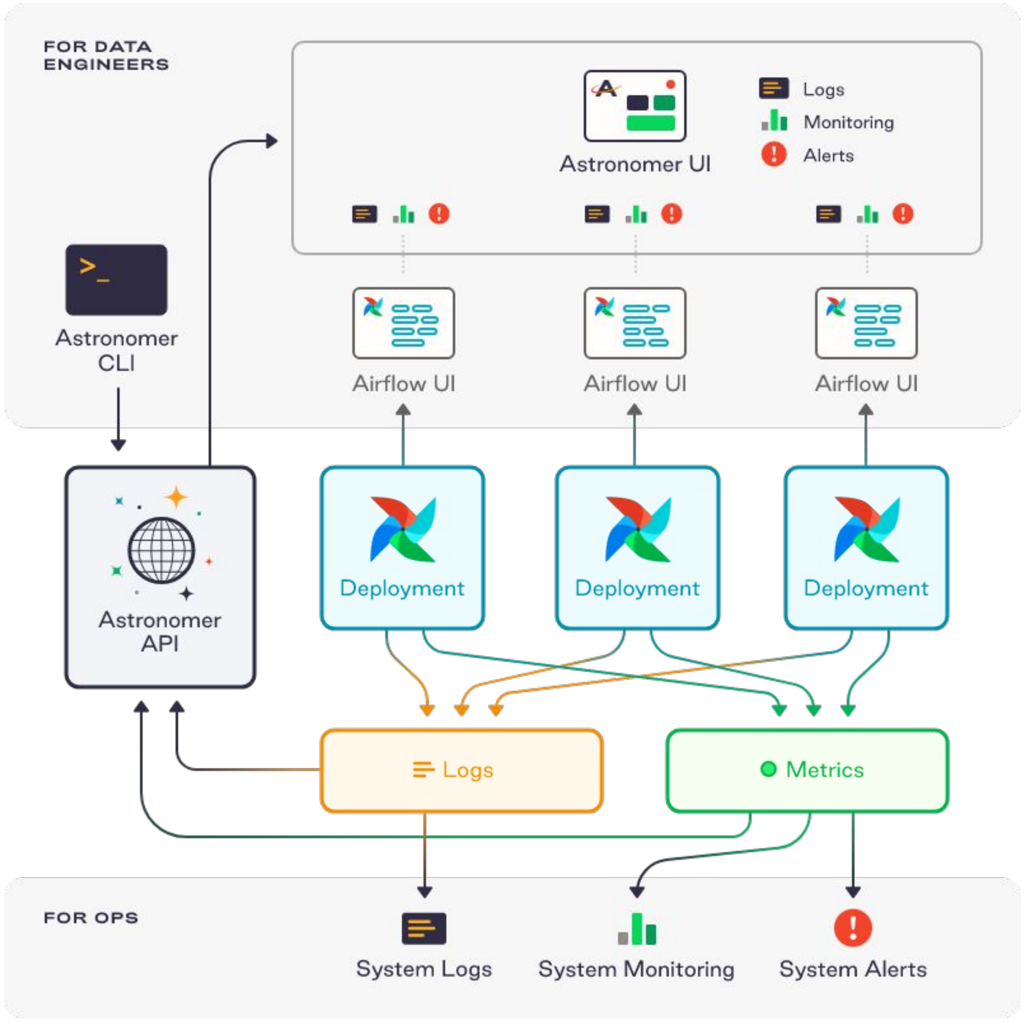

Airflow Overview
Edit Widgets
1h The Past Hour
Saved Views
$scope *
$host *
Apache Airflow
Can Connect
1
Tasks
DAGs
Task Instances Successes & Failures
DAG Run Task Average Duration
DAG Run Failed Duration
Jobs
Job Started
DAG Run Failed Duration
DAG Run Success Duration
Scheduler
Job Ended
Job Scheduler Heatbeat

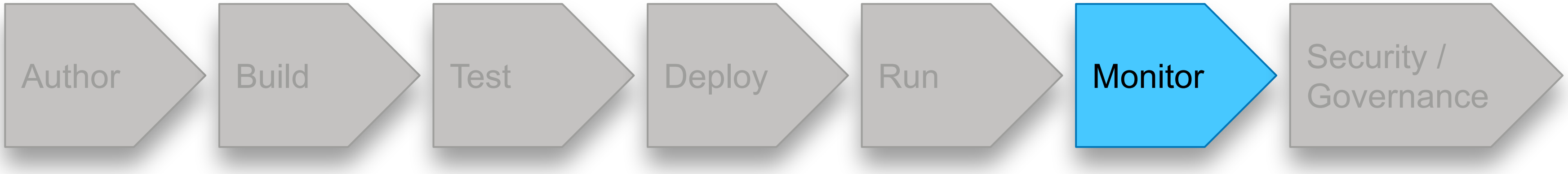


627,095 hits

Jul 8, 2020 @ 22:12:14.044 - Jul 8, 2020 @ 22:27:14.044 — Auto

```
Jul 8, 2020 @ 22:27:13.892  component: webserver workspace: ck1r2rf4g0ec70902zf5qseb1 release: descriptive-gyroscope-6510 message: 10.0.0.171 - -
                            [09/Jul/2020:02:27:13 +0000] "GET /descriptive-gyroscope-6510/airflow/health HTTP/1.1" 200 187 "-" "kube-probe/1.14+"
                            @timestamp: Jul 8, 2020 @ 22:27:13.892 _id: zqllMXMBjXxTYBj___N9 _type: _doc _index: fluentd.descriptive-gyroscope-
                            6510.2020.07.09 _score: -

Jul 8, 2020 @ 22:27:13.677  component: webserver workspace: ck1r1hys70cpk0902v6b5qgob release: asteroidal-crater-4871 message: 127.0.0.1 - -
                            [09/Jul/2020:02:27:13 +0000] "GET /asteroidal-crater-4871/airflow/health HTTP/1.1" 200 187 "-" "kube-probe/1.14+"
                            @timestamp: Jul 8, 2020 @ 22:27:13.677 _id: yallMXMBjXxTYBj___N9 _type: _doc _index: fluentd.asteroidal-crater-
                            4871.2020.07.09 _score: -

Jul 8, 2020 @ 22:27:13.393  component: webserver workspace: ck34r7f052l2i0a19ncavdl6y release: dynamical-ecliptic-0474 message: 10.0.0.128 - -
                            [09/Jul/2020:02:27:13 +0000] "GET /dynamical-ecliptic-0474/airflow/health HTTP/1.1" 200 187 "-" "kube-probe/1.14+"
                            @timestamp: Jul 8, 2020 @ 22:27:13.393 _id: VWJlMXMBWer-zt3w_-b2 _type: _doc _index: fluentd.dynamical-ecliptic-
```


FOR DATA ENGINEERS
Astronomer UI
Logs
Monitoring
Alerts
Astronomer CLI
Airflow UI
Airflow UI
Airflow UI
Astronomer API
Deployment
Deployment
Deployment
Logs
Metrics
FOR OPS
System Logs
System Monitoring
System Alerts



Current

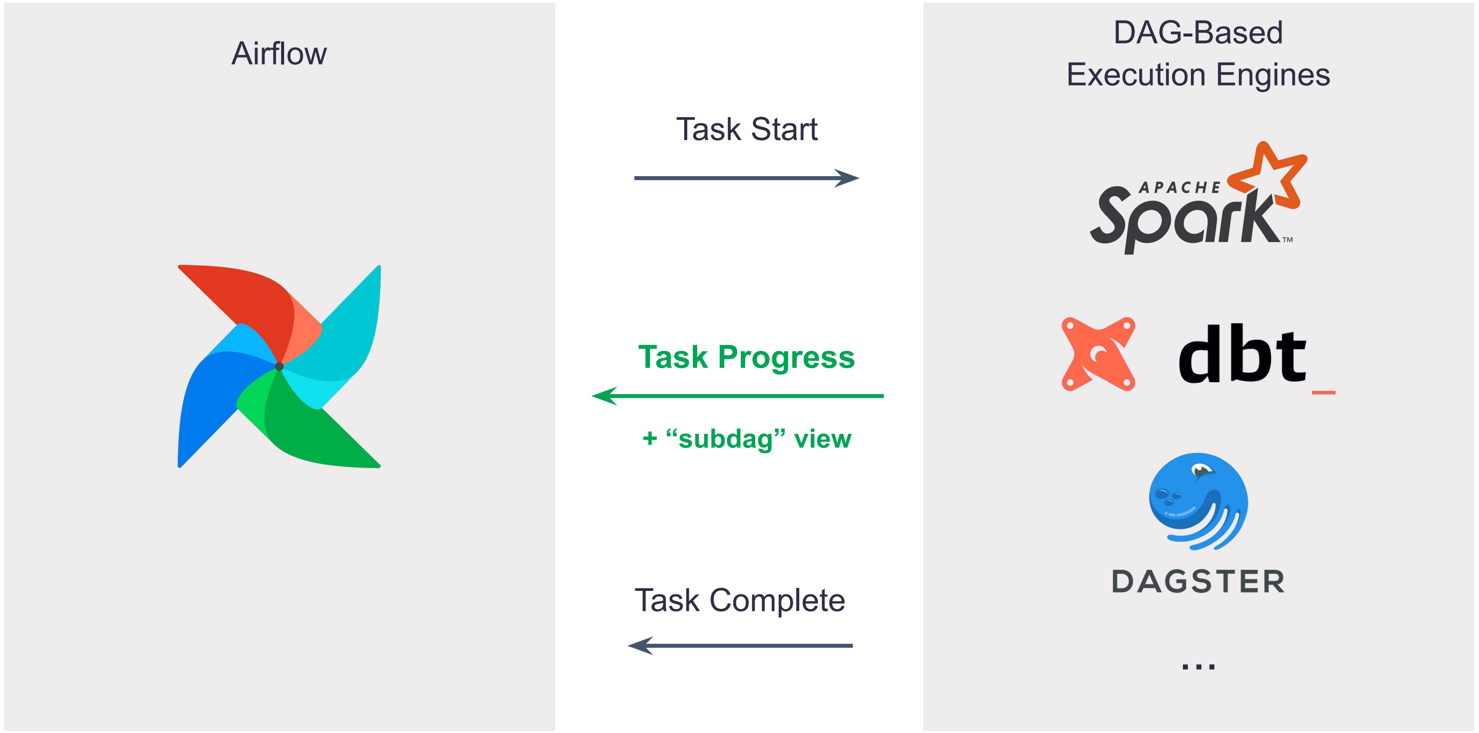
Airflow built-in dashboards based on task metadata

Airflow native statsd exporter offers deeper metrics

Future

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

Task progress API


Airflow
DAG-Based
Execution Engines
Task Start
Task Progress
+ “subdag” view
Task Complete
APACHE Spark
dbt
DAGSTER
...

Thank You!

Now Q&A