

What's new in Airflow 2

Apache Airflow Online Summit
8th of July 2020



Who are we?



Tomek Urbaszek

Committer, **PMC Member**
Software Engineer @ Polidea



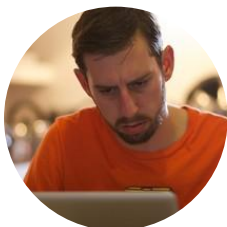
Jarek Potiuk

Committer, **PMC member**
Principal Software Engineer @ Polidea



Kamil Breguła

Committer, **PMC member**
Software Engineer @ Polidea



Ash Berlin-Taylor

Committer, **PMC member**
Airflow Engineering Lead @ Astronomer



Daniel Imberman

Committer, **PMC Member**
Senior Data Engineer @ Astronomer



Kaxil Naik

Committer, **PMC member**
Senior Data Engineer @ Astronomer

Announcements

New PMC members



Tomek Urbaszek
Committer, **PMC Member**
Software Engineer @ Polidea



Daniel Imberman
Committer, **PMC Member**
Senior Data Engineer @ Astronomer



Kamil Breguła
Committer, **PMC member**
Software Engineer @ Polidea

New committer



QP Hou
Committer
Senior Engineer @ Scribd

Talk: Teaching an old DAG new tricks
Friday July 10 th, 5 am UTC

“Ask Me Anything” session with Airflow PMCs

- **Asia friendly time-zone**
- **Thursday 11 pm PDT / Friday 6 am UTC**
- **Hosted by Bangalore Meetup**
- **BYO questions**

High Availability



Scheduler High Availability

Goals:

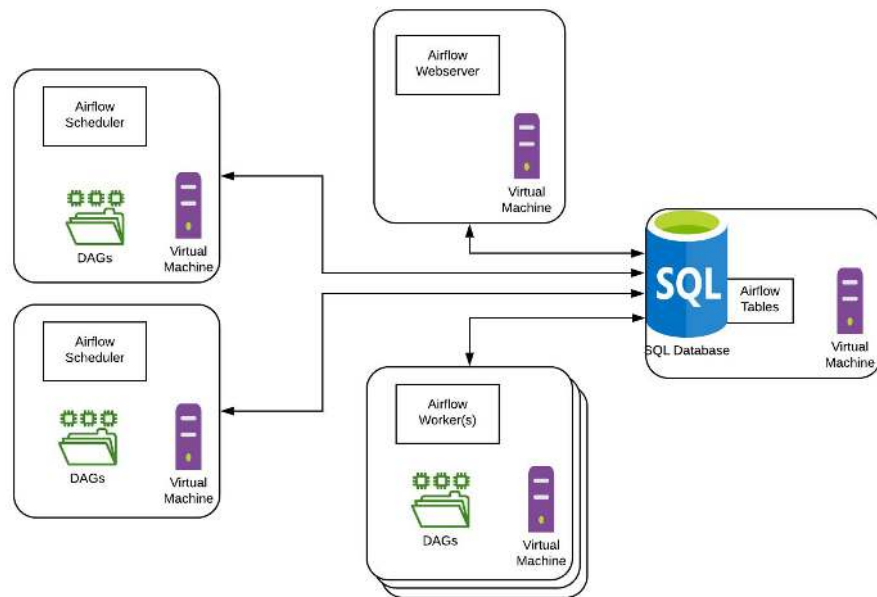
- Performance - reduce task-to-task schedule "lag"
- Scalability - increase task throughput by horizontal scaling
- Resiliency - kill a scheduler and have tasks continue to be scheduled

Scheduler High Availability: Design

- Active-active model. Each scheduler does everything
- Uses existing database - no new components needed, no extra operational burden
- Plan to use row-level-locks in the DB (`SELECT ... FOR UPDATE`)
- Will re-evaluate if performance/stress testing show the need

Example HA configuration

Airflow Schedulers running in High Availability
on virtual machines - example configuration



Scheduler High Availability: Tasks

- Separate DAG parsing from DAG scheduling ✓

This removes the tie between parsing and scheduling that is still present

- Run a mini scheduler *in the worker* after each task is completed ✓

A.K.A. "fast follow". Look at immediate down stream tasks of what just finished and see what we can schedule

- Test it to destruction - In progress

This is a big architectural change, we need to be sure it works well.

Measuring Performance

Key performance we define as "Scheduler lag":

- Amount of "wasted" time not running tasks
- `ti.state_date - max(t.end_date for t in upstream_tis)`
- Zero is the goal (we'll never get to 0.)
- Tasks are "echo true" -- tiny but still executing

Preliminary performance results

Case: 100 DAG files | 1 DAG per file | 10 Tasks per DAG | 1 run per DAG

Workers: 4 | Parallelism: 64

1.10.10: 54.17s (σ 19.38) Total runtime: 22m22s

HA branch - 1 scheduler: 4.39s (σ 1.40) 1m10s

HA branch - 3 schedulers: 1.96s (σ 0.51) Total runtime: 48s

Preliminary performance results

Case: 1 Dag File | 1 Dag Per File | 20 Tasks per DAG | 1000 runs per DAG

Workers: 30 | Parallelism: 40960 | Default pool size 40960

1.10.10: 42.14s (σ 7.06) Total runtime: 1h 30m 14s

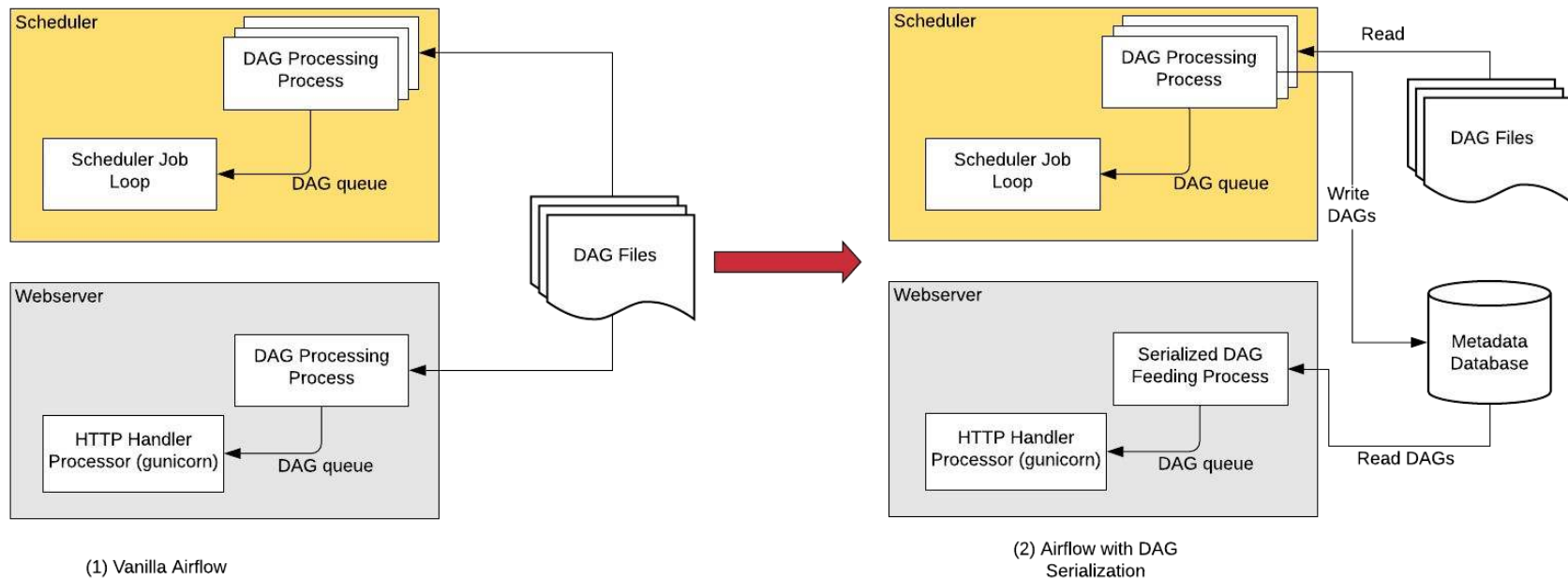
HA branch - 1 scheduler: 0.68s (σ 0.19) Total runtime: 18m 51s

HA branch - 3 schedulers*: 1.54s (σ 1.79) Total runtime: 12m 52s

DAG Serialization



Dag Serialization



Dag Serialization (Tasks Completed)

- **Stateless Webserver:** Scheduler parses the DAG files, serializes them in JSON format & saves them in the Metadata DB.
- **Lazy Loading of DAGs:** Instead of loading an entire DagBag when the Webserver starts we only load each DAG on demand. This helps **reduce Webserver startup time and memory**. This reduction in time is notable with large number of DAGs.
- Deploying new DAGs to Airflow - no longer requires long restarts of webserver (if DAGs are baked in Docker image)
- Feature to use the “JSON” library of choice for Serialization (default is inbuilt ‘json’ library)
- Paves way for **DAG Versioning & Scheduler HA**

Dag Serialization (Tasks In-Progress for Airflow 2.0)

- Decouple DAG Parsing and Serializing from the scheduling loop.
- Scheduler will fetch DAGs from DB
- DAG will be parsed, serialized and saved to DB by a separate component “Serializer”/ “Dag Parser”
- This should reduce the delay in Scheduling tasks when the number of DAGs are large

DAG Versioning



Dag Versioning

Current Problem:

- Change in DAG structure affects viewing previous DagRuns too
- Not possible to view the code associated with a specific DagRun
- Checking logs of a deleted task in the UI is not straight-forward

Dag Versioning (Current Problem)

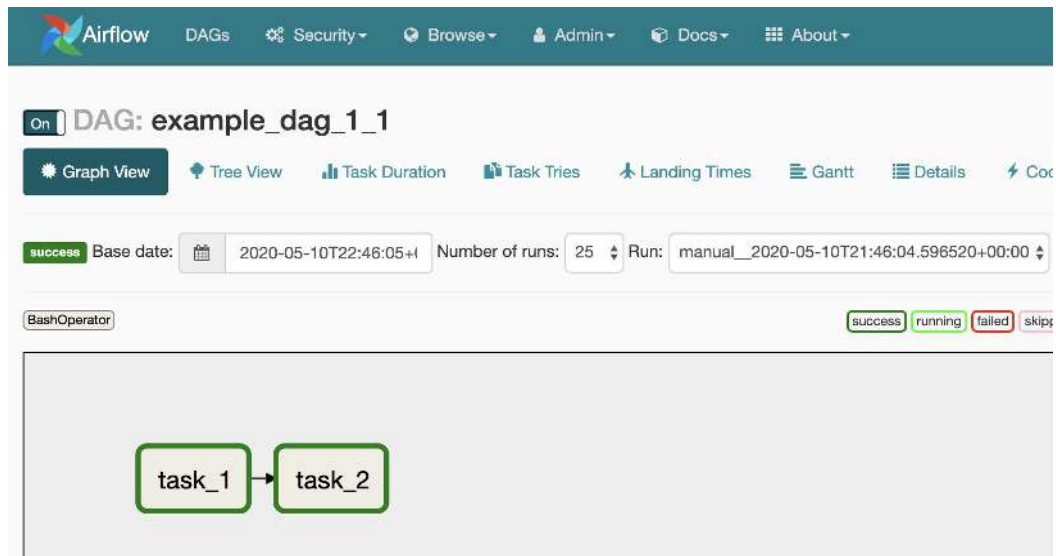
```
from airflow.models.dag import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime

with DAG('example_dag_1_1', schedule_interval=None,
        start_date=datetime(2020, 4, 25)) as example_dag_1_1:

    task_1 = BashOperator(
        task_id='task_1',
        bash_command='echo hello',
    )

    task_2 = BashOperator(
        task_id='task_2',
        bash_command='echo hello',
    )

    task_1 >> task_2
```



Dag Versioning (Current Problem)

```
from airflow.models.dag import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime

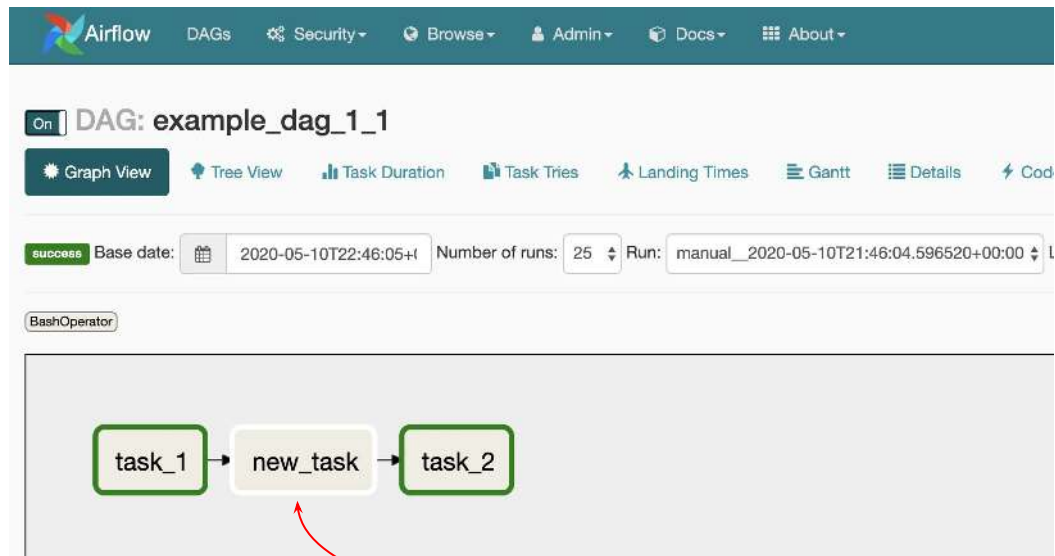
with DAG('example_dag_1_1', schedule_interval=None,
        start_date=datetime(2020, 4, 25)) as example_dag_1_1:

    task_1 = BashOperator(
        task_id='task_1',
        bash_command='echo hello',
    )

    new_task = BashOperator(
        task_id='new_task',
        bash_command='echo hello',
    )

    task_2 = BashOperator(
        task_id='task_2',
        bash_command='echo hello',
    )

    task_1 >> new_task >> task_2
```



New task is shown in Graph View for older DAG Runs too with “no status”.

Dag Versioning

Current Problem:

- Change in DAG structure affects viewing previous DagRuns too
- Not possible to view the code associated with a specific DagRun
- Checking logs of a deleted task in the UI is not straight-forward

Goal:

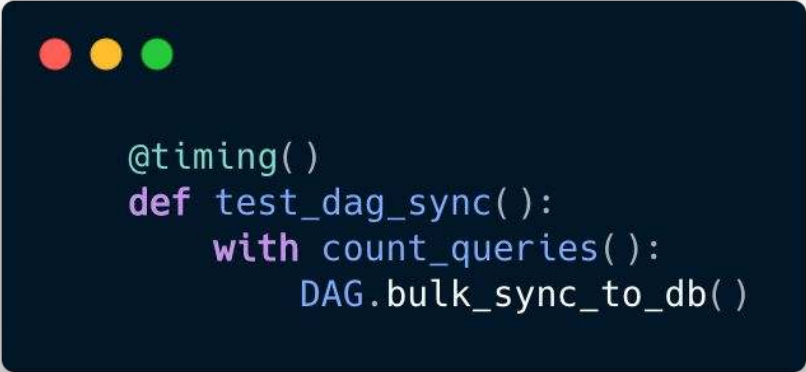
- Support for storing multiple versions of Serialized DAGs
- Baked-In Maintenance DAGs to cleanup old DagRuns & associated Serialized DAGs
- Graph View shows the DAG associated with that DagRun

Performance Improvements



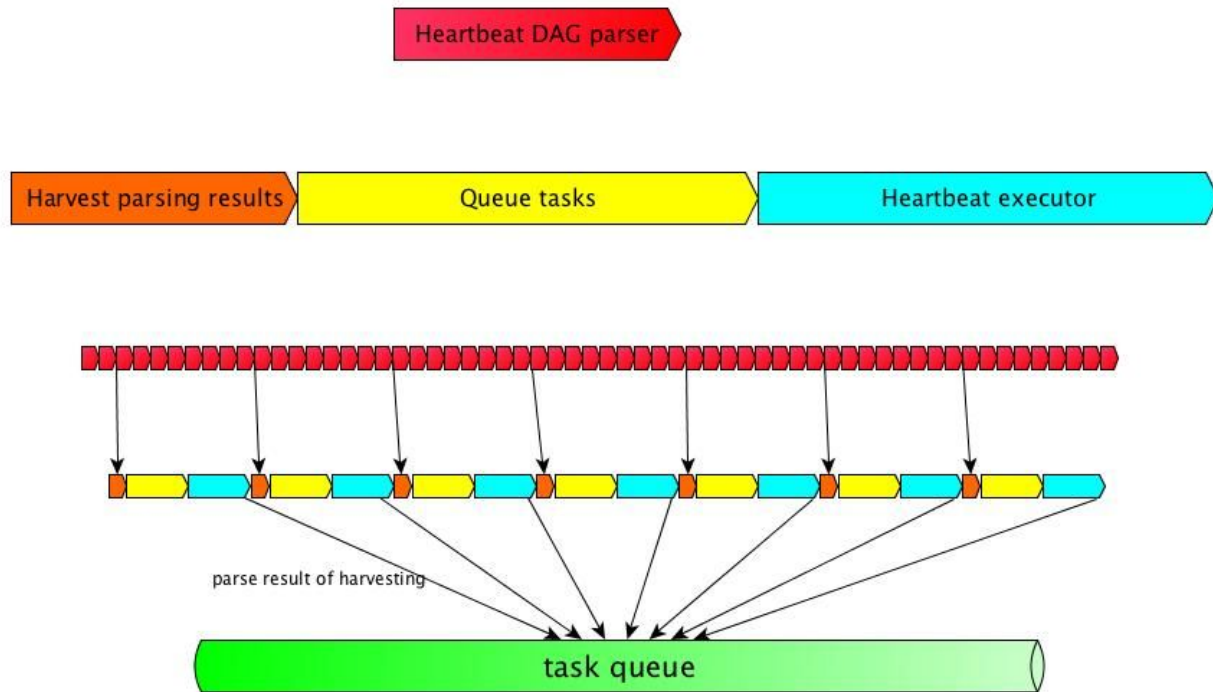
Components performance improvements

- Focus on the current code
 - Reviews each components in turn
- Tools supporting performance tests - *perf_kit*



```
@timing()  
def test_dag_sync():  
    with count_queries():  
        DAG.bulk_sync_to_db()
```

Avoid loading DAGs in the main scheduler loop



Limit queries count

DagFileProcessor:

When we have one DAG file with 200 DAGs, each DAG with 5 tasks:

	Before	After	Diff
Average time:	8080.246 ms	628.801 ms	-7452 ms (92%)
Queries count:	2692	5	-2687 (99%)

Celery Executor:

When we have one DAG file with 200 DAGs, each DAG with 5 tasks:

	Postgres		Redis	
	Before	After	Before	After
Average time	3.1 s	27.825 ms	778.557 ms	3.417 ms
Queries count	5000	1	5000	1

How to avoid regression?



```
with assert_queries_count(3):  
    DAG.bulk_sync_to_db(dags)
```

REST API



API: follows Open API 3.0 specification

The screenshot displays the Swagger Editor interface. On the left, the OpenAPI 3.0 specification is shown in a code editor. The specification includes a title, description, version, license, servers, paths, and components. The paths section defines several endpoints for managing connections and DAGs. The components section defines the data structures used in the API.

```
1 # Licensed to the Apache Software Foundation (ASF) under one
2 # or more contributor license agreements. See the NOTICE file
3 # distributed with this work for additional information
4 # regarding copyright ownership. The ASF licenses this file
5 # to you under the Apache License, Version 2.0 (the
6 # "License"); you may not use this file except in compliance
7 # with the License. You may obtain a copy of the License at
8 #
9 # http://www.apache.org/licenses/LICENSE-2.0
10 #
11 # Unless required by applicable law or agreed to in writing,
12 # software distributed under the License is distributed on an
13 # "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
14 # KIND, either express or implied. See the License for the
15 # specific language governing permissions and limitations
16 # under the License.
17 ---
18 openapi: 3.0.3
19
20 info:
21   title: "Airflow API (Stable)"
22   description: Apache Airflow management API.
23   version: 1.0.0
24   license:
25     name: Apache 2.0
26     url: http://www.apache.org/licenses/LICENSE-2.0.html
27   contact:
28     name: Apache Foundation
29     url: https://airflow.apache.org
30     email: dev@airflow.apache.org
31
32 servers:
33   - url: /api/v1
34     description: Airflow Stable API.
35
36 paths:
37   # Database entities
38   /connections:
39     get:
40       summary: Get all connection entries
41       operationId: getConnections
42       tags: [Connection]
43       parameters:
44         - $ref: '#/components/parameters/PageLimit'
45         - $ref: '#/components/parameters/PageOffset'
46       responses:
47         200:
48           description: List of connection entry.
49           content:
50             application/json:
51               schema:
52                 allOf:
53                   - $ref: '#/components/schemas/ConnectionCollection'
54                   - $ref: '#/components/schemas/CollectionInfo'
55       'x-api':
56         $ref: '#/components/responses/Unauthenticated'
57       'x-api':
58         $ref: '#/components/responses/PermissionDenied'
59
60   post:
61     summary: Create connection entry
62     operationId: createConnection
63     tags: [Connection]
64     requestBody:
65       required: true
66       content:
67         application/json:
68           schema:
69             $ref: '#/components/schemas/Connection'
70     responses:
71       200:
72         description: Successful response.
73       content:
74         application/json:
75           schema:
76             $ref: '#/components/schemas/CollectionInfo'
```

On the right, the configuration and connection endpoints are listed:

- Config**
 - GET /config Get current configuration
- Connection**
 - GET /connections Get all connection entries
 - POST /connections Create connection entry
 - GET /connections/{connection_id} Get a connection entry
 - PATCH /connections/{connection_id} Update a connection entry
 - DELETE /connections/{connection_id} Delete a connection entry
- DAG**
 - GET /dags Get all DAGs
 - GET /dags/{dag_id} Get basic information about a DAG
 - PATCH /dags/{dag_id} Update a DAG
 - POST /dags/{dag_id}/clearTaskInstances Clears a set of task instances associated with the DAG for a specified date range.
 - GET /dags/{dag_id}/structure Get simplified representation of DAG.
 - GET /dags/{dag_id}/tasks Get tasks for DAG
 - GET /dags/{dag_id}/tasks/{task_id} Get simplified representation of a task.
 - GET /dagSources/{file_token} Get source code using file token

Outreachy interns



Ephraim Anierobi



Omair Khan

API development progress

AIP-32 - Airflow REST API

🕒 Updated 5 days ago

	API Endpoints - Read - Connection #8127	Done
	API Endpoints - Read - DAG Model #8128	Community review
	API Endpoints - Read - DAG Runs #8129	Done
	API Endpoints - Read - Task Instance #8132	Development in progress
	API Endpoints - Read - Variable #8133	Done
	API Endpoints - Read - XCOM #8134	Done
	API Endpoint - Dag source #8137	Community review
	API Endpoint - Dags structure/Task #8138	Done
Community tasks		
High level info #8107		
Basic OpenAPI spec #8108		Done
Basic integration Airflow and connexion #8109		Done
API Endpoints #8118	API Endpoints - CRUD - Connection #8127	Done
	API Endpoints - CRUD - DAG Model #8128	Blocked
	API Endpoints - CRUD - DAG Runs #8129	Development in progress
	API Endpoints - CRUD - Import errors #8130	Done
	API Endpoints - CRUD - Pools #8131	Done
	API Endpoints - CRUD - Task Instance #8132	Blocked
	API Endpoints - CRUD - Variable #8133	Done
	API Endpoints - CRUD - XCOM #8134	Development in progress
	API Endpoint - Logs #8135	Done
	API Endpoint - Config #8136	Done
	API Endpoint - Dags structure/Task #8138	Done
	API Endpoint - Extra Links #8140	Done
		Next up
HATEOS for API #8117		Next up
CRUD Framework for API #8116		Next up
Authorization and Permissions #8112		Next up
Authentication in API #8111		Next up
Custom WEB UI screen to control permissions #8124		Blocked
Docs for REST API #8143		Research in progress
API security tests #8113		Blocked

Dev/CI environment



CI environment

- Moved to GitHub Actions
 - Kubernetes Tests ✓
 - Easier way to test Kubernetes Tests locally ✓
- Quarantined tests
 - Fixing the Quarantined tests ✓
- Thinning CI image
 - Moved integrations out of the image ✓
- Future: Automated System Tests (AIP-21)

Dev environment

- Breeze

- unit testing ✓
- package building ✓
- release preparation ✓
- kubernetes tests ✓
- refreshed videos ✓

- Code Spaces / VSCode

```
Usage: breeze [FLAGS] [COMMAND] -- <EXTRA_ARGS>
```

By default the script enters IT environment and drops you to bash shell, but you can choose one of the commands to run specific actions instead. Add --help after each command to see details:

Commands without arguments:

shell	[Default] Enters interactive shell in the container
build-docs	Builds documentation in the container
build-image	Builds CI or Production docker image
cleanup-image	Cleans up the container image created
exec	Execs into running breeze container in new terminal
generate-requirements	Generates pinned requirements for pip dependencies
push-image	Pushes images to registry
initialize-local-virtualenv	Initializes local virtualenv
setup-autocomplete	Sets up autocomplete for breeze
stop	Stops the docker-compose environment
restart	Stops the docker-compose environment including DB cleanup
toggle-suppress-cheatsheet	Toggles on/off cheatsheet
toggle-suppress-asciart	Toggles on/off asciart

Commands with arguments:

docker-compose	<ARG>	Executes specified docker-compose command
kind-cluster	<ARG>	Manages Kind cluster on the host
prepare-backport-readme	<ARG>	Prepares backport packages readme files
prepare-backport-packages	<ARG>	Prepares backport packages
static-check	<ARG>	Performs selected static check for changed files
tests	<ARG>	Runs selected tests in the container

Help commands:

flags	Shows all breeze's flags
help	Shows this help message
help-all	Shows detailed help for all commands and flags

Backport Packages ✓

- Bring Airflow 2.0 providers to 1.10.* ✓
- Packages per-provider ✓
- 58 packages (!) ✓
- Python 3.6+ only(!) ✓
- Automatically tested on CI ✓
- Future
 - Automated System Tests (AIP-4)
 - Split Airflow (AIP-8)?

Talk: Migration to Airflow backport providers, Anita Fronczak

Thursday July 16th, 4 am UTC

✓ Prepare & test backport packages

```
1880 -----
1881 Prepared backporting package jdbc
1882 =====
1883 Preparing backporting package jenkins
1884 -----
1885 Prepared backporting package jenkins
1886 =====
1887 Preparing backporting package jira
1888 -----
1889 Prepared backporting package jira
1890 =====
1891 Preparing backporting package microsoft.azure
1892 -----
1893 Prepared backporting package microsoft.azure
1894 =====
1895 Preparing backporting package microsoft.mssql
1896 -----
1897 Prepared backporting package microsoft.mssql
1898 =====
1899 Preparing backporting package microsoft.winrm
1900 -----
1901 Prepared backporting package microsoft.winrm
1902 =====
1903 Preparing backporting package msopencl
```

✓ Prepare & test backport packages

```
2499 =====
2500 Installing apache-airflow-backport-providers-microsoft-mssql
2501 -----
2502 Installed apache-airflow-backport-providers-microsoft-mssql
2503 -----
2504 Uninstalling apache-airflow-backport-providers-microsoft-mssql
2505 -----
2506 Uninstalled apache-airflow-backport-providers-microsoft-mssql
2507 -----
2508 Airflow version after installation 1.10.10
2509 =====
2510 Installing apache-airflow-backport-providers-microsoft-winrm
2511 -----
2512 Installed apache-airflow-backport-providers-microsoft-winrm
2513 -----
2514 Uninstalling apache-airflow-backport-providers-microsoft-winrm
2515 -----
2516 Uninstalled apache-airflow-backport-providers-microsoft-winrm
2517 -----
2518 Airflow version after installation 1.10.10
2519 =====
```

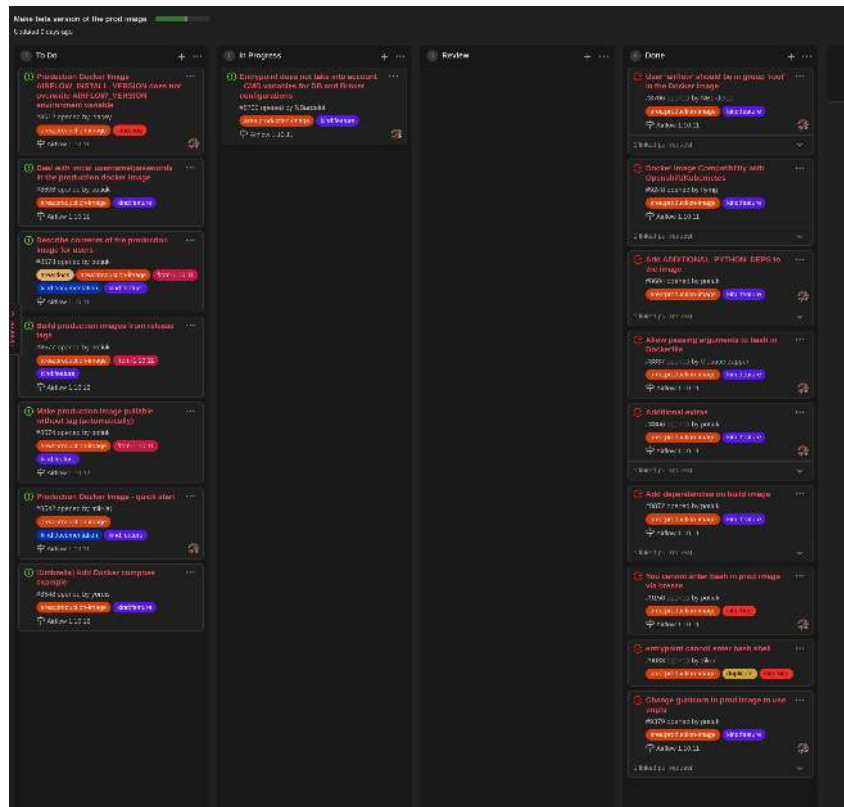
Support for Production Deployments



Production Image

- Beta quality image is nearly ready ✓
- Started with “bare image” ✓
- Listened to use cases from users ✓
- Integration with Helm Chart ✓
- Implemented feedback ✓
- Docker Compose

Talk, Production Docker image for Apache Airflow
Jarek Potiuk, Tuesday July 14th, 5 am UTC



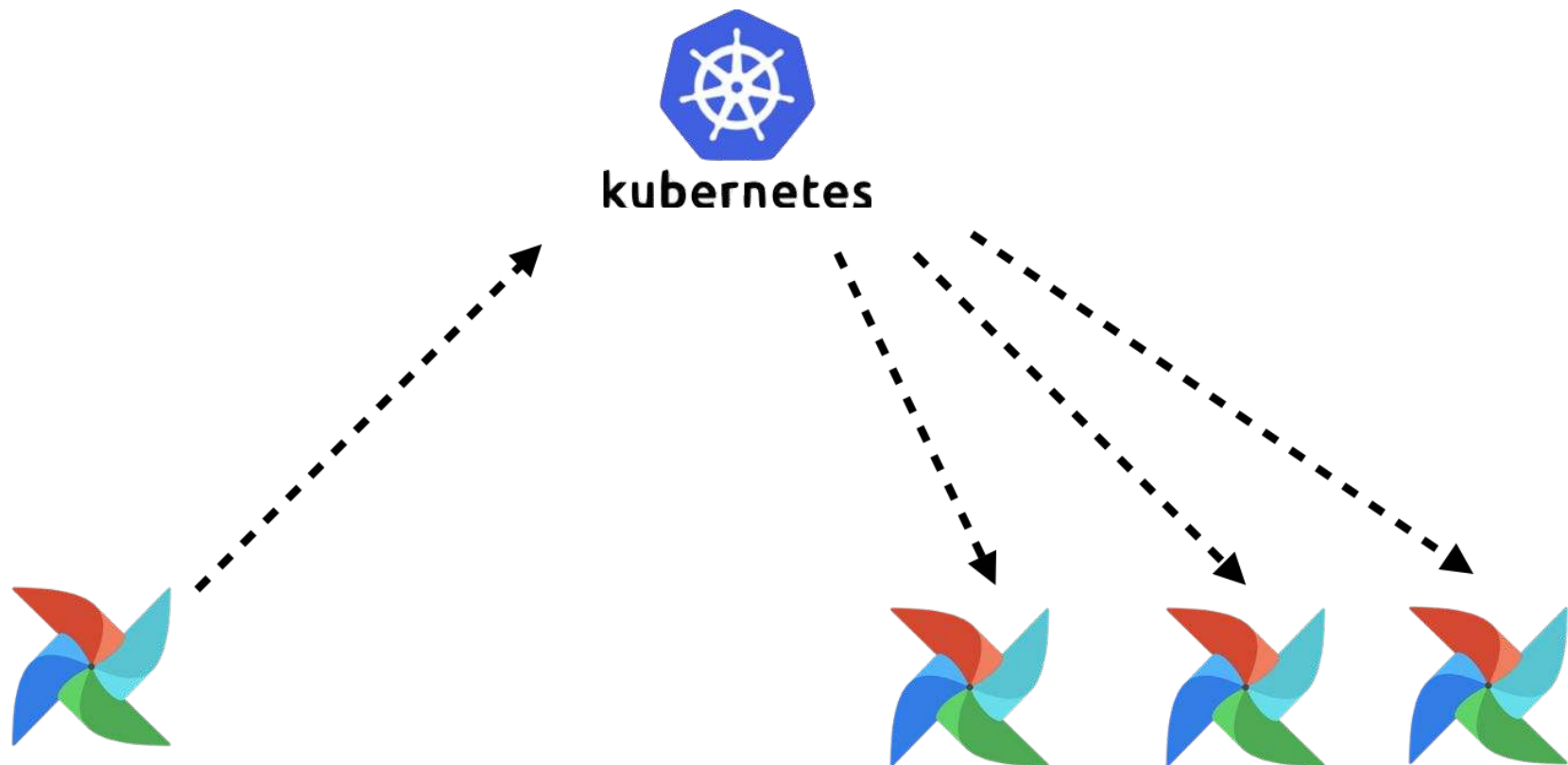
What's new in Airflow + Kubernetes



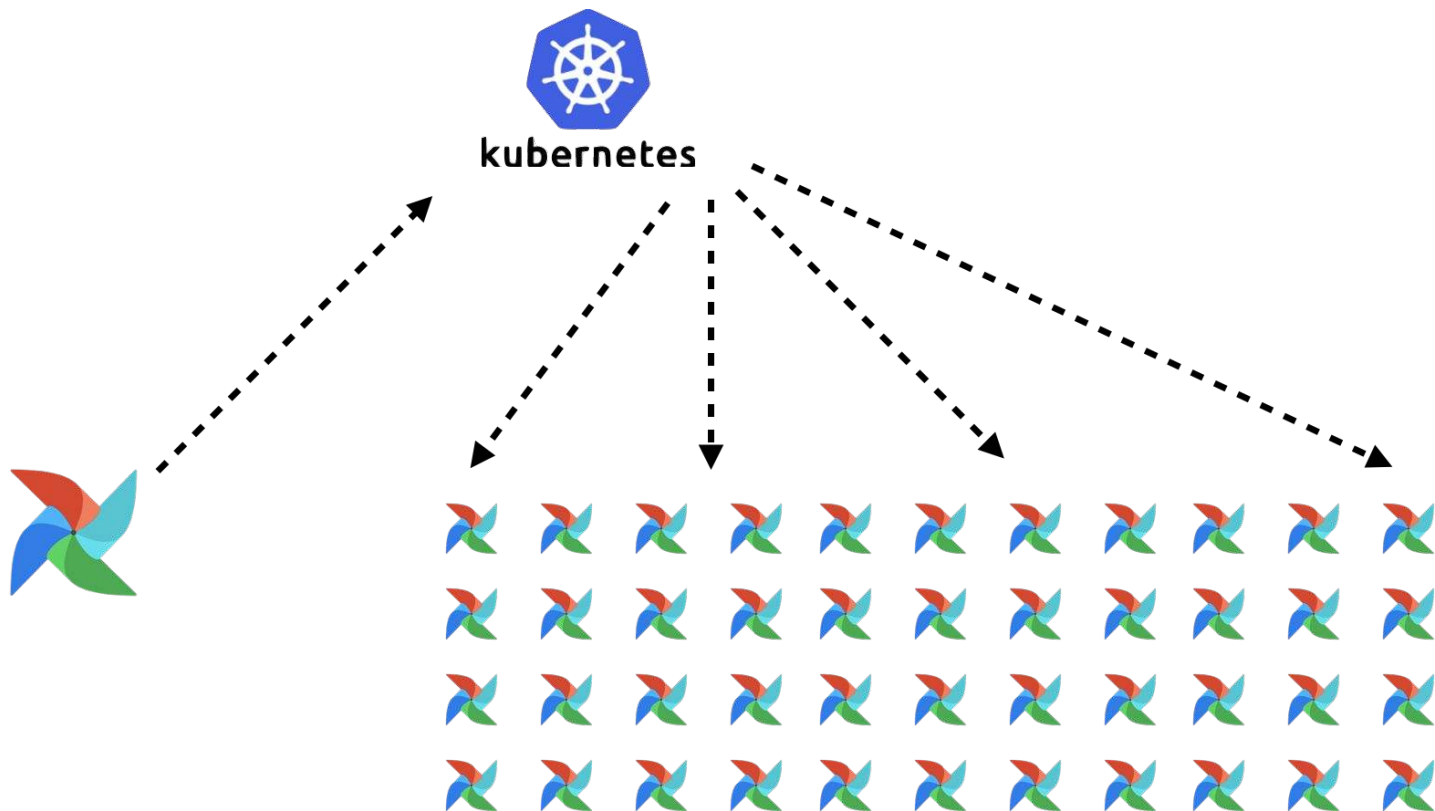
KEDA Autoscaling



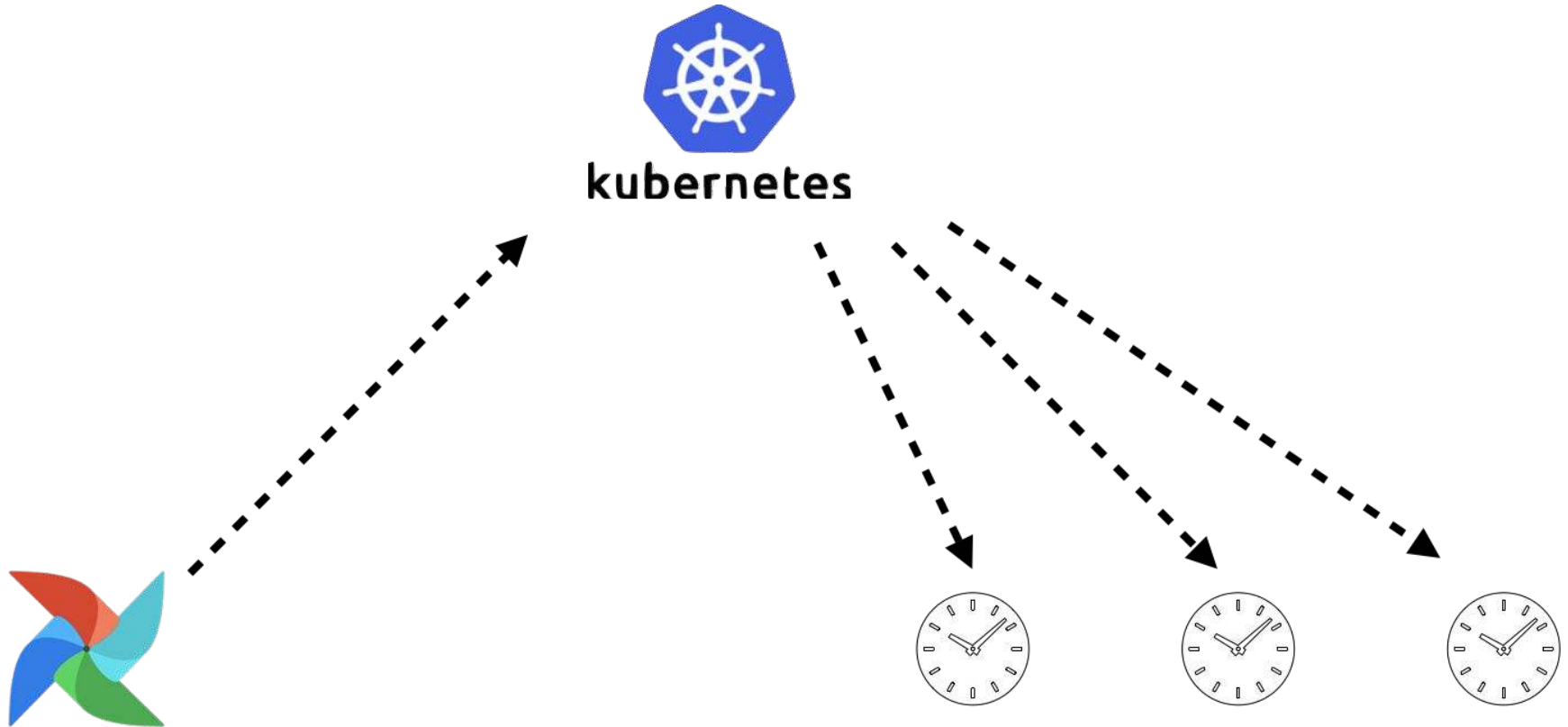
KubernetesExecutor



KubernetesExecutor



KubernetesExecutor



KubernetesExecutor vs. CeleryExecutor

KubernetesExecutor

- Dynamic Allocation
- executor_config

CeleryExecutor

- Immediate SLAs
- Multiple tasks per-worker

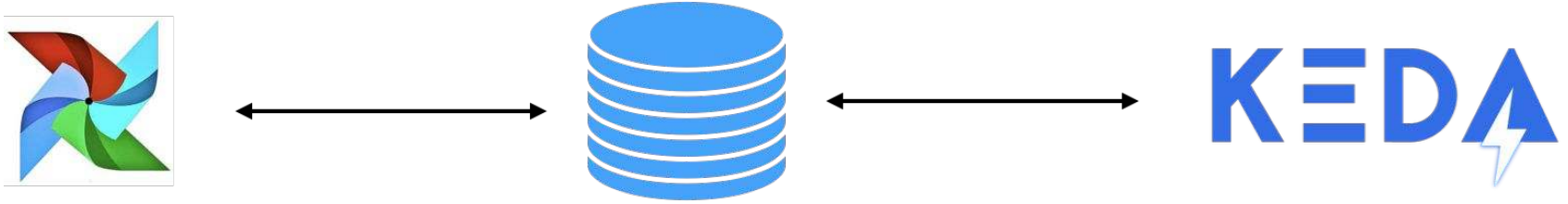
KEDA

The logo for KEDA (Kubernetes Event-driven Autoscaling) features the word "KEDA" in a bold, blue, sans-serif font. A yellow lightning bolt with a blue outline is positioned behind the letter "A", pointing downwards and to the left.

KEDA Autoscaling

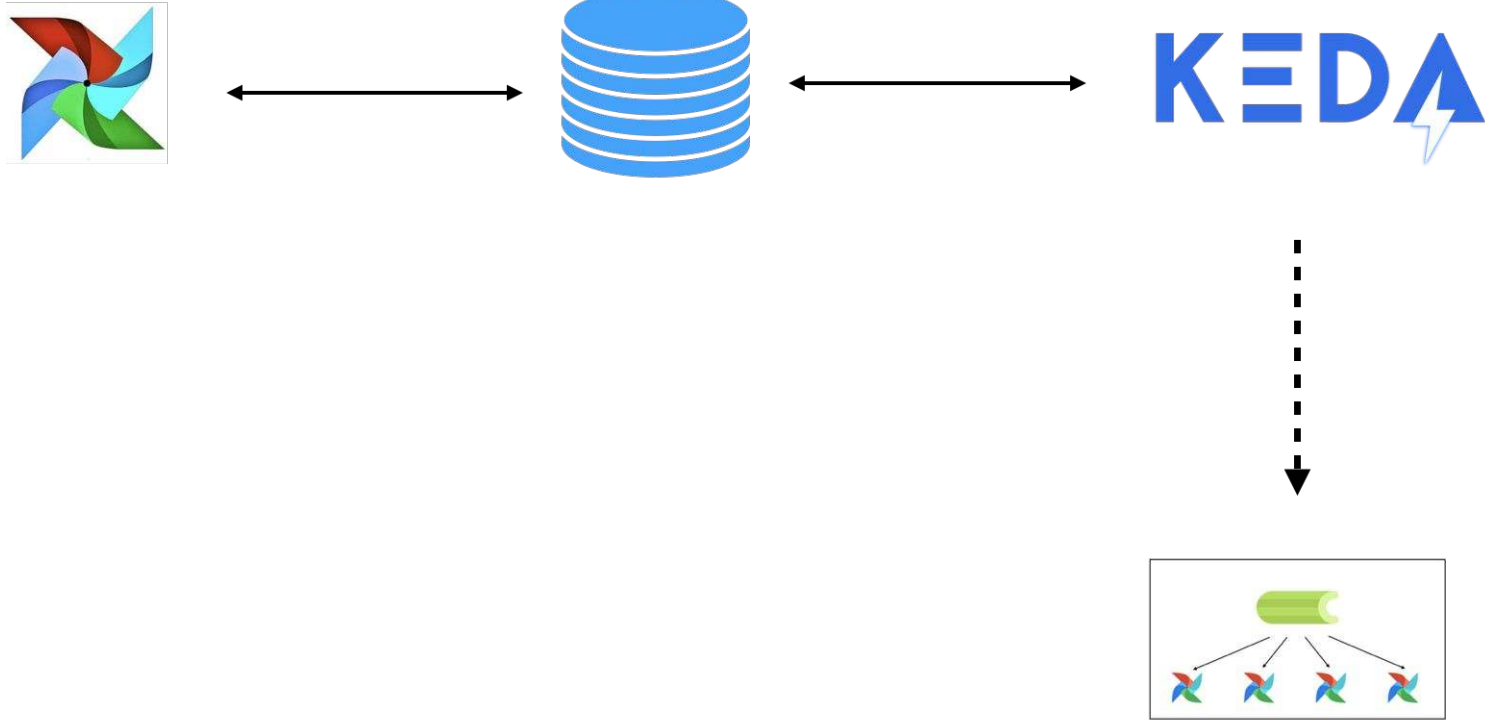
- Kubernetes Event-driven Autoscaler
- Scales based on # of RUNNING and QUEUED tasks in PostgreSQL backend

KEDA Autoscaling



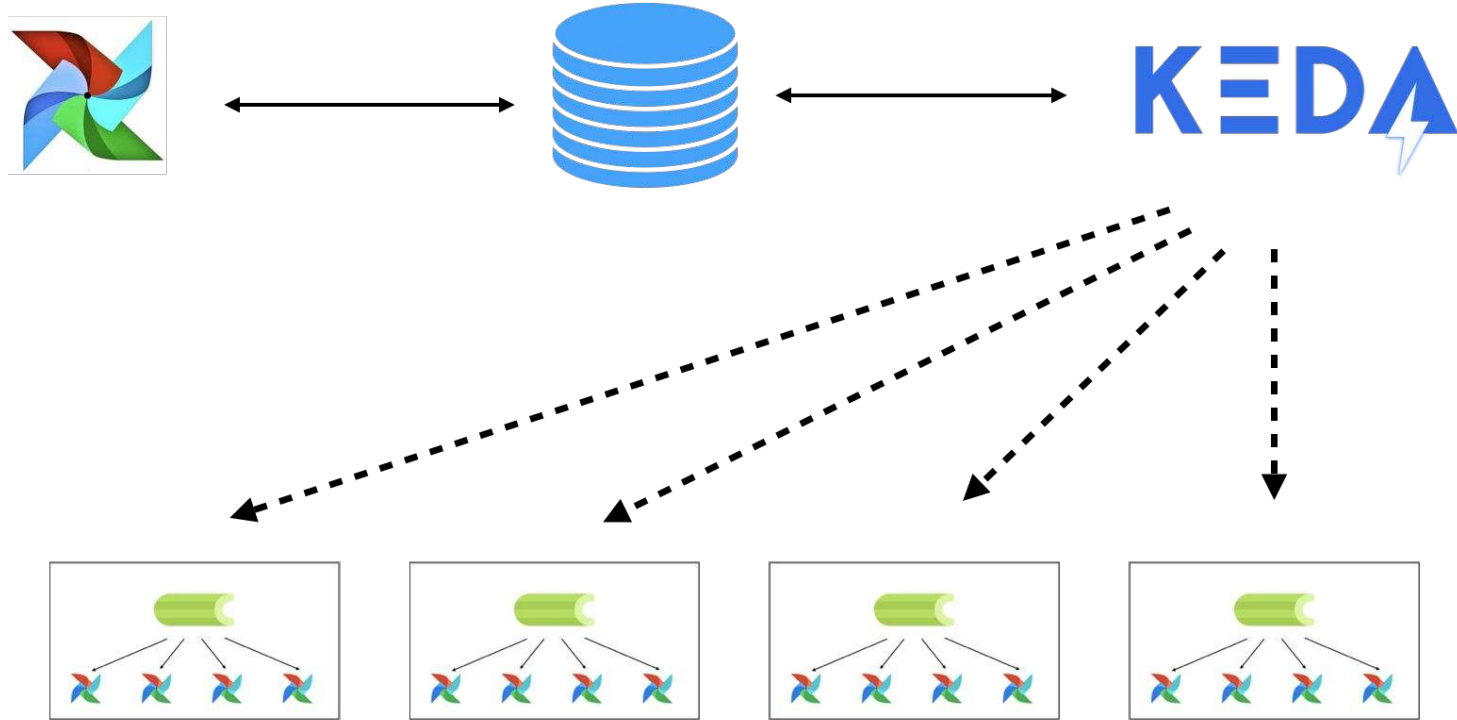
$\text{CEIL}((0 \text{ RUNNING} + 0 \text{ QUEUED})/16) = 0 \text{ workers}$

KEDA Autoscaling



$\text{CEIL}((0 \text{ RUNNING} + 1 \text{ QUEUED})/16) = 1 \text{ workers}$

KEDA Autoscaling



$$\text{CEIL}((20 \text{ RUNNING} + 20 \text{ QUEUED})/16) = 4 \text{ workers}$$

KEDA Queues

- Historically Queues were expensive and hard to allocate
- With KEDA, queues are free! (can have 100 queues)
- KEDA works with k8s deployments so any customization you can make in a k8s pod, you can make in a k8s queue (worker size, GPU, secrets, etc.)

KubernetesExecutor Pod Templating from YAML/JSON



KubernetesExecutor Pod Templating

- In the K8sExecutor currently, users can modify certain parts of the pod, but many features of the k8s API are abstracted away
- We did this because at the time the airflow community was not well acquainted with the k8s API
- We want to enable users to modify their worker pods to better match their use-cases

KubernetesExecutor Pod Templating

- Users can now set the `pod_template_file` config in their `airflow.cfg`
- Given a path, the KubernetesExecutor will now parse the yaml file when launching a worker pod
- Huge thank you to @davlum for this feature

Official Airflow Helm Chart



Helm Chart

- Donated by astronomer.io.
- This is the official helm chart that we have used both in our enterprise and in our cloud offerings (thousands of deployments of varying sizes)
- Helm 3 compliant
- Users can turn on KEDA autoscaling through helm variables
- “helm install apache/airflow”

Helm Chart

- Chart will cut new releases with each airflow release
- Will be tested on official docker image
- Significantly simplifies airflow onboarding process for Kubernetes users

Functional DAGs



Functional DAGs

```
def get_cat_pictures(num: int) -> List[Dict]:
    response = requests.get("https://cat_pictures.com", params={"num": num})
    return response.json()["cats"]

def save_cats(list_of_cats: List[Dict]) -> None:
    for cat in list_of_cats:
        save_it_somewhat(cat)

with DAG("cat_fetcher"):
    get_task = PythonOperator(
        task_id="get_task", python_callable=get_cat_pictures, op_args=[42]
    )
    cats = "{{ task_instance.xcom_pull('get_task') }}"
    save_task = PythonOperator(
        task_id="save_task", python_callable=save_cats, op_args=[cats]
    )
    get_task >> save_task
```

- PythonOperator boilerplate code
- Define separately:
 - ◆ order relation
 - ◆ data relation
- Writing jinja strings by hand

Functional DAGs

```
def get_cat_pictures(num: int) -> List[Dict]:
    response = requests.get("https://cat_pictures.com", params={"num": num})
    return response.json()["cats"]

def save_cats(list_of_cats: List[Dict]) -> None:
    for cat in list_of_cats:
        save_it_somewhat(cat)

with DAG("cat_fetcher"):
    get_task = PythonOperator(
        task_id="get_task", python_callable=get_cat_pictures, op_args=[42]
    )
    cats = "{{ task_instance.xcom_pull('get_task') }}"
    save_task = PythonOperator(
        task_id="save_task", python_callable=save_cats, op_args=[cats]
    )
    get_task >> save_task
```

```
@task
def get_cat_pictures(num: int) -> List[Dict]:
    response = requests.get("https://cat_pictures.com", params={"num": num})
    return response.json()["cats"]

@task
def save_cats(list_of_cats: List[Dict]) -> None:
    for cat in list_of_cats:
        save_it_somewhat(cat)

with DAG("cat_fetcher"):
    get_task = get_cat_pictures(42)
    save_task = save_cats(get_task)
```

Data and order relationship are same!

And works for all operators

Functional DAGs

Data and order relationship are same!

And works for all operators

AIP-31: Airflow functional DAG definition

- Easy way to convert a function to an operator
- Simplified way of writing DAGs
- Pluggable XCom Storage engine

Find out more:

[AIP-31: Airflow functional DAG definition](#)

by Gerard Casas Saez

10th of July

```
@task
def get_cat_pictures(num: int) -> List[Dict]:
    response = requests.get("https://cat_pictures.com", params={"num": num})
    return response.json()["cats"]

@task
def save_cats(list_of_cats: List[Dict]) -> None:
    for cat in list_of_cats:
        save_it_somewhat(cat)

with DAG("cat_fetcher"):
    get_task = get_cat_pictures(42)
    save_task = save_cats(get_task)
```

Example: store and retrieve DataFrames on GCS or S3 buckets without boilerplate code

Smaller changes



Other changes of note

- Connection IDs now need to be unique ([#8608](#))

It was often confusing, and there are better ways to do load balancing

- Python 3 only ✓

Python 2.7 unsupported upstream since Jan 1, 2020

- "RBAC" UI is now the only UI ✓

Was a config option before, now only option. Charts/data profiling removed due to security risks

Road to Airflow 2.0



When will Airflow 2.0 be available?



Airflow 2.0 – deprecate, but (try) not to remove

- Breaking changes should be avoided where we can – if upgrade is too difficult users will be left behind
- Release "backport providers" to make new code layout available "now":

```
pip install apache-airflow-backport-providers-aws \
            apache-airflow-backport-providers-google
```

- Before 2.0 we want to make sure we've fixed everything we want to remove or break.

How to upgrade to 2.0 safely

- Install the latest 1.10 release
- Run `airflow upgrade-check` (doesn't exist, yet [#8765](#))
- Fix any warnings
- Upgrade Airflow

Thank you!

Time for Q & A

