
Intro to Apache Airflow

CCOSS 2019 - Jakob Homan & Temo Ojeda & Aizhamal Numamat kyzy

The Plan

Agenda

- Become familiar with Airflow and its architecture
(this talk - 60 min - Jakob)
 - Break *(11:30)*
 - Learn about Apache (ASF) and how to contribute it
(12:00 - 30 min - Jakob)
 - Stand up a local Airflow for testing and development
(12:20 - 30 min - Temo)
 - Lunch *(13:00 - 60 min)*
 - Create and submit your first patch to Airflow
(15:30 - Temo and Jakob and Aizhamal)
-

Who we are



Jakob Homan

Apache Airflow committer, PMC Member

Data @ Lyft



Temo Ojeda

Apache Airflow contributor

Data @ Lyft

Opportunities to contribute



Low bar to contribution, good for new users



Medium effort, a significant contribution



Big architectural changes that drive Airflow's future

Shout out questions, comments



Creative commons: https://commons.wikimedia.org/wiki/File:Arti_Hands_up.jpg

This talk's goal

- Clear understanding of what Airflow is (and is not)
 - Case study of a common Airflow use case
 - Medium dive into how an Airflow clusters
 - Identify lots of areas for future Airflow improvement
 - Not deep diving into all of Airflow's features:
 - Missing out: xcoms, cli, lots of Operator configs, backfills, etc.
-

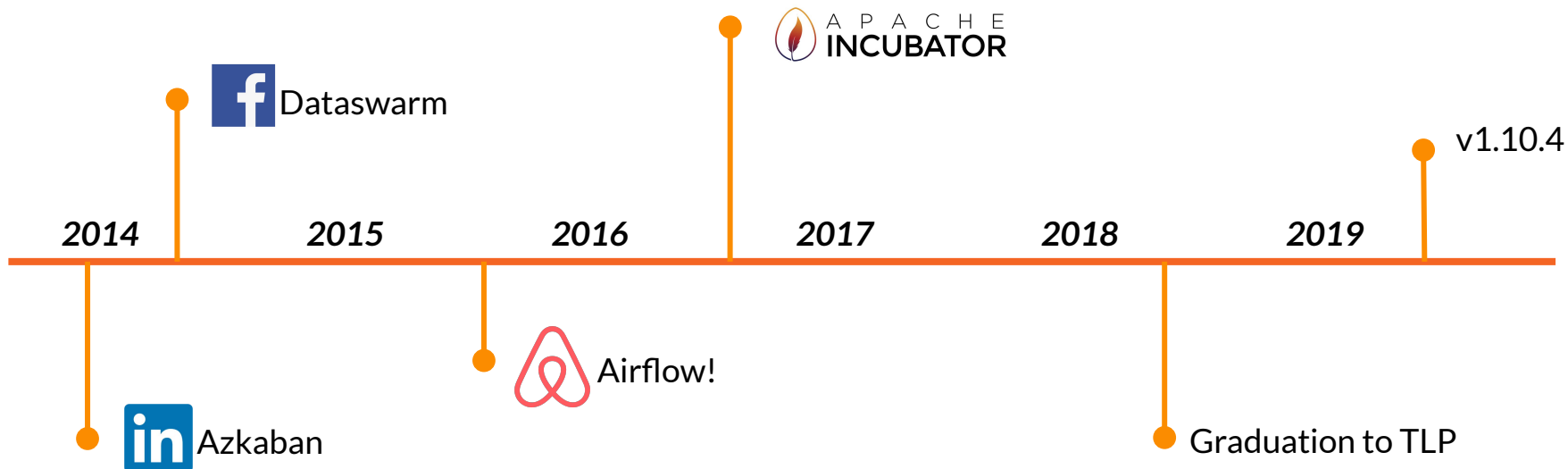
Airflow's backstory

So, what is Airflow anyway?

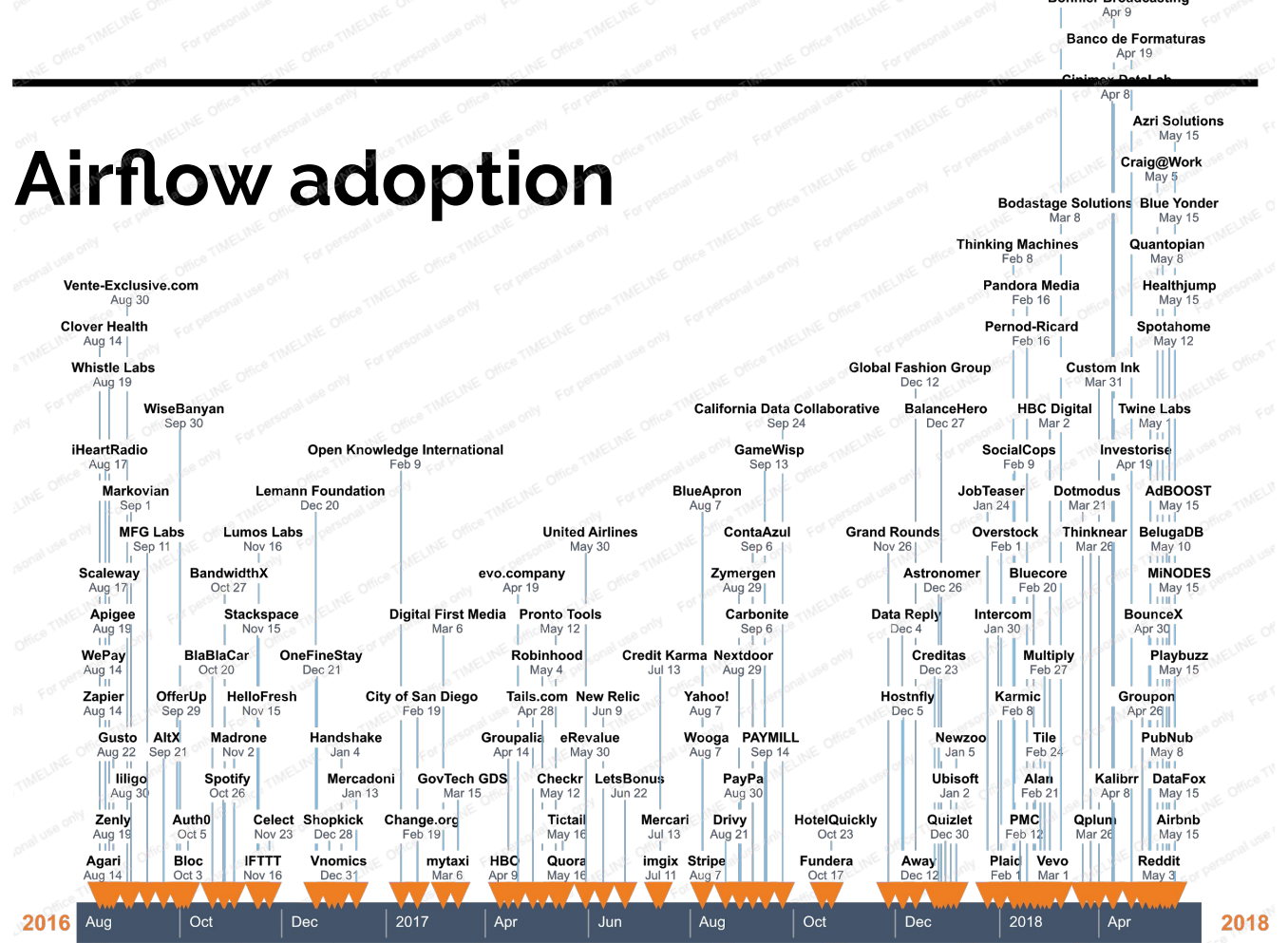


Apache Airflow is a
super-duper
ultra-mega
totally awesome
workflow scheduler

The timeline



Airflow adoption



Add your org!

2016

Aug

Oct

Dec

2017

Apr

Jun

Aug

Oct

Dec

2018

Apr

2018

Very active project

6,985 commits

8 branches

121 releases

922 contributors

Apache-2.0

September 3, 2019 – September 10, 2019

Period: 1 week

Overview

103 Active Pull Requests

0 Active Issues

74

Merged Pull Requests

29

Proposed Pull Requests

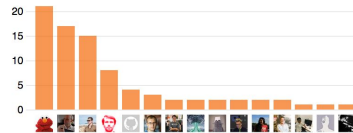
0

Closed Issues

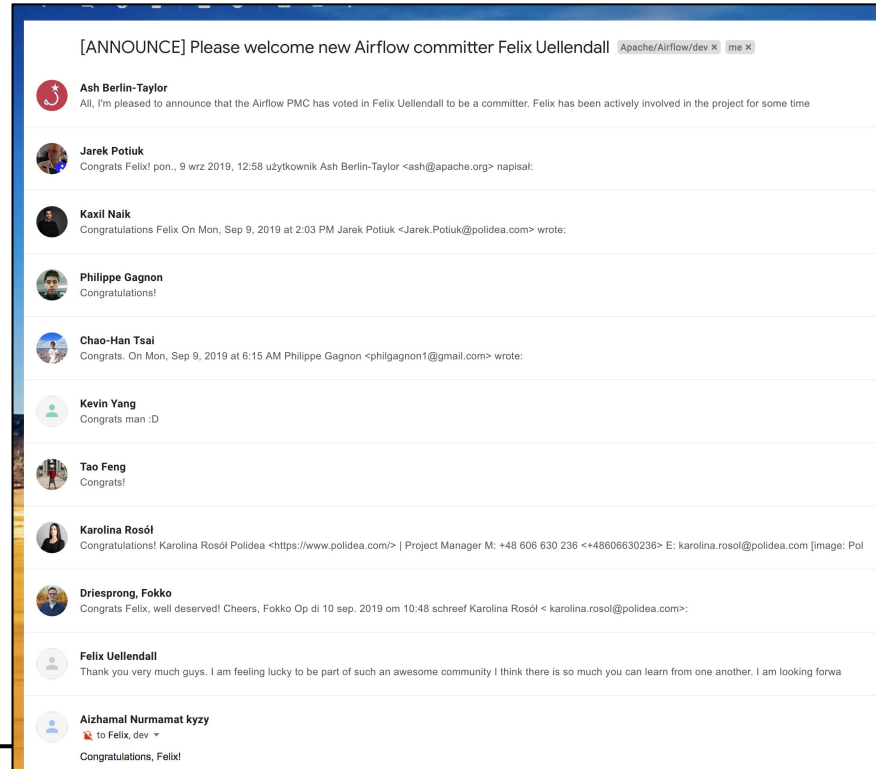
0

New Issues

Excluding merges, **30 authors** have pushed **78 commits** to master and **98 commits** to all branches. On master, **267 files** have changed and there have been **14,271 additions** and **6,250 deletions**.



And very welcoming too...



[ANNOUNCE] Please welcome new Airflow committer Felix Uellendall [Apache/Airflow/dev](#) [me](#) [x](#)

Ash Berlin-Taylor
All, I'm pleased to announce that the Airflow PMC has voted in Felix Uellendall to be a committer. Felix has been actively involved in the project for some time

Jarek Potiuk
Congrats Felix! pon., 9 wrz 2019, 12:58 uzytkownik Ash Berlin-Taylor <ash@apache.org> napisal:

Kaxil Naik
Congratulations Felix On Mon, Sep 9, 2019 at 2:03 PM Jarek Potiuk <Jarek.Potiuk@polidea.com> wrote:

Philippe Gagnon
Congratulations!

Chao-Han Tsai
Congrats. On Mon, Sep 9, 2019 at 6:15 AM Philippe Gagnon <philgagnon1@gmail.com> wrote:

Kevin Yang
Congrats man :D

Tao Feng
Congrats!

Karolina Rosól
Congratulations! Karolina Rosól Polidea <<https://www.polidea.com/>> | Project Manager M: +48 606 630 236 <+48606630236> E: karolina.rosol@polidea.com [image: Pol]

Driesprong, Fokko
Congrats Felix, well deserved! Cheers, Fokko Op di 10 sep. 2019 om 10:48 schreef Karolina Rosól <karolina.rosol@polidea.com>:

Felix Uellendall
Thank you very much guys. I am feeling lucky to be part of such an awesome community I think there is so much you can learn from one another. I am looking forwa

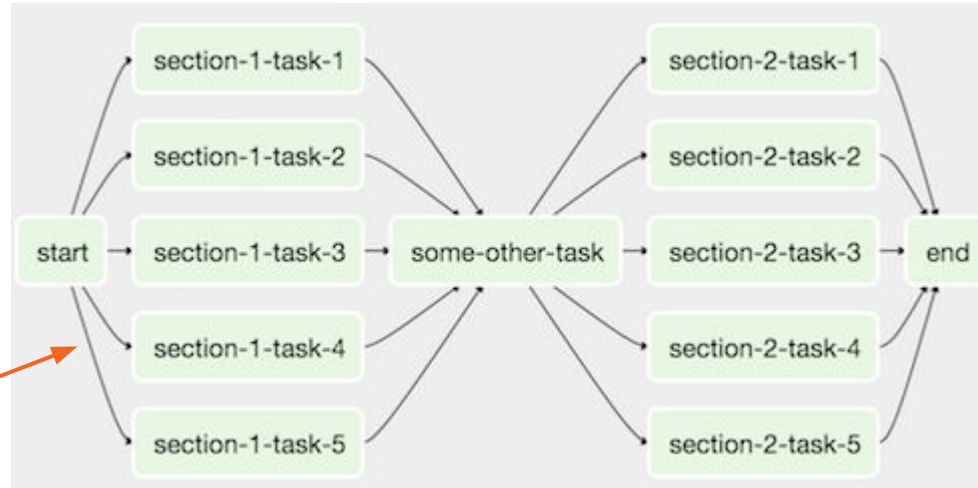
Aizhamal Nurmamat kyzy
[👋](#) to Felix, dev ▾
Congratulations, Felix!

Important Airflow Concepts

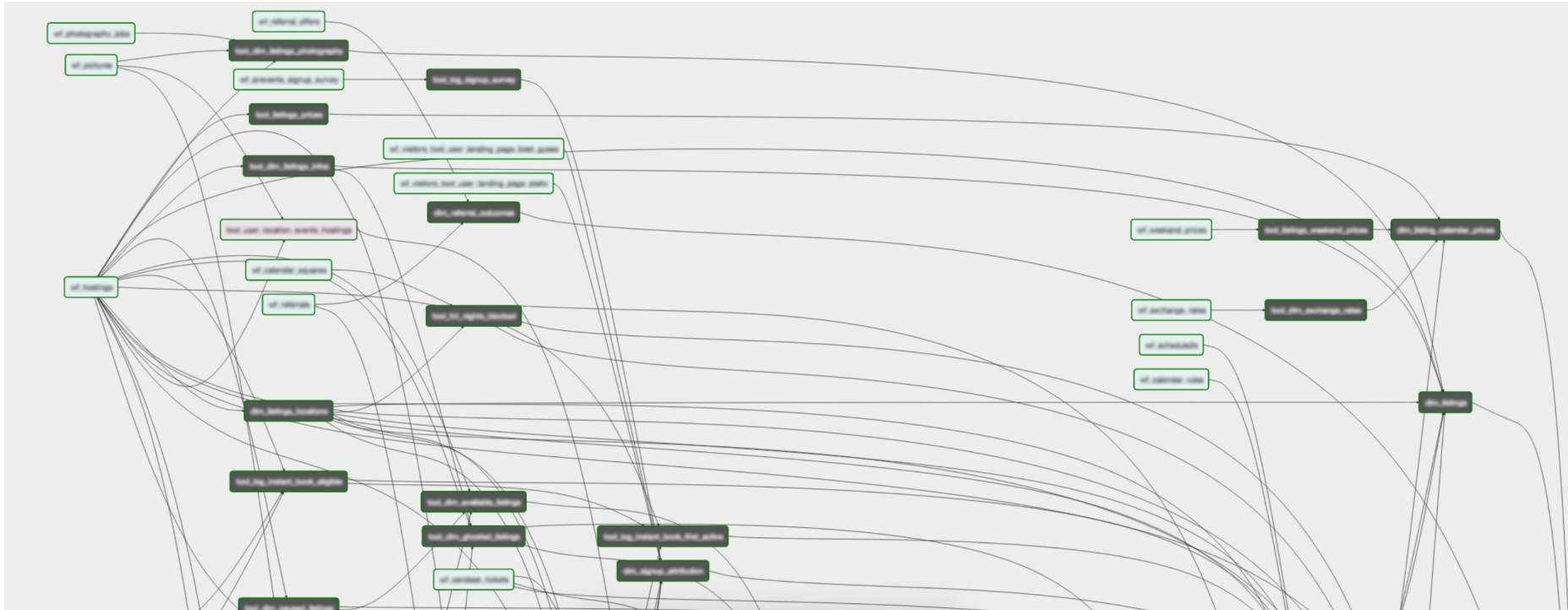
The core concept: The DAG

Dags
have
operators

Operators
are
connected



DAGs can be arbitrarily complex



DAGs are made of Operators

Operators do something... anything.

- BashOperator - run a bash command
 - MySQLOperator - run script on MySQL
 - SlackOperator - send a message to a Slack channel
 - EmailOperator - send an email
 - etc...
-

Some important operators to know

PythonOperator - execute arbitrary method

```
def do_something_really_complex (ds, **kwargs):  
    print("Maybe some numpy or something...")
```

```
run_some_arbitrary_python = PythonOperator(  
    task_id='run_some_arbitrary_python' ,  
    provide_context=True,  
    python_callable=do_something_really_complex,  
    dag=dag)
```

Some important operators to know

BranchingOperator - provide control flow

```
def only_run_on_first_of_month(self, context):  
    """  
    Run an extra branch on the first day of the month  
    """  
    if context['execution_date'].day == 1:  
        return 'monthly_operator'  
    else:  
        return 'daily_operator'
```

```
branch_op = BranchPythonOperator(  
    task_id='choose_monthly_or_daily',  
    provide_context=True,  
    python_callable=only_run_on_first_of_month,  
    dag=dag)
```

Some important operators to know

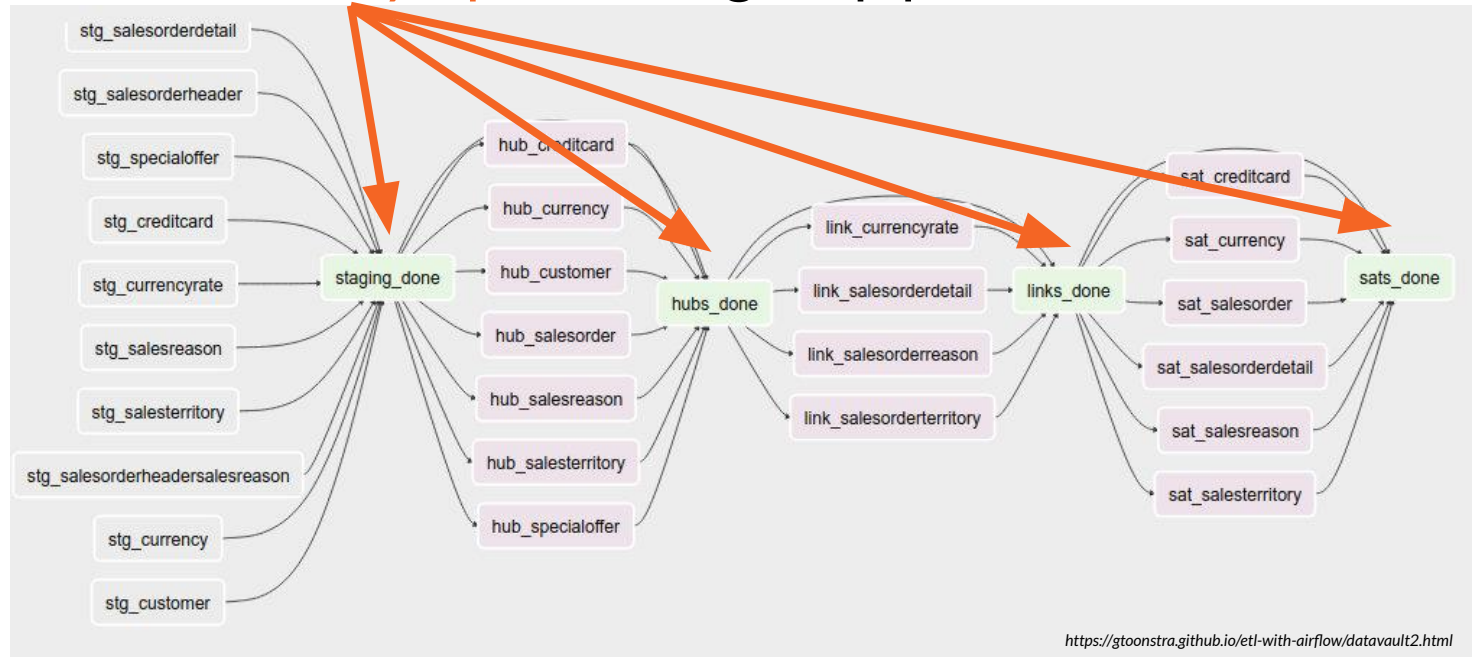
Sensors - wait for some condition

- S3KeySensor - wait for an S3 key to appear
- HttpSensor - wait for a 200 response from some endpoint
- SQLSensor - wait for a specified SQL query to return 0 rows
- etc.

```
wait_for_new_data = SqlSensor(  
    task_id='wait_for_new_data',  
    conn_id='redshift',  
    sql='select * from database where seconds_old > 10',  
    poke_interval=30,  
    timeout=3600,  
    dag=dag)
```

Some important operators to know

DummyOperator - group portions of a DAG



And many, many, many more

adls_list_operator.py
adls_to_gcs.py
aws_athena_operator.py
aws_sqs_publish_operator.py
awsbatch_operator.py
azure_container_instances_operator.py
azure_cosmos_operator.py
bigquery_check_operator.py
bigquery_get_data.py
bigquery_operator.py
bigquery_table_delete_operator.py
bigquery_to_bigquery.py
bigquery_to_gcs.py
bigquery_to_mysql_operator.py
cassandra_to_gcs.py
databricks_operator.py
dataflow_operator.py
dataproc_operator.py
datastore_export_operator.py
datastore_import_operator.py
dingding_operator.py
discord_webhook_operator.py
docker_swarm_operator.py
druid_operator.py
dynamodb_to_s3.py
ecs_operator.py
emr_add_steps_operator.py
emr_create_job_flow_operator.py

emr_terminate_job_flow_operator.py
file_to_gcs.py
file_to_wasb.py
gcp_bigtable_operator.py
gcp_cloud_build_operator.py
gcp_compute_operator.py
gcp_container_operator.py
gcp_dlp_operator.py
gcp_function_operator.py
gcp_natural_language_operator.py
gcp_spanner_operator.py
gcp_speech_to_text_operator.py
gcp_sql_operator.py
gcp_tasks_operator.py
gcp_text_to_speech_operator.py
gcp_transfer_operator.py
gcp_translate_operator.py
gcp_translate_speech_operator.py
gcp_video_intelligence_operator.py
gcp_vision_operator.py
gcs_acl_operator.py
gcs_delete_operator.py
gcs_download_operator.py
gcs_list_operator.py
gcs_operator.py
gcs_to_bq.py
gcs_to_gcs.py
gcs_to_gcs_transfer_operator.py

gcs_to_gdrive_operator.py
gcs_to_s3.py
grpc_operator.py
hipchat_operator.py
hive_to_dynamodb.py
imap_attachment_to_s3_operator.py
jenkins_job_trigger_operator.py
jira_operator.py
kubernetes_pod_operator.py
mlengine_operator.py
mongo_to_s3.py
mssql_to_gcs.py
mysql_to_gcs.py
opsgenie_alert_operator.py
oracle_to_azure_data_lake_transfer.py
oracle_to_oracle_transfer.py
postgres_to_gcs_operator.py
pubsub_operator.py
qubole_check_operator.py
qubole_operator.py
redis_publish_operator.py
s3_copy_object_operator.py
s3_delete_objects_operator.py
s3_list_operator.py
s3_to_gcs_operator.py
s3_to_gcs_transfer_operator.py

s3_to_sftp_operator.py
sagemaker_base_operator.py
sagemaker_endpoint_config_operator.py
sagemaker_endpoint_operator.py
sagemaker_model_operator.py
sagemaker_training_operator.py
sagemaker_transform_operator.py
sagemaker_tuning_operator.py
segment_track_event_operator.py
sftp_operator.py
sftp_to_s3_operator.py
slack_webhook_operator.py
snowflake_operator.py
sns_publish_operator.py
spark_jdbc_operator.py
spark_sql_operator.py
spark_submit_operator.py
sql_to_gcs.py
sqoop_operator.py
ssh_operator.py
vertica_operator.py
vertica_to_hive.py
vertica_to_mysql.py
wasb_delete_blob_operator.py
winrm_operator.py

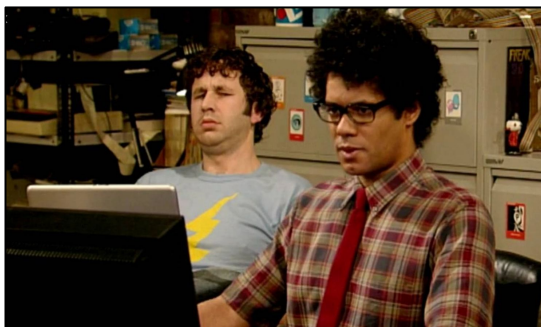
—

Let's talk use cases

Typical first Airflow use case



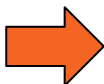
users_funnel.sql
ab_test_results.sql
users_by_region.sql



cron:
0 2 * * * /etl/run_sql.sh users_funnel.sh
0 2 * * * /etl/run_sql.sh ab_test_results.sql
0 2 * * * /etl/run_sql.sh users_by_region.sql

But then comes more SQL

users_funnel.sql
ab_test_results.sql
users_by_region.sql

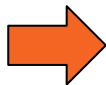


users_funnel.sql, *must run after* normalize_users.sql
ab_test_results.sql
users_by_region.sql *must run* before normalize_users.sql
normalize_users.sql
aggregate_items_by_category.sql
calculate_risk_factors *must run after* normalize_users.sql, incident_reports.sql
user_actions_by_version.sql
some_more_silly_business_stuff.sql
even_more_counting_of_things.sql

Enter the custom etl framework...

cron:

```
0 2 * * * /etl/run_sql.sh users_funnel.sh  
0 2 * * * /etl/run_sql.sh ab_test_results.sql  
0 2 * * * /etl/run_sql.sh users_by_region.sql
```



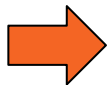
cron:

```
0 2 * * * /etl/etl_runner.py
```



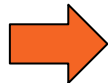
Airflow the rescue

Full set of
reliability features



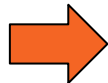
```
args = { 'owner': 'DataEng', 'start_date': datetime(2019, 05, 03),  
        'email_on_failure': de@mycompany.com, 'retries': 3, 'depends_on_past': True,  
        'queue': 'business_etl', 'sla': datetime.timedelta(8 hours),  
        'execution_timeout': datetime.timedelta(1 hour), }
```

Create the DAG



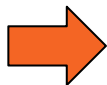
```
dag = DAG(dag_id='daily_business_etl', default_args=args, schedule_interval='0 2 * * *',)
```

Build an Operator
for each SQL



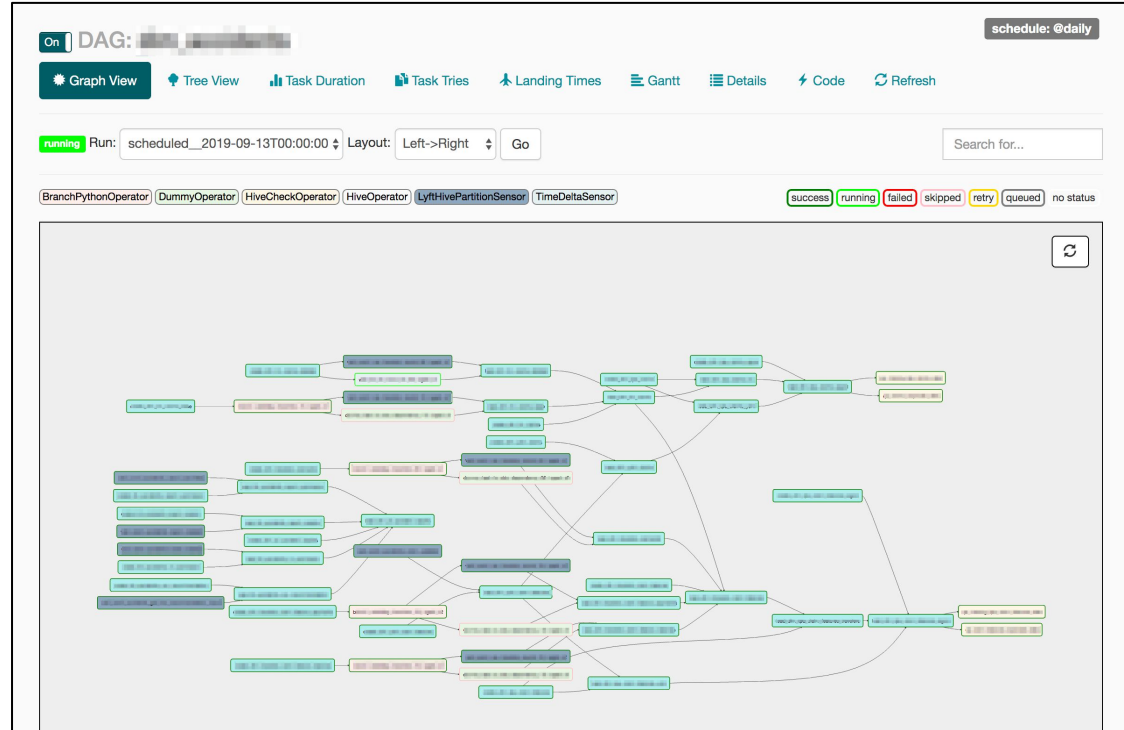
```
# Read in all the sql files we want to run  
sqls_to_run = ['users_funnel', 'ab_test_results', ...]  
  
# And populate them into a map  
sql_ops = dict()  
for sql in sqls_to_run:  
    sql_operator[sql] = SQLOperator(read_sql_file(sql), conn_id='redshift')
```

Wire up
dependencies



```
# Wire up dependencies  
sql_ops['user_funnel'].set_upstream(sql_ops['normalize_users'])  
sql_ops['normalize_users'].set_upstream(sql_ops['users_by_region'])  
sql_ops['calculate_risk_factors'].set_upstream(sql_ops['normalize_users'])  
sql_ops['calculate_risk_factors'].set_upstream(sql_ops['incident_reports'])
```

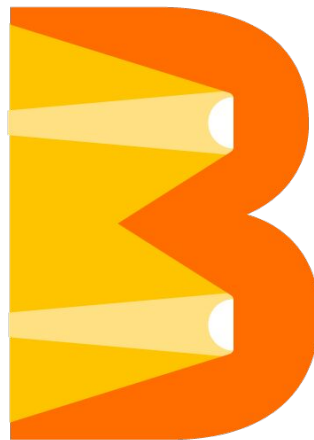
For the win



ETLs like this should be built into Airflow

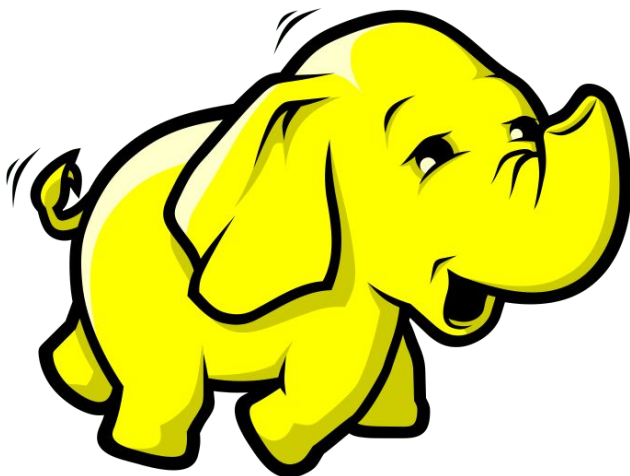
What Airflow is not, part 1

Airflow is not a stream processor, like Apache Flink or Beam

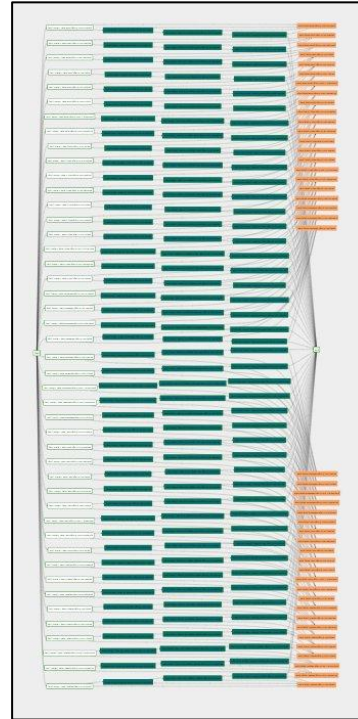


What Airflow is not, part 2

Airflow isn't a Map-Reduce engine, like Hadoop or Hive



A case study in what Airflow isn't



The web interface

DAGs page

The screenshot shows the Airflow DAGs page with a teal header. The main content area is titled 'DAGs' and includes a search bar. Below is a table with the following columns: DAG, Schedule, Owner, Recent Tasks, Last Run, DAG Runs, and Links. The table lists five DAGs: example_bash_operator, example_branch_dop_operator_v3, example_branch_operator, example_xcom, and latest_only. Three orange arrows point to specific elements: the first points to the 'On' status in the first row, the second points to the 'Recent Tasks' progress bar for the second row, and the third points to the 'DAG Runs' count for the fifth row. A pagination control at the bottom left shows '1' selected, and a 'Showing 1 to 5 of 5 entries' message is at the bottom right.

	ⓘ	DAG ⓘ	Schedule	Owner	Recent Tasks ⓘ	Last Run ⓘ	DAG Runs ⓘ	Links
🔗	On	example_bash_operator	00***	airflow	6	2018-09-06 00:00 ⓘ	5	🔗 📊 📈 📉 ⚡ 🔄 🛑
🔗	On	example_branch_dop_operator_v3	*/* * **	airflow	3 1	2018-09-05 00:56 ⓘ	54 3	🔗 📊 📈 📉 ⚡ 🔄 🛑
🔗	On	example_branch_operator	@daily	airflow	5	2018-09-06 00:00 ⓘ	2	🔗 📊 📈 📉 ⚡ 🔄 🛑
🔗	On	example_xcom	@once	airflow	3	2018-09-05 00:00 ⓘ	1	🔗 📊 📈 📉 ⚡ 🔄 🛑
🔗	On	latest_only	4:00:00	Airflow	2	2018-09-07 16:00 ⓘ	35	🔗 📊 📈 📉 ⚡ 🔄 🛑

Showing 1 to 5 of 5 entries

Is the DAG enabled?

How many operators recently succeeded, failed, are retrying, etc.

How many runs recently succeeded, failed, etc.

Graph view

The screenshot displays the Apache Airflow web interface. At the top, there is a navigation bar with the Airflow logo and menu items: DAGs, Data Profiling, Browse, Admin, and Docs. The time 15:13 is shown in the top right corner. Below the navigation bar, the page title is "DAG: core_data". There are several view options: Graph View (selected), Tree View, Task Duration, Landing Times, Gantt, and Code. Below these options, there is a "Run:" field with the value "2015-05-27 00:00:00", a "Layout:" dropdown set to "Left->Right", and a "Go" button. A search bar labeled "Search for..." is also present. Below the search bar, there are buttons for various operators: EmailOperator, HiveOperator, HivePartitionSensor, PrestoCheckOperator, and SubDagOperator. On the right side, there are status indicators: success (green), running (orange), failed (red), and no status (grey). The main area of the interface shows a complex graph view of the DAG. The graph consists of numerous nodes connected by arrows, representing the flow of tasks. The nodes are arranged in a roughly horizontal flow from left to right, with many overlapping lines connecting them, illustrating the complexity of the DAG.



Gets super
slow on big
DAGs

Task Instance details

The screenshot displays the Airflow web interface with a modal window open for 'Task Instance details'. The modal is titled with a task ID and the execution time 'on 2019-09-13T00:00:00'. It contains several sections of buttons:

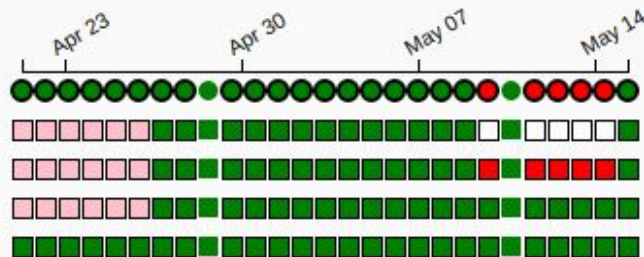
- Task Instance Details** (highlighted), **Rendered**, **Task Instances**, **View Log**
- Run**, **Ignore All Deps**, **Ignore Task State**, **Ignore Task Deps**
- Clear**, **Past**, **Future**, **Upstream**, **Downstream**, **Recursive**
- Mark Success**, **Past**, **Future**, **Upstream**, **Downstream**
- Kibana_Logs**, **Qubole_Log**, **Validate_SQL**
- Close** (bottom right)

The background interface shows the Airflow logo, navigation menu (DAGs, Data Profiling, Browse, Admin, Docs, About, Dependo), and a top right clock showing '21:07 UTC'. Below the modal, a DAG graph is partially visible with task nodes and their dependencies.

Tree view page (super useful)

BashOperator LatestOnlyOperator PythonOperator

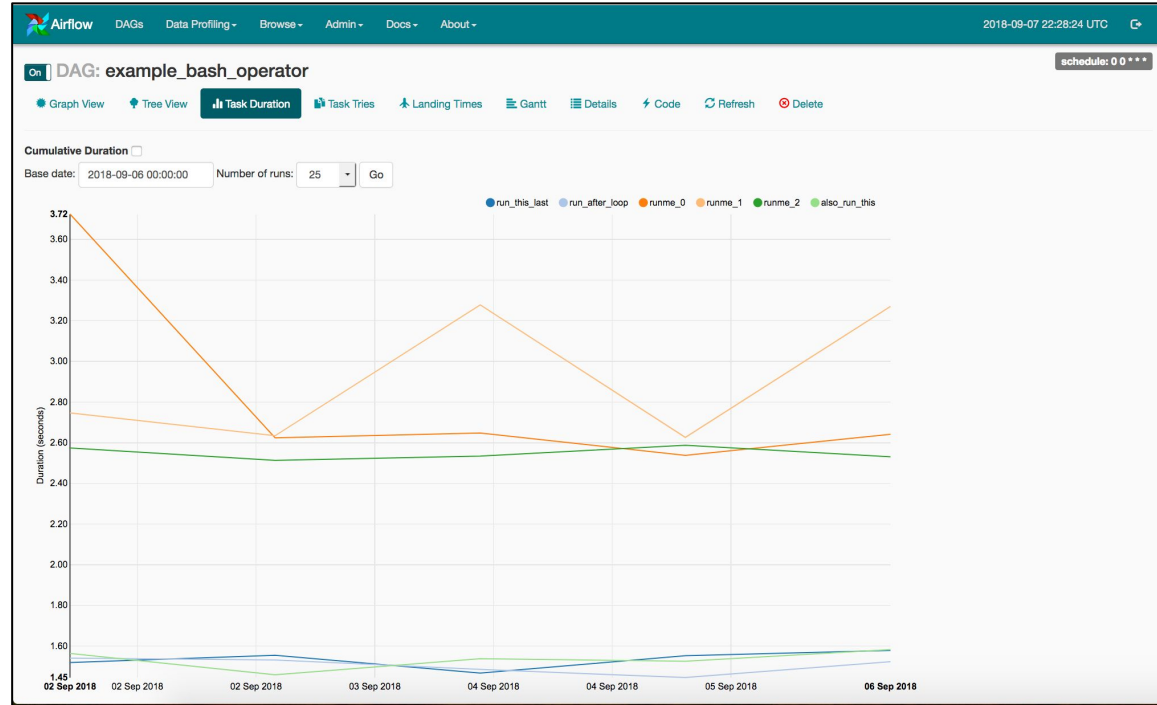
success running failed skipped retry queued no status



Needs to expand, collapse, filter

<http://flagzeta.org/blog/intro-to-apache-airflow/>

Duration view



Totally broken for Really big DAGs

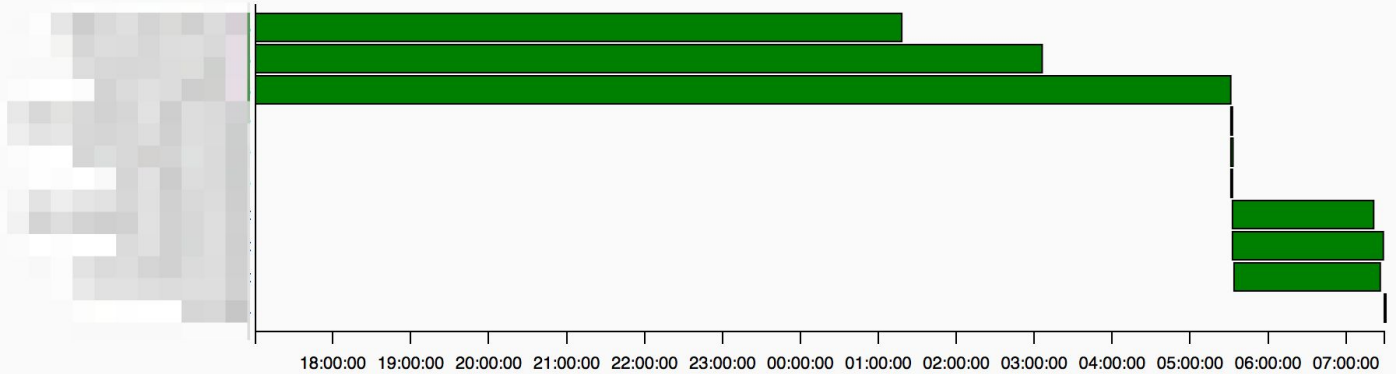
Gantt chart

On DAG: [blurred]

schedule: @daily

- Graph View
- Tree View
- Task Duration
- Task Tries
- Landing Times
- Gantt**
- Details
- Code
- Refresh

Run: 2019-09-13 00:00:00



Details view

On DAG: [blurred] schedule: @daily

Graph View Tree View Task Duration Task Tries Landing Times Gantt **Details** Code Refresh

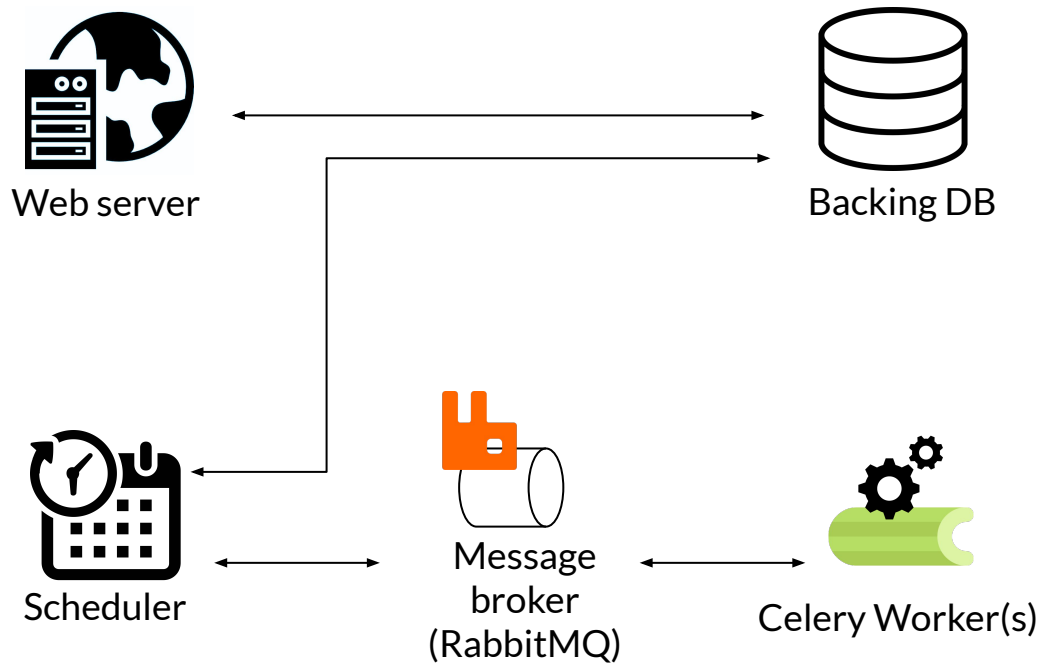
DAG details

failed 60 removed 1 success 1385 upstream_failed 16 None 79

schedule_interval	@daily
max_active_runs	0 / 1
concurrency	125
default_args	[blurred]
tasks count	10
task ids	[blurred]
filepath	[blurred]
owner	[blurred]

Deployment

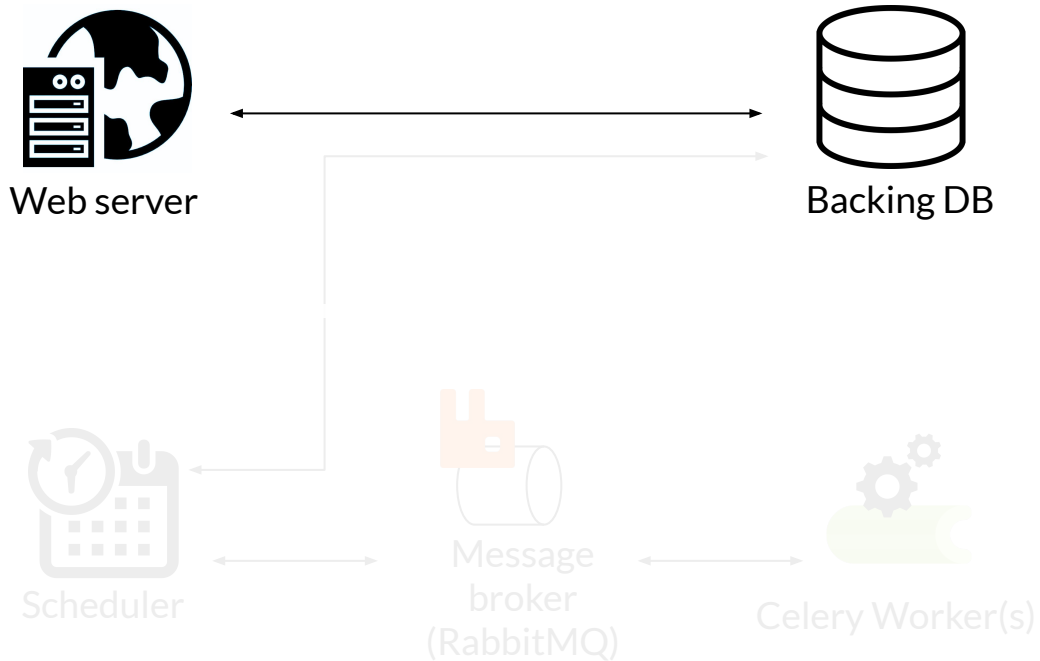
Anatomy of an Airflow cluster



Anatomy of an Airflow cluster

Web server

- Stateless Flask application that talks to Backing DB to get/set task and DAG status.

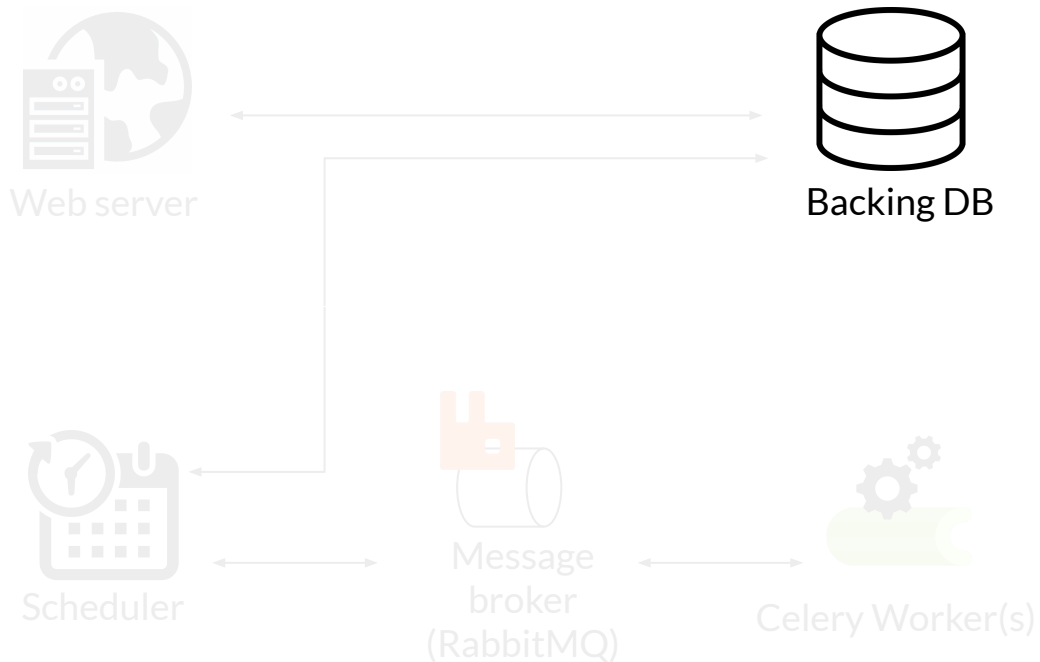


Scaling issues,
UX improvements

Anatomy of an Airflow cluster

Backing DB

- Stores all state for DAGs, tasks, XComs, etc.
- Managed solutions (RDS) work well here.
- Can be perf bottleneck



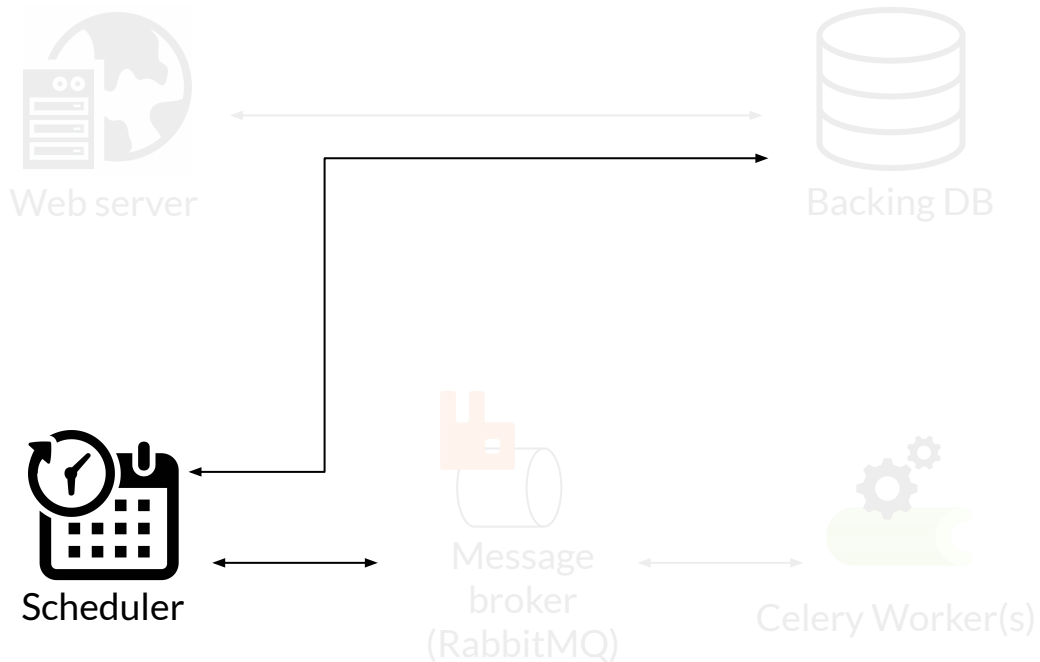
Anatomy of an Airflow cluster

Scheduler

- Kicks off jobs as they're read to be run
- Reasonably buggy
- Written in the wrong language



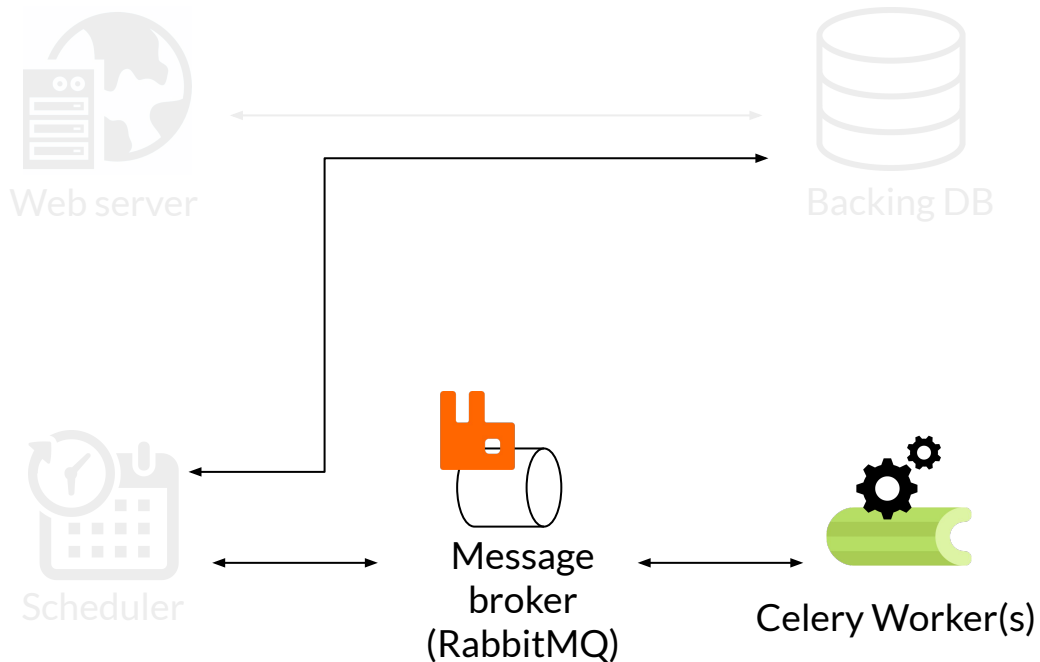
Tests, correctness, features, etc.



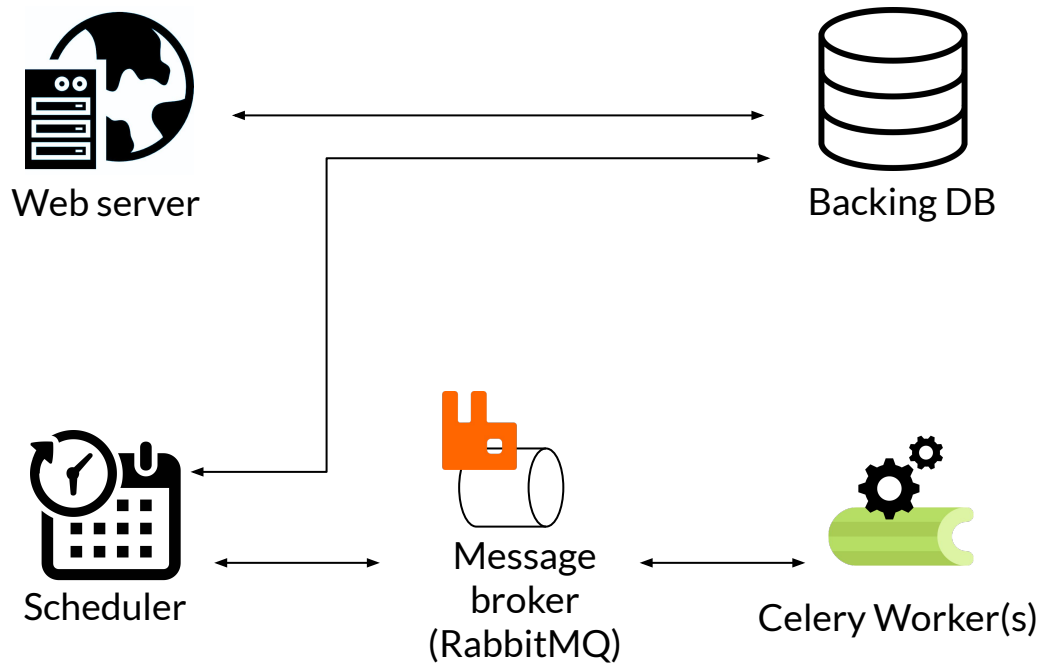
Anatomy of an Airflow cluster

Message Queue and Workers

- Takes requests from scheduler to execute Task Instances
- Scalable, but better support needed



Anatomy of an Airflow cluster



Types of executors

- **CeleryExecutor**
 - Reliable, scalable, hugely popular
- **LocalExecutor**
 - Only for hello world and maybe testing
 - Yeah, don't use it. It's a pain.
- **KubernetesExecutor**
 - Added in 1.10
 - Lots of momentum from Google, others



Docker-based solutions

The screenshot shows the GitHub repository page for `puckel/docker-airflow`. The repository is on the `master` branch and has 212 commits, 1 branch, 34 releases, 32 contributors, and is Apache-2.0 licensed. The most recent commit is by `nsepety` and `puckel`, titled "starting scheduler for SequentialExecutor (#359)", committed 28 days ago. The commit history table lists several recent updates, including bumps to Airflow 1.10.4 and Python 3.7, and updates to the scheduler script. The repository includes files such as `config`, `dags`, `script`, `.dockerignore`, `.gitignore`, `Dockerfile`, `LICENSE`, `README.md`, `docker-compose-CeleryExecutor.yml`, and `docker-compose-LocalExecutor.yml`. At the bottom, the `README.md` file is displayed, showing the repository name `docker-airflow` and a green "PASSED" status for the `docker build` action.

Why GitHub? Enterprise Explore Marketplace Pricing Search Sign in Sign up

puckel / docker-airflow Watch 98 Star 1,679 Fork 1,332

Code Issues 132 Pull requests 23 Projects 0 Security Insights

Docker Apache Airflow

docker-airflow airflow docker scheduler workflow task management

212 commits 1 branch 34 releases 32 contributors Apache-2.0

Branch: master - New pull request Find File Clone or download -

nsepety and puckel starting scheduler for SequentialExecutor (#359) Latest commit c654787 28 days ago

View all commits by nsepety

File	Commit Message	Time
	Bump to Airflow 1.10.4 and Python 3.7	last month
config	Bump to Airflow 1.10.4 and Python 3.7	last month
dags	Bump to 1.9.0-4	last year
script	starting scheduler for SequentialExecutor (#359)	28 days ago
.dockerignore	Initial commit	4 years ago
.gitignore	Update to Apache Airflow 1.9	2 years ago
Dockerfile	Bump to Airflow 1.10.4 and Python 3.7	last month
LICENSE	Create LICENSE	2 years ago
README.md	Bump to Airflow 1.10.4 and Python 3.7	last month
docker-compose-CeleryExecutor.yml	Bump to Airflow 1.10.4 and Python 3.7	last month
docker-compose-LocalExecutor.yml	Bump to Airflow 1.10.4 and Python 3.7	last month

README.md

docker-airflow

PASSED docker build: passing

Gracias!

<https://www.linkedin.com/in/jghoman> - <https://twitter.com/BlueBoxTraveler>

First patch to Airflow



- Trivial change in code base to exercise the mechanics of contributing
 - Checking out code
 - Making a change
 - Testing change
 - Uploading change
 - Getting feedback from the community
 - Visit this spreadsheet to grab a ticket:
<http://bit.ly/CCOSS-Airflow-tickets>
-