Industry DIVE

We're hiring!

How I learned to time travel 

or, data pipelining and scheduling with Airflow 

Laura Lorenz | @lalorenz6 | github.com/lauralorenz

Some truths first 🙌

Data is weird & breaks stuff

User data is particularly untrustworthy. 😈 Don't trust it.

Computers/the Internet/third party services/everything will fail

“

In the beginning, there was Cron.

We had one job, it ran at 1AM, and it was good.

- Pete Owlett, PyData London 2016
  from the outline of his talk:
  “Lessons from 6 months of using Luigi in production”

“

In the beginning, there was ~~Cron~~ CHAOS.

We had ~~one~~ 100 job, it ran at ~~1AM~~ DEPENDS, and it was ~~good~~.

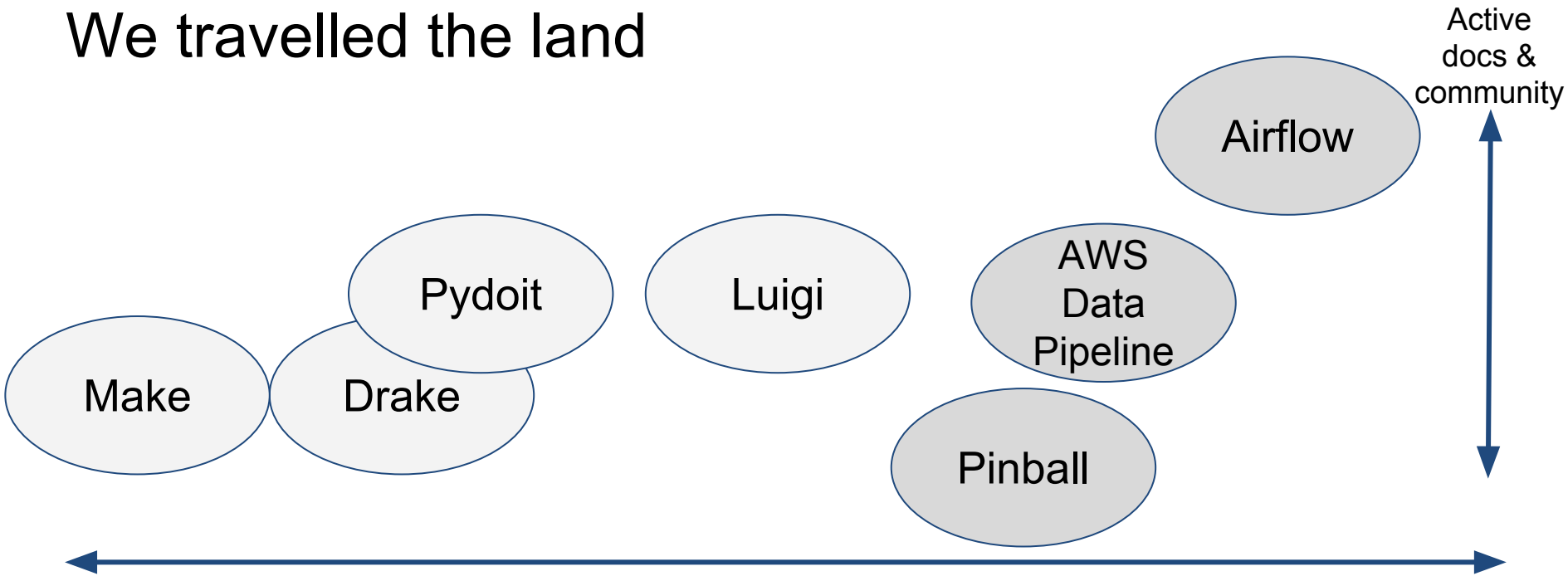
- Pete Owlett, PyData London 2016
  from the outline of his talk:
  “Lessons from 6 months of using Luigi in production”

Plumbing problems suck

We had thoughts about how this should go

- Prefer something in open source Python so we know what's going on and can easily extend or customize
- Resilient
  - Handles failure well; i.e. retry logic, failure callbacks, alerting
- Deals with Complexity Intelligently
  - Can handle complicated dependencies and only runs what it has to
- Flexibility
  - Can run anything we want
- We knew we had batch tasks on daily and hourly schedules

We travelled the land



- File based dependencies
- Dependency framework only
- Lightweight, protocols minimally specified

- Abstract dependencies
- Ships with scheduling & monitoring
- Heavyweight, batteries included

File dependencies/target systems

File dependencies/target systems


```
#Makefile
wrangled.csv : source1.csv source2.csv
    cat source1.csv source2.csv > wrangled.csv
```

File dependencies/target systems


```
#Drakefile
wrangled.csv <- source1.csv, source2.csv [shell]
    cat $INPUT0 $INPUT1 > $OUTPUT
```

File dependencies/target systems


```
#Pydoit
def task_example():
    return {"targets": ['wrangled.csv'],
            "file_deps": ['source1.csv', 'source2.csv'],
            "actions": [concatenate_files_func]
            }
```

File dependencies/target systems


```
# Luigi
class TaskC(luigi.Task):
    def requires(self):
        return output_from_a()

    def output(self):
        return input_for_e()
```

```
# Luigi cont
    def run(self):
        do_the_thing(
         self.requires, self.output)
```

File dependencies/target systems

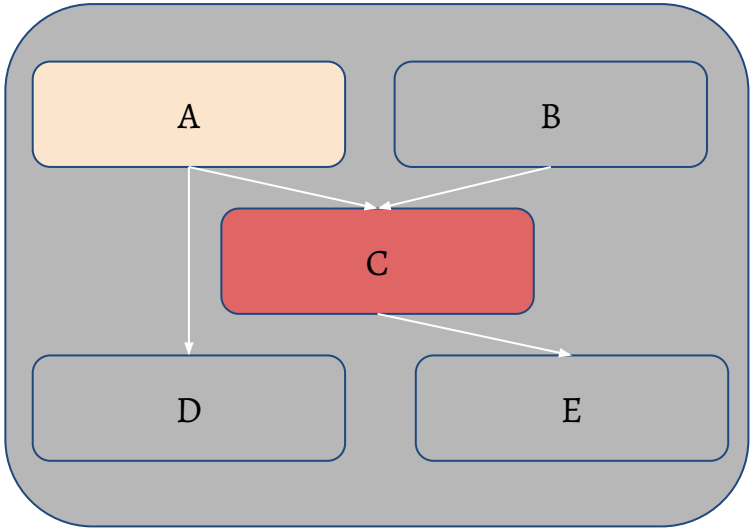
Pros

- Work is cached in files
  - Smart rebuilding
- Simple and intuitive configuration especially for data transformations

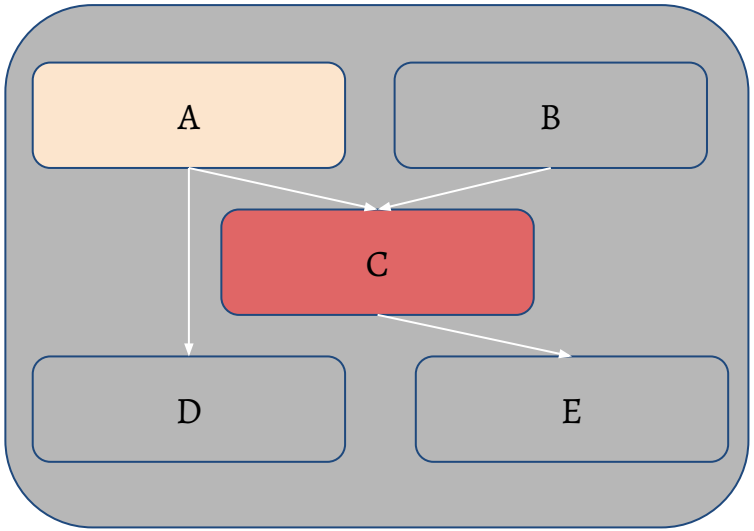
Cons

- No native concept of schedule
  - Luigi is the first to introduce this, but lacks built in polling process
- Alerting systems too basic
- Design paradigm not broadly applicable to non-target operations

Abstract orchestration systems

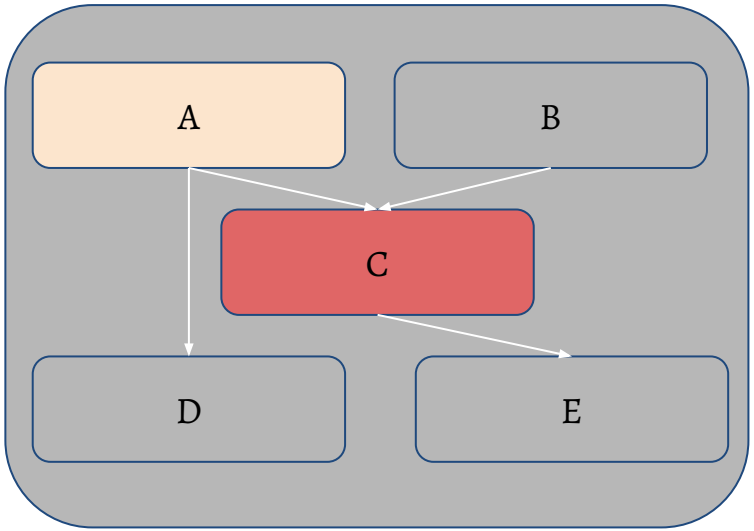


Abstract orchestration systems




```
# Pinball
WORKFLOW = {"ex": WorkflowConfig(
    jobs={
        "A": JobConfig(
                JobTemplate(A),
                []),
        "C": JobConfig(
                JobTemplate(C),
                ["A"],
    …,
    schedule=ScheduleConfig(
      recurrence=timedelta(days=1),
      reference_timestamp=
        datetime(
            year=2016, day=8,
        month=10))
     …,
```

Abstract orchestration systems




```
# Airflow
dag = DAG(schedule_interval=
            timedelta(days=1),
          start_date=
            datetime(2015,10,6))

a =  PythonOperator(
        task_id="A",
        python_callable=ClassA,
        dag=dag)
c =  MySQLOperator(
        task_id="B",
        sql="DROP TABLE hello",
        dag=dag)

c.set_upstream(a)
```

Abstract orchestration systems

Pros

- Support many more types of operations out of the box
- Handles more complicated dependency logic
- Scheduling, monitoring, and alerting services built-in and sophisticated

Cons

- Caching is per service; loses focus on individual data transformations
- Configuration is more complex
- More infrastructure dependencies
  - Database for state
  - Queue for distribution

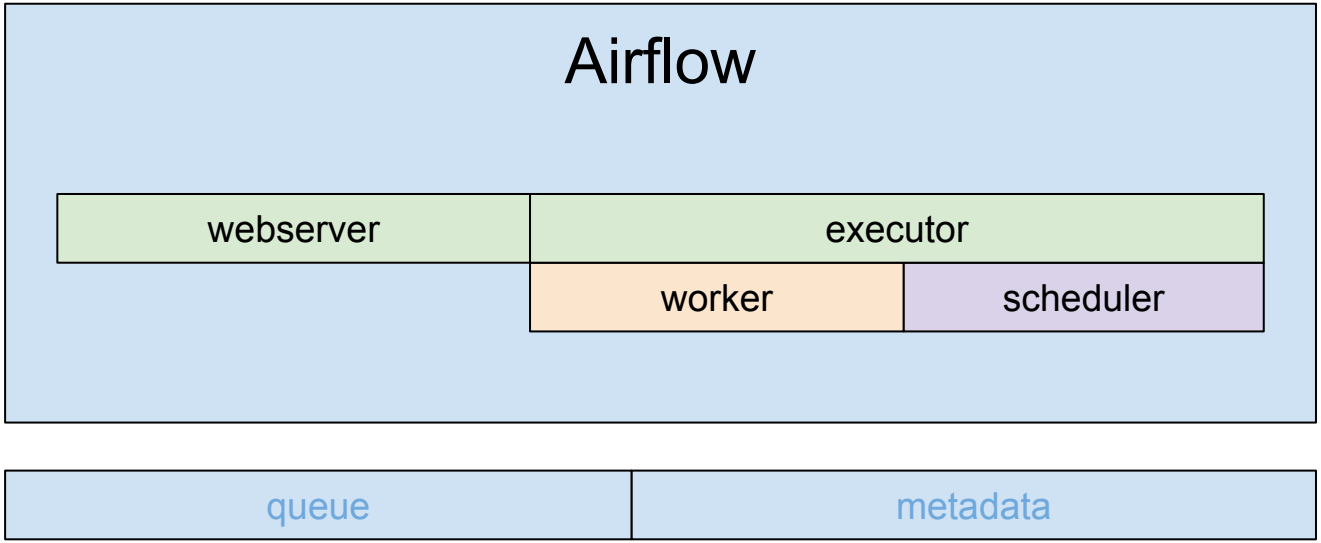
Armed with knowledge, we had more opinions

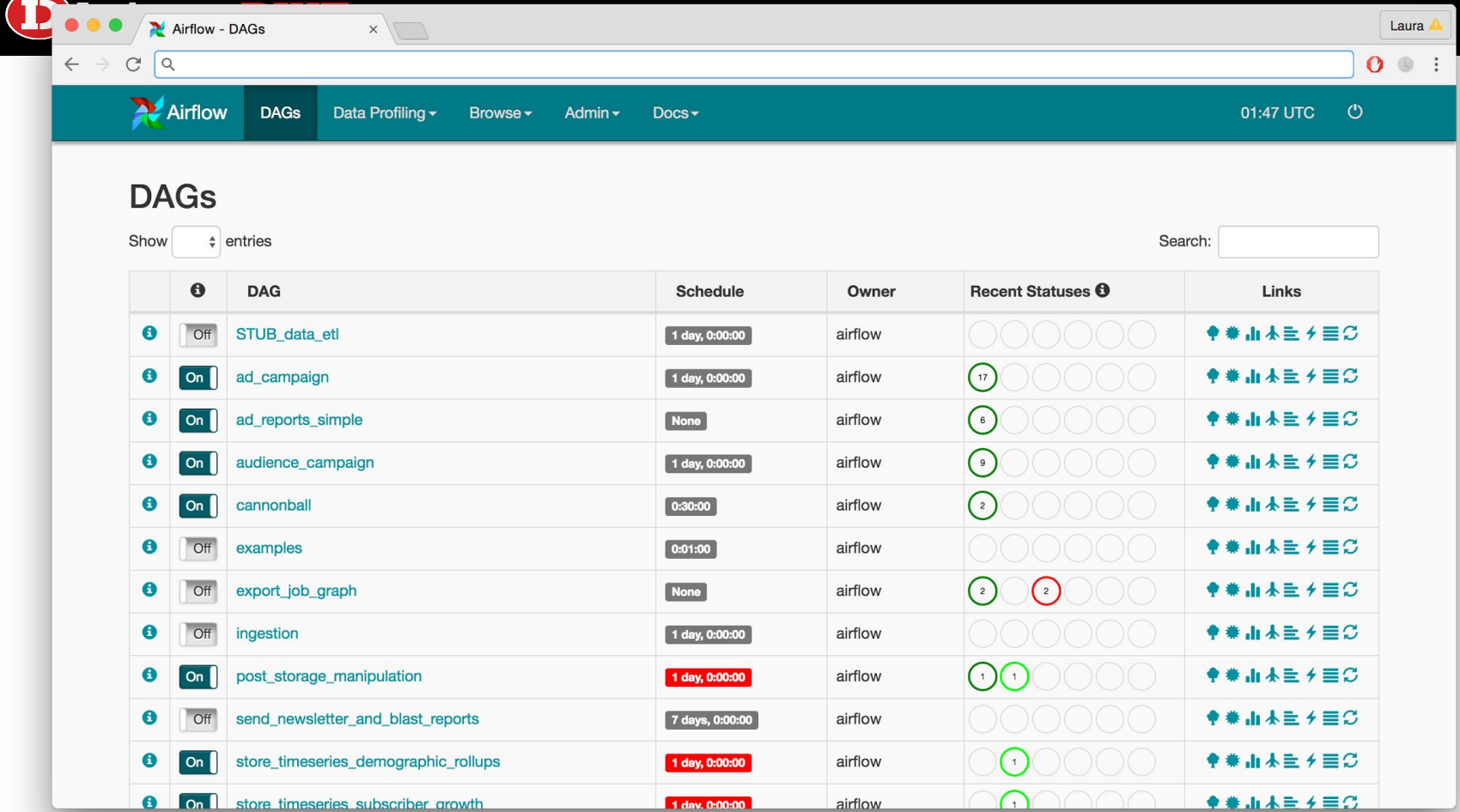
- We like the sophistication of the abstract orchestration systems
- But we also like Drake/Luigi-esque file targeting for transparency and data bug tracking
  - “Intermediate artifacts”
- We (I/devops) don’t want to maintain a separate scheduling service
- We like a strong community and good docs
- We don’t want to be stuck in one ecosystem

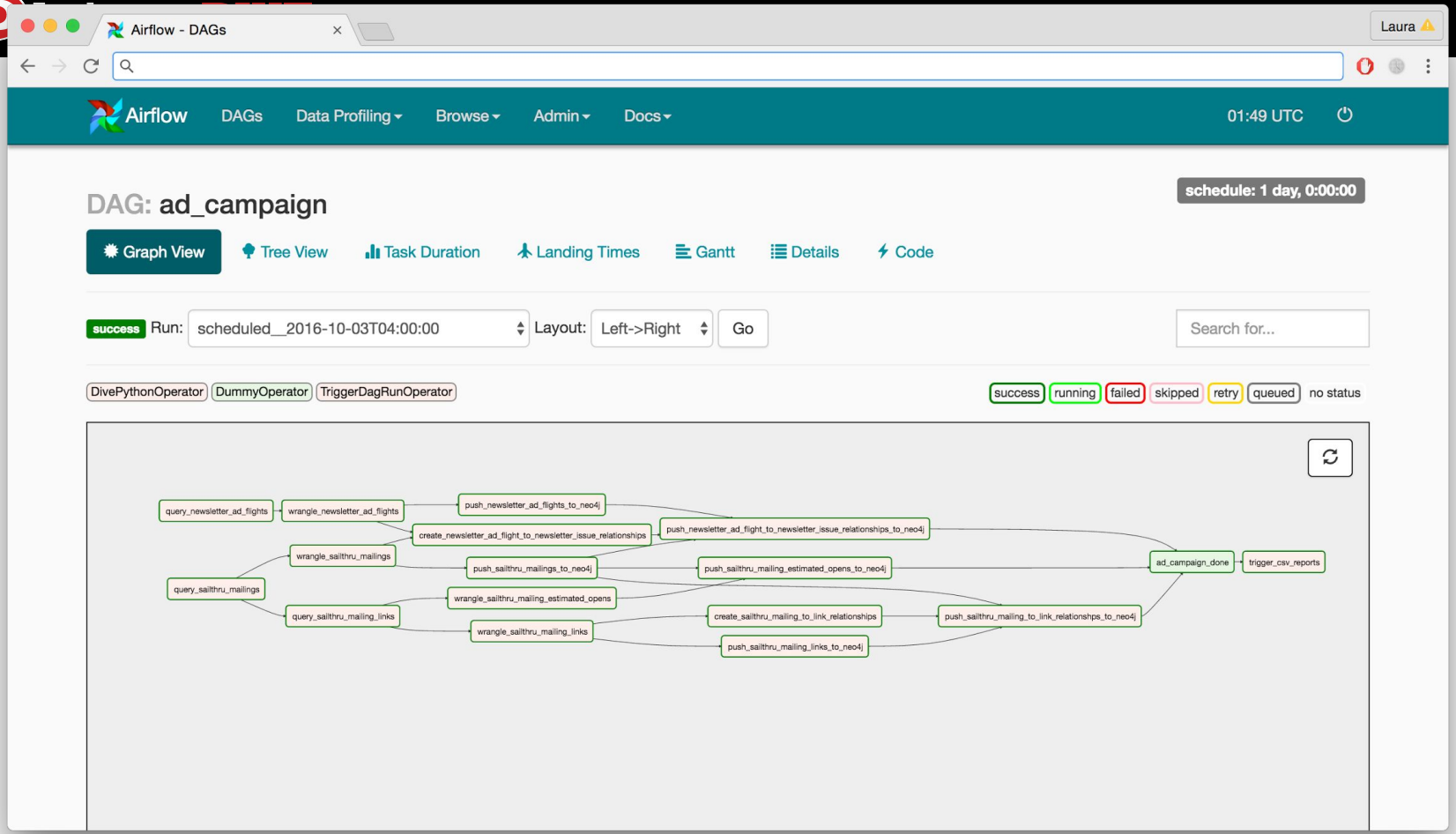
Airflow + “smart-airflow” =

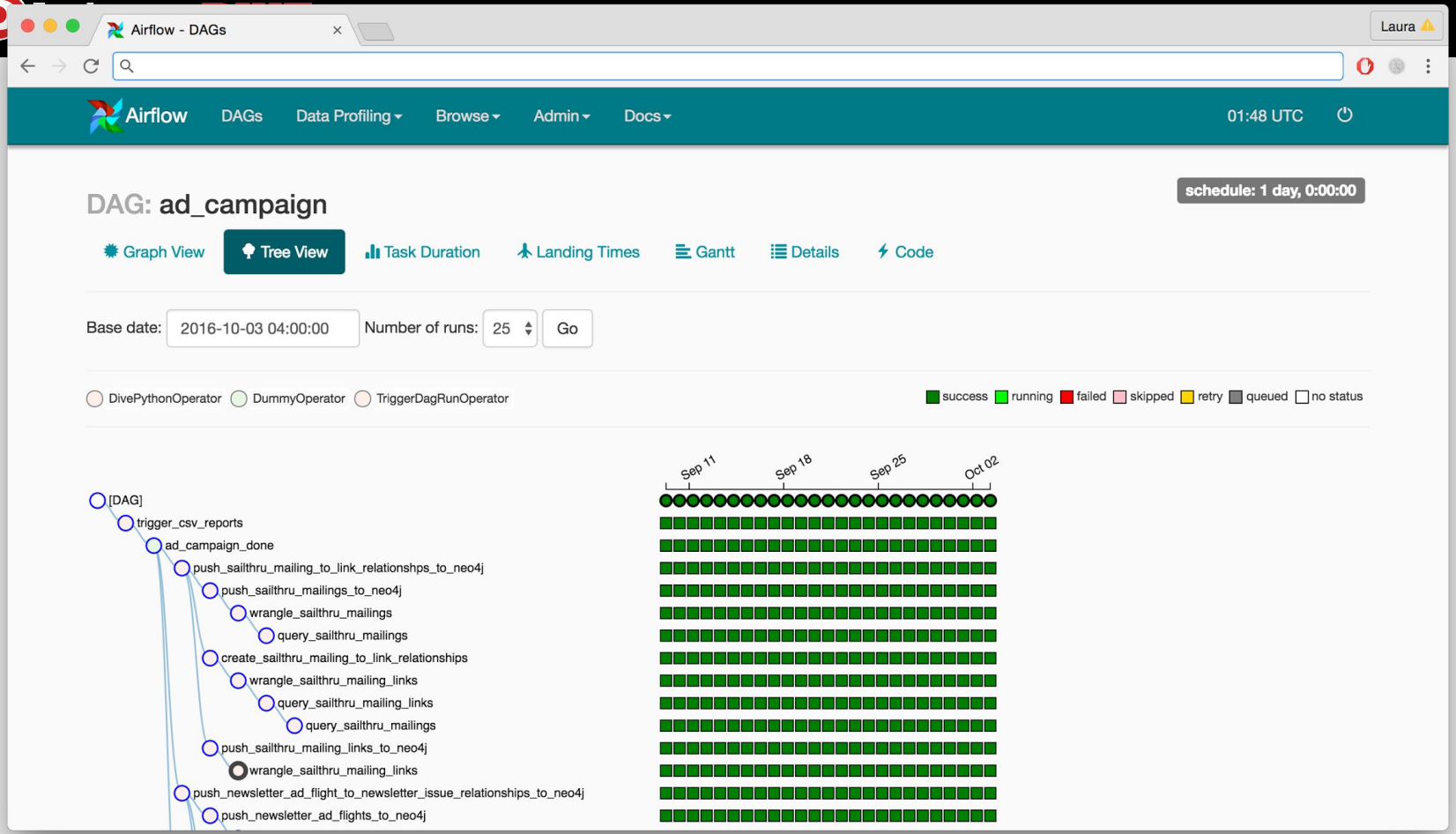
Airflow

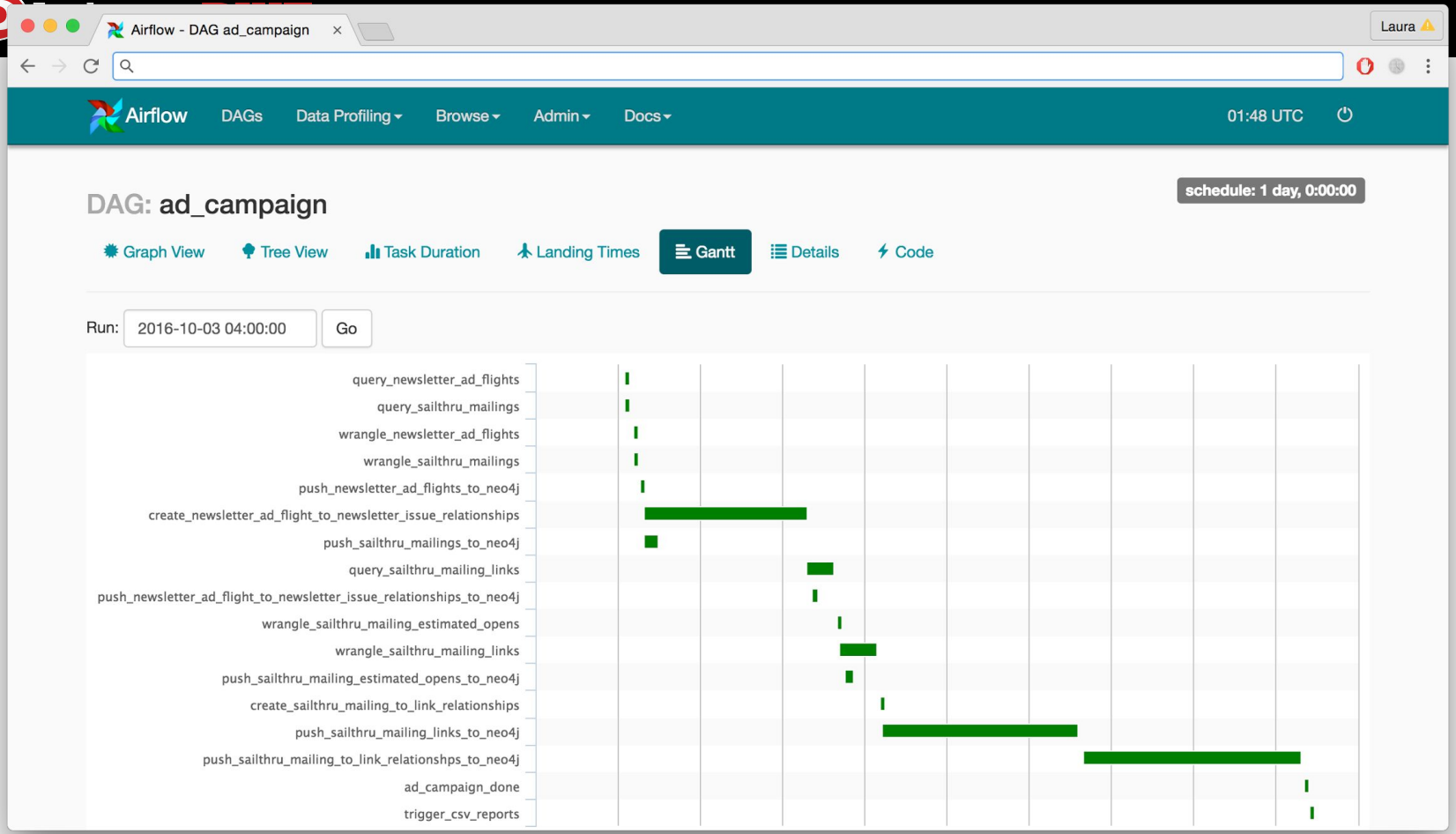
- Scheduler process handles triggering and executing work specified in DAGs on a given schedule
- Built in alerting based on service license agreements or task state
- Lots of sexy profiling visualizations
- `test`, `backfill`, `clear` operations convenient from the CLI
- Operators can come from a number of prebuilt classes like PythonOperator, S3KeySensor, or BaseTransfer, or can obviously extend using inheritance
- Can support local or distributed work execution; distributed needs a celery backend service (RabbitMQ, Redis)


Airflow
webserver
executor
worker
scheduler
queue
metadata

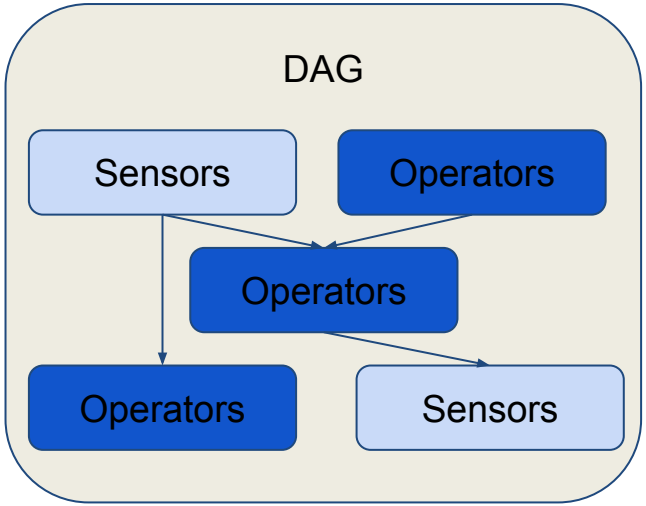

Airflow - DAGs
Laura
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
01:47 UTC
DAGs
Show entries
Search:
DAG
Schedule
Owner
Recent Statuses
Links
STUB_data_etl
1 day, 0:00:00
airflow
ad_campaign
1 day, 0:00:00
airflow
ad_reports_simple
None
airflow
audience_campaign
1 day, 0:00:00
airflow
cannonball
0:30:00
airflow
examples
0:01:00
airflow
export_job_graph
None
airflow
ingestion
1 day, 0:00:00
airflow
post_storage_manipulation
1 day, 0:00:00
airflow
send_newsletter_and_blast_reports
7 days, 0:00:00
airflow
store_timeseries_demographic_rollups
1 day, 0:00:00
airflow
store_timeseries_subscriber_growth
1 day, 0:00:00
airflow


Airflow - DAGs
Laura
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
01:49 UTC
DAG: ad_campaign
schedule: 1 day, 0:00:00
Graph View
Tree View
Task Duration
Landing Times
Gantt
Details
Code
success
Run:
scheduled__2016-10-03T04:00:00
Layout:
Left->Right
Go
Search for...
DivePythonOperator
DummyOperator
TriggerDagRunOperator
success
running
failed
skipped
retry
queued
no status
query_newsletter_ad_flights
wrangle_newsletter_ad_flights
push_newsletter_ad_flights_to_neo4j
create_newsletter_ad_flight_to_newsletter_issue_relationships
push_newsletter_ad_flight_to_newsletter_issue_relationships_to_neo4j
wrangle_sailthru_mailings
push_sailthru_mailings_to_neo4j
push_sailthru_mailing_estimated_opens_to_neo4j
ad_campaign_done
trigger_csv_reports
query_sailthru_mailings
wrangle_sailthru_mailing_estimated_opens
query_sailthru_mailing_links
wrangle_sailthru_mailing_links
create_sailthru_mailing_to_link_relationships
push_sailthru_mailing_to_link_relationshps_to_neo4j
push_sailthru_mailing_links_to_neo4j


Airflow - DAGs
Laura
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
01:48 UTC
DAG: ad_campaign
schedule: 1 day, 0:00:00
Graph View
Tree View
Task Duration
Landing Times
Gantt
Details
Code
Base date: 2016-10-03 04:00:00
Number of runs: 25
Go
DivePythonOperator
DummyOperator
TriggerDagRunOperator
success
running
failed
skipped
retry
queued
no status
Sep 11
Sep 18
Sep 25
Oct 02
[DAG]
trigger_csv_reports
ad_campaign_done
push_sailthru_mailing_to_link_relationshps_to_neo4j
push_sailthru_mailings_to_neo4j
wrangle_sailthru_mailings
query_sailthru_mailings
create_sailthru_mailing_to_link_relationships
wrangle_sailthru_mailing_links
query_sailthru_mailing_links
query_sailthru_mailings
push_sailthru_mailing_links_to_neo4j
wrangle_sailthru_mailing_links
push_newsletter_ad_flight_to_newsletter_issue_relationships_to_neo4j
push_newsletter_ad_flights_to_neo4j


Airflow - DAG ad_campaign
Laura
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
01:48 UTC
DAG: ad_campaign
schedule: 1 day, 0:00:00
Graph View
Tree View
Task Duration
Landing Times
Gantt
Details
Code
Run:
2016-10-03 04:00:00
Go
query_newsletter_ad_flights
query_sailthru_mailings
wrangle_newsletter_ad_flights
wrangle_sailthru_mailings
push_newsletter_ad_flights_to_neo4j
create_newsletter_ad_flight_to_newsletter_issue_relationships
push_sailthru_mailings_to_neo4j
query_sailthru_mailing_links
push_newsletter_ad_flight_to_newsletter_issue_relationships_to_neo4j
wrangle_sailthru_mailing_estimated_opens
wrangle_sailthru_mailing_links
push_sailthru_mailing_estimated_opens_to_neo4j
create_sailthru_mailing_to_link_relationships
push_sailthru_mailing_links_to_neo4j
push_sailthru_mailing_to_link_relationshps_to_neo4j
ad_campaign_done
trigger_csv_reports

Let's talk about DAGs and Tasks

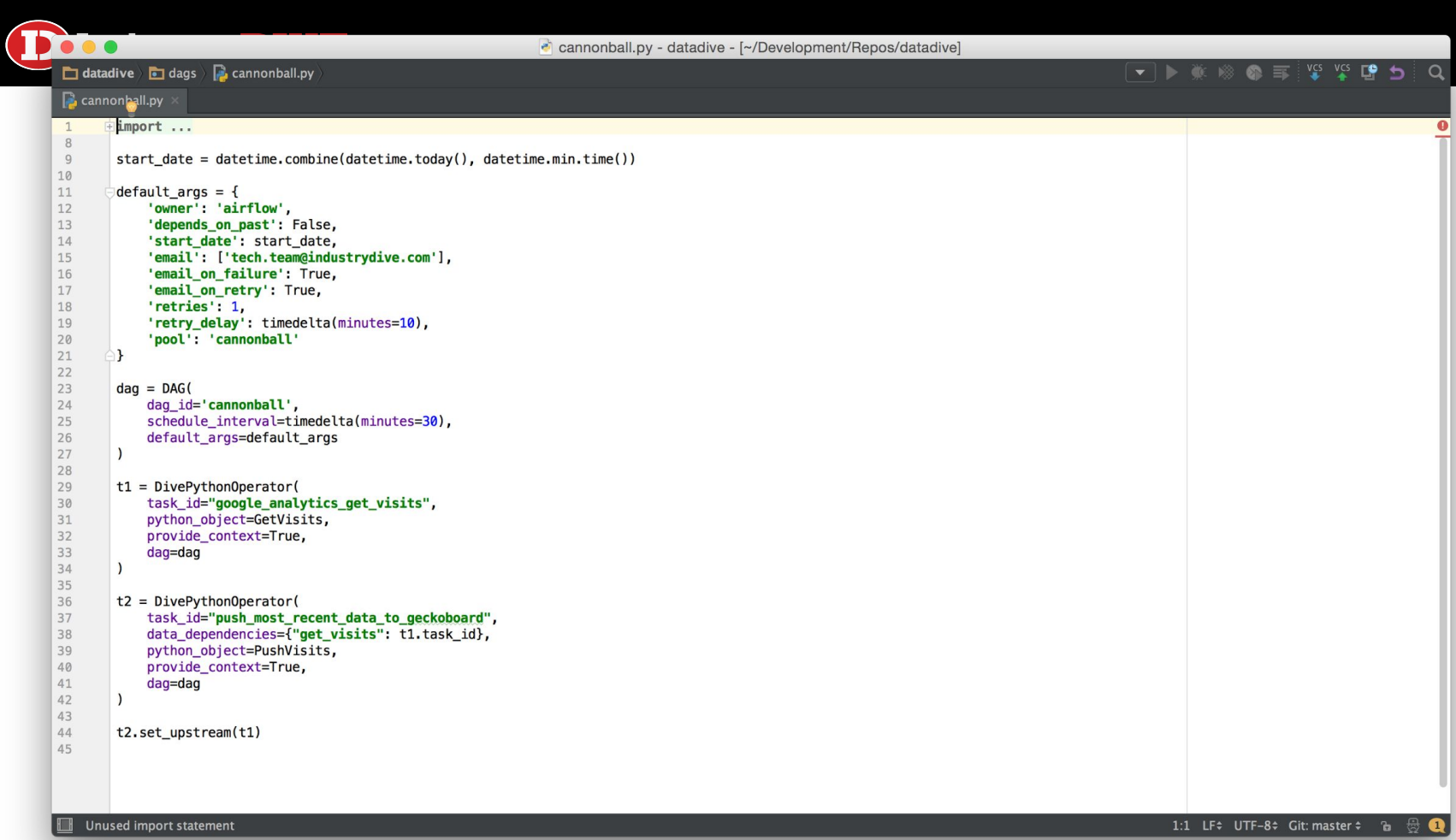


DAG properties

schedule_interval
start_date
max_active_runs

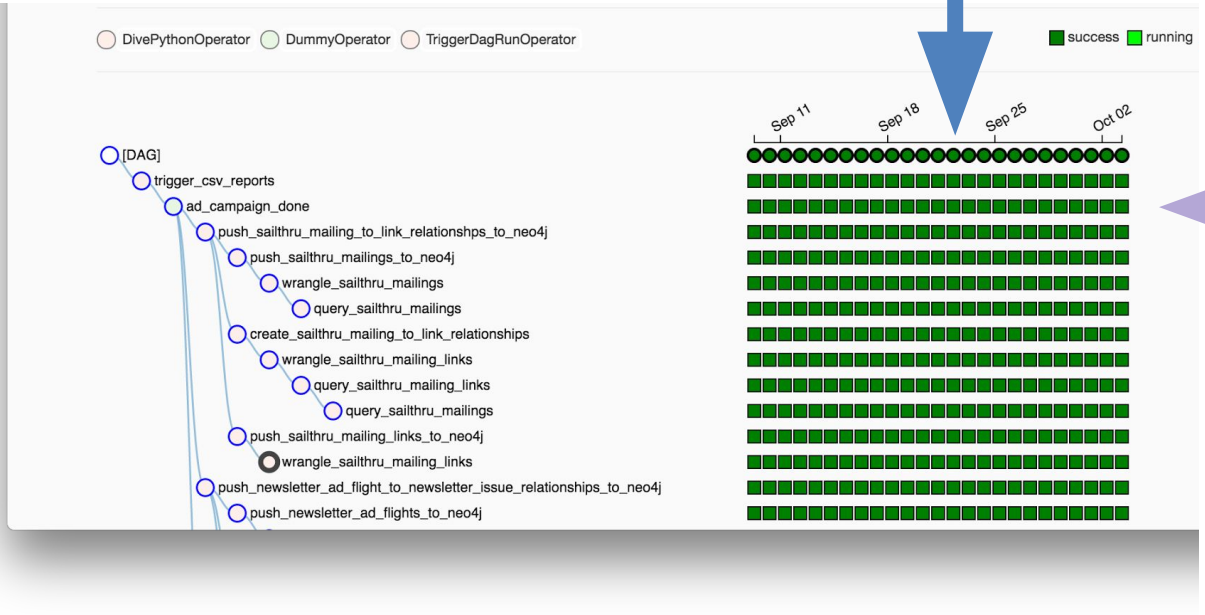
Task properties

poke_interval
timeout
owner
retries
on_failure_callback
data_dependencies


cannonball.py - datadive - [~/Development/Repos/datadive]
datadive
dags
cannonball.py
cannonball.py
import ...
start_date = datetime.combine(datetime.today(), datetime.min.time())
default_args = {
'owner': 'airflow',
'depends_on_past': False,
'start_date': start_date,
'email': ['tech.team@industrydive.com'],
'email_on_failure': True,
'email_on_retry': True,
'retries': 1,
'retry_delay': timedelta(minutes=10),
'pool': 'cannonball'
}
dag = DAG(
dag_id='cannonball',
schedule_interval=timedelta(minutes=30),
default_args=default_args
)
t1 = DivePythonOperator(
task_id="google_analytics_get_visits",
python_object=GetVisits,
provide_context=True,
dag=dag
)
t2 = DivePythonOperator(
task_id="push_most_recent_data_to_geckoboard",
data_dependencies={"get_visits": t1.task_id},
python_object=PushVisits,
provide_context=True,
dag=dag
)
t2.set_upstream(t1)
Unused import statement
1:1
LF
UTF-8
Git: master

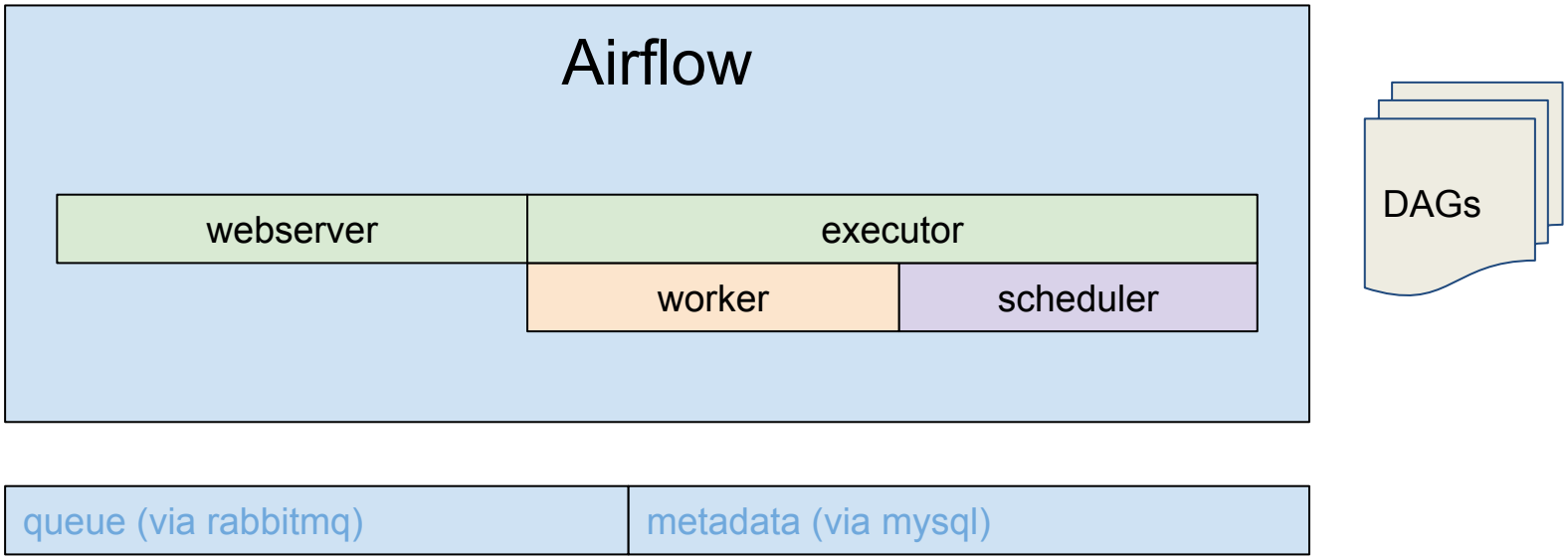
Let's talk about DagRuns and TaskInstances

DagRuns:
DAG by "time"

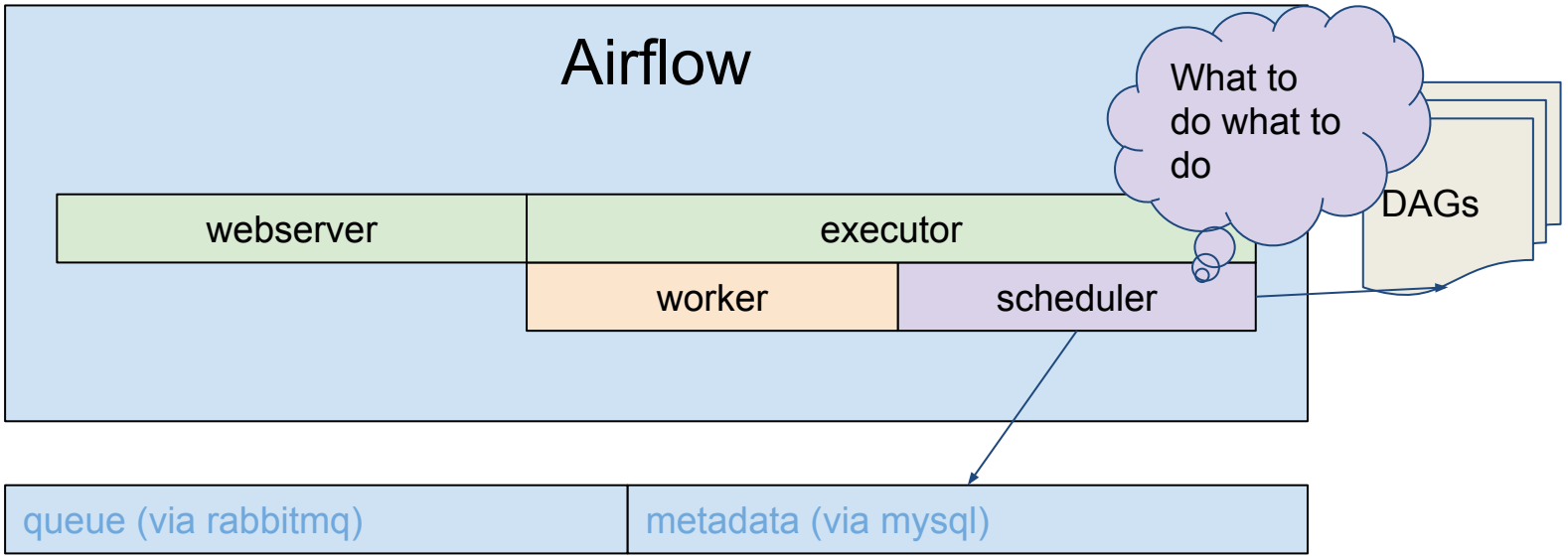


TaskInstances:
Task by DagRun

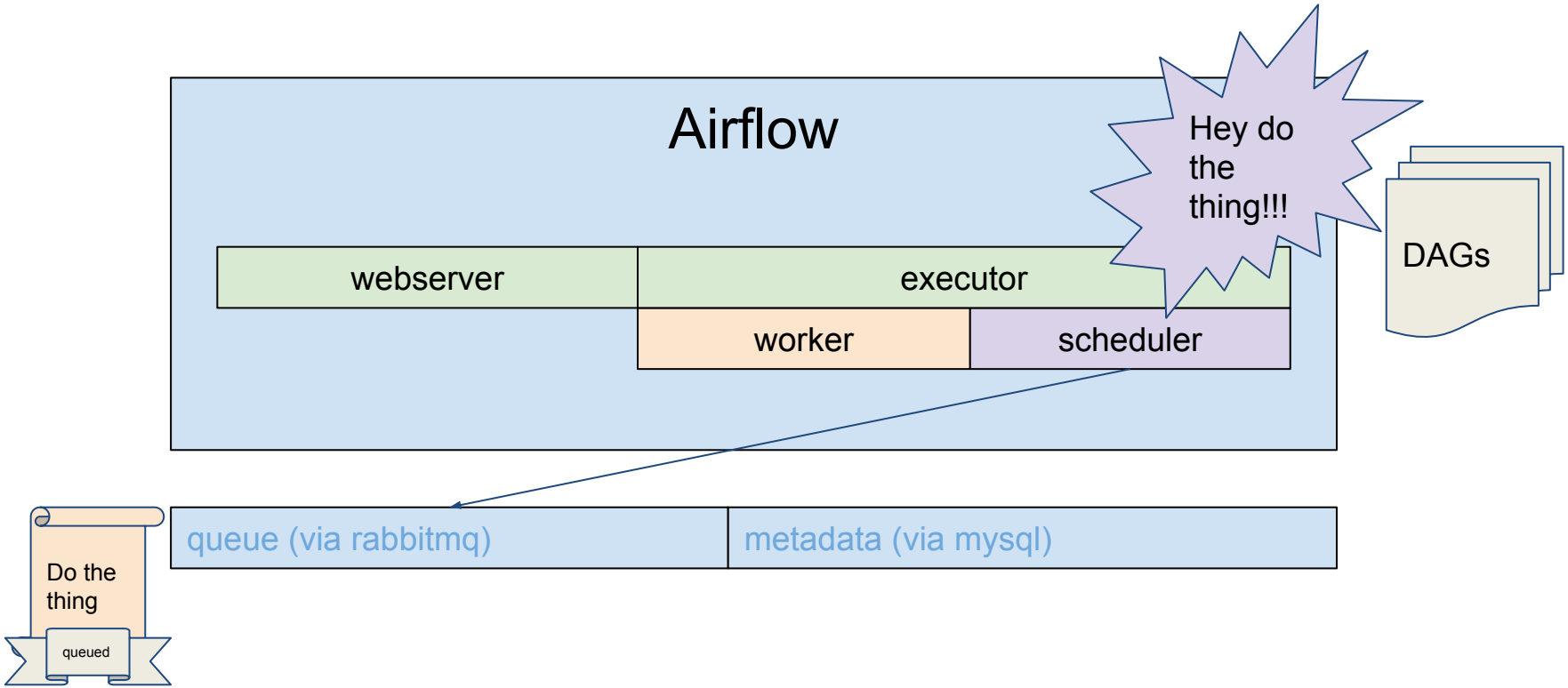
Let's talk about airflow services



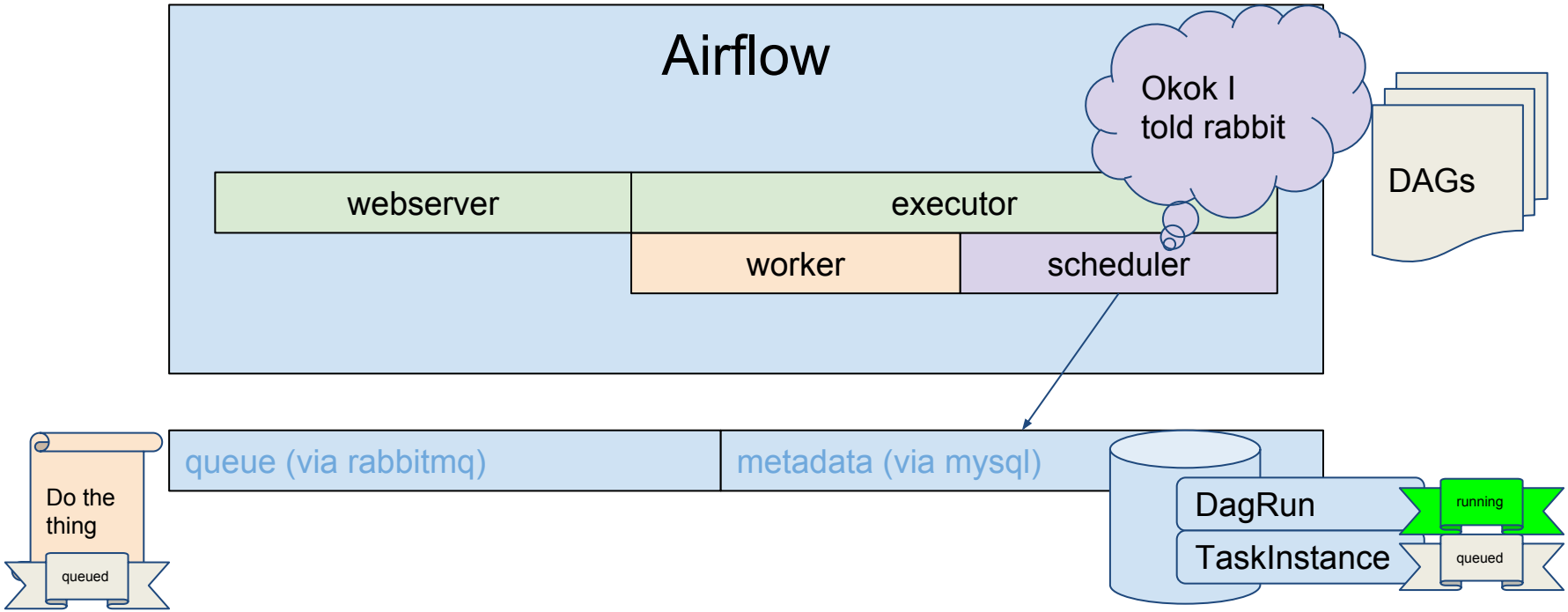
Let's talk about airflow services



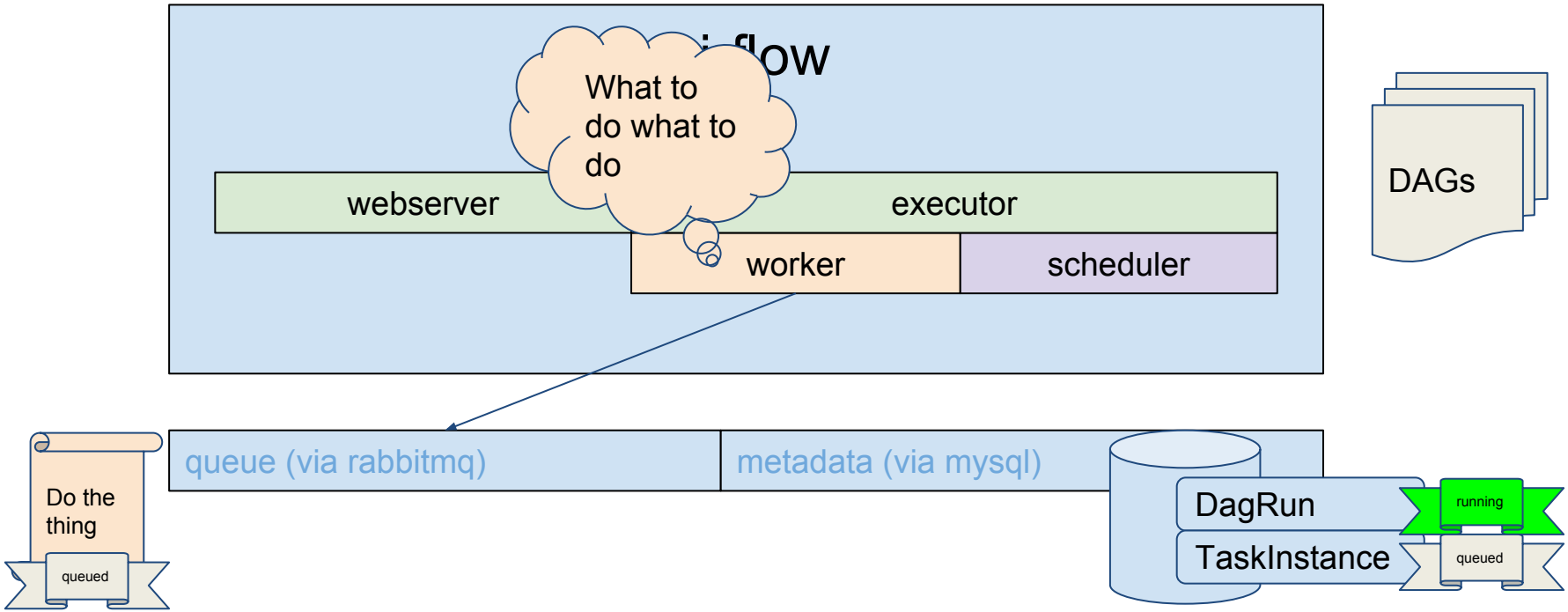
Let's talk about airflow services



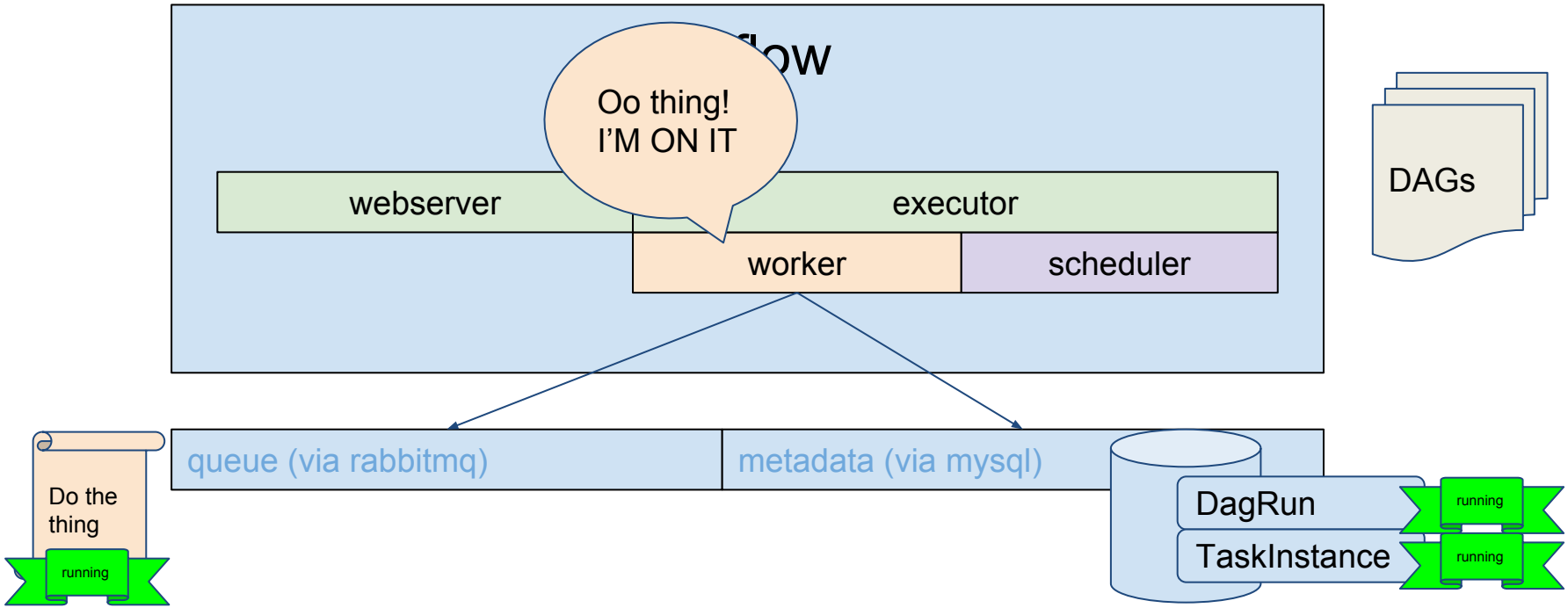
Let's talk about airflow services



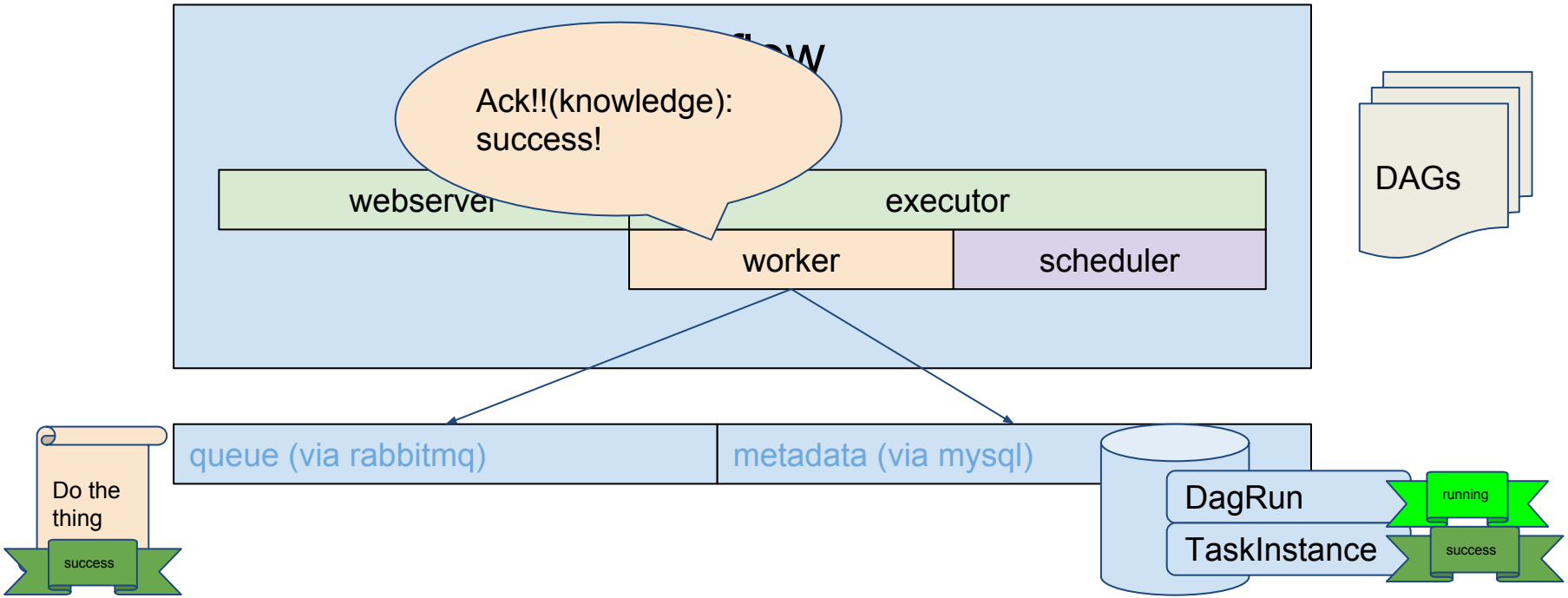
Let's talk about airflow services



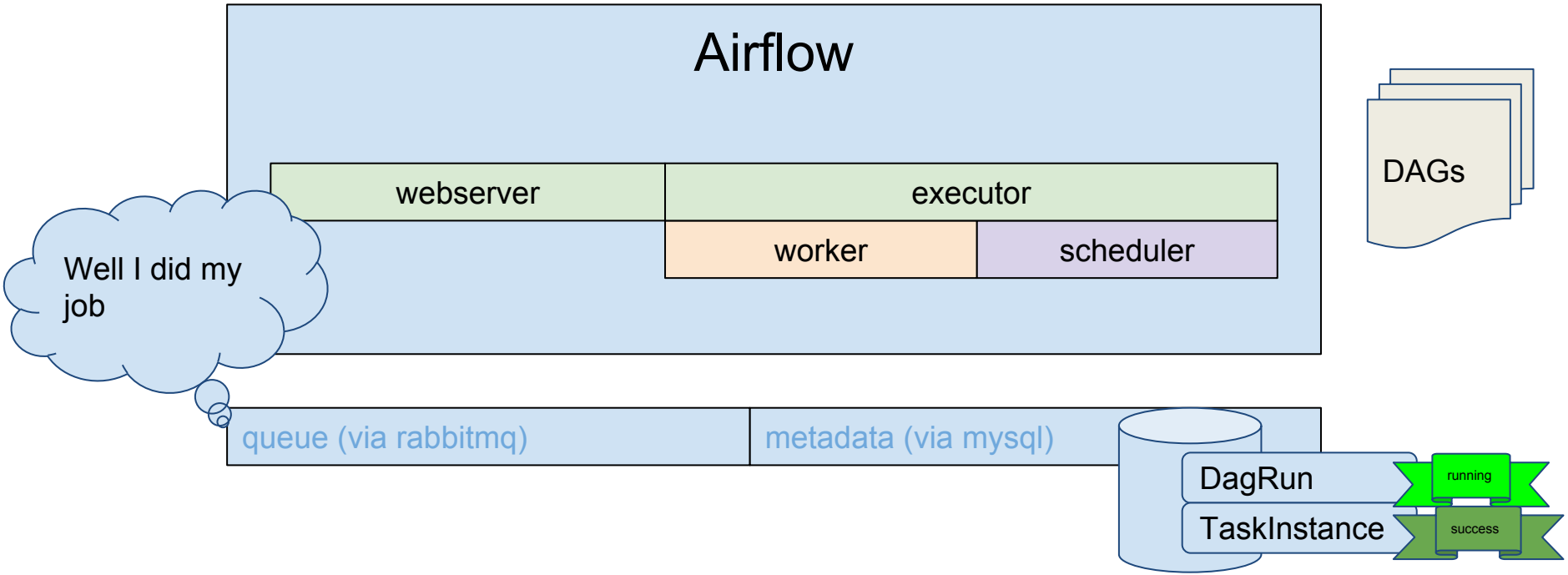
Let's talk about airflow services



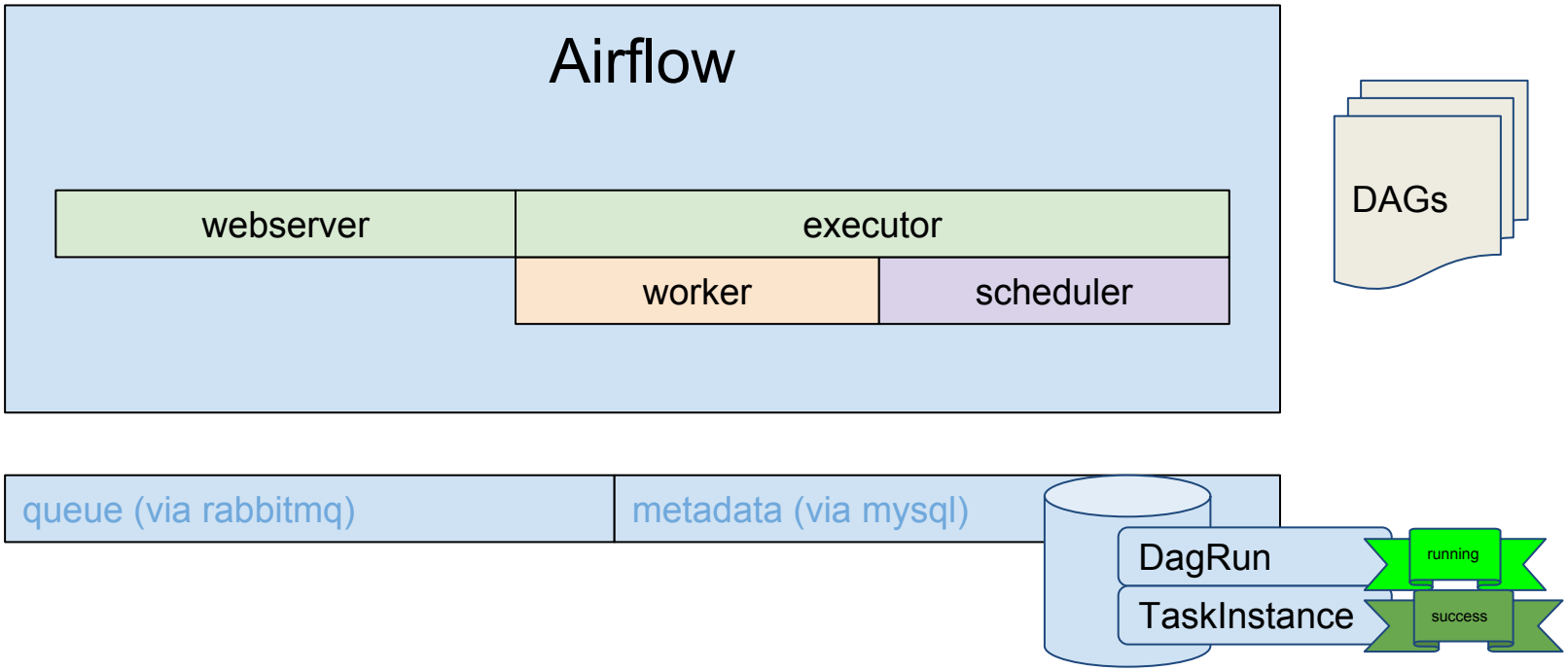
Let's talk about airflow services



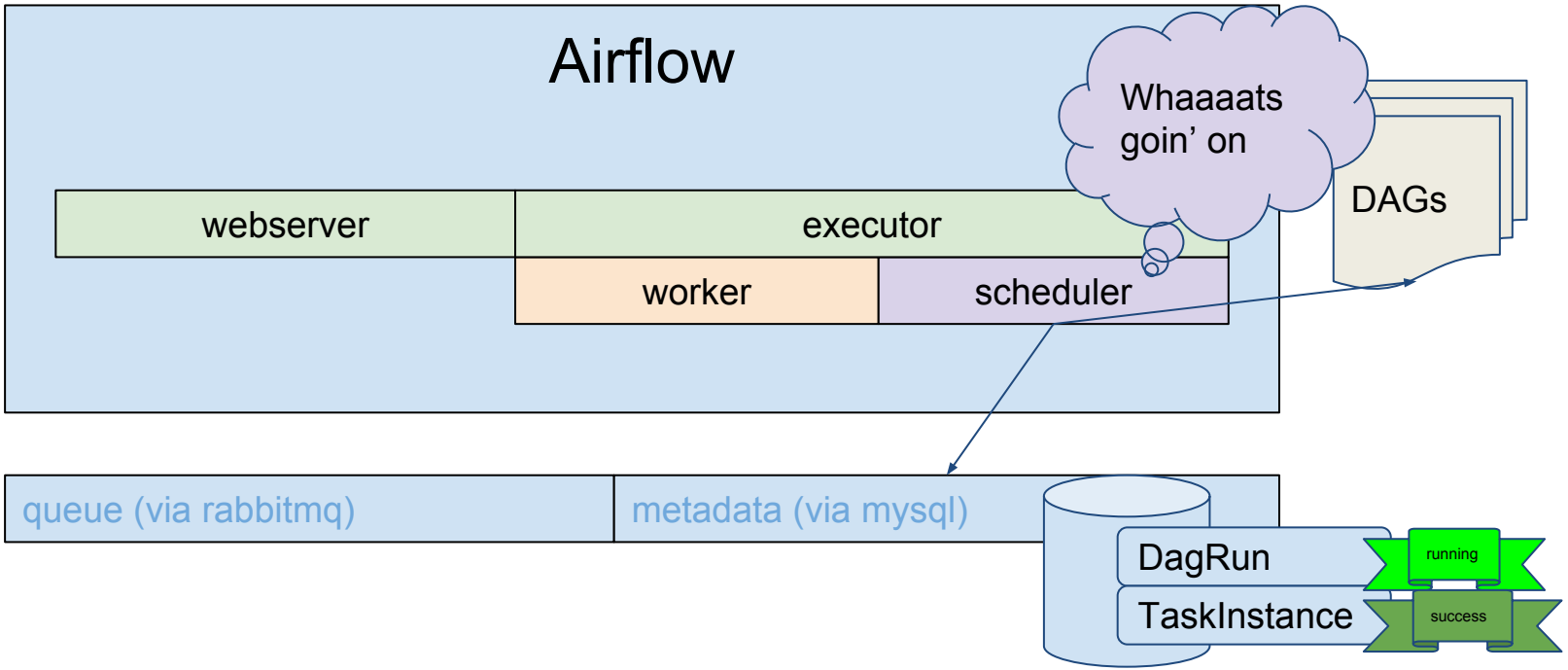
Let's talk about airflow services



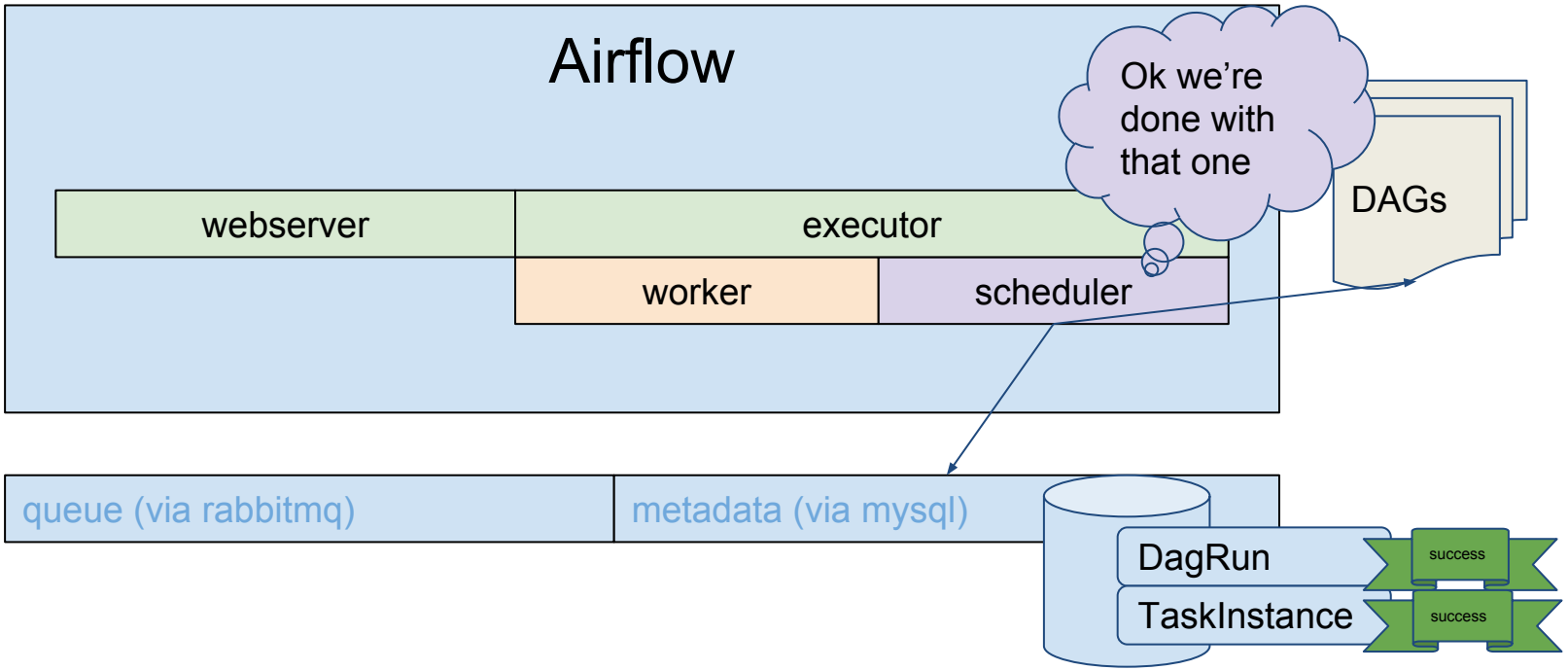
Let's talk about airflow services



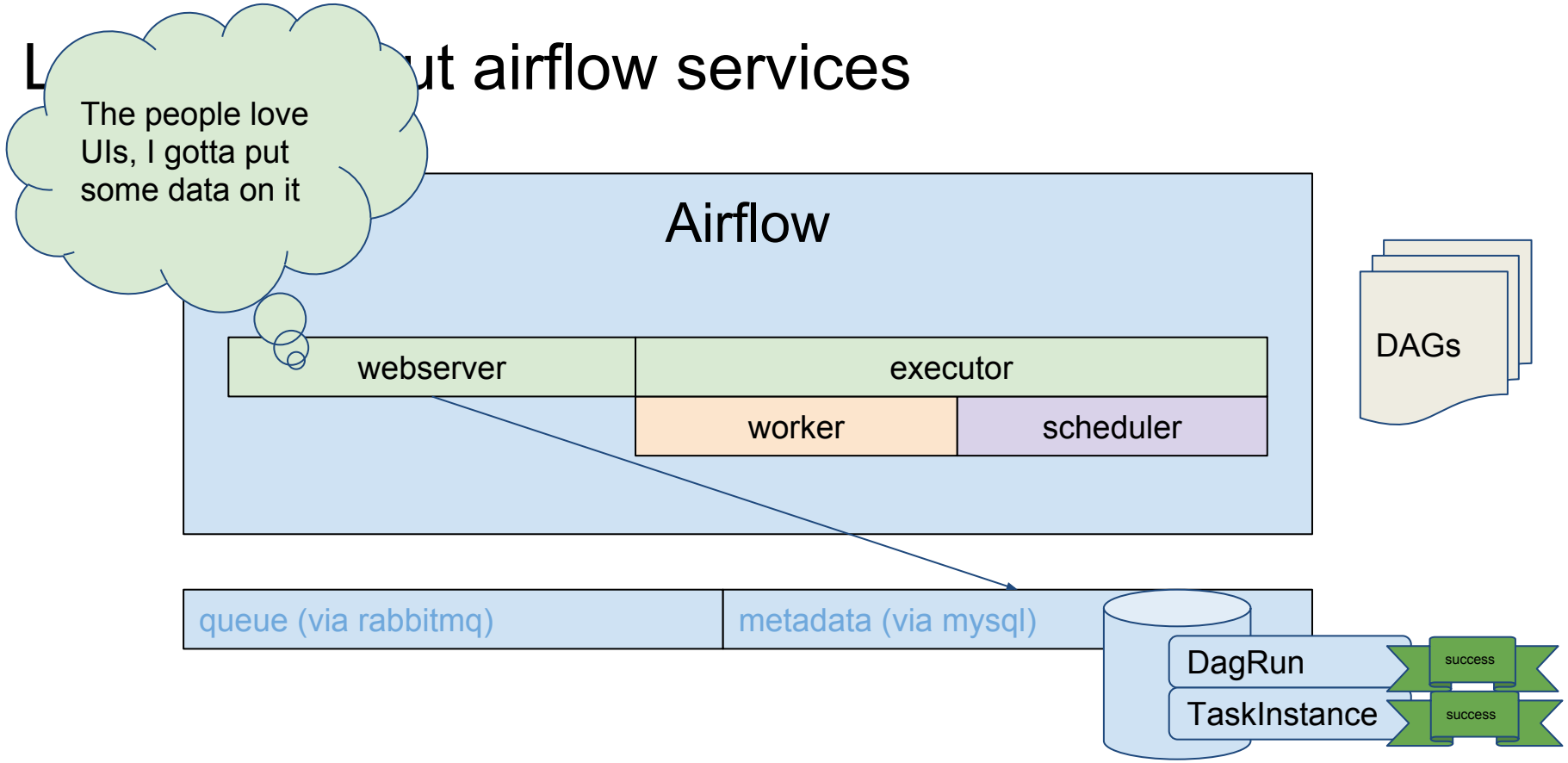
Let's talk about airflow services



Let's talk about airflow services



L ut airflow services



Alerting is fun

SLAs
Callbacks
Email on retry/failure/success
Timeouts
SlackOperator

Configuration abounds

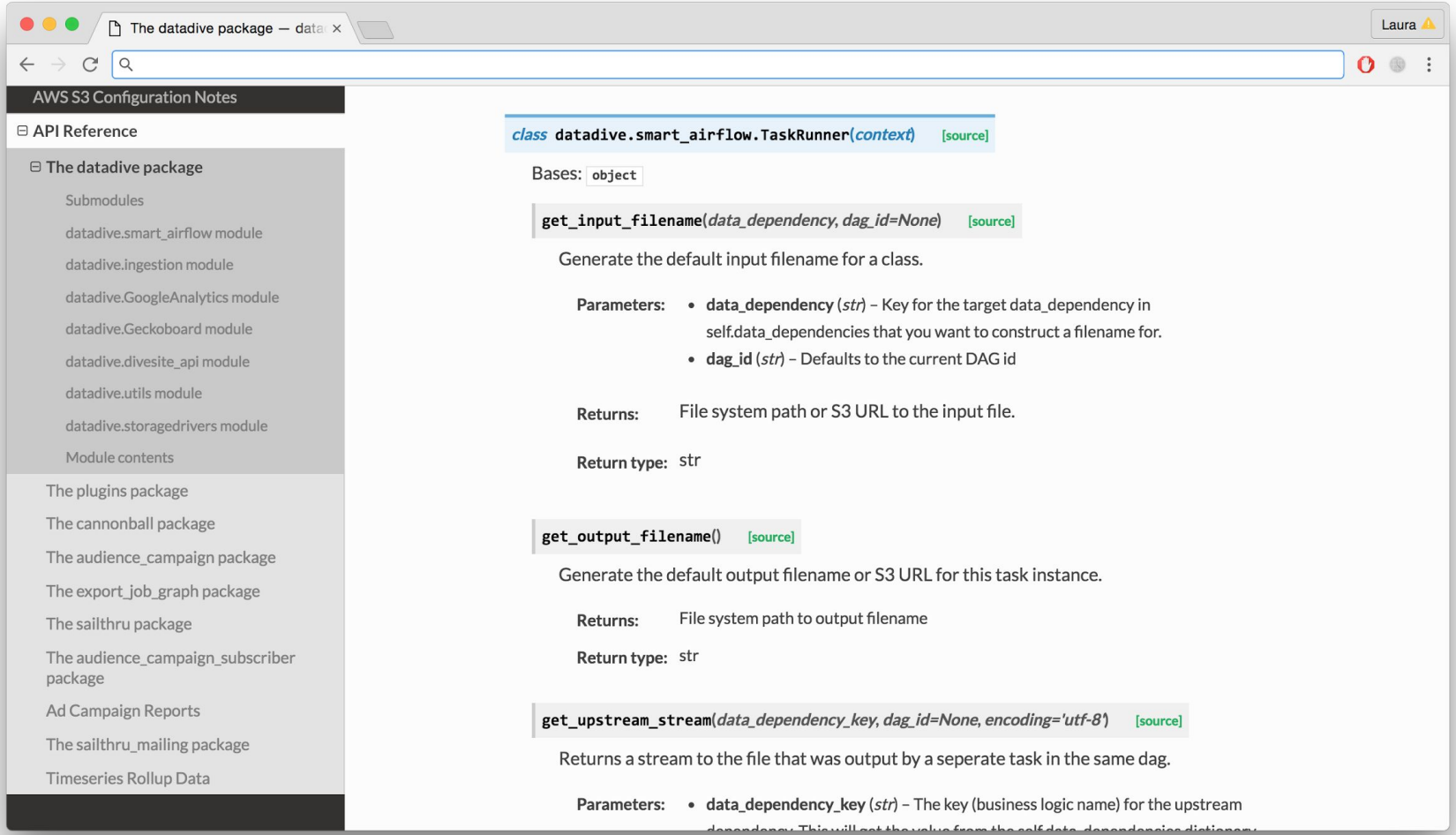
Pools
Queues
Max_active_dag_runs
Max_concurrency
DAGs on/off
Retries

Flexibility

- Operators
  - PythonOperator, BashOperator
  - TriggerDagRunOperator, BranchOperator
  - EmailOperator, MySqlOperator, S3ToHiveTransfer
- Sensors
  - ExternalTaskSensor
  - HttpSensor, S3KeySensor
- Extending your own operators and sensors
  - smart_airflow.DivePythonOperator

smart-airflow

- Airflow doesn't support much data transfer between tasks out of the box
  - only small pieces of data via XCom
- But we liked the file dependency/target concept of checkpoints to cache data transformations to both save time and provide transparency
- `smart-airflow` is a plugin to Airflow that supports local file system or S3-backed intermediate artifact storage
- It leverages Airflow concepts to make file location predictable
  - dag_id/task_id/execution_date


The datadive package — data
Laura
AWS S3 Configuration Notes
API Reference
The datadive package
Submodules
datadive.smart_airflow module
datadive.ingestion module
datadive.GoogleAnalytics module
datadive.Geckoboard module
datadive.divesite_api module
datadive.utils module
datadive.storagedrivers module
Module contents
The plugins package
The cannonball package
The audience_campaign package
The export_job_graph package
The sailthru package
The audience_campaign_subscriber package
Ad Campaign Reports
The sailthru_mailing package
Timeseries Rollup Data
class datadive.smart_airflow.TaskRunner(context) [source]
Bases: object
get_input_filename(data_dependency, dag_id=None) [source]
Generate the default input filename for a class.
Parameters: • data_dependency (str) – Key for the target data_dependency in self.data_dependencies that you want to construct a filename for.
• dag_id (str) – Defaults to the current DAG id
Returns: File system path or S3 URL to the input file.
Return type: str
get_output_filename() [source]
Generate the default output filename or S3 URL for this task instance.
Returns: File system path to output filename
Return type: str
get_upstream_stream(data_dependency_key, dag_id=None, encoding='utf-8') [source]
Returns a stream to the file that was output by a seperate task in the same dag.
Parameters: • data_dependency_key (str) – The key (business logic name) for the upstream

Our `smart-airflow` backed ETL paradigm

1. Make each task as small as possible while maintaining readability
2. Preserve output for each task as a file-based intermediate artifact in a format that is consumable by its dependent task
3. Avoid finalizing artifacts for as long as possible (e.g. update a database table as the last and simplest step in a DAG)


Shut up and take my money!

Setting up Airflow at your organization

- `pip install airflow` to get started - you can instantly get started up with
  - `sqlite` metadata database
  - `SequentialExecutor`
  - Included example DAGs
- Use `puckel/docker-airflow` to get started quickly with
  - MySQL metadata database
  - `CeleryExecutor`
  - Celery Flower
  - RabbitMQ messaging backend with Management plugin
- `upstart` and `systemd` templates available through the `apache/incubator-airflow` repository

Tips, tricks, and gotchas for Airflow

- Minimize your dev environment with `SequentialExecutor` and use `airflow test {dag_id} {task_id} {execution_date}` in early development to test tasks
- To test your DAG with the scheduler, utilize the `@once` schedule_interval and clear the `DagRun`s and `TaskInstance`s between tests with `airflow clear` or the fancy schmancy UI
- Don’t bother with the nascent plugin system, just package your custom operators with your DAGs for deployment
- No built in log rotation - default logging is pretty verbose and if you add your own, this might get surprisingly large. As of Airflow 1.7 you can back them up to S3 with simple configuration

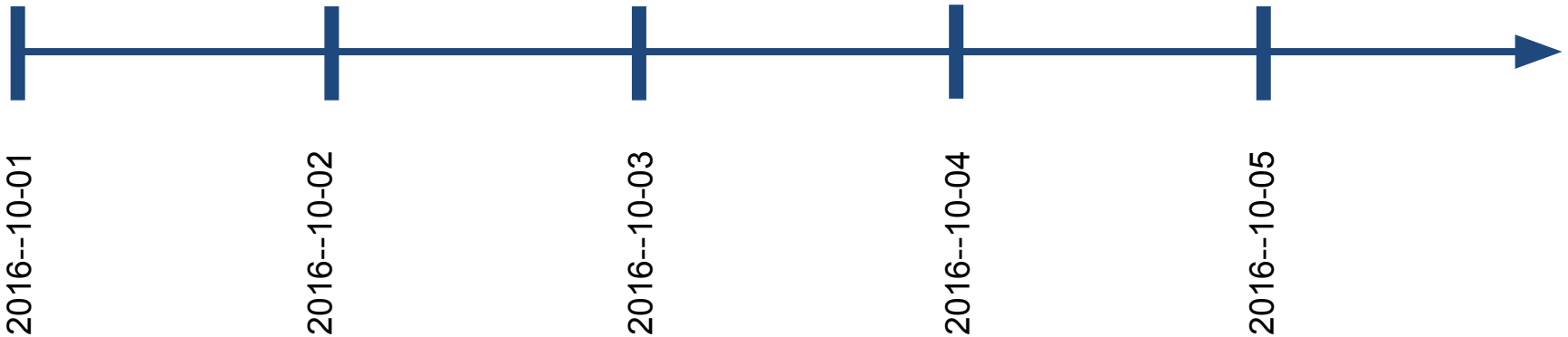
Tips, tricks, and gotchas for Airflow

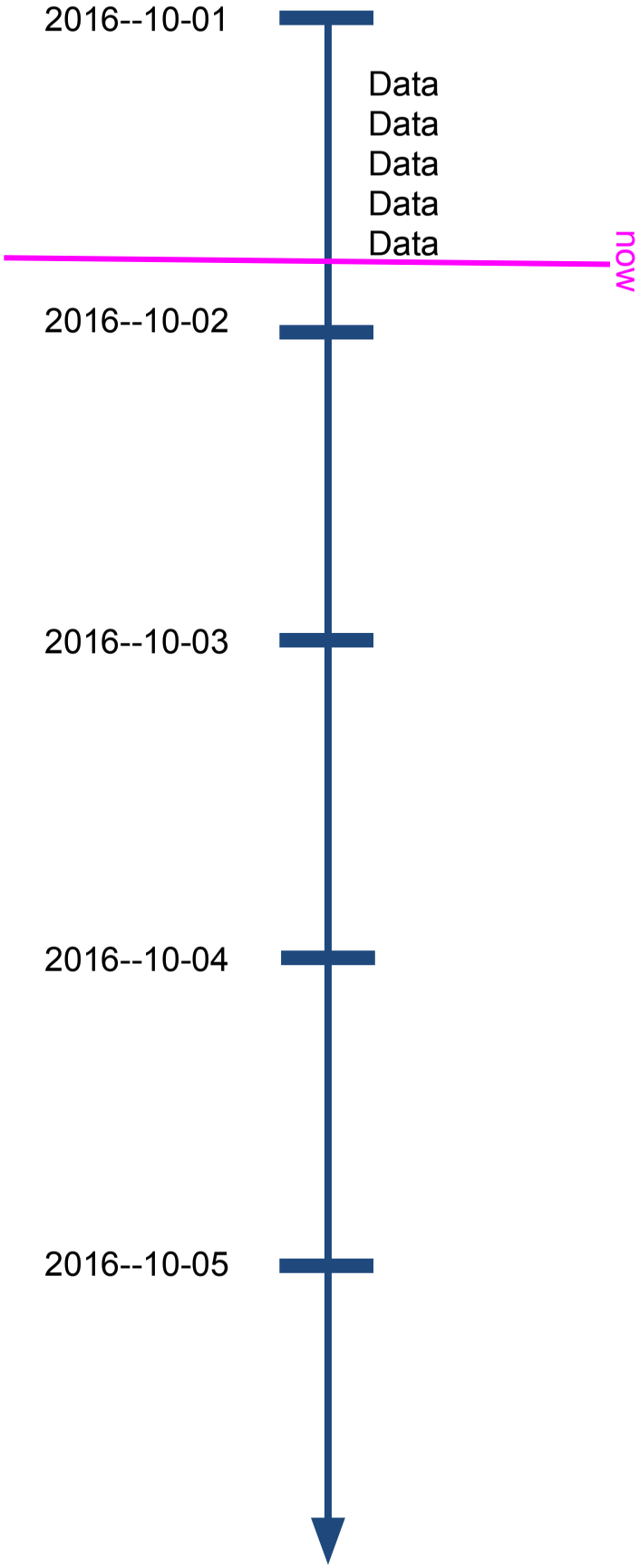
- We have about ~1300 tasks across 8 active DAGs and 27 worker processes on an m4.xlarge AWS EC2 instance.
  - We utilize pools (which have come a long way since their buggy inception) to manage resources
- Consider using queues if you have disparate types of work to do
  - our tasks are currently 100% Python but you could support scripts in any other language by redirecting those messages to worker servers with the proper executables and dependencies installed
- Your tasks must be idempotent; we're using retries here

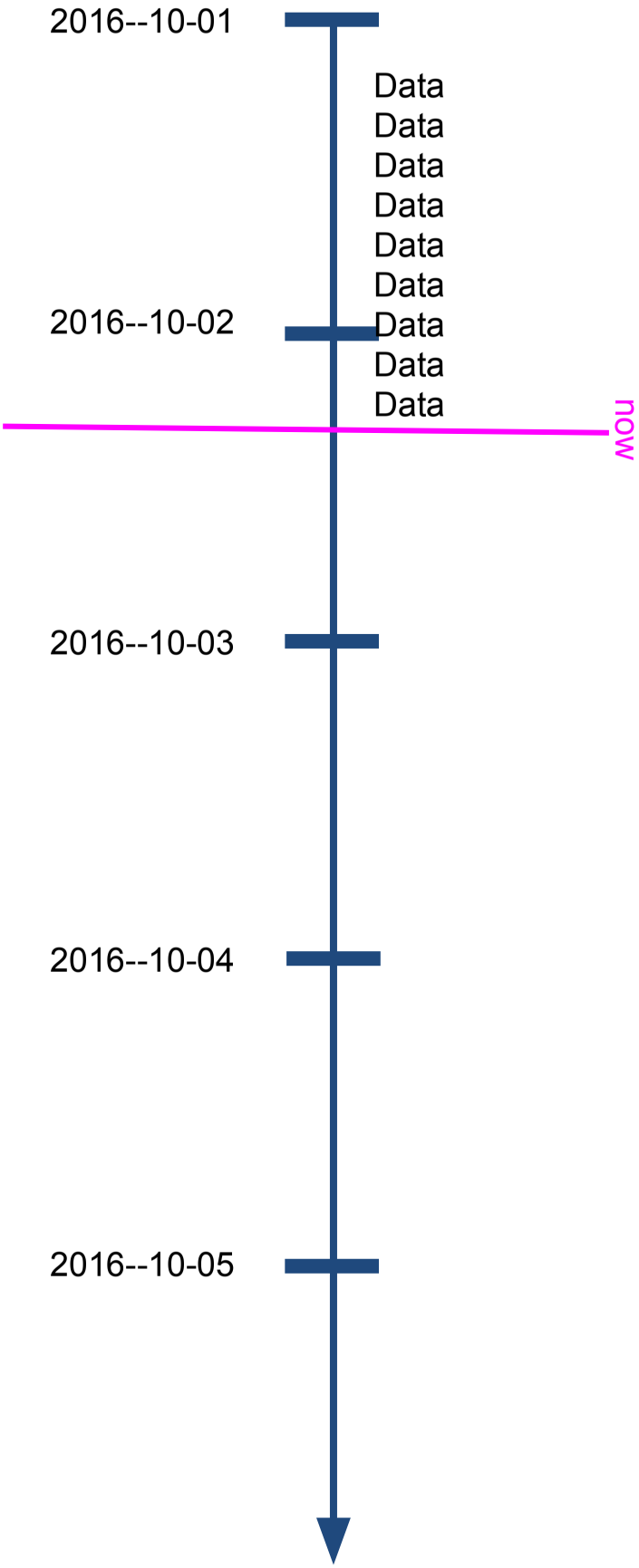
Let’s talk about TIME TRAVEL

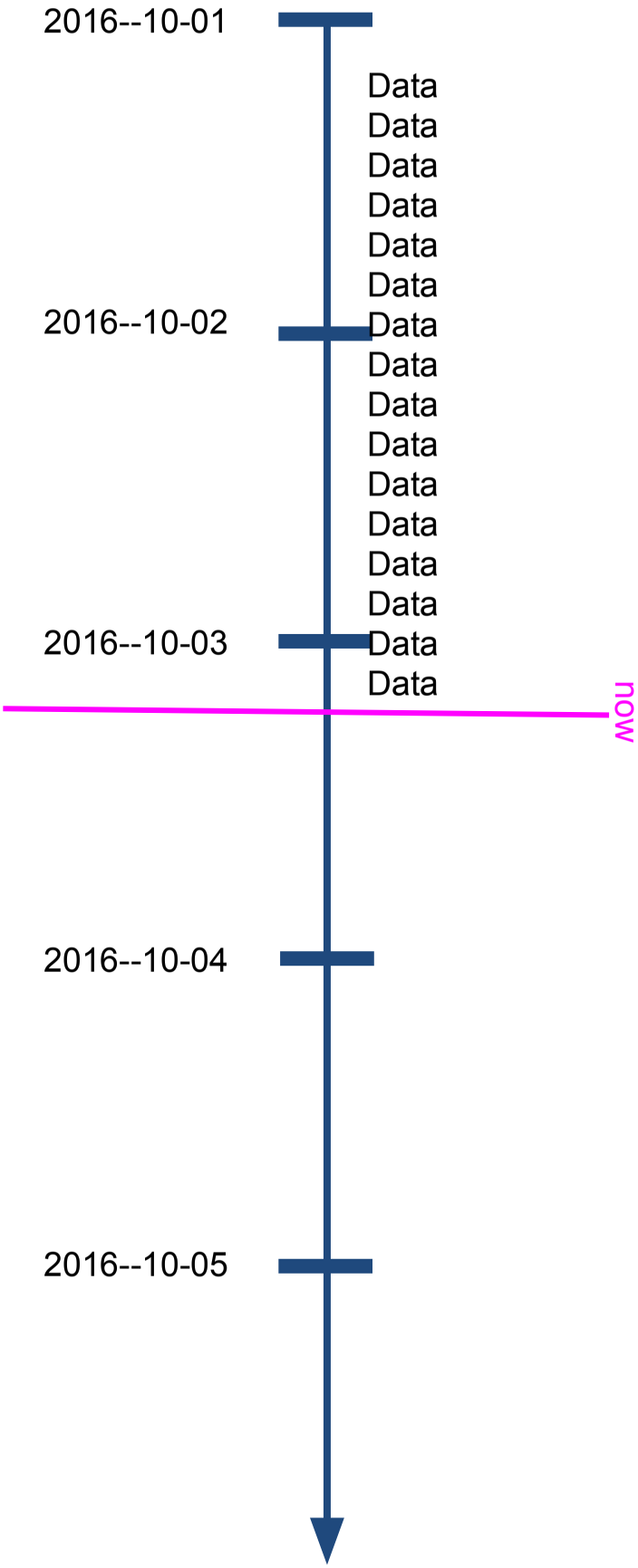
```
schedule_interval = timedelta(day=1)
```

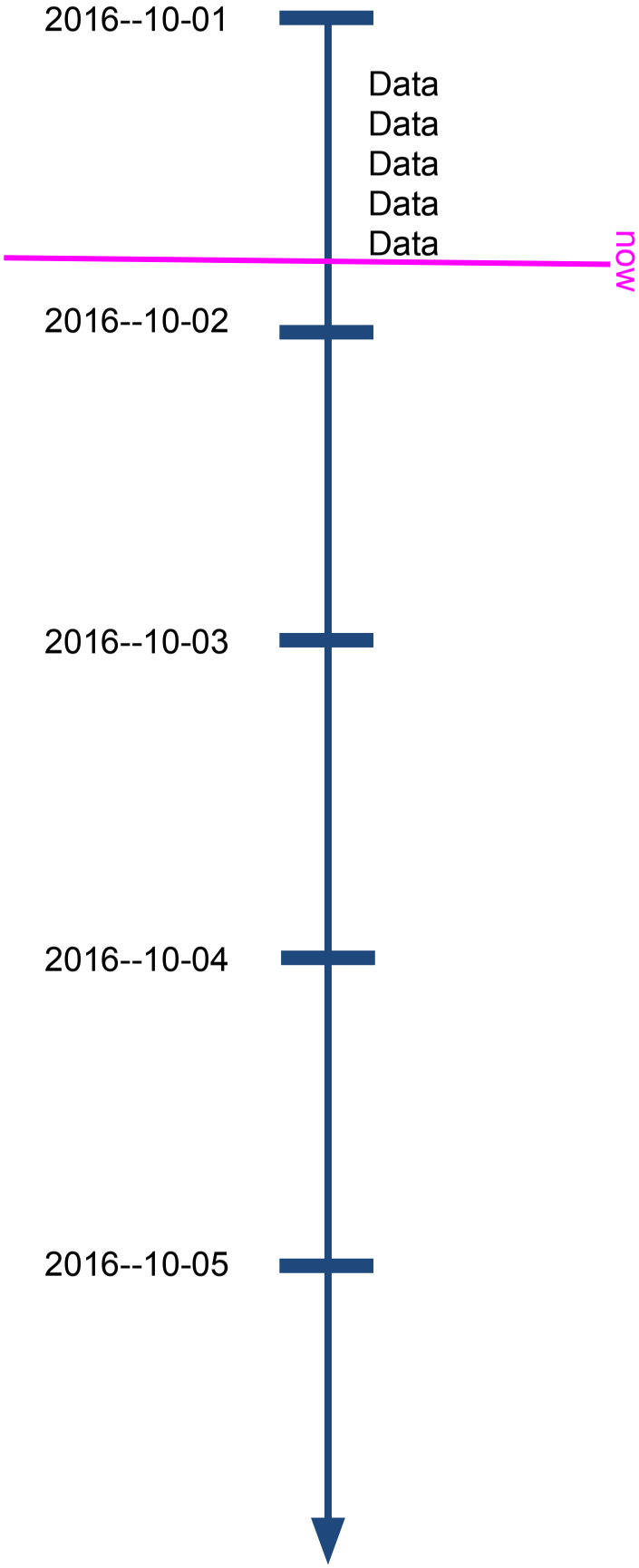
execution_date



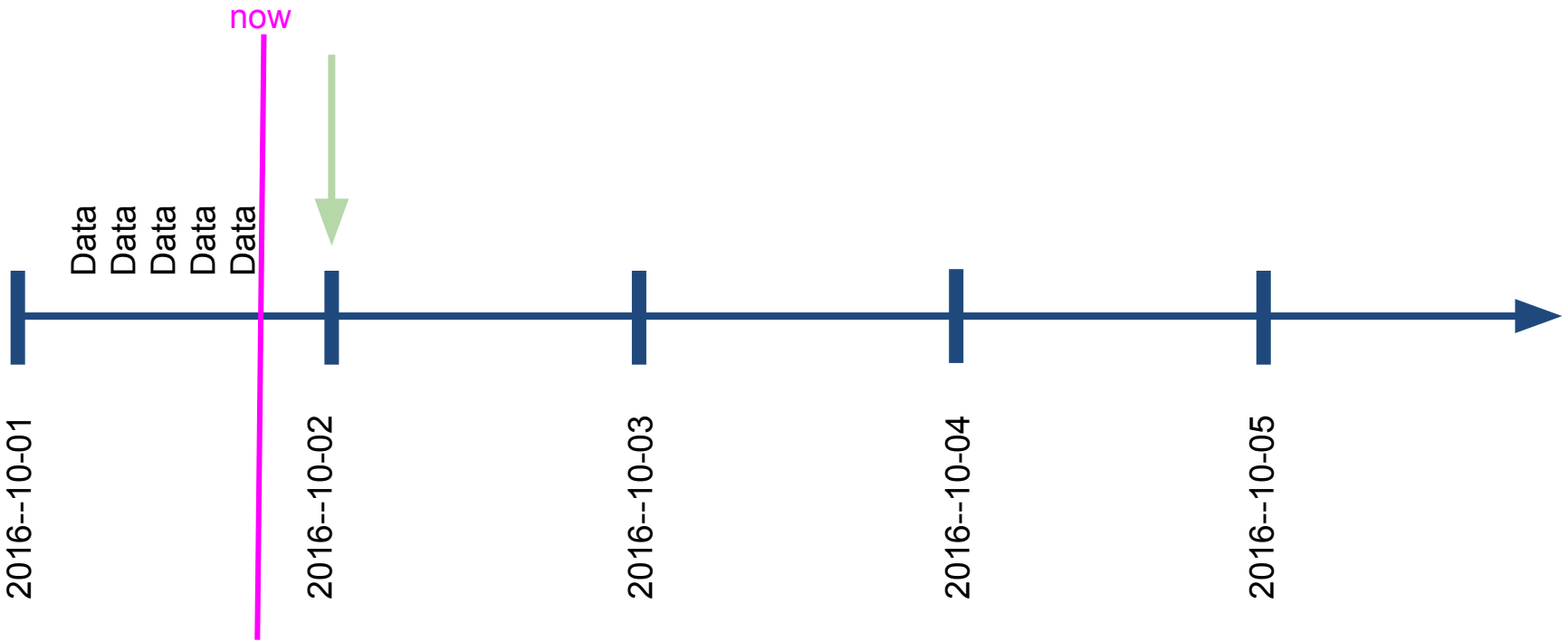




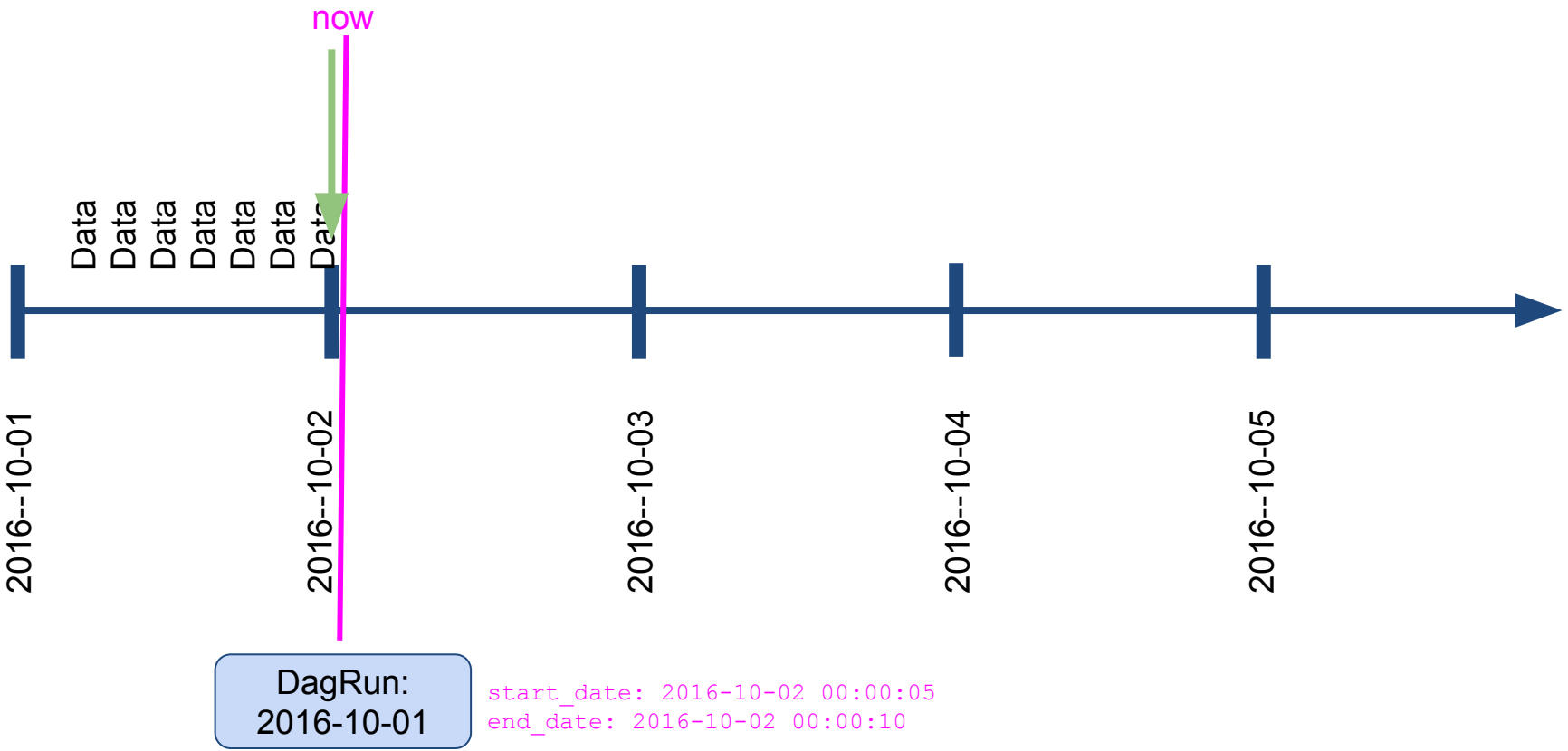




"allowable" = `execution_date + schedule_interval`

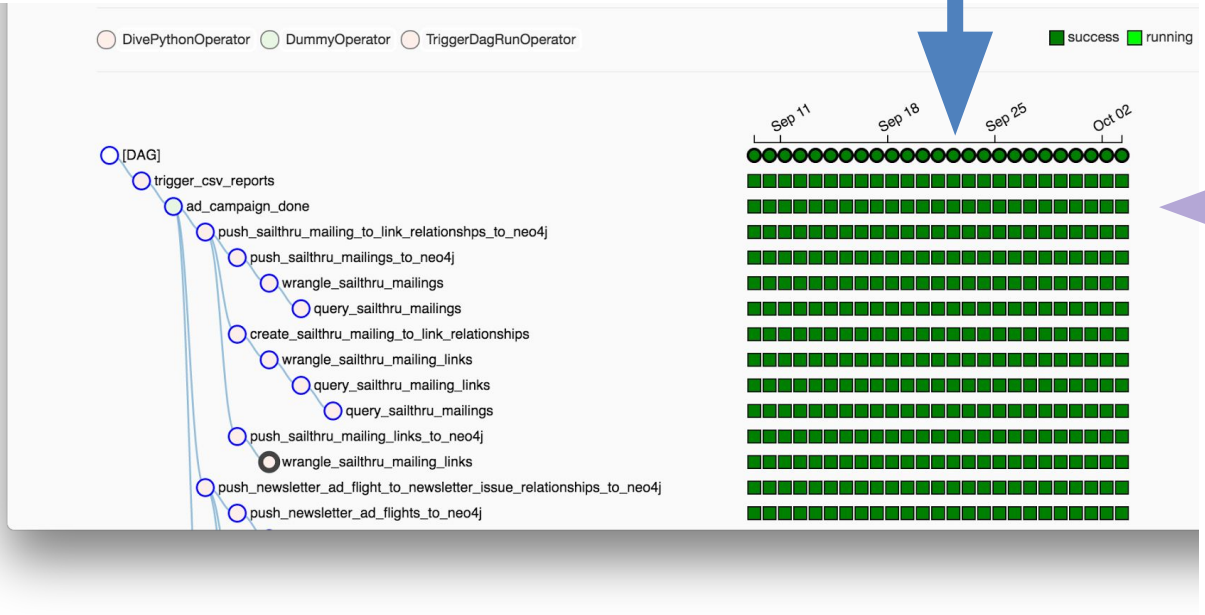


execution_date != start_date

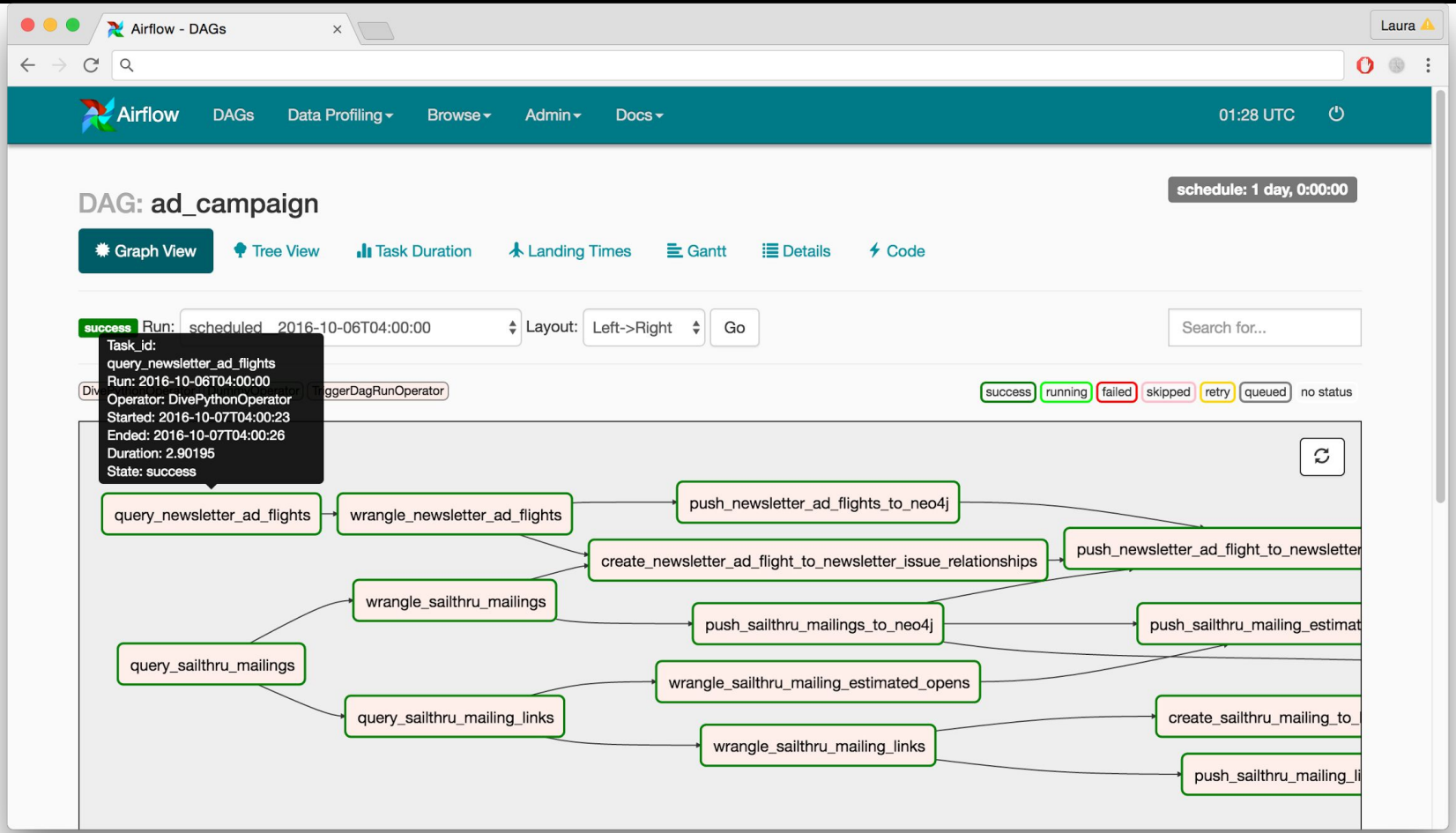


It’s the same for TaskInstances

DagRuns:
DAG by “time”



TaskInstances:
Task by DagRun

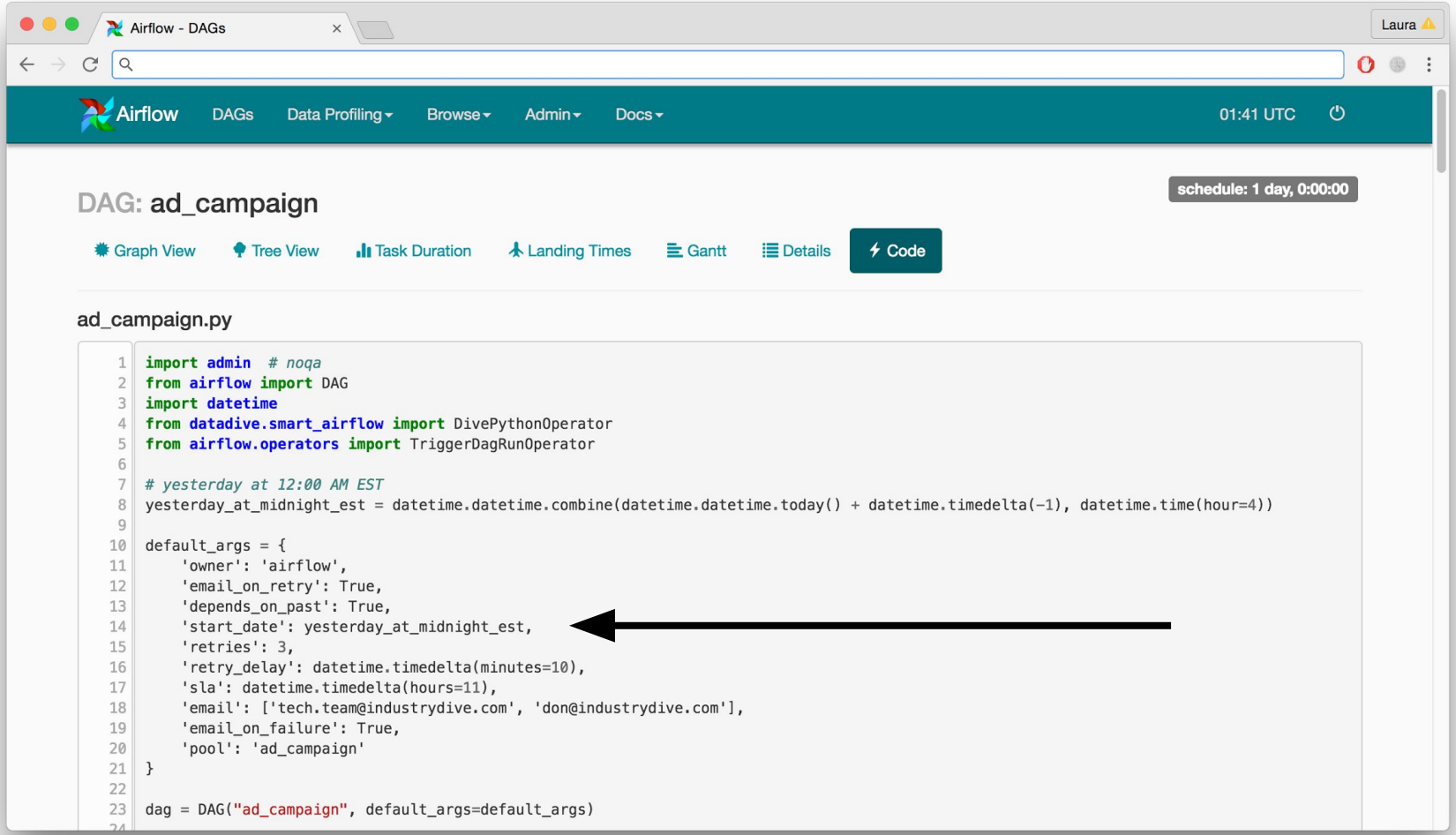

Airflow - DAGs
Laura
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
01:28 UTC
DAG: ad_campaign
schedule: 1 day, 0:00:00
Graph View
Tree View
Task Duration
Landing Times
Gantt
Details
Code
success
Run: scheduled__2016-10-06T04:00:00
Layout: Left->Right
Go
Search for...
Task_id:
query_newsletter_ad_flights
Run: 2016-10-06T04:00:00
Operator: DivePythonOperator
Started: 2016-10-07T04:00:23
Ended: 2016-10-07T04:00:26
Duration: 2.90195
State: success
success
running
failed
skipped
retry
queued
no status
query_newsletter_ad_flights
wrangle_newsletter_ad_flights
push_newsletter_ad_flights_to_neo4j
create_newsletter_ad_flight_to_newsletter_issue_relationships
wrangle_sailthru_mailings
push_sailthru_mailings_to_neo4j
query_sailthru_mailings
wrangle_sailthru_mailing_estimated_opens
query_sailthru_mailing_links
wrangle_sailthru_mailing_links

`execution_date != start_date`

but...

`start_date != start_date`

...sorta


Airflow - DAGs
Laura
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
01:41 UTC
DAG: ad_campaign
schedule: 1 day, 0:00:00
Graph View
Tree View
Task Duration
Landing Times
Gantt
Details
Code
ad_campaign.py
import admin # noqa
from airflow import DAG
import datetime
from datadive.smart_airflow import DivePythonOperator
from airflow.operators import TriggerDagRunOperator
# yesterday at 12:00 AM EST
yesterday_at_midnight_est = datetime.datetime.combine(datetime.datetime.today() + datetime.timedelta(-1), datetime.time(hour=4))
default_args = {
'owner': 'airflow',
'email_on_retry': True,
'depends_on_past': True,
'start_date': yesterday_at_midnight_est,
'retries': 3,
'retry_delay': datetime.timedelta(minutes=10),
'sla': datetime.timedelta(hours=11),
'email': ['tech.team@industrydive.com', 'don@industrydive.com'],
'email_on_failure': True,
'pool': 'ad_campaign'
}
dag = DAG("ad_campaign", default_args=default_args)

Still don’t get it?

- Time travel yourself!!!
  - Rewind
  - Ask me I’m friendly 👻
  - Google Groups, Gitter, Airflow docs, dev mailing list archives

Industry DIVE

Thank you!

Questions?

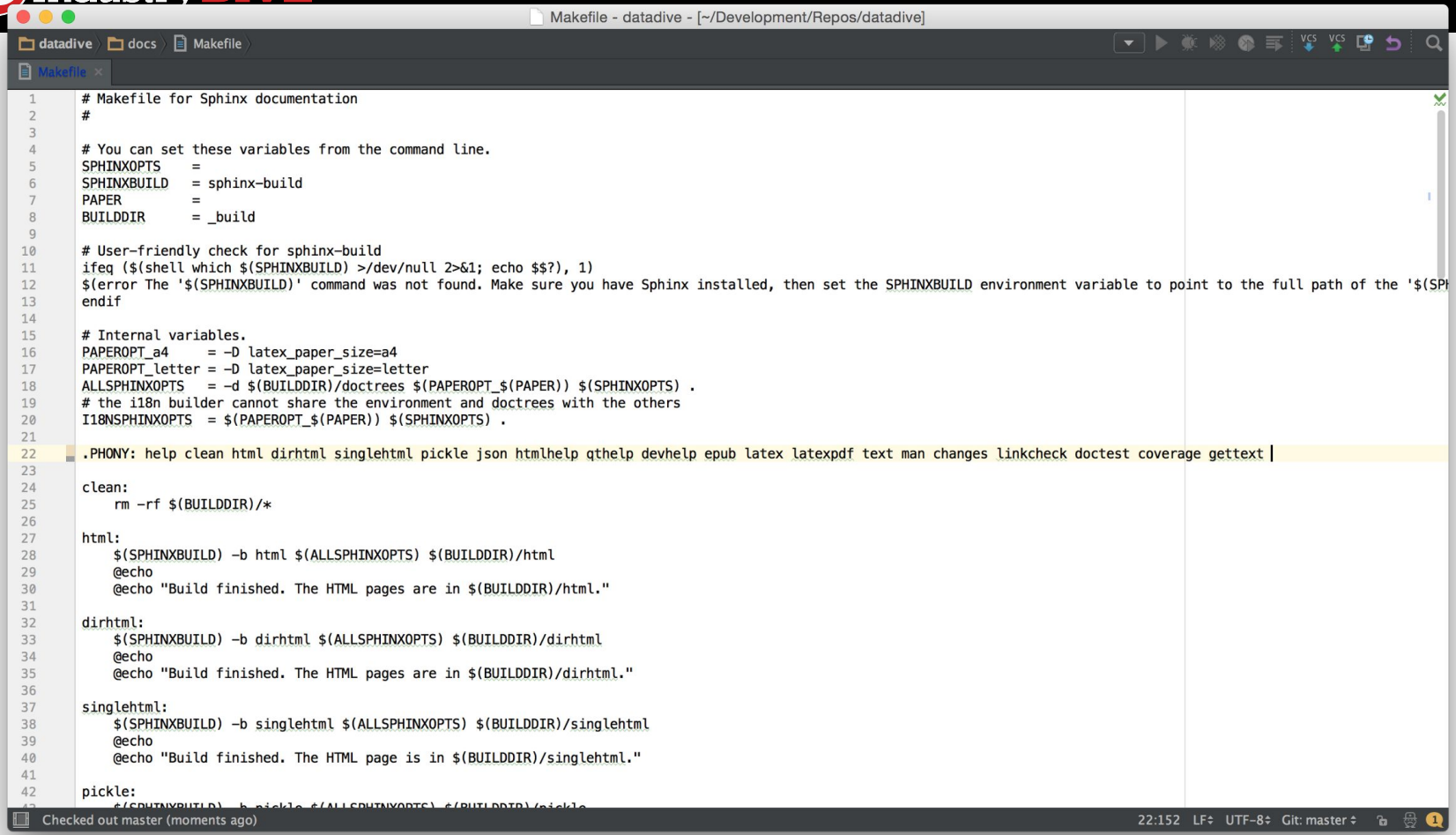
PS there's an appendix

Appendix

More details on the pipeline tools not covered in depth from an earlier draft of this talk 😊

Make

- Originally/often used to compile source code
- Defines
  - targets and any prerequisites, which are potential file paths
  - recipes that can be executed by your shell environment
- Specify batch workflows in stages with file storage as atomic intermediates (local FS only).
- Rebuilding logic based on target existence and other file dependencies (“prerequisites”) existence/metadata.
- Supports basic conditionals and parallelism


Makefile - datadive - [~/Development/Repos/datadive]
datadive
docs
Makefile
Makefile
Makefile for Sphinx documentation
#
You can set these variables from the command line.
SPHINXOPTS =
SPHINXBUILD = sphinx-build
PAPER =
BUILDDIR = _build
# User-friendly check for sphinx-build
ifeq ($(shell which $(SPHINXBUILD) >/dev/null 2>&1; echo $$?), 1)
$(error The '$(SPHINXBUILD)' command was not found. Make sure you have Sphinx installed, then set the SPHINXBUILD environment variable to point to the full path of the '$(SPH
endif
# Internal variables.
PAPEROPT_a4 = -D latex_paper_size=a4
PAPEROPT_letter = -D latex_paper_size=letter
ALLSPHINXOPTS = -d $(BUILDDIR)/doctrees $(PAPEROPT_$(PAPER)) $(SPHINXOPTS) .
# the i18n builder cannot share the environment and doctrees with the others
I18NSPHINXOPTS = $(PAPEROPT_$(PAPER)) $(SPHINXOPTS) .
.PHONY: help clean html dirhtml singlehtml pickle json htmlhelp qthelp devhelp epub latex latexpdf text man changes linkcheck doctest coverage gettext
clean:
rm -rf $(BUILDDIR)/*
html:
$(SPHINXBUILD) -b html $(ALLSPHINXOPTS) $(BUILDDIR)/html
@echo
@echo "Build finished. The HTML pages are in $(BUILDDIR)/html."
dirhtml:
$(SPHINXBUILD) -b dirhtml $(ALLSPHINXOPTS) $(BUILDDIR)/dirhtml
@echo
@echo "Build finished. The HTML pages are in $(BUILDDIR)/dirhtml."
singlehtml:
$(SPHINXBUILD) -b singlehtml $(ALLSPHINXOPTS) $(BUILDDIR)/singlehtml
@echo
@echo "Build finished. The HTML page is in $(BUILDDIR)/singlehtml."
pickle:
Checked out master (moments ago)
22:152
LF
UTF-8
Git: master

```
some_data.tsv :
 hive -e "select * from my_data" > some_data.tsv
other_data.json :
 curl example.com/some_data_file.json > other_data.json
```

```
some_data.csv : some_data.tsv src/tsv_to_csv.py
 src/tsv_to_csv.py some_data.tsv some_data.csv
other_data.csv : other_data.json src/json_to_csv.py
 src/json_to_csv.py other_data.json other_data.csv
```

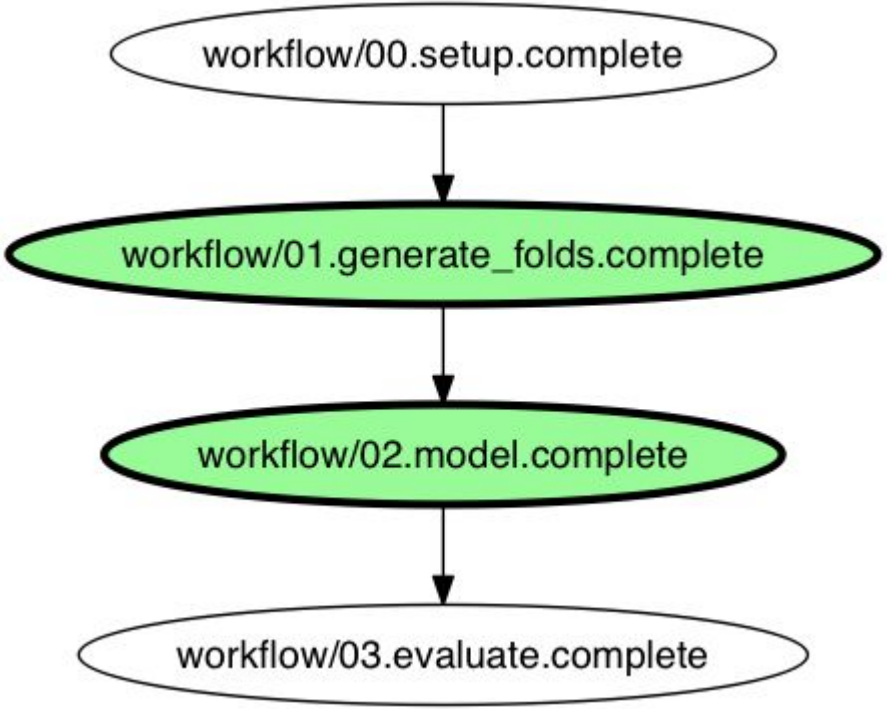
From http://blog.kaggle.com/2012/10/15/make-for-data-scientists/

Drake

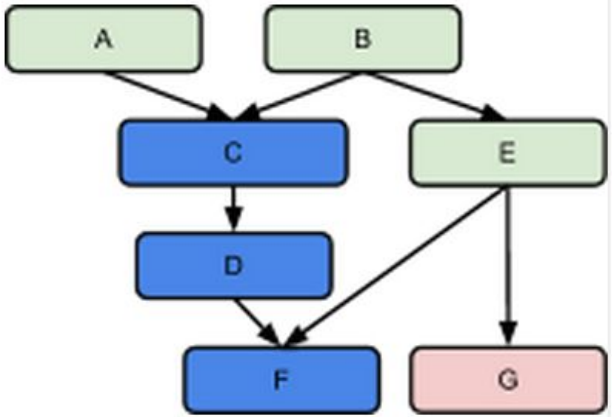
- “Make for data”
- Specify batch workflows in stages with file storage as atomic intermediates (backend support for local FS, S3, HDFS, Hive)
- Early support for alternate branches and branch merging
- Smart rebuilding against targets and target metadata, or a quite sophisticated command line specification system
- Parallel execution
- Workflow graph generation
- Expanding protocols support to facilitate common tasks like python, HTTP GET, etc

```
clusters <- filtered.csv
  java -jar resolve.jar $INPUT $OUTPUT

; Apply UUIDs according to clusters
cpg.json <- filtered.csv, clusters
  ruby generate_uuids.rb --data=$INPUT1 --clusters=$INPUT2 --output=$OUTPUT

; Push to production servers
PUBLISH=/apps/extract/v3_data_for_materialization/cpg

; this target has no output files, but just a tag to reference it by
; ("push"). it will always run if selected, but only after confirmation
; from the user
%push <- cpg.json [+confirm]
  hadoop fs -cp $INPUT $[PUBLISH]

; Generate some statistics (can also be referred by "stats")
report.html, %stats <- cpg.json
  datatool $INPUT --columns=brand,name --report=$OUTPUT
```

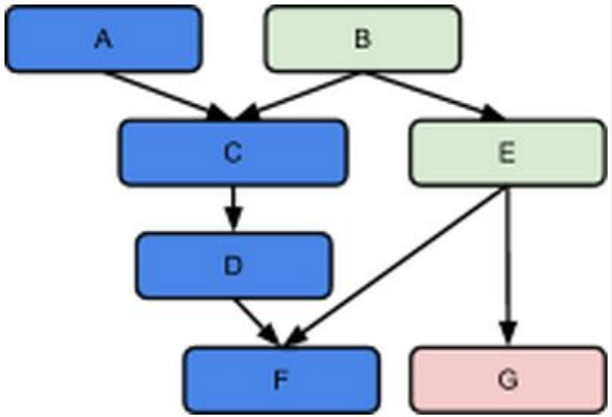
```
drake --graph +workflow/02.model.complete
```



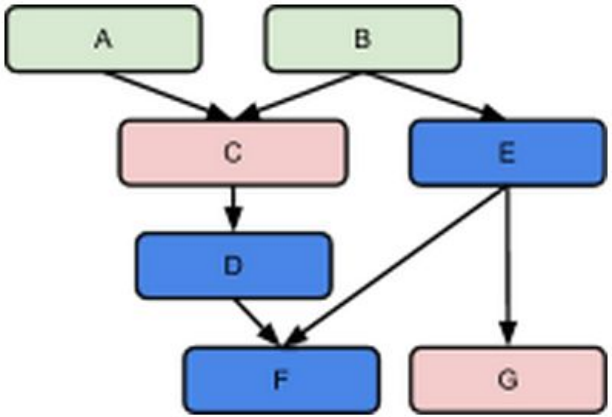
`drake ^A`



`drake +^A`



`drake +F -C`



Pydoit

- “doit comes from the idea of bringing the power of build-tools to execute any kind of task”
- flexible build tool used to glue together pipelines
- Similar to Drake but much more support for Python as opposed to bash.
- Specify actions, file_deps, and targets for tasks
- Smart rebuilding based on target/file_deps metadata, or your own custom logic
- Parallel execution
- Watcher process that triggers based on target/file_deps file changes
- Can define failure/success callbacks against the project

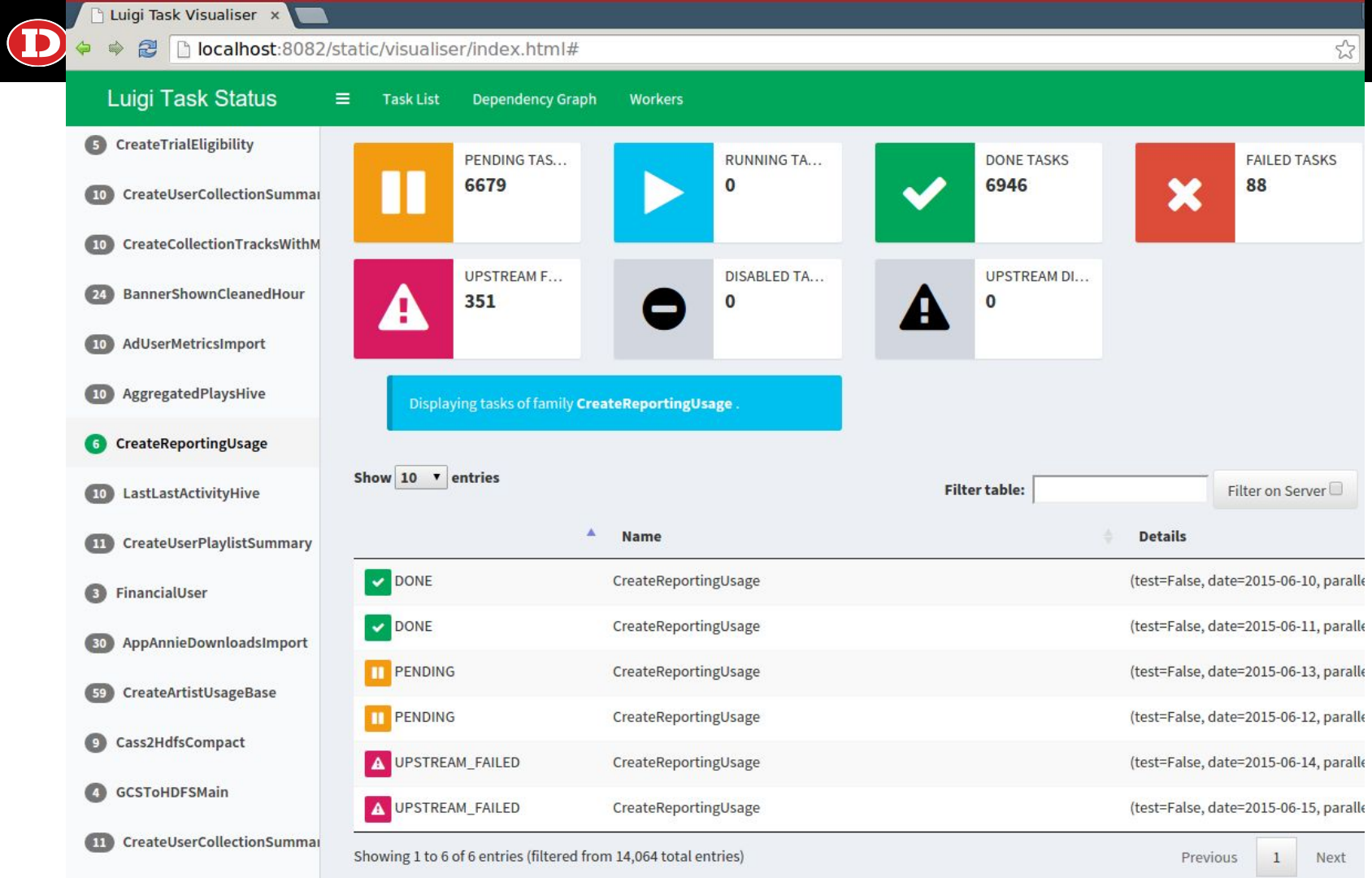
```
def task_example():
    return {
        'actions': ['myscript'],
        'file_dep': ['my_input_file'],
        'targets': ['result_file'],
    }
```

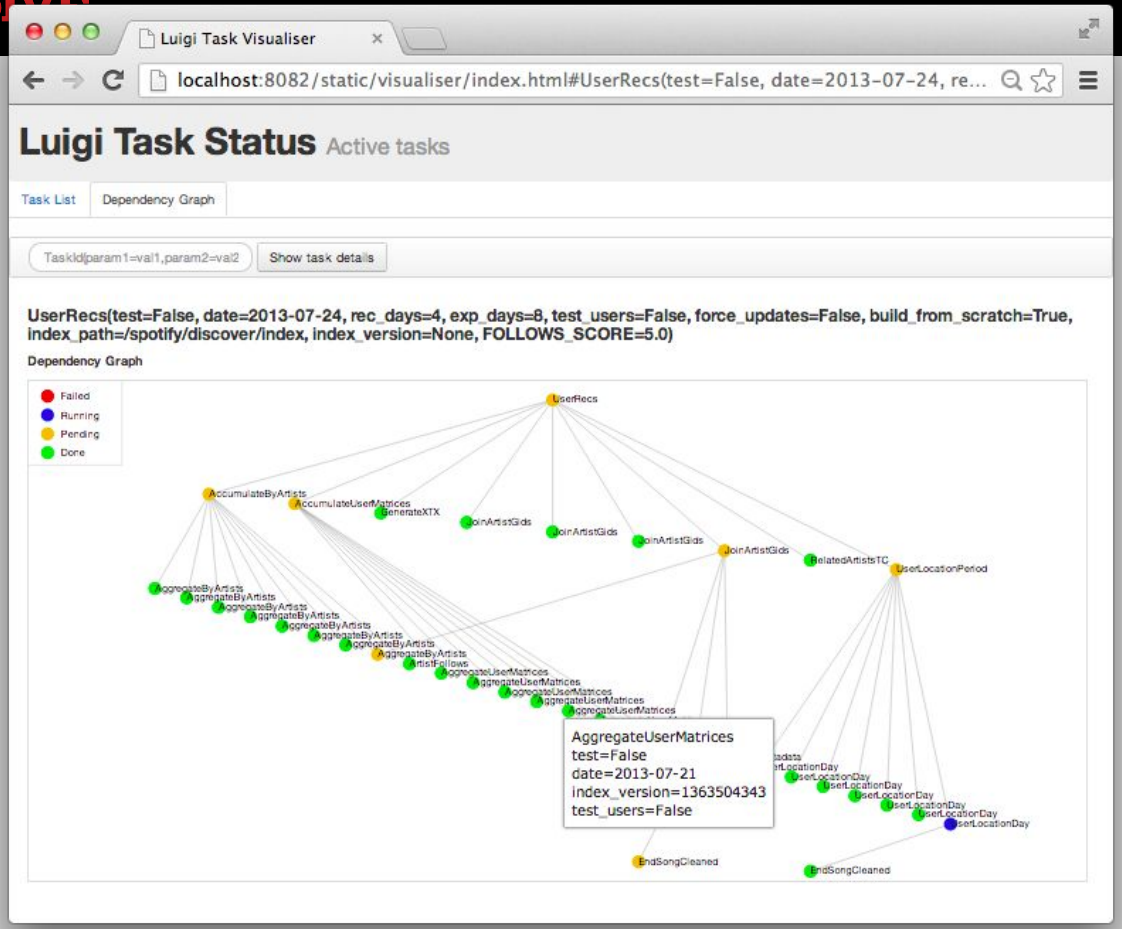
```
def task_hello():
    """hello"""

    def python_hello(targets):
        with open(targets[0], "a") as output:
            output.write("Python says Hello World!!!\n")

    return {
        'actions': [python_hello],
        'targets': ["hello.txt"],
        }
```

Luigi

- Specify batch workflows with different job type classes including postgres.CopyToTable, hadoop.JobTask, PySparkTask
- Specify depedencies with class `requires()` method and record via `output()` method targets against supported backends such as S3, HDFS, local or remote FS, MySQL, Redshift, etc.
- Event system provided to add callbacks to task returns, basic email alerting
- A central task scheduler (`luigid`) that provides a web frontend for task reporting, prevents duplicate task execution, and basic task history browsing
- Requires a separate triggering mechanism to submit tasks to the central scheduler
- `RangeDaily` and `RangeHourly` parameters as a dependency for backfill or recovery from extended downtime

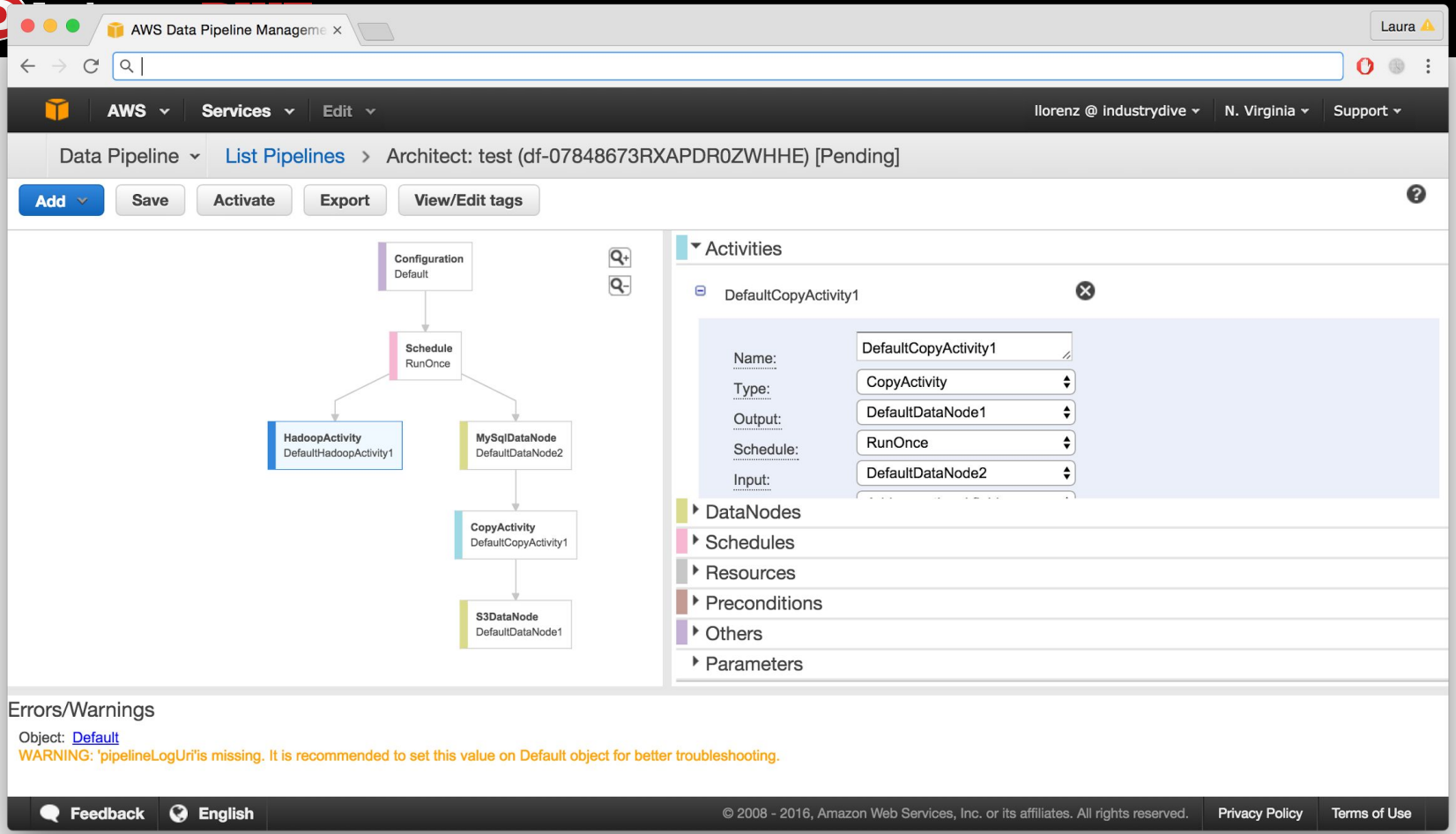
```
class AggregateArtists(luigi.Task):
    date_interval = luigi.DateIntervalParameter()

    def output(self):
        return luigi.LocalTarget("data/artist_streams_%s.tsv" % self.date_interval)

    def requires(self):
        return [Streams(date) for date in self.date_interval]

    def run(self):
        artist_count = defaultdict(int)

        for input in self.input():
            with input.open('r') as in_file:
                for line in in_file:
                    timestamp, artist, track = line.strip().split()
                    artist_count[artist] += 1

        with self.output().open('w') as out_file:
            for artist, count in artist_count.iteritems():
                print >> out_file, artist, count
```

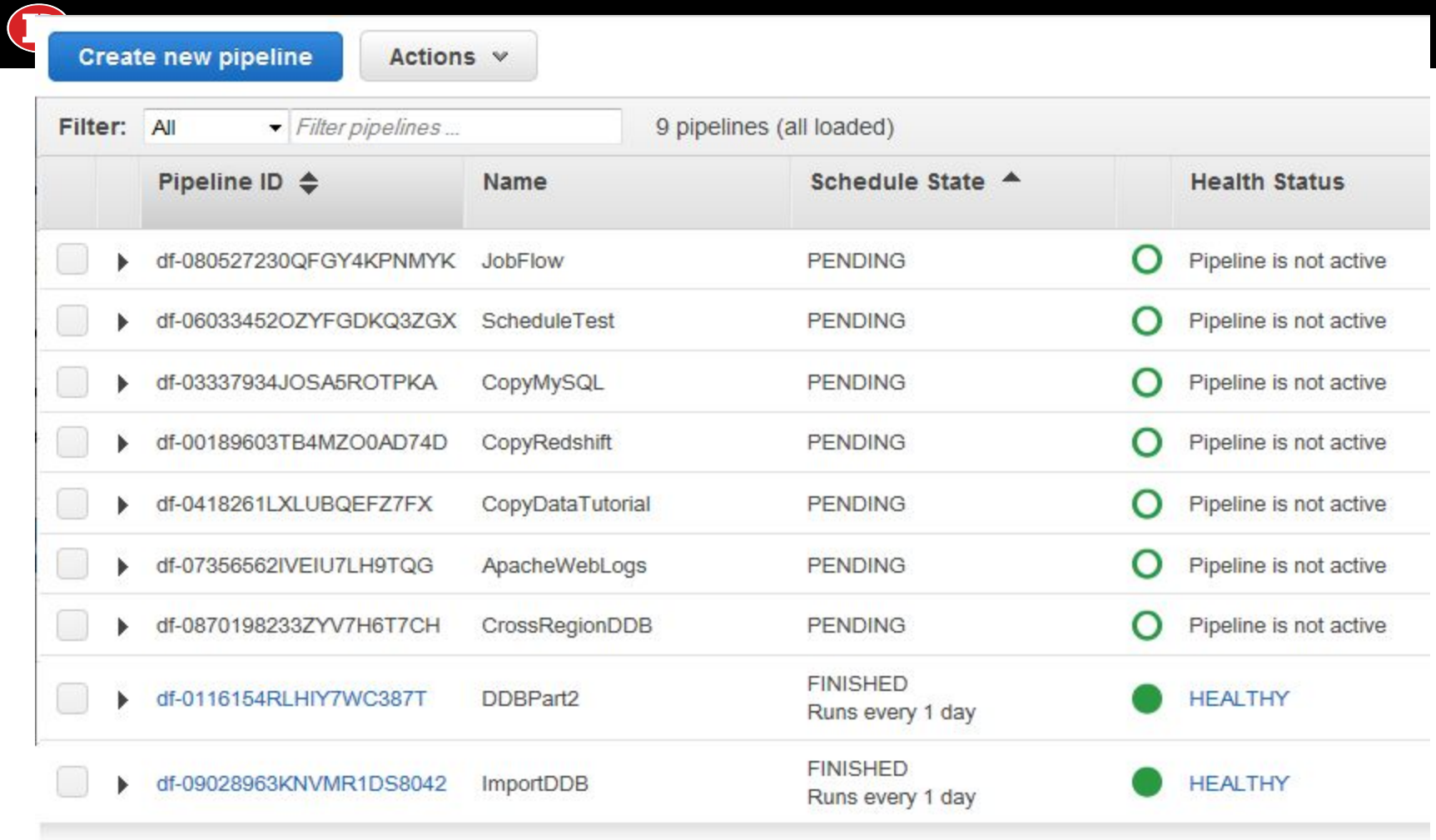

Luigi Task Visualiser
localhost:8082/static/visualiser/index.html#
Luigi Task Status
Task List
Dependency Graph
Workers
5 CreateTrialEligibility
10 CreateUserCollectionSummar
10 CreateCollectionTracksWithM
24 BannerShownCleanedHour
10 AdUserMetricsImport
10 AggregatedPlaysHive
6 CreateReportingUsage
10 LastLastActivityHive
11 CreateUserPlaylistSummary
3 FinancialUser
30 AppAnnieDownloadsImport
59 CreateArtistUsageBase
9 Cass2HdfsCompact
4 GCSToHDFSMain
11 CreateUserCollectionSummar
PENDING TAS...
6679
RUNNING TA...
0
DONE TASKS
6946
FAILED TASKS
88
UPSTREAM F...
351
DISABLED TA...
0
UPSTREAM DI...
0
Displaying tasks of family CreateReportingUsage .
Show 10 entries
Filter table:
Filter on Server
Name
Details
DONE CreateReportingUsage (test=False, date=2015-06-10, paralle
DONE CreateReportingUsage (test=False, date=2015-06-11, paralle
PENDING CreateReportingUsage (test=False, date=2015-06-13, paralle
PENDING CreateReportingUsage (test=False, date=2015-06-12, paralle
UPSTREAM_FAILED CreateReportingUsage (test=False, date=2015-06-14, paralle
UPSTREAM_FAILED CreateReportingUsage (test=False, date=2015-06-15, paralle
Showing 1 to 6 of 6 entries (filtered from 14,064 total entries)
Previous
1
Next


Luigi Task Visualiser
localhost:8082/static/visualiser/index.html#UserRecs(test=False, date=2013-07-24, re...
Luigi Task Status Active tasks
Task List
Dependency Graph
TaskId[param1=val1,param2=val2
Show task details
UserRecs(test=False, date=2013-07-24, rec_days=4, exp_days=8, test_users=False, force_updates=False, build_from_scratch=True, index_path=/spotify/discover/index, index_version=None, FOLLOWS_SCORE=5.0)
Dependency Graph
Failed
Running
Pending
Done
UserRecs
AccumulateByArtists
AccumulateUserMatrices
GenerateXTX
JoinArtistGids
JoinArtistGids
JoinArtistGids
JoinArtistGids
RelatedArtistsTC
UserLocationPeriod
AggregateByArtists
ArtistFollows
AggregateUserMatrices
AggregateUserMatrices
AggregateUserMatrices
AggregateUserMatrices
AggregateUserMatrices
AggregateUserMatrices
test=False
date=2013-07-21
index_version=1363504343
test_users=False
UserLocationDay
EndSongCleaned
EndSongCleaned

AWS Data Pipeline

- Cloud scheduler and resource instigator on hosted AWS hardware
- Can define data pipeline jobs, some of which come built-in (particularly AWS-to-AWS data transfer, complete with blueprints), but you can run custom scripts by hosting them on AWS
- Get all the AWS goodies: CloudWatch, IAM roles/policies, Security Groups
- Spins up target computing instances to run your pipeline activities per your configuration
- No explicit file based dependencies


AWS Data Pipeline Manageme
Laura
AWS
Services
Edit
llorenz @ industrydive
N. Virginia
Support
Data Pipeline
List Pipelines
Architect: test (df-07848673RXAPDR0ZWHHE) [Pending]
Add
Save
Activate
Export
View/Edit tags
Configuration
Default
Schedule
RunOnce
HadoopActivity
DefaultHadoopActivity1
MySqlDataNode
DefaultDataNode2
CopyActivity
DefaultCopyActivity1
S3DataNode
DefaultDataNode1
Activities
DefaultCopyActivity1
Name:
DefaultCopyActivity1
Type:
CopyActivity
Output:
DefaultDataNode1
Schedule:
RunOnce
Input:
DefaultDataNode2
DataNodes
Schedules
Resources
Preconditions
Others
Parameters
Errors/Warnings
Object: Default
WARNING: 'pipelineLogUri'is missing. It is recommended to set this value on Default object for better troubleshooting.
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

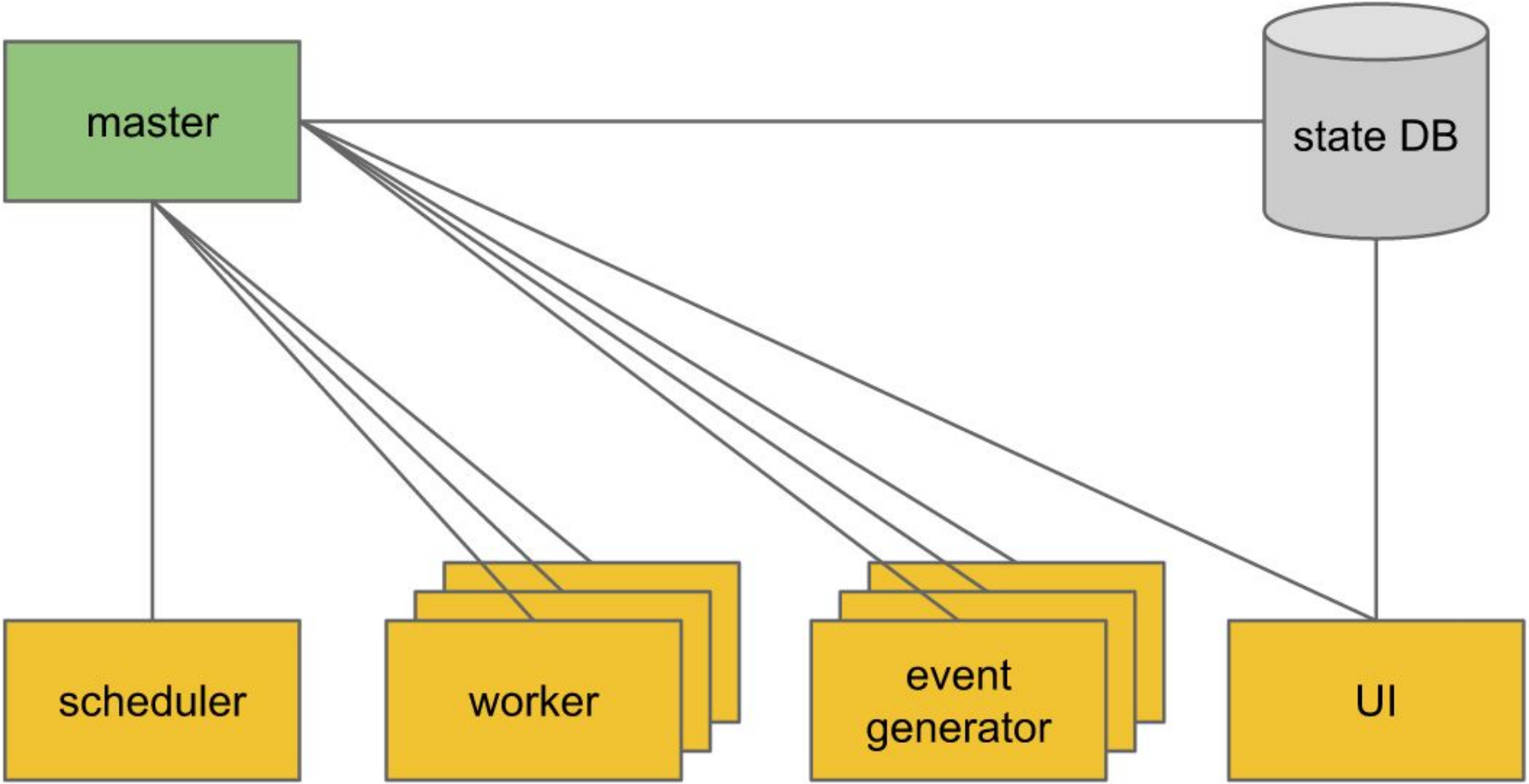

Create new pipeline
Actions
Filter: All
Filter pipelines ...
9 pipelines (all loaded)
Pipeline ID
Name
Schedule State
Health Status
df-080527230QFGY4KPNMYK
JobFlow
PENDING
Pipeline is not active
df-06033452OZYFGDKQ3ZGX
ScheduleTest
PENDING
Pipeline is not active
df-03337934JOSA5ROTPKA
CopyMySQL
PENDING
Pipeline is not active
df-00189603TB4MZO0AD74D
CopyRedshift
PENDING
Pipeline is not active
df-0418261LXLUBQEFZ7FX
CopyDataTutorial
PENDING
Pipeline is not active
df-07356562IVEIU7LH9TQG
ApacheWebLogs
PENDING
Pipeline is not active
df-0870198233ZYV7H6T7CH
CrossRegionDDB
PENDING
Pipeline is not active
df-0116154RLHIY7WC387T
DDBPart2
FINISHED
Runs every 1 day
HEALTHY
df-09028963KNVMR1DS8042
ImportDDB
FINISHED
Runs every 1 day
HEALTHY

Pinball

- Central scheduler server with monitoring UI built in
- Parses a file based configuration system into JobToken or EventToken instances
- Tokens are checked against Events associated with upstream tokens to determine whether or not they are in a runnable state.
- Tokens specify their dependencies to each other - not file based.
- More of an task manager abstraction than the other tools as it doesn't have a lot of job templates built in.
- Overrun policies deal with dependencies on past successes

```
15  import datetime
16
17  from pinball_ext.job.basic_jobs import PythonJob
18
19
20  class ExamplePythonJob(PythonJob):
21
22      def _setup(self):
23          print 'Do some setup in example python job!'
24
25      def _execute(self):
26          print 'Current time is %s' % str(datetime.datetime.now())
27
28
29  class ExamplePinballMagicPythonJob(PythonJob):
30      def _setup(self):
31          print 'Do some setup in example python job!'
32
33      def _execute(self):
34          # a_python_key=a_python_value pair will be passed to
35          # the downstream job as event attribute.
36          print 'PINBALL:EVENT_ATTR:a_python_key=a_python_value'
```

```
WORKFLOWS = {
    'tutorial_workflow': WorkflowConfig(
        jobs={
            'example_python_job':
            JobConfig(JobTemplate('ExamplePythonJob'), []),
            'example_command_job':
            JobConfig(JobTemplate('ExampleCommandJob'), ['example_python_job']),
            'example_quoble_hive_job':
            JobConfig(ExampleQuboleJobTemplate('ShowTableHiveJob'), ['example_command_job']),
            'example_emr_hive_job':
            JobConfig(ExampleEMRJobTemplate('RandomUsersHiveJob'), ['example_command_job']),
            'example_emr_hadoop_job':
            JobConfig(ExampleEMRJobTemplate('EmrWordCount'), ['example_emr_hive_job']),
        },
        final_job_config=JobConfig(FINAL_JOB),
        schedule=ScheduleConfig(recurrence=timedelta(days=1),
                                reference_timestamp=datetime(
                                year=2015, month=2, day=1, second=1)),
        notify_emails='your@email.com'),

```


master
state DB
scheduler
worker
event
generator
UI

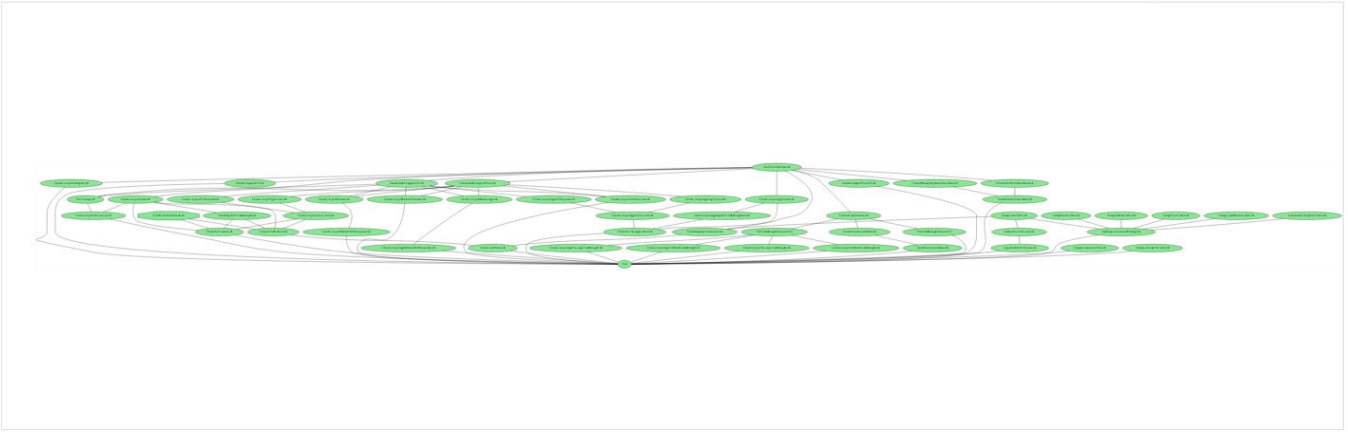
workflows | schedules | tokens

2015-03-12 18:42:45 UTC | pawel@pinterest.com | logout

/ core / 1426118415508 / Action

jobs list

Search:



All records per page

Search:

Job	Info	Progress	Status	Last Start	Last End	Run Time
AddHivePartitionsJob	**ShellJob**: command=cd /mnt/pinboard && /mnt/virtualenv/bin/python reporter/job_runner/data_job_runner.py --use_batch_table --auto_end_date --enable_io_logging --data_job_class=AddHivePartitionsJob --executor=datacore_qubole --job_args="end_date=2015-03-11" **priority**: 959 **max_attempts**: 2 **warn_timeout**: 6 hrs		SUCCESS	2015-03-12 00:01:03 UTC	2015-03-12 03:25:36 UTC	3 hrs 24 mins 33 secs
DumpUsersTableJob	**ShellJob**: command=cd /mnt/pinboard && /mnt/virtualenv/bin/python reporter/job_runner/data_job_runner.py --use_batch_table --auto_end_date --enable_io_logging --data_job_class=DumpUsersTableJob --executor=datacore_qubole --write_lock=db.dump.lock --job_args="end_date=2015-03-11" **priority**: 349 **max_attempts**: 2 **warn_timeout**: 12 hrs		SUCCESS	2015-03-12 00:01:06 UTC	2015-03-12 00:51:59 UTC	50 mins 52 secs
IncrementalDumpPinsTableJob	**ShellJob**: command=cd /mnt/pinboard && /mnt/virtualenv/bin/python reporter/job_runner/data_job_runner.py --use_batch_table --auto_end_date --enable_io_logging --data_job_class=IncrementalDumpPinsTableJob --executor=datacore_qubole --write_lock=db.dump.lock --job_args="end_date=2015-03-11" **priority**: 187 **max_attempts**: 2 **warn_timeout**: 12 hrs		SUCCESS	2015-03-12 00:01:38 UTC	2015-03-12 00:24:37 UTC	22 mins 59 secs