OPEN DATA SCIENCE CONFERENCE

#ODSC

@ODSC

Boston | April 30 - May 4, 2019

#ODSC

BOSTON

APR 30 - MAY 3

From Zero to Airflow: Bootstrapping into a Best-in-Class Risk Analytics Platform

Ido Shlomo

Data Science Manager,
BlueVine

About BlueVine

- Fintech startup up based in Redwood City, CA
- Provides working capital (loans) to small & medium sized businesses
- Over $1.6 BN funded to date
- Data Science challenges:
  - Consume many different semi structured / unstructured data sources
  - Deal with noisy / weak signals
  - Make decisions fast
  - Build models that are stable and accurate

About Me

- Data Science Manager @ BlueVine
- Lead BlueVine's DS team in Redwood City, CA (total of ~20 people across RWC & TLV)
- Team focus:
  - NLP & text mining
  - Anomaly detection
  - Probabilistic ML
  - Response modeling
- Personal interests: Unstructured data and DS Infrastructure.

What this presentation is about

Case study: **Deploying a best-in-class ML analytics platform into production using Apache Airflow.**

<u>Main points</u>:

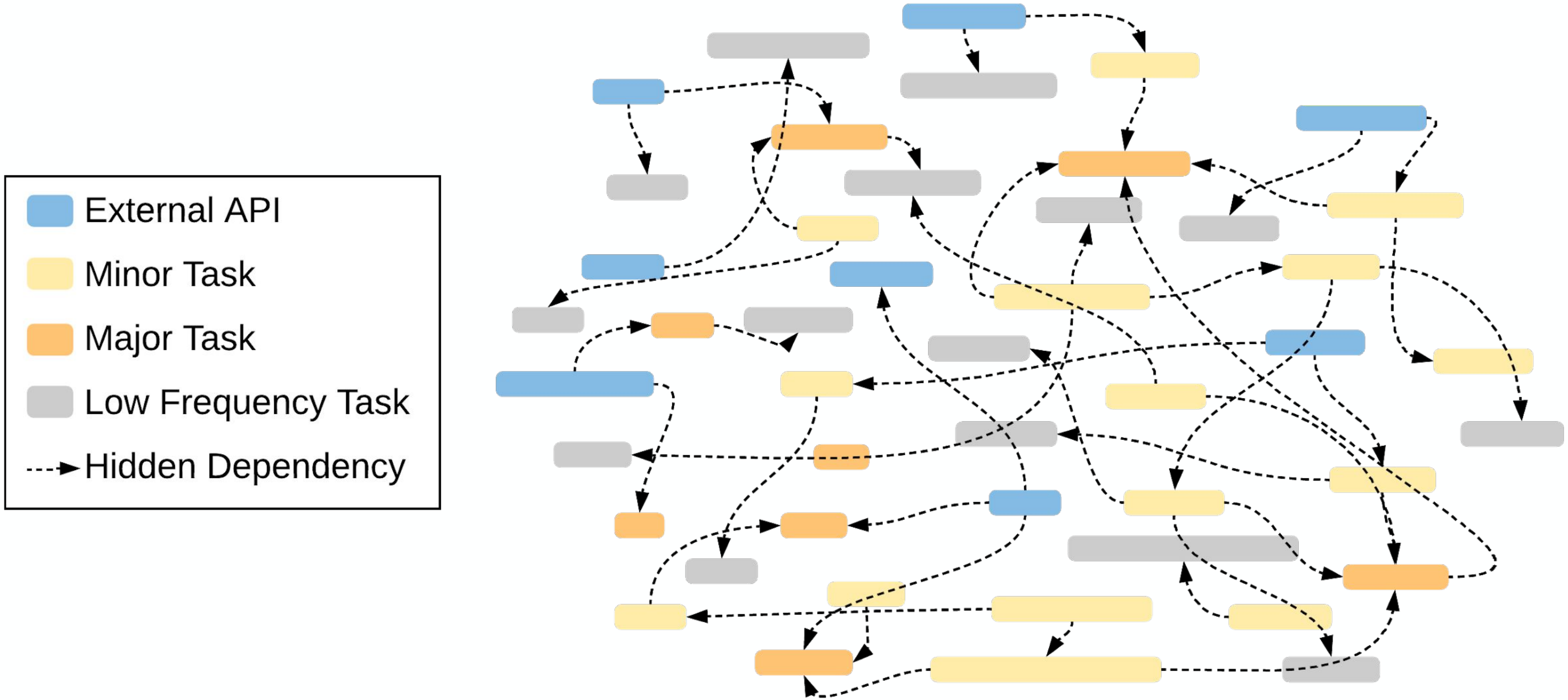
● The starting point	*What was already in place?*
● Business goals	*What did we need to achieve?*
● Initial design and plan	*What did we set up?*
● Real world behavior	*What went right / wrong + solutions!*
● The system in place today	*Tech breakdown*
● Questions	*Please nothing too hard…*

The starting point – *What was already in place?*

Lots (and lots) of cron-jobs!

- Every logic ran as an independent cron
- Every logic / cron figured out its own triggering mechanism
- Every logic / cron figured out its own dependencies
- No communication between logics

The starting point – *What was already in place?*



Business Goals – *What did we need to achieve?*

Desired	Existing
Ability to process **one** client end-to-end	Scope defined by # of clients in data batch
Decision within a few minutes	Over 15 minutes
Map and centrally control dependencies	Hidden and distributed dependencies
Easy and simple monitoring	Hard and confusing monitoring
Easy to scale	Impractical to scale
Efficient error recovery	"All or nothing" error recovery

Business Goals – *What did we need to achieve?*

Possible solutions: Lower is better!

	Cronjobs	Workflow Managers	Streaming
Achievable Runtime Latency	Minutes to hours	Seconds to Minutes	Milliseconds to Seconds
Effort to Implement & Transition	Low	Medium	High
Effort to use by data teams	High	Low	Medium

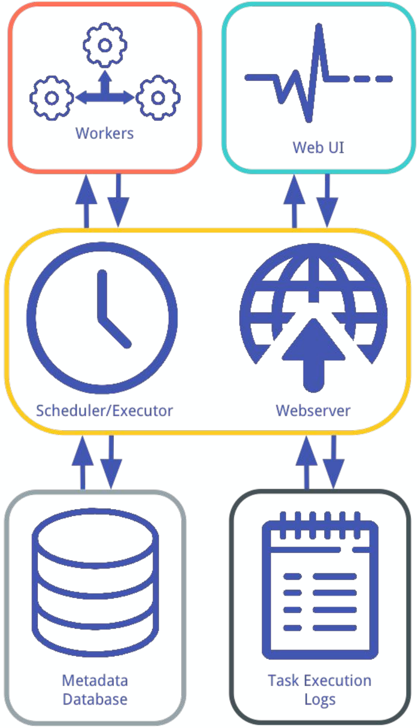
Initial Design and Plan – *What did we set up?*

We chose Apache Airflow

Brief intro:

- Core component is the scheduler / executor
- Uses dedicated metadata DB to figure out current status of tasks
- Uses workers to execute new ones
- Webserver allows live interaction and monitoring

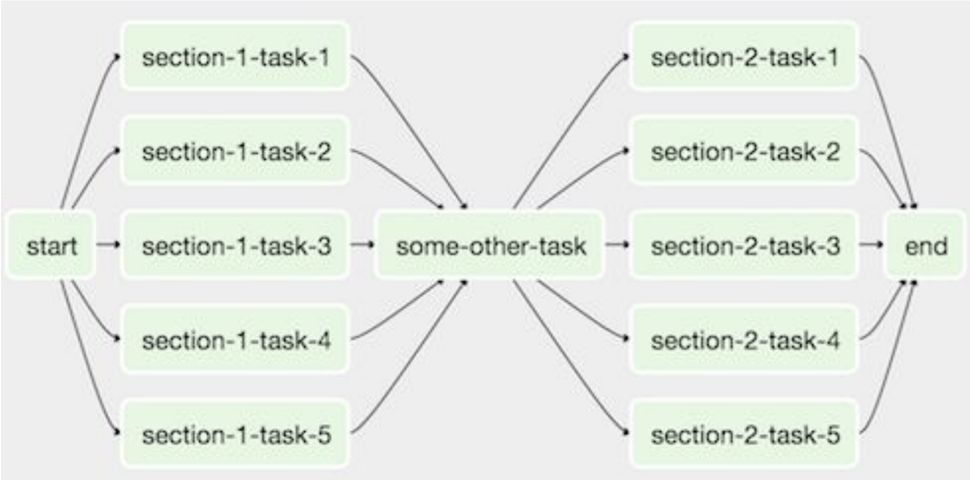
Airflow's General Architecture



Initial Design and Plan – *What did we set up?*

DAG: Directed Acyclic Graph

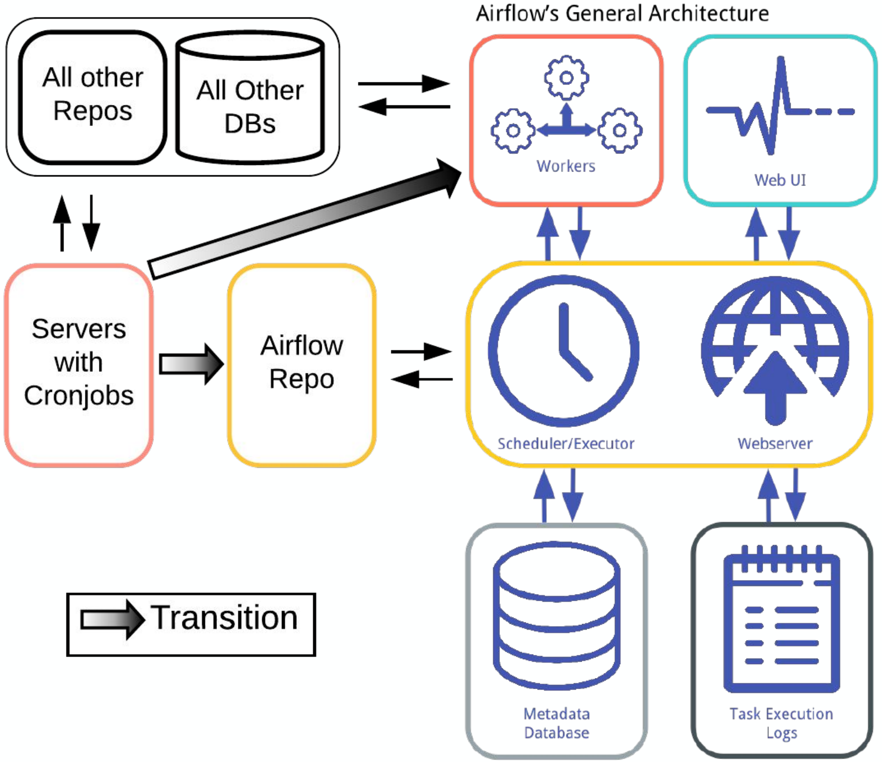
- Basically a map of tasks run in a certain dependency structure
- Each DAG has a run frequency (e.g. every 10 seconds)
- Both DAGs and tasks can run concurrently



Initial Design and Plan – *What did we set up?*

Transition:

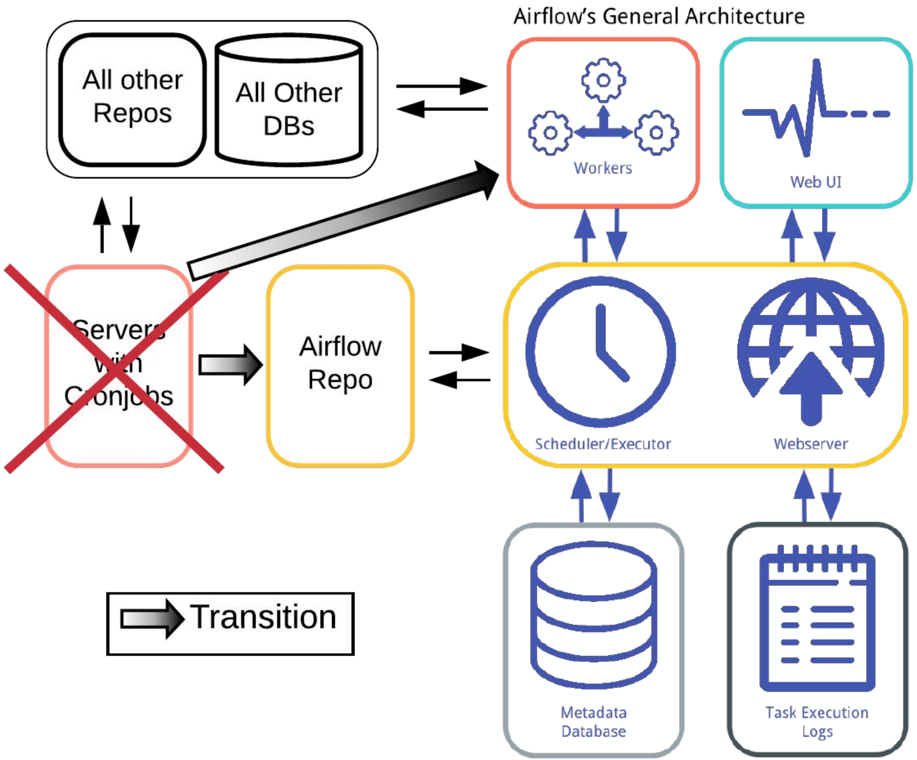
- Spin up Airflow *alongside* existing Data DBs, servers and cronjobs.
- Translate every cronjob into DAG with *one task* that points to same python script (Bash Operator).
- For each cron:
    - Turn off cronjob
    - Turn on “Singleton” DAG
- When all crons off → Kill old servers
- Done!



Initial Design and Plan – *What did we set up?*

Transition:

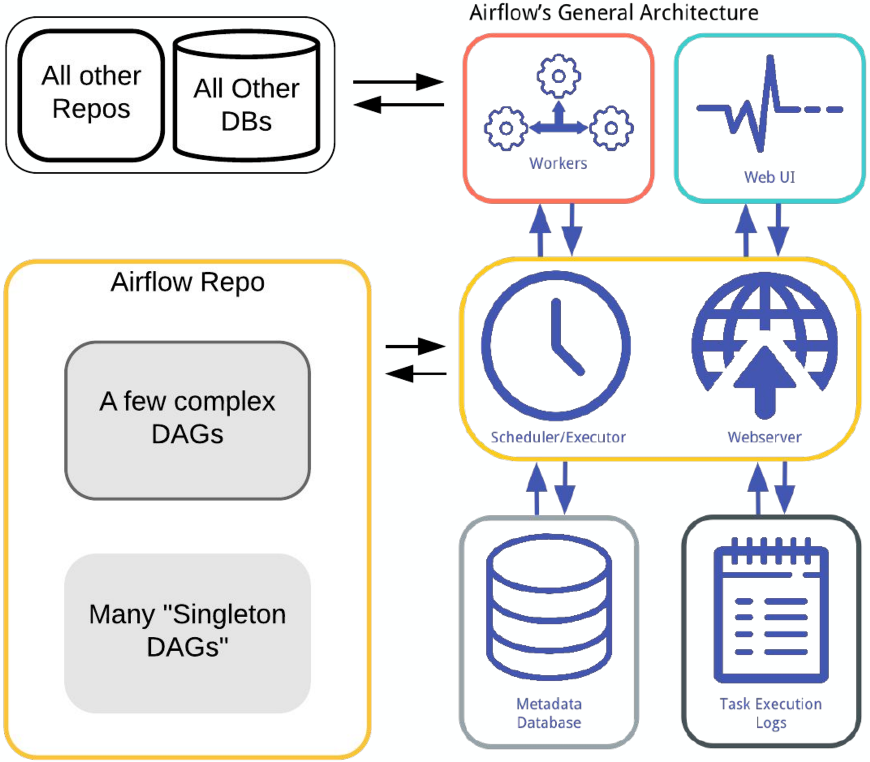
- Spin up Airflow <u>*alongside*</u> existing Data DBs, servers and cronjobs.
- Translate every cronjob into DAG with <u>*one task*</u> that points to same python script (Bash Operator).
- For each cron:
  - Turn off cronjob
  - Turn on “Singleton” DAG
- When all crons off → Kill old servers
- Done!



Initial Design and Plan – *What did we set up?*

Design:

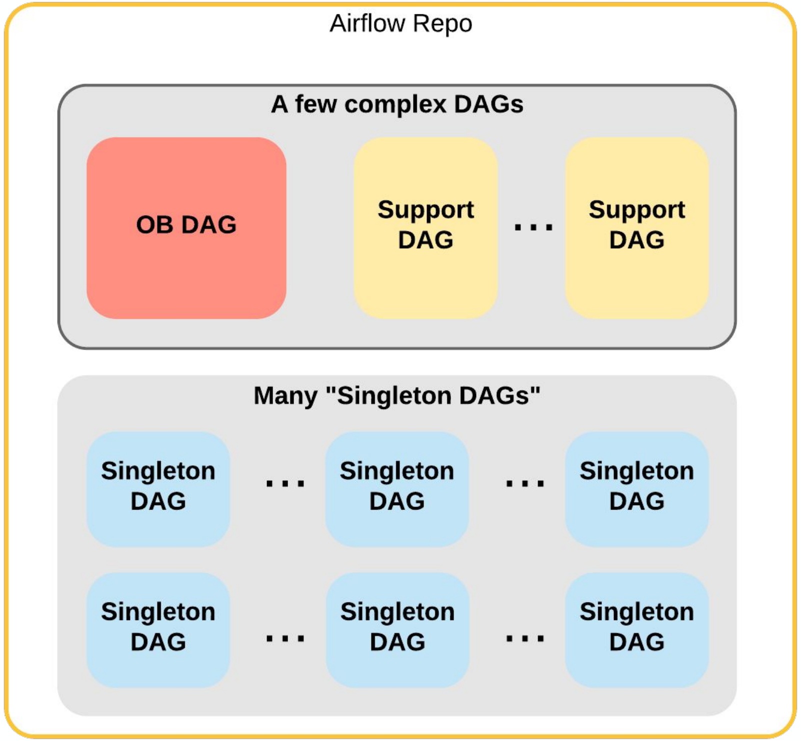
- Now we have approx 200 “Singleton” DAGs
- This does not leverage Airflows features at all
- Next step is to expose hidden dependencies by designing complex DAGs and removing ”Singletons”



Initial Design and Plan – *What did we set up?*

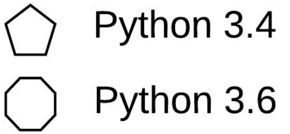
Design:

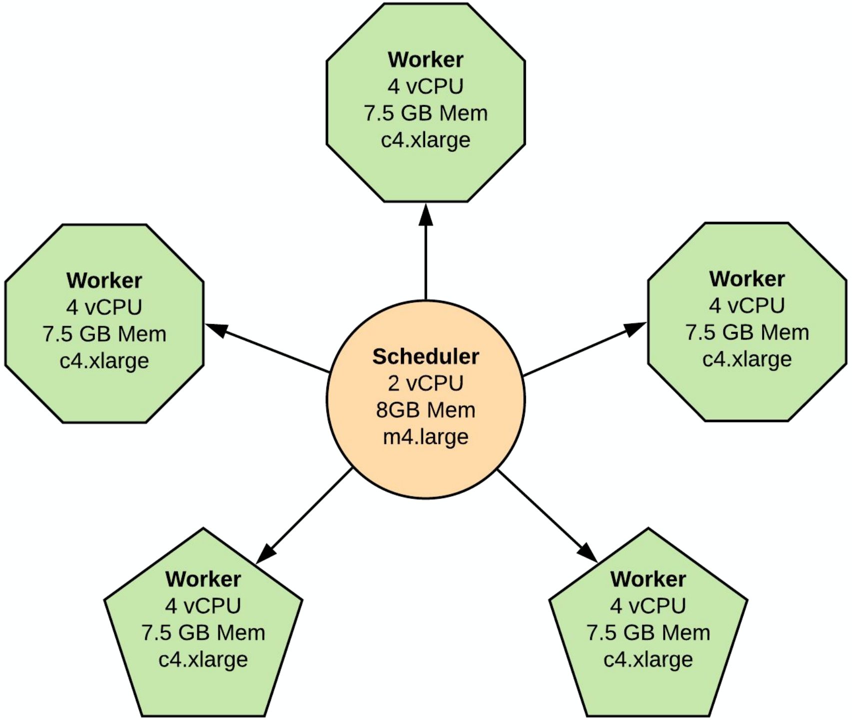
- Main focus of our work has been the complex “Onboarding” DAG:
  - Make funding decisions immediately after signup.
  - Ensure all requirements run in time and in the right order.
- Rest of this talk will focus on this.
- All other DAGs are of secondary importance.



Initial Design and Plan – *What did we set up?*

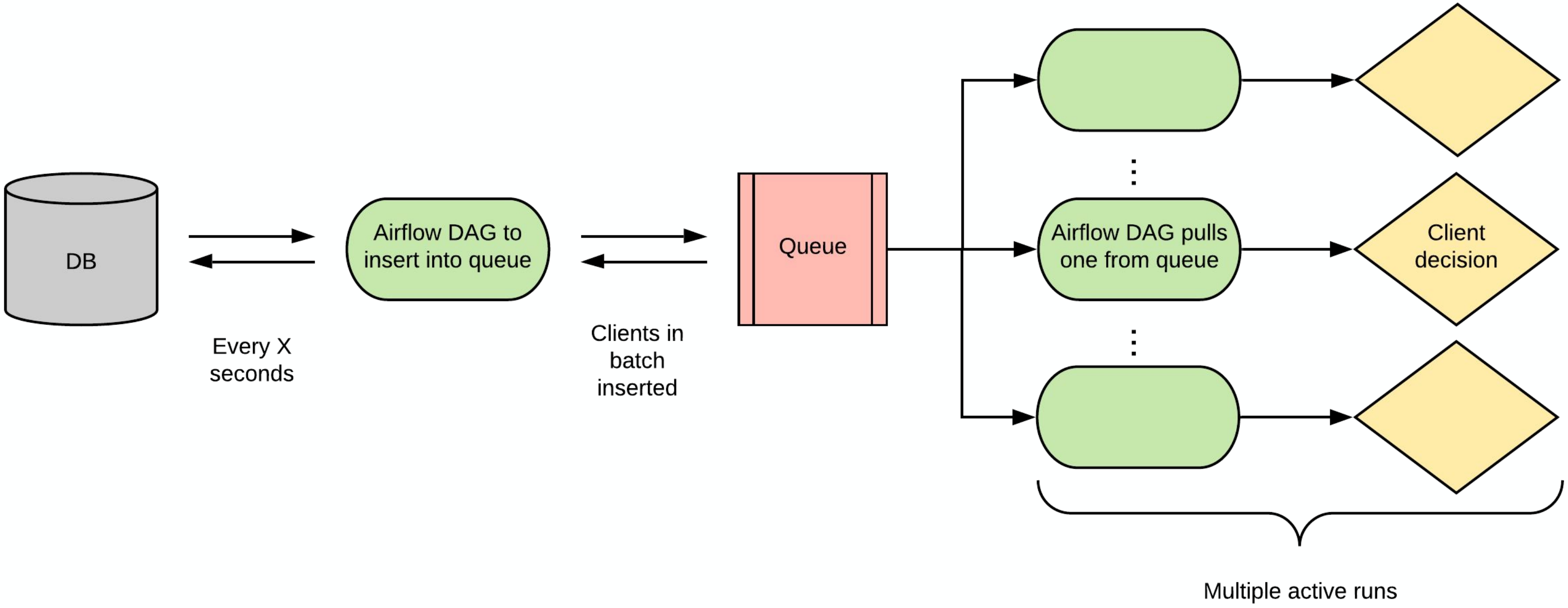
- Airflow Ver: 1.9





Initial Design and Plan – *What did we set up?*

Airflow is still a **batch processing** service → Need to convert batches into **single client units**.



Initial Design and Plan – *What did we set up?*

DAG to **insert into** queue (in batches), no concurrency

```
from datetime import datetime, timedelta
from airflow import DAG
from bv_airflow.operators import OnboardingQueueTriggerOperator
from bv_airflow.constants import DBReplica

dag = DAG(
    dag_id='onboarding.trigger',
    description='add new clients to onboarding queues',
    default_args=default_args, max_active_runs=1, concurrency=1, catchup=False,
    start_date=datetime(year=2018, month=7, day=1),
    schedule_interval=timedelta(seconds=15))

add_entities_to_queue_operator = OnboardingQueueTriggerOperator(
    dag=dag, dag_id=str(dag.dag_id), task_id='add_onboarding_entities_to_queue',
    pack_file='airflow_triggers/airflow_trigger_manager.py',
    risk_env=2, queue='risk2-onboarding', db_replica=DBReplica.orange,
    owner='ido.shlomo')
```

Initial Design and Plan – *What did we set up?*

Queue manager (airflow_trigger_manager.py)

```python
def message_handler(message_name):
    if message_name == 'add_onboarding_entities_to_queue':
        pit_old = args.pit_old
        pit_new = args.pit_new
        entities_ids = trigger_handler('new_onboarding_clients', pit_old, pit_new)
        TriggerQueueManager.add_entities(entities_ids, args.dag_id, args.execution_date)

    elif message_name == 'pull_entities_from_queue':
        _, entities_ids = TriggerQueueManager.pull_entities(size=1)
        entity_ids = ' '.join(str(entity_id) for entity_id in entities_ids)
        publish_result(entities_ids if entities_ids != ''
                       else 'NO-ENTITIES-PULLED-FROM-QUEUE')

    elif message_name == 'set_entities_as_done':
        TriggerQueueManager.return_entities_with_success(args.entities_ids)

    elif message_name == 'set_entities_as_error':
        TriggerQueueManager.return_entities_with_error(args.entities_ids)

    else:
        ...
```

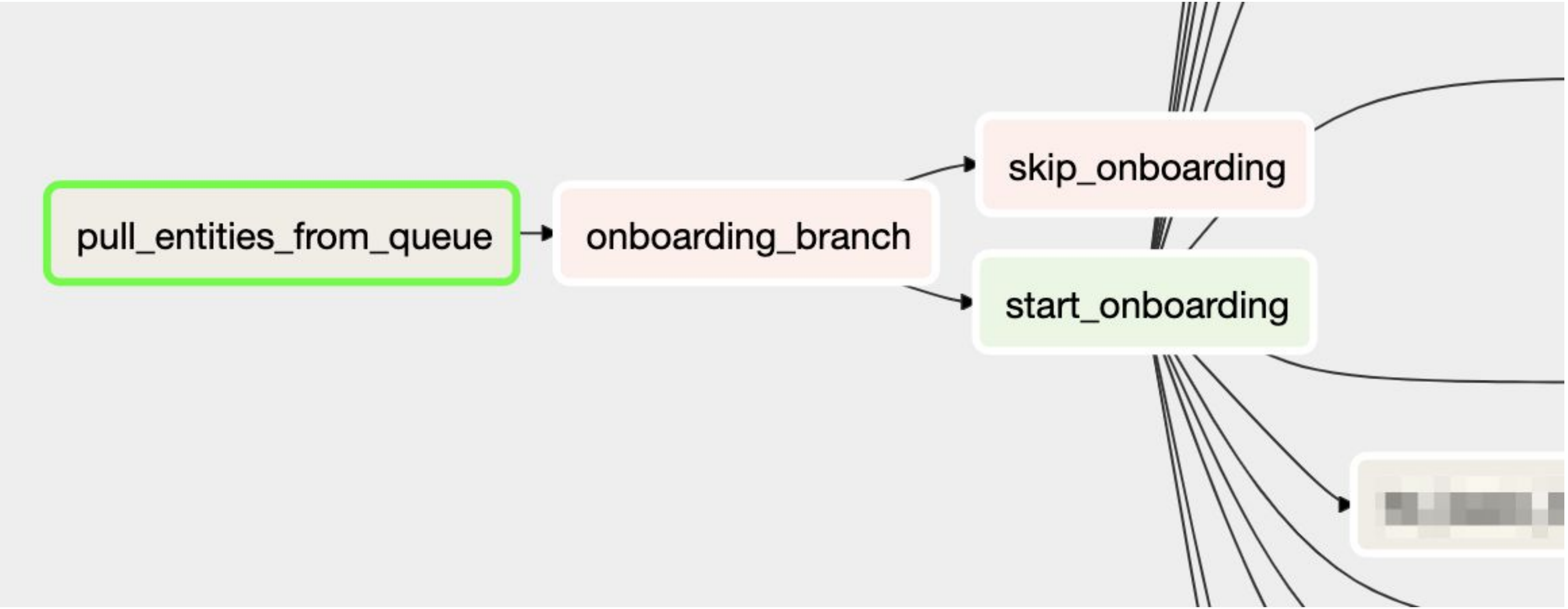
Initial Design and Plan – *What did we set up?*

DAG to **pull from** queue (in single units), high concurrency

```
from datetime import datetime, timedelta
from airflow import DAG
from bv_airflow.operators import OnboardingQueueTriggerOperator
from bv_airflow.constants import DBReplica

dag = DAG(
    dag_id='research.onboarding',
    description='consume clients from onboarding queue and run onboarding flow',
    default_args=default_args, max_active_runs=8,
    concurrency=100, catchup=False, start_date=datetime(year=2018, month=7, day=1),
    schedule_interval=timedelta(seconds=15),
)

pull_entities_from_queue_operator = PackOperator(
    dag=dag, task_id=TRIGGER_TASK_ID, pack_identifier=TRIGGER_TASK_ID,
    pack_file='airflow_triggers/airflow_trigger_manager.py',
    risk_env=2, queue='risk2-onboarding', db_replica=DBReplica.orange,
    xcom_push=True, owner='liana.diesendruck'
)
```

Initial Design and Plan – *What did we set up?*

DAG to **pull from** queue (in single units), high concurrency

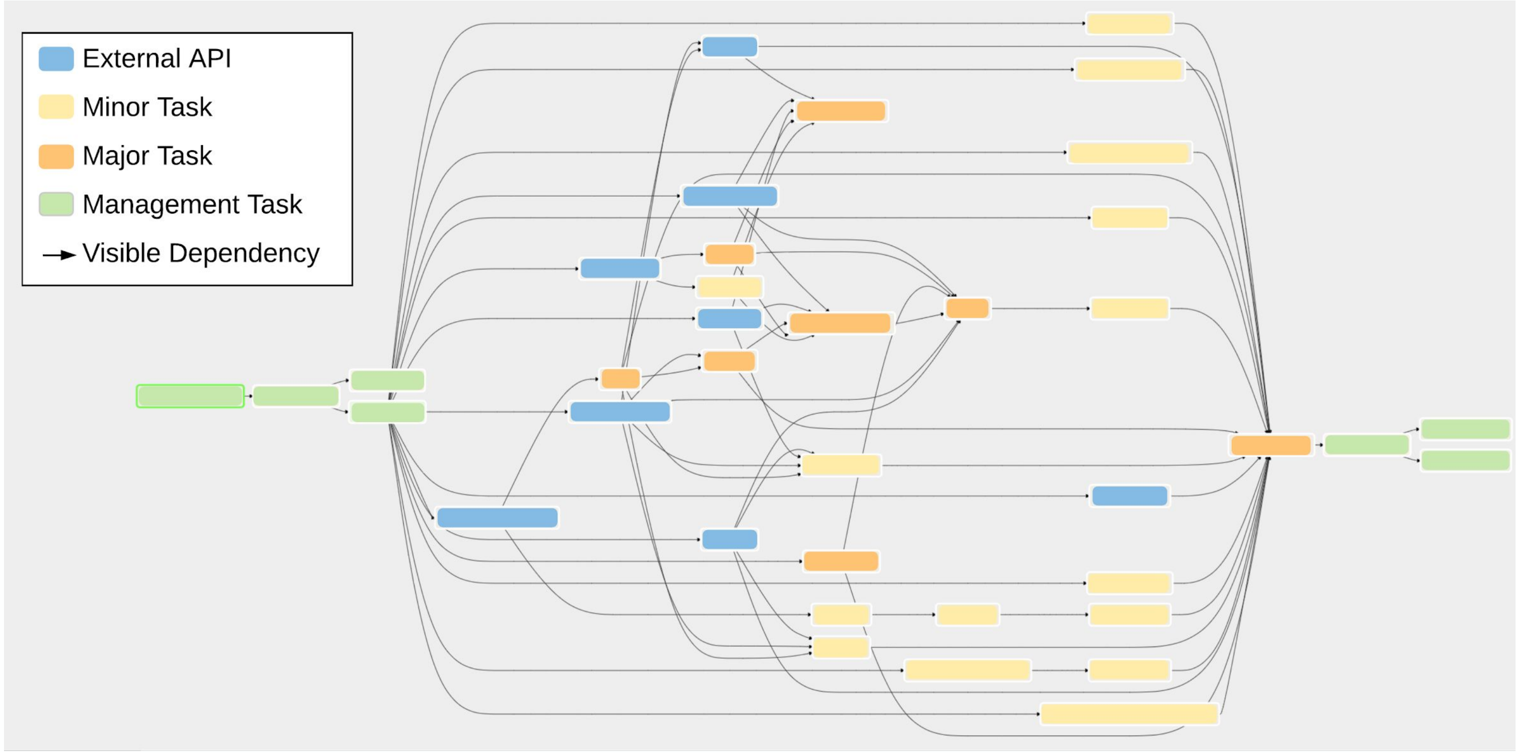


Initial Design and Plan – *What did we set up?*

DAG to **pull from** queue (in single units), high concurrency → run entire logic

```
# init
pull_entities_from_queue_operator >> onboarding_branch >> (skip_onboarding, start_onboarding)
start_onboarding >> onboarding_tasks

# bv score
[redacted] >> [redacted]
[redacted] >> [redacted]
[redacted] >> [redacted]
[redacted] >> [redacted] >> [redacted]

[redacted] >> [redacted] >> [redacted] >> [redacted]
[redacted] >> [redacted]
[redacted] >> [redacted]
[redacted] >> [redacted]

[redacted] >> [redacted]

# bv strategy
[redacted] >> [redacted] >> \
entity_state_branch >> (set_entities_as_done_operator, set_entities_as_error_operator)
```

Initial Design and Plan – *What did we set up?*



Real World Behavior – *What went right / wrong + solutions!*

The good:

- Transition passed smoothly
- UI works well (except some quirks)
- System is mostly stable
- Immediate gains seen in overall time-to-decision

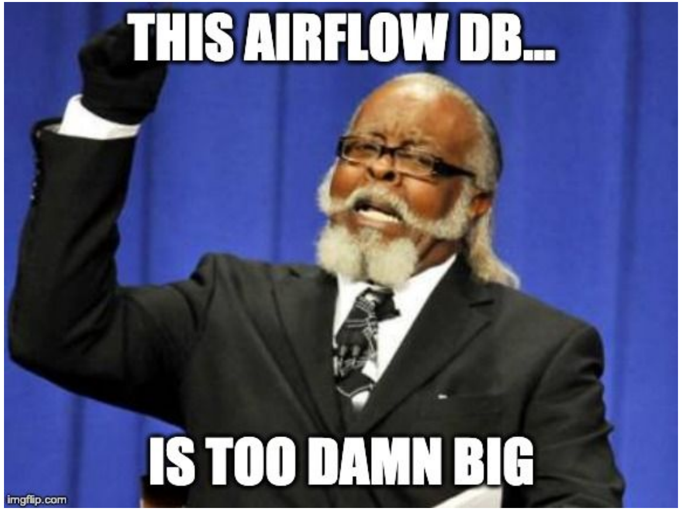
The bad:

- No user specific access roles
- Scheduler can silently die
- Tasks can become ”zombies”
- Scheduler performance quickly becomes a major (!) bottleneck

Real World Behavior – *What went right / wrong + solutions!*

Problem: Bloated Airflow DB

- Big DB → slower queries → slower scheduling & execution
- DB contains metadata for all dag / task runs
- High dag frequency + many dags + many tasks == many rows
- Under our setup, within first two months, the DB:
    - Had over 4 BN calls
    - Was over 15 GB in size



Solution: Run a weekly archive of data older than 1 week.

Real World Behavior – *What went right / wrong + solutions!*

Problem: Inefficient querying mechanism

- OB Dag has 40 tasks with 20 parallel runs, so scheduler does ~800 (!) queries every pass just for this one Dag.
- Then there are all the other ~200 Dags...

Solution:

- Instead of a query per task per Dag run, make query per Dag run.
- This is our humble contribution to the Airflow source code: https://github.com/apache/airflow/pull/4751.

Real World Behavior – *What went right / wrong + solutions!*

Overall Results:

- Average scheduling delay between tasks decreased by ~50%: from 1.5 sec to 0.8 sec.
- Max delay between tasks decreased by ~80%: from 6.4 sec to 1.3 sec.

Real World Behavior – *What went right / wrong + solutions!*

Problem: Scheduler overloaded

- Scheduler has to continually parse all DAGs
- Many dags + many tasks == lots to parse

Solution: Strengthen scheduler instance

- Airflow supports parallel parsing
- Strong instance → more processes → faster scheduling

Real World Behavior – *What went right / wrong + solutions!*

Problem: Scheduler can't prioritize

- Scheduler has to continually parse all DAGs
- Not all DAGs are equally important
- But all are given the same *scheduling* resources

Solution: Spin up a 2nd Airflow just for time-sensitive processes!

- Servers are cheap, time is expensive
- Dedicated instance → less dags / tasks → faster scheduling

Real World Behavior – *What went right / wrong + solutions!*

Overall results:

- OB DAG can never be “starved” for resources due to competition from other DAGs.
- Approx 30% reduction in average end-to-end OB flow runtime.
- Approx 60% reduction in average time spent on transitions between tasks.

Real World Behavior – *What went right / wrong + solutions!*

Airflow updates are already addressing some of the issues that we found!

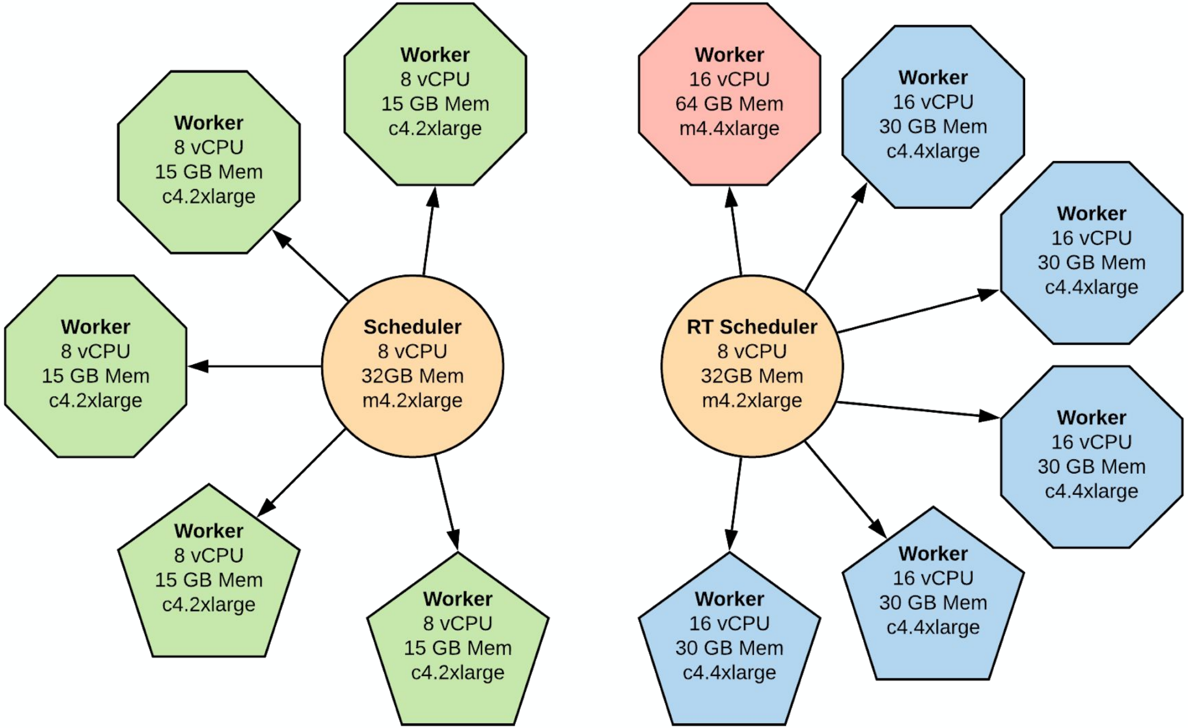
- Role based access control (RBAC) introduced in Airflow V1.10:
  https://issues.apache.org/jira/browse/AIRFLOW-1433
  https://issues.apache.org/jira/browse/AIRFLOW-85
- DAG parsing and task execution decoupled in the scheduler:
  https://github.com/apache/airflow/pull/3873
- Parallelize celery executor state fetching in the scheduler:
  https://github.com/apache/airflow/pull/3830

The system in place today – *Tech Breakdown*

- Airflow Ver: 1.10.3
- DB Cleanup: Weekly

⬠ Python 3.4

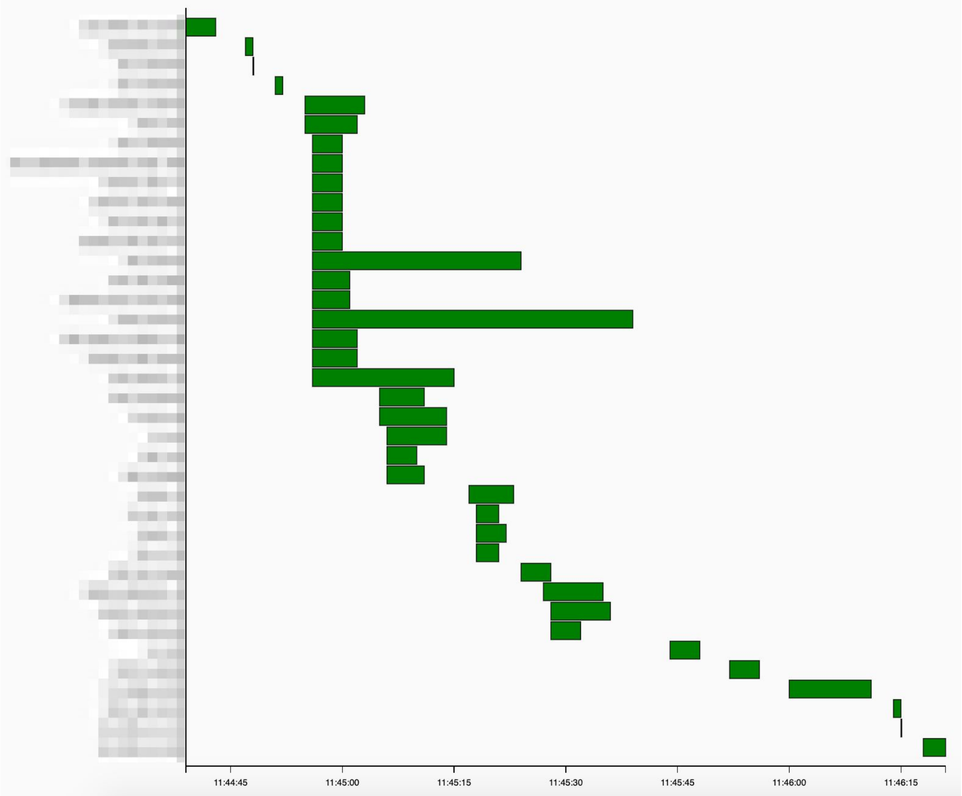
⯃ Python 3.6



The system in place today – *Tech Breakdown*

SLA Highlights:

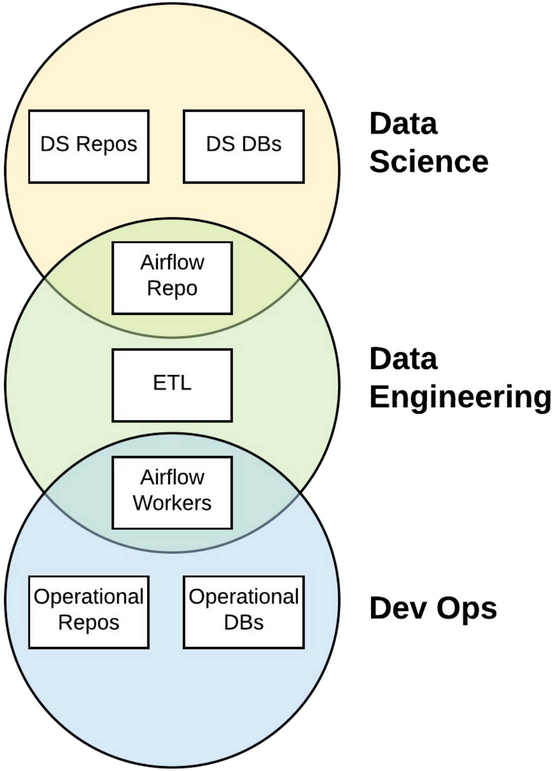
- Overall runtime is ***under 3 minutes*** for 95% of the cases
- Any given task runs ***under 1 minute*** for 95% of the cases
- Time between dependent tasks is ***under 3 seconds***



The system in place today – *Tech Breakdown*

Data division of labor:

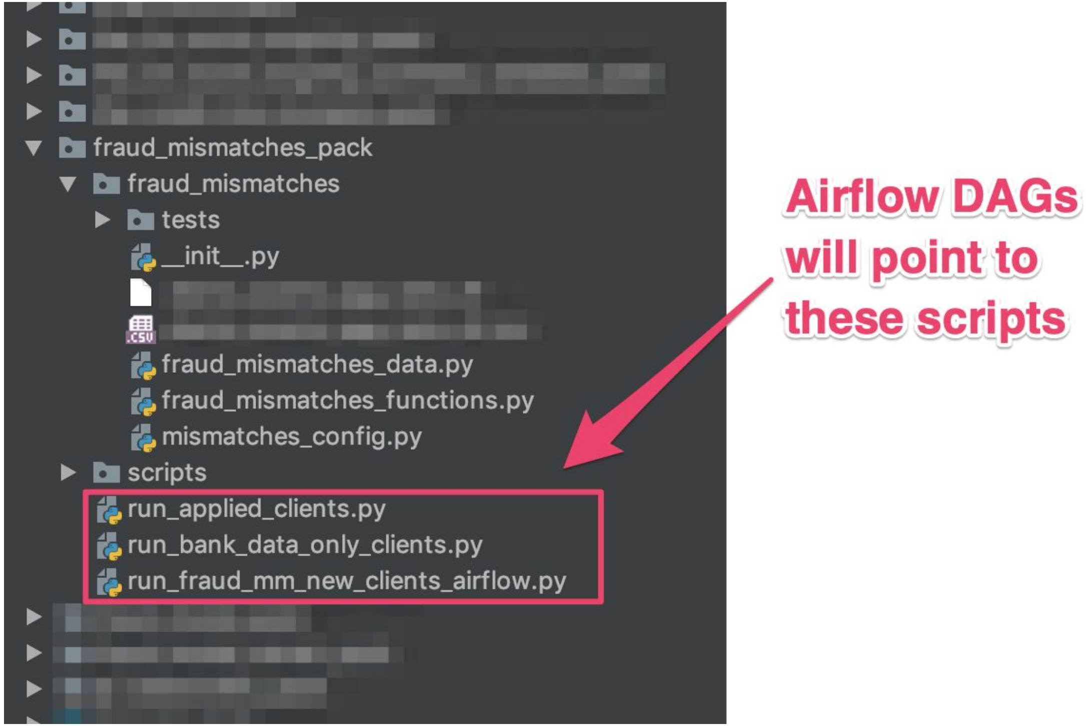
- DS owns models & analytics
- DS owns workflow logic via PR to DE
- DE owns workflow implementation via PR by DS
- DE owns Airflow settings and architecture
- DO owns Airflow implementation via PR by DE
- DO owns source-of-truth operational DBs and repos



The system in place today – *Tech Breakdown*

DS:

Define logic independently



The system in place today – *Tech Breakdown*

DS PR to

DE:

Adding new

logic to

Airflow

35 dags/packs/fraud_distributions.py

```
+
+ dag = DAG(
+     dag_id='packs.fraud_distributions',
+     description='calculating fraud probability features using pre-defined distributions',
+     default_args=default_args,
+     max_active_runs=1,
+     catchup=False,
+     start_date=datetime(year=2019, month=4, day=24),
+     schedule_interval=timedelta(minutes=5),
+
+ )
+
+ main_task = PackOperator(
+     task_id='main_task',
+     dag=dag,
+     risk_env=2,
+     pack_file='fraud_distributions_pack/run.py',
+     pack_identifier='fraud_distributions',
+     owner=default_args.get('owner'),
+     db_replica=DBReplica.orange,
+     execution_timeout=timedelta(minutes=90),
+ )
```

Define DAG and run settings

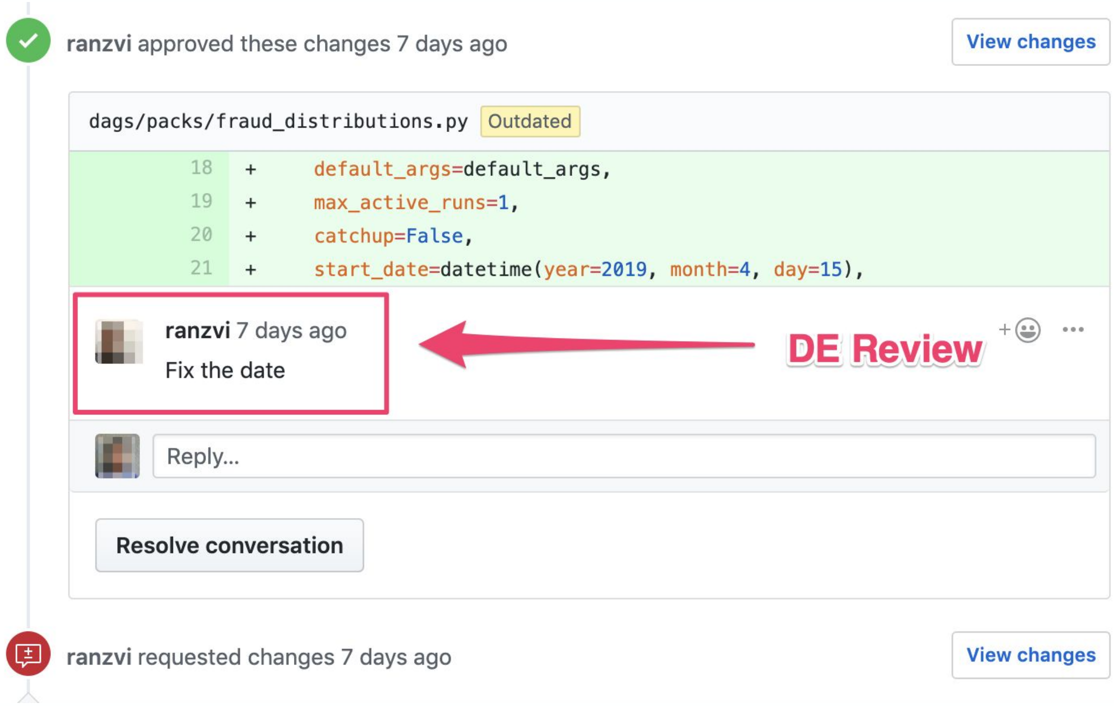
Define task that points to DS script

The system in place today – *Tech Breakdown*

DS PR to DE:

Adding new logic to

Airflow



Questions? + Thanks!