BlueVine

From Zero to Airflow bootstrapping a ML platform

About us

Bluevine

- Fintech startup up based in Redwood City, CA and Tel Aviv, Israel
- Provides working capital (loans) to small & medium sized businesses
- Over $2 BN funded to date
- Over 3.5$ BN delivered in Payment Protection Program

Me

- Noam Elfanbaum (@noamelf), Data Engineering team lead @ BlueVine
- Live in Tel-Aviv with my wife, kid and dog.
- My colleague Ido Shlomo created the original presentation for OSDC 2019 conference.

Case study

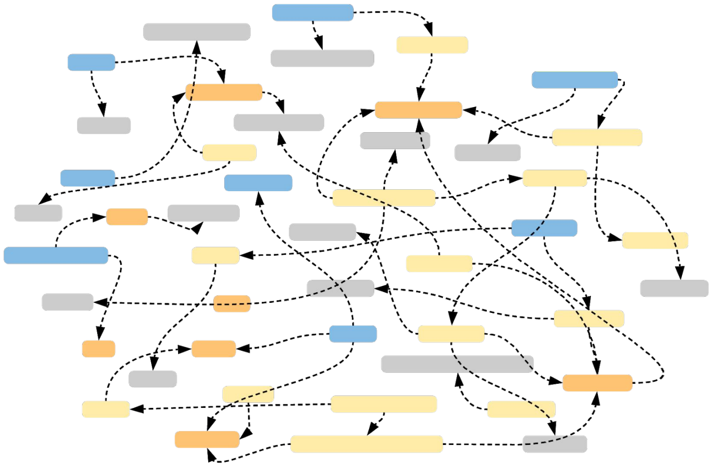
Building a ML analytics platform into production using Apache Airflow at Bluevine. This includes:

- Migrating our ML workload to Airflow
- Hacking at Airflow to provide a semi-streaming solution
- Monitoring business sensitive processes

Part 1: Migrating to Airflow

What was in place?

- Lots (and lots) of cron-jobs on a single server!
- 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



Goals

Desired

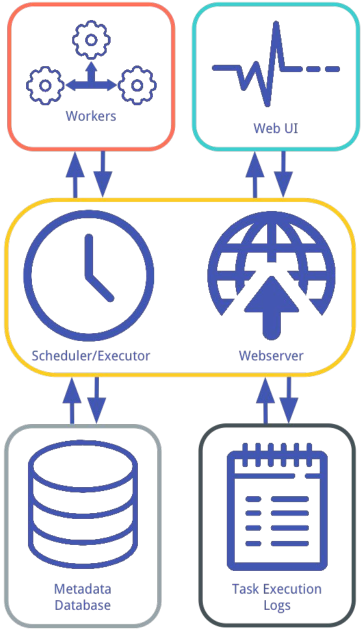
- Ability to process one client end-to-end
- Decision within a few minutes
- Map and centrally control dependencies
- Easy and simple monitoring
- Easy to scale
- Efficient error recovery

Existing

- Scope defined by # of clients in data batch
- Over 15 minutes
- Hidden and distributed dependencies
- Hard and confusing monitoring
- Impractical to scale
- “All or nothing” error recovery

Airflow brief intro

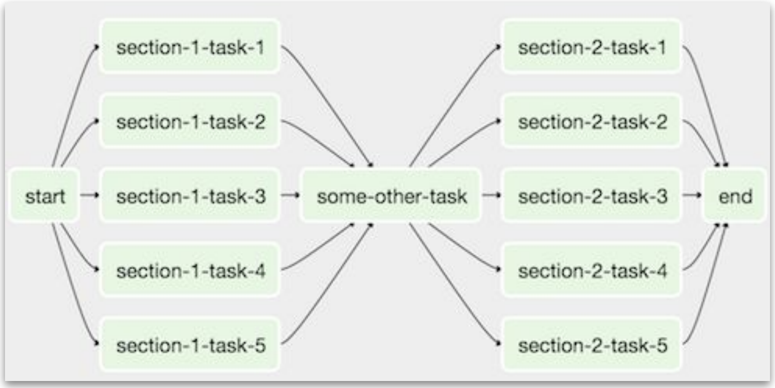
- Core component is the scheduler / executor
- Uses dedicated metadata DB to figure out current status of tasks
- Uses workers to execute new ones
- Web server allows live interaction and monitoring



What is a DAG?

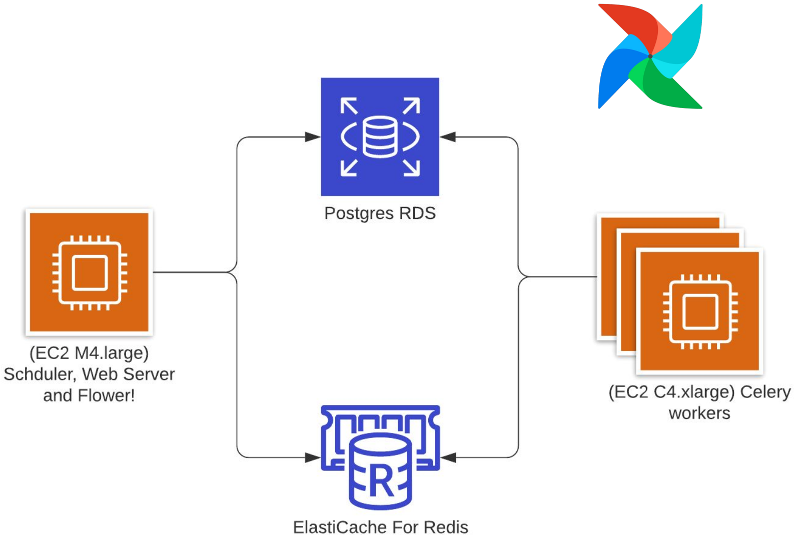
DAG: Directed Acyclic Graph

- 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



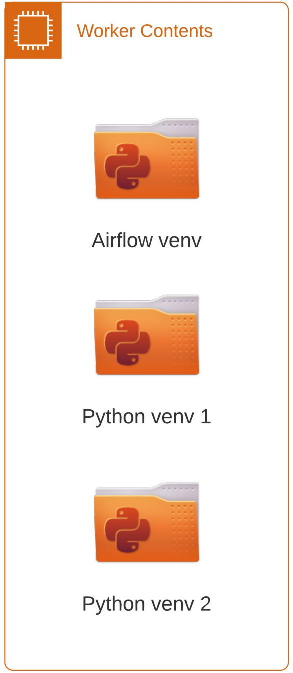
Infrastructure setup

- We run on AWS - and prefer managed services
- Celery is the executor
- Flower proved very useful for monitoring workers state
- No thrills setup!


Postgres RDS
(EC2 M4.large) Schduler, Web Server and Flower!
(EC2 C4.xlarge) Celery workers
ElastiCache For Redis

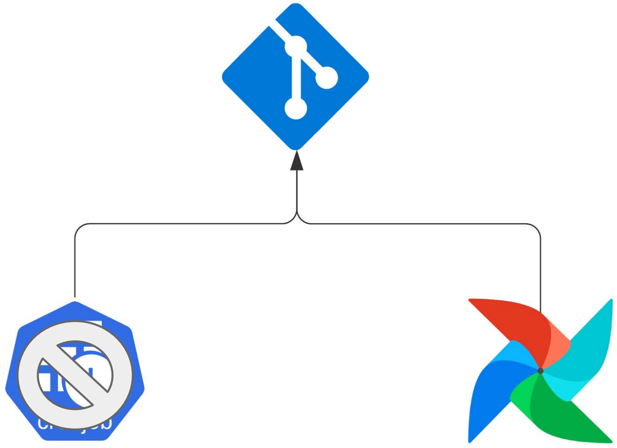
Isolated environments

- Isolation between Airflow environment and our scripts
- BashOperator is executing the script under the correct virtual environment



Phasing out cron jobs

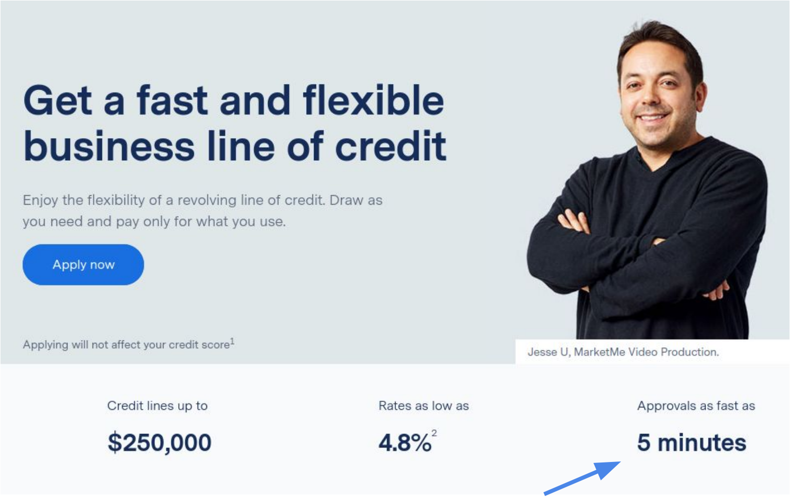
- Spin up Airflow alongside existing Data DBs, servers and cron jobs.
- Translate every cron job into DAG with one task that points to same python script (Bash Operator).
- For each cron (200 of them):
  - Turn off cron job
  - Turn on “Singleton” DAG
  - When all crons off → Kill old servers



Part 2: Hacking a streaming solution

User onboarding

- Airflow is built for batch processing
- We needed to support streaming user processing
- Airflow is not a good fit for that!
- Nevertheless, due to time constraints and familiarity, we chose to start with it



Get a fast and flexible business line of credit

Enjoy the flexibility of a revolving line of credit. Draw as you need and pay only for what you use.

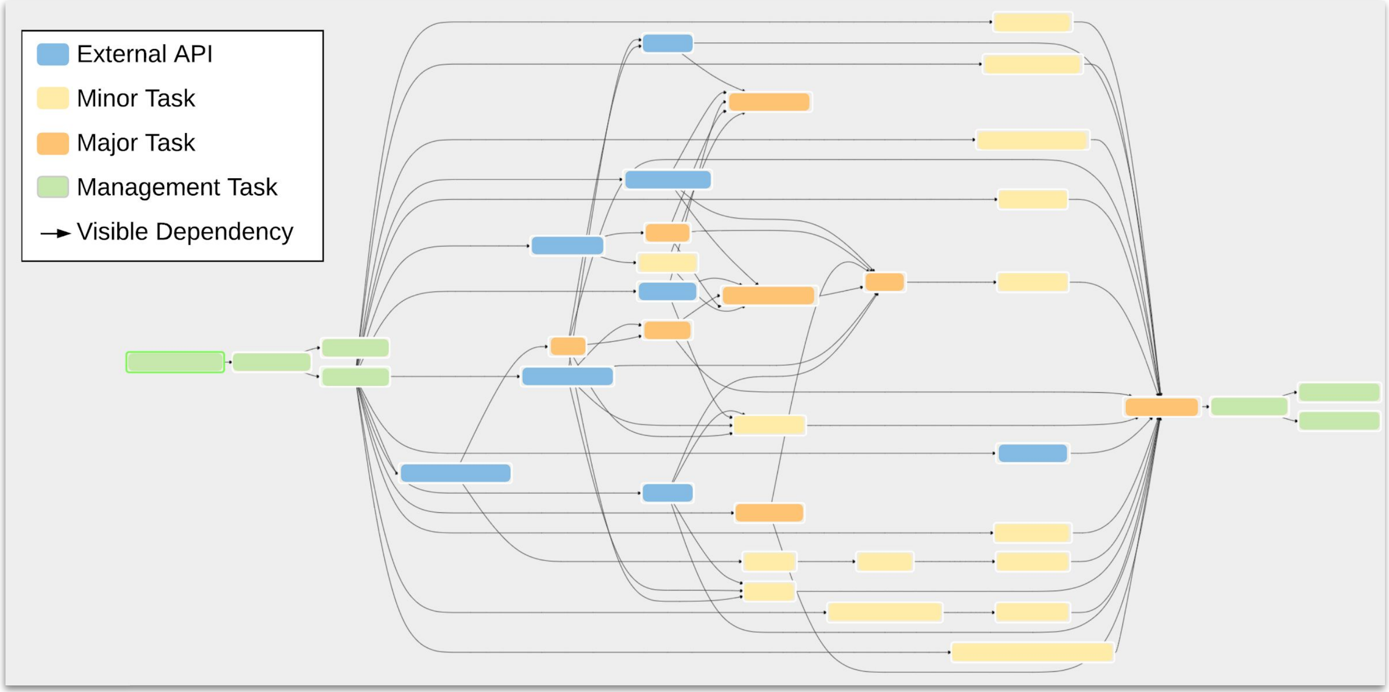
Applying will not affect your credit score[1]

Credit lines up to $250,000

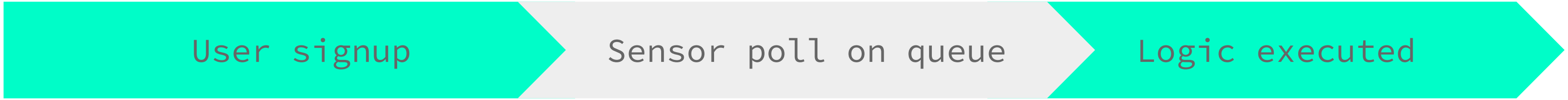
Rates as low as 4.8%[2]

Approvals as fast as 5 minutes

THE ONBOARDING DAG (SORT OF)



Onboarding "streaming" Design



User signup

Sensor poll on queue

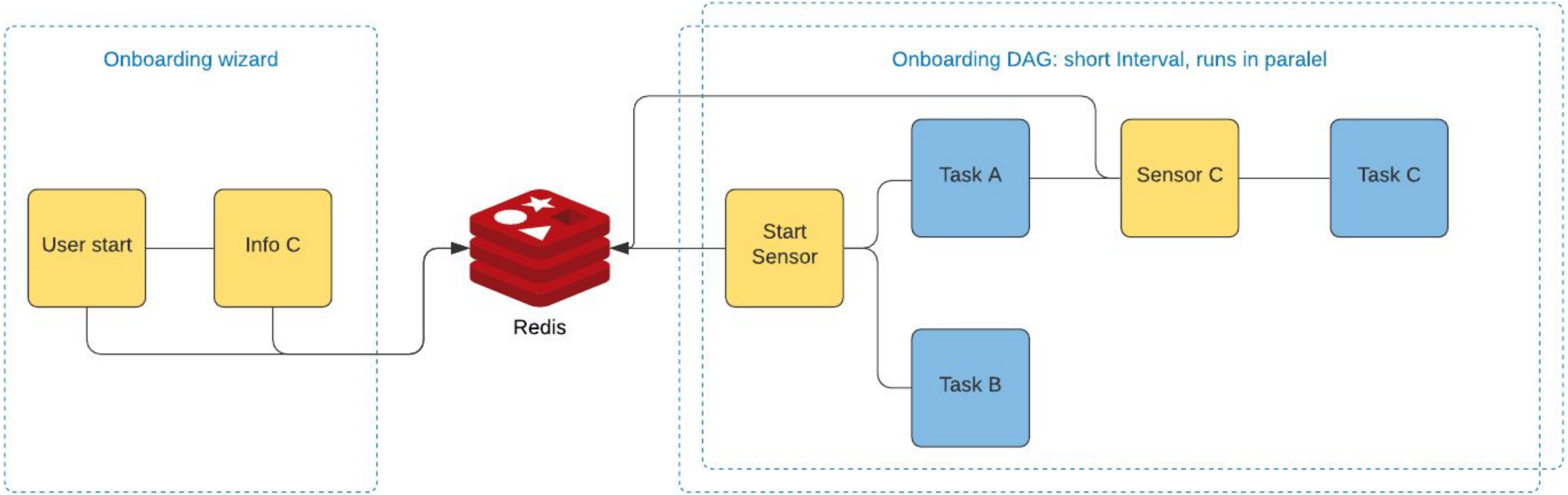
Logic executed

A "new user" event is sent. As user goes through the application forms the relevant events are sent

Onboarding DAG poll for the events using the SensorOperator. Once a "new user" event is received, the user ID is saved in XCOM to share it between the tasks

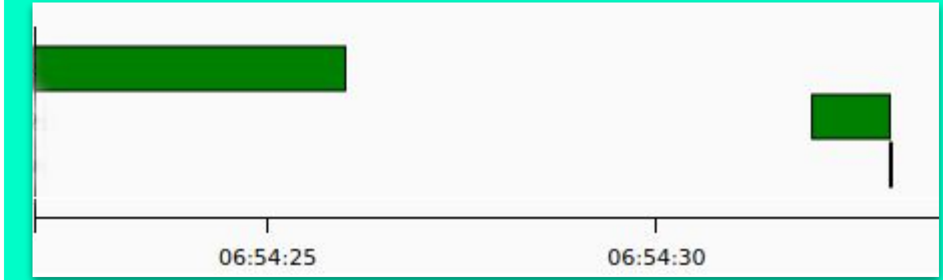
Related functionality is executed, as the user progress through the application form

Onboarding design



Hitting a performance wall

Airflow scheduler took up to 30 seconds to compute the next task to run (i.e. step)!



Hack #1 - standalone trigger

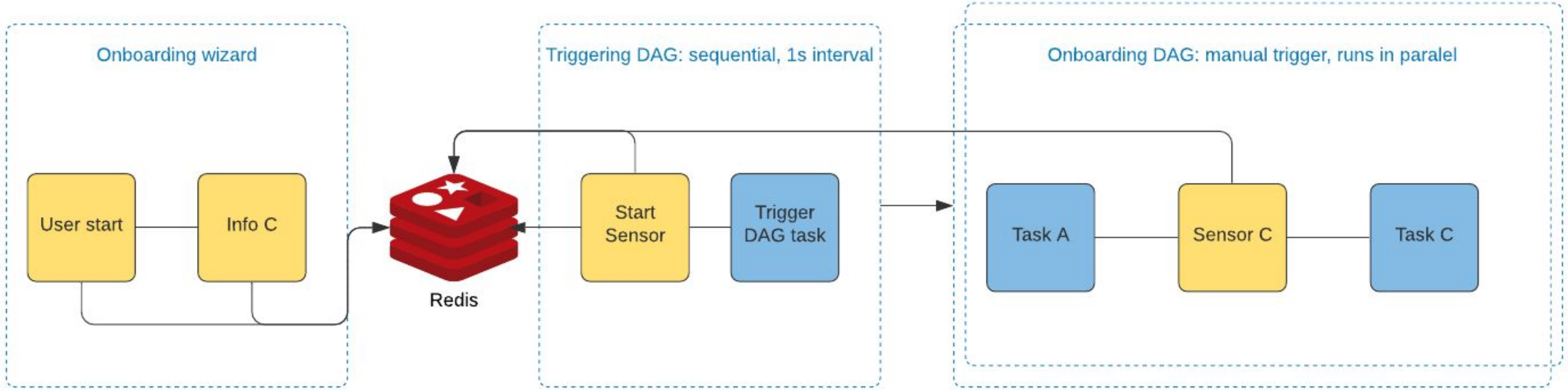
Problem

- Airflow scheduler is creating all tasks objects on DAG start
- The onboarding DAG has ~40 tasks, and the scheduler works hard to figure out each task dependencies
- A new DAG run starts on interval and a sensor is polling for new user
- This creates a lot of “live” pending DAGs

Solution

- Have a triggering DAG that only contains a sensor and a triggering task
- It triggers the large on-boarding DAG

Hack #1 - standalone trigger


Onboarding wizard
User start
Info C
Redis
Triggering DAG: sequential, 1s interval
Start Sensor
Trigger DAG task
Onboarding DAG: manual trigger, runs in paralel
Task A
Sensor C
Task C

Hack #2: Archive DB tables

Problem

- 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 was over 15 GB in size

Solution

- Archive DB data to keep 1 week of history
- **Gotcha!** Also make sure to keep a DAG last run, not doing so will make Airflow think it didn’t run and rerun it.

Hack #3 - Patch scheduler DAG's state queries

Problem

- In order to determine if a task met its dependencies, the scheduler query the DB for each task in the DAG
- The Onboarding Dag has 40 tasks and can have 20 parallel runs.
- This means ~800 (!) DB queries every pass just for this one Dag.

Solution

- Patch Airflow to query the DAG state by sending one query per DAG instead of a query per DAG task.
- PR made to Airflow team: AIRFLOW-3607, to be released in Airflow 2.0
- Results:
  - 90th percentile delay was decreased by 30%
  - DB CPU usage decreased by 20%
  - Avg delay was decreased 18%

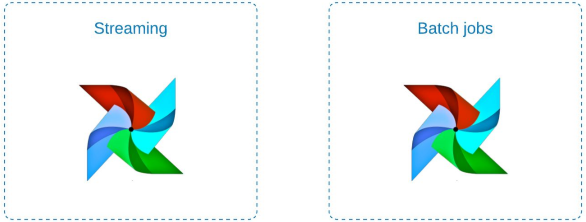
Hack #4 - Create a dedicated "fast" Airflow

Problem

- Scheduler has to continually parse all DAGs
- Not all DAGs are equally latency sensitive but all are given the same scheduling resources

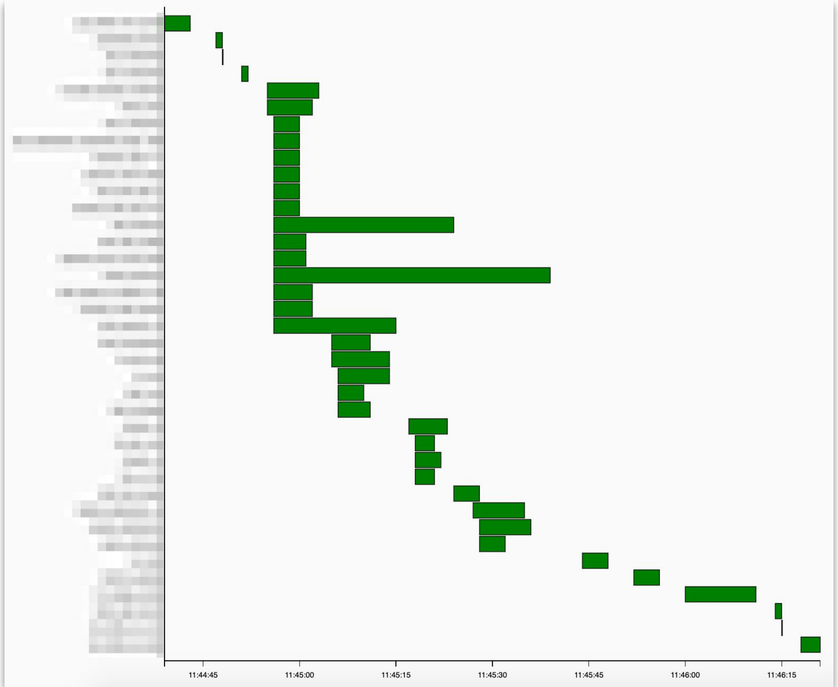
Solution

- Spin up a 2nd Airflow just for time-sensitive processes!
- Dedicated instance → less dags / tasks → faster scheduling
- Approx 60% reduction in average time spent on transitions between tasks.



Final results

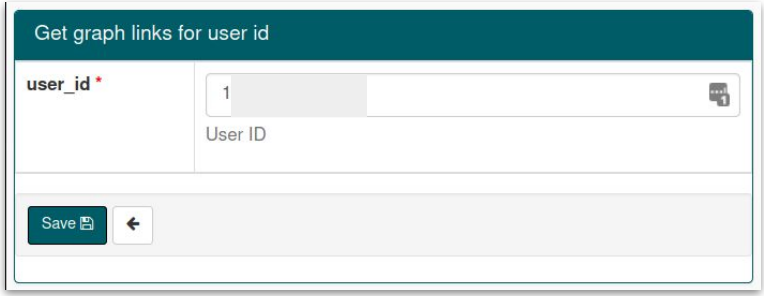
- Time between dependent tasks is **consistently** under 3 seconds
- Overall runtime is under 3 minutes for 95% of the cases



Part 3: Monitoring

Plugin to match users with runs

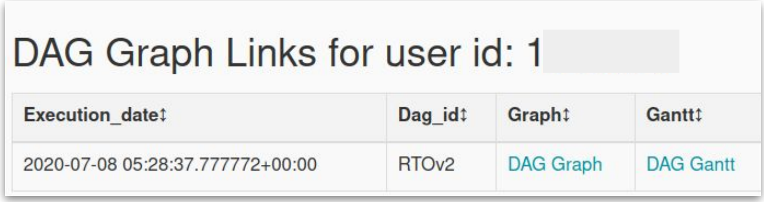
- Locates the Airflow DAG run for a given user ID
- Helps to track down issues found with users



Get graph links for user id

user_id *: 1[redacted]

User ID

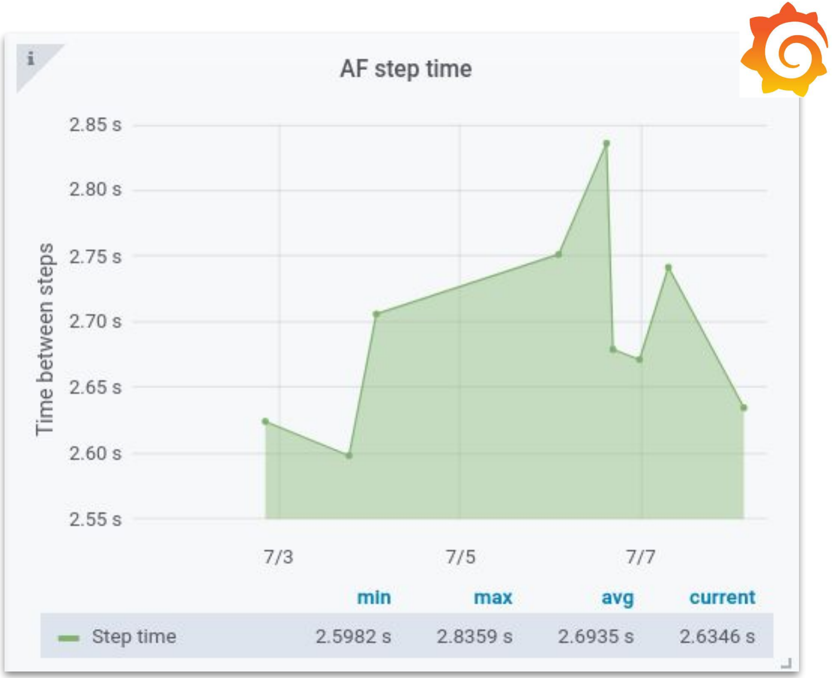


DAG Graph Links for user id: 1[redacted]

Execution_date↕	Dag_id↕	Graph↕	Gantt↕
2020-07-08 05:28:37.777772+00:00	RTOv2	DAG Graph	DAG Gantt

Track scheduler latencies

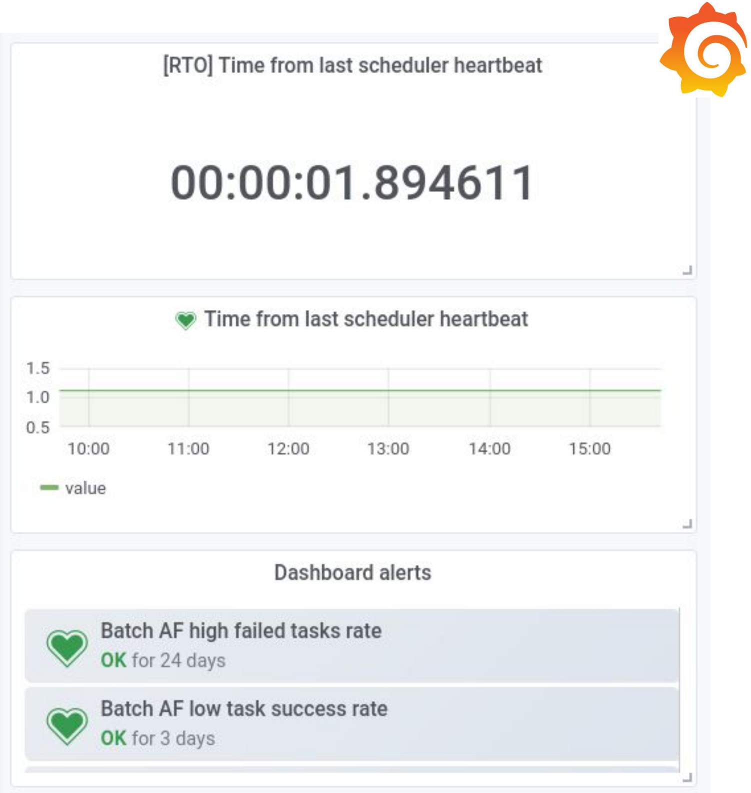
- Query Airflow DB from Grafana
- Query the delta between a time that a task finishes and the time the next one starts



	min	max	avg	current
🟩 Step time	2.5982 s	2.8359 s	2.6935 s	2.6346 s

Scheduler outage alerts

- Airflow most critical component is the scheduler - nothing happens without it
- The scheduler sends a heartbeat to the DB
- Grafana polls on that table to and sends us an alert if the scheduler is down



[RTO] Time from last scheduler heartbeat

00:00:01.894611

Time from last scheduler heartbeat

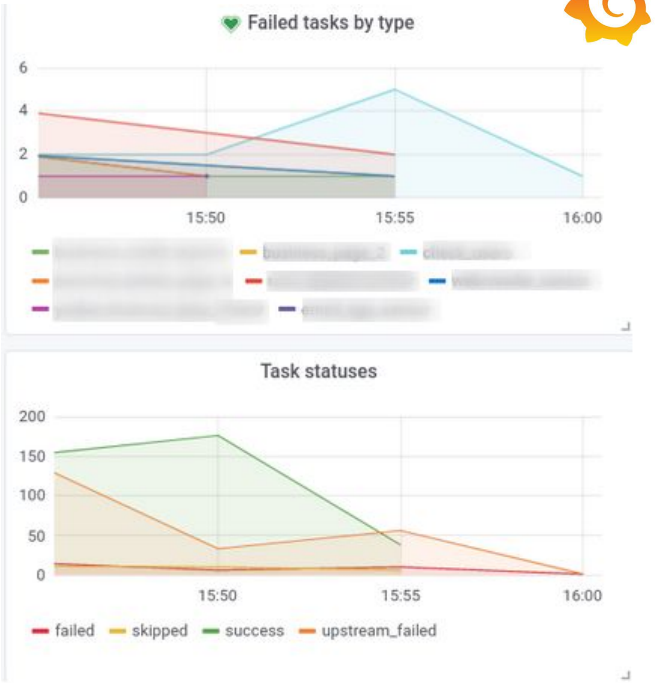
Dashboard alerts

Batch AF high failed tasks rate
OK for 24 days

Batch AF low task success rate
OK for 3 days

Track flow latencies

- Airflow UI is great! But, it doesn’t allow to view aggregated data
- Querying the DB allows to extract great aggregated view that can show the state of the system in a glance
- Grafana is great!



Questions?