AIRFLOW SUMMIT

Purple is the New Green

Harnessing Deferrable Operators to Improve Performance & Reduce Costs

Ethan Shalev

3.0

AIRFLOW SUMMIT

Blue is the New Blue

Harnessing Deferrable Operators to Improve Performance & Reduce Costs

Ethan Shalev

3.1

Ethan Shalev

- Data Engineer @ Wix
- Airflow tech-lead & evangelist
- 20+ years in the field of data

/in/eshalev

Agenda

1. About Wix
2. Motivation for using deferrable operators
3. Explanation of deferrable operators and their design
4. Implementation approach and impact
5. Pitfalls, lessons learned and next steps

WIX

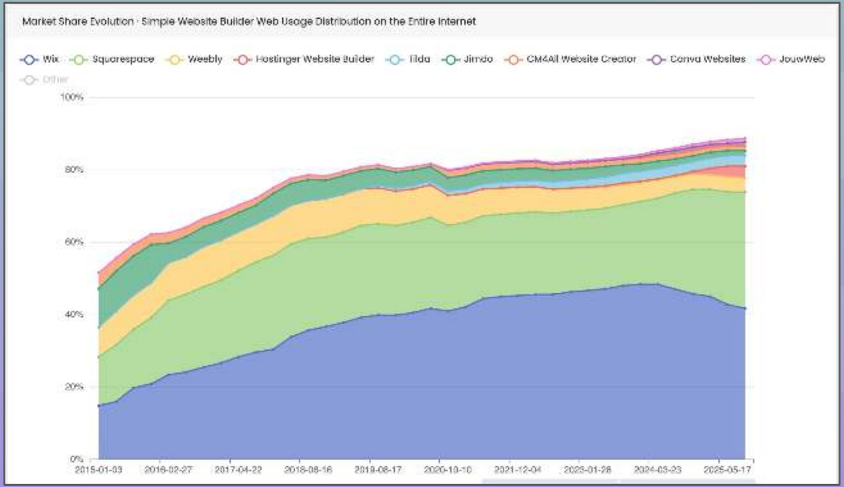
1.

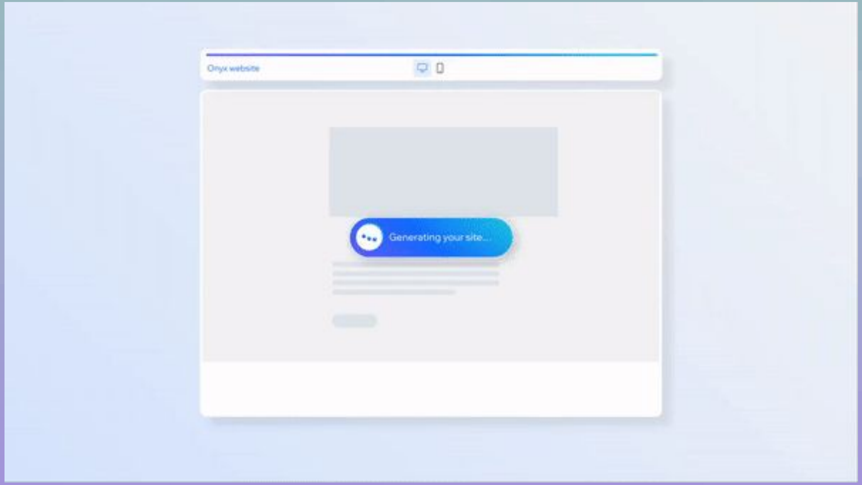
About Wix

WIX

About Wix

- The leading SaaS website builder platform
- Runs ~4% of all active sites on the WWW[1]
- Drives ~40% of traffic to sites created with simple website builders[2]





Data @ Wix

People

- 75 Data Scientists
- 120 Data Engineers
- 240 Business Analysts

Data

- 20TB of data added and processed daily
- 1.2M SQL Queries per day

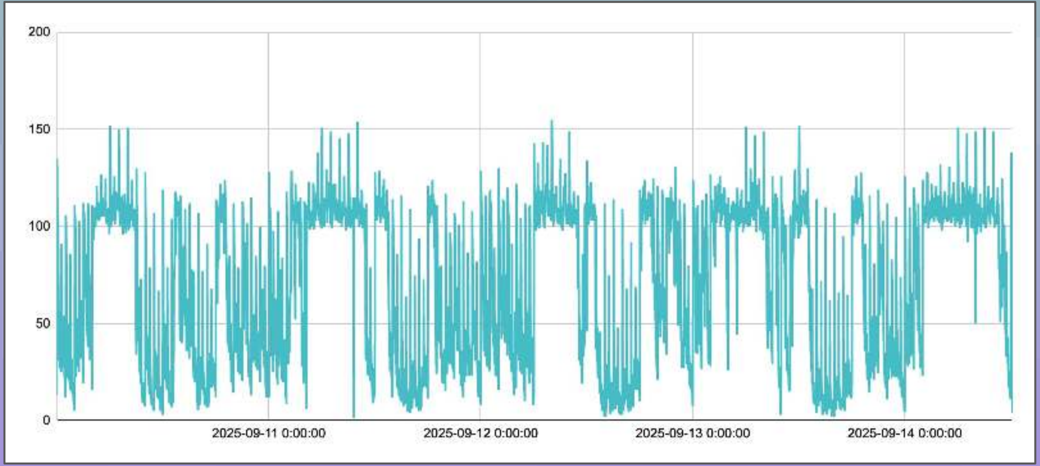
Airflow

- 3 Production Airflow clusters
  - Migrating from 2.6.3 >> 3.1
- 7,500 DAGs
- 270,000 Daily Airflow tasks
- 10,000+ worker hours per day

2
.Motivation

Motivation

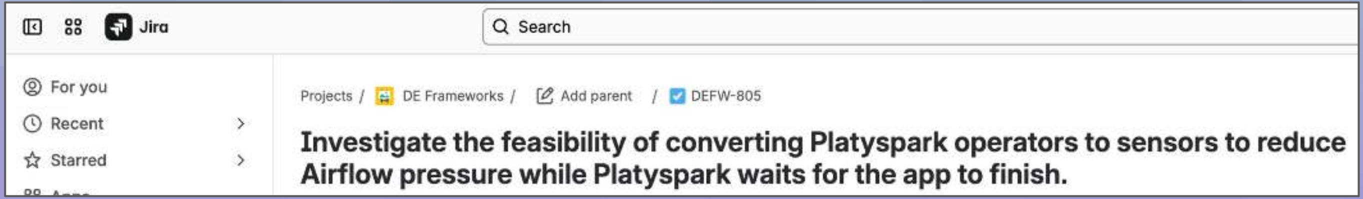
Reduce data-delivery bottlenecks | Reduce Airflow Worker load | Free-up resources | Reduce costs



Exploration

Just increase resources?

Airflow Sensors?

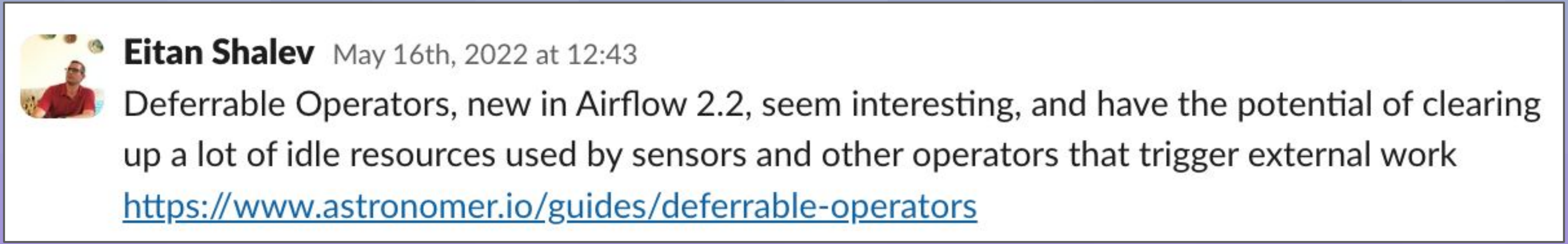


Projects / DE Frameworks / Add parent / DEFW-805

Investigate the feasibility of converting Platyspark operators to sensors to reduce Airflow pressure while Platyspark waits for the app to finish.

Solution

Deferrable operators!


Eitan Shalev May 16th, 2022 at 12:43
Deferrable Operators, new in Airflow 2.2, seem interesting, and have the potential of clearing up a lot of idle resources used by sensors and other operators that trigger external work
https://www.astronomer.io/guides/deferrable-operators

3.

Deferral explained

De·fer (/dəˈfər/):

- To postpone or delay
- To yield to another

Deferrable Operators

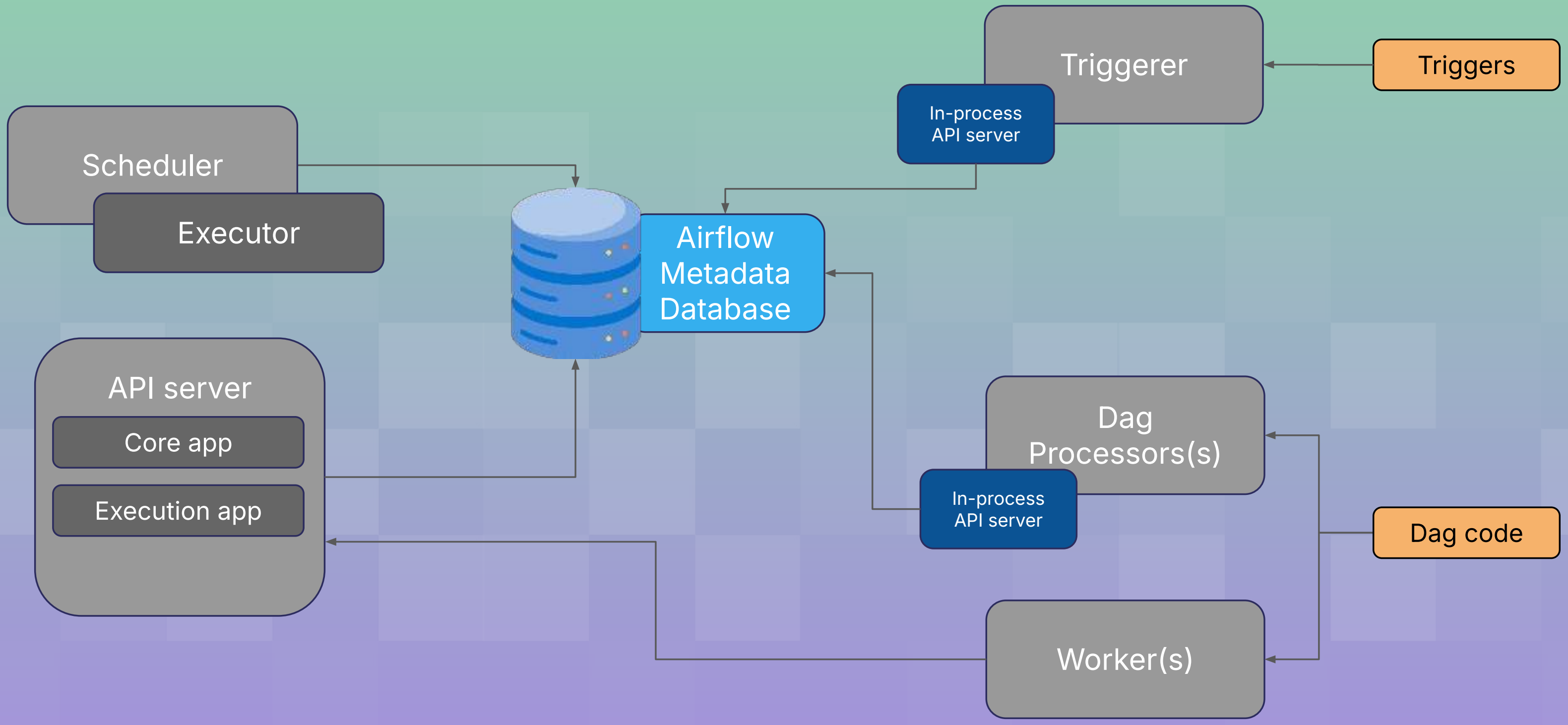
Transfer the management of a task from a worker to a triggerer service while an external process is executing

Standalone process, capable of handling 1,000+ asynchronous triggers

Frees up Airflow worker slot; increases Airflow availability

Deferral is defined at the operator level, invoked either by default or per-task.

Typical execution vs. deferred execution



Typical execution vs. deferred execution

No deferral:

```
class MyShinyOperator(BaseOperator):

    def execute(self, context: Context):
        result = call_external(context)
        # connection open, waiting...

        return result if self.do_xcom_push
```

```
class MyShinyOperator(BaseOperator):

    def execute(self, context: Context):
        request_id = call_external(context)
        while True:
            result = poll_external(request_id)
            if result:
                return result if self.do_xcom_push
            else:
                sleep(30)
```

Typical execution vs. deferred execution

Deferral:

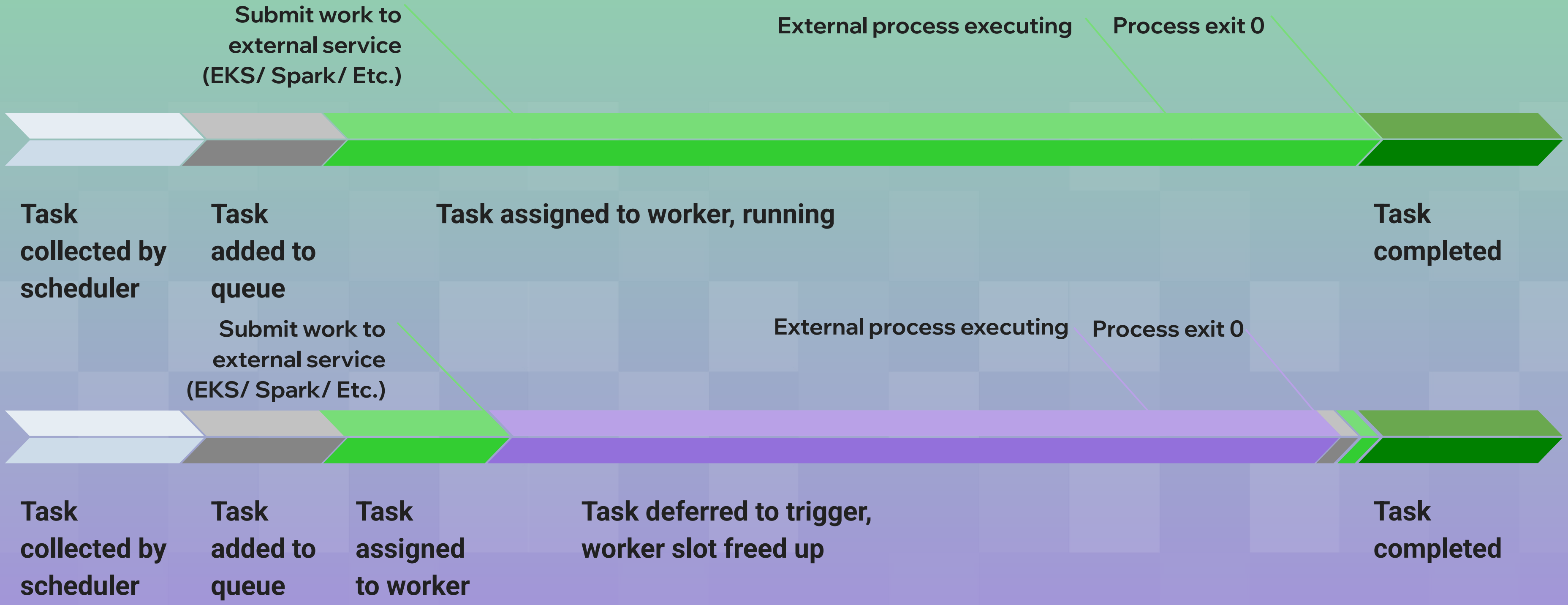
```
class MyShinyOperator(BaseOperator):

    def execute(self, context: Context):
        request_id = call_external(context)

        self.defer(trigger=MyShinyTrigger(request_id),
                   method_name="all_shiny",
                   kwargs=context, #optional
                   timeout=timedelta(minutes=15) #optional
                )

    def all_shiny(context)
        return context.result if self.do_xcom_push
```

What makes a trigger?

```python
import asyncio

from airflow.triggers.base import BaseTrigger, TriggerEvent
from airflow.utils import timezone


class DateTimeTrigger(BaseTrigger):
    def __init__(self, moment):
        super().__init__()
        self.moment = moment

    def serialize(self):
        return (
"airflow.providers.standard.triggers.temporal.DateTimeTrigger",
            {"moment": self.moment}
        )

    async def run(self):
        while self.moment > timezone.utcnow():
            await asyncio.sleep(1)
        yield TriggerEvent(self.moment)
```

GitHub

Typical execution vs. deferred execution


Submit work to external service (EKS/ Spark/ Etc.)
External process executing
Process exit 0
Task collected by scheduler
Task added to queue
Task assigned to worker, running
Task completed
Submit work to external service (EKS/ Spark/ Etc.)
External process executing
Process exit 0
Task collected by scheduler
Task added to queue
Task assigned to worker
Task deferred to trigger, worker slot freed up
Task completed

Applying deferral in your DAG

```
from airflow import DAG
from myOperators.shiny import MyShinyOperator
from datetime import datetime

with DAG(
    dag_id="deferrable_dag_demo",
    start_date=datetime(2025, 10, 8),
    schedule=None
):
    MyShinyOperator(
        task_id="so_shiny",
        shiny_params=params,
        deferrable=True
    )
```

4. Implementation

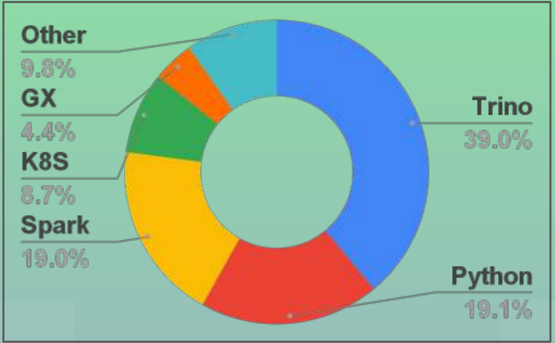
Approach

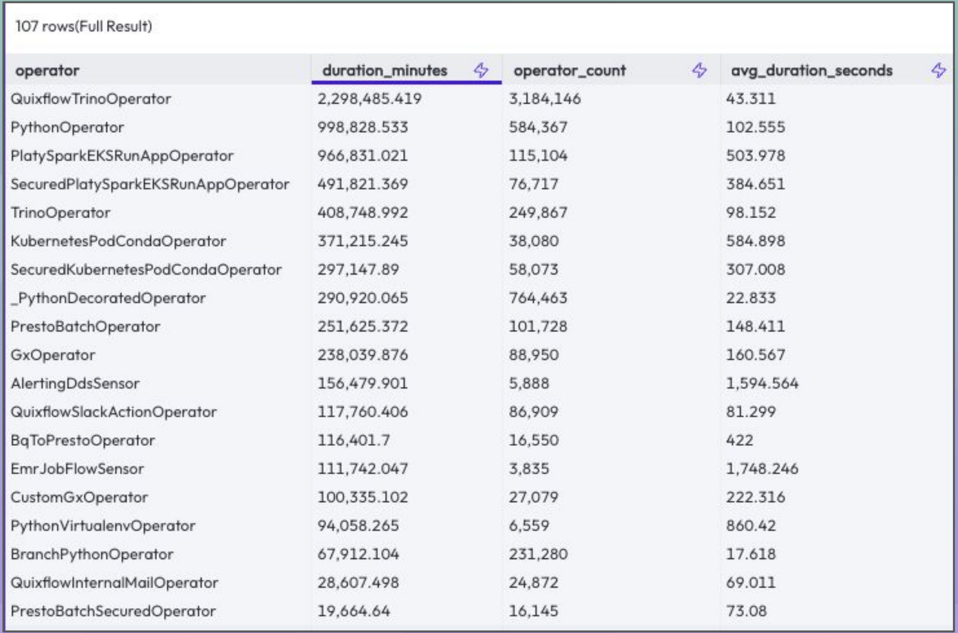
Query Airflow metadata DB | Map Operator-types by average task duration and count | Identify quick-wins | Long average run time, Lots of them

```
SELECT
      Operator,
      Sum(duration) AS duration_seconds,
      Count(1) AS operator_count
FROM
      airflow_db..task_instance
WHERE
      state = 'success'
      AND start_date > date'2025-08-01'
      AND start_date < date'2025-09-01'
GROUP BY 2;
```

Findings

100+ distinct types of operators





107 rows(Full Result)

operator	duration_minutes	operator_count	avg_duration_seconds
QuixflowTrinoOperator	2,298,485.419	3,184,146	43.311
PythonOperator	998,828.533	584,367	102.555
PlatySparkEKSRunAppOperator	966,831.021	115,104	503.978
SecuredPlatySparkEKSRunAppOperator	491,821.369	76,717	384.651
TrinoOperator	408,748.992	249,867	98.152
KubernetesPodCondaOperator	371,215.245	38,080	584.898
SecuredKubernetesPodCondaOperator	297,147.89	58,073	307.008
_PythonDecoratedOperator	290,920.065	764,463	22.833
PrestoBatchOperator	251,625.372	101,728	148.411
GxOperator	238,039.876	88,950	160.567
AlertingDdsSensor	156,479.901	5,888	1,594.564
QuixflowSlackActionOperator	117,760.406	86,909	81.299
BqToPrestoOperator	116,401.7	16,550	422
EmrJobFlowSensor	111,742.047	3,835	1,748.246
CustomGxOperator	100,335.102	27,079	222.316
PythonVirtualenvOperator	94,058.265	6,559	860.42
BranchPythonOperator	67,912.104	231,280	17.618
QuixflowInternalMailOperator	28,607.498	24,872	69.011
PrestoBatchSecuredOperator	19,664.64	16,145	73.08

Custom TrinoOperators
Spark on EKS
Custom KubernetesPod Operators
Custom Great-Expectations (GX) Operators
Data-transfer Operators
(Snowflake to Iceberg to BQ…)
Data-freshness sensors
Slack Operator
OpsGenie Operator
PythonOperator(s)

Where do you start?

trino?

APACHE Spark?

Complex

Long time to delivery

Chance of failure too high

Start where it’s easy!

Create MVP

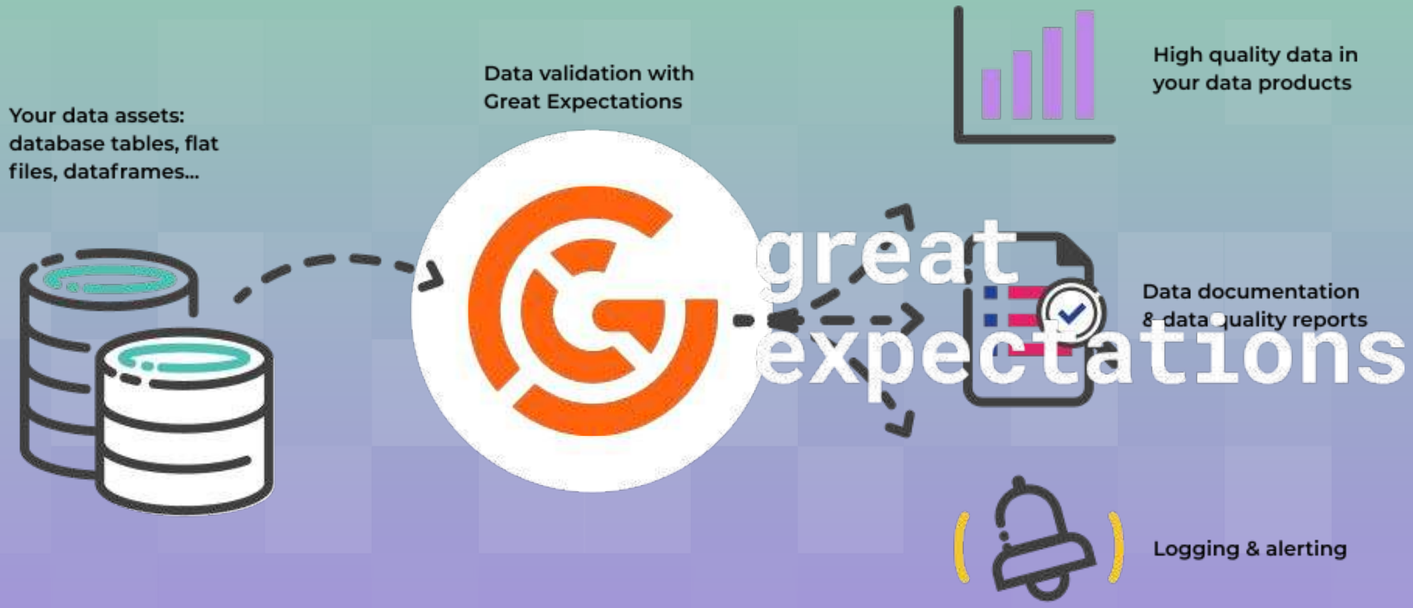
Communicate the feature

Encourage adoption

Recruit support and get others involved

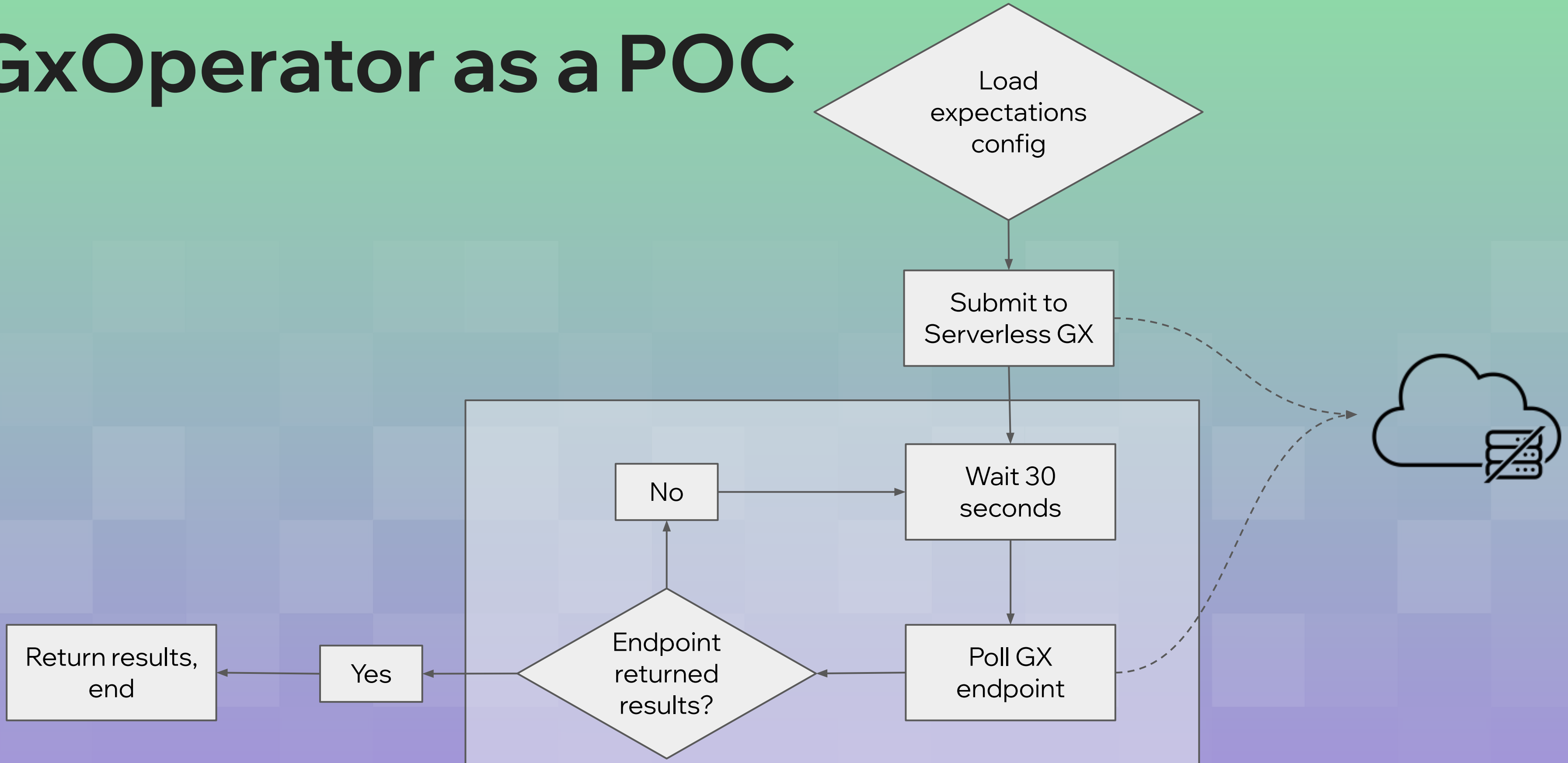
Use momentum to continue

GxOperator as a POC


Your data assets: database tables, flat files, dataframes...
Data validation with Great Expectations
great expectations
High quality data in your data products
Data documentation & data quality reports
Logging & alerting

Great Expectations docs

GxOperator as a POC



Refactoring GxOperator

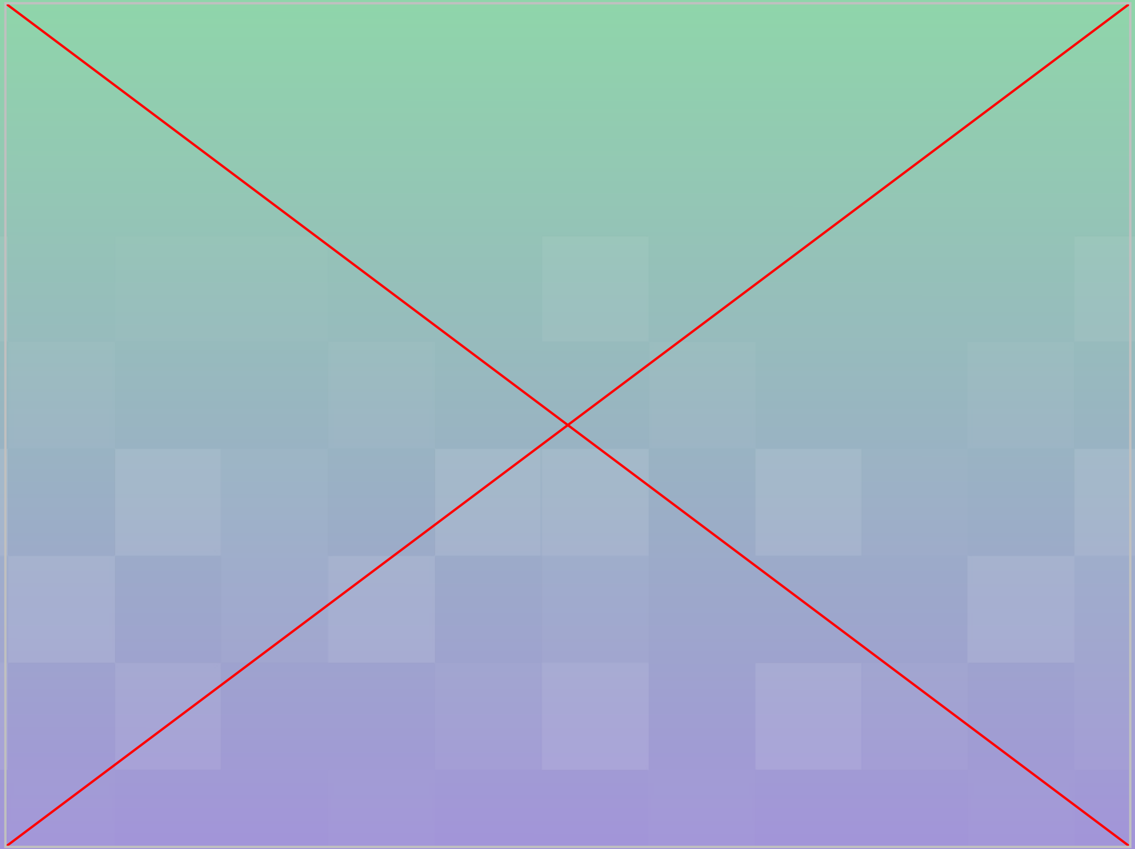
- Examine Execute() method
- Identify where external polling is done
- Replace it with trigger call
- Implement asynchronous trigger

```
    retry_call(self.create_expectation, fargs=(context,), tries=self.create_expectation_config.tries,
                          delay=self.create_expectation_config.delay_in_seconds)
if self.deferrable:
    self.defer(trigger=GxTrigger(self.run_guid),
method_name="process_expectation_results_returned_from_trigger")
else:
    retry_call(self.get_expectation_results, tries=self.get_expectation_config.tries,
                          delay=self.get_expectation_config.delay_in_seconds, exceptions=(ValueError,))
```

GxTrigger

```
async def run(self):
    await asyncio.sleep(self.initial_wait_delay)
    session = requests.Session()
    response = session.get(self.endpoint, params={"run_guid": self.run_guid})

    while not "success" in json.loads(response.text):
        await asyncio.sleep(self.wait_delay)
        response = session.get(self.endpoint, params={"run_guid": self.run_guid})
    yield TriggerEvent({"run_guid": self.run_guid, "response": json.loads(response.text)})
```

Demo

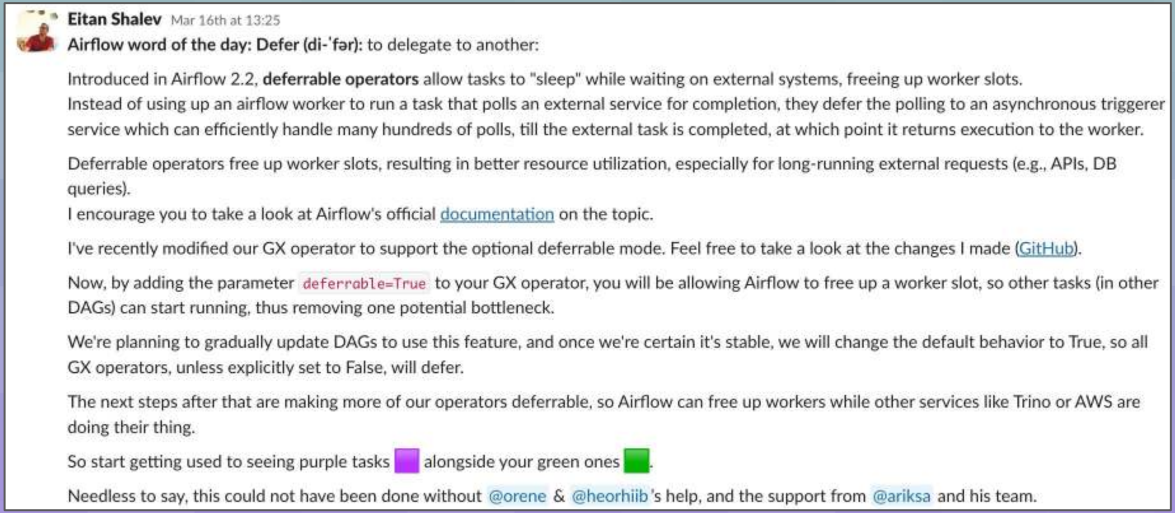


5.

Next Steps

Rollout

- Test locally
- Deploy on select DAGs
- Monitor
- Communicate
- Track adoption
- Scale triggerer
- Set deferral by default



Eitan Shalev Mar 16th at 13:25

Airflow word of the day: Defer (di-ˈfər): to delegate to another:

Introduced in Airflow 2.2, **deferrable operators** allow tasks to "sleep" while waiting on external systems, freeing up worker slots.
Instead of using up an airflow worker to run a task that polls an external service for completion, they defer the polling to an asynchronous triggerer service which can efficiently handle many hundreds of polls, till the external task is completed, at which point it returns execution to the worker.

Deferrable operators free up worker slots, resulting in better resource utilization, especially for long-running external requests (e.g., APIs, DB queries).
I encourage you to take a look at Airflow's official documentation on the topic.

I've recently modified our GX operator to support the optional deferrable mode. Feel free to take a look at the changes I made (GitHub).

Now, by adding the parameter `deferrable=True` to your GX operator, you will be allowing Airflow to free up a worker slot, so other tasks (in other DAGs) can start running, thus removing one potential bottleneck.

We're planning to gradually update DAGs to use this feature, and once we're certain it's stable, we will change the default behavior to True, so all GX operators, unless explicitly set to False, will defer.

The next steps after that are making more of our operators deferrable, so Airflow can free up workers while other services like Trino or AWS are doing their thing.

So start getting used to seeing purple tasks 🟪 alongside your green ones 🟩.

Needless to say, this could not have been done without @orene & @heorhiib's help, and the support from @ariksa and his team.

Pitfalls

Ensure **gradual** rollout

Don't start by deferring a DAG with 1,000 tasks

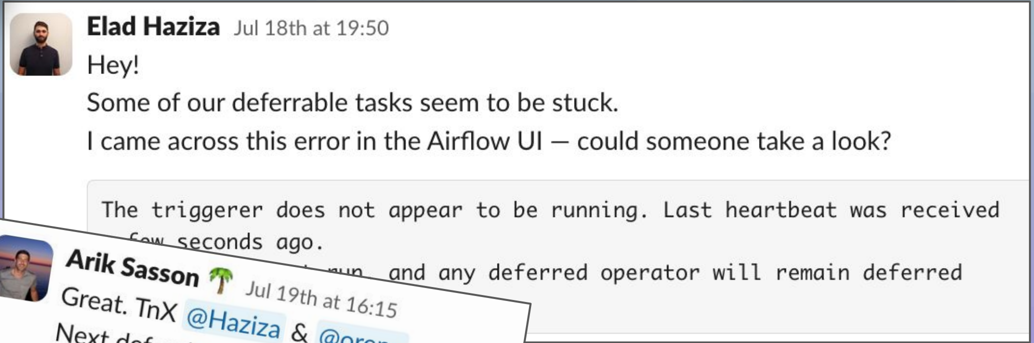
On a weekend

All at once

Track Triggerer process performance

Scale up as needed

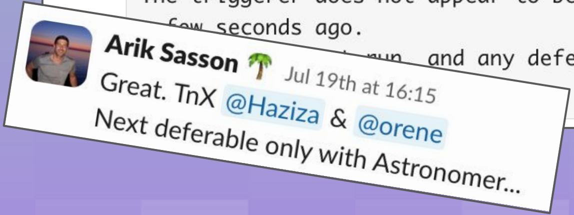
Track # of concurrent deferred tasks



Elad Haziza Jul 18th at 19:50

Hey!
Some of our deferrable tasks seem to be stuck.
I came across this error in the Airflow UI — could someone take a look?

```
The triggerer does not appear to be running. Last heartbeat was received
 few seconds ago.
 run, and any deferred operator will remain deferred
```

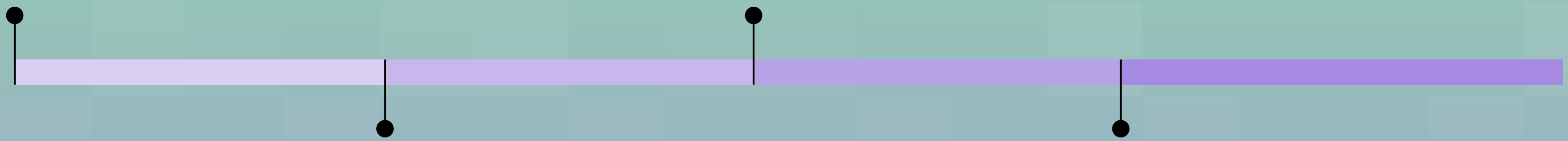


Arik Sasson 🌴 Jul 19th at 16:15

Great. TnX @Haziza & @orene
Next deferable only with Astronomer...

WIX

Next steps

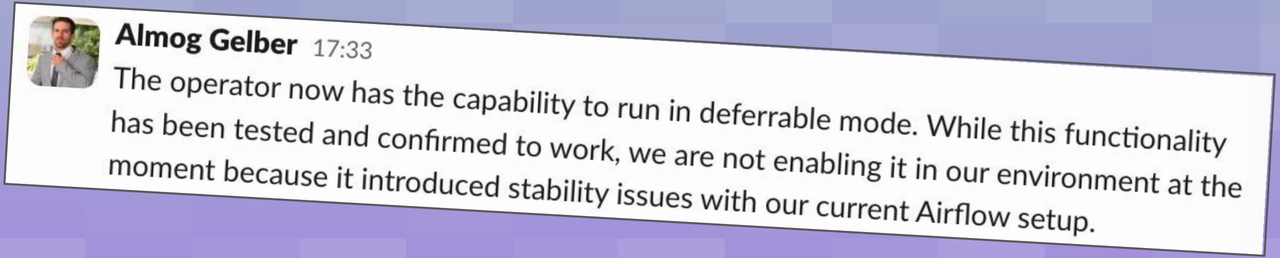


Custom SparkOperators

Defer natively deferrable operators (Amazon, etc.)

TrinoOperators (SqlOperator)

Migrate custom PythonOperators to deferrables



Almog Gelber 17:33

The operator now has the capability to run in deferrable mode. While this functionality has been tested and confirmed to work, we are not enabling it in our environment at the moment because it introduced stability issues with our current Airflow setup.

Expected Impact

Operator	Worker Hr/Day saved	Cost savings $/Month	CO_2 Emissions reduced (Estimate)
GxOperator	450 (4.5%)	$600	0.6 tCO_2/ Year
SparkOperators	2,000 (20%)	$3,200	3 tCO_2/ Year
TrinoOperators	4,000 (40%)	$6,400	6 tCO_2/ Year

Conclusions

Deferrable operators improve performance & reduce costs

Easier than they seem at first glance

Do it!

Thank you!

Questions?

/in/eshalev

eitansh@wix.com