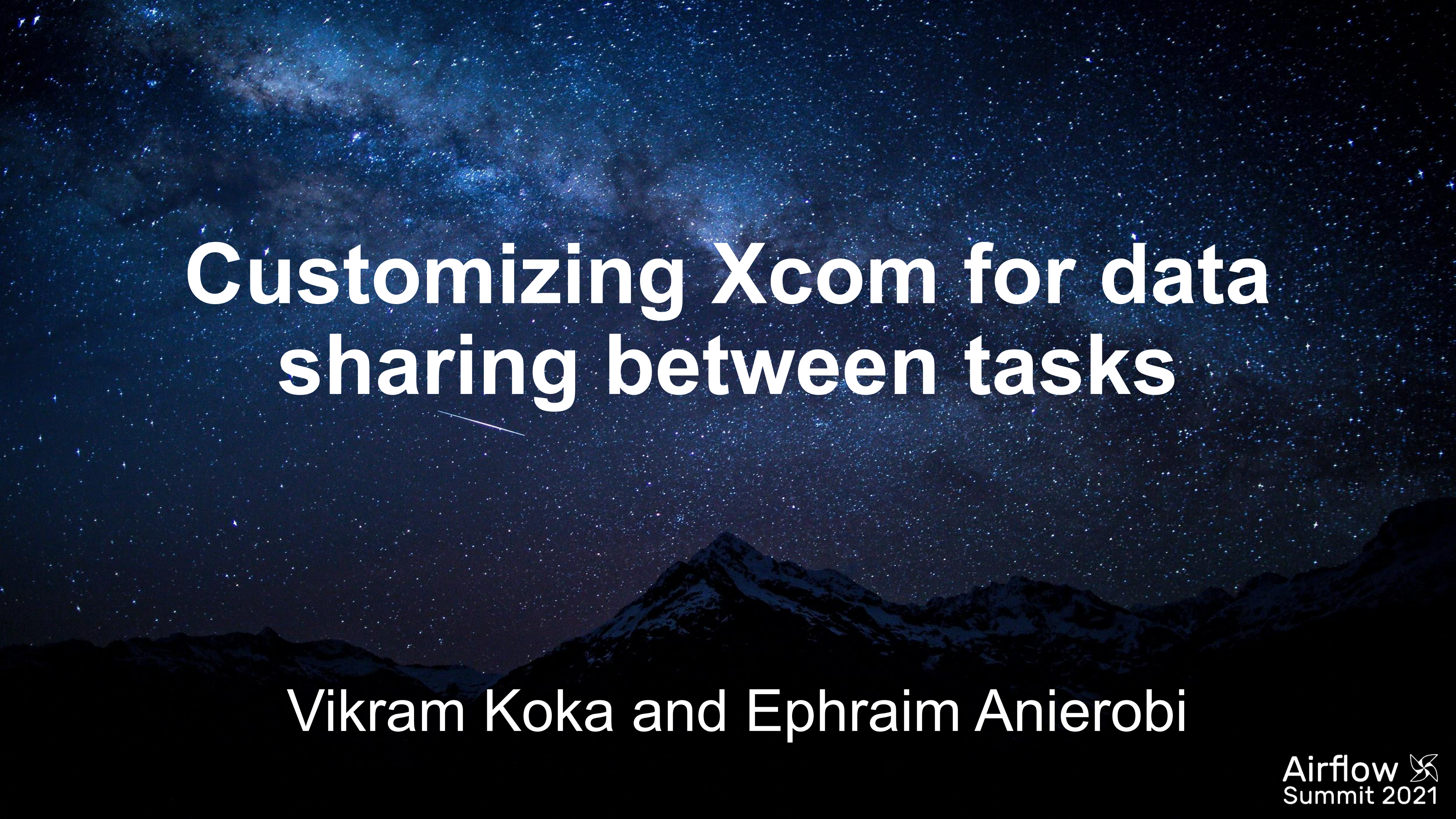


Customizing Xcom for data sharing between tasks

Vikram Koka and Ephraim Anierobi

Airflow
Summit 2021

Introductions

Vikram Koka

Apache Airflow Committer

Senior Vice President Engineering at Astronomer

Silicon Valley

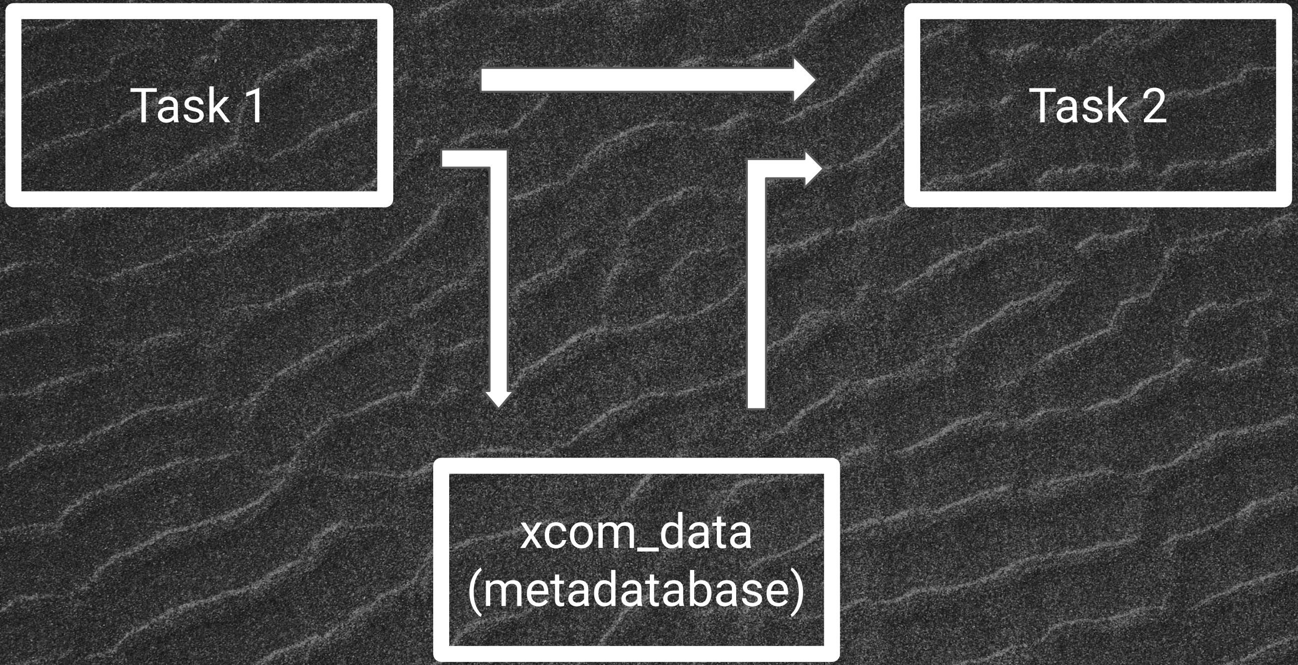
Ephraim Anierobi

Apache Airflow Committer

Software Engineer, Open Source at Astronomer

Nigeria

Xcom in a nutshell



Xcom Overview

Cross communication between tasks

- Pass parameters from one task to another
- Supports multiple parameters
- Identified by key
- Intended for use within a single DAG

Usage:

- “push” and “pull”

Uses the Airflow metadatabase (Postgres / MySQL)

```
xcom_push(
    key = ‘return_value’,
    value = ‘my value’
)

value = xcom_pull(
    task_ids=‘pushing_task’,
    key=‘return_value’
)
```

Xcom with TaskFlow API

Greater Abstraction

- Return values implicitly use xcom
- Focused on the most common pattern
- Supports python native types including dict

```
def extract:
    …
    return order_data
    …
```

Pythonic functional use

```
order_data  = extract()
order_summary =
transform(order_data)
```

Xcom limitations

Data types

As it stands, only the following datatypes are supported in Airflow 2.0

- Python native: dict, list, tuple, str, int, long, float, True, False, None
- Future: Airflow supported objects such as numpy objects, datetime, date, etc
- For security, pickling is no longer recommended

Github issues

Sample questions / problems

- Unable to store xcom because of MySQL Blob type limitation 65,535
- Data too long when pushing to XCOM
- Raise do_xcom_push size limit
- Lambda to transform response before xcom push
- Provide shared storage between task via pluggable storage providers akin to S3 remote logging

Custom XCom backends

Persistence class

- Python class specified in config
- Read at Airflow start up,

  Class needs to be in Airflow path

Methods needed:

- `serialize_value`
- `deserialize_value`

Used for storing and restoring data

- `orm_deseralize_value`

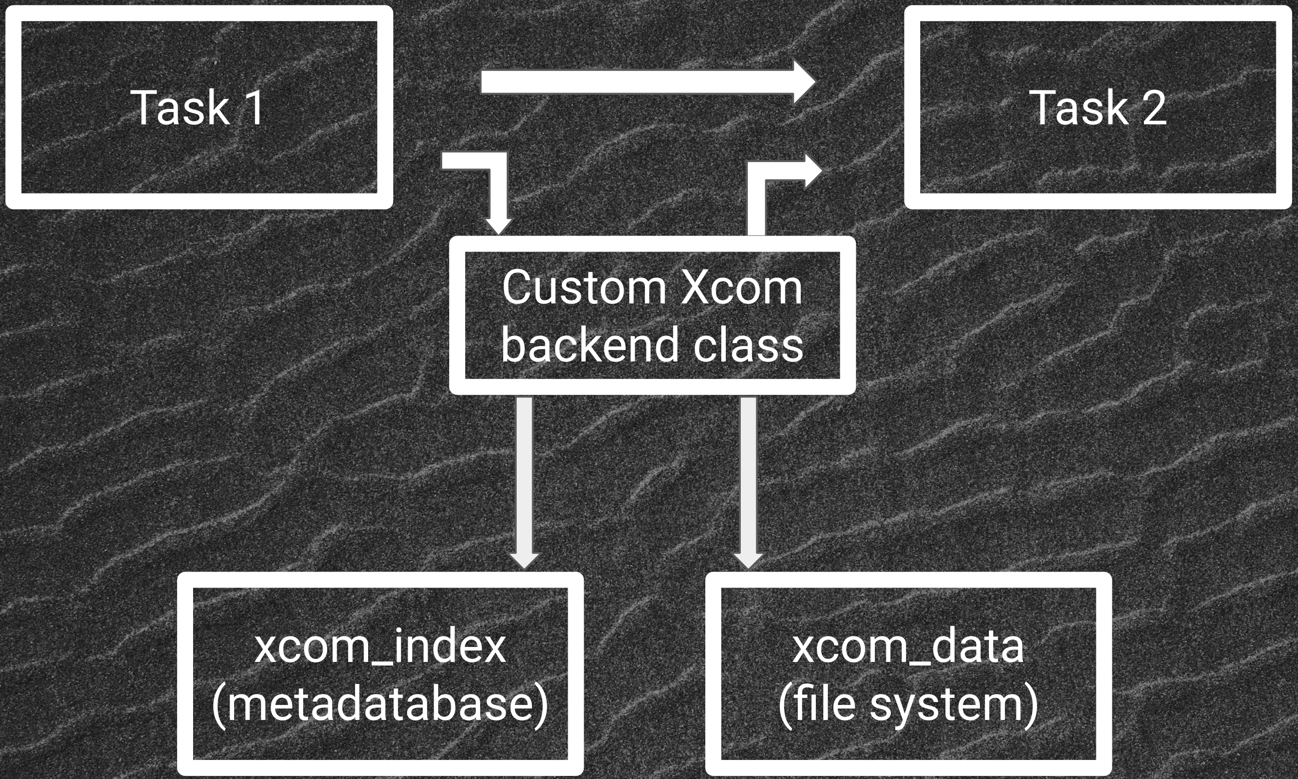
Used to display XCom data in UI

Custom XCom for Local Execution

- Write / read local file system
- Essential for development
- Local Executor

Not for distributed deployments with celery and Kubernetes Executors

Xcom stored in local file system



Task 1

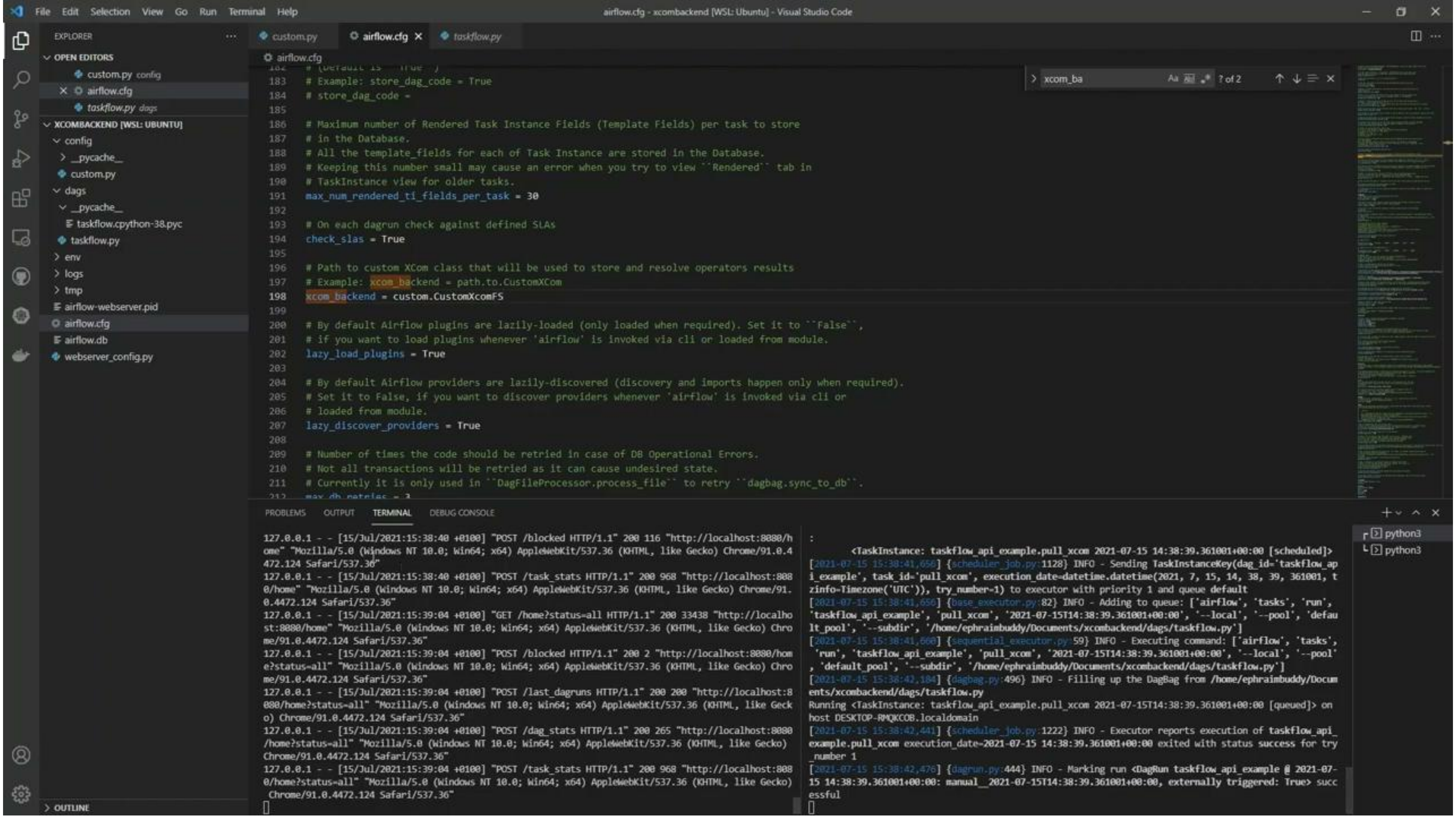
Task 2

Custom Xcom backend class

xcom_index (metadatabase)

xcom_data (file system)

Code walk through and demo

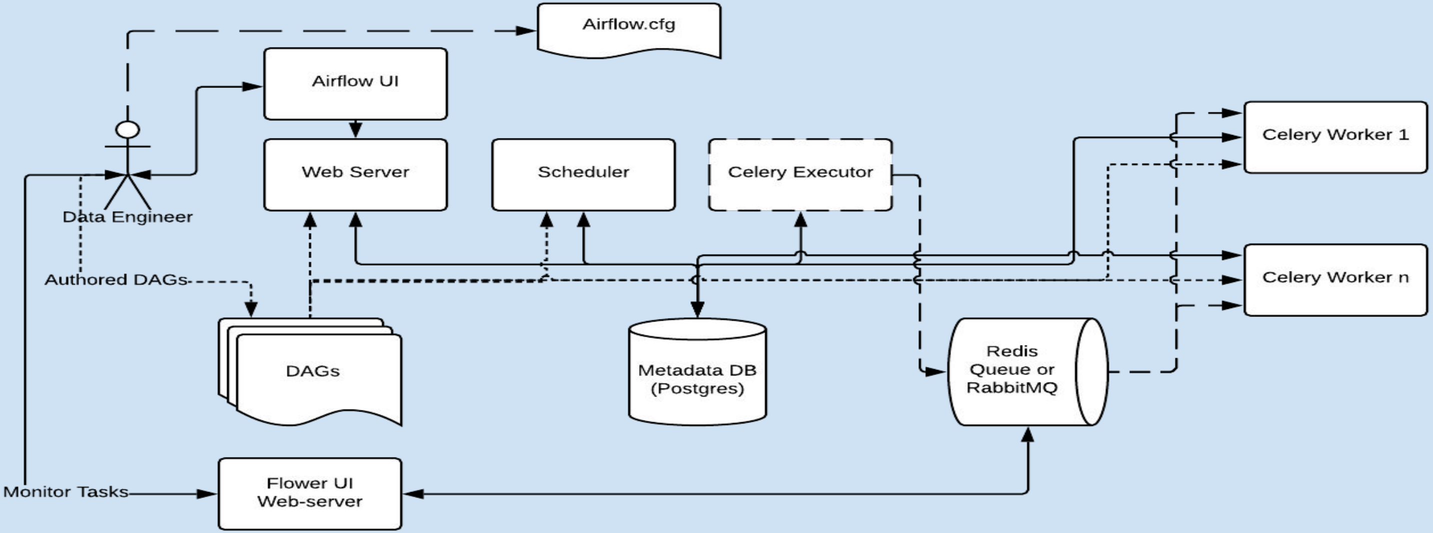


```
# (Default is ``True``)
# Example: store_dag_code = True
# store_dag_code =

# Maximum number of Rendered Task Instance Fields (Template Fields) per task to store
# in the Database.
# All the template_fields for each of Task Instance are stored in the Database.
# Keeping this number small may cause an error when you try to view ``Rendered`` tab in
# TaskInstance view for older tasks.
max_num_rendered_ti_fields_per_task = 30

# On each dagrun check against defined SLAs
check_slas = True

# Path to custom XCom class that will be used to store and resolve operators results
# Example: xcom_backend = path.to.CustomXCom
xcom_backend = custom.CustomXcomFS

# By default Airflow plugins are lazily-loaded (only loaded when required). Set it to ``False``,
# if you want to load plugins whenever 'airflow' is invoked via cli or loaded from module.
lazy_load_plugins = True

# By default Airflow providers are lazily-discovered (discovery and imports happen only when required).
# Set it to False, if you want to discover providers whenever 'airflow' is invoked via cli or
# loaded from module.
lazy_discover_providers = True

# Number of times the code should be retried in case of DB Operational Errors.
# Not all transactions will be retried as it can cause undesired state.
# Currently it is only used in ``DagFileProcessor.process_file`` to retry ``dagbag.sync_to_db``.
max_db_retries = 3
```

```
127.0.0.1 - - [15/Jul/2021:15:38:40 +0100] "POST /blocked HTTP/1.1" 200 116 "http://localhost:8080/h
ome" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4
472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:15:38:40 +0100] "POST /task_stats HTTP/1.1" 200 968 "http://localhost:808
0/home" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.
0.4472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:15:39:04 +0100] "GET /home?status=all HTTP/1.1" 200 33438 "http://localho
st:8080/home" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
me/91.0.4472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:15:39:04 +0100] "POST /blocked HTTP/1.1" 200 2 "http://localhost:8080/hom
e?status=all" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
me/91.0.4472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:15:39:04 +0100] "POST /last_dagruns HTTP/1.1" 200 200 "http://localhost:8
080/home?status=all" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/91.0.4472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:15:39:04 +0100] "POST /dag_stats HTTP/1.1" 200 265 "http://localhost:8080
/home?status=all" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:15:39:04 +0100] "POST /task_stats HTTP/1.1" 200 968 "http://localhost:808
0/home?status=all" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/91.0.4472.124 Safari/537.36"
```

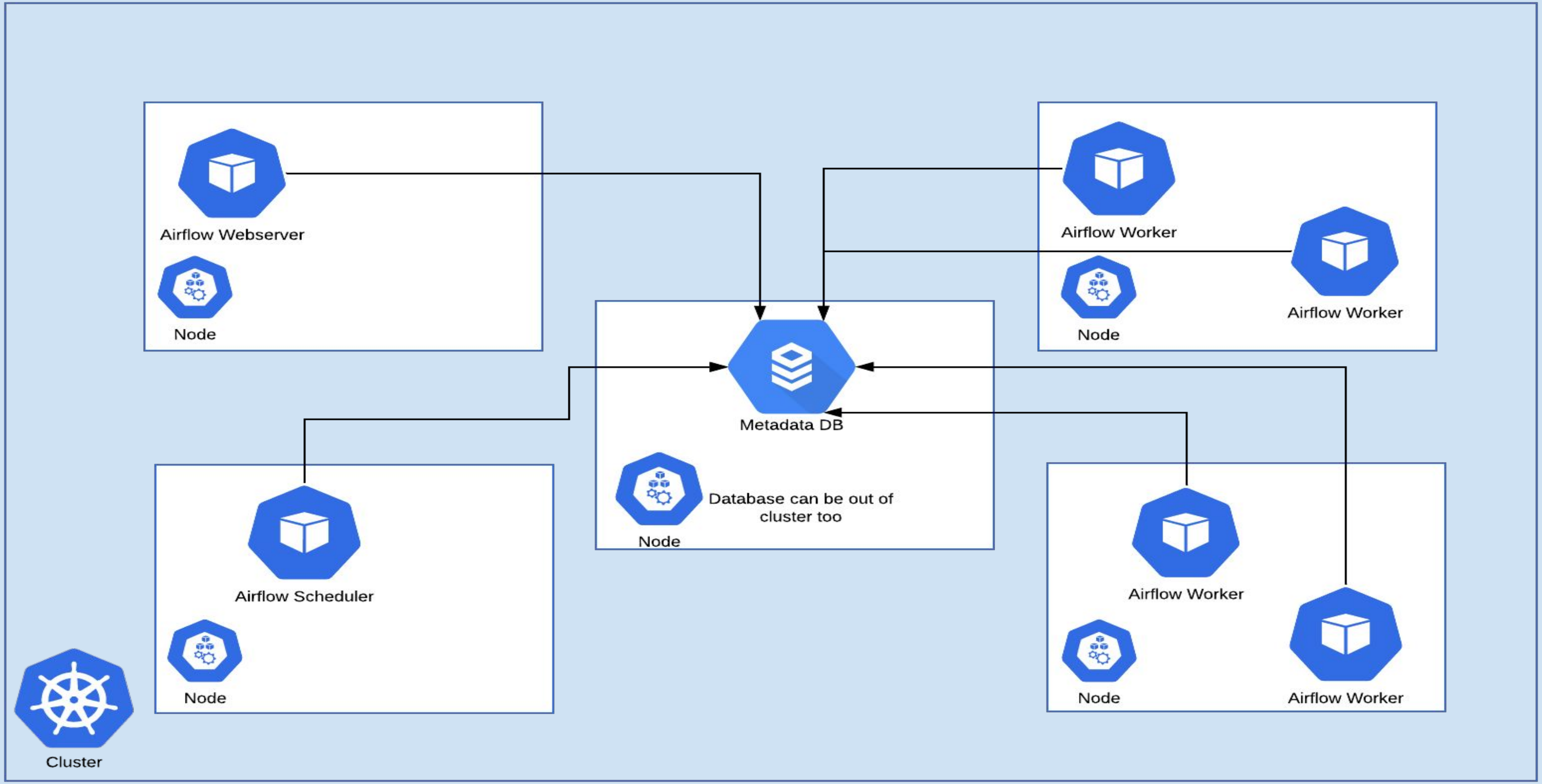
```
:
        <TaskInstance: taskflow_api_example.pull_xcom 2021-07-15 14:38:39.361001+00:00 [scheduled]>
[2021-07-15 15:38:41,656] {scheduler_job.py:1128} INFO - Sending TaskInstanceKey(dag_id='taskflow_ap
i_example', task_id='pull_xcom', execution_date=datetime.datetime(2021, 7, 15, 14, 38, 39, 361001, t
zinfo=Timezone('UTC')), try_number=1) to executor with priority 1 and queue default
[2021-07-15 15:38:41,656] {base_executor.py:82} INFO - Adding to queue: ['airflow', 'tasks', 'run',
'taskflow_api_example', 'pull_xcom', '2021-07-15T14:38:39.361001+00:00', '--local', '--pool', 'defau
lt_pool', '--subdir', '/home/ephraimbuddy/Documents/xcombackend/dags/taskflow.py']
[2021-07-15 15:38:41,660] {sequential_executor.py:59} INFO - Executing command: ['airflow', 'tasks',
 'run', 'taskflow_api_example', 'pull_xcom', '2021-07-15T14:38:39.361001+00:00', '--local', '--pool'
, 'default_pool', '--subdir', '/home/ephraimbuddy/Documents/xcombackend/dags/taskflow.py']
[2021-07-15 15:38:42,184] {dagbag.py:496} INFO - Filling up the DagBag from /home/ephraimbuddy/Docum
ents/xcombackend/dags/taskflow.py
Running <TaskInstance: taskflow_api_example.pull_xcom 2021-07-15T14:38:39.361001+00:00 [queued]> on
host DESKTOP-RMQKCOB.localdomain
[2021-07-15 15:38:42,441] {scheduler_job.py:1222} INFO - Executor reports execution of taskflow_api_
example.pull_xcom execution_date=2021-07-15 14:38:39.361001+00:00 exited with status success for try
_number 1
[2021-07-15 15:38:42,476] {dagrun.py:444} INFO - Marking run <DagRun taskflow_api_example @ 2021-07-
15 14:38:39.361001+00:00: manual__2021-07-15T14:38:39.361001+00:00, externally triggered: True> succ
essful
```

Airflow distributed execution

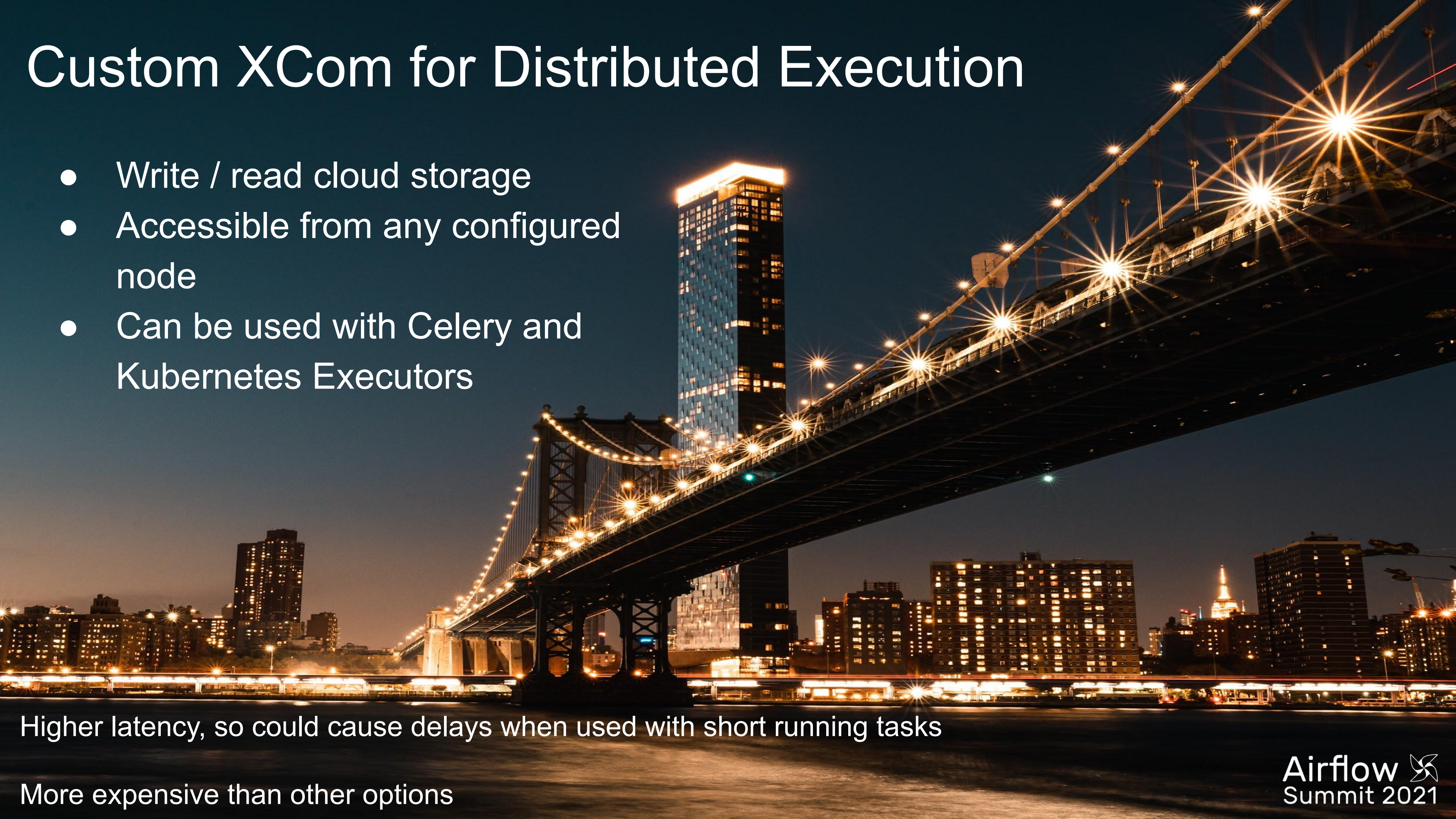


Airflow Summit 2021

Airflow distributed execution - kubernetes



Custom XCom for Distributed Execution



- Write / read cloud storage
- Accessible from any configured node
- Can be used with Celery and Kubernetes Executors

Higher latency, so could cause delays when used with short running tasks

More expensive than other options

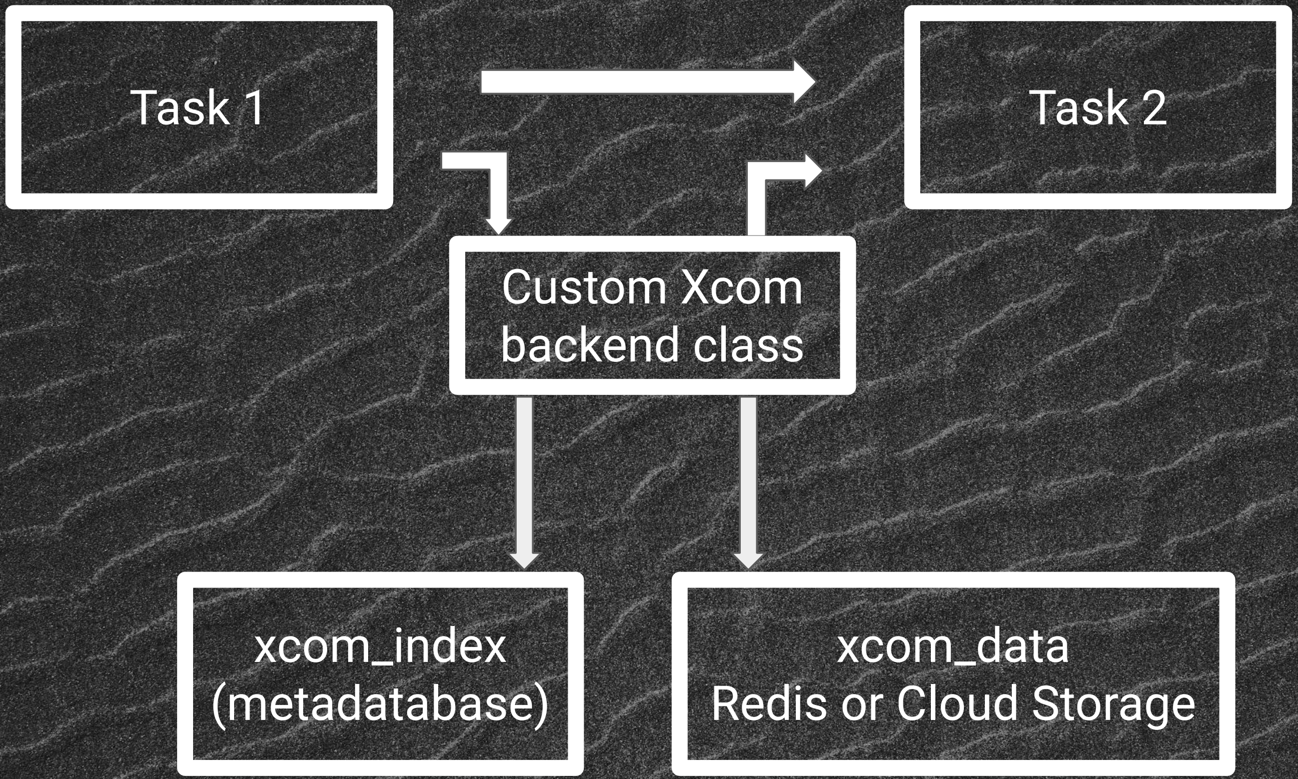
Custom Xcom for Distributed Execution

- Write / read from Redis
- Accessible from any configured node
- Can be used with Celery and Kubernetes Executors
- Already part of the Airflow stack

Size limit of 512MB, so ideal for smaller dataset between short running tasks.

Another caveat that Redis keeps everything in memory.

Xcom stored in Redis or Cloud Storage



Task 1

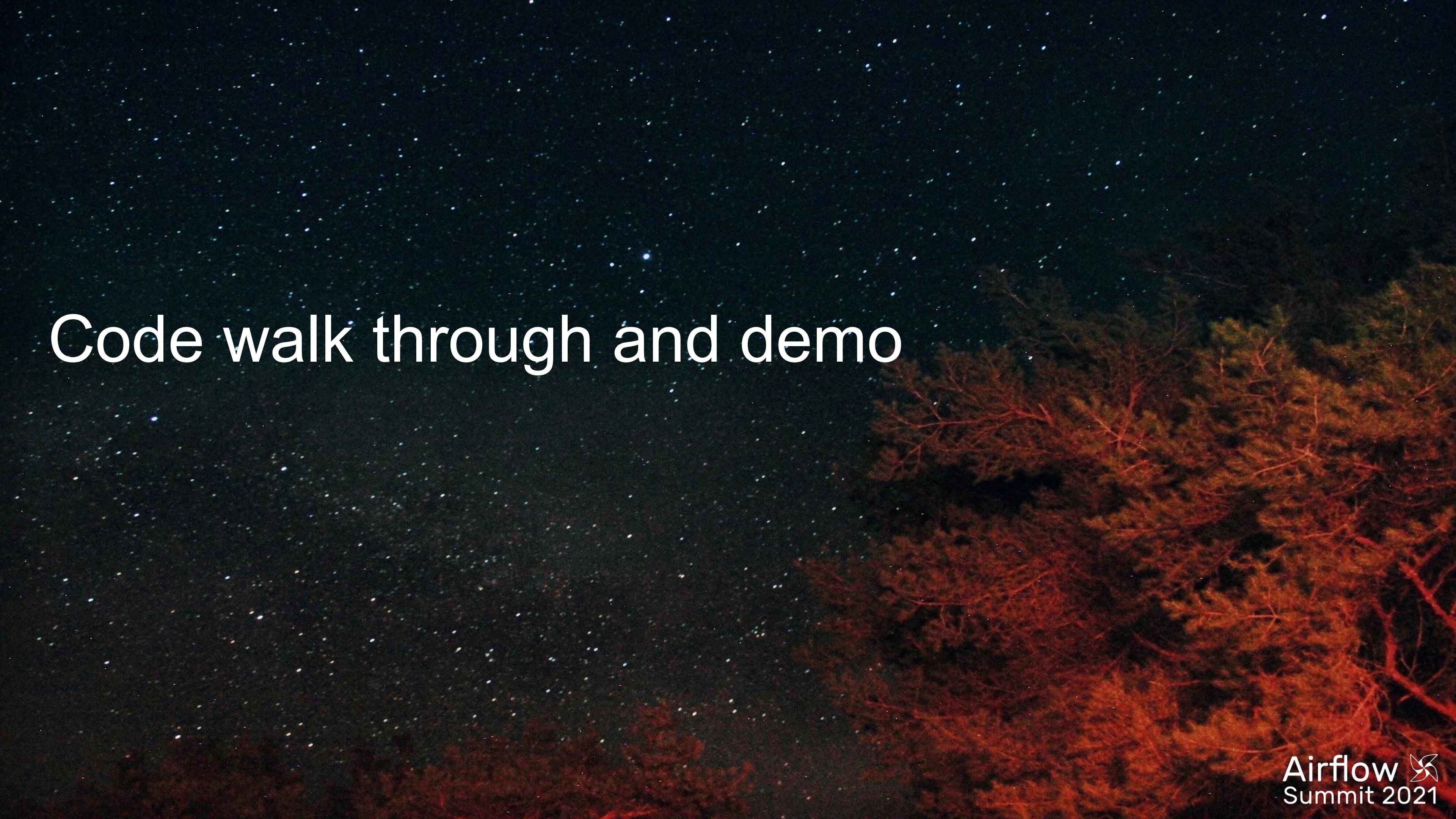
Task 2

Custom Xcom backend class

xcom_index (metadatabase)

xcom_data Redis or Cloud Storage

Code walk through and demo



Clutter

- Clean-up of old data in cloud storage or elsewhere

- As data gets larger, data cleanup becomes more important

- System performance can degrade

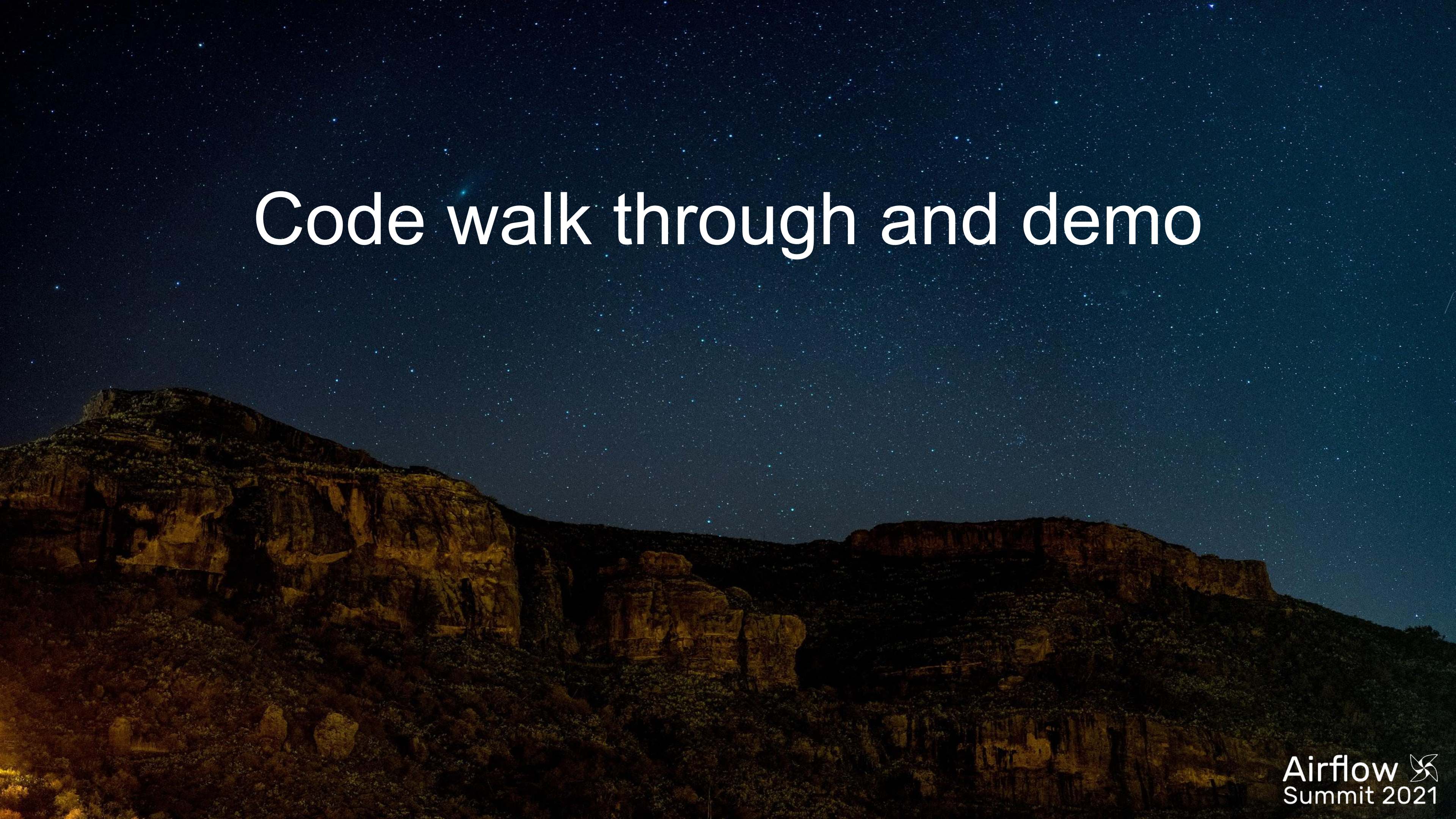
Clean-up DAG

- Maintenance DAG to clean-up old Xcom data

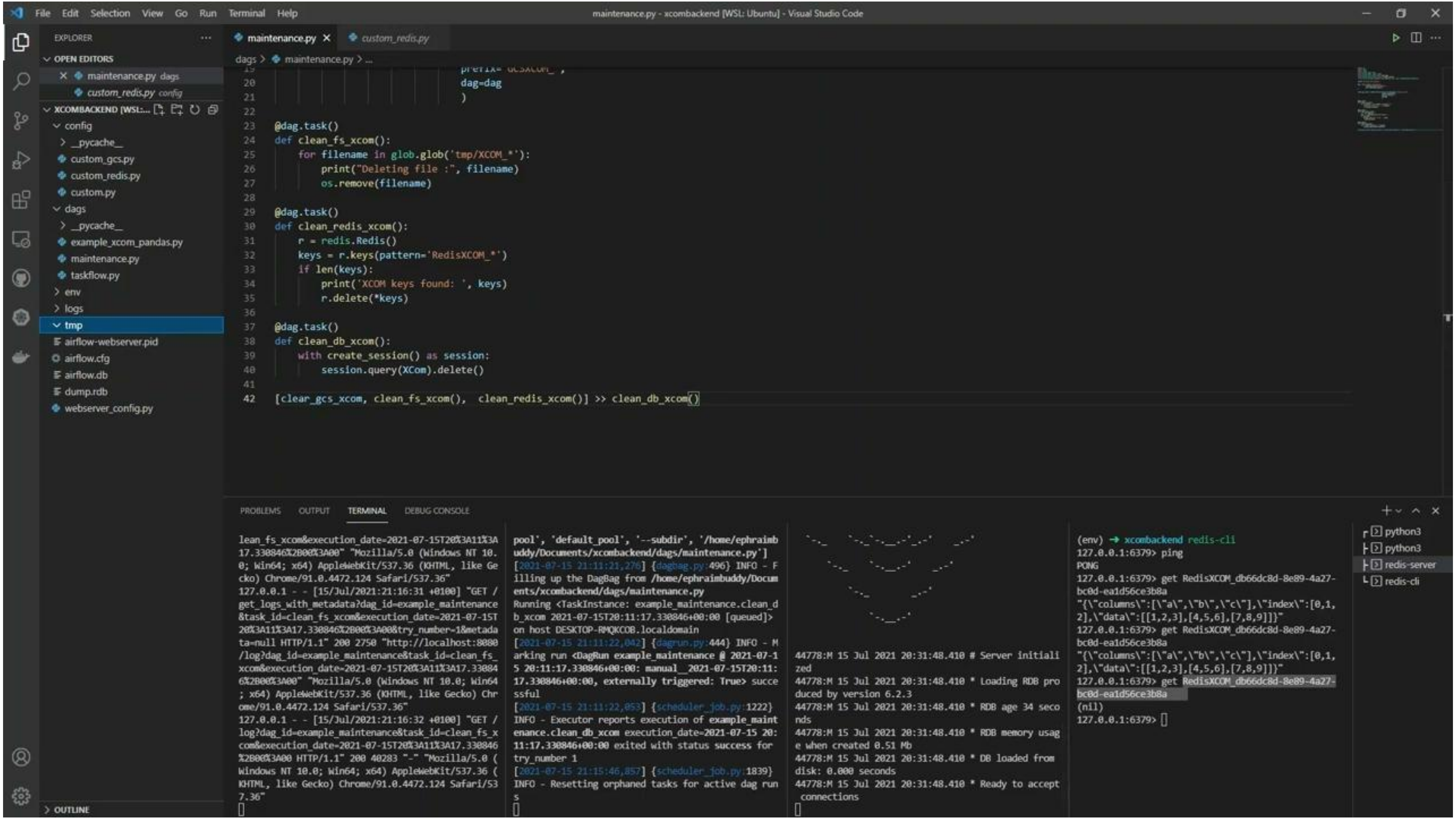
- Deletes data from metadatabase and external locations

- Not tied to DAG lifecycle- needs to be configured carefully

- Downside if trying to rerun old tasks



Code walk through and demo



```python
                                prefix='GCSXCOM_',
                                dag=dag
                                )

@dag.task()
def clean_fs_xcom():
    for filename in glob.glob('tmp/XCOM_*'):
        print("Deleting file :", filename)
        os.remove(filename)

@dag.task()
def clean_redis_xcom():
    r = redis.Redis()
    keys = r.keys(pattern='RedisXCOM_*')
    if len(keys):
        print('XCOM keys found: ', keys)
        r.delete(*keys)

@dag.task()
def clean_db_xcom():
    with create_session() as session:
        session.query(XCom).delete()

[clear_gcs_xcom, clean_fs_xcom(),  clean_redis_xcom()] >> clean_db_xcom()
```

```
lean_fs_xcom&execution_date=2021-07-15T20%3A11%3A
17.330846%2B00%3A00" "Mozilla/5.0 (Windows NT 10.
0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/91.0.4472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:21:16:31 +0100] "GET /
get_logs_with_metadata?dag_id=example_maintenance
&task_id=clean_fs_xcom&execution_date=2021-07-15T
20%3A11%3A17.330846%2B00%3A00&try_number=1&metada
ta=null HTTP/1.1" 200 2750 "http://localhost:8080
/log?dag_id=example_maintenance&task_id=clean_fs_
xcom&execution_date=2021-07-15T20%3A11%3A17.33084
6%2B00%3A00" "Mozilla/5.0 (Windows NT 10.0; Win64
; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr
ome/91.0.4472.124 Safari/537.36"
127.0.0.1 - - [15/Jul/2021:21:16:32 +0100] "GET /
log?dag_id=example_maintenance&task_id=clean_fs_x
com&execution_date=2021-07-15T20%3A11%3A17.330846
%2B00%3A00 HTTP/1.1" 200 40283 "-" "Mozilla/5.0 (
Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
KHTML, like Gecko) Chrome/91.0.4472.124 Safari/53
7.36"
```

```
pool', 'default_pool', '--subdir', '/home/ephraimb
uddy/Documents/xcombackend/dags/maintenance.py']
[2021-07-15 21:11:21,276] {dagbag.py:496} INFO - F
illing up the DagBag from /home/ephraimbuddy/Docum
ents/xcombackend/dags/maintenance.py
Running <TaskInstance: example_maintenance.clean_d
b_xcom 2021-07-15T20:11:17.330846+00:00 [queued]>
on host DESKTOP-RMQKCOB.localdomain
[2021-07-15 21:11:22,042] {dagrun.py:444} INFO - M
arking run <DagRun example_maintenance @ 2021-07-1
5 20:11:17.330846+00:00: manual__2021-07-15T20:11:
17.330846+00:00, externally triggered: True> succe
ssful
[2021-07-15 21:11:22,053] {scheduler_job.py:1222}
INFO - Executor reports execution of example_maint
enance.clean_db_xcom execution_date=2021-07-15 20:
11:17.330846+00:00 exited with status success for
try_number 1
[2021-07-15 21:15:46,857] {scheduler_job.py:1839}
INFO - Resetting orphaned tasks for active dag run
s
```

```
 `-._    `-._`-.__.-'_.-'    _.-'
     `-._     `-.__.-'     _.-'
         `-._          _.-'
             `-.__.-'

44778:M 15 Jul 2021 20:31:48.410 # Server initiali
zed
44778:M 15 Jul 2021 20:31:48.410 * Loading RDB pro
duced by version 6.2.3
44778:M 15 Jul 2021 20:31:48.410 * RDB age 34 seco
nds
44778:M 15 Jul 2021 20:31:48.410 * RDB memory usag
e when created 0.51 Mb
44778:M 15 Jul 2021 20:31:48.410 * DB loaded from
disk: 0.000 seconds
44778:M 15 Jul 2021 20:31:48.410 * Ready to accept
 connections
```

```
(env) ➜ xcombackend redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> get RedisXCOM_db66dc8d-8e89-4a27-
bc0d-ea1d56ce3b8a
"{\"columns\":[\"a\",\"b\",\"c\"],\"index\":[0,1,
2],\"data\":[[1,2,3],[4,5,6],[7,8,9]]}"
127.0.0.1:6379> get RedisXCOM_db66dc8d-8e89-4a27-
bc0d-ea1d56ce3b8a
"{\"columns\":[\"a\",\"b\",\"c\"],\"index\":[0,1,
2],\"data\":[[1,2,3],[4,5,6],[7,8,9]]}"
127.0.0.1:6379> get RedisXCOM_db66dc8d-8e89-4a27-
bc0d-ea1d56ce3b8a
(nil)
127.0.0.1:6379>
```

Success

We have addressed the core questions raised

- Handling of non-native objects such as Dataframes
- Large data sets between tasks
- Leveraging cloud storage
- Maintenance and cleanup

Limitations

Not tied to DAG life cycle management

- Data sharing across DAGs is difficult
- Maintenance DAGs for clean-up is a kludge

- Should be cleanly handled by Airflow when DAG is done

Future: Top level data object in Airflow

- Result of DAGs from one team is data
- Can be used by DAGs from other teams

- Key for cross-DAG dependencies
- Availability can be used to trigger follow-on DAGs

Integrated with DAG life cycle management and with Event driven DAGs

Airflow Improvement Proposal upcoming

Jobs at Astronomer

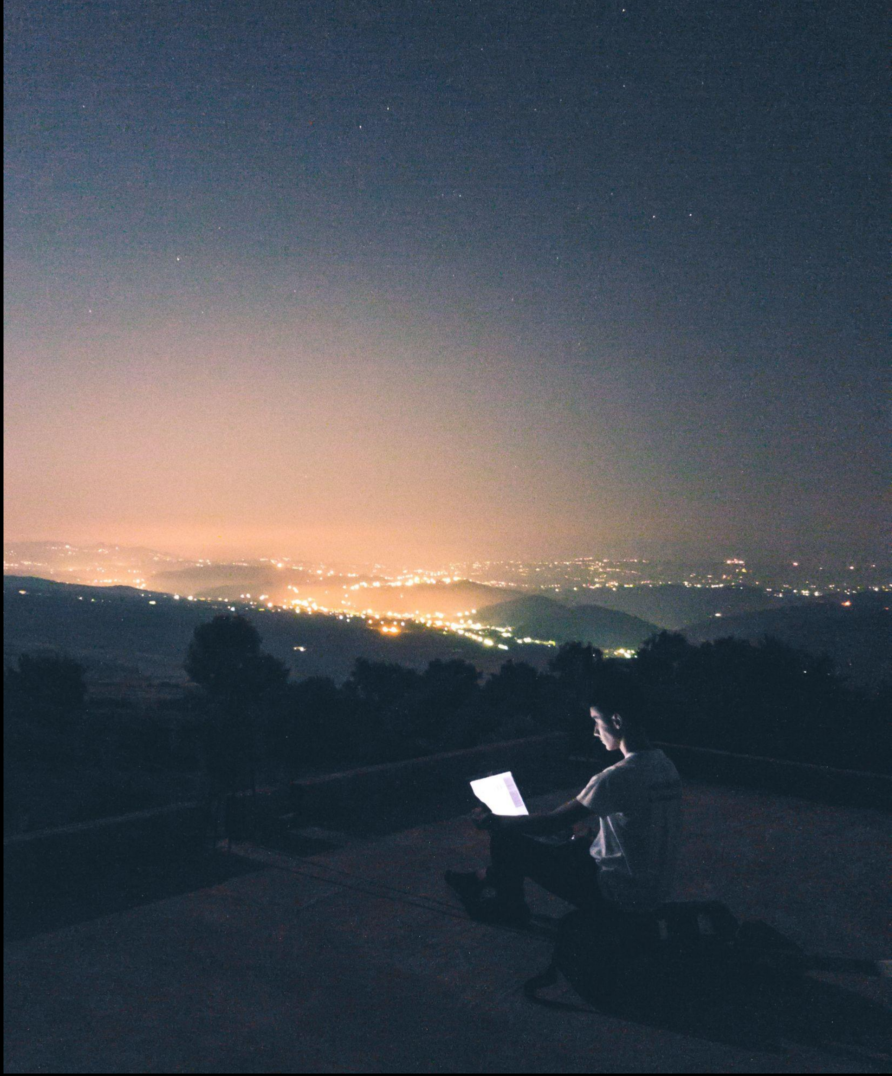
We are hiring Airflowers all over the world!

https://careers.astronomer.io/

https://linkedin.com/vikramkoka

Contact us: We would love to hear from you!

ASTRONOMER



Questions?