From Tech Specs to Business Impact:

How to Design a Truly End-to-End Airflow Project

Taylor Facen

Late Wednesday Afternoon…

Kevin 4:19 PM

We need access to Stripe's revenue

Kevin 4:19 PM

Sounds good? I'm heading off for some PTO. Looking forward to seeing this on Monday!

before they slip through the cracks. Revenue is top of mind for everyone on the team, so it's going to be top priority in our Monday standups.

Project Specs

We need access to Stripe's revenue recognition data so that the accounting team can do their month end close.

Throughout the month we'll review the data on an ad-hoc basis.

Hopefully, we're able to catch issues before they slip through the cracks.

Revenue is top of mind for everyone on the team, so it's going to be top priority in our Monday standups.

Ingestion

“We need access to Stripe’s revenue recognition data so that the accounting team can do their month end close.”

Scan me for code!

Step 1: Fetch the data

`SimpleHttpOperator`

Step 2: Store the raw data

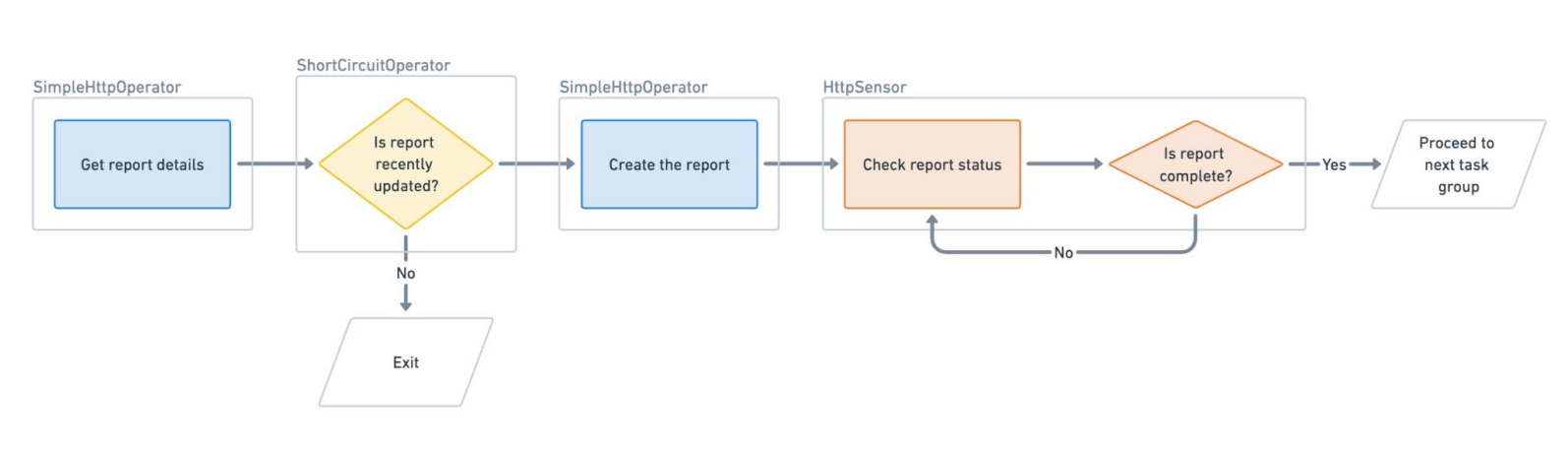
`GCSHook`

Step 3: Transfer the data

`GCSToBigQueryOperator`

Extract

1. Use `SimpleHttpOperator` to fetch basic details about the report
2. Use `ShortCircuitOperator` to only proceed if the report has been recently updated
3. Use `SimpleHttpOperator` to create a report run
4. Use `HttpSensor` to poll the report to wait for it to complete



Load

```python
from airflow.operators.python import PythonOperator

...

def download_and_upload_to_gcs(ti, **kwargs):
    ...

upload_csv_to_gcs = PythonOperator(
    task_id='upload_csv_to_gcs',
    python_callable=download_and_upload_to_gcs,
    provide_context=True
)
```

```python
from airflow.providers.google.cloud.transfers.gcs_to_bigquery import GCSToBigQueryOperator

...

load_to_bigquery = GCSToBigQueryOperator(
    task_id='load_to_bigquery',
    bucket=GCS_BUCKET_NAME,
    source_objects=["{{ ti.xcom_pull(task_ids='load.upload_csv_to_gcs',
key='storage_object_name') }}"],
    destination_project_dataset_table=f'{GCP_PROJECT}.airflow_stripe.revenue_recognition',
    write_disposition='WRITE_TRUNCATE',
    source_format='CSV',
    autodetect=True
)

upload_csv_to_gcs >> load_to_bigquery
```

Transformation and Testing

“Throughout the month we’ll review the data on an ad-hoc basis. ”


```
.
└── dags/
    ├── dbt/
    │   ├── models/
    │   │   └── staging/
    │   │       └── stripe/
    │   │           └── stg_stripe__revenue_recognition.sql
    │   └── dbt_project.yml
    └── rev-rec.py
```

Run Model

```sql
with source as (
    select * from {{ source('airflow_stripe', 'revenue_recognition')}}
),

renamed as (

    select
        coalesce(open_accounting_period, accounting_period) accounting_period,
        currency,
        debit,
        credit,
        amount
    from source

)

select * from renamed
```

```
INFO - 04:14:28  Running with dbt=1.8.6
INFO - 04:14:29  Registered adapter: bigquery=1.8.2
INFO - 04:14:29  Unable to do partial parsing because saved manifest not found. Starting full parse.
INFO - 04:14:30  Found 1 model, 5 data tests, 1 source, 741 macros
INFO - 04:14:30
INFO - 04:14:32  Concurrency: 1 threads (target='dev')
INFO - 04:14:32
INFO - 04:14:32  1 of 1 START sql view model stripe.revenue_recognition ......................... [RUN]
INFO - 04:14:33  1 of 1 OK created sql view model stripe.revenue_recognition .................... [CREATE VIEW (0 processed) in 1.30s]
INFO - 04:14:33
INFO - 04:14:33  Finished running 1 view model in 0 hours 0 minutes and 3.02 seconds (3.02s).
INFO - 04:14:33
INFO - 04:14:33  Completed successfully
```

Test Model

```yaml
version: 2

models:
  - name: stg_stripe__revenue_recognition
    description: This table details how our revenue is recognized each month.
    config:
      alias: revenue_recognition

    columns:
      - name: accounting_period
        data_tests:
          - not_null
      - name: debit
        data_tests:
          - not_null
      - name: credit
        data_tests:
          - not_null
      - name: amount
        data_tests:
          - not_null
```

```python
def warning_callback_func(context: Context):
  ...

  transform = DbtTaskGroup(
      group_id="transform",
      project_config=ProjectConfig(DBT_PROJECT_PATH),
      profile_config=cosmos_profile_config,
      execution_config = ExecutionConfig(
          dbt_executable_path=DBT_EXECUTABLE_PATH,
      ),
      default_args={"retries": 0},
      operator_args={
          "install_deps": True
      },
      render_config=RenderConfig(
          select=["stg_stripe__revenue_recognition"]
      ),
      on_warning_callback=warning_callback_func
  )
```

Detection

“Hopefully, we’re able to catch issues before they slip through the cracks.”

```yaml
version: 2

models:
    ...

    tests:
      - dbt_expectations.expect_row_values_to_have_data_for_every_n_datepart:
          date_col: accounting_period
          date_part: month
```

airflow APP 3:08 PM

🟡 Airflow-DBT task with WARN.

Task: transform.stg_stripe__revenue_recognition.test

Dag: revenue_recognition

Execution Time: 2024-09-08 22:08:18.785621+00:00

Log Url: http://localhost:8080/dags/revenue_recognition/grid?dag_run_id=manual__2024-09-08T22%3A08%3A18.785621%2B00%3A00&task_id=transform.stg_stripe__revenue_recognition.test&base_date=2024-09-08T22%3A08%3A18%2B0000&tab=logs

Test: dbt_expectations_expect_row_values_to_have_data_for_every_n_datepart_stg_stripe__revenue_recognition_accounting_period__month

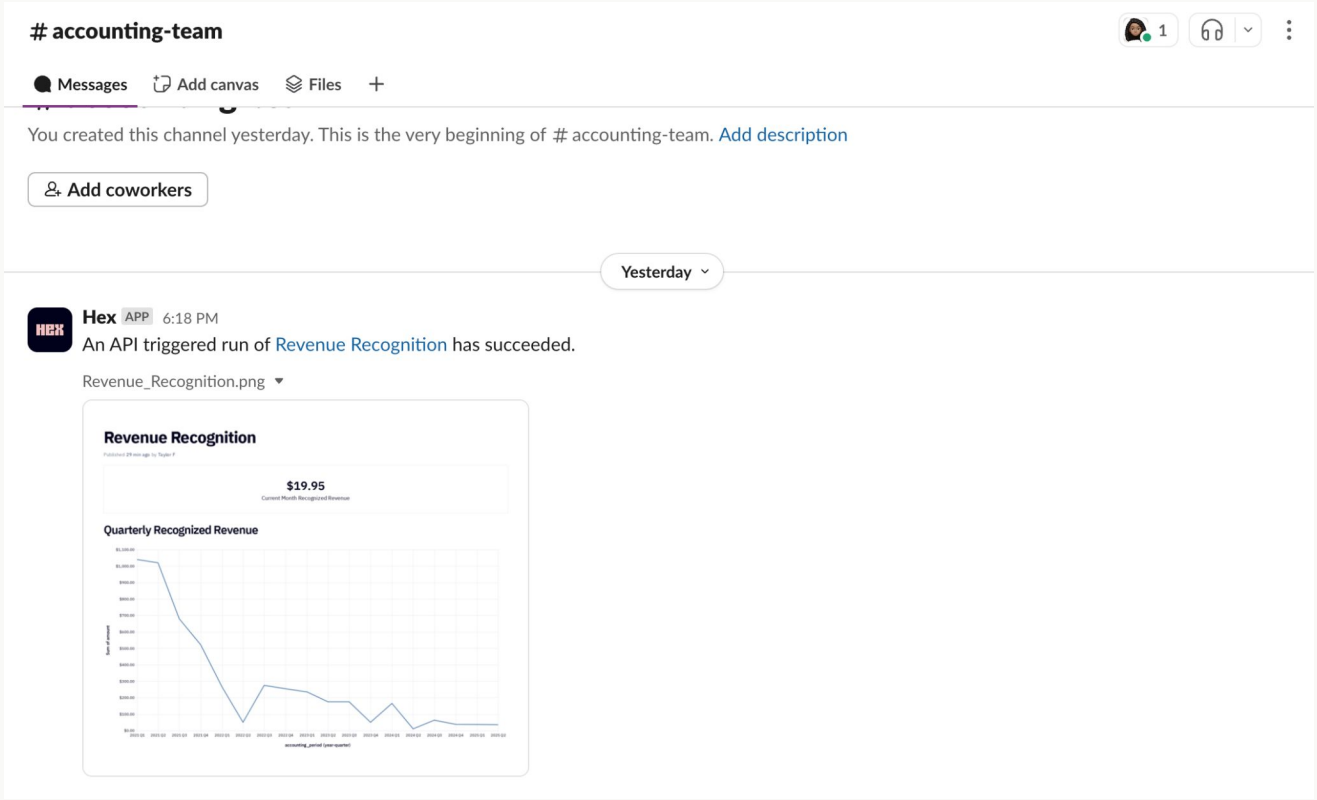
Result: Got 9 results, configured to warn if != 0

Report

“Revenue is top of mind for everyone on the team, so it’s going to be top priority in our Monday standups.”

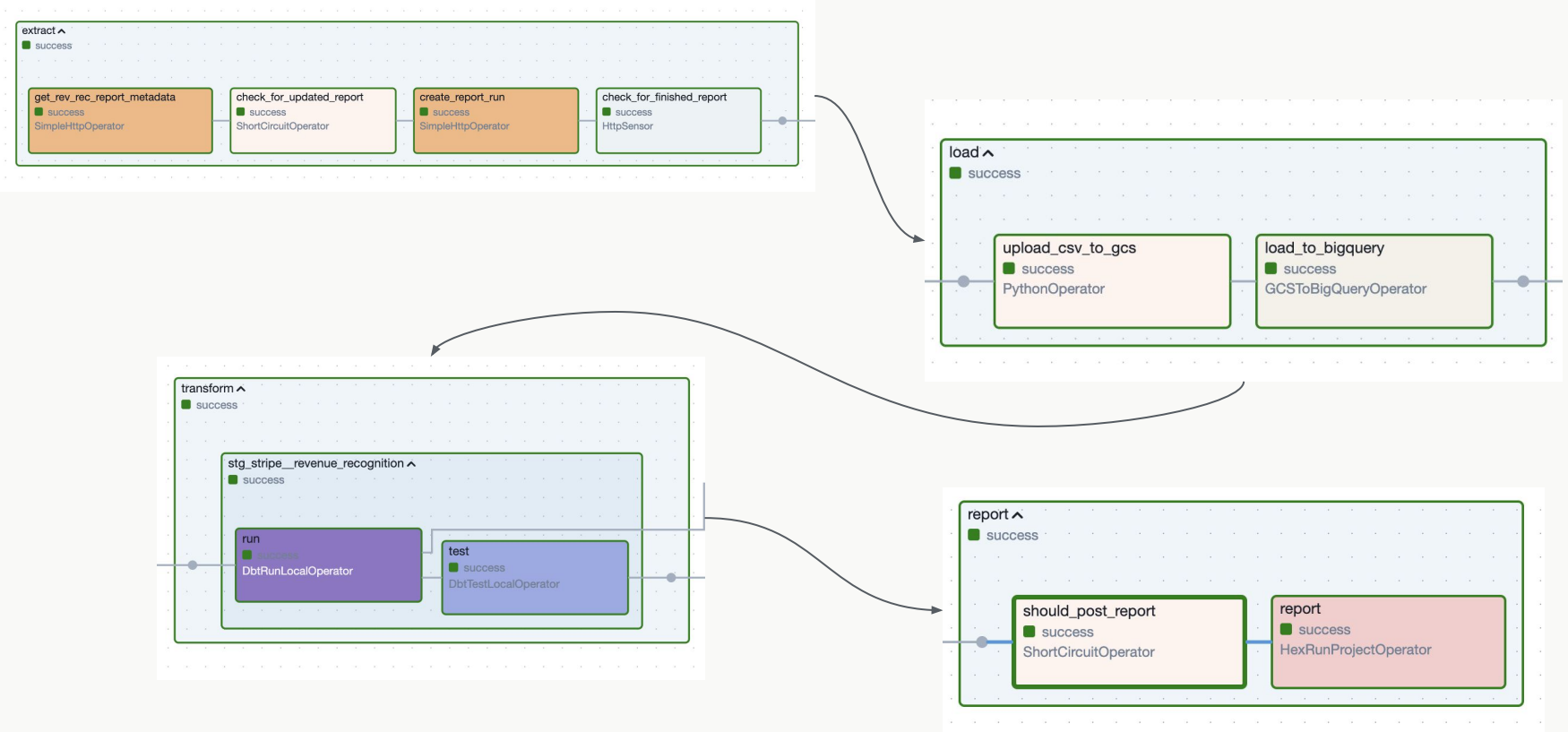
```python
from airflow.models import DagRun
from datetimetime import timedelta
...

def monday_check(**kwargs):
    dag_logical_date = kwargs['dag_run'].logical_date
    is_monday = dag_logical_date.weekday() == 0
    if not is_monday:
        return False

    dag_runs = DagRun.find(
        dag_id=DAG_ID,
        execution_start_date=(kwargs['dag_run'].logical_date - timedelta(days=1)),
        execution_end_date=kwargs['dag_run'].logical_date,
        state="success"
    )

    if len(dag_runs) == 0:
        return True
    dag_runs.sort(key=lambda x: x.execution_date, reverse=True)
    last_dag_run = dag_runs[0]
    return last_dag_run.execution_date.date() != dag_logical_date.date()
```

Hex Report



Check out the report!

Project DAG


extract
success
get_rev_rec_report_metadata
success
SimpleHttpOperator
check_for_updated_report
success
ShortCircuitOperator
create_report_run
success
SimpleHttpOperator
check_for_finished_report
success
HttpSensor
load
success
upload_csv_to_gcs
success
PythonOperator
load_to_bigquery
success
GCSToBigQueryOperator
transform
success
stg_stripe__revenue_recognition
success
run
DbtRunLocalOperator
test
success
DbtTestLocalOperator
report
success
should_post_report
success
ShortCircuitOperator
report
success
HexRunProjectOperator

Three Key Traits of a Data Product

- **Accurate** - Can we trust the data?

- **Proactive** - Can we catch issues early?

- **Actionable** - Does the product drive data-driven decision-making? / Does the end-product solve the customer's problem?

Questions?

@ItsTayFay

taylorfacen.com