Airflow

- Installing Airflow Locally
- TriggerDAGRunOperator
- Handling non-string DAG parameters

Installing Airflow Locally

Install Script

```
# Airflow needs a home. `~/airflow` is the default, but you can put it
# somewhere else if you prefer (optional)
export AIRFLOW_HOME=~/airflow

# Install Airflow using the constraints file
AIRFLOW_VERSION=2.5.3
PYTHON_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)"
# For example: 3.7
CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-
${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt"
# For example: https://raw.githubusercontent.com/apache/airflow/constraints-2.5.3/constraints-
3.7.txt
pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"

# The Standalone command will initialise the database, make a user,
# and start all components for you.
airflow db init

# Visit localhost:8080 in the browser and use the admin account details
# shown on the terminal to login.
# Enable the example_bash_operator DAG in the home page
```

Config File

There is a file named `airflow.cfg` which contains configuration for your airflow instance including the full path to the DAGs folder and also SQLalchemy connection credentials.

Google Cloud Compatibility

Install airflow with `google` optional module: `pip install apache-airflow[google]`.

Credenials can be side-loaded in via an environment variable

```
export AIRFLOW_CONN_GOOGLE_CLOUD_DEFAULT='{"conn_type": "google-cloud-platform", "key_path": "/secrets/key.json", "scope": "https://www.googleapis.com/auth/cloud-platform", "project": "airflow", "num_retries": 5}'
```

Google OIDC auth

It's possible to authenticate airflow against your google org using SAML/OpenID flow:

https://airflow.apache.org/docs/apache-airflow-providers-google/stable/api-auth-backend/google-openid.html

TriggerDAGRunOperator

- You can use `wait_for_completion=True` and it will detect whether the dag passed or failed.
- If you have a cleanup function make sure that it marks the upstream task as failed as appropriate.

```
for task_instance in kwargs['dag_run'].get_task_instances():
        if task_instance.current_state() not in [State.SUCCESS, State.SKIPPED] and \
                task_instance.task_id != kwargs['task_instance'].task_id:
            raise Exception("Task {} failed. Failing this DAG
run".format(task_instance.task_id))
```

Handling non-string DAG parameters

You may need to pass a dag parameter to an operator using templates - however, it may be important that this is the right data type (e.g. bool). By default all template values are stringified.

You can override this by passing `render_template_as_native_obj=True` to the DAG constructor which fills the template and then casts the value to the expected type.

For example

```
with DAG(
    dag_id="example",
    catchup=False,
    start_date=datetime(2023, 4, 26),
    render_template_as_native_obj=True,
    params={
        "test_run": Param(False, title="Test run only", description="If true, don't do the
transaction, just spoof it", type='boolean')
    }
) as dag:



    task_id = f"trigger_build_table_{table}"

    ts = datetime.now().replace(tzinfo=tz)
    dag_execution_ts = TriggerDagRunOperator(
            trigger_dag_id="exampledag",
            wait_for_completion=True,
            poke_interval=10,
            # reset_dag_run=True,tk
            execution_date=ts.isoformat(),
            trigger_run_id=f"example_dag_{ts.isoformat()}",
            conf={
                "table_name": table,
                "date_suffix": date_suffix,
                "test_run": "{{params.test_run}}"
```

```
        },
    )
```