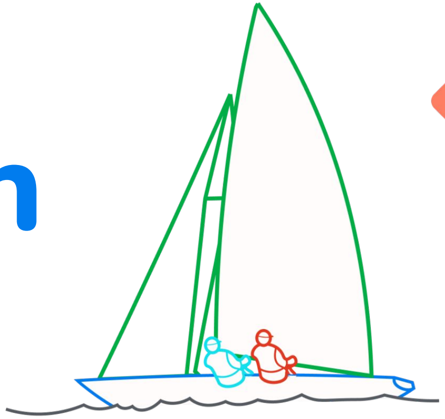
Migrate Apache Oozie Workflows to Airflow and Run with Amazon EMR

Dipankar Ghosal



Airflow Summit

Let's flow together

September 19-21, 2023,
Toronto, Canada

About me

- Principal Architect at Amazon Web Services
- Advisory and Transformation
- Excited for the football season
- Still like to code ☺

Linked In - www.linkedin.com/in/dipankarghosal

Motivation…need

Computing workloads are moving to the cloud to achieve greater business agility, access to modern technology, and reduce operational costs.

Whether it's Data First approach or Workload First migration strategy

- Decouple Storage and Compute
- Decouple Workload management

Apache Oozie

- Apache Oozie is a workflow management system to manage Hadoop jobs.
- It is deeply integrated with the rest of Hadoop stack supporting a number of Hadoop jobs out-of-the-box.
- Workflow is expressed as XML and consists of two types of nodes: control and action.
- Scalable, reliable and extensible system.

Apache Airflow

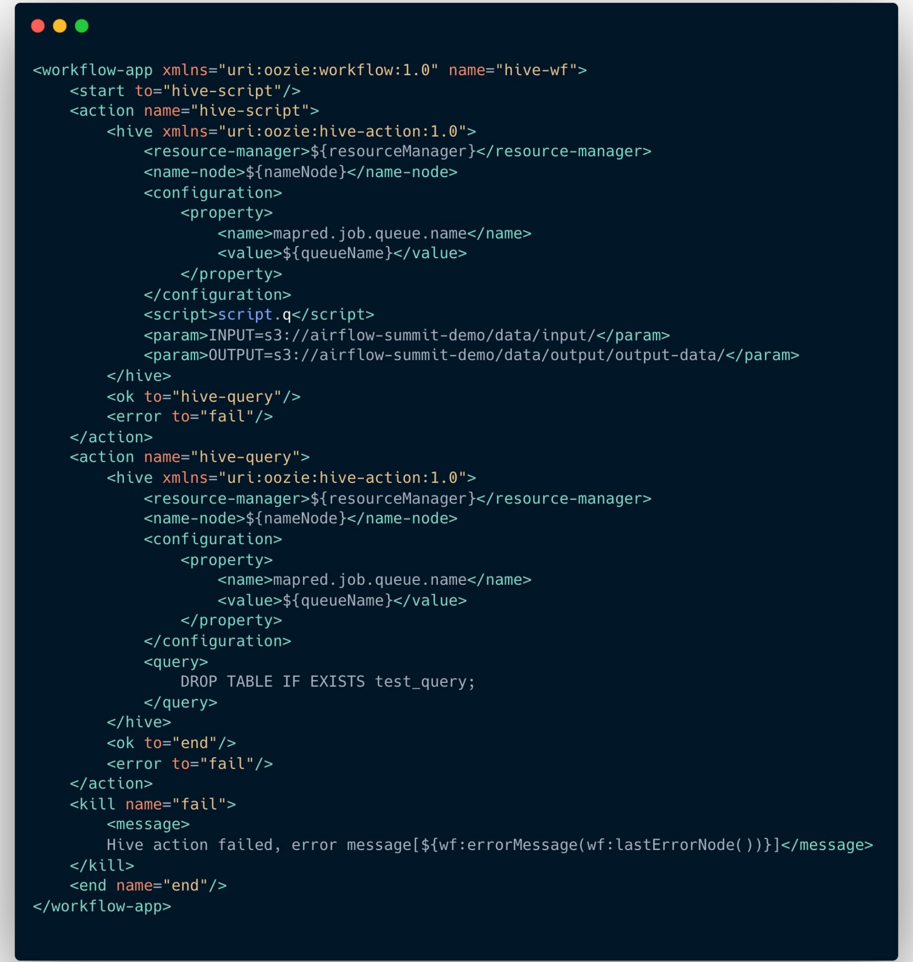
- Apache Airflow is a platform to programmatically author, schedule, and monitor workflows.
- Workflows are authored as directed acyclic graphs (DAGs), and can be configured as code - using Python 3.x.
- The rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed.

Workflow vs DAG

Oozie workflow

This is a very simple Oozie workflow that performs a hive action. Pay attention to the following XML elements:

1. start
2. action
3. kill
4. end



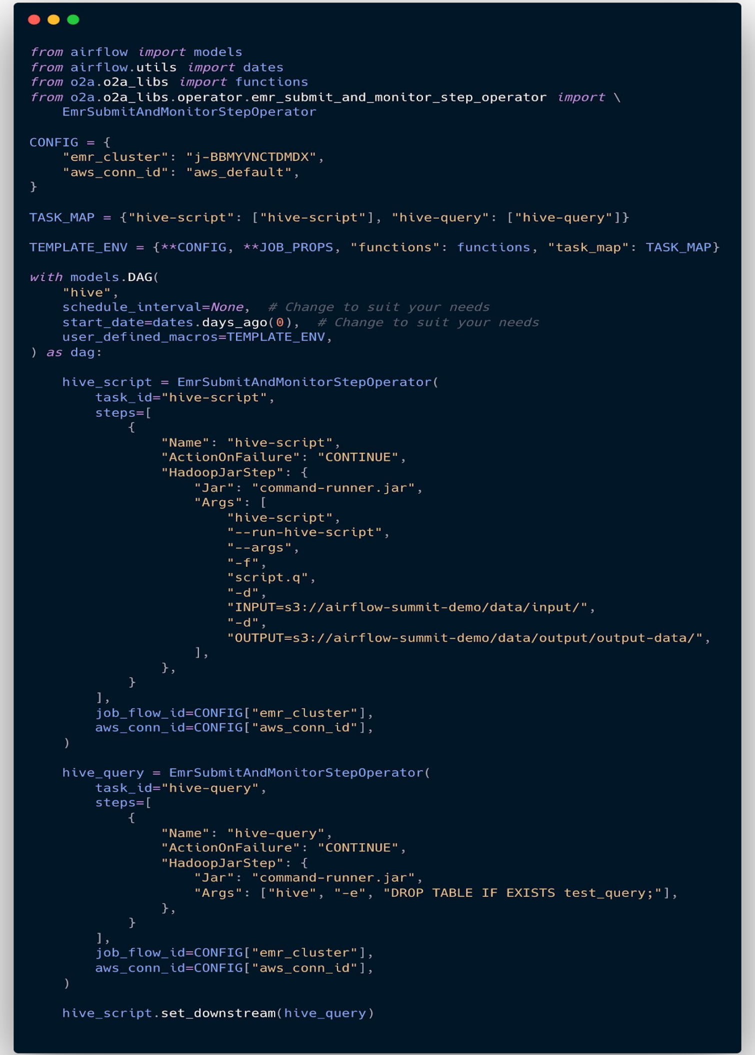
```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="hive-wf">
    <start to="hive-script"/>
    <action name="hive-script">
        <hive xmlns="uri:oozie:hive-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
            <script>script.q</script>
            <param>INPUT=s3://airflow-summit-demo/data/input/</param>
            <param>OUTPUT=s3://airflow-summit-demo/data/output/output-data/</param>
        </hive>
        <ok to="hive-query"/>
        <error to="fail"/>
    </action>
    <action name="hive-query">
        <hive xmlns="uri:oozie:hive-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
            <query>
                DROP TABLE IF EXISTS test_query;
            </query>
        </hive>
        <ok to="end"/>
        <error to="fail"/>
    </action>
    <kill name="fail">
        <message>
        Hive action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>
    </kill>
    <end name="end"/>
</workflow-app>
```

Workflow vs DAG

DAG code

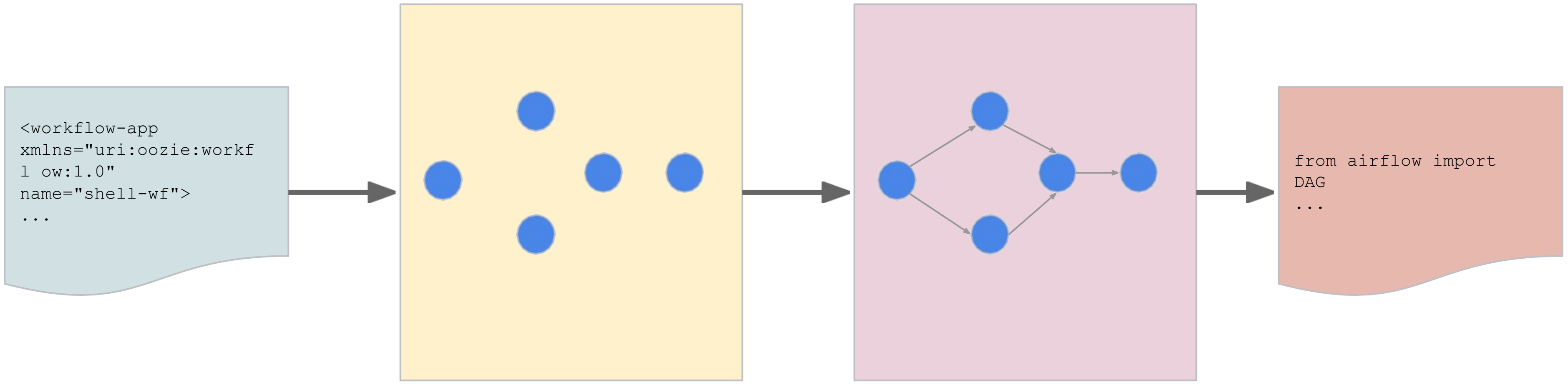
This is a Airflow DAG equivalent of Oozie Workflow. Important elements to note are

1. Task Map
2. Operator
3. Relationship



```python
from airflow import models
from airflow.utils import dates
from o2a.o2a_libs import functions
from o2a.o2a_libs.operator.emr_submit_and_monitor_step_operator import \
    EmrSubmitAndMonitorStepOperator

CONFIG = {
    "emr_cluster": "j-BBMYVNCTDMDX",
    "aws_conn_id": "aws_default",
}

TASK_MAP = {"hive-script": ["hive-script"], "hive-query": ["hive-query"]}

TEMPLATE_ENV = {**CONFIG, **JOB_PROPS, "functions": functions, "task_map": TASK_MAP}

with models.DAG(
    "hive",
    schedule_interval=None,  # Change to suit your needs
    start_date=dates.days_ago(0),  # Change to suit your needs
    user_defined_macros=TEMPLATE_ENV,
) as dag:

    hive_script = EmrSubmitAndMonitorStepOperator(
        task_id="hive-script",
        steps=[
            {
                "Name": "hive-script",
                "ActionOnFailure": "CONTINUE",
                "HadoopJarStep": {
                    "Jar": "command-runner.jar",
                    "Args": [
                        "hive-script",
                        "--run-hive-script",
                        "--args",
                        "-f",
                        "script.q",
                        "-d",
                        "INPUT=s3://airflow-summit-demo/data/input/",
                        "-d",
                        "OUTPUT=s3://airflow-summit-demo/data/output/output-data/",
                    ],
                },
            }
        ],
        job_flow_id=CONFIG["emr_cluster"],
        aws_conn_id=CONFIG["aws_conn_id"],
    )

    hive_query = EmrSubmitAndMonitorStepOperator(
        task_id="hive-query",
        steps=[
            {
                "Name": "hive-query",
                "ActionOnFailure": "CONTINUE",
                "HadoopJarStep": {
                    "Jar": "command-runner.jar",
                    "Args": ["hive", "-e", "DROP TABLE IF EXISTS test_query;"],
                },
            }
        ],
        job_flow_id=CONFIG["emr_cluster"],
        aws_conn_id=CONFIG["aws_conn_id"],
    )

    hive_script.set_downstream(hive_query)
```

o2a- Converter Design


<workflow-app xmlns="uri:oozie:workfl ow:1.0" name="shell-wf"> ...
from airflow import DAG ...

workflow XML ⟶ [nodes] where node: name, attributes, child elements, etc.

[nodes] -> workflow object:

- dependencies
- relationship
- airflow-nodes

workflow object ⟶ dag.py

Source - https://conferences.oreilly.com/strata/strata-eu-2019/public/schedule/detail/74271.html

We wanted to make it work with Amazon EMR

Run big data applications and petabyte-scale data analytics faster, and at less than half the cost of on-premises solutions.

To submit work, you can add steps, or you can interactively submit Hadoop jobs to the primary node.

Amazon EMR

https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-work-with-steps.html

Operator Jinja Templates

Dataproc

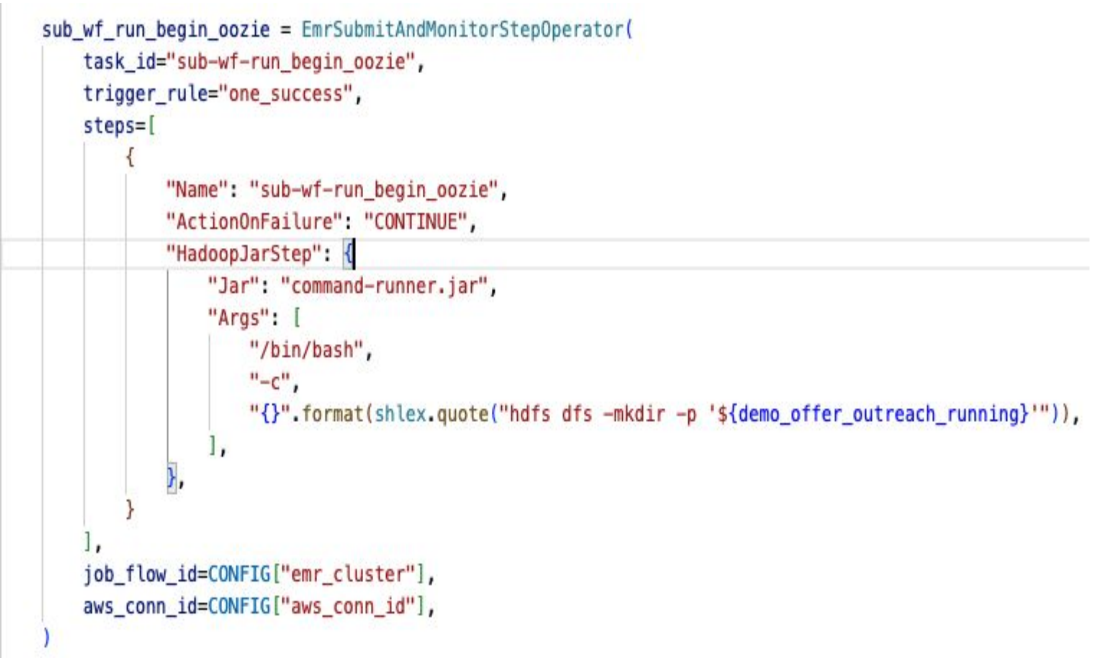
```
{{ task_id | to_var }} =
   **dataproc_operator.DataProcPigOperator**(
   task_id={{ task_id | to_python }},
   trigger_rule={{ trigger_rule | to_python }},
   query_uri='%s/%s' % (CONFIG['gcp_uri_prefix'], {{
   script_file_name | to_python }}),
   variables={{ params_dict | to_python }},
   dataproc_pig_properties={{
   props_macro.props(action_node_properties=action_n
   ode_properties, xml_escaped=True) }},
   cluster_name=CONFIG['dataproc_cluster'],
   gcp_conn_id=CONFIG['gcp_conn_id'],
   region=CONFIG['gcp_region'],
   dataproc_job_id={{ task_id | to_python }},
   params={{
   props_macro.props(action_node_properties=action_n
   ode_properties) }},
)
```

EMR

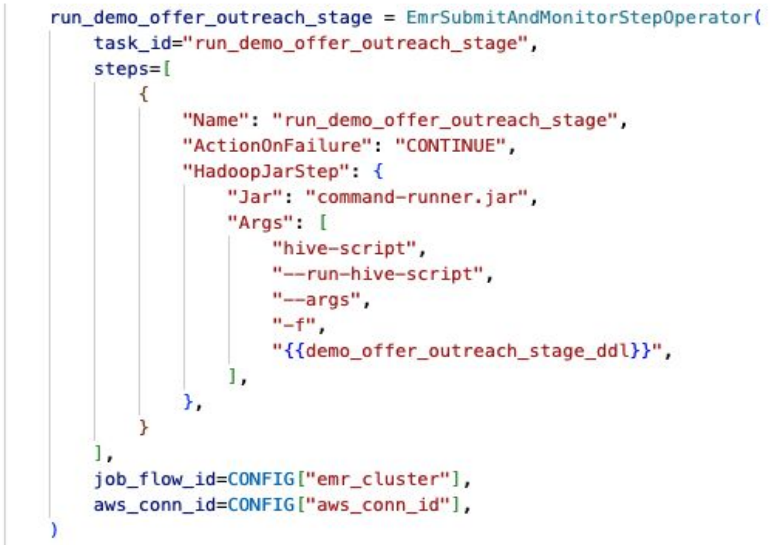
```
{{ task_id | to_var }} =
   **EmrSubmitAndMonitorStepOperator**(
   task_id={{ task_id | to_python }},
   steps=[{'Name': {{ task_id | to_python
   }},'ActionOnFailure': 'CONTINUE','HadoopJarStep':
   {'Jar': 'command-runner.jar','Args': {{ params_dict |
   to_python }},},}],
   job_flow_id=CONFIG['emr_cluster'],
   aws_conn_id=CONFIG['aws_conn_id'],
)
```

Pig action template

Single Operator for EMR



```
sub_wf_run_begin_oozie = EmrSubmitAndMonitorStepOperator(
    task_id="sub-wf-run_begin_oozie",
    trigger_rule="one_success",
    steps=[
        {
            "Name": "sub-wf-run_begin_oozie",
            "ActionOnFailure": "CONTINUE",
            "HadoopJarStep": {
                "Jar": "command-runner.jar",
                "Args": [
                    "/bin/bash",
                    "-c",
                    "{}".format(shlex.quote("hdfs dfs -mkdir -p '${demo_offer_outreach_running}'")),
                ],
            },
        }
    ],
    job_flow_id=CONFIG["emr_cluster"],
    aws_conn_id=CONFIG["aws_conn_id"],
)
```



```
run_demo_offer_outreach_stage = EmrSubmitAndMonitorStepOperator(
    task_id="run_demo_offer_outreach_stage",
    steps=[
        {
            "Name": "run_demo_offer_outreach_stage",
            "ActionOnFailure": "CONTINUE",
            "HadoopJarStep": {
                "Jar": "command-runner.jar",
                "Args": [
                    "hive-script",
                    "--run-hive-script",
                    "--args",
                    "-f",
                    "{{demo_offer_outreach_stage_ddl}}",
                ],
            },
        }
    ],
    job_flow_id=CONFIG["emr_cluster"],
    aws_conn_id=CONFIG["aws_conn_id"],
)
```

Demo

Demo Recap

- Oozie to Airflow converter repo: https://github.com/dgghosalaws/oozie-to-airflow-emr
- Successfully converted a representative complex Oozie bundle workflow to Airflow DAG with Amazon EMR compatibility
    - Supports Coordinator and bundle parsing
    - Single Airflow EMR Operator
- No post-conversion modification and runs well out of the box

Questions?

Other Articles

Building complex workflows with Amazon MWAA, AWS Step Functions, AWS Glue, and Amazon EMR