Pipelines on Pipelines: Creating Agile CI/CD Workflows for Airflow DAGs

By Victor Shafran
CPO at databand.ai

About Me

- Founder and CPO at Databand.ai
- Background in Machine Learning
- Working with data from 2008

- In my spare time:
  - Proud father of 2 daughters.
  - Run, Hike

My Nightmares 😱

- Junior Engineer push new code -> Spark cluster stalled.
- Senior Engineer push new code -> Overwrite production partition. Took 24 hours to recreate.
- New Spark Operator introduced new version of JAR, the rest of DAGs has failed. Ruined a weekend while discovering and fixing
- Partner change data format. Discovered after 3 month

I had this kind of issues daily ....

- But, I do not want to spent all my money on sleeping pills 😄
- I also do not want my weekend ruined 🏕️
- -> I want to create an environment where every change can be tested end to end

CI/CD pipeline for my DAGs

What is CI/CD

- Integration
- Stress
- Regressions

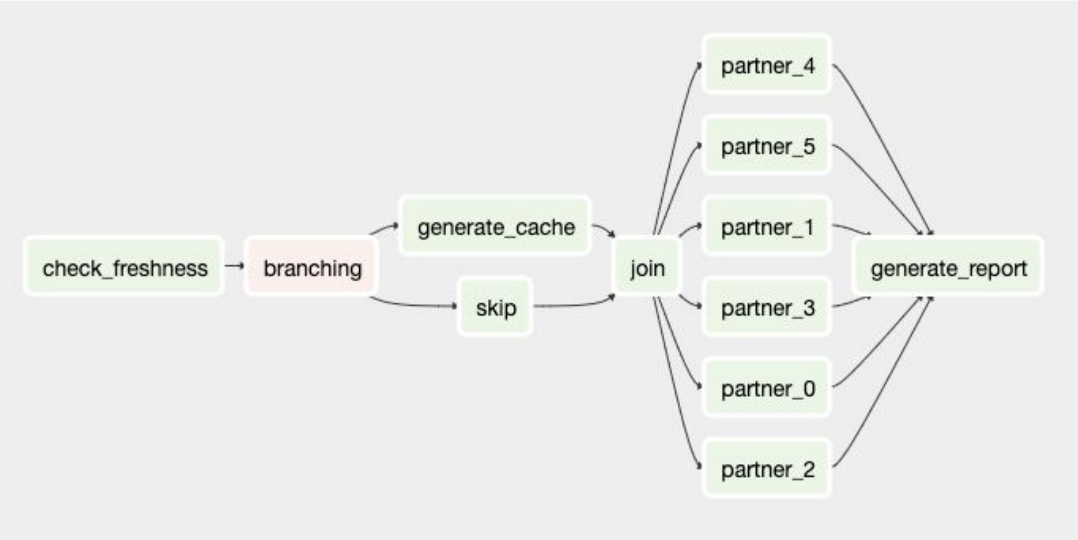
CI/CD Pipeline == End to End Automation

CI/CD for Data DAGs. Spark Operator

- Spark is a de-facto standard in Data Processing
- Spark - A good example of Data intensive operator (applicable for ..PythonOperator, …)
- **Spark is the most used tool by Airflow Community:**
    - Spark Operator,
    - EmrStep Operator,
    - Dataproc Operator,
    - Databricks Operator

CI/CD

- Business Logic
- DAG code - is it wiring or business logic?
- Testing DAG structure...



We want CI/CD → running END TO END!

SparkSubmitOperator

- Spark Cluster selector (conn_id)
- Spark Job Configuration
  - Python/Java Dependencies
  - Resources
- Spark CLI

```python
task1 = SparkSubmitOperator(
    task_id="generate_report",
    conn_id="spark_default",
    application="script.py",
    application_args=["input_file.csv", "output.csv"],
    jars="/jars/lib.jar",
    py_files="/libs/library.py",
    driver_memory="3g"
)
```

Execution Isolation: Cluster Environments

- Production - final code
- Staging
  - Multiple Version
  - Custom Resources
- → Parametrize JAR/PY Locations
- → For example, use git commit

```
task1 = SparkSubmitOperator(
    task_id="generate_report",
    conn_id="spark_default",
    application="script.py",
    application_args=["input_file.csv", "output.csv"],
    packages="com.my_company:my_jar_2_ci_a6416d2.11:3.2.0",
    repositories="http://myrepo.org",
    py_files="/libs/library_ci_a6416d2.py",
    driver_memory="3g"
)
```

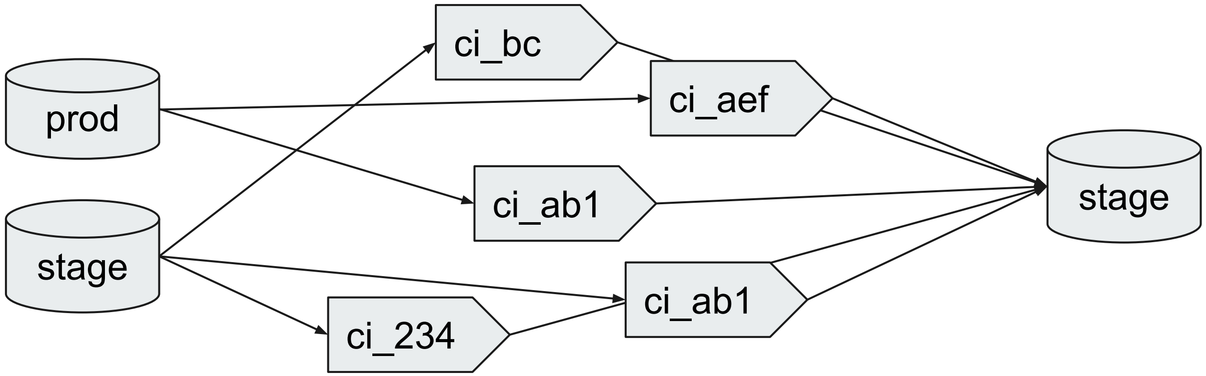
Rendered Operator Example

What about Data?

No batteries included!

Requirements
for Data intensive DAG CI/CD

- Data inputs/outputs isolation for every CI/CD cycle
  - You want every feature in separate area,
  - Sometime you don't want to start every time from scratch
- No unexpected side effects (people connects jobs to different systems/DB/Files)
- Being able to inject different data into your pipeline (small/big/production/errors)



Simple: Jinja + xCom

```
prefix = "{{var.value.output_root}}/{{ dag.dag_id }}/{{ task.task_id }}/{{ts}}"

def join_task_callable(**kwargs):
    output = kwargs["ti"].render_template(prefix)
    kwargs["ti"].xcom_push(key="partners", value=[output+"/a.csv", output+"/b.csv", output+"/f.csv"])

join = PythonOperator(
    task_id="join",
    python_callable=join_task_callable,
)

task = SparkSubmitOperator(
    task_id="generate_report",
    conn_id="spark_default",
    application="script.py",
    application_args=["{{ti.xcom_pull(task_ids='join',key='partners')}}", prefix+"/report.csv"],
)
```

Library of Jinja Macros

```
task = SparkSubmitOperator(
    task_id="generate_report",
    conn_id="spark_default",
    application="script.py",
    application_args=["{{my_company_data_repo.get_artifact('partners')",
                      "{{my_company_data_repo.get_artifact('report')"],
)
```

- Create your own JINJA plugin
- Register it to Airflow macros JINJA framework

Custom Operator

```
join = MyPythonOperator(
    task_id="join",
    python_callable=join_task_callable,
)

task = MySparkSubmitOperator(
    task_id="generate_report",
    conn_id="spark_staging",
    application="script.py",
    application_args=[join.output.partners, MySparkSubmitOperator.output("report")],
)
```

Benefits:

- Check inputs before running
- Serialize outputs automatically
- Automatic wiring of Task

-> Full control over inputs and outputs

Now you can!

- Run iterations on CI/CD
- Validate DAGS with different DATA
- Inject data with errors! (Chaos Monkey for Data!)
- Reuse Same clusters for different versions
- Enable End Users to run Regressions on their own!
- Multiple REGRESSIONS at all stages(dev,int,stg,prd) -> Successful CI/CD process!

References and Next Steps

- AIP-31: The initial solution
- AIP-<> More to come
- dbnd-airflow - extension that does data management on it’s own

Recap

What’s real CI/CD for data intensive DAGs

Effective CI/CD for SparkOperator

Data Management Layer role in CI/CD process

Topics for the next lecture....

Automation of CI/CD:

- Deployment DAG is a separate lecture
- Dags migration from research to production and vice versa.

Shameless Promotion

- July 14, Achieving Airflow observability with Databand by Josh Benamram
- July 17, Data Observability by Evgeniy Shulman

Thanks you!