


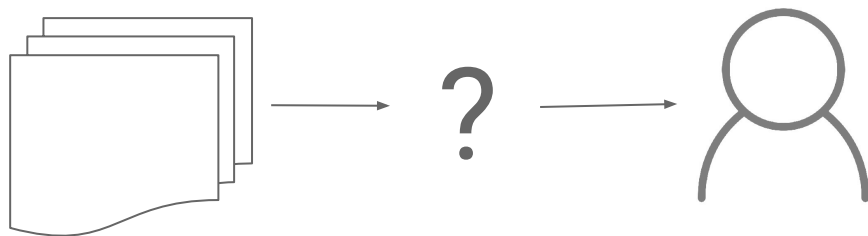
Apache Airflow and Amazon Redshift

Applications to Machine Learning and Analytics

Stephanie Kirmer
www.stephaniekirmer.com
 @data_stephanie

Learning Goals

- ❖ What are Airflow and Redshift?
- ❖ How can they be used to build and manage a data warehouse?
- ❖ How can machine learning workflows employ them?
- ❖ What are the risks and pitfalls to avoid?



Toolkit

Data storage solution:



- Installation instructions:
<https://docs.aws.amazon.com/redshift/latest/gsg/getting-started.html>

Job scheduler:



Installation instructions:

<https://airflow.apache.org/docs/stable/start.html>

Optional, nice to have: a helpful Ops colleague

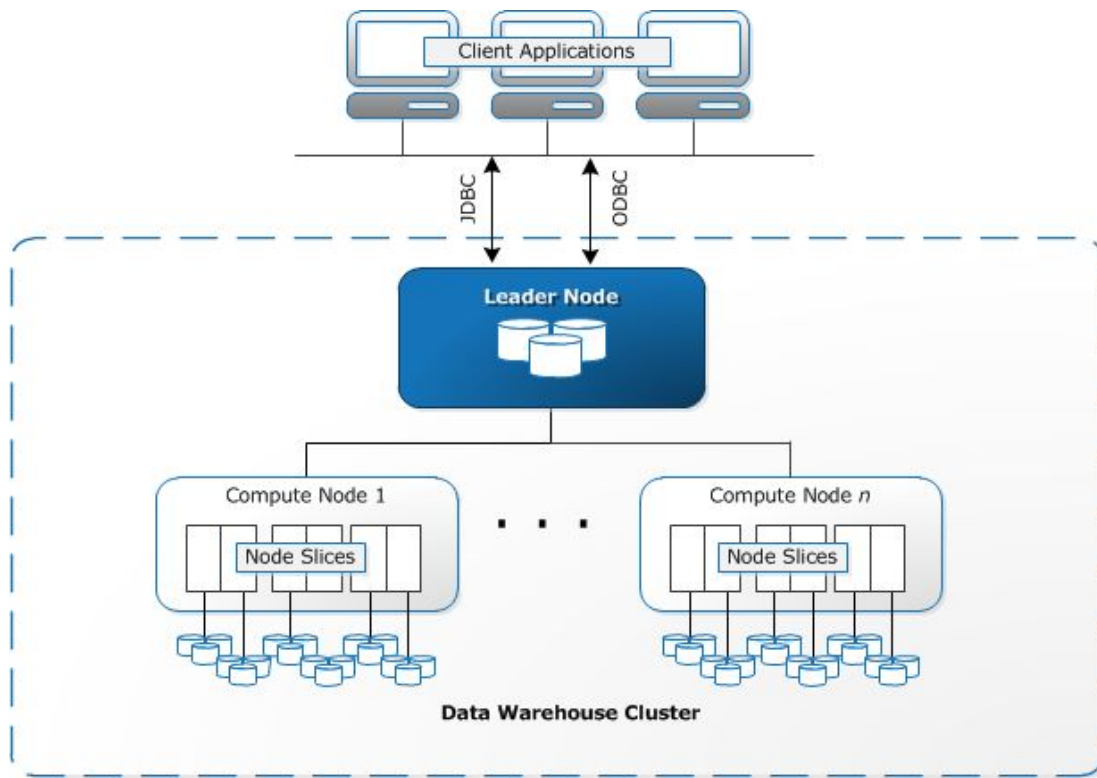
Amazon Redshift



Redshift is a data storage product from Amazon.

- ❖ Common ancestor with PostgreSQL - branched off from PostgreSQL 8.0.2
- ❖ Suited for columnar/tabular data at large scale
- ❖ Gives lots of opportunities for parallelization because of distributed nature, but query clashes still can happen
- ❖ AWS tools allow you to programmatically create and destroy instances/clusters

Architecture Overview



Leader receives queries and develops plans for execution.

Compute Nodes receive instruction from Leader and hand off to Slices to run.

Data is split into **Slices**, divided according to your chosen dist keys.

https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html

Distributed data storage allows for rapid retrieval if organized well

Take these steps to make Redshift work for you.

Understand the underlying framework

- ❖ Relational and distributed
- ❖ Distribution key helps Redshift choose what data to store together in nodes
- ❖ Data compression- set encodings that make sense for your data (ANALYZE COMPRESSION)

Plan ahead and be strategic

- ❖ Difficult to change architecture midstream
- ❖ Assign sort key: what do you usually filter/sort by? This should be it.
- ❖ PLUS: if you have a relational schema with key fields for joining, these might be sort AND dist keys
- ❖ Redshift wants to help you- use EXPLAIN to test how beefy your query is going to be

Tell Redshift what to expect.

```
CREATE TABLE IF NOT EXISTS public.customer(  
    customer_key integer not null IDENTITY(1,1) encode zstd sortkey,  
    customer_uuid char(36) encode zstd,  
    customer_home_base char(20) encode zstd,  
    customer_home_base_key integer encode zstd,  
    primary key(customer_key));
```

When creating long term persistent tables, set your schema in advance.

Protect yourself when inserting data.

```
COPY public.customer(customer_uuid, customer_home_base, customer_home_base_key)
  from 's3://s3_bucket/s3_key_customer.csv'
  iam_role '{self.iam}'
  region 'us-east-1'
  removequotes
  NULL 'None'
  emptyasnull
  blanksasnull
  delimiter ','
;
```

When INSERT-ing or COPY-ing, explicitly state the column names – data coming from elsewhere may be ordered wrong.

A Moment on Schemas

Redshift will allow you to build unkeyed, unlinked tables willy-nilly - *I advise against this!*

Employing a systematic, planned data architecture schema will save you pain and stress in the future.

Redshift does not enforce foreign or primary key restrictions the way other SQL types may - you have to maintain your own discipline.

Using table keys can make choosing highly performant sort keys and dist keys easier too!

Mistakes I've Made With Redshift

Not optimizing queries to match keys

- ❖ No query should run for 66 hours.

Making a table for every occasion

- ❖ Think about how the table fits with your architecture. You have an architecture, right?
- ❖ Don't gunk up your data warehouse with tables that are not documented or reusable. Clean house sometimes!

Making tables without keys or clear relationships

- ❖ Optimize, optimize, optimize - you will have more data, and more users, eventually

Making tables without compression

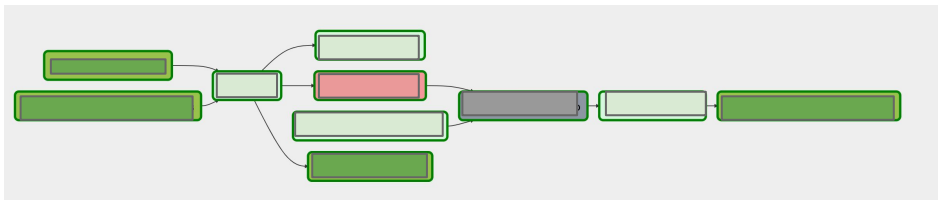
- ❖ This is one of the big advantages of Redshift, so why not use it?

Apache Airflow

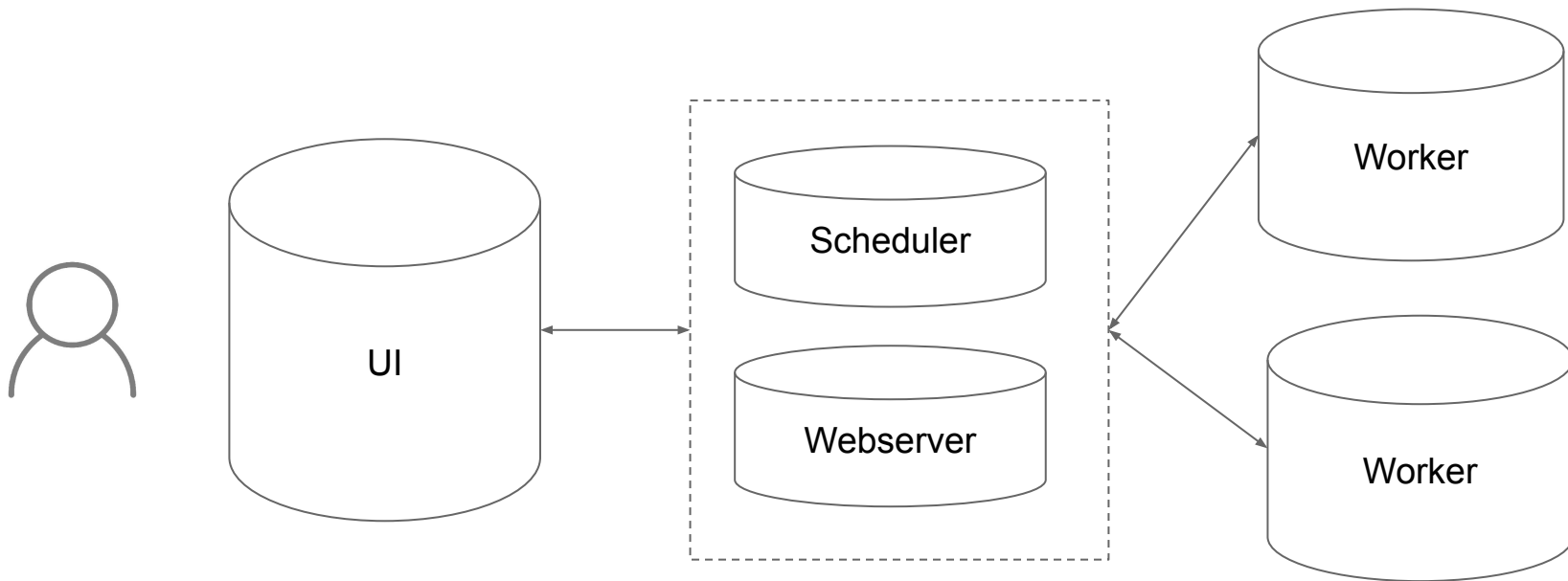


Airflow is an open source task management tool.

- ❖ Tool for programmatically building, scheduling, and maintaining jobs
- ❖ Python based, uses Directed Acyclic Graphs (DAGs)
- ❖ Allows testing, versioning, and monitoring
- ❖ Open source and free - you can contribute!



Airflow is structured in clusters.



You might need a friendly Ops person to help with installation. Note: I have omitted some elements of metadata management and logging here for simplicity's sake.

Meet the Directed Acyclic Graph.

Airflow wants to know...	Write in your DAG definition file...
<i>What tools/resources do you need?</i>	Calls of python libraries, airflow modules and operators, etc
<i>What is this DAG all about?</i>	Call DAG function with arguments of DAG characteristics- name, run schedule, etc
<i>What information will all your tasks want to know?</i>	Dictionary of default arguments- retry rules, failure notifications, etc
<i>What is each task you want to perform?</i>	Call each operator, and define each task with the arguments
<i>What order should things happen?</i>	Define dependencies (task >> next_task)

For further reading: <https://airflow.apache.org/docs/stable/tutorial.html#it-s-a-dag-definition-file>

Airflow standards

```
from airflow import DAG
```

```
from airflow.contrib.operators.aws_athena_operator import AWSAthenaOperator
```

Other python
libraries

```
from datetime import datetime, timedelta
```

Custom written
operators

```
from operators.trigger_error_alert import trigger_pagerduty_alert
```

```
from operators.trigger_error_alert import resolve_pagerduty_alert
```

```
from operators.redshift_runner import RedshiftOperator
```

Custom SQL
saved in separate
.py file

```
from sql import records_query as hdq
```



```
default_args = {  
    "owner": "airflow",  
    "start_date": datetime(2020, 4, 5),  
    "depends_on_past": False,  
    "email": ["stephanie@stephaniekirmer.com"],  
    "email_on_failure": True,  
    "email_on_retry": False,  
    "retries": 3,  
    "retry_delay": timedelta(minutes=1),  
    "on_failure_callback": partial(trigger_pagerduty_alert, environ),  
    "on_success_callback": partial(resolve_pagerduty_alert, environ),  
    "provide_context": True,  
}
```

```
dag = DAG(  
    dag_id="my_dag_name",  
    default_args=default_args,  
    schedule_interval="30 3 1 * *",  
    catchup=True  
)
```

Note: it accepts the default args here

Scheduling intervals are in CRON syntax: crontab.guru is a big help

Operators define individual actions for Airflow to complete.

A task in Airflow is asking for an action to happen. An Operator is a class that you call to initialize the action.

What kind of task?

- ❖ Run a bash script?
- ❖ Run a python script?
- ❖ Take data from S3 and drop it in Redshift?
- ❖ Run a command on Redshift?
- ❖ Something else?

Many operators are already defined, and you can use these off the shelf.

If you need something else:

- ❖ Start with an existing operator, look at the source code.
- ❖ Can you adapt it? Copy and paste the source, make your edits, save in your environment.
- ❖ If not, use it as a model and write your own from scratch

Ask Airflow to Initialize an Athena Query.

Locate within the dag, get all the default args

```
find_records = AWSAthenaOperator(  
    task_id="find_records",  
    dag=dag,  
    query=hdq.query_text.format(datestr="{{ ds }}"),  
    aws_conn_id="aws_default",  
    output_location="s3://my_bucket/my_folder/",  
    database="production",  
)
```

A built-in Airflow macro!

Most arguments are specific to the AWS Athena operator

```
class RedshiftOperator(BaseOperator):
```

```
    template_fields = ["sql_statement"]
```

```
    @apply_defaults
```

```
    def __init__(self, redshift_conn_id, sql_statement, *args, **kwargs):
```

```
        self.redshift_conn_id = redshift_conn_id
```

```
        self.sql_statement = sql_statement
```

```
        super().__init__(*args, **kwargs)
```

```
    def execute(self, context):
```

```
        self.hook = PostgresHook(postgres_conn_id=self.redshift_conn_id)
```

```
        conn = self.hook.get_conn()
```

```
        cursor = conn.cursor()
```

```
        log.info("Connected to " + self.redshift_conn_id)
```

```
        cursor.execute(self.sql_statement)
```

```
        cursor.close()
```

```
        conn.commit()
```

```
        log.info("Redshift SQL command completed")
```

```
        return True
```

This lets us pass
Airflow macros inside
the query body

Take these steps to make Airflow work for you.

Understand Acyclic Graphing (order of tasks)

- ❖ What really needs to happen first?
- ❖ Think creatively, so you can use the parallelization benefits

Use built in credentials management

- ❖ Let Airflow manage the creds for database connections

Practice navigating the GUI

- ❖ Find your logs, read the graphs to debug easily

Mistakes I've Made with Airflow

Lousy memory management

- ❖ Holding large data volumes in memory is unnecessary expense - your workers may have issues

Too much repetition in DAG code

- ❖ Think programmatically, be concise
- ❖ Use the default args, try Jinja templating

Sloppy Logging/Failure Tracking

- ❖ Quietly record metadata on results of your jobs in case you need to postmortem
- ❖ Don't page for every little error, but make sure someone is informed if a job fails

Practical Examples



Airflow + Redshift can help you achieve your goals.

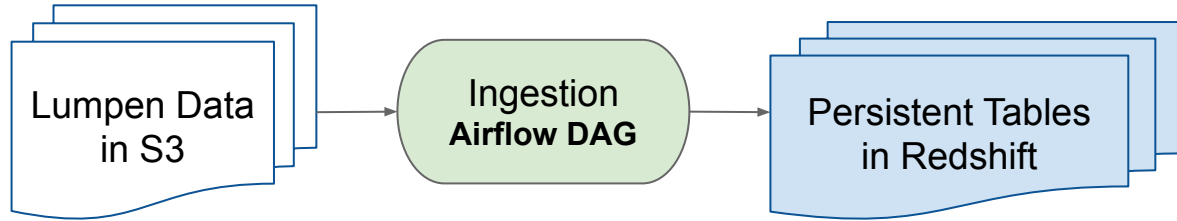
- ❖ Get data from source to an end-user friendly storage solution
- ❖ Establish reliable, transparent, and low-maintenance update, analytics, and management processes
- ❖ Make sure end users can build, test, and evaluate their models with minimal angst

Success can be measured concretely.

Build and populate a data warehouse	Run scheduled data analysis tasks and models	Update and manage data warehouse	Make data warehouse easy to use
Scheduled, transparent jobs Built-in checks for data quality and typing	Train models on a schedule or with one click Store results in Redshift hands-free	Use Redshift features to minimize resource use Monitor and log changes to data warehouse	Redshift <> R/Python Direct querying with SQL from IDE

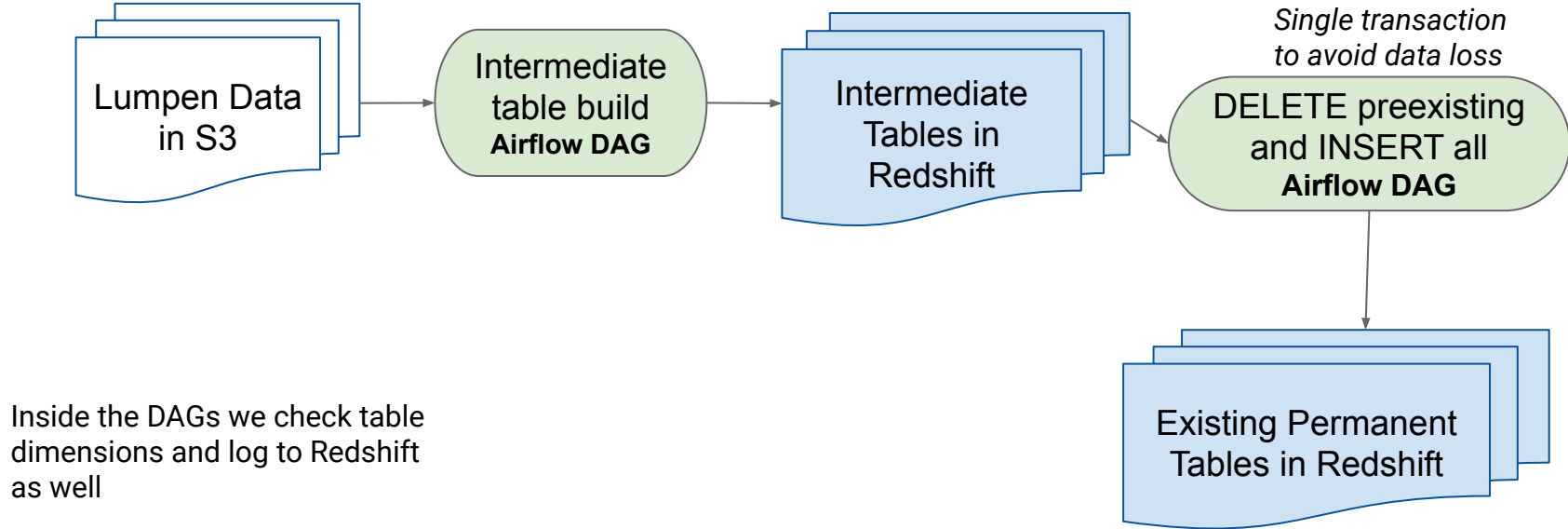
As in any technical implementation, you should have acceptance criteria to define success.

Example New Data Ingestion Workflow



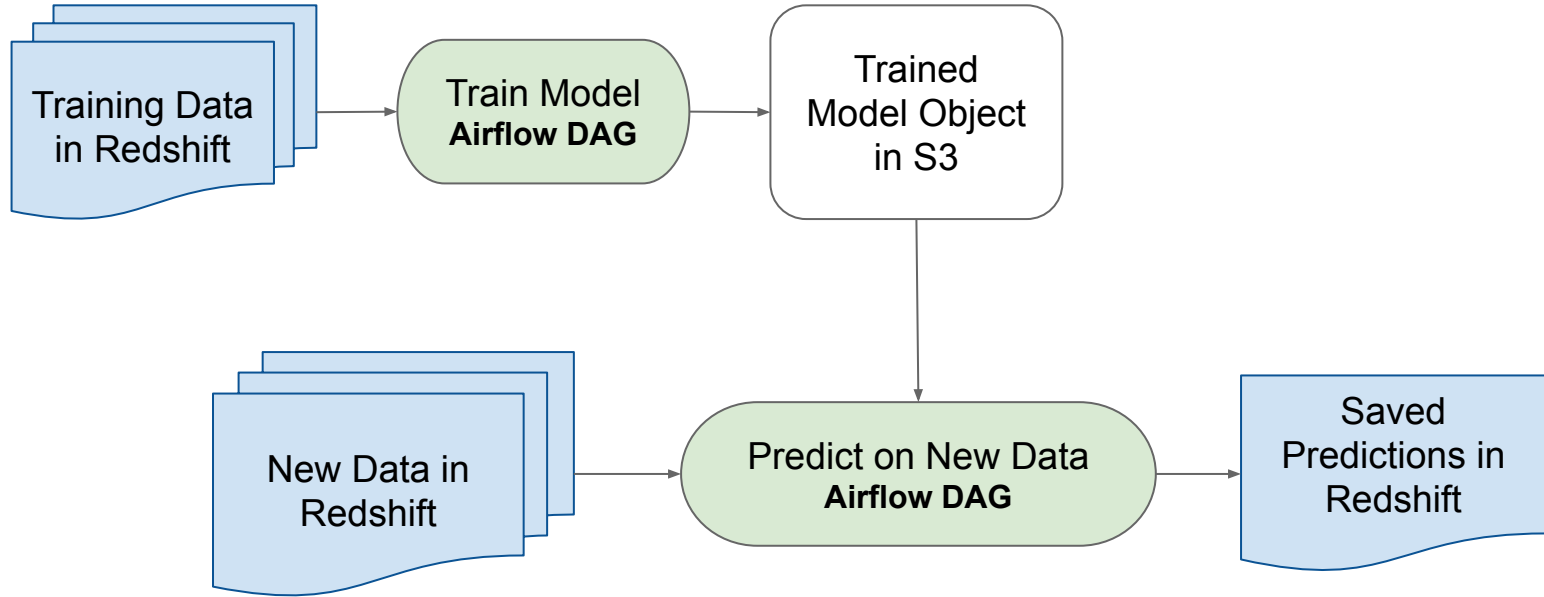
Airflow is more or less running COPY for you on a schedule with robust error handling.

Example Data Updating Workflow



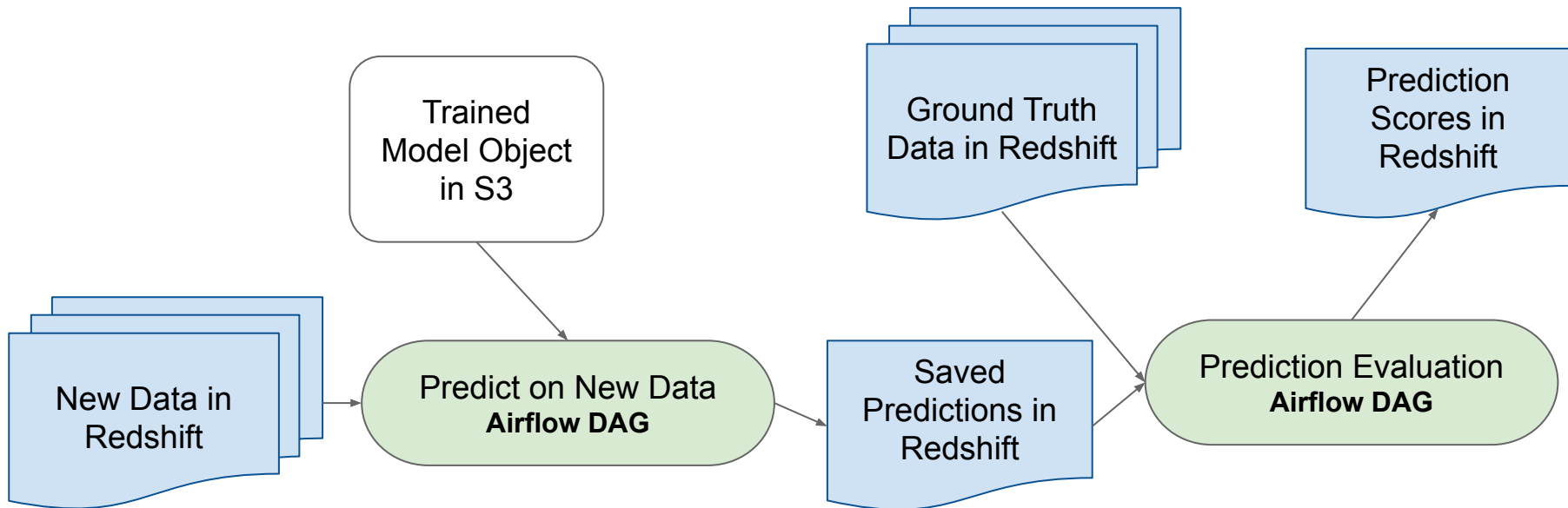
Why DELETE + INSERT, not UPDATE? Put a pin in it, I will explain in a moment!

Example ML Workflow



Airflow runs two different jobs to make the pipeline work – training will be less frequent.

Example Prediction Evaluation Workflow



This can be one DAG with many tasks in it, depending on how you get ground truth data.

A Moment on UPDATE

Redshift would rather not

TL;DR: Redshift has to process every column in order to update one row

Long answer:

Compression and columnar data storage means that it's unusually resource intensive to update row by row. The resource intensiveness and slowness is dramatic.

Instead:

- ❖ Create Temp Table 1 with the new row values
- ❖ Delete any rows from Existing Table that would have been amended
- ❖ INSERT rows from Temp Table 1 into Existing Table
- ❖ Delete Temp Table 1 as cleanup

INSERT is much easier for Redshift to handle.

Finishing touches can make your life much easier.

Develop **logging and monitoring** processes in the DAGs so that you can catch any problems

- ❖ Table size checks
- ❖ Data quality checks
- ❖ DAG failure monitoring/log review

Teach end users to write their own DAGs - empower users to generate value themselves.

Document, document, document - make sure you have a data dictionary or API guide.

Additional Links/Resources

Redshift

- ❖ Using COPY:
https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html
- ❖ Analyze compression:
https://docs.aws.amazon.com/redshift/latest/dg/r_ANALYZE_COMPRESSION.html
- ❖ Sort keys:
https://docs.aws.amazon.com/redshift/latest/dg/c_best-practices-sort-key.html
- ❖ Dist keys:
https://docs.aws.amazon.com/redshift/latest/dg/c_best-practices-best-dist-key.html
- ❖ Optimizing queries:
https://docs.aws.amazon.com/redshift/latest/dg/c_designing-queries-best-practices.html
- ❖ https://docs.aws.amazon.com/redshift/latest/dg/r_EXPLAIN.html


Additional Links/Resources

Airflow

- ❖ Tutorial:
<https://airflow.apache.org/docs/stable/tutorial.html>
- ❖ Operators:
<https://airflow.apache.org/docs/stable/howto/operator/index.html>
- ❖ Built in macros:
<https://airflow.apache.org/docs/stable/macros-ref.html>
- ❖ The codebase! <https://github.com/apache/airflow>
- ❖ Structuring a DAG:
<https://airflow.apache.org/docs/stable/tutorial.html#it-s-a-dag-definition-file>

Thank You!

Questions?

Stephanie Kirmer
www.stephaniekirmer.com
 @data_stephanie