Twiggle

Data Pipelines: **Airflow vs. Luigi** (by people who've made mistakes in both)

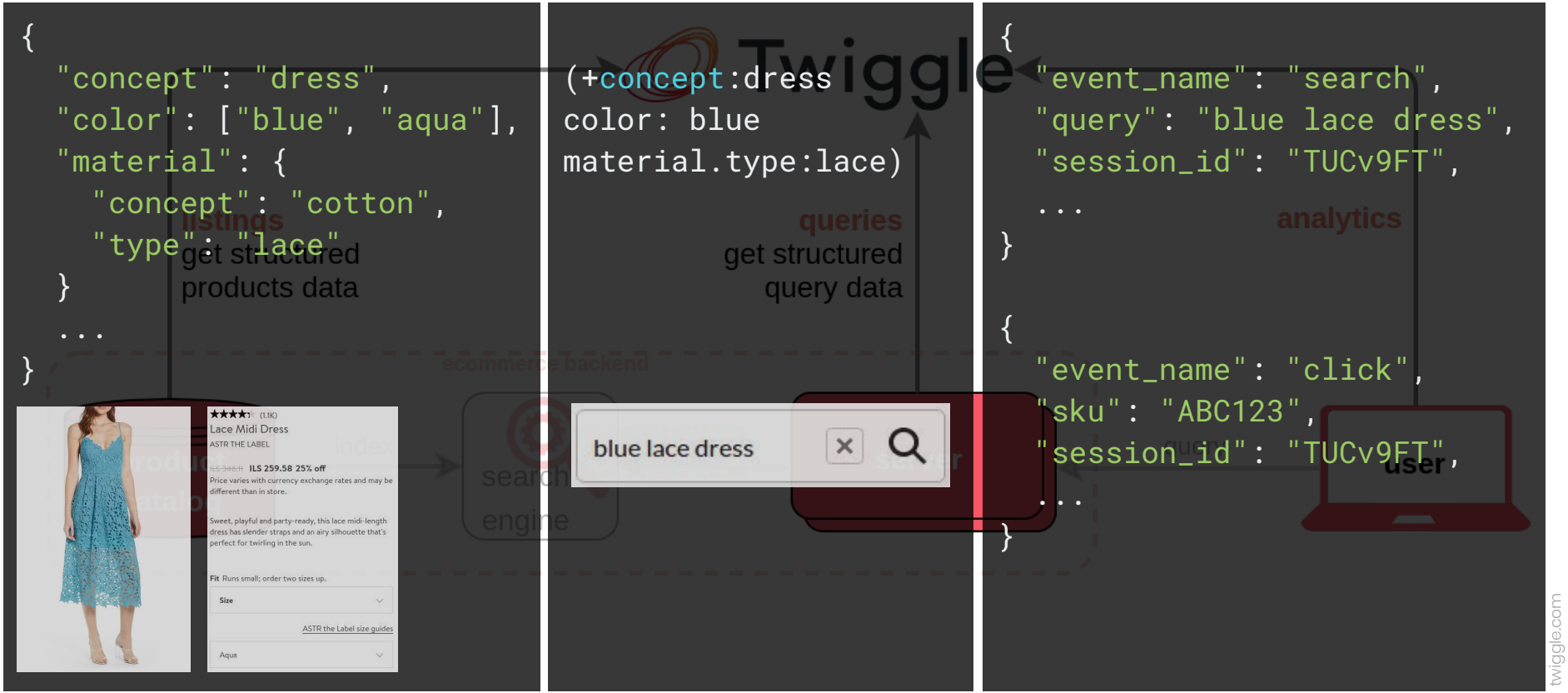
Alex Levin & Orr Shilon

PyCon IL, June 2019

Twiggle

- Search for ecommerce
  - semantically understanding products and queries
  - creating structured data for them
  - matching structured queries to structured products
- e.g. “long dress short sleeves” is hard to match by keywords

Twiggle search flow



Agenda

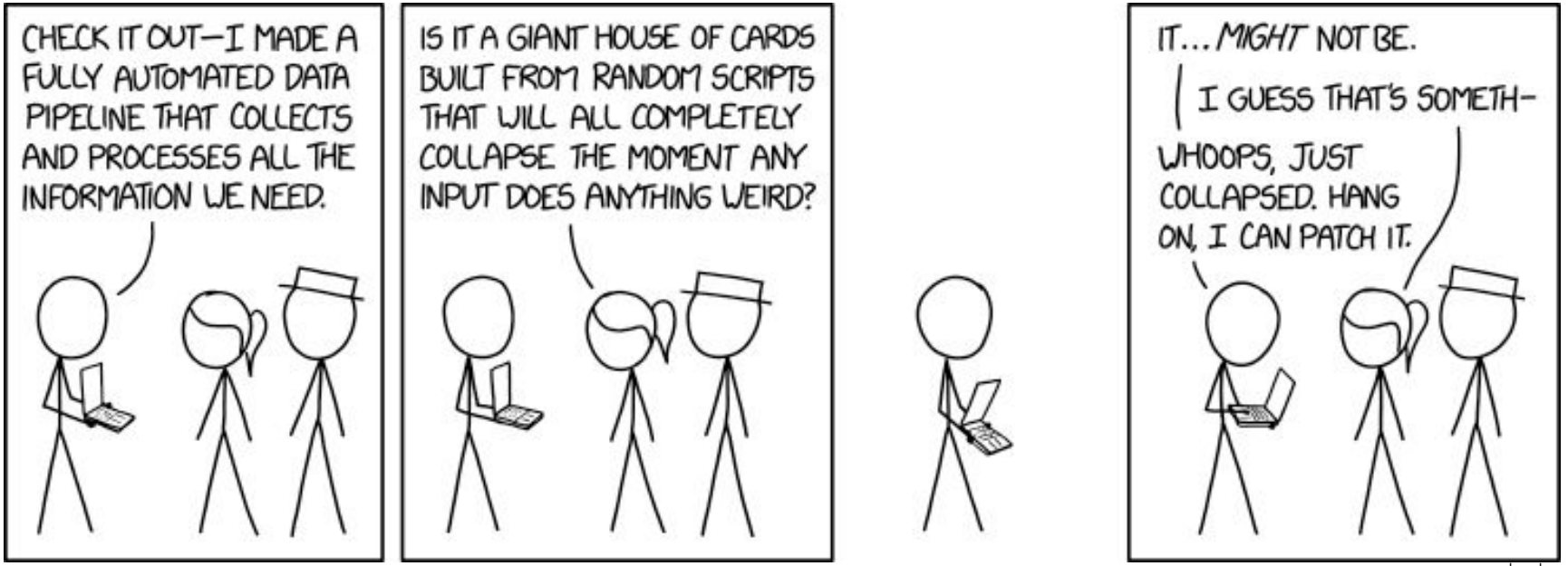
- Workflows
  - Why?
  - Workflow Framework
- Airflow
  - Overview
  - Code Example
- Luigi
  - Overview
  - Code Example
- Comparison

Workflows

Workflows

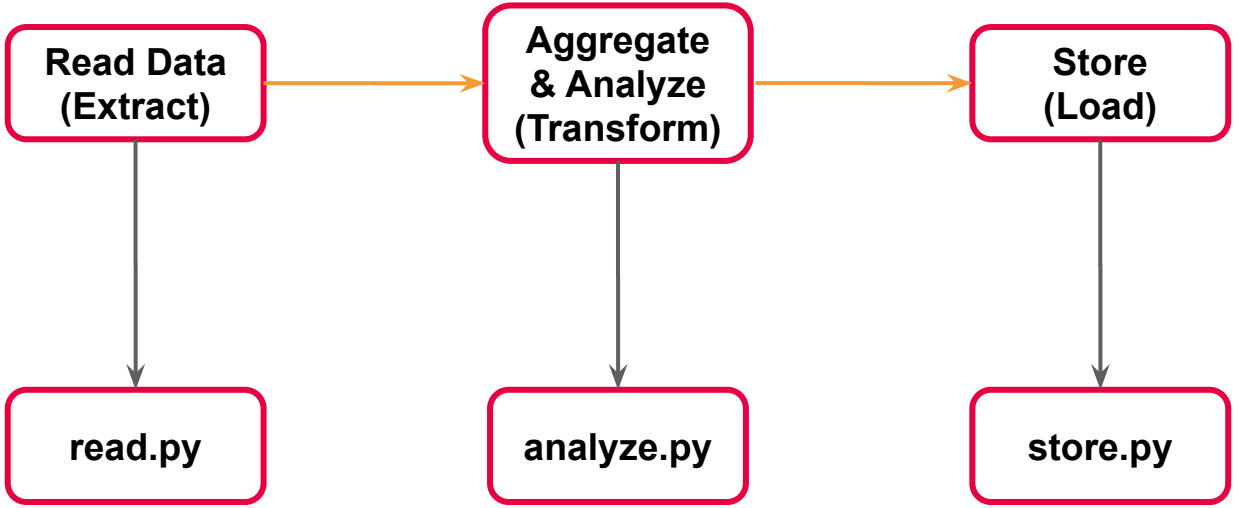
Work·flow /ˈwərkflō/

noun Just a fancy word for sequential (or concurrent) tasks, sometimes with some conditionals in between



xkcd

Workflows: What is it good for?



Workflows: What is it good for?

```
* 0 * * * ~/workdir/read.py
* 1 * * * ~/workdir/analyze.py
* 2 * * * ~/workdir/store.py
```

Workflows: What is it good for?

- ✘ Add more time between the cron jobs?? (hmmmm)
- ✘ Write a wrapper around the jobs? (sounds right but why work hard?)

- ✔ Use an existing workflow framework

Programmatic Workflow Framework

- Makes building workflows consistent and easy
- Must include:
  - Syntax to code each task
  - Ability to configure sequence, dependency and branching of tasks
- May include:
  - Logging / metrics
  - UI
  - Scheduler
  - Error handling
  - Notifications

Apache Airflow

Luigi

OOZIE

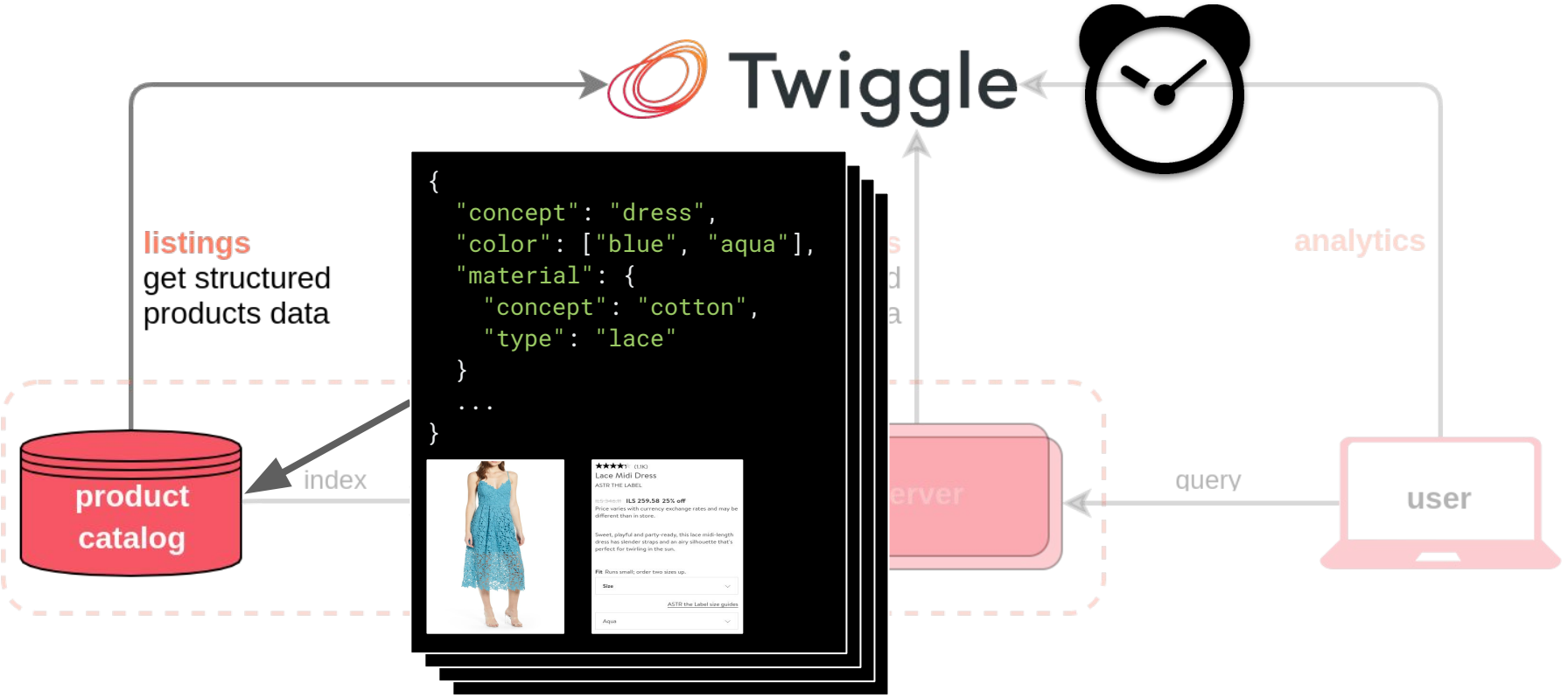
PREFECT

Airflow

Airflow

- Open source - created by Airbnb (2014), maintained by Apache Foundation
- Scheduling
- Error handling and recovery
- Extensive UI - logs, metrics, graphs..
- Many plugins:
  AWS, GCP, bash, Slack, RDBMS, Docker, Spark, ..
- Easily extensible
- Requires a few nodes: scheduler, webserver, workers, database
  - GCP has a managed option

Use Case - Enrich Customers' Product Catalog



Use Case - Enrich Customers' Product Catalog

- Steps
  - Download customer catalog
  - Enrich with Twiggle structured data
  - Upload enriched catalog
- Run on a nightly schedule
- Allow customers to trigger enrichment workflow via API
- UI (nice to have)
- Logs / Metrics
- Notifications

Airflow – basic DAG (direct acyclic graph)

```
dag = DAG('my_dag', schedule_interval='* 0 * * *')
```

```
print_context_task = PythonOperator(
    task_id='print_context',
    provide_context=True,
    python_callable=lambda ds, **context: print(context),
    dag=dag)
```

```
sleep_task = PythonOperator(
    task_id='sleep_task',
    python_callable=lambda: time.sleep(5),
    dag=dag)
```

```
print_context_task >> sleep_task
```

Airflow – Running a DAG

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About | 2019-05-29 09:22:43 UTC

DAGs

Search:

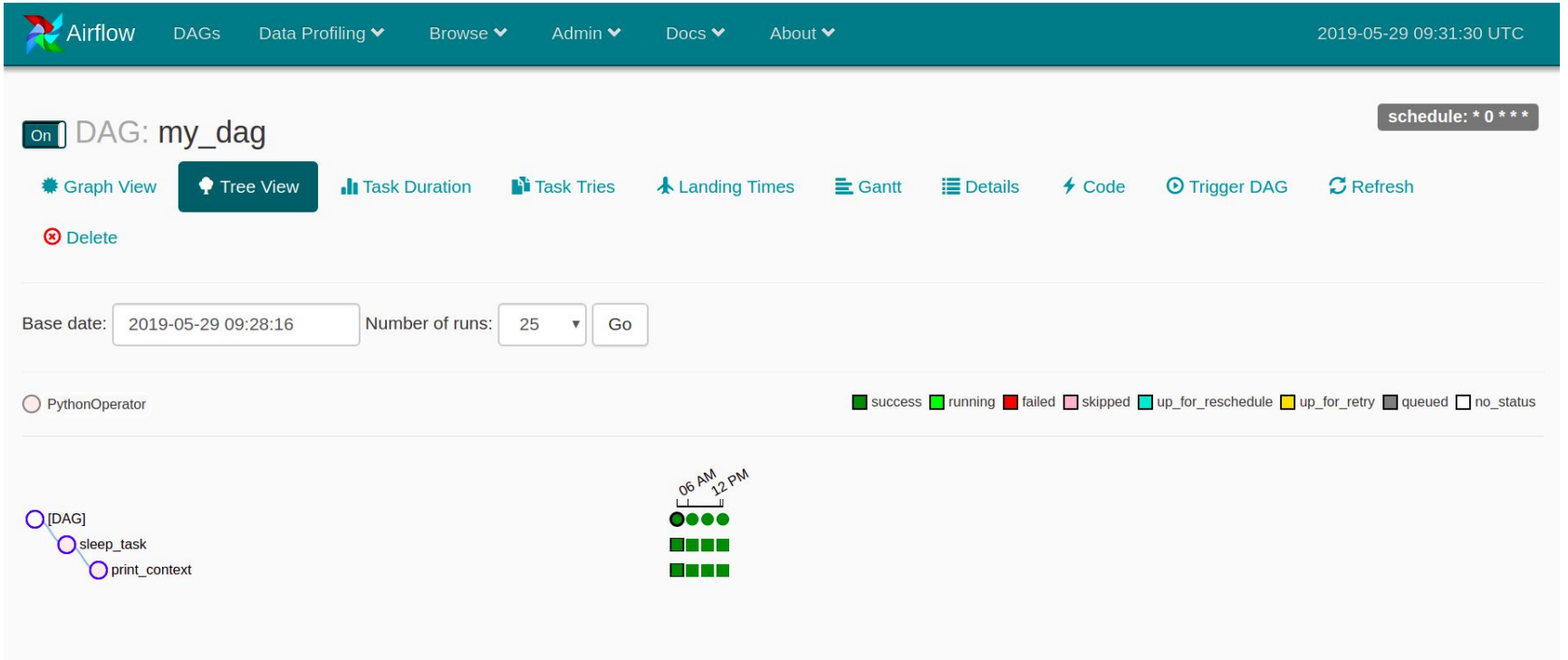
		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	On	my_dag	* 0 * * *	Airflow	2	2019-05-29 00:58	1	
	On	sleep	None	Airflow	2	2019-05-12 13:19	1	
	On	sleep_triggered	None	Airflow	2	2019-05-12 13:19	6	
	On	sleep_variable	None	Airflow	2	2019-05-12 13:19	1	

Showing 1 to 4 of 4 entries

« < 1 > »

Hide Paused DAGs

Airflow - DAG metrics



Airflow – task details

Airflow DAGs Data Profiling Browse Admin Docs About 2019-05-30 11:34:05 UTC

Task Instance Details

Dependencies Blocking Task From Getting Scheduled

Dependency	Reason
Task Instance State	Task is in the 'success' state which is not a valid state for execution. The task must be cleared in order to be run.

Attribute: python_callable

```
1    python_callable=lambda ds, **context: pprint(context),
```

Task Instance Attributes

Attribute	Value
dag_id	my_dag
duration	None
end_date	2019-05-30 11:33:05.502600+00:00
execution_date	2019-05-30T00:58:00+00:00
executor_config	{}
generate_command	<function TaskInstance.generate_command at 0x7f9a4a31a048>

Airflow – task logs

Airflow DAGs Data Profiling Browse Admin Docs About 2019-05-29 09:36:16 UTC

On DAG: my_dag

schedule: * 0 * * *

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code Trigger DAG Refresh

Delete

Task Instance: print_context 2019-05-29 09:28:16

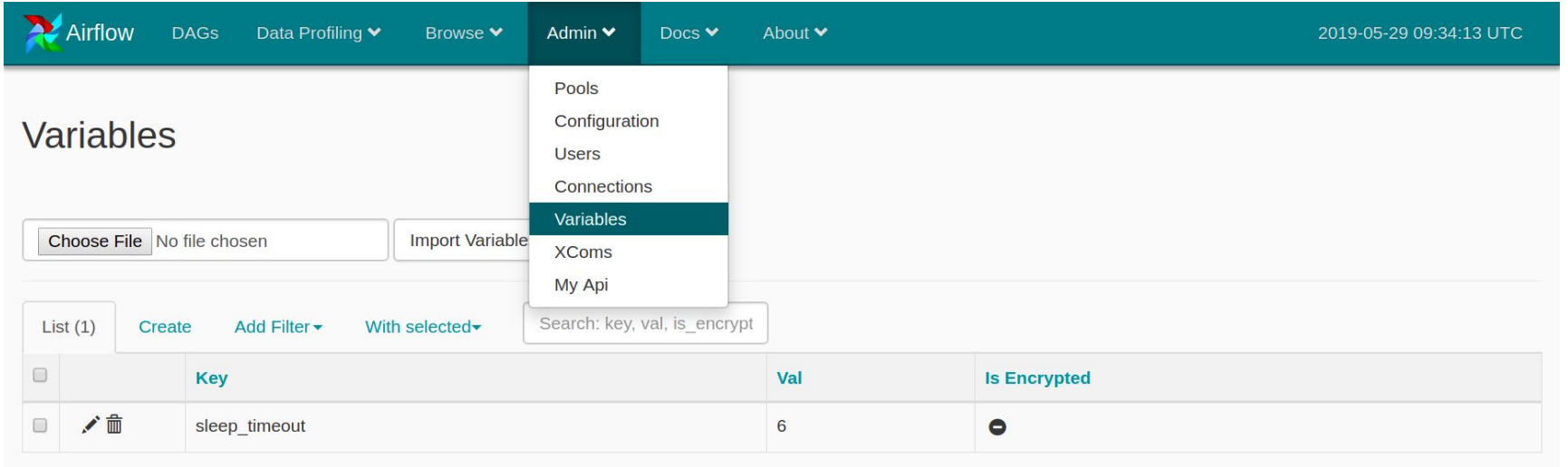
Task Instance Details Rendered Template Log XCom

Log by attempts

1

```
*** Reading local file: /home/orr/dev/pycon-israel-2019-airflow-luigi/airflow/logs/my_dag/print_context/2019-05-29T09:28:16.112570+00:00/1.log
[2019-05-29 12:28:27,960] {__init__.py:1139} INFO - Dependencies all met for <TaskInstance: my_dag.print_context 2019-05-29T09:28:16.112570+00:00 [queue
[2019-05-29 12:28:27,969] {__init__.py:1139} INFO - Dependencies all met for <TaskInstance: my_dag.print_context 2019-05-29T09:28:16.112570+00:00 [queue
[2019-05-29 12:28:27,969] {__init__.py:1353} INFO -
--------------------------------------------------------------------------------
[2019-05-29 12:28:27,969] {__init__.py:1354} INFO - Starting attempt 1 of 1
[2019-05-29 12:28:27,969] {__init__.py:1355} INFO -
--------------------------------------------------------------------------------
[2019-05-29 12:28:27,985] {__init__.py:1374} INFO - Executing <Task(PythonOperator): print_context> on 2019-05-29T09:28:16.112570+00:00
[2019-05-29 12:28:27,986] {base_task_runner.py:119} INFO - Running: ['airflow', 'run', 'my_dag', 'print_context', '2019-05-29T09:28:16.112570+00:00', '-
```

Airflow – using variables to configure DAGs

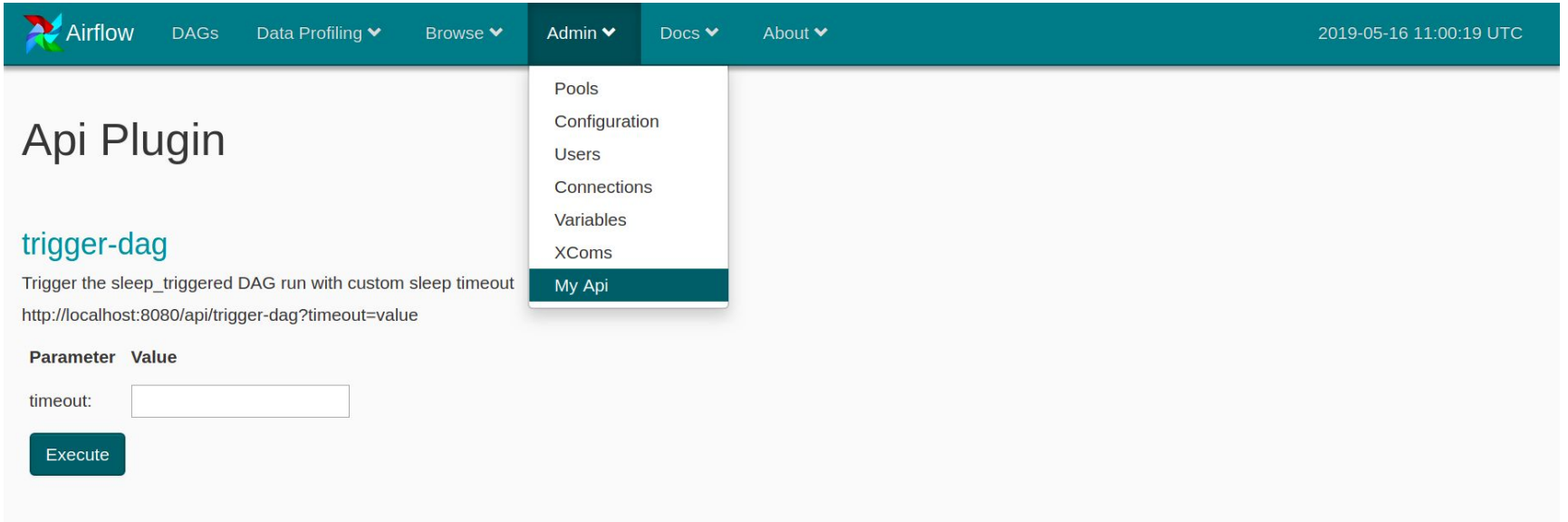



```
sleep_timeout = float(Variable.get('sleep_timeout', default_var=5))
```

```
sleep_task = PythonOperator(
    task_id='sleep_task',
    python_callable=lambda: time.sleep(sleep_timeout),
    dag=dag)
```

Airflow plugin - API to trigger DAG with param

http://localhost:8080/api/trigger-dag?timeout=7



Airflow plugin – API to trigger DAG with param

rewrite our sleep task yet again..

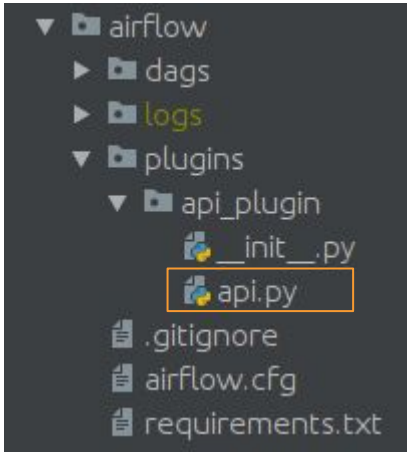
```
def sleep_by_configuration(**context):
    sleep_timeout = float(context['dag_run'].conf.get('sleep_timeout', 5))
    time.sleep(sleep_timeout)
```

```
sleep_task = PythonOperator(
    task_id='sleep_task',
    python_callable=sleep_by_configuration,
    provide_context=True,
    dag=dag)
```

Airflow plugin – flask API endpoint

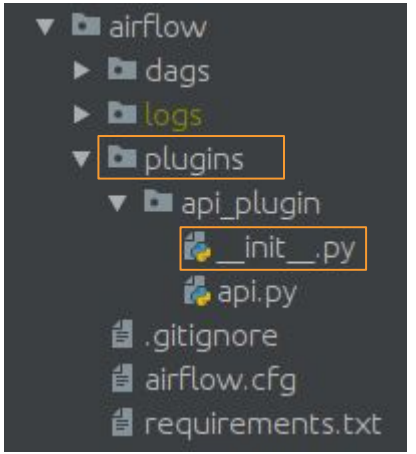
```
api_bp = Blueprint('api_bp', __name__, url_prefix='/api')
```

```
@api_bp.route('/trigger-dag', methods=['GET'])
def trigger_dag():
    try:
        sleep_timeout = request.args.get('timeout', 5)
        trigger_dag.trigger_dag(
            'my_dag',
            conf={'sleep_timeout': sleep_timeout}
        )
        return jsonify({'result': 'success'}), 200
    except Exception:
        return jsonify({'result': 'failure'}), 400
```



Airflow plugin - folder structure and init

```
from api_plugin.api import api_bp

class ApiPlugin(AirflowPlugin):
    name = "api_plugin"
    flask_blueprints = [api_bp]
    # operators = []
    # hooks = []
    # executors = []
    # admin_views = []
    # menu_links = []
```



A working plugin!
code in https://github.com/orrshilon/pycon-israel-2019-airflow-luigi

LUigt

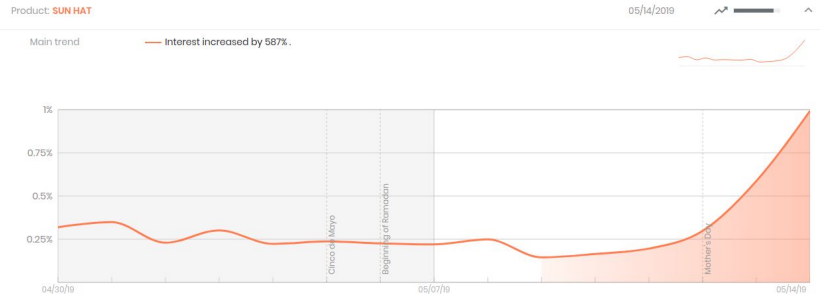
Luigi

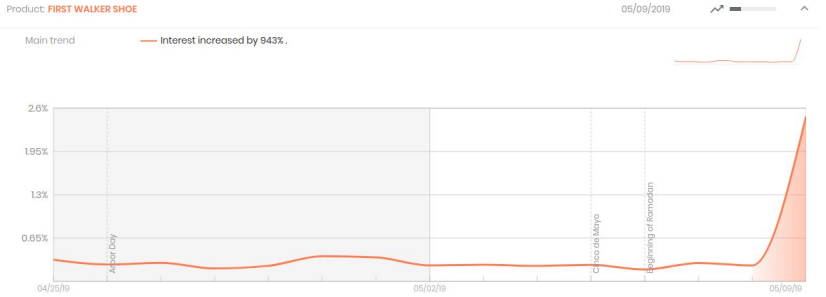
- Open source maintained by Spotify
- Task dependence management
- Minimal boilerplate
- Error control, recovery
- Great off the shelf support:
  AWS, GCP, Spark, Elastic, RMDBS, local, .

Use Case - Twiggle Trends Workflow

Use Case - Twiggle Trends Workflow

↗↙		BRAND	VOLUME	GRAPH
1	–	DISNEY	37%	
2	–	DOLCE GABBANA	23.6%	
3	↑	GEORGE	7.8%	
4	↓	GYMBOREE	6.6%	
5	NEW	CROCS	6.5%	
6	↓	M.A.C	5.5%	
7	↑	GUCCI	4.2%	
8	↑	BIG BROTHER	3%	
9	NEW	ANA	3%	
10	NEW	BOOHOO	2.9%	

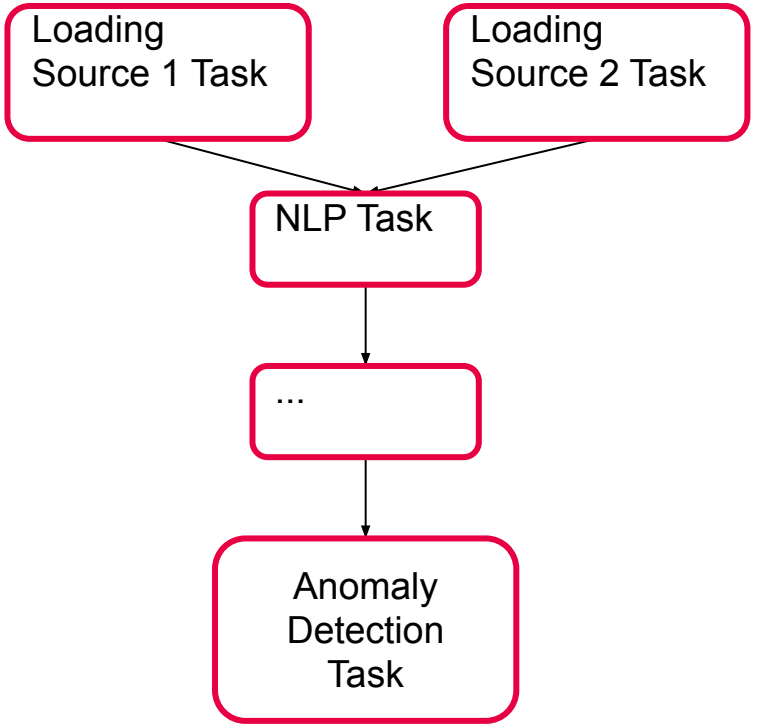




Use Case - Twiggle Trends Workflow

- Main goal: run anomaly detection algos to detect anomalies and trends in user behavior
- But beforehand we need to:
  1. Load data (from different sources)
  2. Run Twiggle NLP
  3. Run sanitation and overrides
  4. Prepare data
  5. **Run Anomaly Detection**

Twiggle Trends Workflow




```python
class NewTask(luigi.task):
    def requires(self):
        # check dependencies
        return PreviousTask()

    def output(self):
        # save output
        return luigi.LocalTarget()

    def run(self):
        # execute
        pass
```

Twiggle Trends Workflow

```
class
LoadDataFromSource
    def requires(se
        # the data
    def output(self
        # save in a
    def run(self):
        # load and
```

```
class
Task(luigi.task):
f):
xists in source 2
:
new location

ranslate data
```

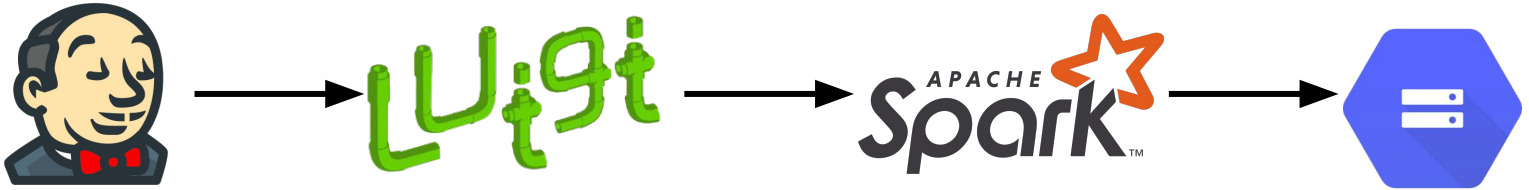
```
class NLPTask(luigi.task):
    def requires(self):
        yield LoadDataFromSource1Task()
        yield LoadDataFromSource2Task()

    def output(self):
        # save NLP output

    def run(self):
        # Run NLP on the data
```

```
python luigi_example.py NLPTask
```

Luigi - How to Scale

Luigi - How to Scale

- Luigi ♥ Apache Spark
- We can run Spark jobs as Luigi tasks! (using PySparkTask)
- In Twiggle, we use the following:
  - Single Luigi server for task orchestration
  - Each (non trivial) task is executed as SparkTask
  - Luigi main task is invoked using Jenkins (once a day)
  - GCS as our Data Lake



Luigi – Spark Example

```
class FileExistsTask
```

```
class SparkWordCountTask(luigi.PySparkTask):
    input_path = luigi.Parameter()
    output_path = luigi.Parameter()

f.input_path)]

KEY', 'SECRET_KEY')
h, client=client)

ath) \
line.split()) \
d, 1)) \
b: a + b) \
.saveAsTextFile(self.output().path)
```

```
# luigi.cfg file

[core]
default-scheduler-host:name_of_your_luigi_server
python-home-dir:$PYTHON_HOME

[spark]
spark-submit:$SPARK_HOME/bin/spark-submit
hadoop-conf-dir:$HADOOP_HOME
yarn-conf-dir:$YARN_HOME
master:yarn
num-executors:10
```

Luigi VS

Comparison

Luigi vs. Airflow - Comparison

Luigi	Airflow
tasks are dependant on data	tasks are dependant on tasks
executed by triggering last task	executed by triggering a workflow
✔ easy native python code reuse	✘ difficult to reuse code - DAGs are dynamically built
✔ easy configuration - single process	✘ hard to configure - runs on multiple processes and a database * managed services exist
✘ no scheduler	✔ built in scheduler

Luigi vs. Airflow - Comparison continued

Luigi	Airflow
✘ poor web UI	✔ nicer web UI
✘ no web server	✔ includes a web server
✔ highly scalable	✔ highly scalable
✔ simple api	✔ simple api
✔ easy to extend	✔ easy to extend - e.g. trigger via HTTP
✔ few dependencies	✘ many dependencies

Twiggle (We are hiring!)

Questions?

Alex Levin : alexl@twiggle.com

Orr Shilon: orrs@twiggle.com

https://github.com/orrshilon/pycon-israel-2019-airflow-luigi