AIRFLOW - DATA FLOW ENGINE FROM AIRBNB

By Walter Liu 2016/01/28

Who am I?

- ~~Archer~~ Architect of TrendMicro Coretech backend
- A new user in Airflow

Website Blocked by Trend Micro™ Titanium™ 2014

Dangerous Page

Address: [illegible]

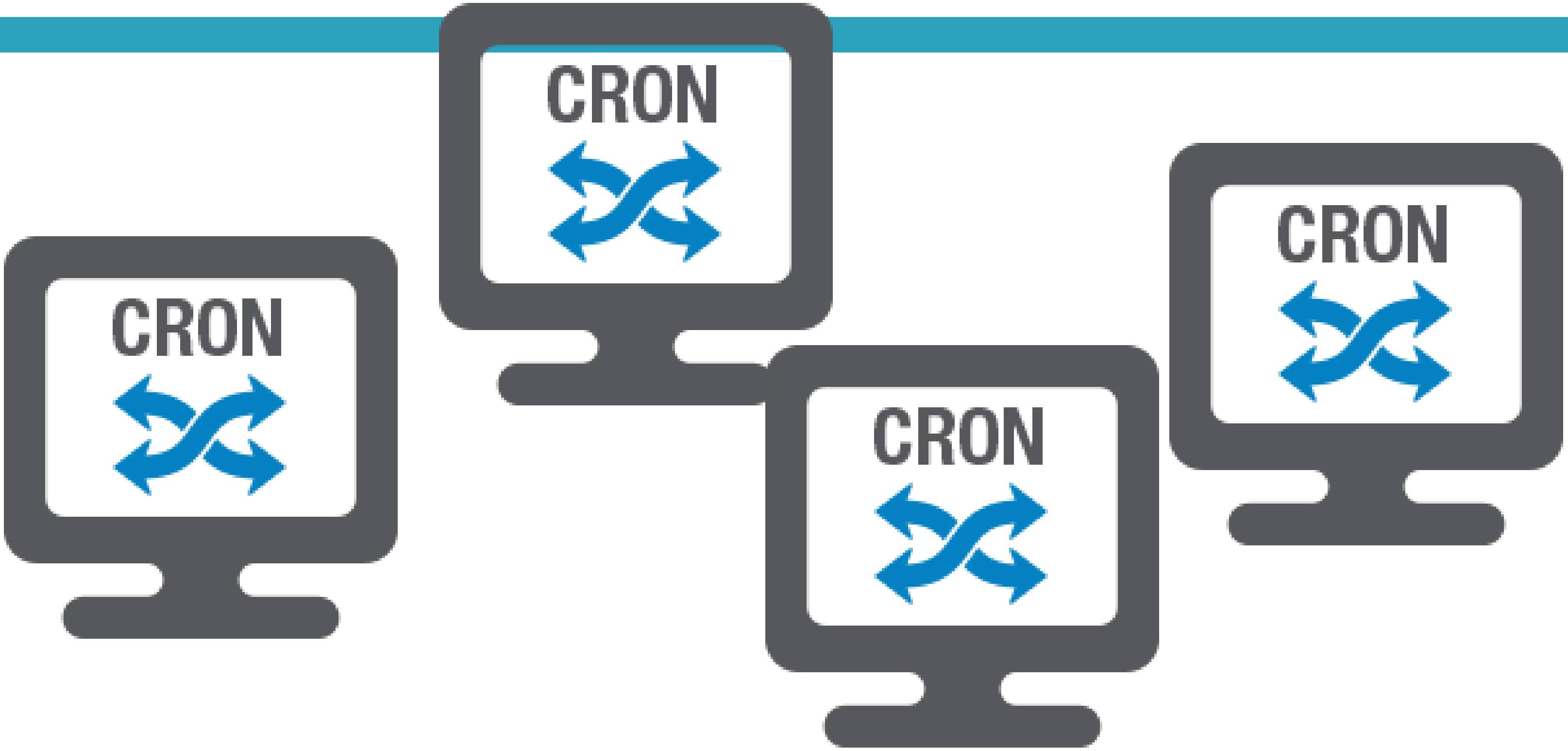
Trend Micro has confirmed that this website can transmit malicious software or has been involved in online scams or fraud.

Please close this page.

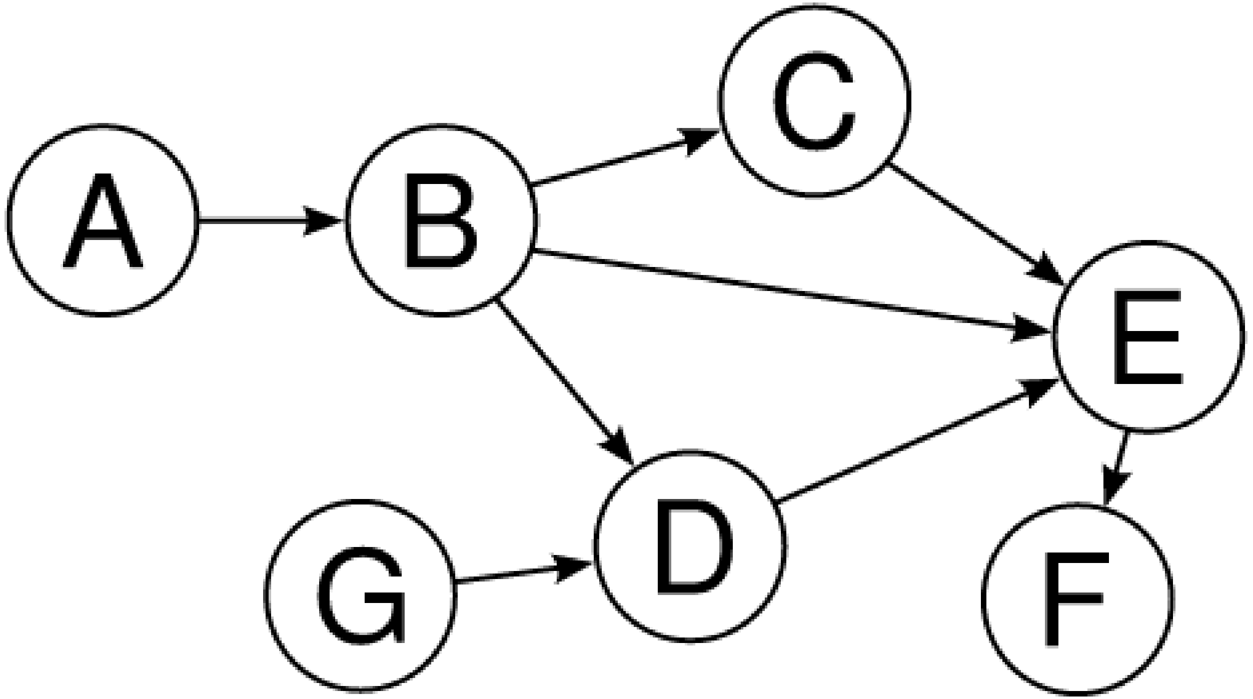
Still want to open this page, despite the risk?

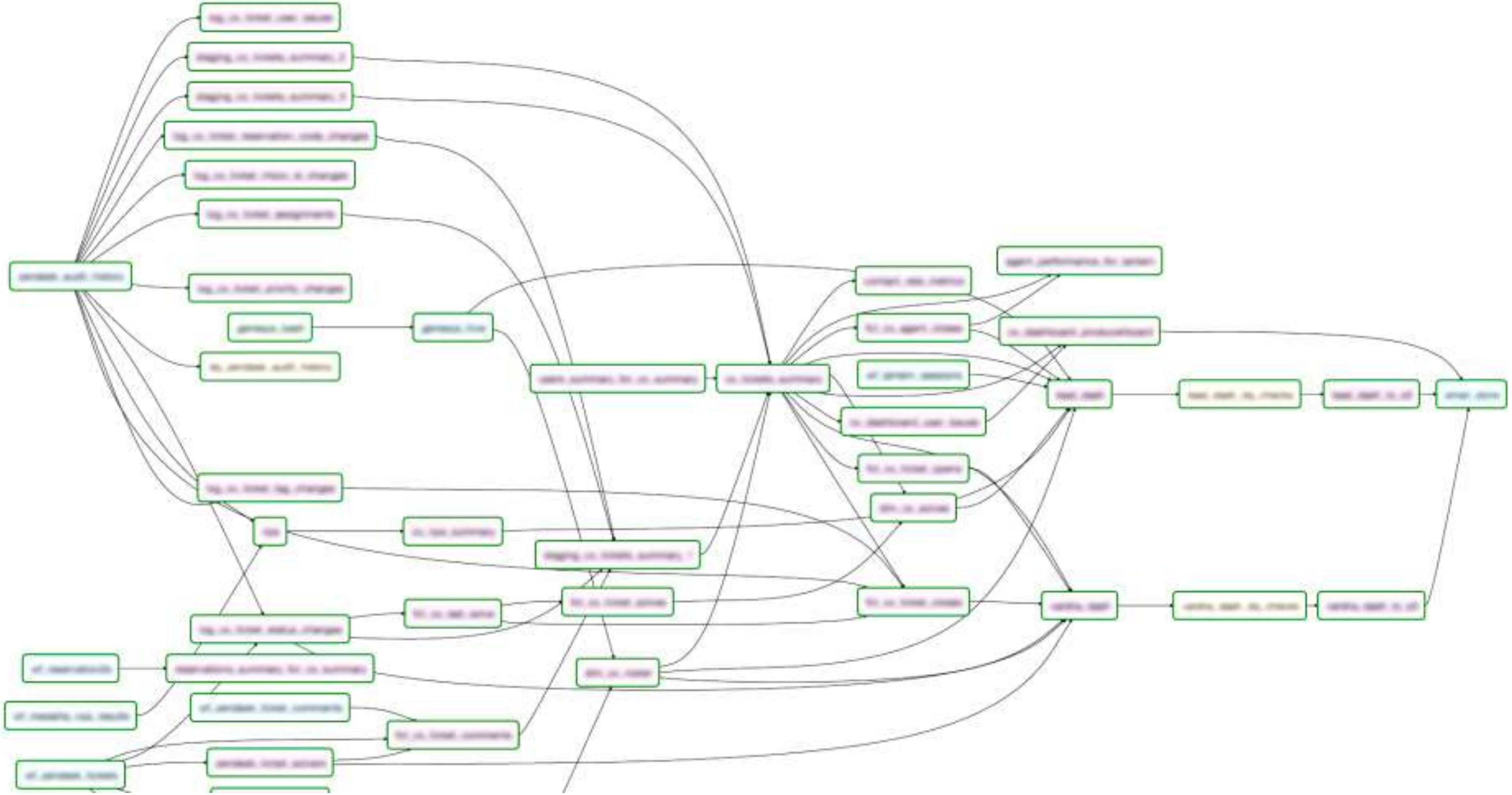
Why Data Flow Engine?

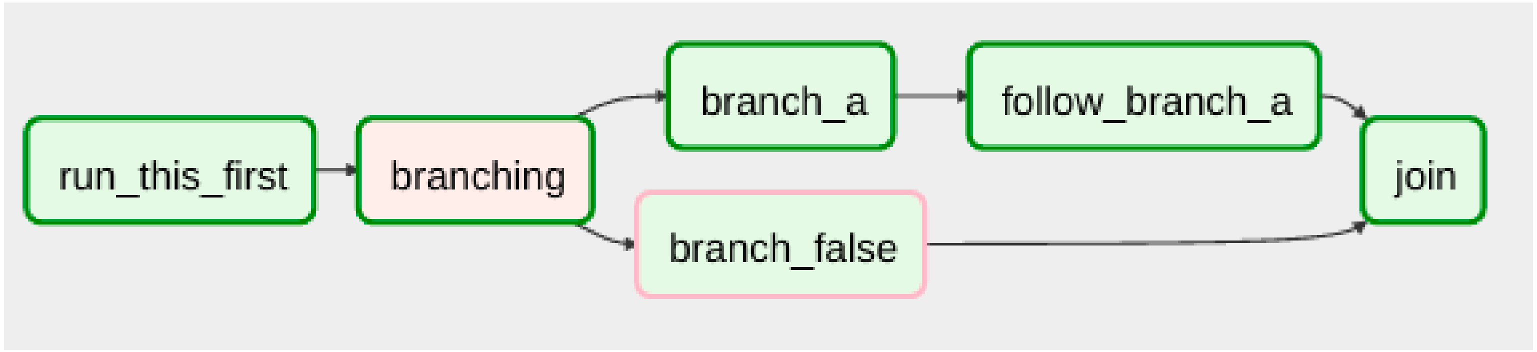
Cron Job



Direct Acyclic Graph (DAG)






run_this_first
branching
branch_a
follow_branch_a
branch_false
join

Data relationships

- Data availability
  - if the data is not there, trigger the process to generate the data.
- Data dependency
  - Some data relies on other data to generate.

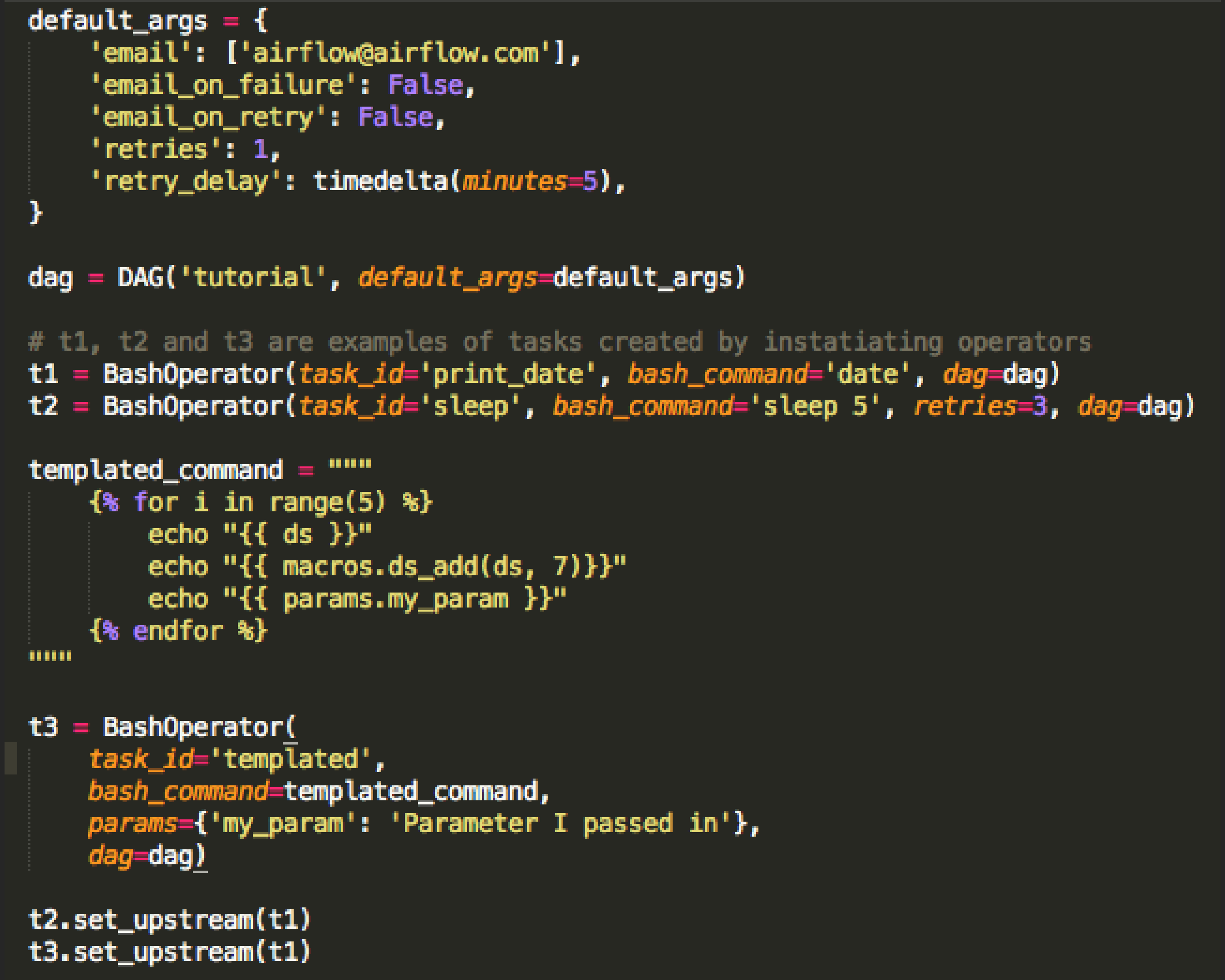
Operability

- Job failed and resume
- Job monitor
- Backfill

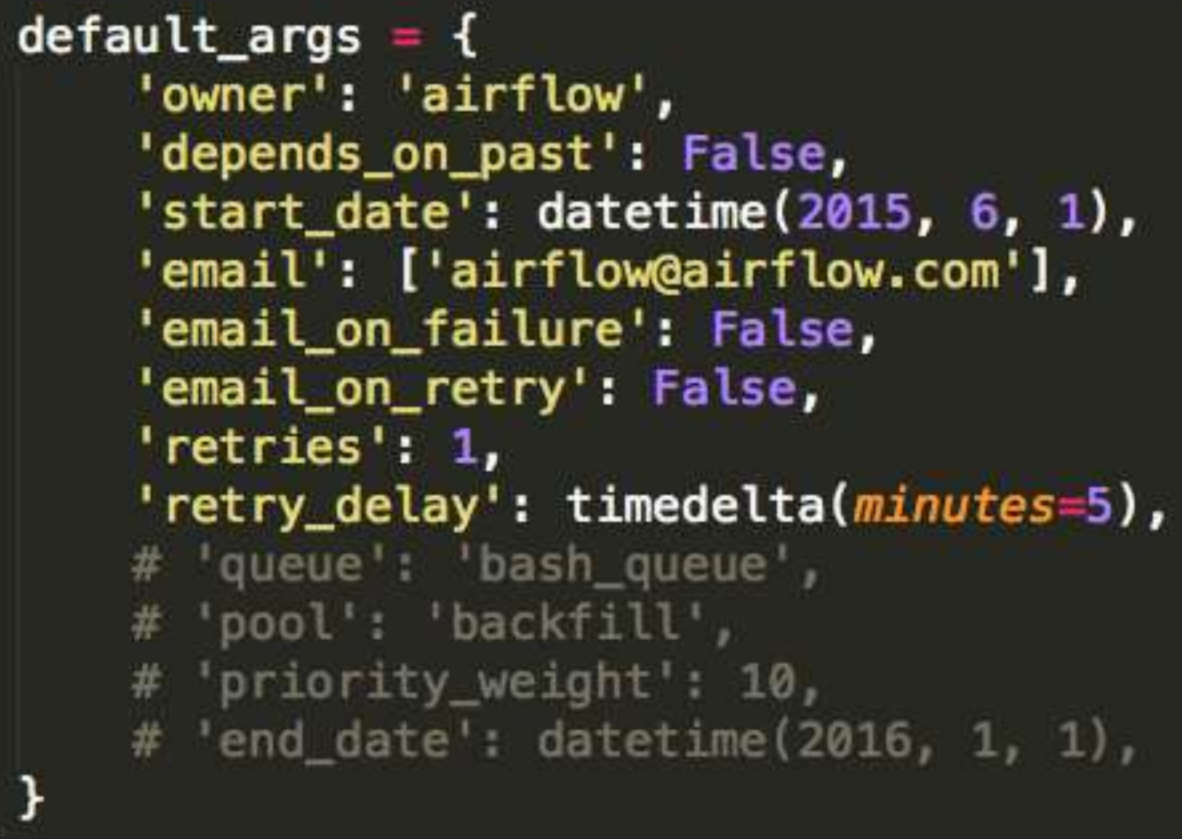
Airflow

airbnb

DAG structure as code



```python
default_args = {
    'email': ['airflow@airflow.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG('tutorial', default_args=default_args)

# t1, t2 and t3 are examples of tasks created by instatiating operators
t1 = BashOperator(task_id='print_date', bash_command='date', dag=dag)
t2 = BashOperator(task_id='sleep', bash_command='sleep 5', retries=3, dag=dag)

templated_command = """
    {% for i in range(5) %}
        echo "{{ ds }}"
        echo "{{ macros.ds_add(ds, 7)}}"
        echo "{{ params.my_param }}"
    {% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t2.set_upstream(t1)
t3.set_upstream(t1)
```



```python
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2015, 6, 1),
    'email': ['airflow@airflow.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
}
```

```
templated_command = """
    {% for i in range(5) %}
        echo "{{ ds }}"
        echo "{{ macros.ds_add(ds, 7)}}"
        echo "{{ params.my_param }}"
    {% endfor %}
"""
```

```
{{ ds }} => now YYYY-MM-DD
{{ yesterday_ds }} => yesterday YYYY-MM-DD
{{ tomorrow_ds }} => tomorrow YYYY-MM-DD
...
```

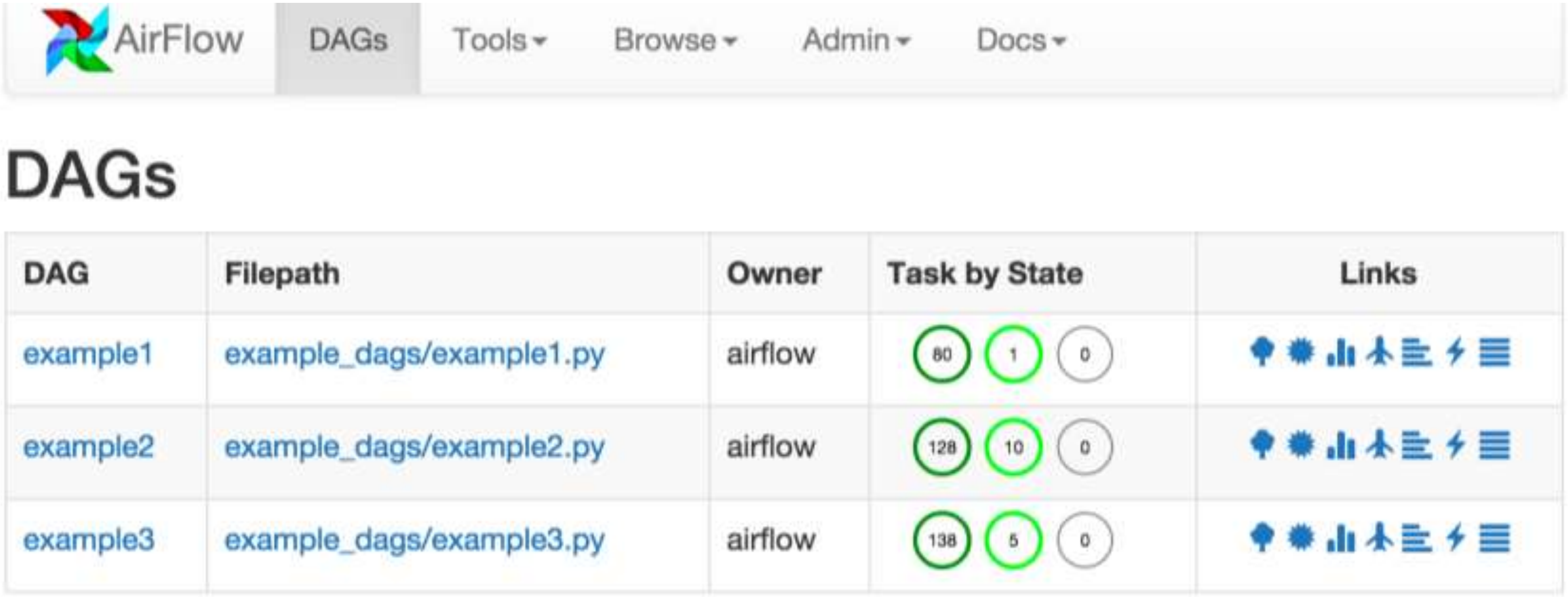
```python
dag = DAG('tutorial', default_args=default_args)

# t1, t2 and t3 are examples of tasks created by instatiating operators
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)

t3 = BashOperator(
    task_id='templated',
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t2.set_upstream(t1)
t3.set_upstream(t1)
```

Demo

- python tutorial.py
- airflow list_dags
- airflow list_tasks tutorial
- airflow list_tasks tutorial --tree
- airflow test tutorial print_date 2015-06-01
- airflow test tutorial sleep 2015-06-01
- airflow run tutorial templated 2015-06-01
- Backfill: airflow backfill tutorial -s 2015-06-07 -e 2015-06-10
  - Run again
  - Run another date range

Site: http://localhost:8080/admin/

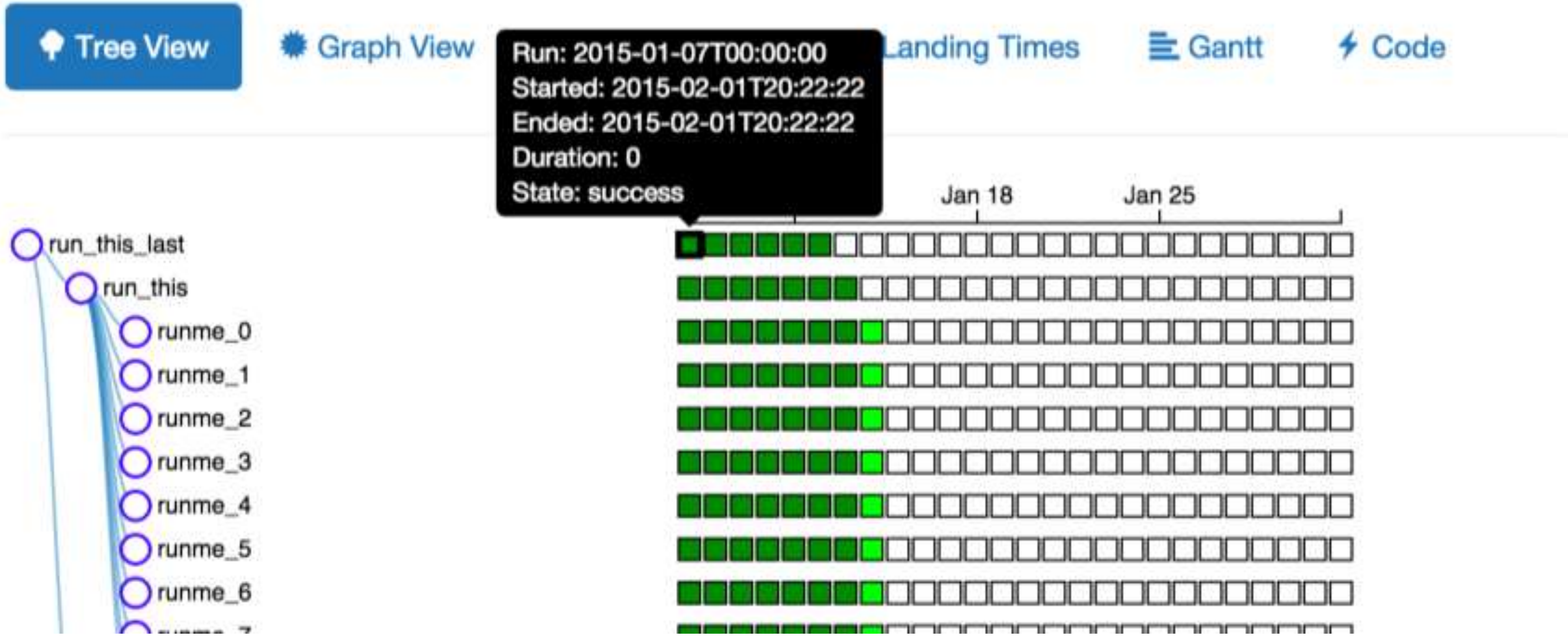
UI – DAG view



DAGs

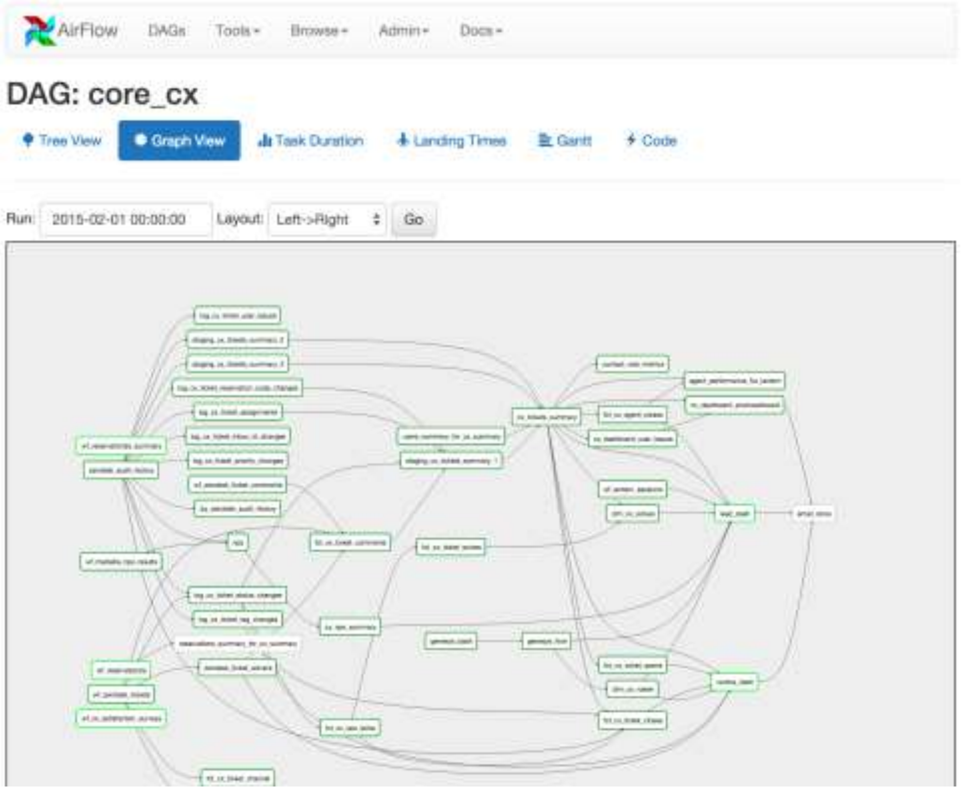
DAG	Filepath	Owner	Task by State	Links
example1	example_dags/example1.py	airflow	80 1 0	
example2	example_dags/example2.py	airflow	128 10 0	
example3	example_dags/example3.py	airflow	138 5 0	

UI – Tree View



Run: 2015-01-07T00:00:00
Started: 2015-02-01T20:22:22
Ended: 2015-02-01T20:22:22
Duration: 0
State: success

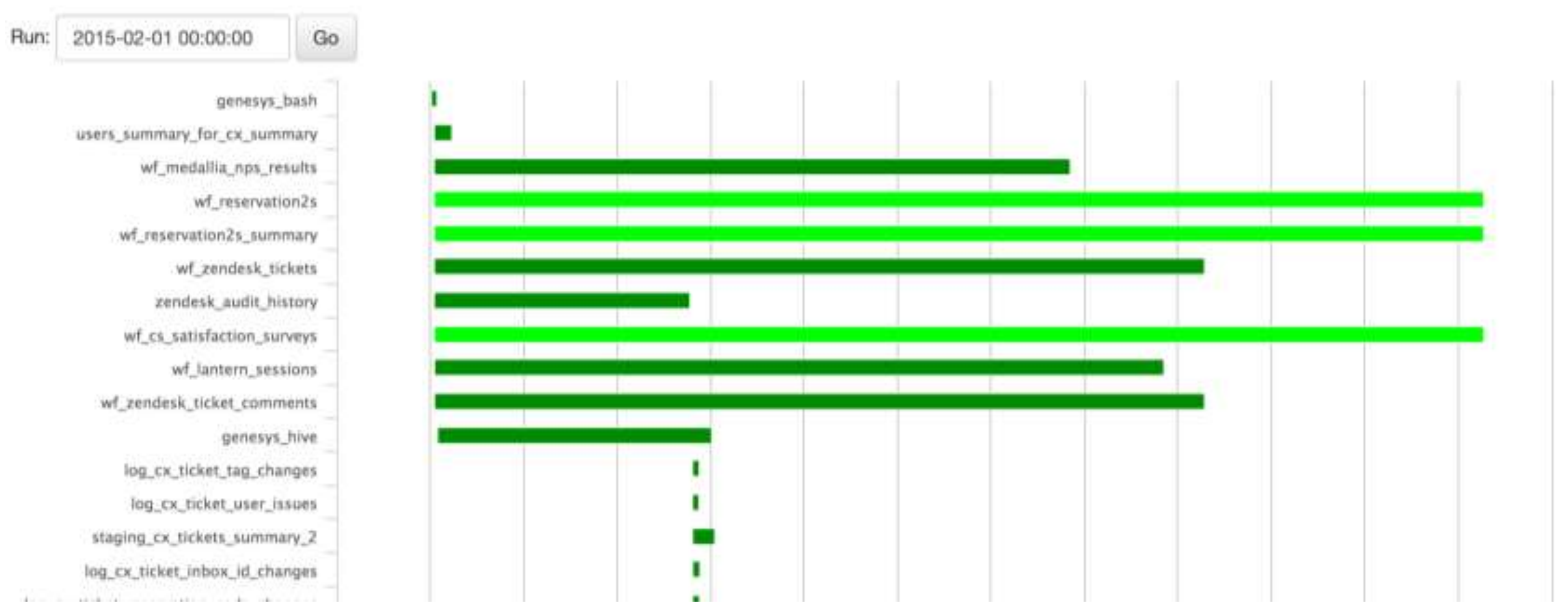
UI – Graph View



DAG: core_cx

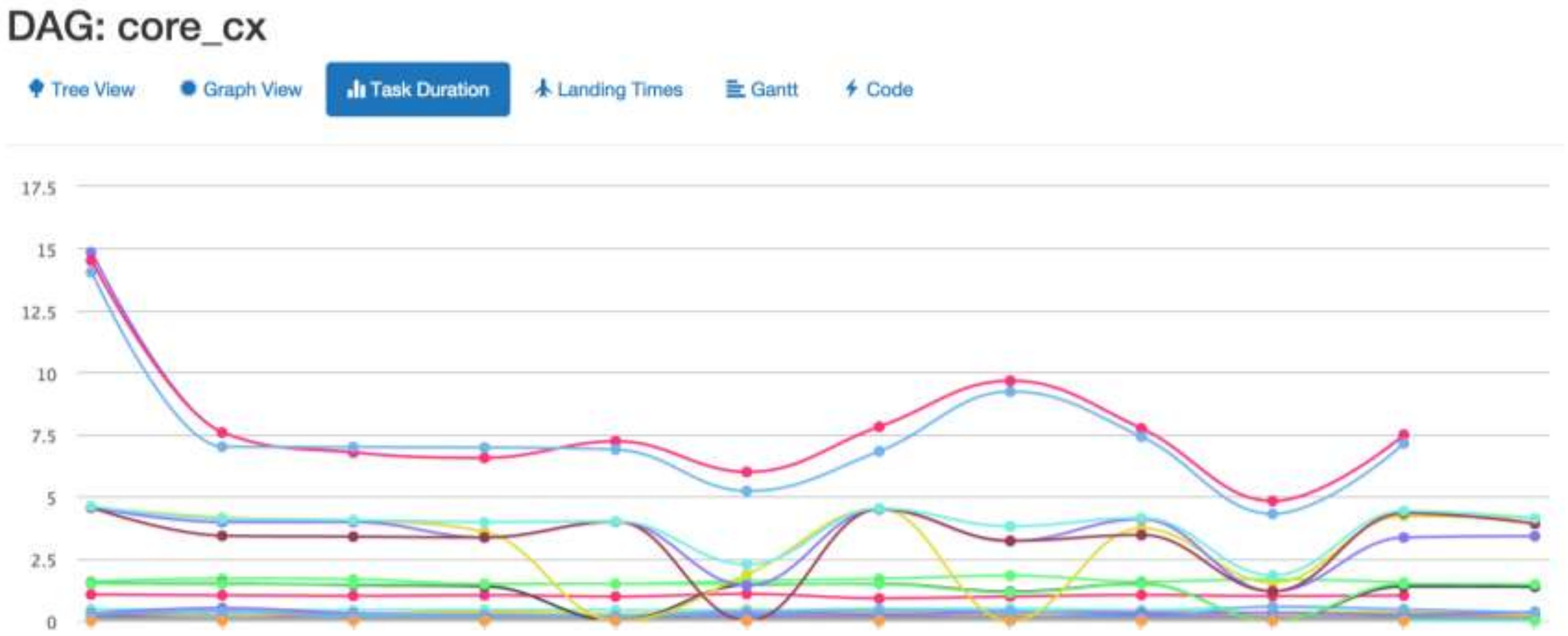
Run: 2015-02-01 00:00:00 Layout: Left->Right

UI - Gantt

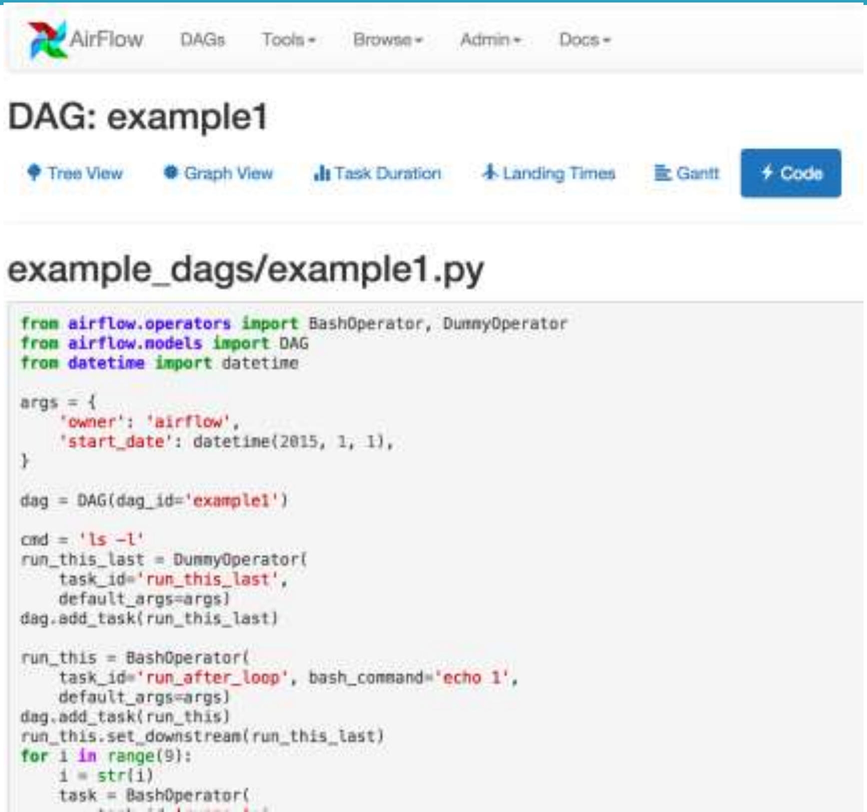

Run: 2015-02-01 00:00:00 Go

genesys_bash
users_summary_for_cx_summary
wf_medallia_nps_results
wf_reservation2s
wf_reservation2s_summary
wf_zendesk_tickets
zendesk_audit_history
wf_cs_satisfaction_surveys
wf_lantern_sessions
wf_zendesk_ticket_comments
genesys_hive
log_cx_ticket_tag_changes
log_cx_ticket_user_issues
staging_cx_tickets_summary_2
log_cx_ticket_inbox_id_changes

UI – Task duration


DAG: core_cx

UI – Code



DAG: example1

example_dags/example1.py

```
from airflow.operators import BashOperator, DummyOperator
from airflow.models import DAG
from datetime import datetime

args = {
    'owner': 'airflow',
    'start_date': datetime(2015, 1, 1),
}

dag = DAG(dag_id='example1')

cmd = 'ls -l'
run_this_last = DummyOperator(
    task_id='run_this_last',
    default_args=args)
dag.add_task(run_this_last)

run_this = BashOperator(
    task_id='run_after_loop', bash_command='echo 1',
    default_args=args)
dag.add_task(run_this)
run_this.set_downstream(run_this_last)
for i in range(9):
    i = str(i)
    task = BashOperator(
```

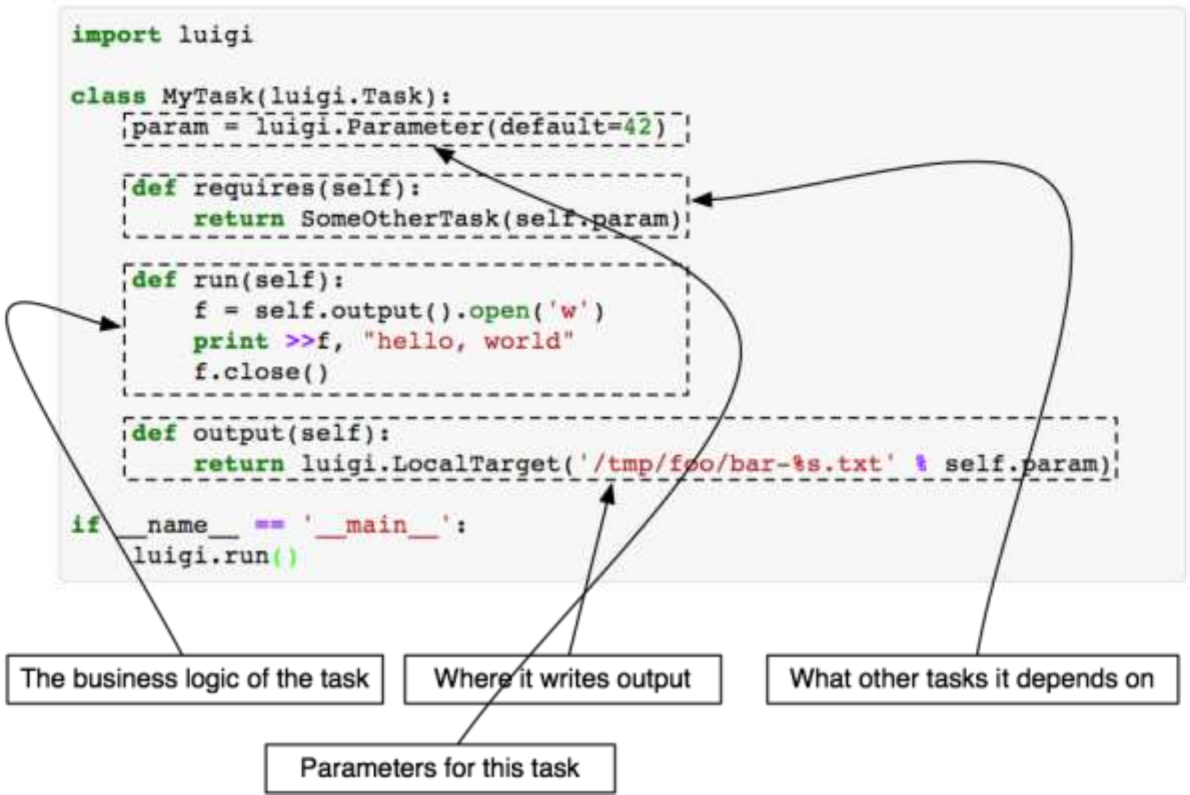
Airflow - Pros

- Dynamic generating path
- Have both Time scheduler and Command line trigger
- Has Master/Worker model (automatically distribute tasks)
- Scale if you have many tasks in a chain.
  - But not be so useful to most of our tasks. Maybe useful for full dump.
- Fancy UI
  - Dependencies of tasks
  - Task success/failure
  - Scheduled tasks status
- Has utility lib to wait_data for S3, Hadoop …

Airflow - Cons

- Additional DB/Redis or Rabbitmq for Celery
  - HA design: Use RDBMS/redis-cache in AWS
- Require python 2.7 and many other libraries.
- Not dependent on data. Just task dependency. (Not big cons)
  - Write check data file code.

A snippet of the task of Luigi



```
import luigi

class MyTask(luigi.Task):
    param = luigi.Parameter(default=42)

    def requires(self):
        return SomeOtherTask(self.param)

    def run(self):
        f = self.output().open('w')
        print >>f, "hello, world"
        f.close()

    def output(self):
        return luigi.LocalTarget('/tmp/foo/bar-%s.txt' % self.param)

if __name__ == '__main__':
    luigi.run()
```

The business logic of the task
Where it writes output
What other tasks it depends on
Parameters for this task

Recap

- Solving DAG job management problem
- Features to improve daily operation
  - Monitor/Visualization
  - Job failure and resume
  - Backfill

Backup slides

Other Selections?

- Oozie (on Hadoop)
- Luigi (by Spotify)
  - Mario (Luigi in Scala)
  - Ruigi (Luigi in R)
- Airflow (by Airbnb)
- Mistral (by OpenStack)
- Pinball (by Pinterest)
- …

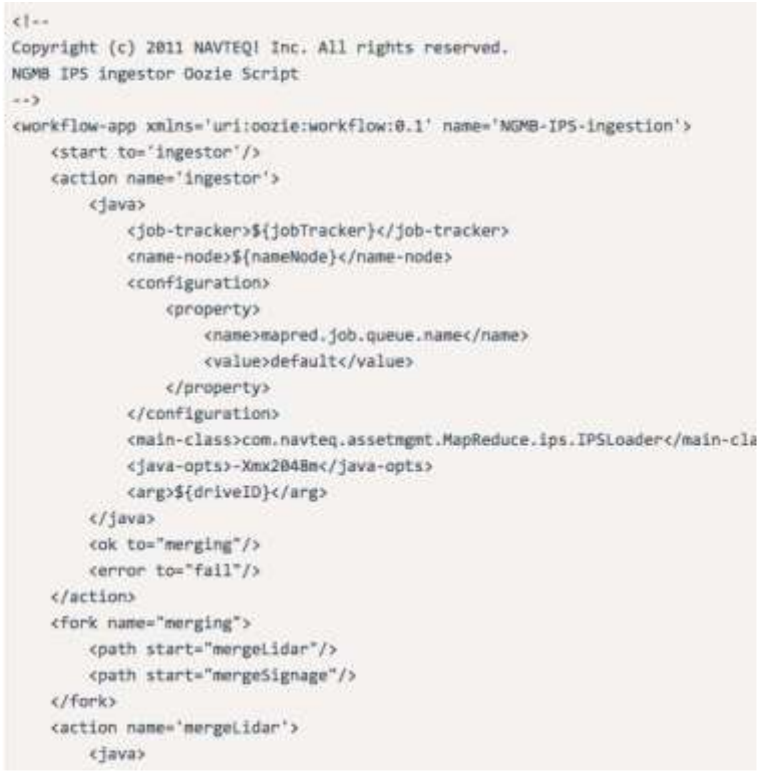
Github statistics

	Start Date	Star	Commits in recent 2 months
Airflow	2014/10/05	1516	362
Luigi	2011/11/13	3777	105
Pinball	2015/05/01	476	0
Oozie	2011/08/28	167	14

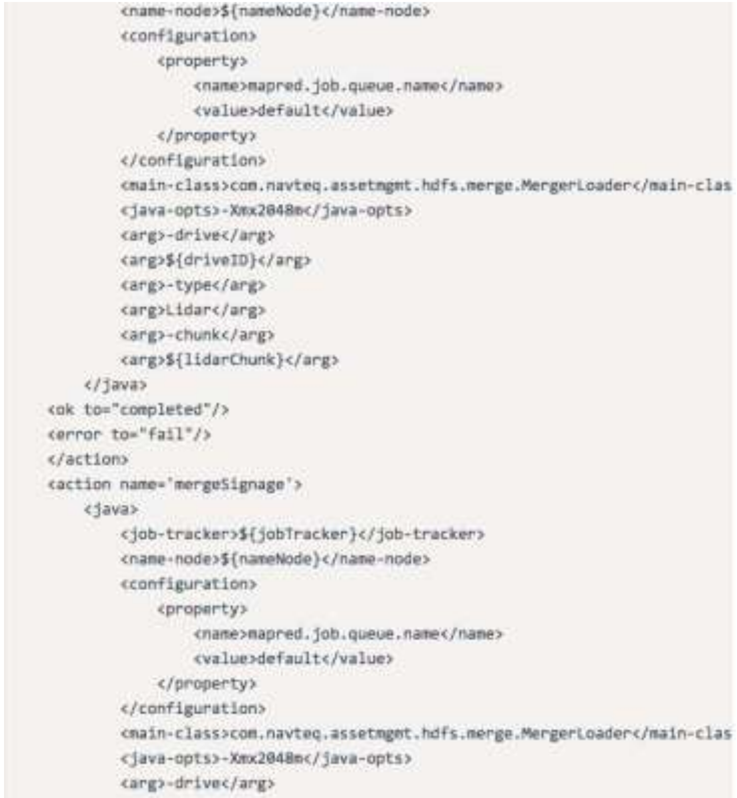
Oozie

- Pros
  - Mature
  - Native support of Hadoop ECO system
  - Python is not required.
- Cons
  - Big and complex XML to define the flow and config.
  - Control flow is somehow restrictive
  - Hadoop ECO system is required.
    - Unix box, Java, Hadoop, Pig, ExtJS library

Oozie XML Example



```
<!--
Copyright (c) 2011 NAVTEQ! Inc. All rights reserved.
NGMB IPS ingestor Oozie Script
-->
<workflow-app xmlns='uri:oozie:workflow:0.1' name='NGMB-IPS-ingestion'>
    <start to='ingestor'/>
    <action name='ingestor'>
        <java>
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>default</value>
                </property>
            </configuration>
            <main-class>com.navteq.assetmgmt.MapReduce.ips.IPSLoader</main-cla
            <java-opts>-Xmx2048m</java-opts>
            <arg>${driveID}</arg>
        </java>
        <ok to="merging"/>
        <error to="fail"/>
    </action>
    <fork name="merging">
        <path start="mergeLidar"/>
        <path start="mergeSignage"/>
    </fork>
    <action name='mergeLidar'>
        <java>
```



```
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>default</value>
                </property>
            </configuration>
            <main-class>com.navteq.assetmgmt.hdfs.merge.MergerLoader</main-clas
            <java-opts>-Xmx2048m</java-opts>
            <arg>-drive</arg>
            <arg>${driveID}</arg>
            <arg>-type</arg>
            <arg>Lidar</arg>
            <arg>-chunk</arg>
            <arg>${lidarChunk}</arg>
        </java>
    <ok to="completed"/>
    <error to="fail"/>
    </action>
    <action name='mergeSignage'>
        <java>
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>default</value>
                </property>
            </configuration>
            <main-class>com.navteq.assetmgmt.hdfs.merge.MergerLoader</main-clas
            <java-opts>-Xmx2048m</java-opts>
            <arg>-drive</arg>
```

Luigi - Pros

- Makefile like, dependencies on data (Upward instead of downward)
- Command line trigger
- Not only Hadoop
- Support target: HDFS/S3/RDBMS/…. (not limited)
  - Write your own code to support Casandra, etc.
- Dependencies are decentralized, no big XML.
- UI
- Dynamic dependency/task is supported. (Don’t know it is better than airflow or not)
- [util] Date algebra

Luigi - Cons

- No built-in triggering
  - No crontab like things (could borrow Chronos)
  - Use cron to run tasks on specific nodes. (normally)

Luigi - Notes

- Local execution
  - Pros: Easy to debug
  - Cons:
    - need access to other system. (Well, other system didn’t achieve this either.)
- No scalability if you have many tasks in a chain.
- Centralized scheduler servers (optional, recommended for production)
  - Make sure two instances of the same task are not running simultaneously
  - Provide visualization of everything that’s going on.
- HA option
  - Run 2+ tasks on 2+ machines at the same time

References

- http://erikbern.com/2015/07/02/more-luigi-alternatives/