

# Fuss Free ETL

Tame your data pipelines with Airflow

Vijay Bhat



@vijaysbhat



/in/vijaysbhat

# About Me

## 13 Years In The Industry



Mobile



Financial Services



Smart Meter Analytics



Social Media

## Data Science Applications



Forecasting



Recommendation Systems



Fraud Detection



Growth Analytics

## For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights

By STEVE LOHR AUG. 17, 2014



Monica Rogati, Jawbone's vice president for data science, with Brian Wilt, a senior data scientist. Peter DaSilva for The New York Times

Email

Share

Tweet

Save

More

Technology revolutions come in measured, sometimes foot-dragging steps. The lab science and marketing enthusiasm tend to underestimate the bottlenecks to progress that must be overcome with hard work and practical engineering.

The field known as "big data" offers a contemporary case study. The catchphrase stands for the modern abundance of digital data from many sources — the web, sensors, smartphones and corporate databases — that can be mined with clever software for discoveries and insights. Its promise is smarter, data-driven decision-making in every field. That is why data scientist is the economy's hot new job.

Yet far too much handcrafted work — what data scientists call "data wrangling," "data munging" and "data janitor work" — is still required. Data scientists, according to interviews and expert estimates, spend from 50 percent to 80 percent of their time mired in this more mundane labor of collecting and preparing unruly digital data, before it can be explored for useful nuggets.

# 80%

# What



# How



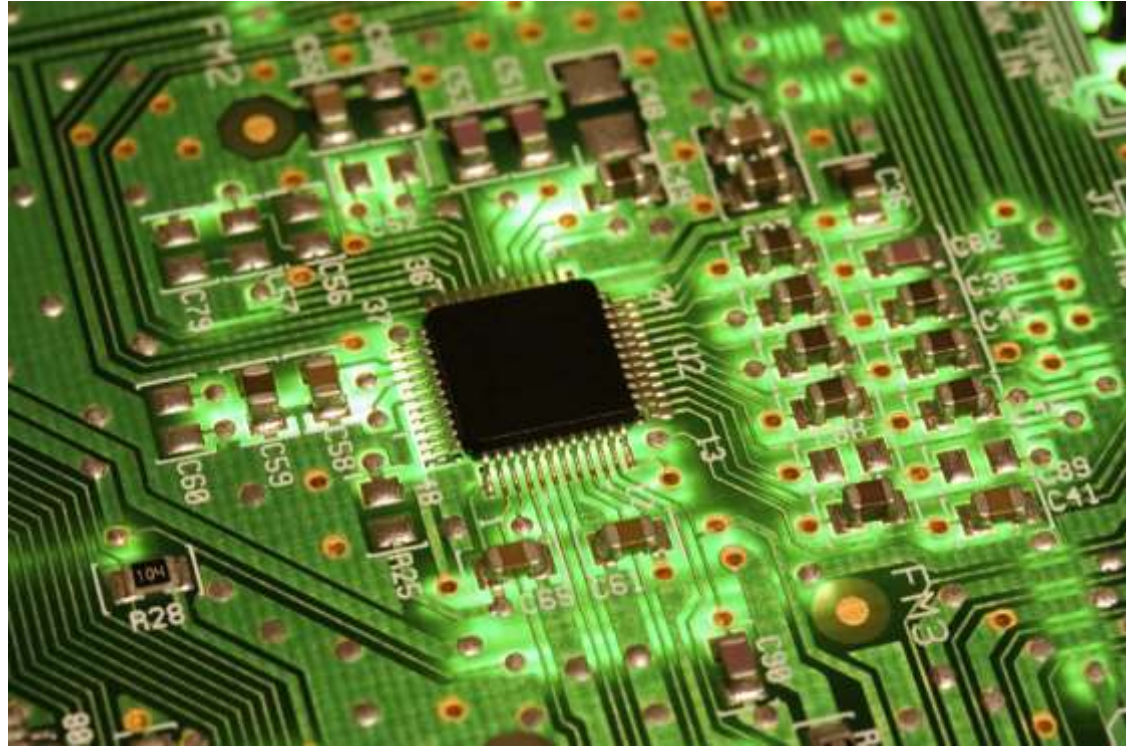
# When



What does your  
ETL process look  
like?



What do we do to  
get here?



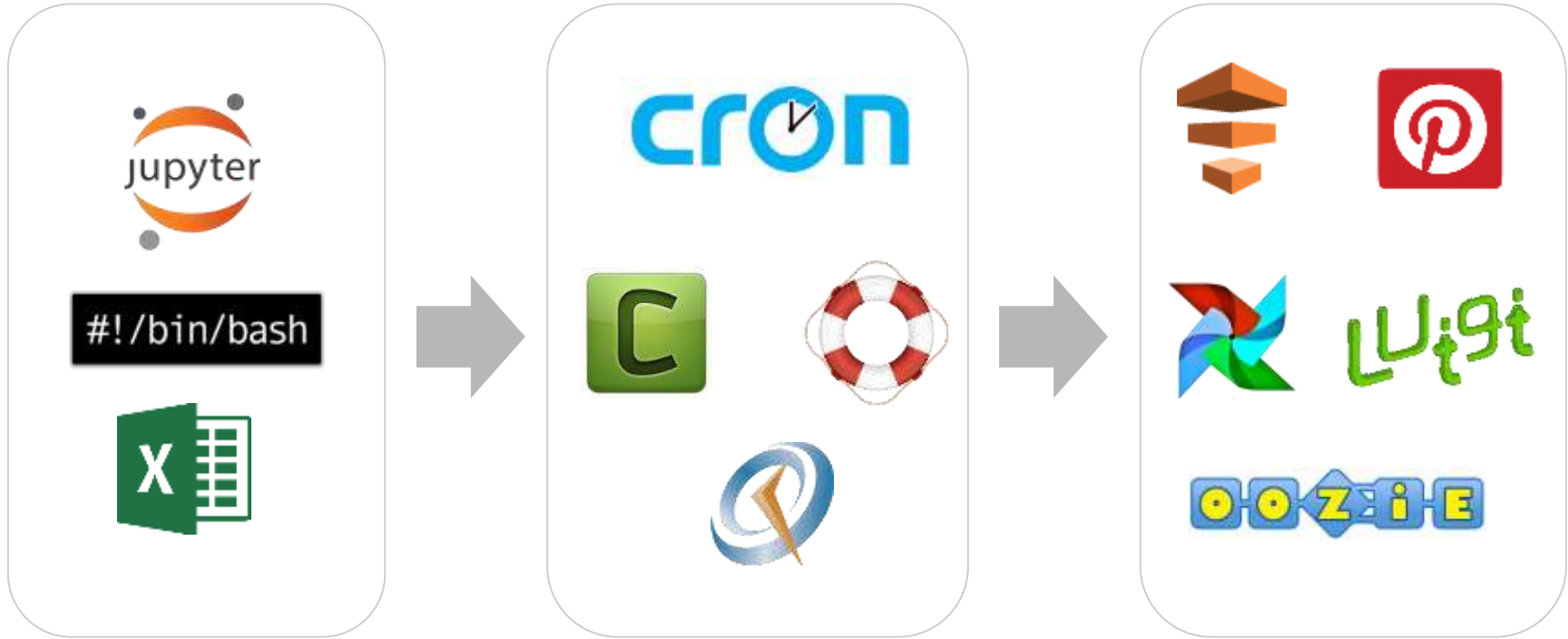
# BEST PRACTiCE



- ❑ Automation
- ❑ Scheduling
- ❑ Version Control
- ❑ Redundancy
- ❑ Error Recovery
- ❑ Monitoring



# Evolution





# Introducing Airflow



- Open source ETL workflow engine
- Developed by Airbnb
- Inspired by Facebook's Dataswarm
- Production ready
- Pipelines written in Python



@vijaysbhat



/in/vijaysbhat

# Defining Pipelines

AirFlow DAGs Tools Browse Admin Docs

## DAG: example1

Tree View Graph View Task Duration Landing Times Gantt Code

### example\_dags/example1.py

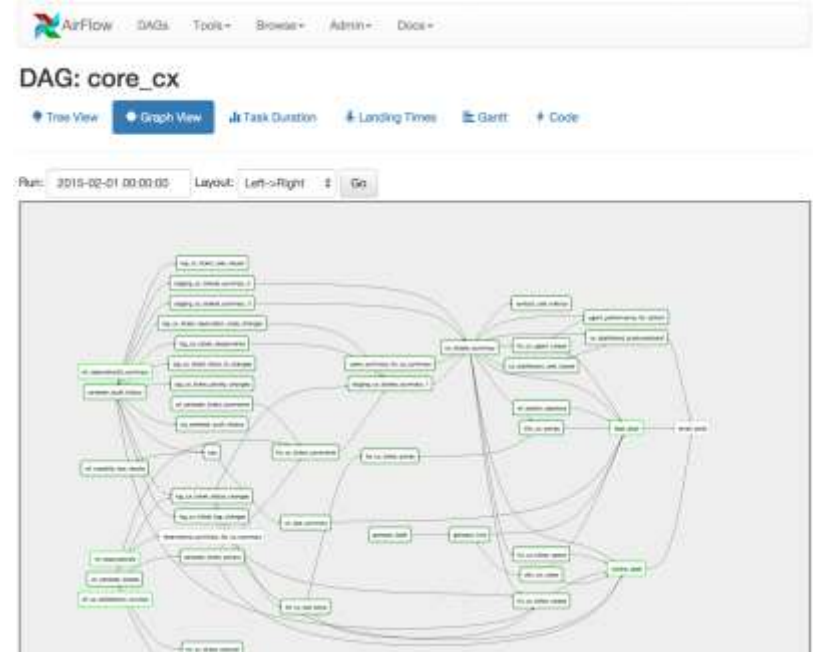
```
from airflow.operators import BashOperator, DummyOperator
from airflow.models import DAG
from datetime import datetime

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(
```



# Pipeline Code Structure

```
from datetime import datetime, timedelta

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),
}
```

Define default arguments



@vijaysbhat



/in/vijaysbhat

# Pipeline Code Structure

```
dag = DAG(  
    'tutorial', default_args=default_args, schedule_interval=timedelta(1))
```

Instantiate DAG

# Pipeline Code Structure

```
t1 = BashOperator(  
    task_id='print_date',  
    bash_command='date',  
    dag=dag)  
  
t2 = BashOperator(  
    task_id='sleep',  
    bash_command='sleep 5',  
    retries=3,  
    dag=dag)
```

Define tasks

# Pipeline Code Structure

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

```
# This means that t2 will depend on t1
```

```
# running successfully to run
```

```
# It is equivalent to
```

```
# t1.set_downstream(t2)
```

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

```
# all of this is equivalent to
```

```
# dag.set_dependency('print_date', 'sleep')
```

```
# dag.set_dependency('print_date', 'templated')
```

Chain tasks

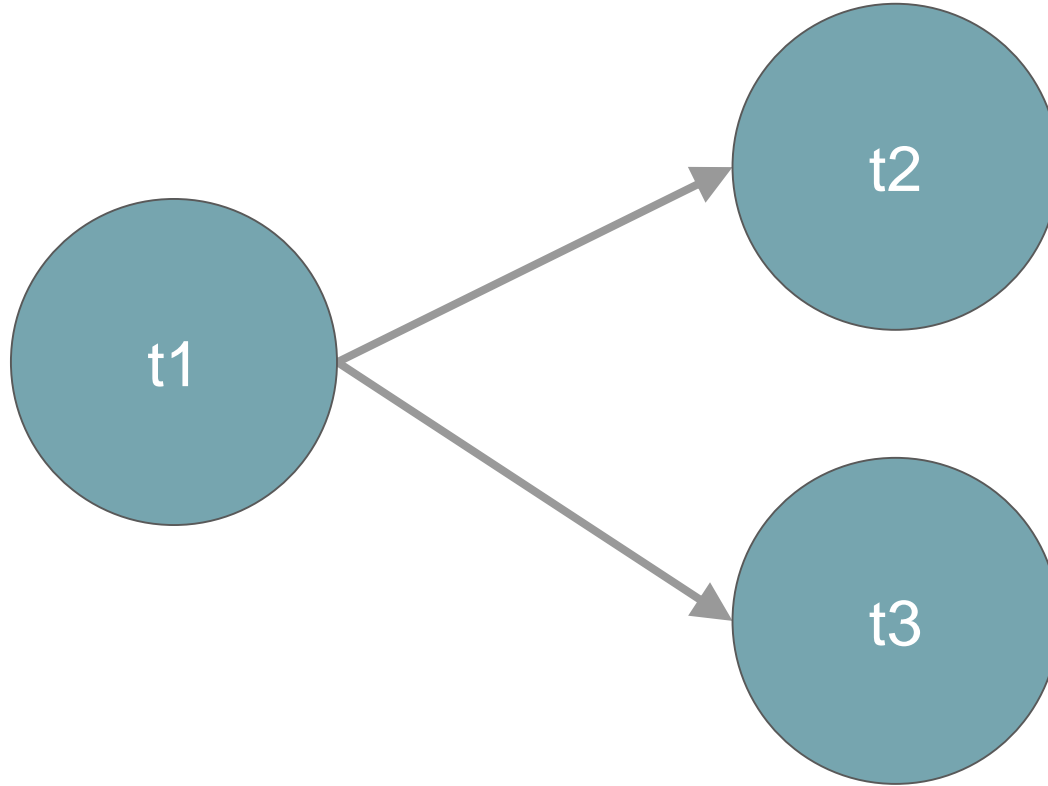


@vijaysbhat



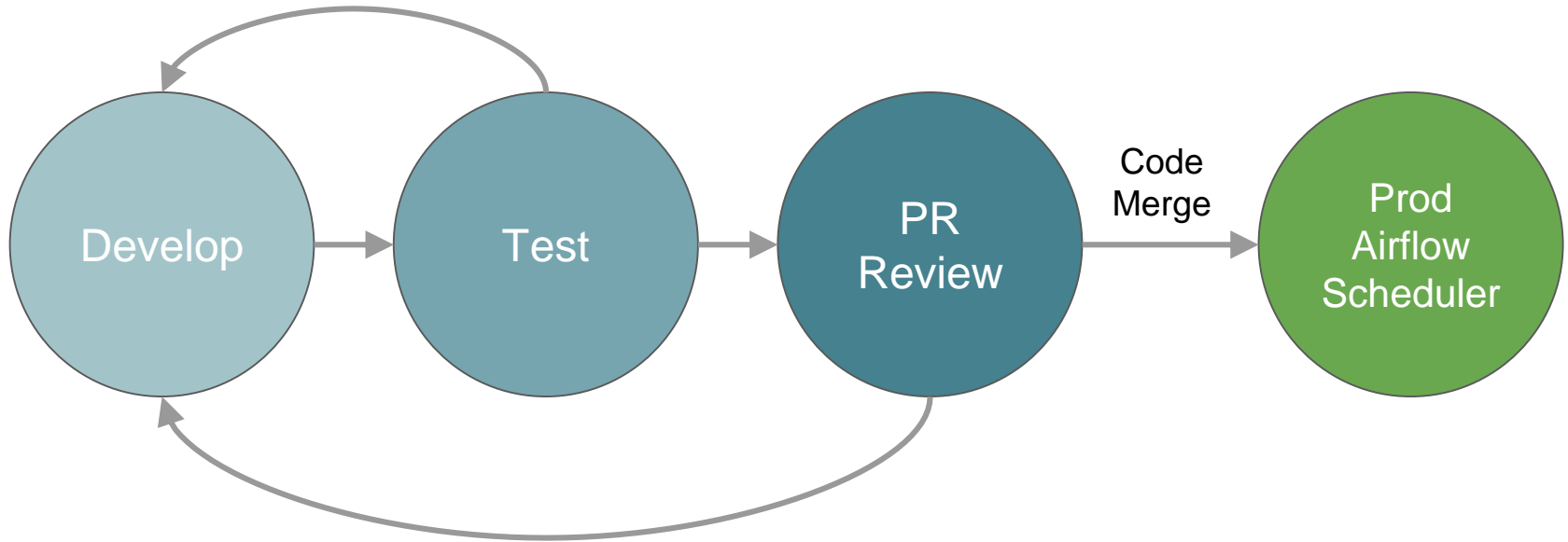
/in/vijaysbhat

Then we get this pipeline

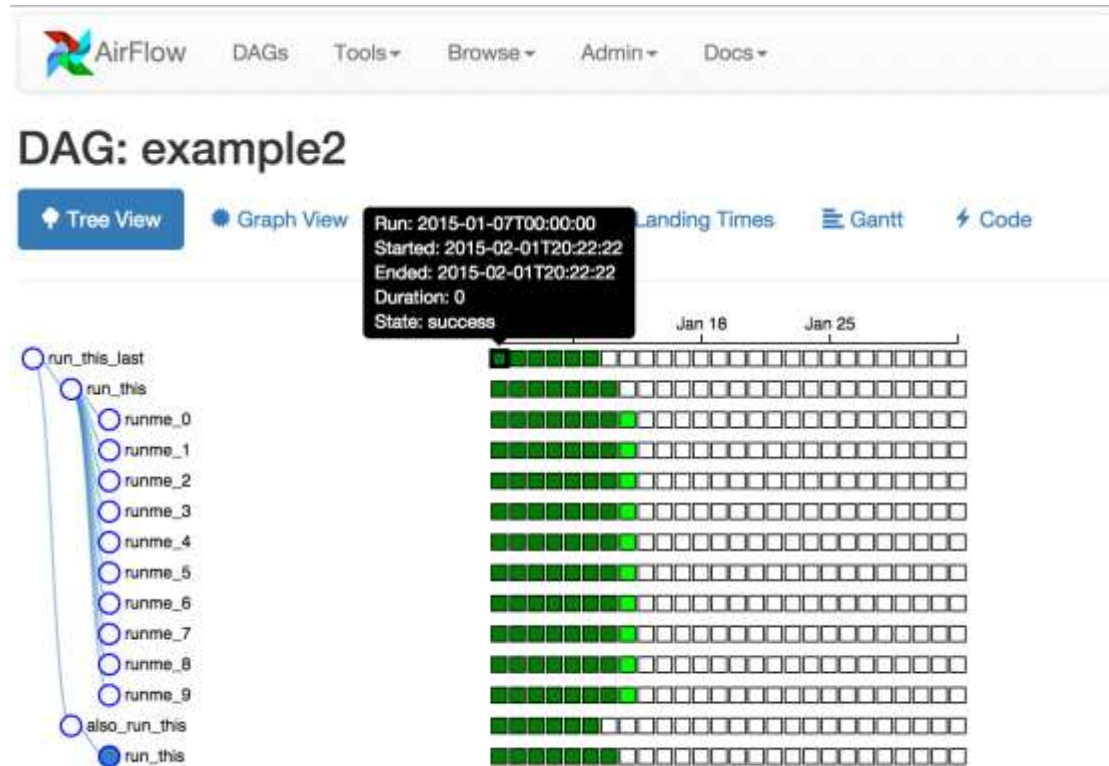




# Deployment Process



# Job Runs



# Logs

**DAG: hacker\_news\_sentiment\_dag** scheduler: @daily

● Graph View ● Tree View ▴ Task Duration ⬆ Landing Times ⚙ Garb ⚙ Details ⚡ Code

Task Instance: **watson\_sentiment\_analysis** 2016-06-20 00:00:00

● Task Details ● Rendered Template ● **Log** ● XCom

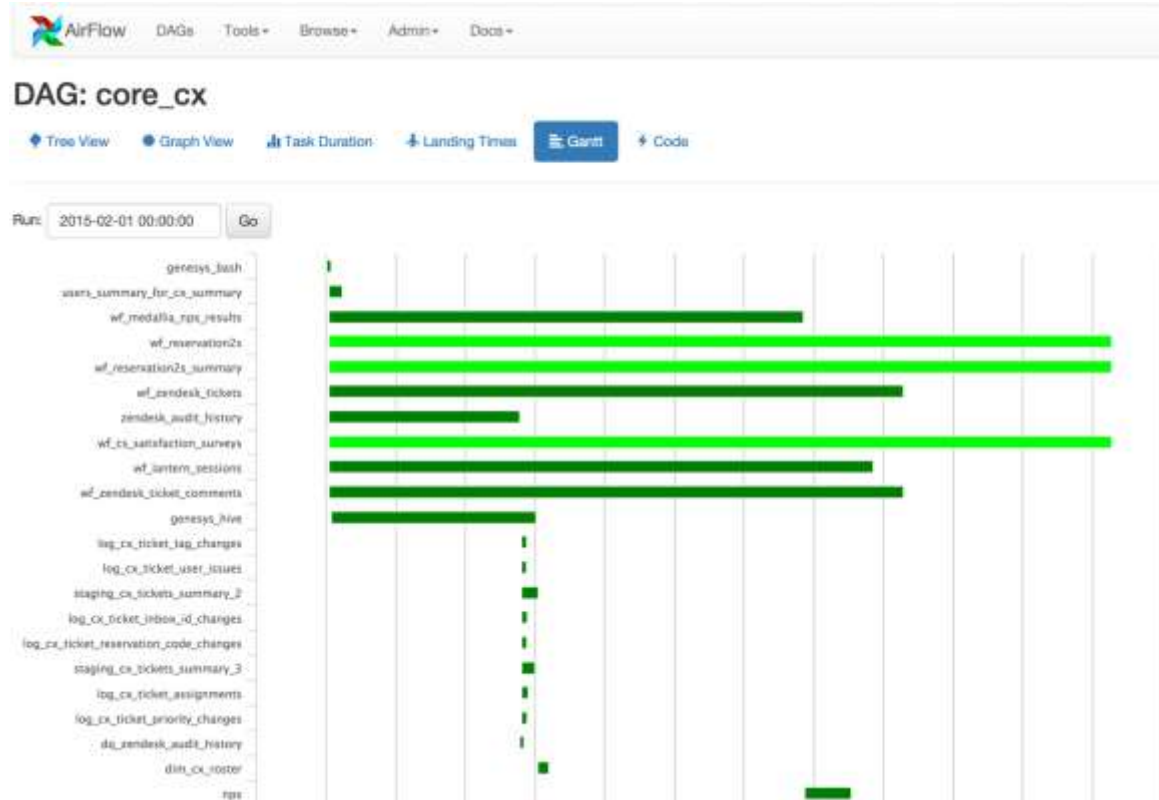
### Log

```
[2016-06-21 00:00:30,145] {models.py:154} INFO - Filling up the DagBag from /home/ubuntu/airflow/dags/hacker_news_sentiment_dag.py
[2016-06-21 00:00:30,905] {models.py:154} INFO - Filling up the DagBag from /home/ubuntu/airflow/dags/hacker_news_sentiment_dag.py
[2016-06-21 00:00:31,124] {models.py:1196} INFO -

Starting attempt 1 of 1

[2016-06-21 00:00:31,139] {models.py:1219} INFO - Executing <Task(PythonOperator): watson_sentiment_analysis> on 2016-06-20 00:00:00
[2016-06-21 00:00:31,184] {connectionpool.py:213} INFO - Starting new HTTP connection (1): access.alchemyapi.com
[2016-06-21 00:00:31,564] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:31,674] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:31,943] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:32,054] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:32,163] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:32,275] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:32,384] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:38,697] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:38,699] {connectionpool.py:240} INFO - Resetting dropped connection: access.alchemyapi.com
[2016-06-21 00:00:38,905] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:39,151] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:39,257] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'ERROR', 'usage': 'By accessing AlchemyAPI or using informatio
[2016-06-21 00:00:39,364] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:39,473] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:39,581] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:39,695] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
[2016-06-21 00:00:39,954] {hacker_news_sentiment_dag.py:80} INFO - {'status': 'OK', 'usage': 'By accessing AlchemyAPI or using information g
```

# Performance - Gantt Chart



# Operators

## Action

- PythonOperator
- HiveOperator
- ...

## Transfer

- S3ToHiveTransfer
- HiveToDruidTransfer
- ...

## Sensor

- HdfsSensor
- HivePartitionSensor
- ...

# Useful Configuration Options

- `depends_on_past`
  - wait until task run for previous day is complete?
- `wait_for_downstream`
  - dependency on downstream tasks for previous day.
- `sla`
  - send email alerts if SLA is missed.



@vijaysbhat



/in/vijaysbhat

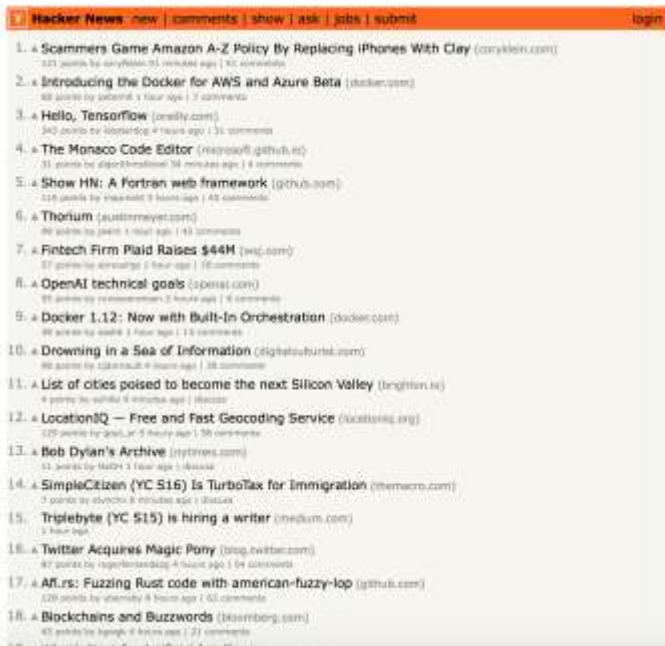
# CLI Commands

- airflow [-h]
  - webserver
  - scheduler
  - test
  - run
  - backfill
  - ...

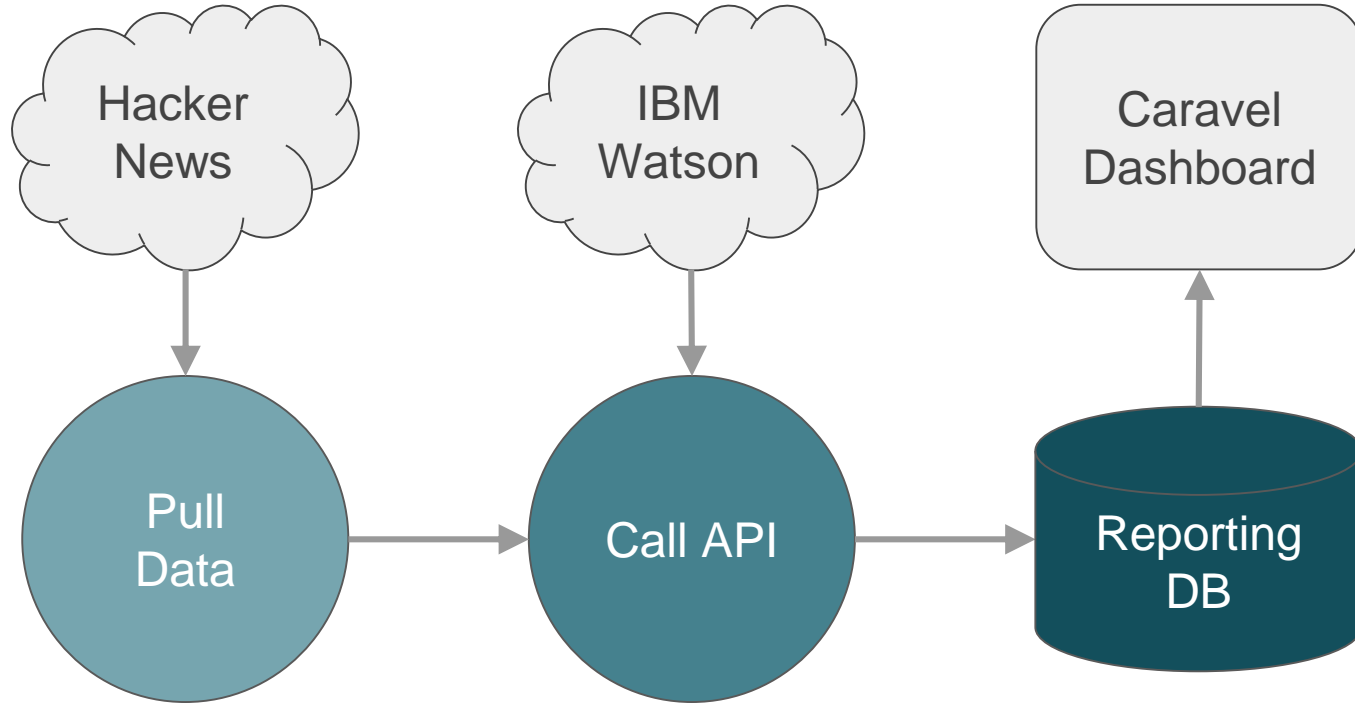
```
vijay — ubuntu@ip-172-31-10-255: ~/airflow — ssh — 80x24
[2016-06-22 23:28:17,730] {models.py:305} INFO - Finding 'running' jobs without
a recent heartbeat
[2016-06-22 23:28:17,731] {models.py:311} INFO - Failing jobs without heartbeat
after 2016-06-22 23:26:02.731208
[2016-06-22 23:28:22,667] {jobs.py:574} INFO - Prioritizing 0 queued jobs
[2016-06-22 23:28:22,673] {jobs.py:726} INFO - Starting 1 scheduler jobs
/usr/local/lib/python2.7/dist-packages/sqlalchemy/sql/default_comparator.py:153:
SAWarning: The IN-predicate on "task_instance.execution_date" was invoked with
an empty sequence. This results in a contradiction, which nonetheless can be exp
ensive to evaluate. Consider alternative strategies for improved performance.
'strategies for improved performance.' % expr)
[2016-06-22 23:28:22,705] {jobs.py:498} INFO - Getting list of tasks to skip for
active runs.
[2016-06-22 23:28:22,705] {jobs.py:514} INFO - Checking dependencies on 0 tasks
instances, minus 0 skippable ones
[2016-06-22 23:28:22,726] {jobs.py:741} INFO - Done queuing tasks, calling the e
xecutor's heartbeat
[2016-06-22 23:28:22,726] {jobs.py:744} INFO - Loop took: 0.061736 seconds
[2016-06-22 23:28:22,735] {models.py:305} INFO - Finding 'running' jobs without
a recent heartbeat
[2016-06-22 23:28:22,735] {models.py:311} INFO - Failing jobs without heartbeat
after 2016-06-22 23:26:07.735932
```



# Example: Hacker News Sentiment Tracker



# Hacker News Example: Data Flow



# Hacker News Example: Demo

pip install airflow!

# Thank You.

@vijaysbhat 

/in/vijaysbhat 

vijay@innosparkdata.com 