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

@vijaysbhat
/in/vijaysbhat

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%

@vijaysbhat
/in/vijaysbhat

What

How

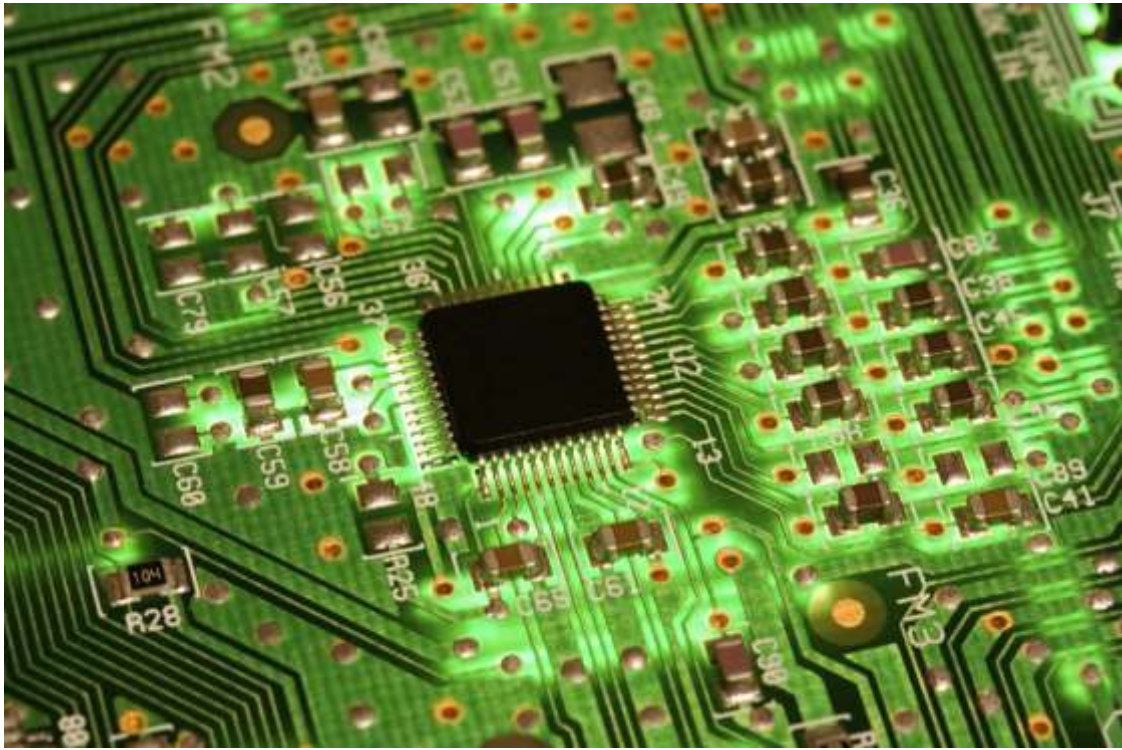
When

@vijaysbhat
/in/vijaysbhat

What does your ETL process look like?

@vijaysbhat
/in/vijaysbhat

What do we do to get here?



@vijaysbhat
/in/vijaysbhat

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

@vijaysbhat
/in/vijaysbhat

Evolution

@vijaysbhat
/in/vijaysbhat

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

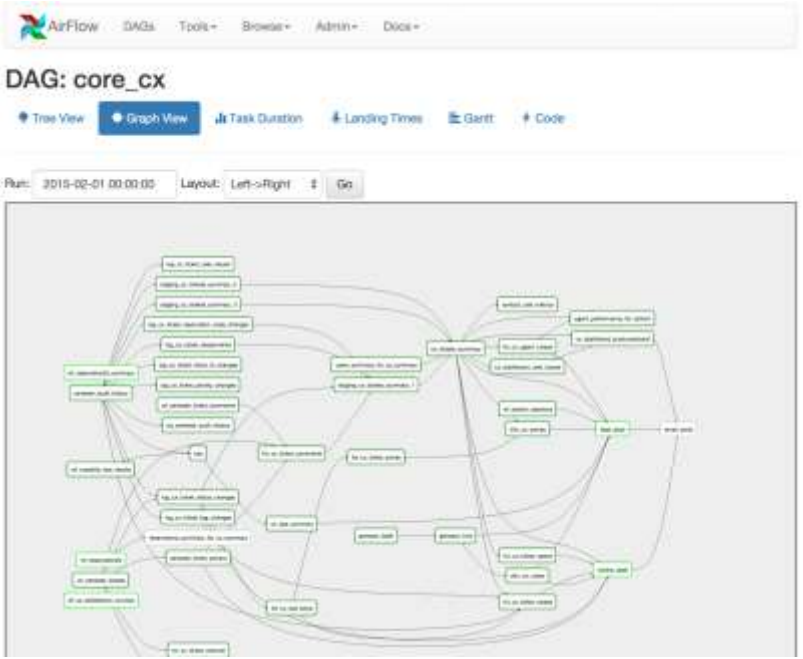
AirFlow DAGs Tools Browse Admin Docs

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(
```



@vijaysbhat
/in/vijaysbhat

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

@vijaysbhat
/in/vijaysbhat

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

@vijaysbhat
/in/vijaysbhat

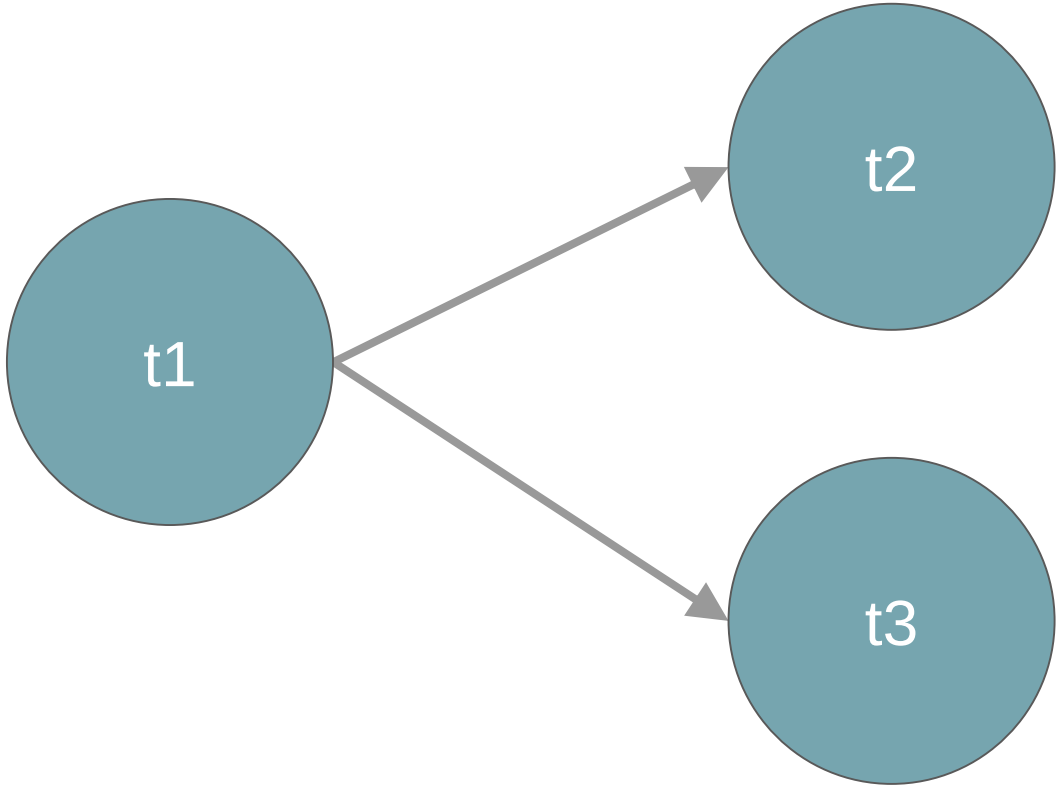
Pipeline Code Structure

```
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

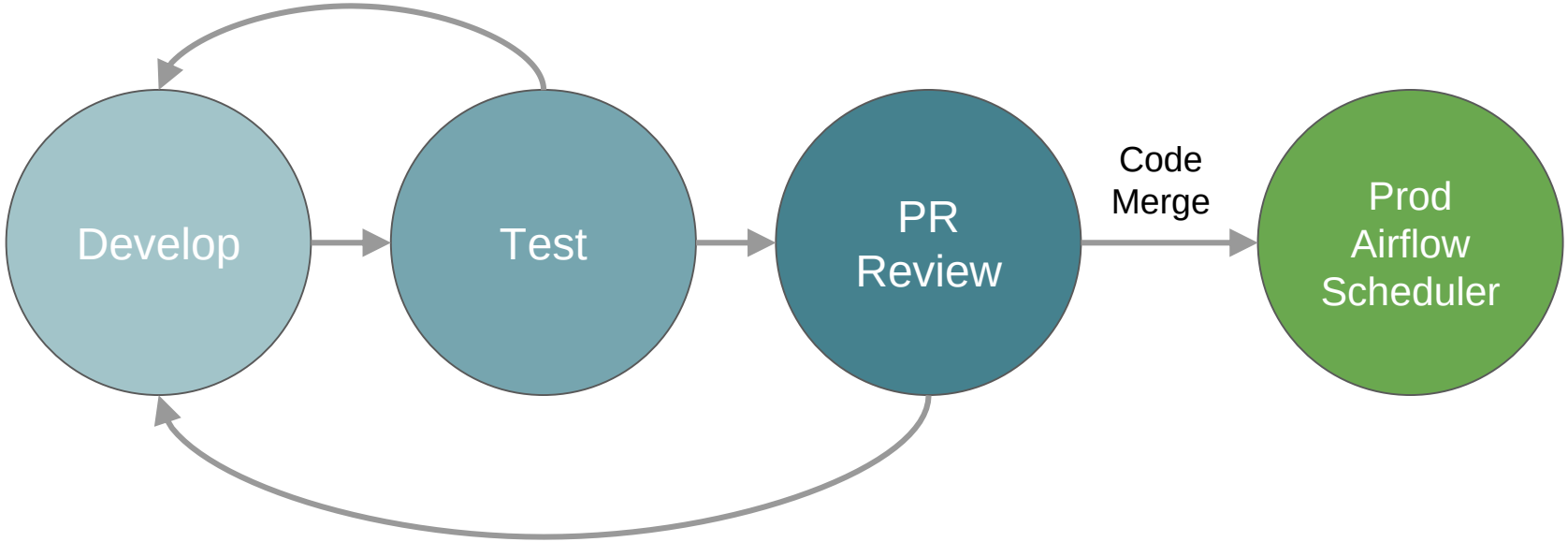
@vijaysbhat
/in/vijaysbhat

Then we get this pipeline



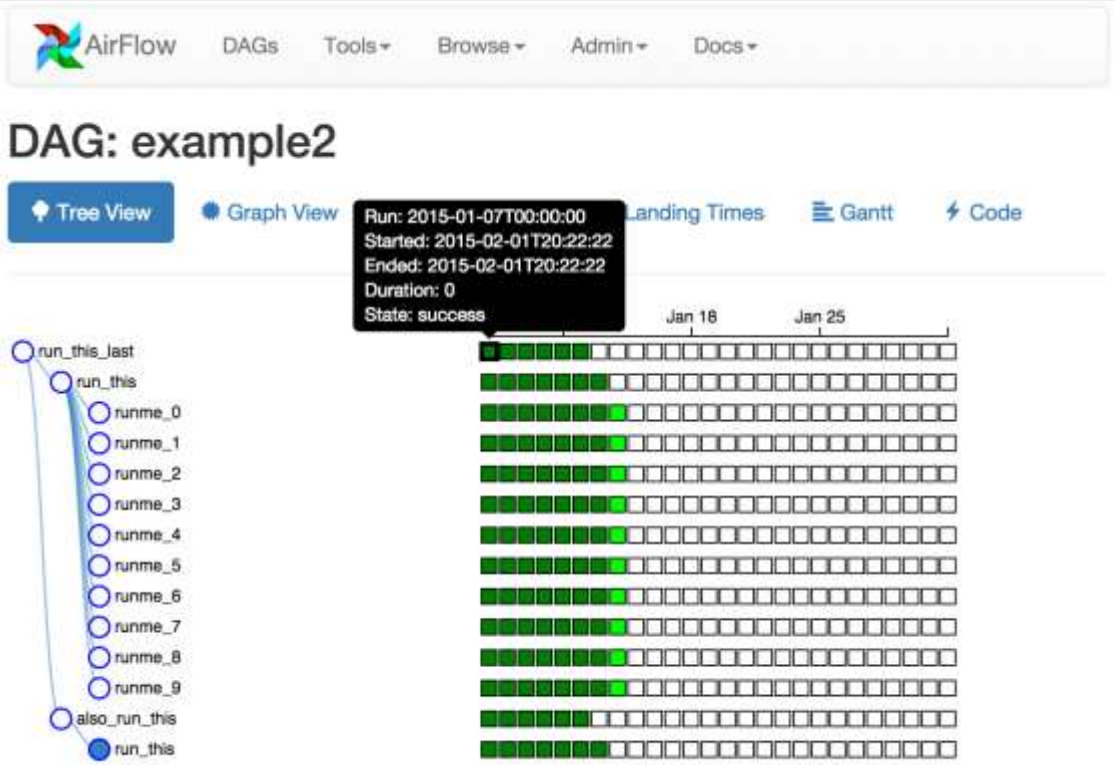
@vijaysbhat
/in/vijaysbhat

Deployment Process



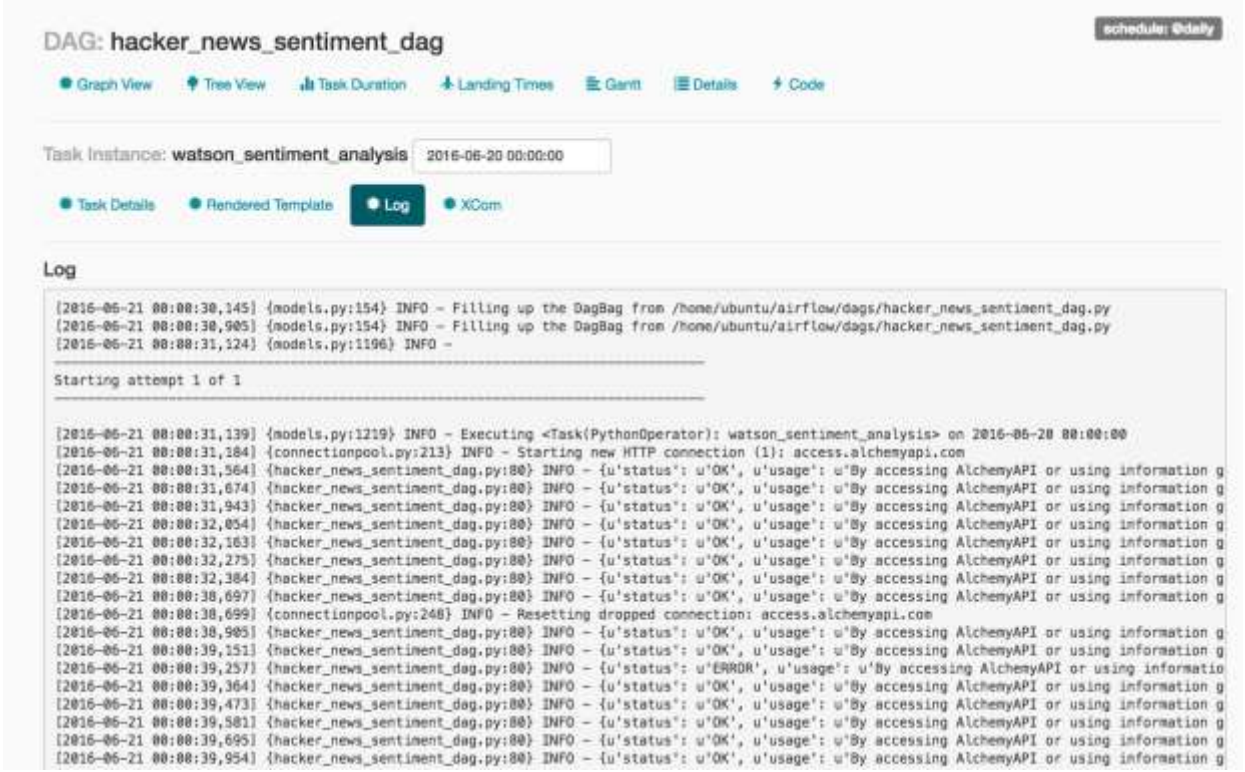
@vijaysbhat
/in/vijaysbhat

Job Runs



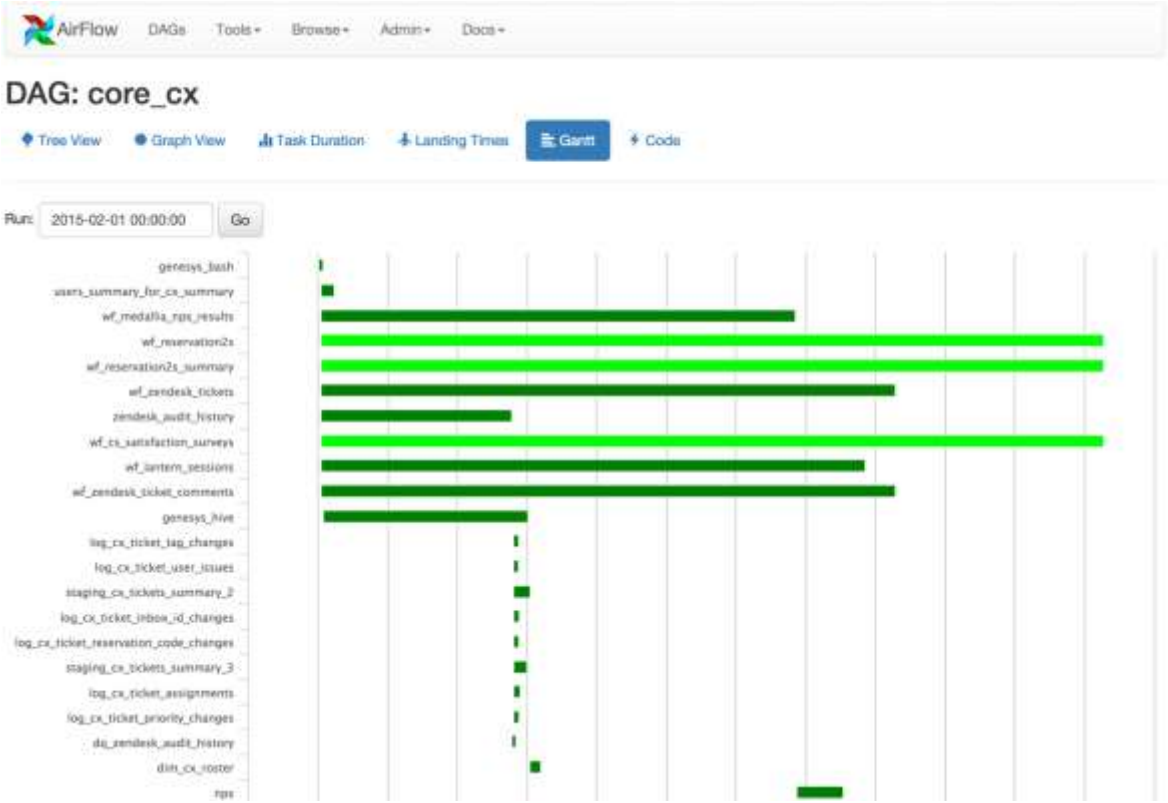
@vijaysbhat
/in/vijaysbhat

Logs



@vijaysbhat
/in/vijaysbhat

Performance - Gantt Chart



@vijaysbhat
/in/vijaysbhat

Operators

Action

- PythonOperator
- HiveOperator
- ...

Transfer

- S3ToHiveTransfer
- HiveToDruidTransfer
- ...

Sensor

- HdfsSensor
- HivePartitionSensor
- ...

@vijaysbhat
/in/vijaysbhat

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

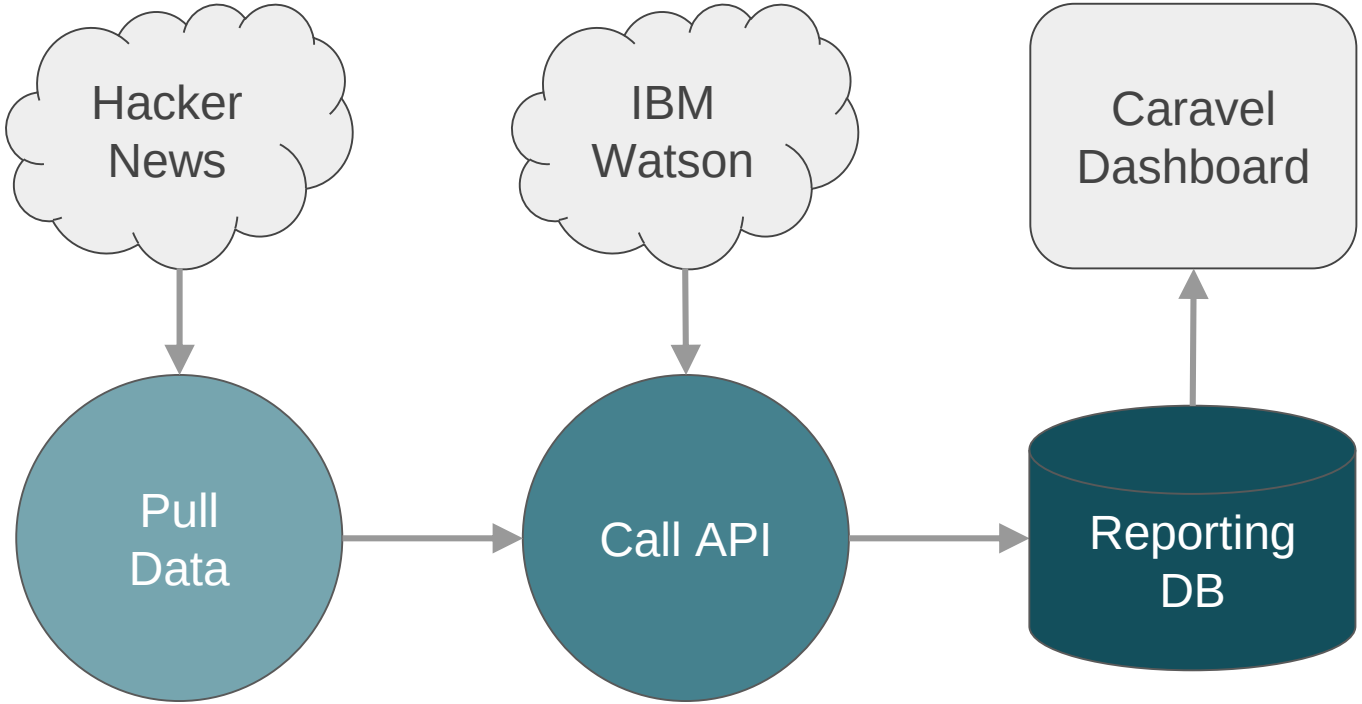
- airflow [-h]
  - webserver
  - scheduler
  - test
  - run
  - backfill
  - ...

@vijaysbhat
/in/vijaysbhat

Example: Hacker News Sentiment Tracker

@vijaysbhat
/in/vijaysbhat

Hacker News Example: Data Flow



@vijaysbhat
/in/vijaysbhat

Hacker News Example: Demo

pip install airflow!

@vijaysbhat
/in/vijaysbhat

Thank You.

@vijaysbhat 
/in/vijaysbhat 
vijay@innosparkdata.com