Airflow + Kubernetes

Daniel Imberman, Bloomberg
Barni Seetharaman, Google

Bio: Daniel + Barni

Raw Data ⟶ **Actionable Data**

Pipelines are hard

Pipelines are hard

Raw Data  **Actionable Data**

Pipelines are hard

Raw Data → 

→ **Actionable Data**

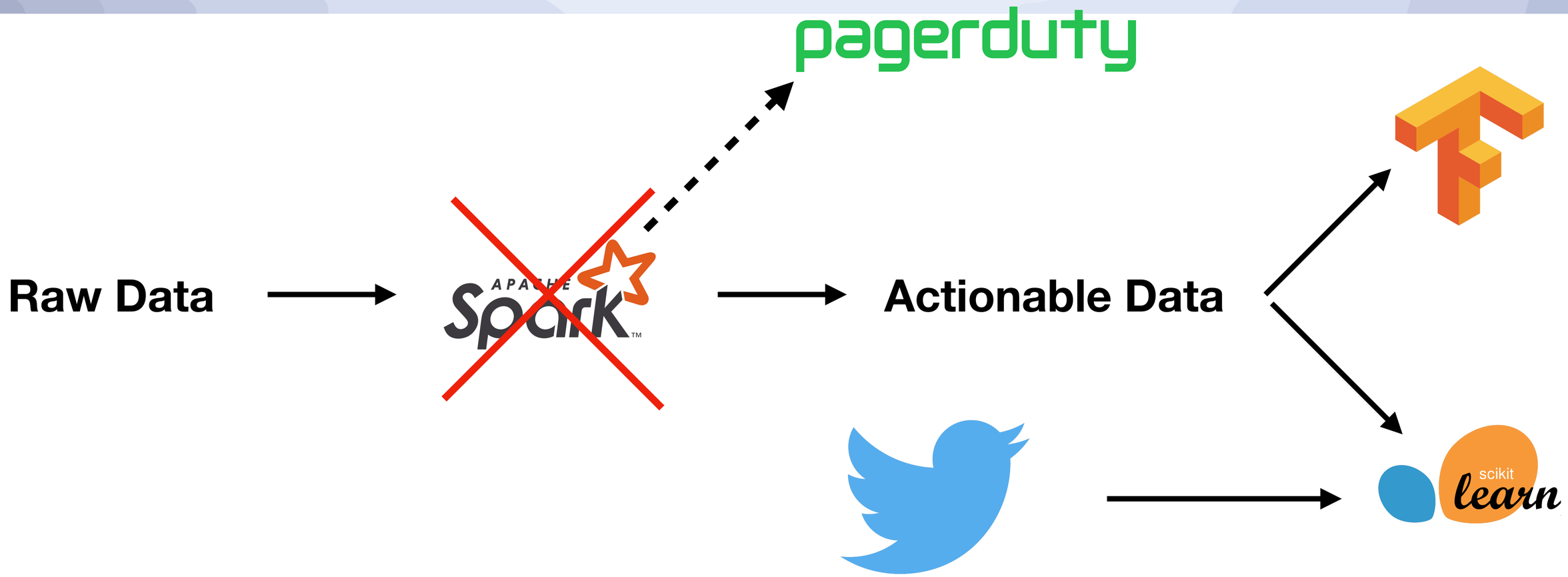
Pipelines are hard

Raw Data → 

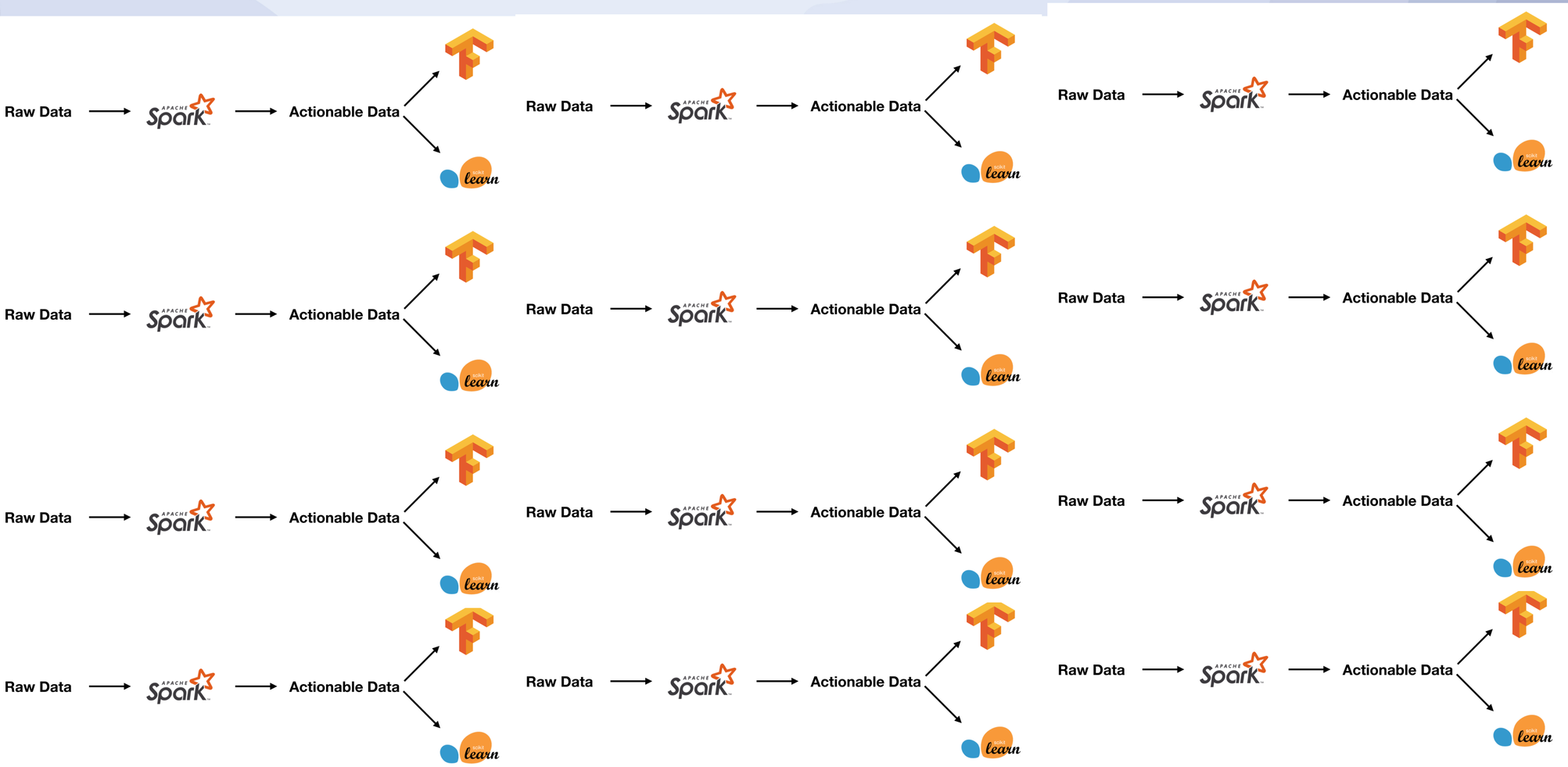
→ **Actionable Data** → 

Pipelines are hard

Pipelines are hard



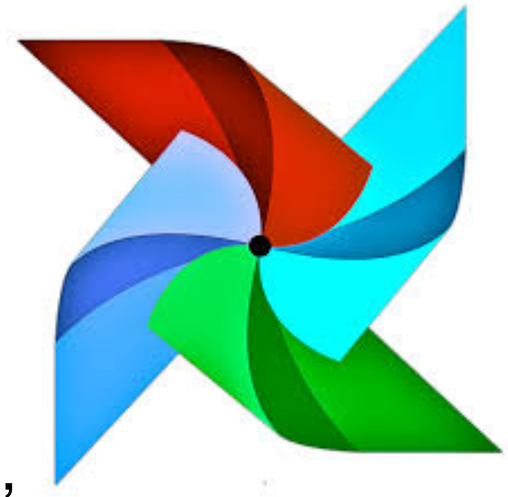
Lots of pipelines are really hard



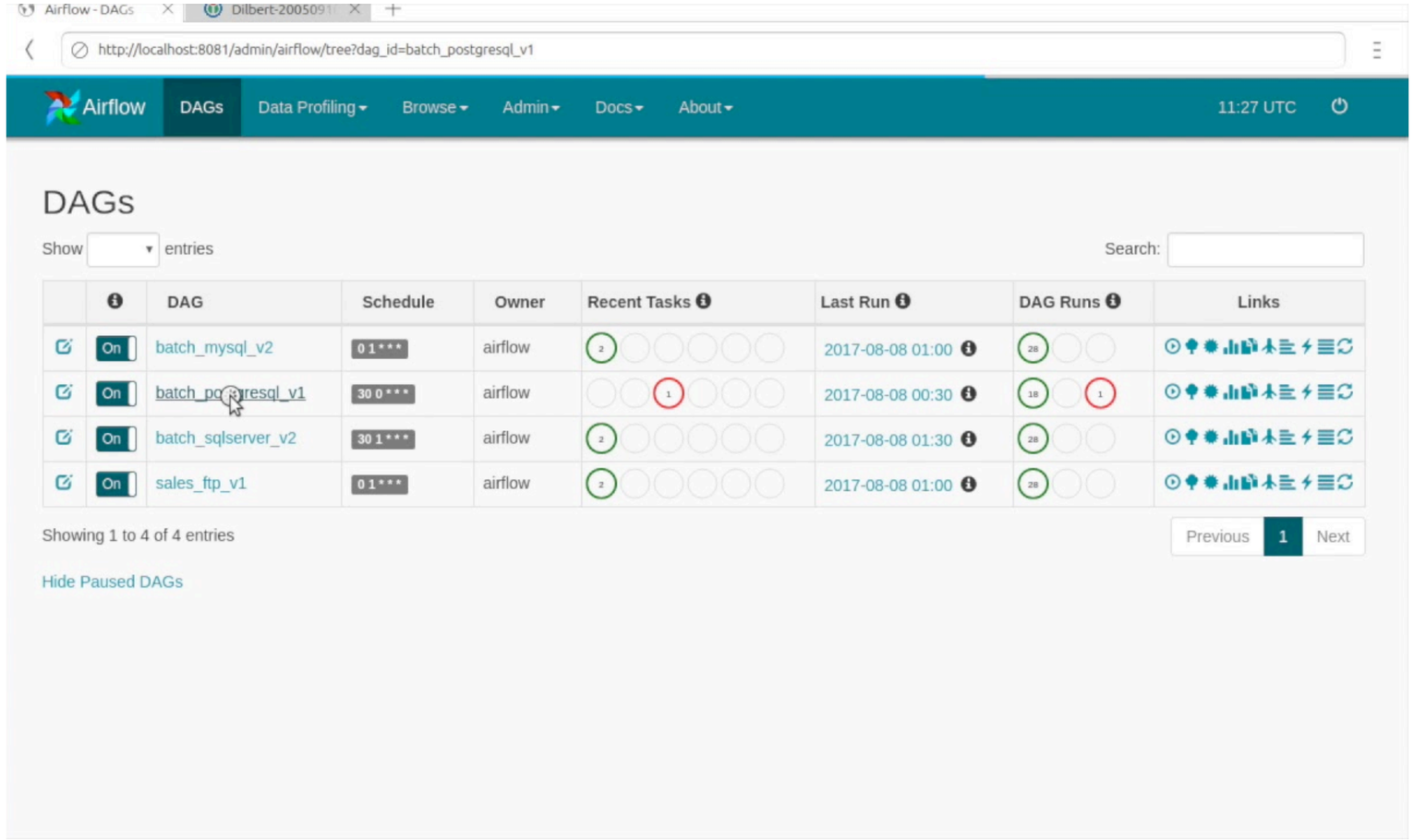
Enter Apache Airflow

Apache Airflow

- Workflow Scheduler developed @ Airbnb
- Converts Python code into DAGs
- Has large number of operators/hooks (HDFS, Spark, Bash, Hive, etc…)

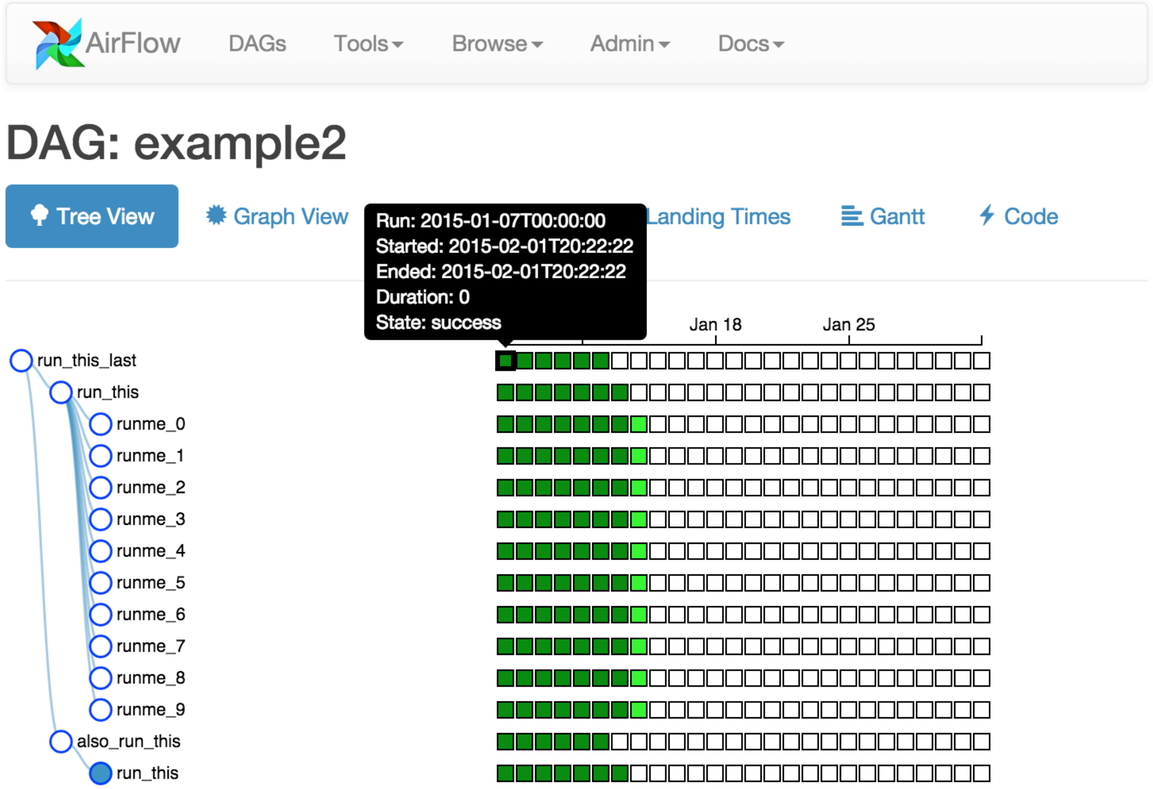


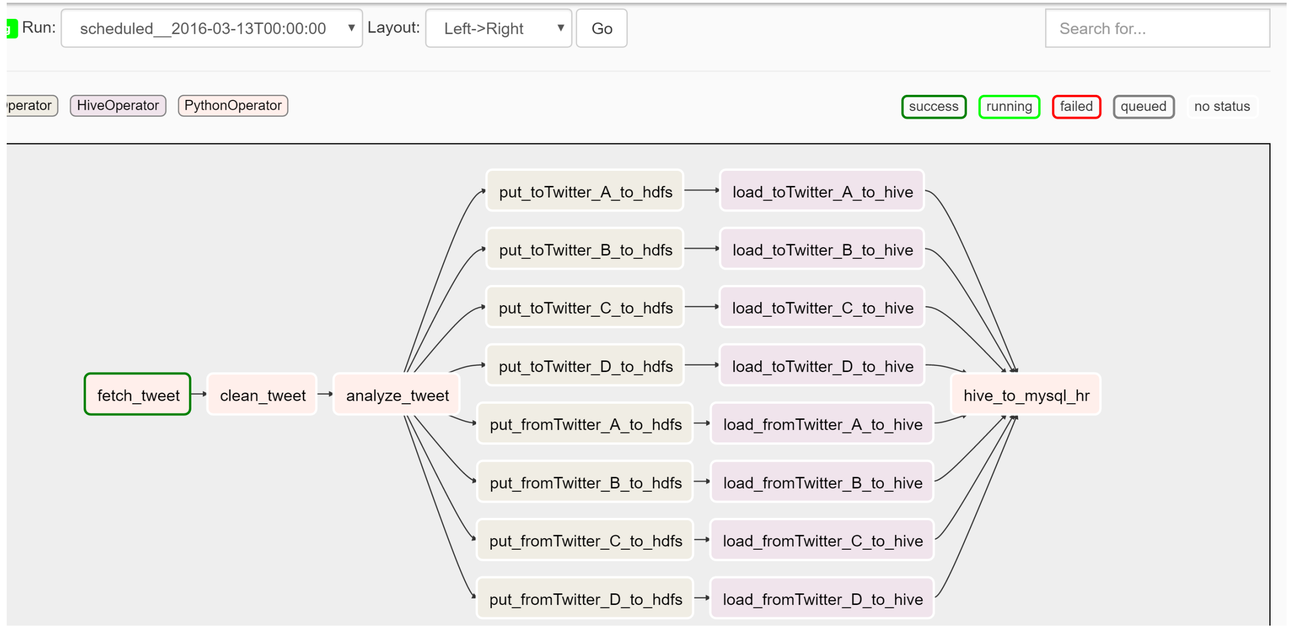
Apache Airflow



Apache Airflow

KubeCon | CloudNativeCon
North America 2018





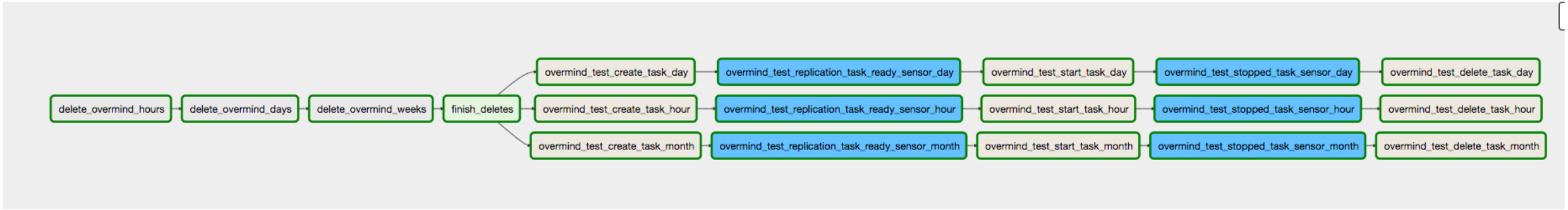
Creating a pipeline with Airflow


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

# t1, t2 and t3 are examples of tasks created by instantiating 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)
```

Define Operators

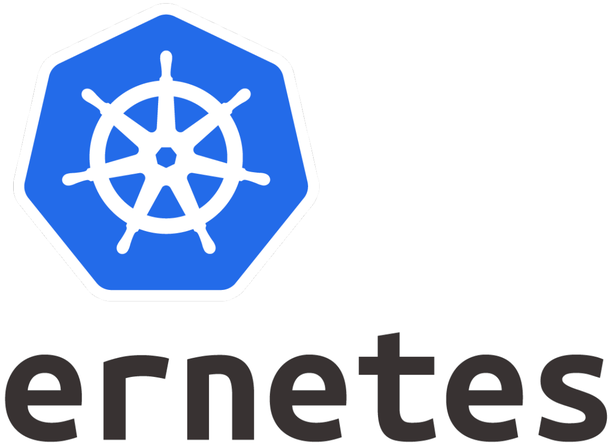
Set Dependencies

Managing a pipeline with Airflow



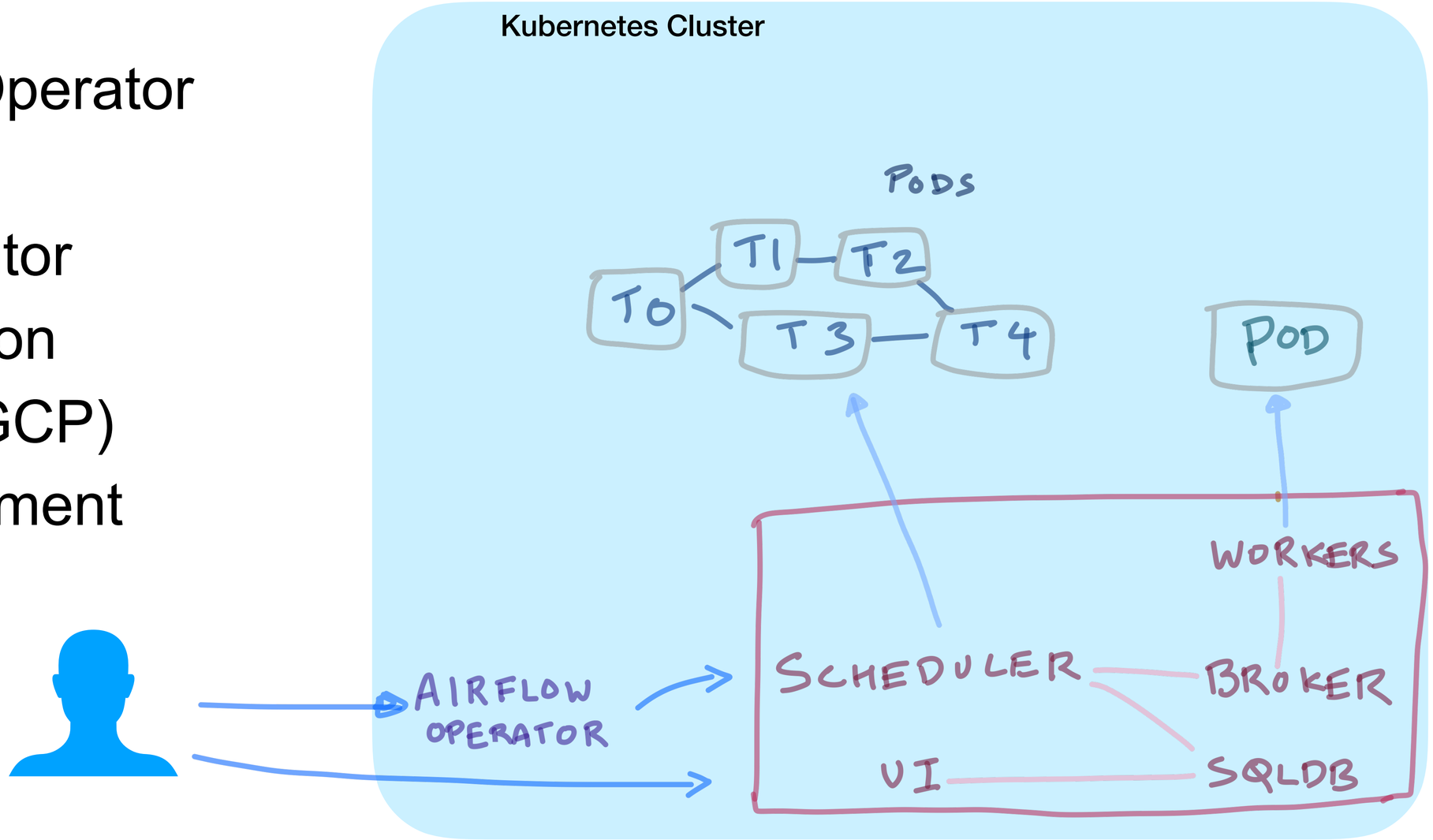
Why Airflow on Kubernetes?

- More flexibility
- Dynamic Resource Allocation
- Lower Barrier to Entry



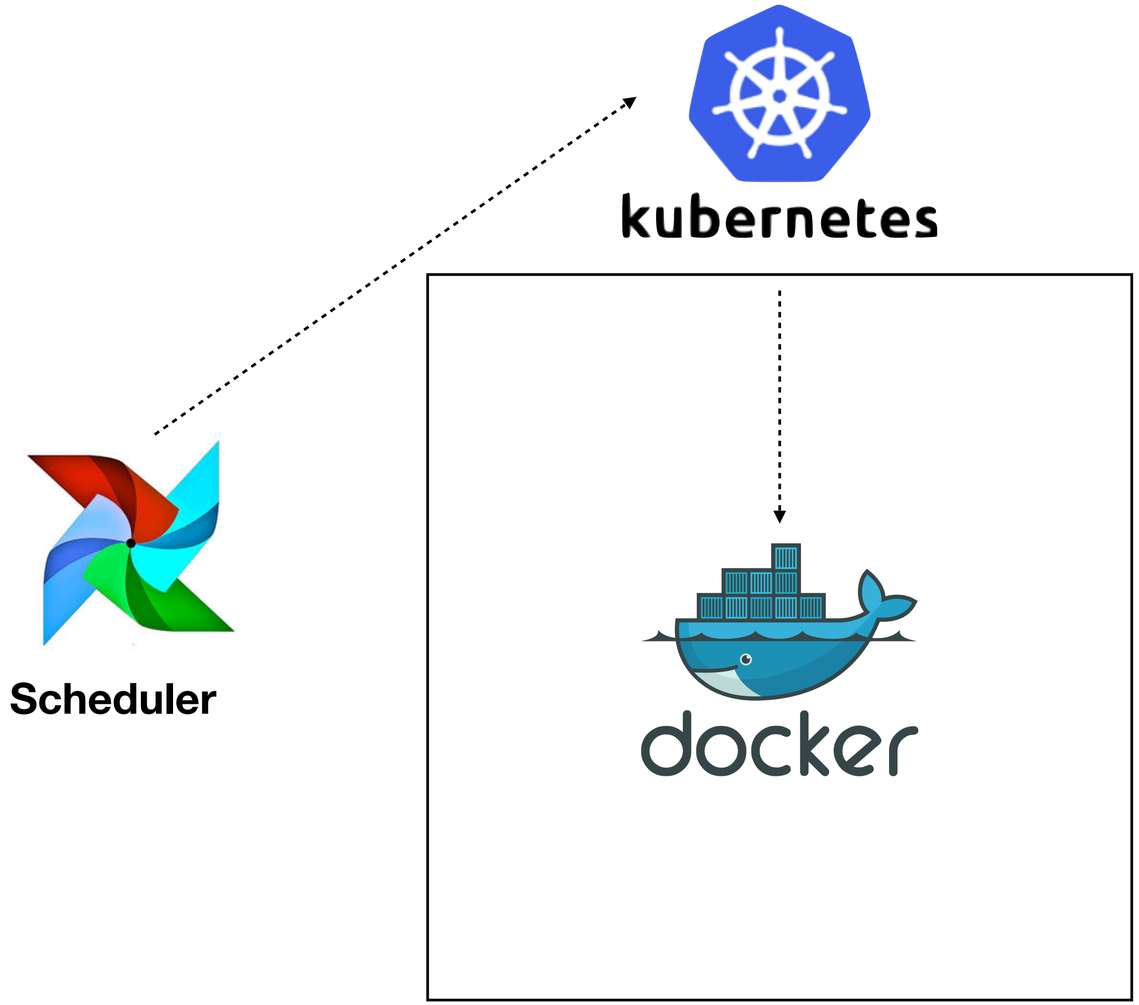
Airflow + Kubernetes

- Kubernetes Pod Operator
  - Flexibility
- Kubernetes Executor
  - Dynamic Allocation
- AirflowOperator (GCP)
  - One-step deployment



KubernetesPodOperator

- Allow users to deploy arbitrary Docker images
- Users can offload dependencies to containers
- “Lets Airflow focus on scheduling tasks”

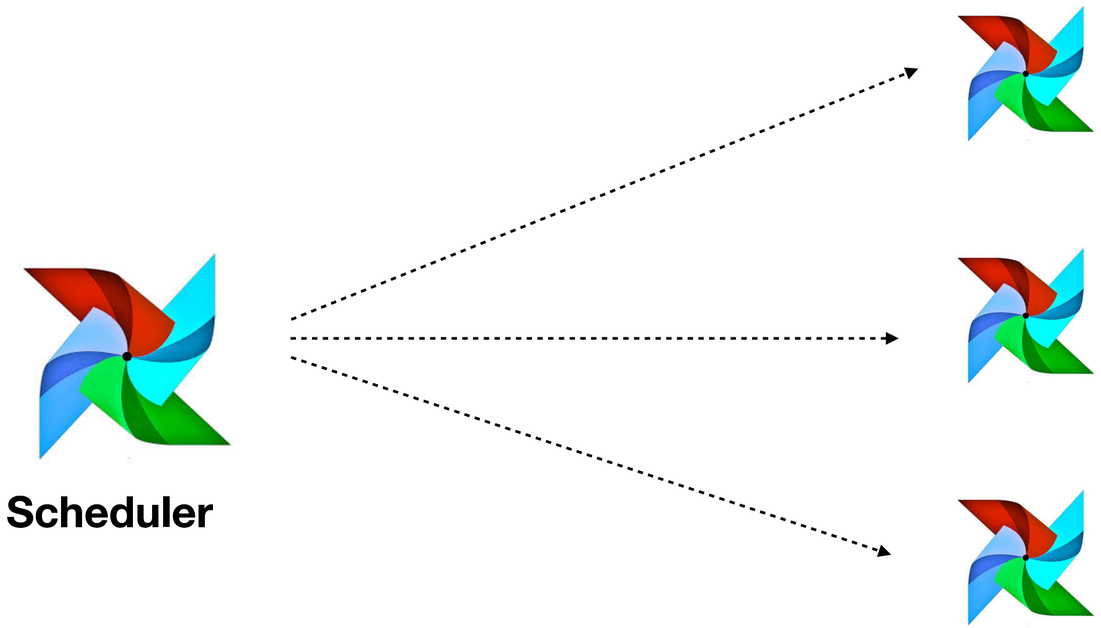


KubernetesPodOperator

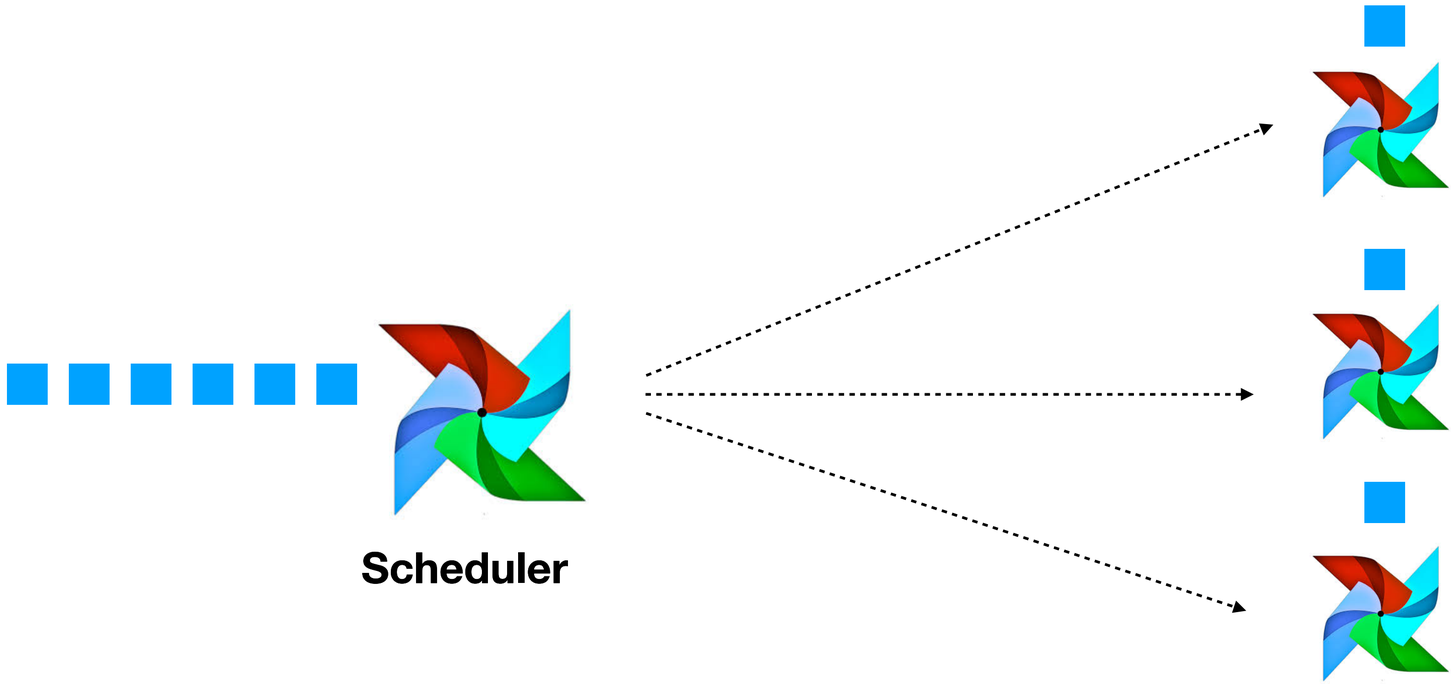
- Airflow workers are much lighter (don’t require extra libraries)
- Easy rollbacks + deployments through tags

```
dag = DAG(
    'kubernetes_sample', default_args=default_args, schedule_interval=timedelta(minutes=10))


start = DummyOperator(task_id='run_this_first', dag=dag)

passing = KubernetesPodOperator(namespace='default',
                                image="Python:3.6",
                                cmds=["Python","-c"],
                                arguments=["print('hello world')"],
                                labels={"foo": "bar"},
                                name="passing-test",
                                task_id="passing-task",
                                get_logs=True,
                                dag=dag
                                )

failing = KubernetesPodOperator(namespace='default',
                                image="ubuntu:1604",
                                cmds=["Python","-c"],
                                arguments=["print('hello world')"],
                                labels={"foo": "bar"},
                                name="fail",
                                task_id="failing-task",
                                get_logs=True,
                                dag=dag
                                )

passing.set_upstream(start)
failing.set_upstream(start)
```

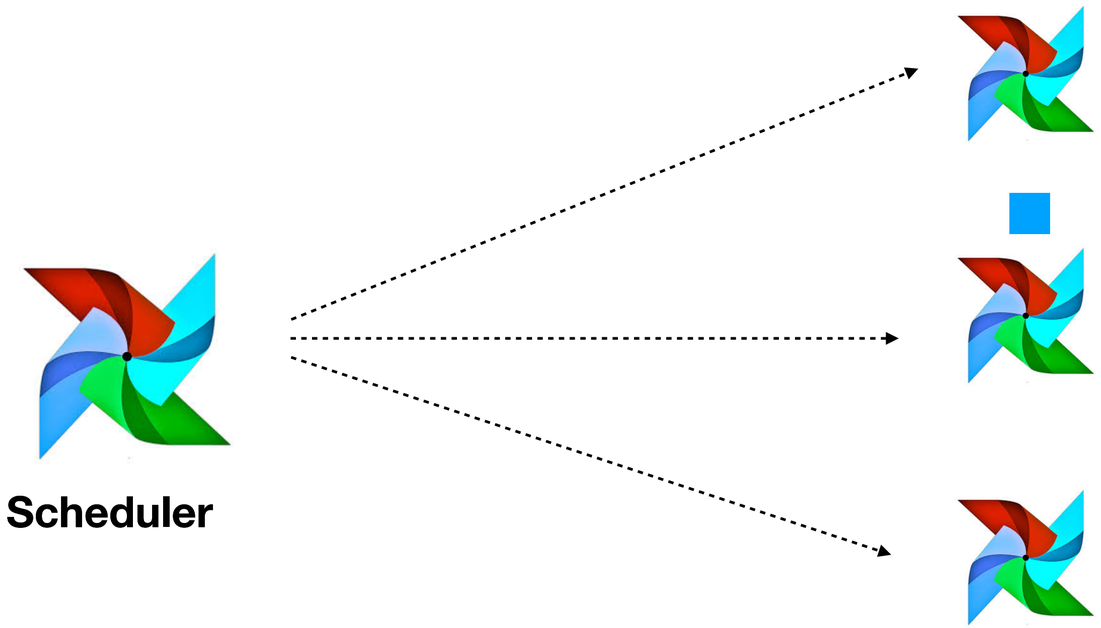
Static Allocation



Static Allocation



Static Allocation



Dynamic Resource Allocation

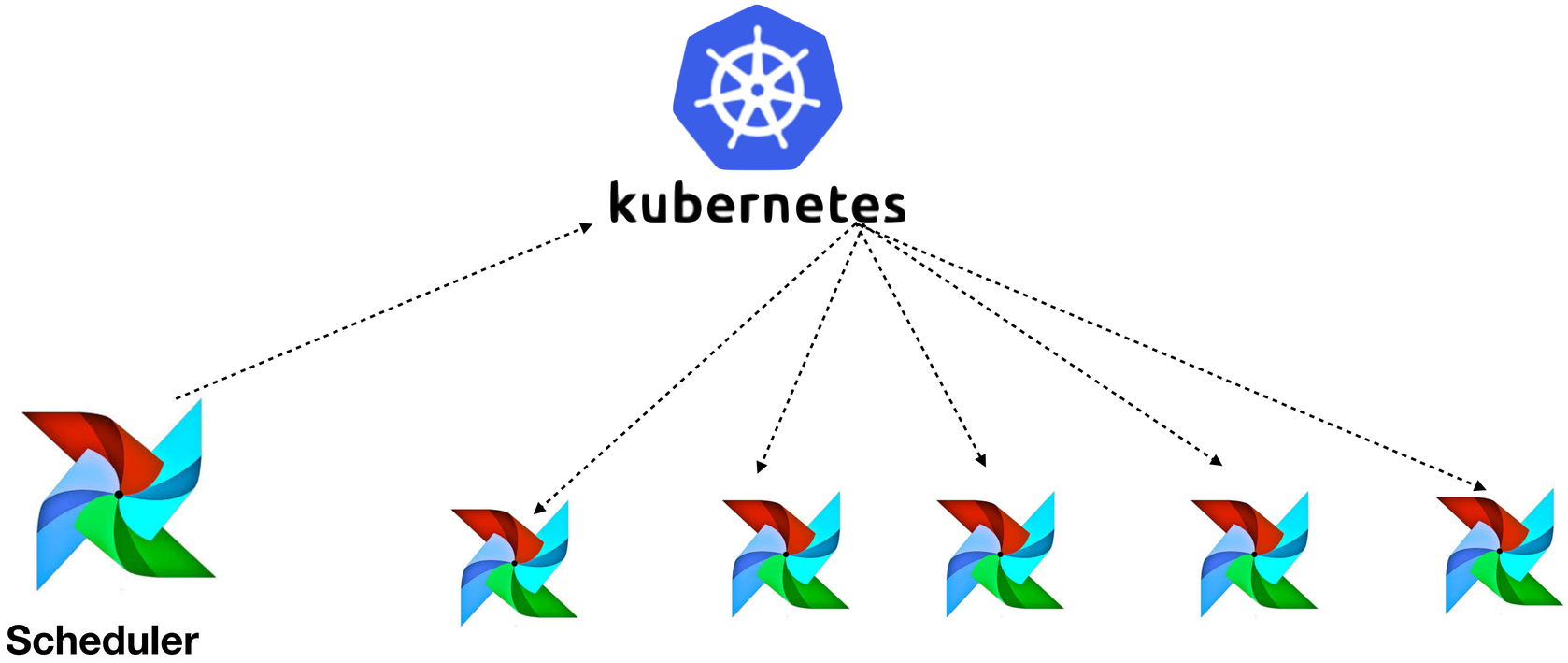
- All previous Airflow solutions involved static clusters of workers
- Require over-provisioning
- Airflow workers need to handle dependencies for multiple job types

shoiuld we keep this?

Kubernetes Executor

- High levels of parallelism (dynamic allocation)
- Task-level pod configuration
- Fault Tolerance

Dynamic Allocation



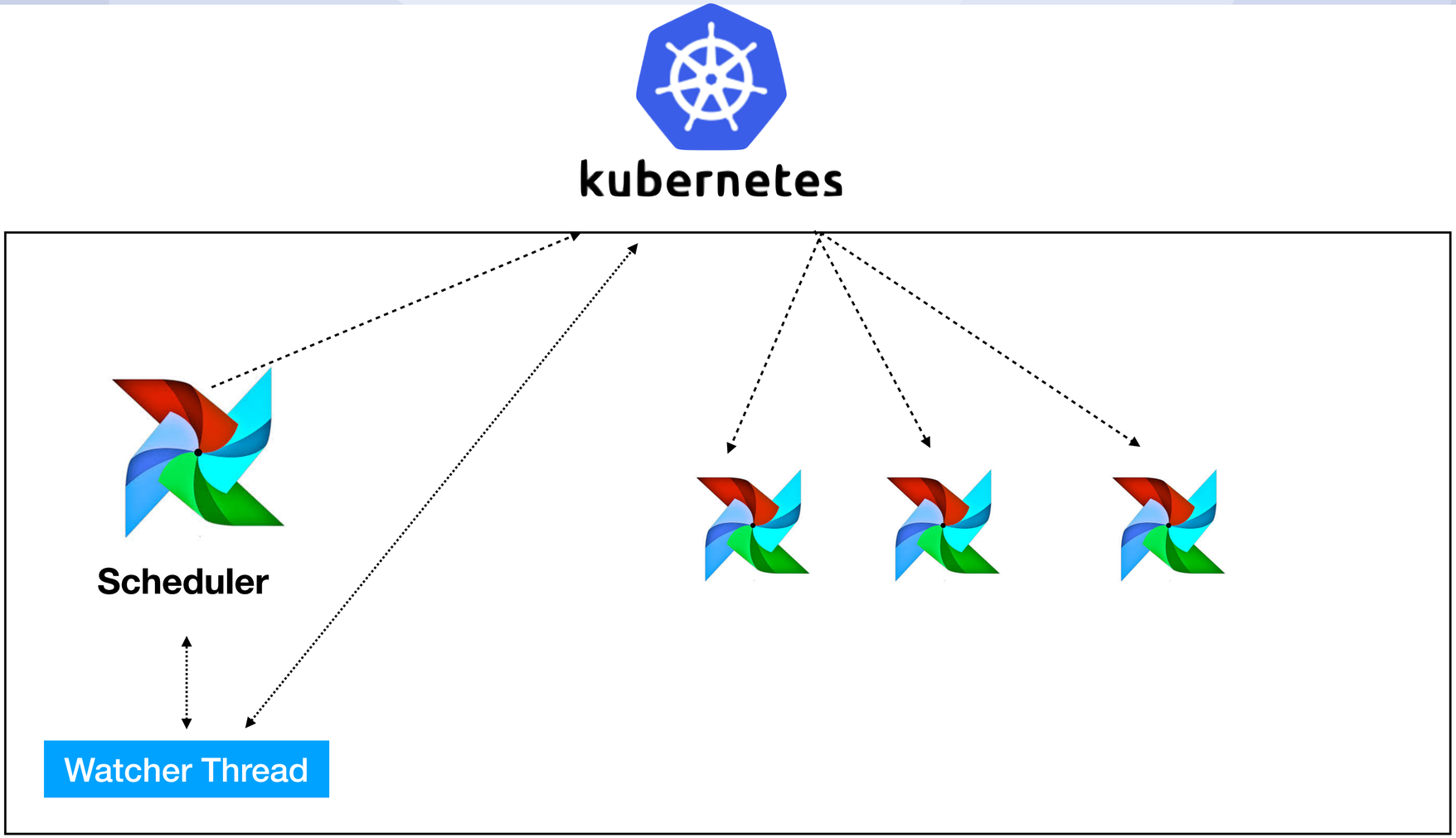
Task Level Configs


```
t = BashOperator(
  task_id = ‘account-test’,
  bash_command = ‘gcloud auth application-default login’,
  dag = dag,

  executor_config = {
  ‘request_memory’: ‘128Mi’,
  ‘limit_memory’: ‘128Mi’
  ‘image’: ‘airflow/scipy:1.1.5’
  ‘gcp-service-account’: ‘service-account@xxx.iam.gserviceaccount.com’
  }

)
```

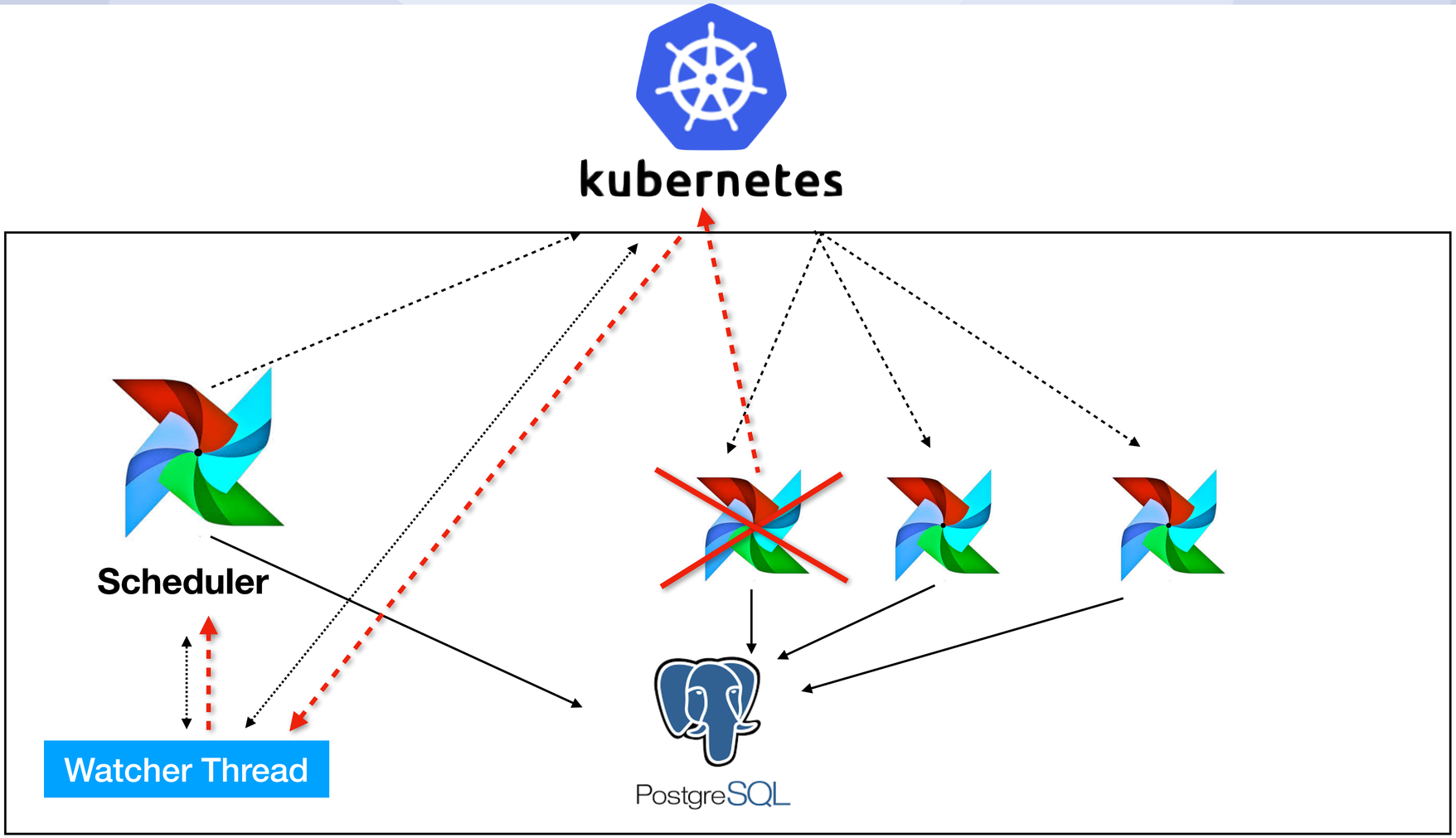
Dynamic Allocation

- Don’t need to think about capacity planning
- Great for time-sensitive jobs, which would traditionally require over-provisioning

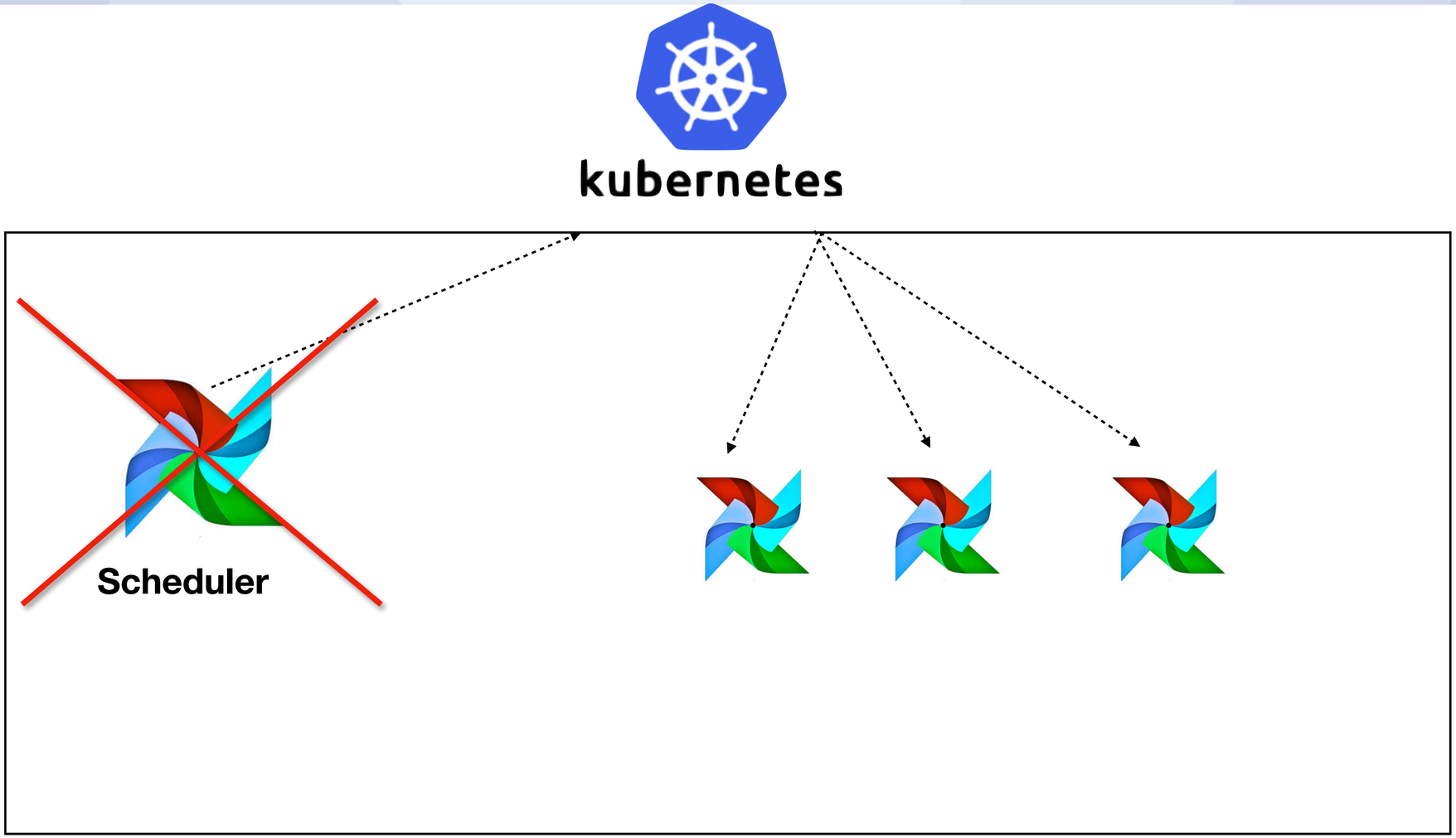
Kubernetes Executor



Kubernetes Executor



Fault Tolerance



Fault Tolerance

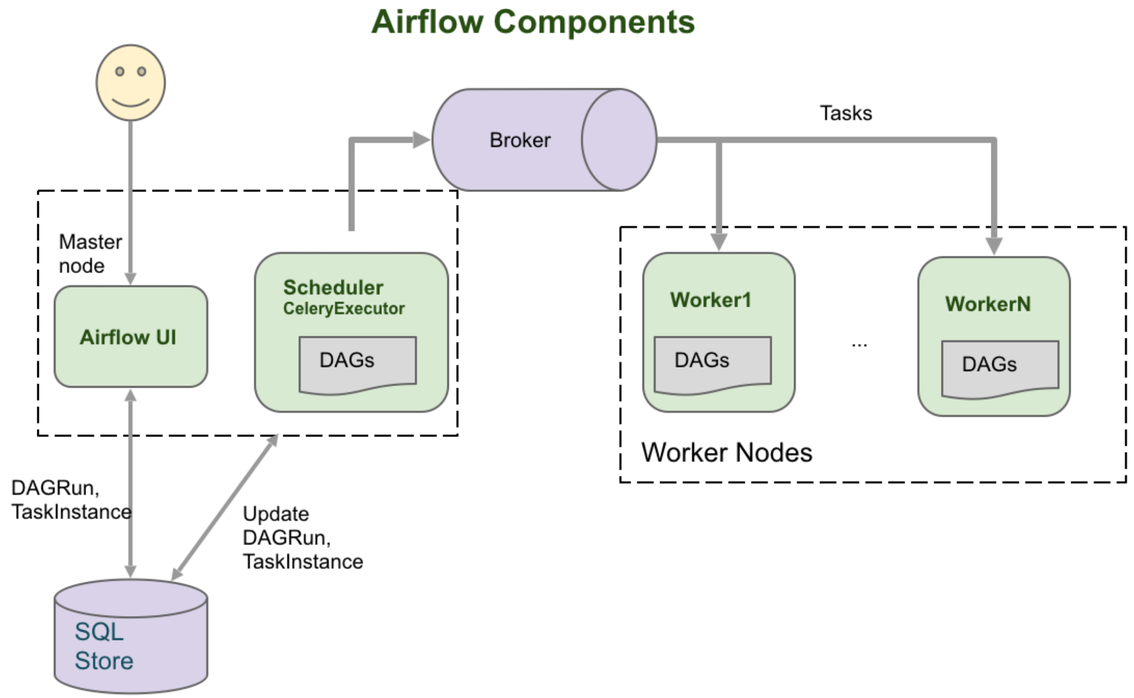
- Uses “resourceVersion” to re-create state
- Maintain a resourceVersion in SQL table for state recovery

AirflowOperator

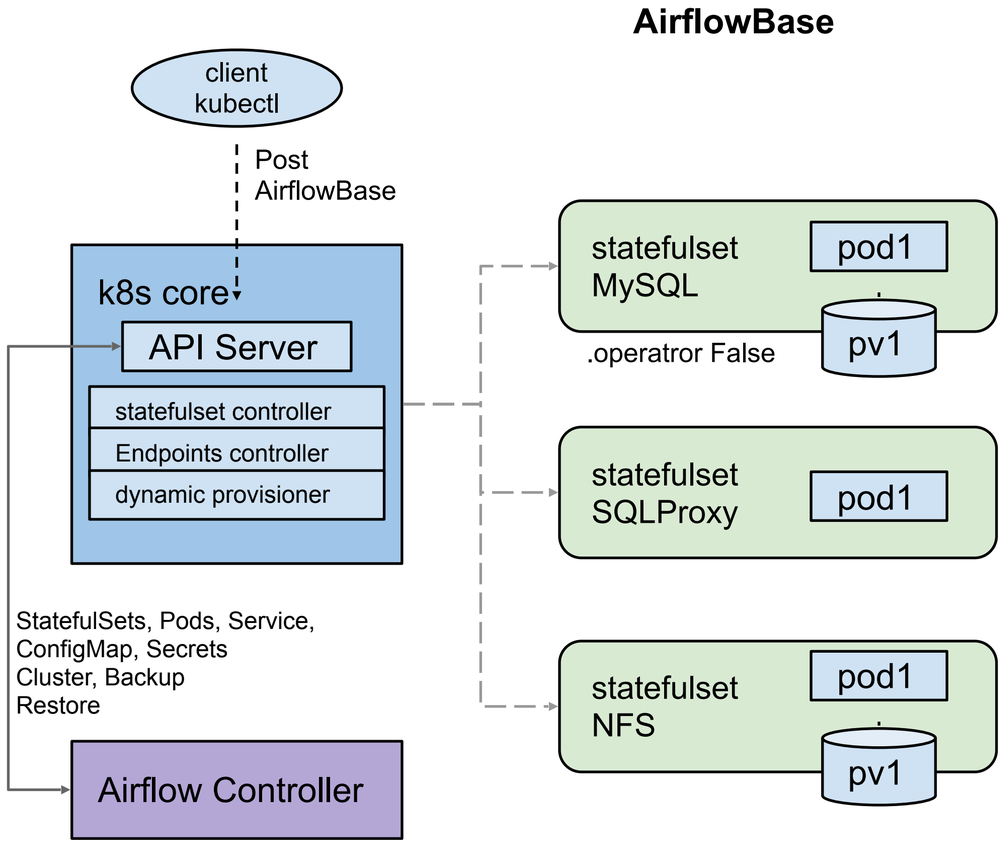
- AirflowOperator custom controller for deploying and managing Airflow
- **CRD(declarative spec) + custom controller**
- AirflowOperator watches for AirflowBase, AirflowCluster CRDs and deploys airflow components

Lower Barrier to Entry

- Airflow can involve a somewhat onerous set-up to run at scale
- Requires multiple technologies (Celery, Redis, SQL, etc)
- Have to set up alerting/monitoring for Airflow system



AirflowBase CRD

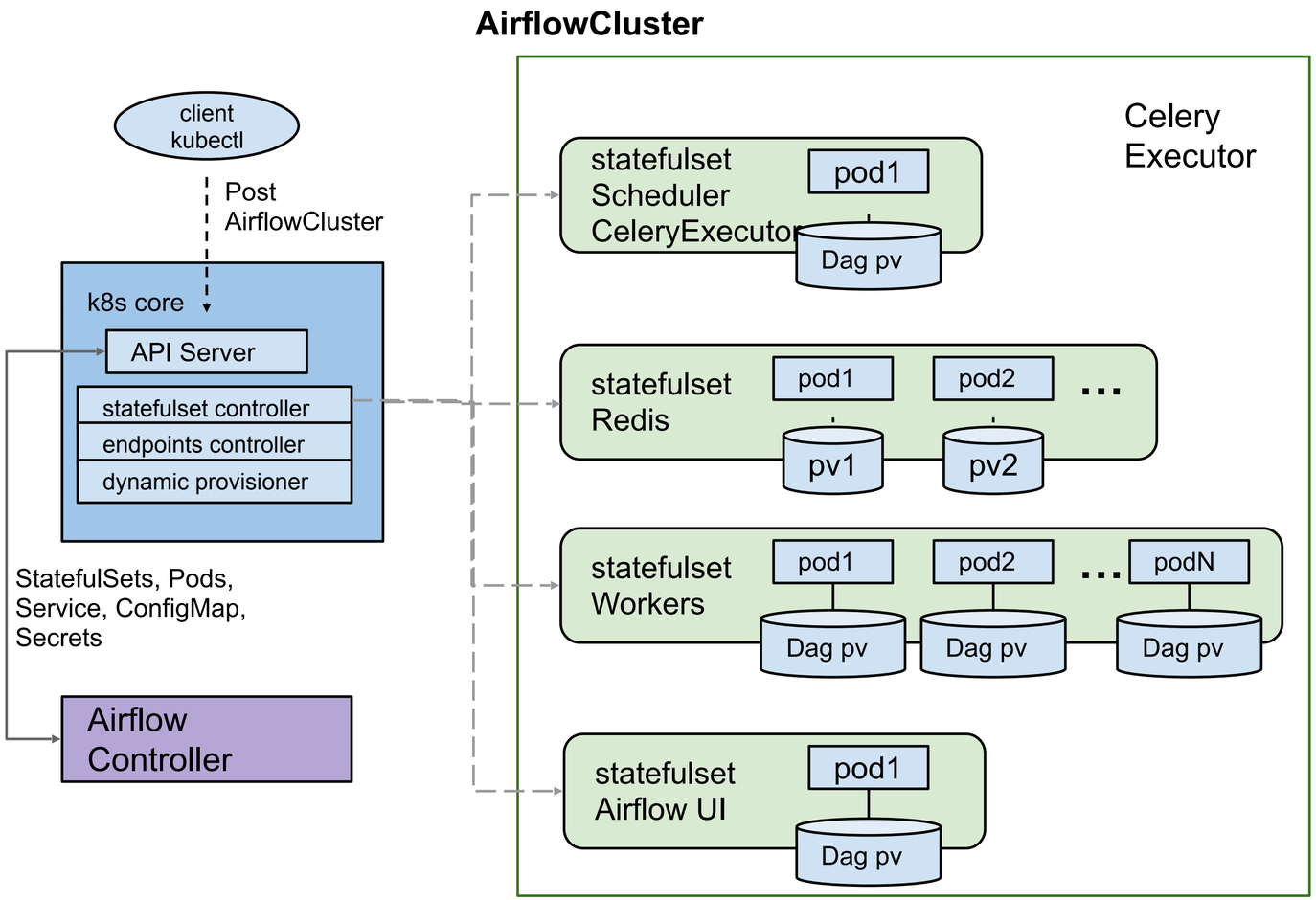


- AirflowBase CRD
  - MySQL/Postgres/SQLProxy
  - NFS
- Used by multiple Airflow Clusters

```
apiVersion: airflow.k8s.io/v1alpha1
kind: AirflowBase
metadata:
  name: mc-base
spec:
  mysql:
    operator: False
  storage:
    version: ""
```

```
apiVersion: airflow.k8s.io/v1alpha1
kind: AirflowBase
metadata:
  name: ck-base
spec:
  sqlproxy:
          project: someproject
          region: us-central1
          instance: testsql-cluster
  storage:
    version: ""
```

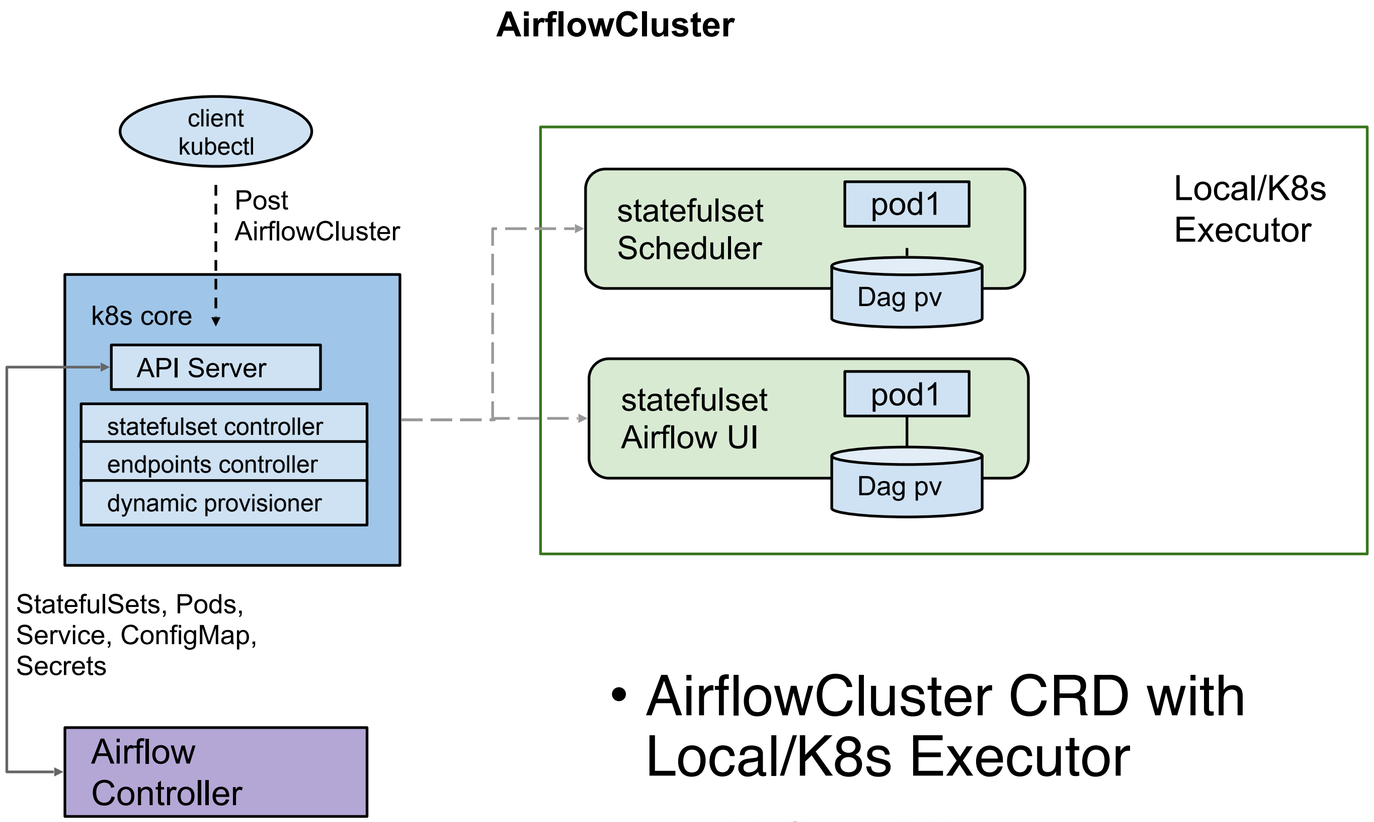
AirflowCluster CRD



- Celery Executor
  - Redis
  - Airflow UI
  - Airflow Scheduler
  - Airflow Workers
- Each cluster gets its own unique SQL connection string (user:password/dB).

```
apiVersion: airflow.k8s.io/v1alpha1
kind: AirflowCluster
metadata:
  name: mc-cluster
spec:
  executor: Celery
  config:
    airflow:
      AIRFLOW_SOME_CONFIG: SomeValue
  redis:
    operator: False
  scheduler:
    version: "1.10.1"
  ui:
    replicas: 1
    version: "1.10.1"
  worker:
    replicas: 2
    version: "1.10.1"
  flower:
    replicas: 1
    version: "1.10.1"
  dags:
    subdir: "airflow/example_dags/"
    git:
      repo: "https://github.com/apache/incubator
airflow/"
      once: true
  airflowbase:
    name: mc-base
```

AirflowCluster CRD

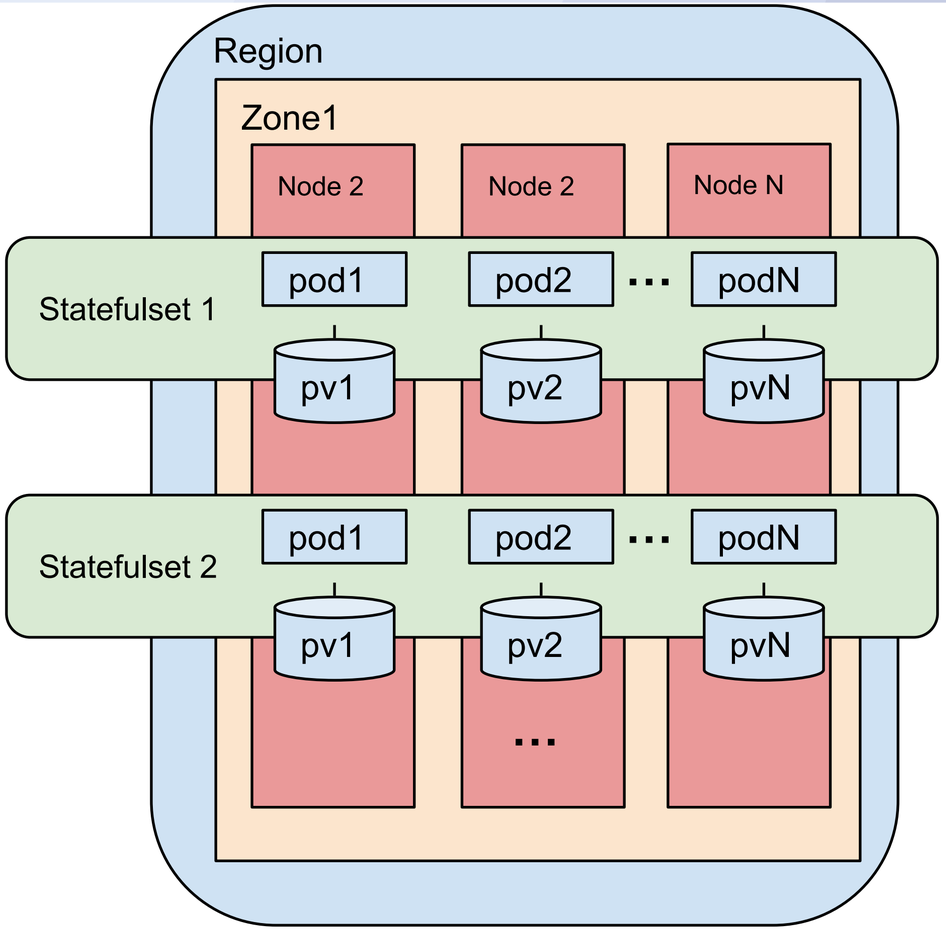


- AirflowCluster CRD with Local/K8s Executor
  - Airflow UI
  - Airflow Scheduler

```
apiVersion: airflow.k8s.io/v1alpha1
kind: AirflowCluster
metadata:
  name: mk-cluster
spec:
  executor: Kubernetes
  ui:
    replicas: 1
    version: "1.10.1"
  scheduler:
    version: "1.10.1"
  worker:
    version: "1.10.1"
  dags:
    subdir: "airflow/example_dags/"
    git:
      repo: "https://github.com/apache/
incubator-airflow/"
      once: true
      branch: master
  airflowbase:
    name: mc-base
```

AirflowOperator

- cluster.Spec.Affinity.*.topology can be set to “kubernetes.io/hostname” to spread Pods across Nodes within a Zone.
- Limit the effect of node failures within a zone



Pods spread across Nodes in Zone

Monitoring

- Can use existing Kubernetes infrastructure
- Only needed to think about Airflow, not machines

Prometheus


Kubernetes Node Exporter Full
Last 24 hours
Refresh every 1m
Host:
172.20.100.44
Basic CPU / Mem / Disk Gauge
CPU Busy
30%
Used RAM Memory
4%
Used SWAP
N/A
Used Root FS
10%
CPU System Load (1m avg)
30%
CPU System Load (5m avg)
34%
Basic CPU / Mem / Disk Info
CPU Cores
4
Total RAM
29.96 GiB
Total SWAP
0 B
Total RootFS
93.6 GiB
System Load (1m avg)
1.18
Uptime
1.5 weeks
Basic CPU / Mem Graph
CPU Basic
100%
Memory Basic
37 GiB

Elasticsearch

airflow

Selected Fields

? _source

Available Fields

Popular

@timestamp

t kubernetes.container_n...

t kubernetes.labels.app

t kubernetes.namespace...

t log

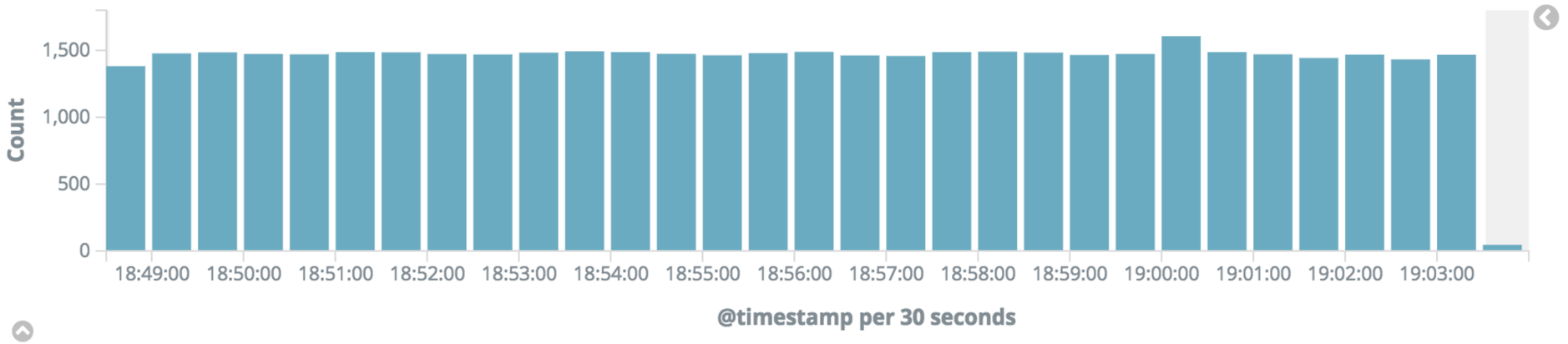
t _id

t _index

_score

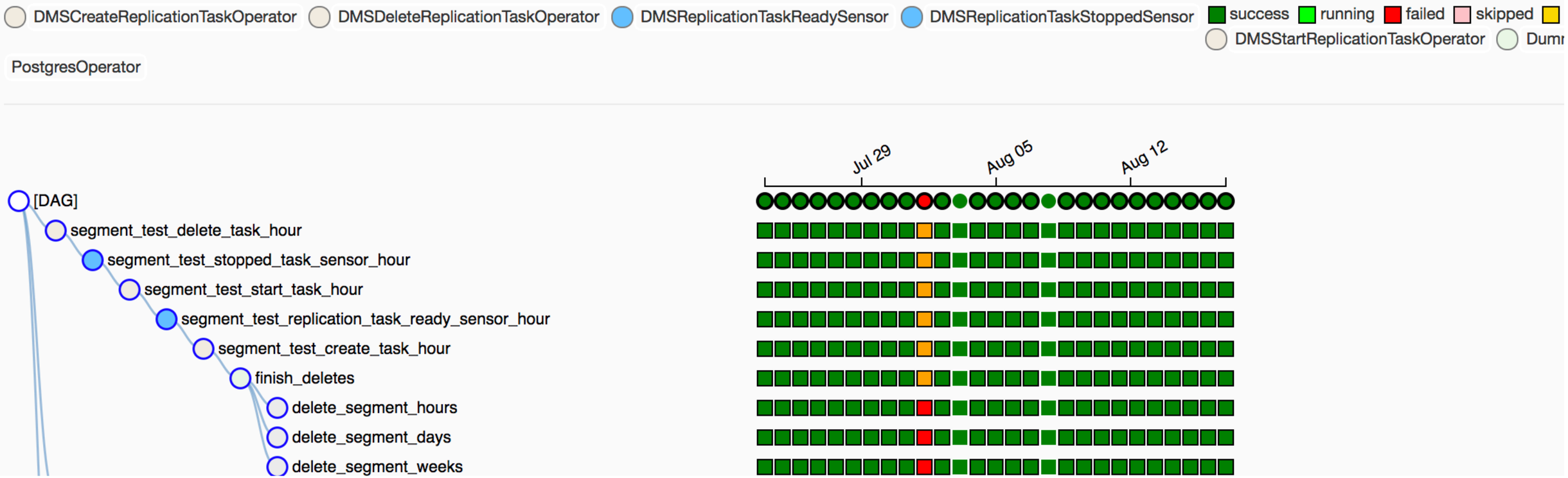
t _type

t kubernetes.annotations...

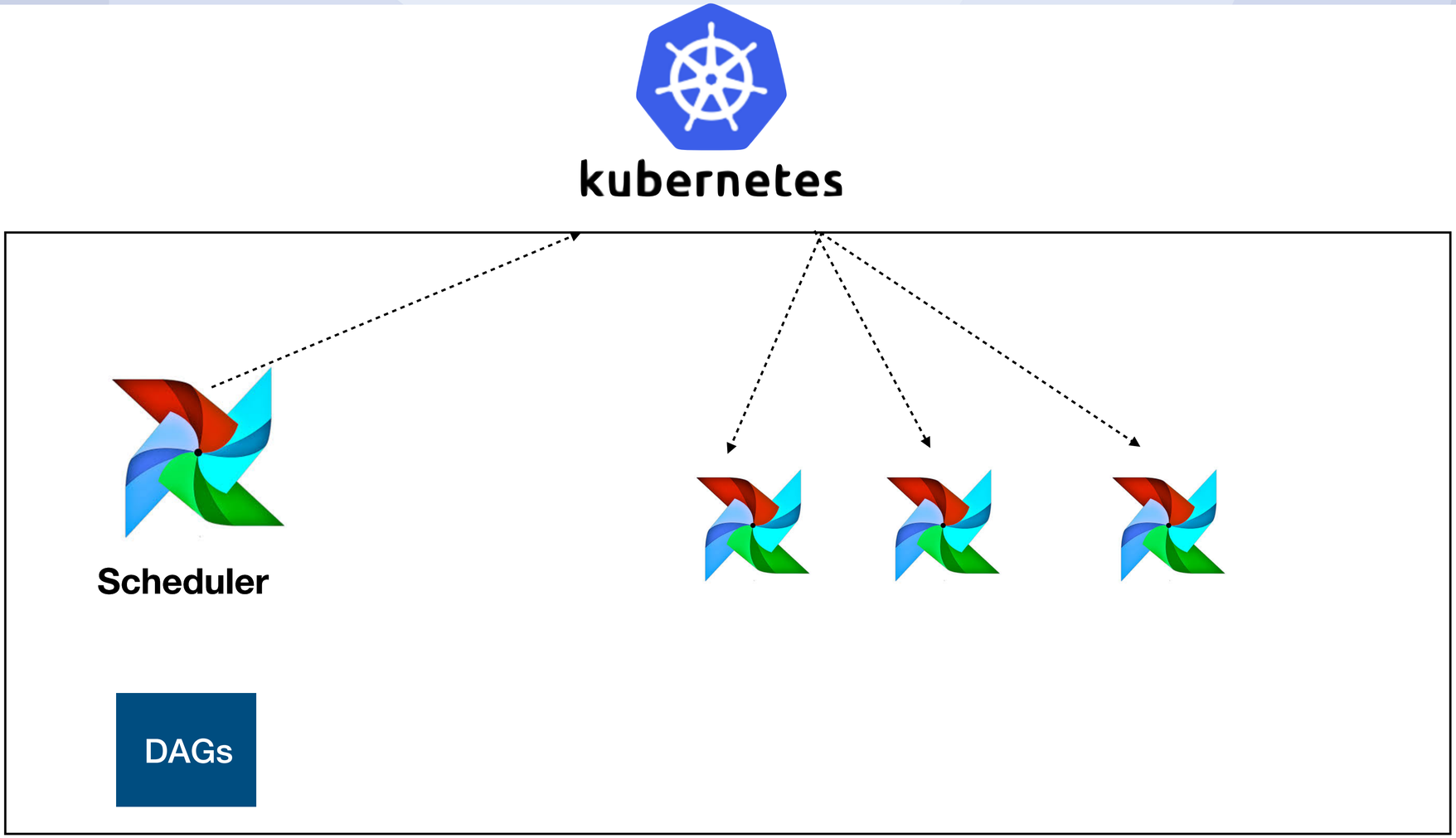


Time	_source
August 17th 2018, 19:03:30.801	log: [2018-08-18 02:03:30,801] {jobs.py:378} INFO - Processing /root/airflow/dags/mtd_tx_usage_load.py took 0.428 seconds\n kubernetes.pod_name: pn-gsd-airflow-airflow-2587458194-79ts7 kubernetes.namespace_name: pn-gsd-airflow kubernetes.labels.app: airflow kubernetes.labels.release: pn-gsd-airflow kubernetes.annotations.kubernetes.io/created-by: {\"kind\":\"SerializedReference
August 17th 2018, 19:03:30.793	kubernetes.pod_name: pn-gsd-airflow-airflow-2587458194-79ts7

Airflow Dashboard

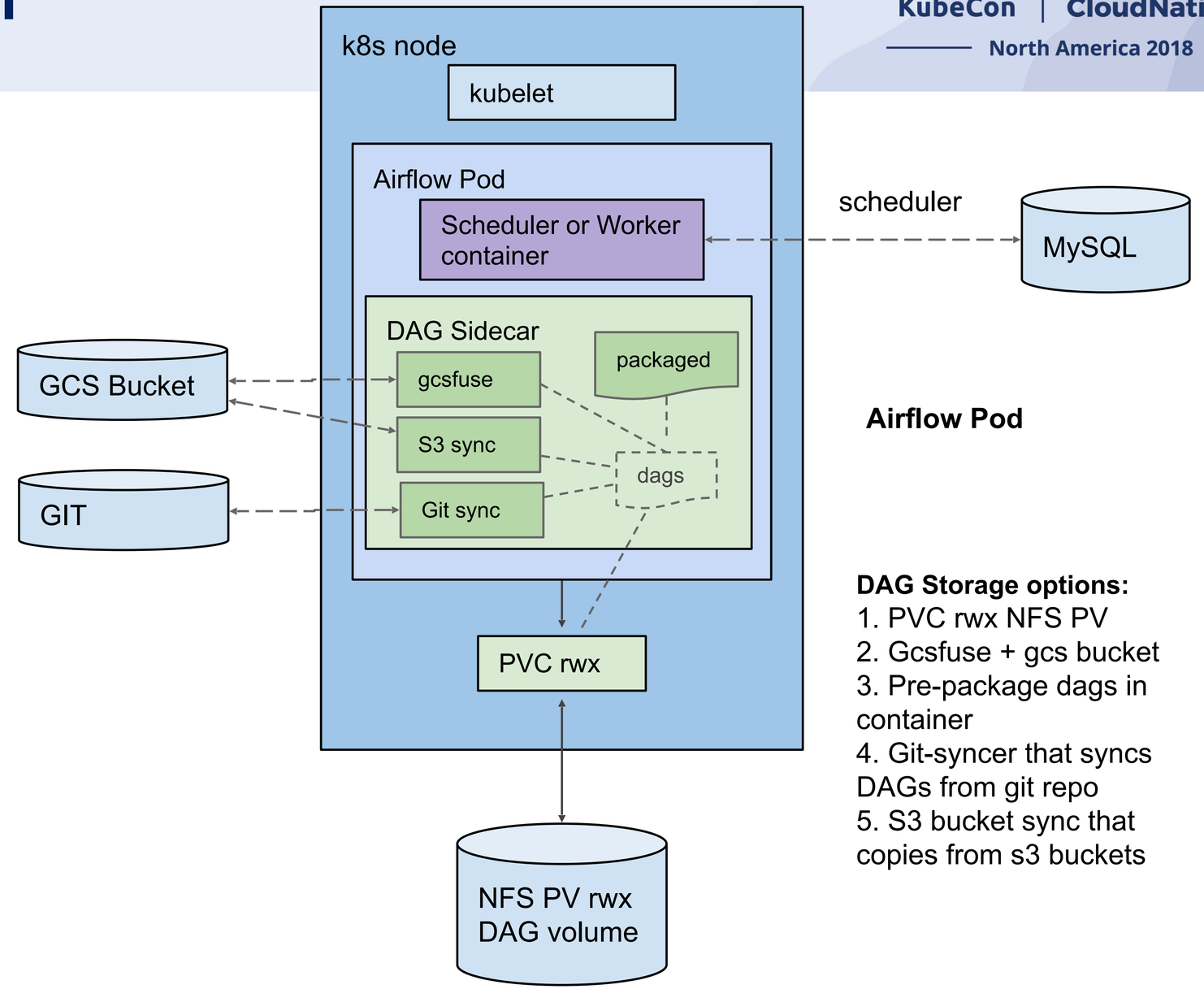


DAG Propagation



AirflowOperator

- Multiple DAG sources are supported via a DAG Sidecar
- Custom Airflow Pod images are supported



Airflow Pod

DAG Storage options:
1. PVC rwx NFS PV
2. Gcsfuse + gcs bucket
3. Pre-package dags in container
4. Git-syncer that syncs DAGs from git repo
5. S3 bucket sync that copies from s3 buckets

DAG Injection

- Three modes: Git-init mode, persistent volume mode, and “pre-baked” mode (1.10.1)
- Git-init mode + pre-baked is recommended for development and small instances of Airflow, because it does not involve any distributed file systems
- Persistent volume mode recommended for large DAG folders

Do we need this slide ?

K8sExecutor Status

- Has been released with Airflow 1.10 in experimental mode
- Multiple companies already using in production
- Helm chart in progress
- Active community in #sig-big-data on kubernetes.slack.com

Airflow Operator Status

- Supports Airflow 1.10.1
- We are currently active with the #sig-big-data and #airflow-operator channels on kubernetes.slack.com
- Available on Kubernetes Cloud Marketplace in GCP

Demo

What’s Next?

KubeCon

CloudNativeCon

North America 2018

Throttling

- The Kubernetes Scheduler is naturally greedy (it will take as many tasks as it can, even if it doesn’t have available resources)
- Want to ensure that if the cluster is in trouble, that we stop launching tasks until resources are available
- Maintain a “pending set” to ensure that no more than 5 Airflow tasks are pending at once

Scale Testing

- Have been working with members of Kubernetes team to create GKE instances for scale testing
- Writing scripts to generate giant DAGs to test DAG propagation and parsing times
- Seeing what scale of tasks begin to place pressure on PostgreSQL instance

How can you get involved?

- Looking for brave souls interested in experimenting with Executor and AirflowOperator to take it to beta
- Distributed testing using Chaos Monkey/Kubernetes
- Currently active with the #sig-big-data and #airflow-operator channels on kubernetes.slack.com
- Scale testing
- Throttle DAG tasks in Kubernetes executor

Thank You

Learn more:

github.com/apache/incubator-airflow/
@danimberman

github.com/GoogleCloudPlatform/airflow-operator
@bharanis


KubeCon
CloudNativeCon
North America 2018