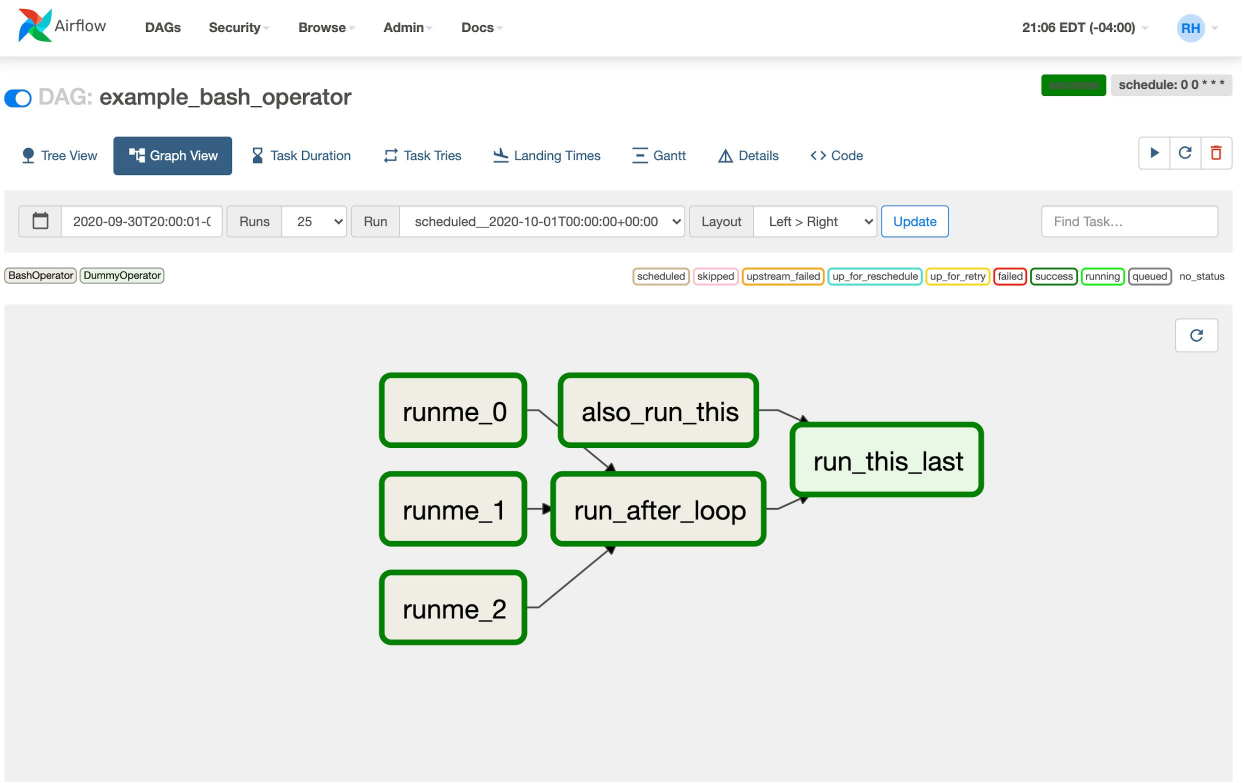
Airflow on Kubernetes

tips, tricks and pitfalls

What is airflow?

- Workflow
  - Execution
  - Scheduling
  - Monitoring
- Extensible (providers, plugin)
- Scalable
- Python

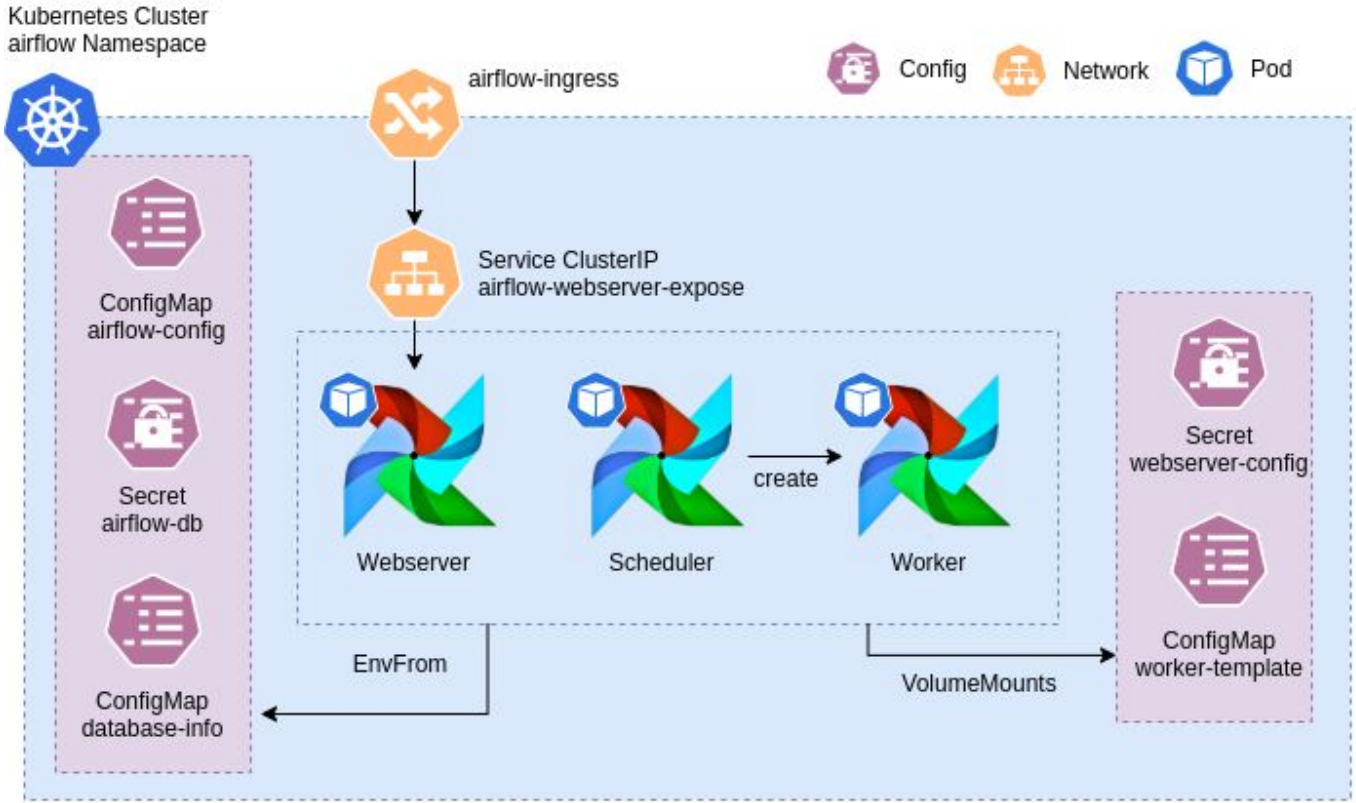
Vocabulary


Airflow
DAGs
Security
Browse
Admin
Docs
21:06 EDT (-04:00)
RH
DAG: example_bash_operator
success
schedule: 0 0 * * *
Tree View
Graph View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2020-09-30T20:00:01-(
Runs
25
Run
scheduled__2020-10-01T00:00:00+00:00
Layout
Left > Right
Update
Find Task...
BashOperator
DummyOperator
scheduled
skipped
upstream_failed
up_for_reschedule
up_for_retry
failed
success
running
queued
no_status
runme_0
also_run_this
run_this_last
runme_1
run_after_loop
runme_2

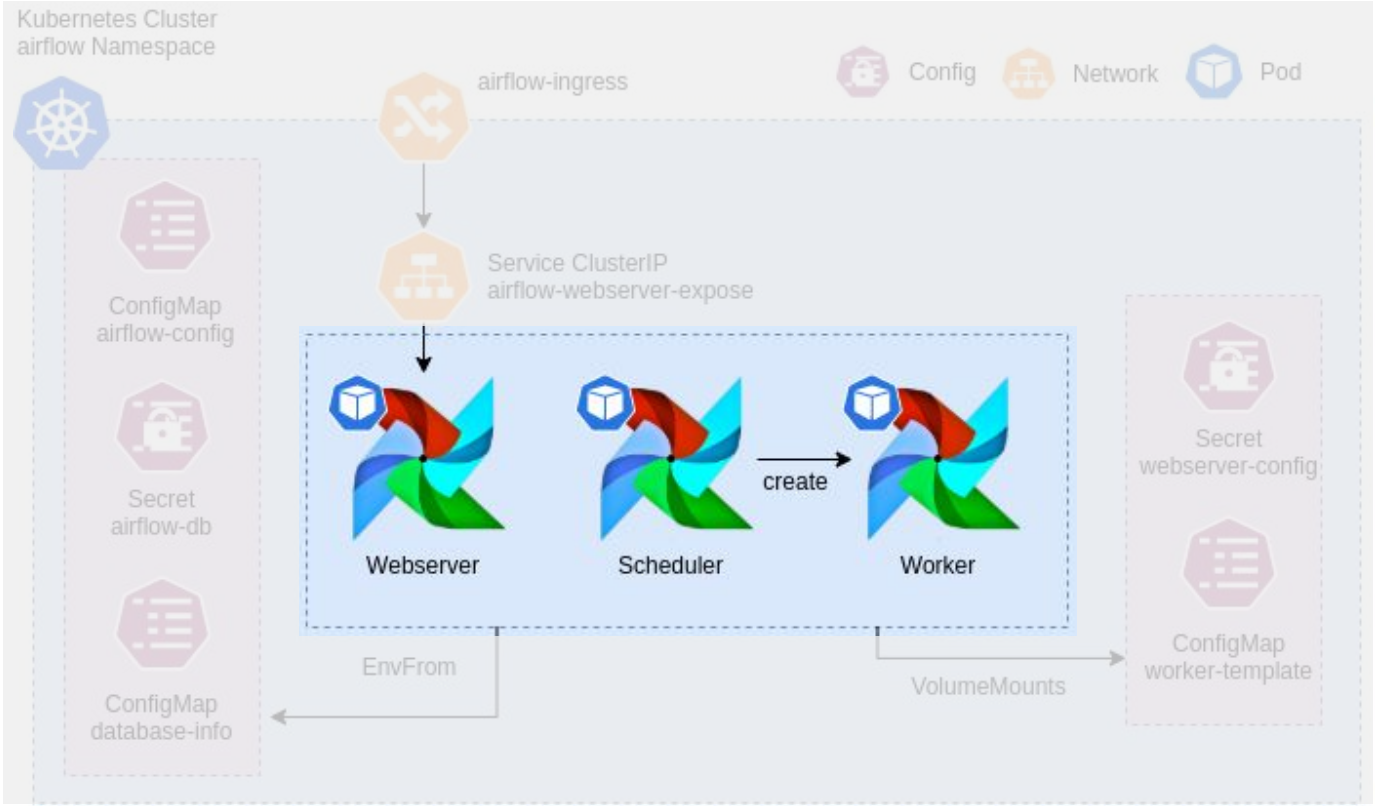
What is Kubernetes?

- Containerized workloads
  - Manage
  - Self Heal
  - Scale
- Scalable
- High Availability
- Go

Airflow on K8S Overview

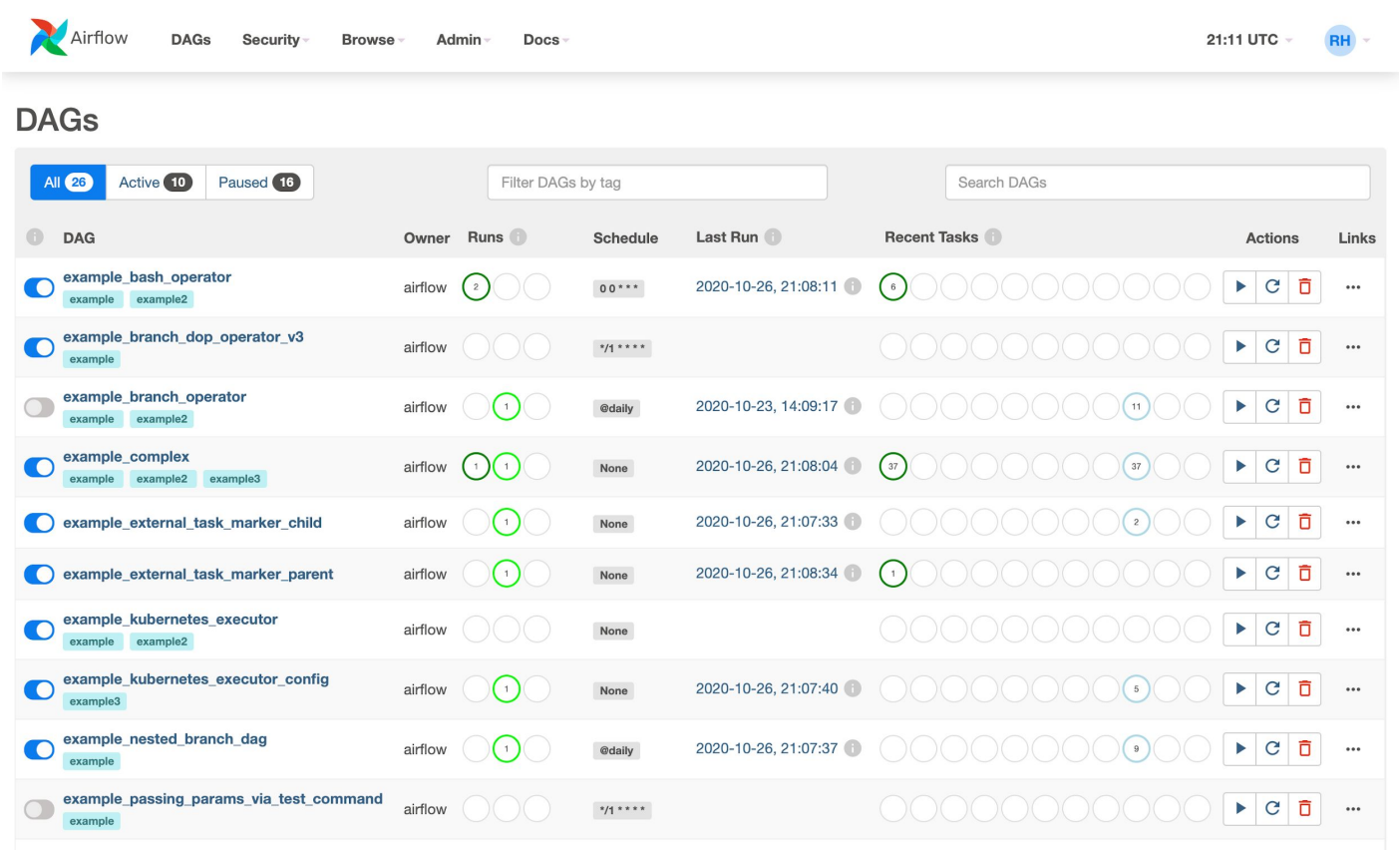


Airflow Components



Webserver

- WebUI
- REST API
- Authentication



Scheduler

- Parse DAG
- Trigger DAG and Task
- Monitor DAG and Task
- High Availability since airflow 2.0
- Check Concurrency and resource Limits

Workers

- Execute the Task
- Can use different backend
  - Dask
  - Celery
  - Kubernetes
  - Local

DB

- Configurable
  - MySQL
  - Postgres
  - SQLite (for dev only)
- Airflow use SQLAlchemy
- Used by scheduler, webserver and worker
- Store DAG and Task Execution
- Store DAG and Task State
- Store xcom

Let's Begin

Webserver

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: airflow-webserver
spec:
 replicas: 1
 revisionHistoryLimit: 1
 selector:
   matchLabels:
     app: airflow-webserver
 template:
   metadata:
     labels:
       app: airflow-webserver
   spec:
     containers:
       - image: apache/airflow:2.0.0-python3.8
         name: airflow-webserver
         command: ["/entrypoint", "webserver"]
```

Scheduler

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: airflow-scheduler
spec:
 replicas: 1
 revisionHistoryLimit: 1
 selector:
   matchLabels:
     app: airflow-scheduler
 template:
   metadata:
     labels:
       app: airflow-scheduler
   spec:
     containers:
       - image: apache/airflow:2.0.0-python3.8
         name: airflow-scheduler
         command: ["/entrypoint", "scheduler"]
```

The End

```
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) no such table: dag
```

Init DB

- /!\ use volume and set pgdata
- separate database info from secret since you can use the database for other application
- if you reuse this DB you will have to create a database with correct user and permission before launching airflow
- if airflow don't find database on startup it fail

```yaml
apiVersion: v1
kind: Service
metadata:
 labels:
   app: db
 name: "postgres-db"
spec:
 ports:
   - port: 5432
     protocol: TCP
     targetPort: 5432
 selector:
   app: db
 type: ClusterIP
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
   app: db
 name: postgres-db
spec:
 replicas: 1
 revisionHistoryLimit: 1
 selector:
   matchLabels:
     app: db
 template:
   metadata:
     labels:
       app: db
   spec:
     containers:
       - image: postgres:13.0-alpine
         name: postgres
         envFrom:
           - secretRef:
               name: airflow-db
           - configMapRef:
               name: database-info
         ports:
           - containerPort: 5432
```

```yaml
apiVersion: v1
kind: Secret
metadata:
 name: airflow-db
type: Opaque
stringData:
 POSTGRES_USER: airflow
 POSTGRES_PASSWORD: airflow
 POSTGRES_DB: airflow
 FERNET_KEY: "cxIPNJl4G6trLTpzFvVpPdi1yZWnnbnsnzzKanFyWfY="
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: database-info
data:
 POSTGRES_HOST: postgres-db
 POSTGRES_PORT: "5432"
```

```
FERNET_KEY: {{ randAscii 32 | b64enc | quote }}
```

Set DB parameter in Airflow

- ***envFrom*** ConfigMap and secret are populated before ***env***
- set DB env variable in webserver scheduler and worker
- Fernet key is used to encrypt connection password and variables

```
envFrom:
 - secretRef:
     name: airflow-db
 - configMapRef:
     name: database-info
env:
 - name: AIRFLOW__CORE__SQL_ALCHEMY_CONN
   value: "postgres://$(POSTGRES_USER):$(POSTGRES_PASSWORD)@$(POSTGRES_HOST):$(POSTGRES_PORT)/$(POSTGRES_DB)"
 - name: AIRFLOW__CORE__FERNET_KEY
   value: "$(FERNET_KEY)"
```

```
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) no such table: dag
```

Init Container

- You have to init Airflow db with init container
- Always use ***db upgrade*** it will create tables or upgrade them
- init DB in only one airflow container (webserver)
- the other component will start and fail during tables creation

```
initContainers:
 - name: upgrade-db
   image: apache/airflow:2.0.0-python3.8
   command: [ "/entrypoint", "db", "upgrade" ]
   envFrom:
     - secretRef:
         name: airflow-db
     - configMapRef:
         name: database-info
   env:
     - name: AIRFLOW__CORE__SQL_ALCHEMY_CONN
       value: "postgres://$(POSTGRES_USER):$(POSTGRES_PASSWORD)@$(POSTGRES_HOST):$(POSTGRES_PORT)/$(POSTGRES_DB)"
```

```
DB_BACKEND=postgres
DB_HOST=postgres-db
DB_PORT=5432

/home/airflow/.local/lib/python3.8/site-packages/azure/cosmos/session.py:186 SyntaxWarning: "is not" with a literal. Did you mean "!="?
/home/airflow/.local/lib/python3.8/site-packages/azure/storage/blob/_upload_chunking.py:384 SyntaxWarning: "is" with a literal. Did you mean "=="?
/home/airflow/.local/lib/python3.8/site-packages/azure/storage/blob/baseblobservice.py:975 SyntaxWarning: "is not" with a literal. Did you mean "!="?
/home/airflow/.local/lib/python3.8/site-packages/azure/storage/blob/baseblobservice.py:2527 SyntaxWarning: "is not" with a literal. Did you mean "!="?
/home/airflow/.local/lib/python3.8/site-packages/azure/storage/_connection.py:84 SyntaxWarning: "is" with a literal. Did you mean "=="?
[2021-01-20 09:11:11,265] {opentelemetry_tracing.py:28} INFO - This service is instrumented using OpenTelemetry. OpenTelemetry could not be imported;
please add opentelemetry-api and opentelemetry-instrumentatio
n packages in order to get BigQuery Tracing data.
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2021-01-20 09:11:11,856] {dagbag.py:440} INFO - Filling up the DagBag from /dev/null
[2021-01-20 09:11:11,936] {manager.py:727} WARNING - No user yet created, use flask fab command to do it.
Running the Gunicorn Server with:
Workers: 4 sync
Host: 0.0.0.0:8080
Timeout: 120
Logfiles: - -
Access Logformat:
================================================================
[2021-01-20 09:11:18 +0000] [14] [INFO] Starting gunicorn 19.10.0
[2021-01-20 09:11:18 +0000] [14] [INFO] Listening at: http://0.0.0.0:8080 (14)
[2021-01-20 09:11:18 +0000] [14] [INFO] Using worker: sync
/usr/local/lib/python3.8/os.py:1023 RuntimeWarning: line buffering (buffering=1) isn't supported in binary mode, the default buffer size will be used
[2021-01-20 09:11:18 +0000] [16] [INFO] Booting worker with pid: 16
[2021-01-20 09:11:18 +0000] [17] [INFO] Booting worker with pid: 17
[2021-01-20 09:11:18 +0000] [18] [INFO] Booting worker with pid: 18
[2021-01-20 09:11:18 +0000] [19] [INFO] Booting worker with pid: 19
```

```
DB_BACKEND=postgres
DB_HOST=postgres-db
DB_PORT=5432


  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /    _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/     /_/    /_/  \____/____/|__/
[2021-01-20 09:11:18,259] {scheduler_job.py:1241} INFO - Starting the scheduler
[2021-01-20 09:11:18,259] {scheduler_job.py:1246} INFO - Processing each file at most -1 times
[2021-01-20 09:11:18,263] {dag_processing.py:250} INFO - Launched DagFileProcessorManager with pid: 14
[2021-01-20 09:11:18,265] {scheduler_job.py:1751} INFO - Resetting orphaned tasks for active dag runs
[2021-01-20 09:11:18,269] {settings.py:52} INFO - Configured default timezone Timezone('UTC')
[2021-01-20 09:16:18,367] {scheduler_job.py:1751} INFO - Resetting orphaned tasks for active dag runs
```

The End

WebUI ?

Exposing Webserver

Service

```yaml
apiVersion: v1
kind: Service
metadata:
 labels:
   app: airflow-webserver
 name: airflow-expose
spec:
 ports:
   - port: 8080
     protocol: TCP
     targetPort: 8080
 selector:
   app: airflow-webserver
 type: ClusterIP
```

Ingress

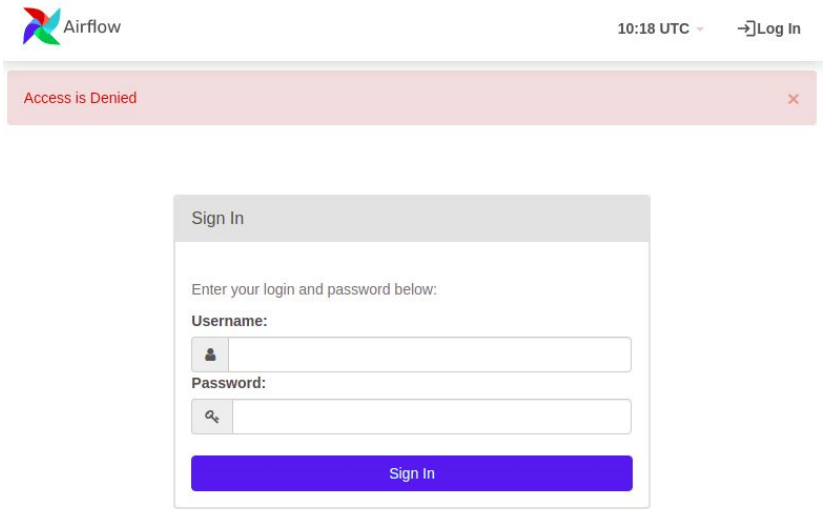
```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 name: airflow-ingress
 annotations:
   kubernetes.io/ingress.class: nginx
spec:
 rules:
   - host: airflow.local
     http:
       paths:
         - path: /airflow
           pathType: Prefix
           backend:
             serviceName: airflow-expose
             servicePort: 8080
```

Config

- Airflow can use environment variable as config
- You can also set config with airflow.cfg file
- ConfigMap will ensure airflow params consistency across service (webserver, scheduler)
- Using env is easier than putting airflow.cfg in a ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: airflow-config
data:
 AIRFLOW__WEBSERVER__BASE_URL: "http://minikube.info/airflow"
 AIRFLOW__WEBSERVER__ENABLE_PROXY_FIX: "True"
 AIRFLOW__WEBSERVER__EXPOSE_CONFIG: "True"
```

```yaml
envFrom:
 - configMapRef:
     name: airflow-config
```




```
[2021-01-20 10:19:03,988] {manager.py:727} WARNING - No user yet created, use flask fab command to do it.
```

```
$ airflow users create --username admin --role Admin --password admin \
                       --firstname FIRST_NAME --lastname LAST_NAME --email admin@example.org
```

The End

DAGs

Where is my DAG?

- /opt/airflow/dags (`AIRFLOW__CORE__DAGS_FOLDER`)
- mandatory in scheduler
- in webserver if you want to see code in the UI
- using configMap mounted in container doesn't work

Where is my DAG?

	In Image	Git Sync	Shared Volume
pros	easy setup no poll DAG consistency no volume	RWO Volume	DAG consistency across service Can use a webserver or something else to push DAG in the volume
cons	build and deploy an entire airflow for each DAG update	poll delay pull an entire repository (unless you use some git black magic with a custom git-sync container) DAG inconsistency Deploy key or public repository	need RWX volumes

Git Sync

- use kubernetes git-sync
- Create a multi container pod
- Share an emptyDir volume
- Use an init container for Worker
- Share ssh keys with kubernetes secrets

```yaml
containers:
 - image: k8s.gcr.io/git-sync/git-sync:v3.2.2
   name: git-sync
   env:
     - name: GIT_SYNC_REPO
       value: git@github.com:teamclairvoyant/airflow-maintenance-dags.git
     - name: GIT_SYNC_BRANCH
       value: airflow2.0
     - name: GIT_SYNC_WAIT
       value: "60"
     - name: GIT_SYNC_SSH
       value: "true"
     - name: GIT_SYNC_ROOT
       value: /tmp/git/
     - name: GIT_SSH_KEY_FILE
       value: /tmp/ssh/airflow-id-rsa
     - name: GIT_KNOWN_HOSTS
       value: "false"
     - name: GIT_SYNC_ADD_USER
       value: "true"
   volumeMounts:
     - name: dags
       mountPath: /tmp/git
     - name: airflow-git-sync-ssh
       mountPath: /tmp/ssh
```

Kubernetes Executor

- set worker pod template in `/opt/airflow/template` folder
- use security context to mount this volume with the right owner (50000)
- create a custom image with kubernetes dependencies
- configure worker image in airflow.cfg or with env var
- configure RBAC to allow the scheduler to create worker pod
- configure RBAC to allow the webserver to get pods logs
- configure RBAC to allow worker to create pod if you want to use KubernetesPodOperator

```
securityContext:
 runAsNonRoot: true
 runAsUser: 50000
 runAsGroup: 50000
 fsGroup: 50000
```

Worker Pod Template

	in Image	in ConfigMap
pros	easy setup	hot editing deploy with other airflow services
cons	no hot editing	

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: worker-pod-template
data:
 worker_pod_template.yaml: |-
   apiVersion: v1
   kind: Pod
   metadata:
     name: airflow-worker
   spec:
     serviceAccountName: airflow-worker
     containers:
       - image: apache/airflow:2.0.0-python3.8
         name: base
         envFrom:
           - secretRef:
               name: airflow-db
           - configMapRef:
               name: database-info
           - configMapRef:
               name: airflow-config
         env:
           - name: AIRFLOW__CORE__EXECUTOR
             value: LocalExecutor
           - name: AIRFLOW__CORE__SQL_ALCHEMY_CONN
             value: "postgres://..."
           - name: AIRFLOW__CORE__FERNET_KEY
             value: "$(FERNET_KEY)"
     restartPolicy: Never
     securityContext:
       runAsNonRoot: true
       runAsUser: 50000
       runAsGroup: 50000
       fsGroup: 50000
```

KubernetesExecutor config

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: airflow-config
data:
 ...
 AIRFLOW__CORE__EXECUTOR: "KubernetesExecutor"
 AIRFLOW__KUBERNETES__NAMESPACE: "airflow"
 AIRFLOW__KUBERNETES__POD_TEMPLATE_FILE: "/opt/airflow/template/worker_pod_template.yaml"
 AIRFLOW__KUBERNETES__WORKER_SERVICE_ACCOUNT_NAME: "airflow-worker"
 AIRFLOW__KUBERNETES__WORKER_CONTAINER_REPOSITORY: "apache-airflow"
 AIRFLOW__KUBERNETES__WORKER_CONTAINER_TAG: "2.0.0-python3.8"
 AIRFLOW__KUBERNETES__DELETE_WORKER_PODS: "True"
 AIRFLOW__KUBERNETES__DELETE_WORKER_PODS_ON_FAILURE: "True"
 AIRFLOW__KUBERNETES__IN_CLUSTER: "True"
```

RBAC: scheduler

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: airflow-scheduler-k8s-auth
rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list", "watch", "create", "patch", "delete"]
 - apiGroups: [""]
   resources: ["pods/log"]
   verbs: ["get"]
 - apiGroups: [""]
   resources: ["pods/exec"]
   verbs: ["create", "get"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: airflow-scheduler-k8s-auth
subjects:
 - kind: ServiceAccount
   name: airflow-scheduler
roleRef:
 kind: Role
 name: airflow-scheduler-k8s-auth
 apiGroup: rbac.authorization.k8s.io
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
 labels:
   name: airflow-scheduler
 name: airflow-scheduler
```

RBAC: webserver

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: airflow-webserver-k8s-auth
rules:
 - apiGroups: [""]
   resources: ["pods/log"]
   verbs: ["get"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: airflow-webserver-k8s-auth
subjects:
 - kind: ServiceAccount
   name: airflow-webserver
roleRef:
 kind: Role
 name: airflow-webserver-k8s-auth
 apiGroup: rbac.authorization.k8s.io
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
 labels:
   name: airflow-webserver
 name: airflow-webserver
```

RBAC: worker

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: airflow-worker
rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list", "watch", "create", "patch", "delete"]
 - apiGroups: [""]
   resources: ["pods/log"]
   verbs: ["get"]
 - apiGroups: [""]
   resources: ["pods/exec"]
   verbs: ["create", "get"]
 - apiGroups: [""]
   resources: ["secrets"]
   verbs: ["get", "watch", "list"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: airflow-worker
subjects:
 - kind: ServiceAccount
   name: airflow-worker
roleRef:
 kind: Role
 name: airflow-worker
 apiGroup: rbac.authorization.k8s.io
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
 labels:
   name: airflow-worker
 name: airflow-worker
```

Build your own Image

- why?
  - dag in image
  - custom providers (aws, k8s)
- **`--user`** because airflow image are not root
- **`--chown`** when you copy dags

```
FROM apache/airflow:2.0.0-python3.6

RUN pip install --user --no-cache-dir \
   apache-airflow-providers-amazon \
   apache-airflow-providers-cncf-kubernetes

COPY --chown=airflow:airflow dags /opt/airflow/dags/
```

Where are my Logs?

- Logs are in kubernetes worker that get destroy when task end
- You have to enable **`AIRFLOW__LOGGING__REMOTE_LOGGING`**
- Use service account to allow worker to write on the logs bucket

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: airflow-config
data:
 ...
 AIRFLOW__LOGGING__REMOTE_LOGGING: "True"
 AIRFLOW__LOGGING__REMOTE_BASE_LOG_FOLDER: "s3://logs-airflow"
```

Where are my Logs? Azure Edition

- You need to create a wasb connection on airflow (extra can take a **`connection_string`**)
- You need to define a log_config.py (example is airflow_local_settings.py)
- override wasb_container with your container name or fetch it from config
- add the log_config.py in PYTHONPATH
- use "secret backend" to set connection

```
if REMOTE_LOGGING:
   REMOTE_BASE_LOG_FOLDER: str = conf.get('logging', 'REMOTE_BASE_LOG_FOLDER')
   ...
   elif REMOTE_BASE_LOG_FOLDER.startswith('wasb'):
       WASB_CONTAINER: str = conf.get('logging', 'WASB_CONTAINER')
       WASB_REMOTE_HANDLERS: Dict[str, Dict[str, Union[str, bool]]] = {
               'task': {
                       'class': 'airflow.providers.microsoft.azure.log.wasb_task_handler.WasbTaskHandler',
                       'formatter': 'airflow',
                       'base_log_folder': str(os.path.expanduser(BASE_LOG_FOLDER)),
                       'wasb_log_folder': REMOTE_BASE_LOG_FOLDER,
                       'wasb_container': WASB_CONTAINER,
                       'filename_template': FILENAME_TEMPLATE,
                       'delete_local_copy': False,
               },
       }

       DEFAULT_LOGGING_CONFIG['handlers'].update(WASB_REMOTE_HANDLERS)
```

Auth (LDAP)

- LDAP and other auth provider are configured in /opt/airflow/webserver_config.py
- it's use Flask appbuilder and flask-LDAP
- you have to install flask LDAP dependency in airflow image
  - `apt-get install -y build-essential gcc libsasl2-dev libldap2-dev libssl-dev`
  - `pip install --user python-ldap`
- store webserver_config.py in secret and use subpath to mount it

```
volumeMounts:
 - name: airflow-webserver-config
   mountPath: /opt/airflow/webserver_config.py
   subPath: config.py
```

```
volumes:
 - name: airflow-webserver-config
   secret:
     secretName: airflow-webserver-config
```

Liveness

- Webserver
  - Liveness on /health
  - /health always return 200 even if DB or Scheduler are down
- Scheduler
  - Custom heartbeat dag

```
livenessProbe:
 httpGet:
   path: /health
   port: 8080
 periodSeconds: 600
 initialDelaySeconds: 240
```

Scheduler Liveness

- create a DAG that update a timestamp in DB
- check that last timestamp in DB is less than delay
- we can't rely on airflow healthcheck
  - it's depends on webserver
  - it don't test that the scheduler can schedule dag
- It's recommended by [airflow documentation]
- may fail if you reach pool limit

```
#!/usr/local/bin/python
from sqlalchemy.exc import ProgrammingError
from airflow import settings

session = settings.Session()
session.execute("INSERT INTO healthcheck(worker_heartbeat_ts) VALUES (current_timestamp);")
session.commit()
session.close()
```

```
from datetime import datetime, timedelta
from airflow.models import DAG
from airflow.operators.bash import BashOperator

"""
Updates timestamp in the airflow db, healthcheck table
"""

default_args = {
   'owner': 'operations',
   'depends_on_past': False,
   'execution_timeout': timedelta(minutes=1),
}

dag = DAG('zz-airflow-heartbeat-dag',
          default_args=default_args,
          schedule_interval=timedelta(minutes=10),
          max_active_runs=1,
          start_date=(datetime.now() - timedelta(days=1)),
          is_paused_upon_creation=False,
          catchup=False)

update_task = BashOperator(
   task_id='update_timestamp',
   bash_command="/usr/local/bin/python /opt/airflow/bin/heartbeat.py",
   dag=dag
)
```

Scheduler Liveness

```
livenessProbe:
 exec:
   command:
     - python
     - /opt/airflow/bin/healthcheck.py
 failureThreshold: 1
 initialDelaySeconds: 600
 periodSeconds: 1200
 successThreshold: 1
 timeoutSeconds: 20
```

```
#!/usr/local/bin/python
import sys

from airflow import settings
from datetime import datetime, timezone

# 20 minutes, the maximum number of seconds that we wait for the liveness task to update value in
the database.
# After this number is reached, kubernetes will kill the scheduler pod.
max_execution_delay = 20 * 60
session = settings.Session()

result = session.execute("select max(worker_heartbeat_ts) from healthcheck;").fetchall()
session.commit()
session.close()

if (len(result)) == 0:
  raise Exception('No execution_ts found!')
else:
  most_recent_ts = dict(result[0])['max']

  delta = datetime.now(timezone.utc) - most_recent_ts
  print(f'Last execution was {delta.seconds} seconds ago')

  if delta.seconds > max_execution_delay:  # 20 minutes
     raise Exception(f'Last execution was on {most_recent_ts}. This is {delta.seconds} second ago,
maximum permitted: {max_execution_delay}!')

print("everything is fine")
sys.exit(0)
```

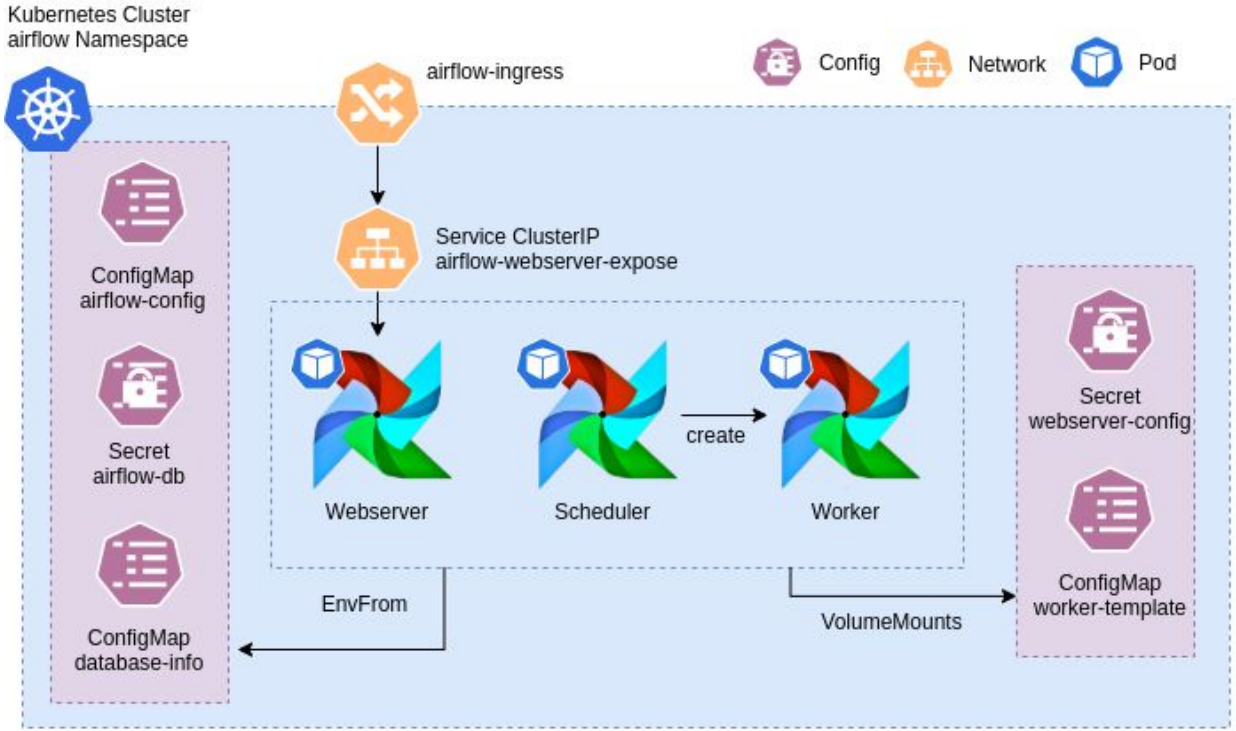
Scheduler Liveness (2)

- You can use scheduler heartbeat
- Don't fail if you reach pool limit
- May not fail when scheduler deadlock

```
livenessProbe:
  initialDelaySeconds: 600
  timeoutSeconds: 5
  failureThreshold: 10
  periodSeconds: 30
  exec:
    command:
      - python3
      - /opt/airflow/bin/healthcheck.py
```

```
#!/usr/local/bin/python
from airflow.jobs.scheduler_job import SchedulerJob
from airflow.utils.db import create_session
from airflow.utils.net import get_hostname
import sys
with create_session() as session:
   job = session.query(SchedulerJob).filter_by(hostname=get_hostname()).order_by(SchedulerJob.latest_heartbeat.desc()).limit(1).first()
sys.exit(0 if job.is_alive() else 1)
```

Airflow on K8S Overview



One More Thing

- use one navbar color by env `AIRFLOW__WEBSERVER__NAVBAR_COLOR`
- Use podAntiAffinity to avoid having scheduler and webserver on the same node if they are small (2 Core 4Go)
- Use Dedicated Nodes for airflow and other static workload (taint or labels with nodeSelector)
- Use autoscalling for airflow worker (taint or labels with nodeSelector)
- You must have a decent machine for your database (2core)

Generate Random secret on helm

```
{{/* 1. Query your secret for "existence" and return in the $secret variable*/}}
{{ $secret_airflow := (lookup "v1" "Secret" .Release.Namespace "airflow-db") -}}
apiVersion: v1
kind: Secret
metadata:
 name: airflow-db
type: Opaque
{{/* 2. If the secret exists, write it back out (it'll be Base64 encoded so used the "data" key)*/}}
{{ if $secret_airflow -}}
data:
 POSTGRES_USER: {{ $secret_airflow.data.POSTGRES_USER }}
 POSTGRES_PASSWORD: {{ $secret_airflow.data.POSTGRES_PASSWORD }}
 POSTGRES_DB: {{ $secret_airflow.data.POSTGRES_DB }}
 FERNET_KEY: {{ $secret_airflow.data.FERNET_KEY }}
{{/* 3. If it doesn't exist ... create it (this time as "stringData" as it will be a raw value) !*/}}
{{ else -}}
stringData:
 POSTGRES_USER: airflow
 POSTGRES_PASSWORD: {{ randAlphaNum 32 | quote }}
 POSTGRES_DB: airflow
 FERNET_KEY: {{ randAscii 32 | b64enc | quote }}
{{ end }}
```

The End

for real
Any Questions?