Apache Airflow

@ Jagex – Anum Sheraz

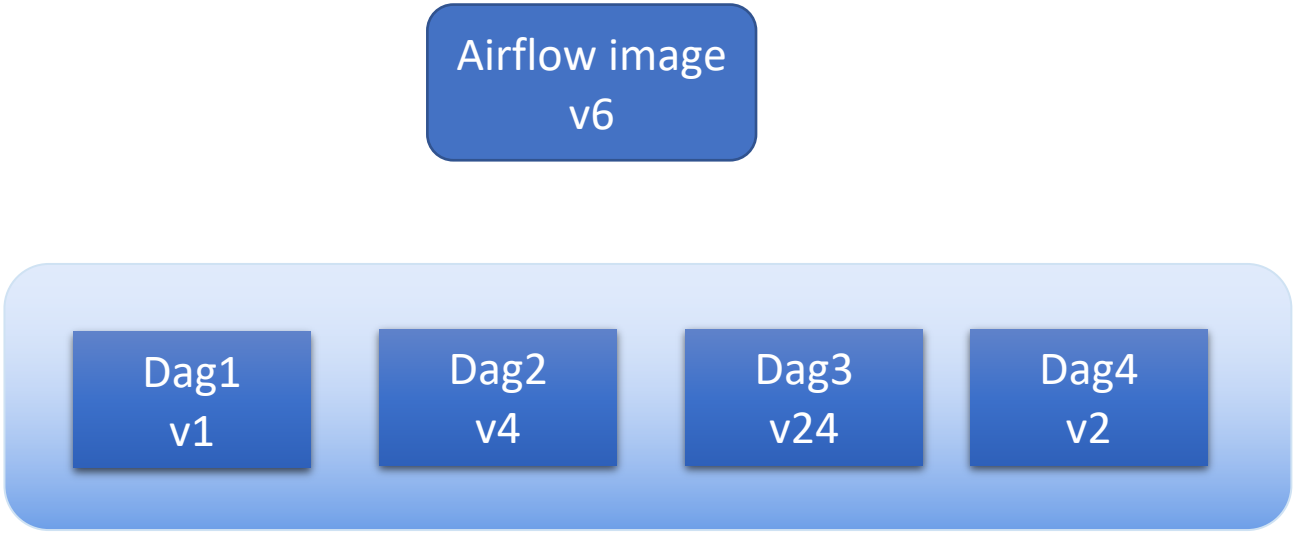
Managing DAGs at scale

Deployment, versioning & package management

Agenda

- Keep track of airflow state
- keep track of DAGs states
- Revert to previous state
- Manging DAGs code base
- Releasing next airflow version
- Automate airflow deployment process
- How to manage DAGs package dependencies

Airflow/Dags States



Airflow Repository

Contains

- Dags code
- Dockerfile
  - Inherit from airflow base image
  - install custom_dags/ into airflow dags/ directory
  - Install additional py packages
  - generate new custom image version *x*

custom_dags/
- dag_1 (v1)
- dag_2 (v3)
- dag_3 (v8)

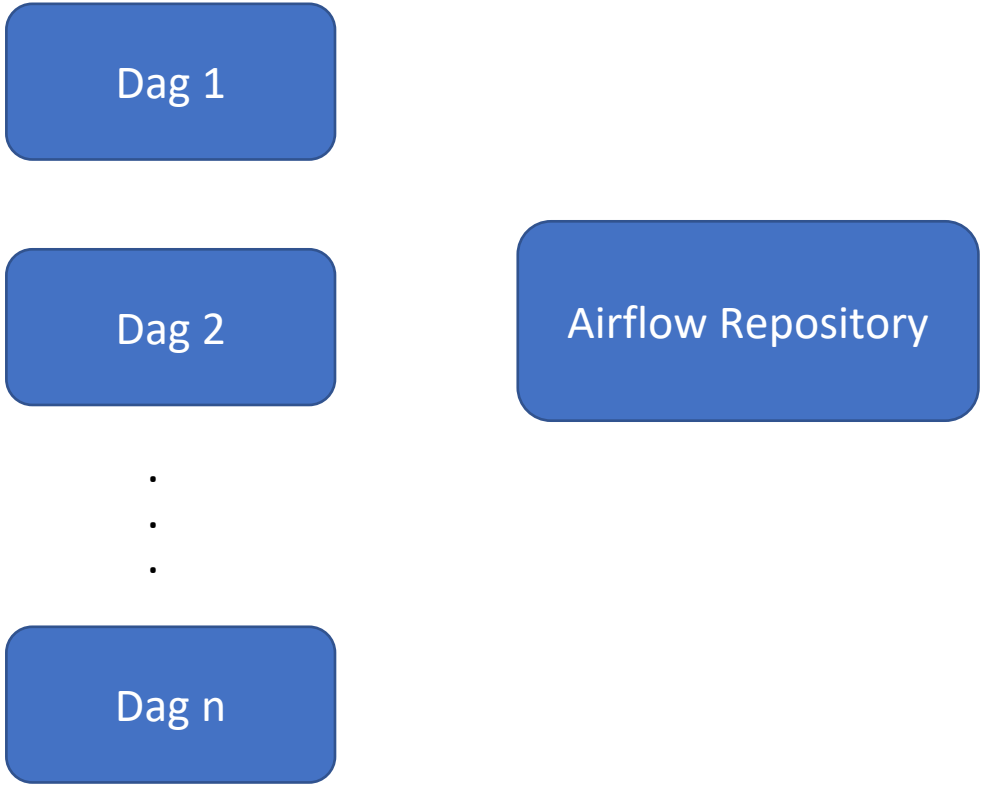
Dockerfile

Problems:

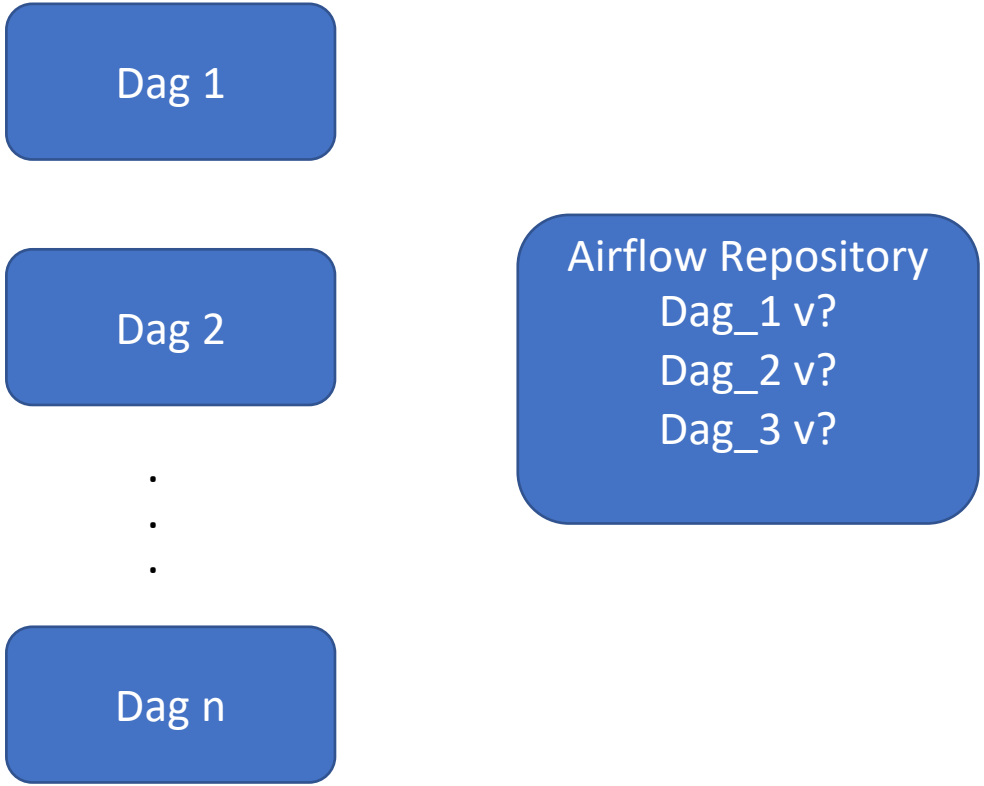
➔ All dags code in single repository

➔ hard to manage

Split Dags repositories



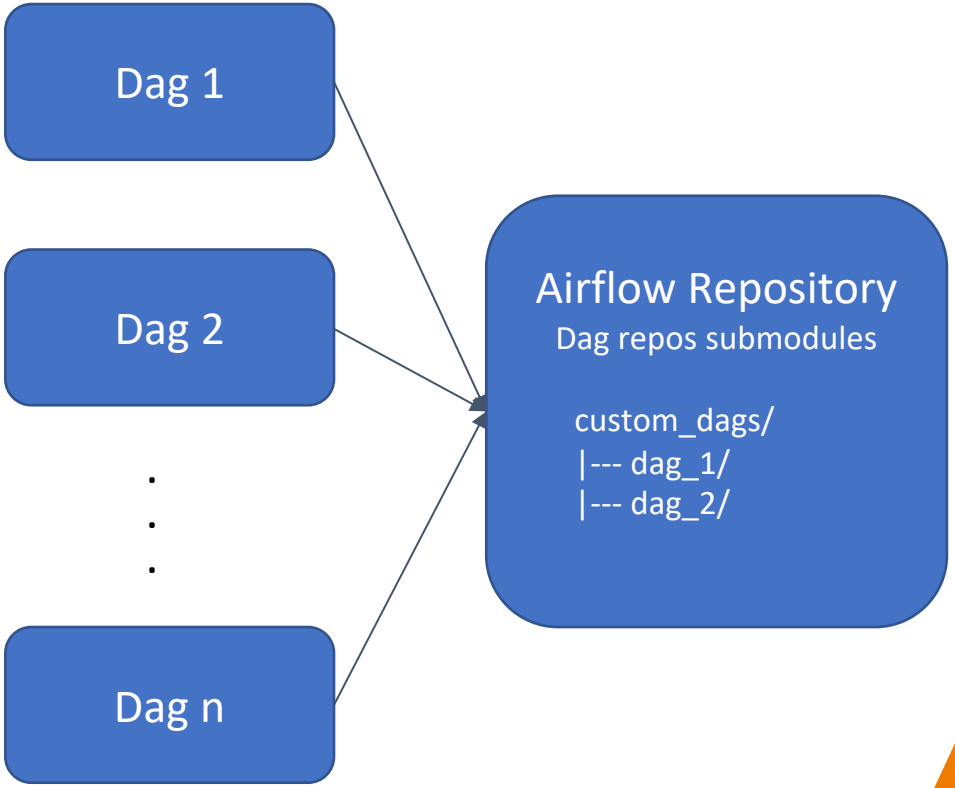
Split Dags repositories



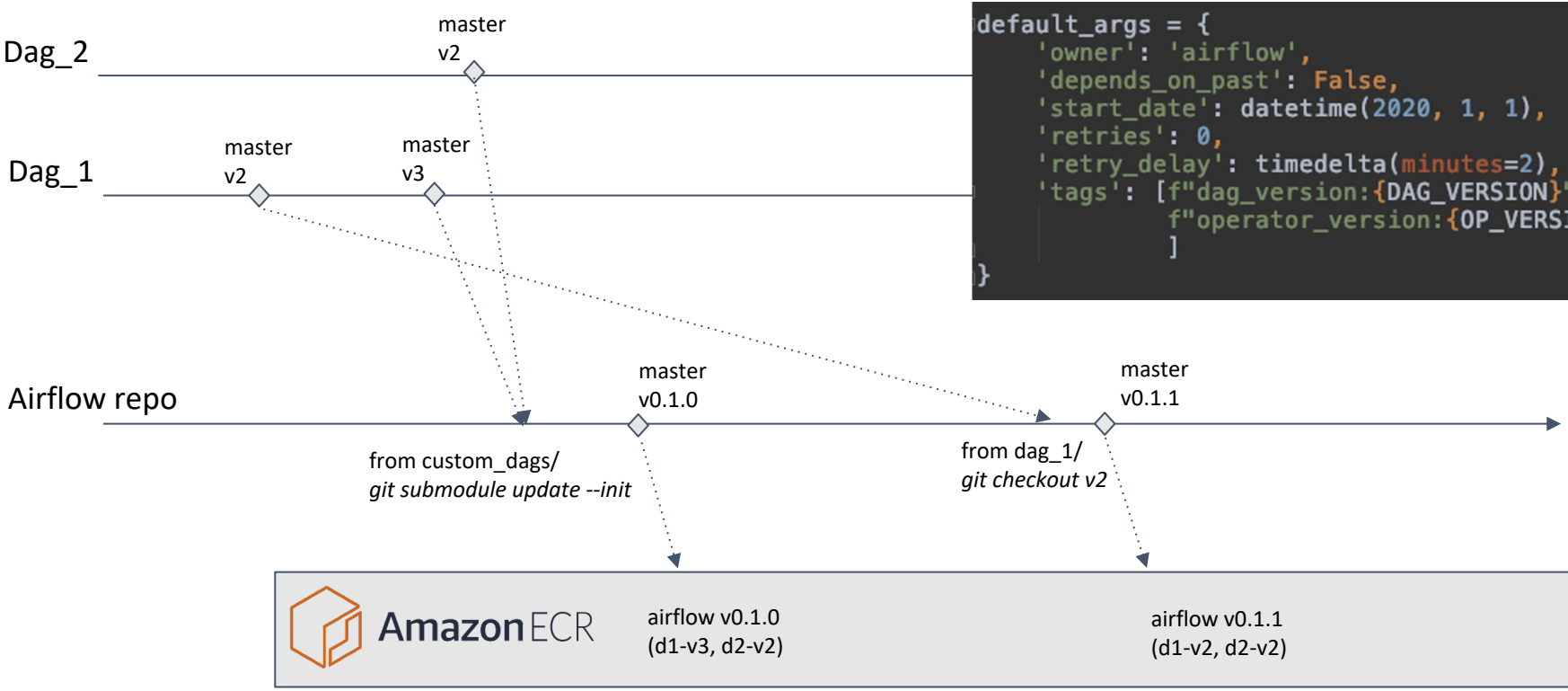
Dags | Git submodules

Airflow Repository

- Git submodule add https://<dag_1_repo>
- Git submodule add https://<dag_2_repo>
- Git submodule add https://<dag_n_repo>



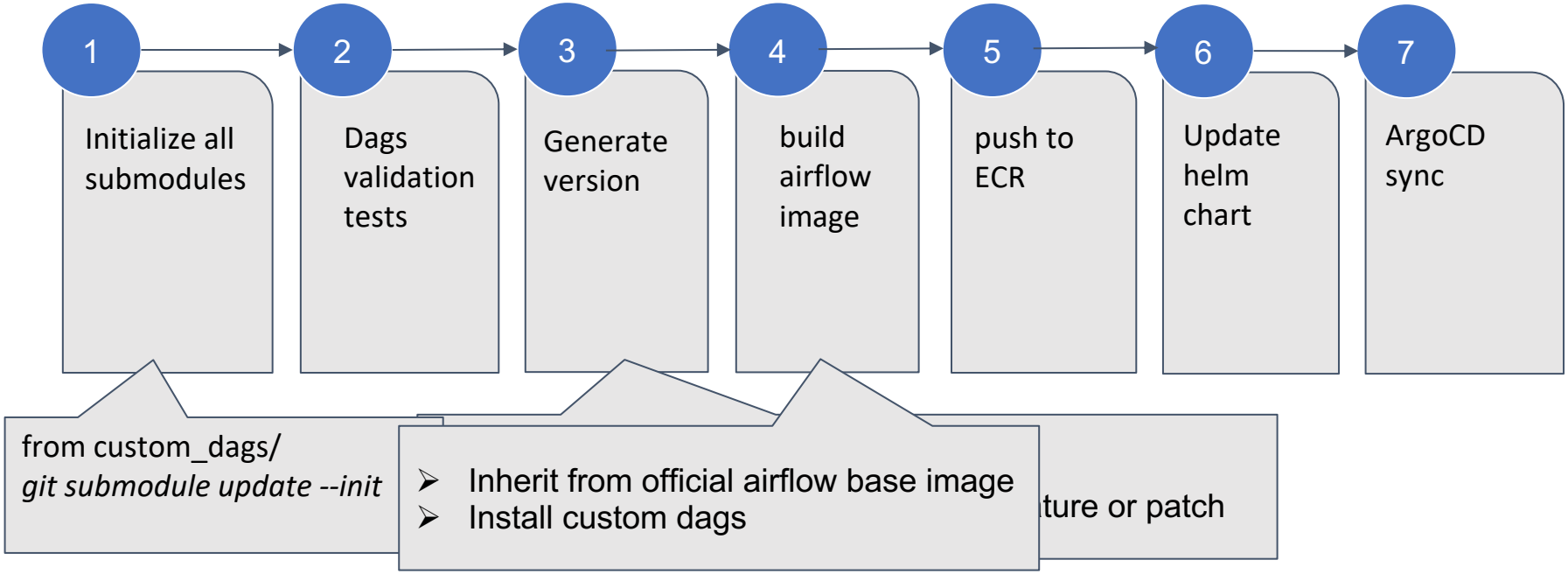
Dags | Git submodules



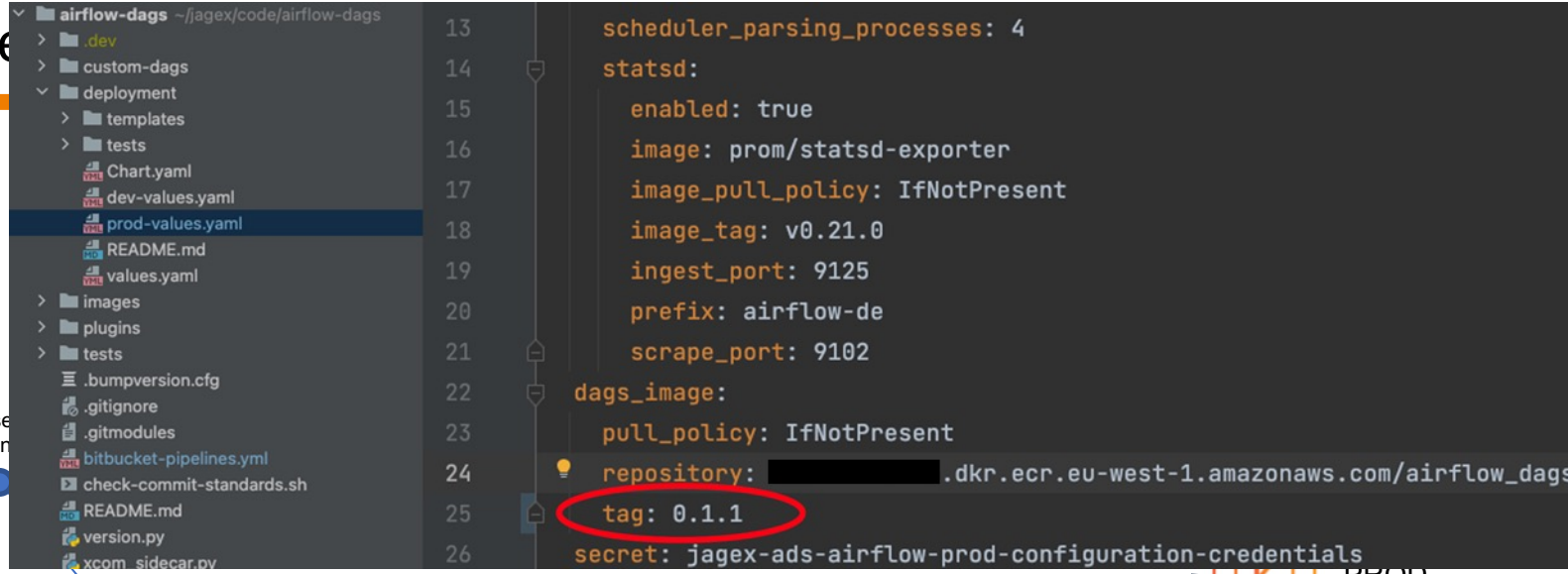

```
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2020, 1, 1),
    'retries': 0,
    'retry_delay': timedelta(minutes=2),
    'tags': [f"dag_version:{DAG_VERSION}",
             f"operator_version:{OP_VERSION}",
             ]
}
```

CICD Pipeline Steps

Pipeline triggers on Airflow repository master branch



Airflow De



release version

v0.1.1

master

feature-branch

Kubernetes manifests

HELM prod-values.yaml

tag: 0.1.1

push to master again

commit from cicd pipeline

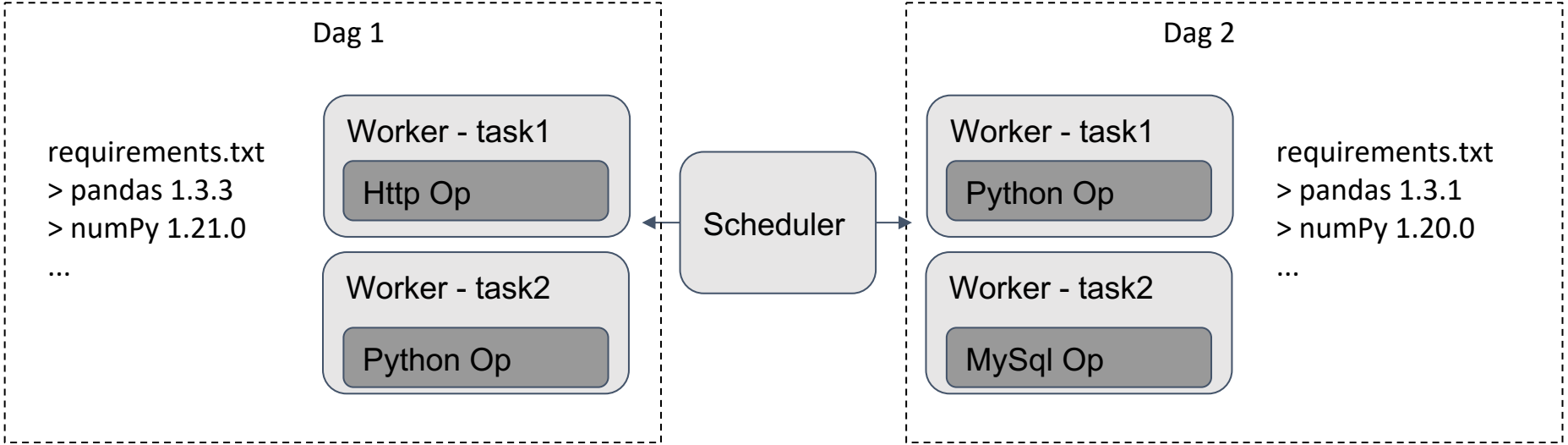
CD sync

ArgoCD

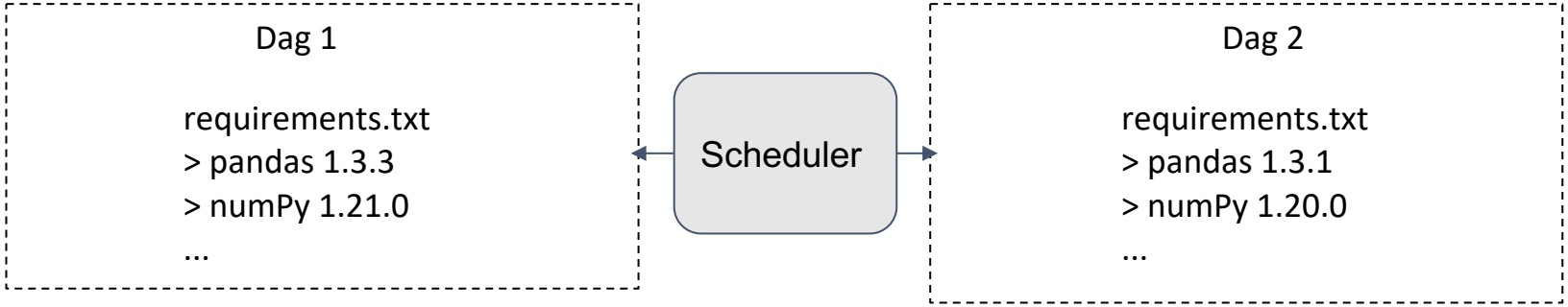
PROD

DEV

Package Dependency management



Package Dependency management | Problems



- Airflow workers also use same airflow image (v0.1.1)
- install packages required by all DAGs into airflow image?
  - Hard to track which package is used by which DAG
  - Hard to cleanup packages while DAG removal/upgrade
- Create virtual environment for each DAG ?
  - ramp-up and ramp-down venvs while DAG starts/stops
  - Slows down the workflow
  - needs extra work to maintain venvs

Package Dependency management | Solution

- Workflow execution: business logics performed inside DAG’s tasks
- Workflow management: Dags and tasks triggering, alerts, monitoring etc

Need Isolation between these two layers… idea ?

1. Build docker container to perform business logics
2. Let airflow run those containers

How ?

- Docker Operator
- Kubernetes Pod Operator

Docker Operator

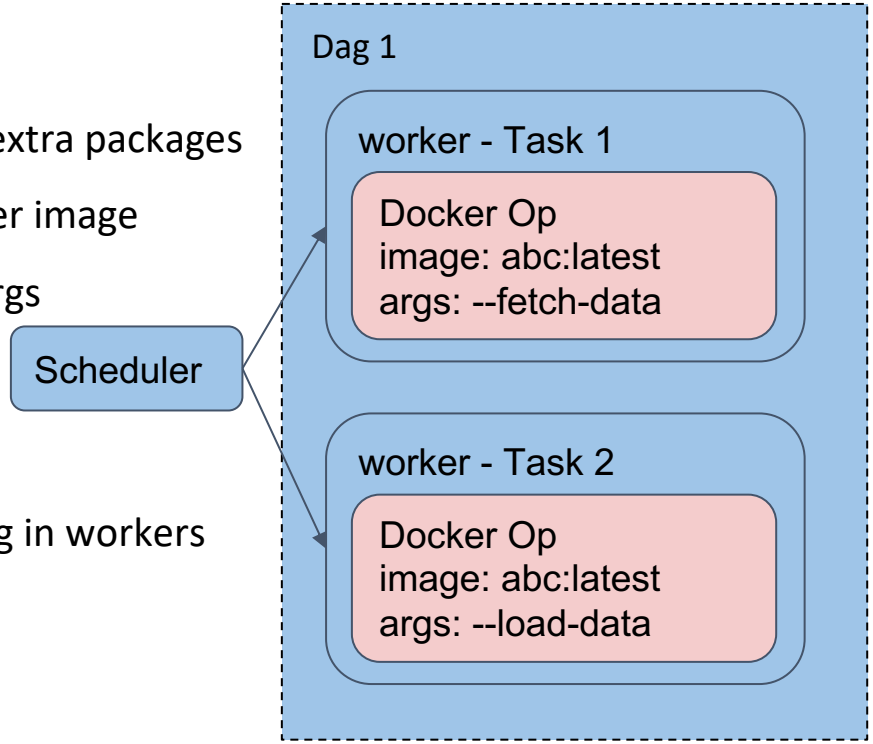
Lets you run docker container within same airflow worker node

Pros:

- not populating airflow image with extra packages
- Tasks packages installed in container image
- Reuse same image with different args
- Not bound to python anymore

Cons:

- resource limited - container running in workers

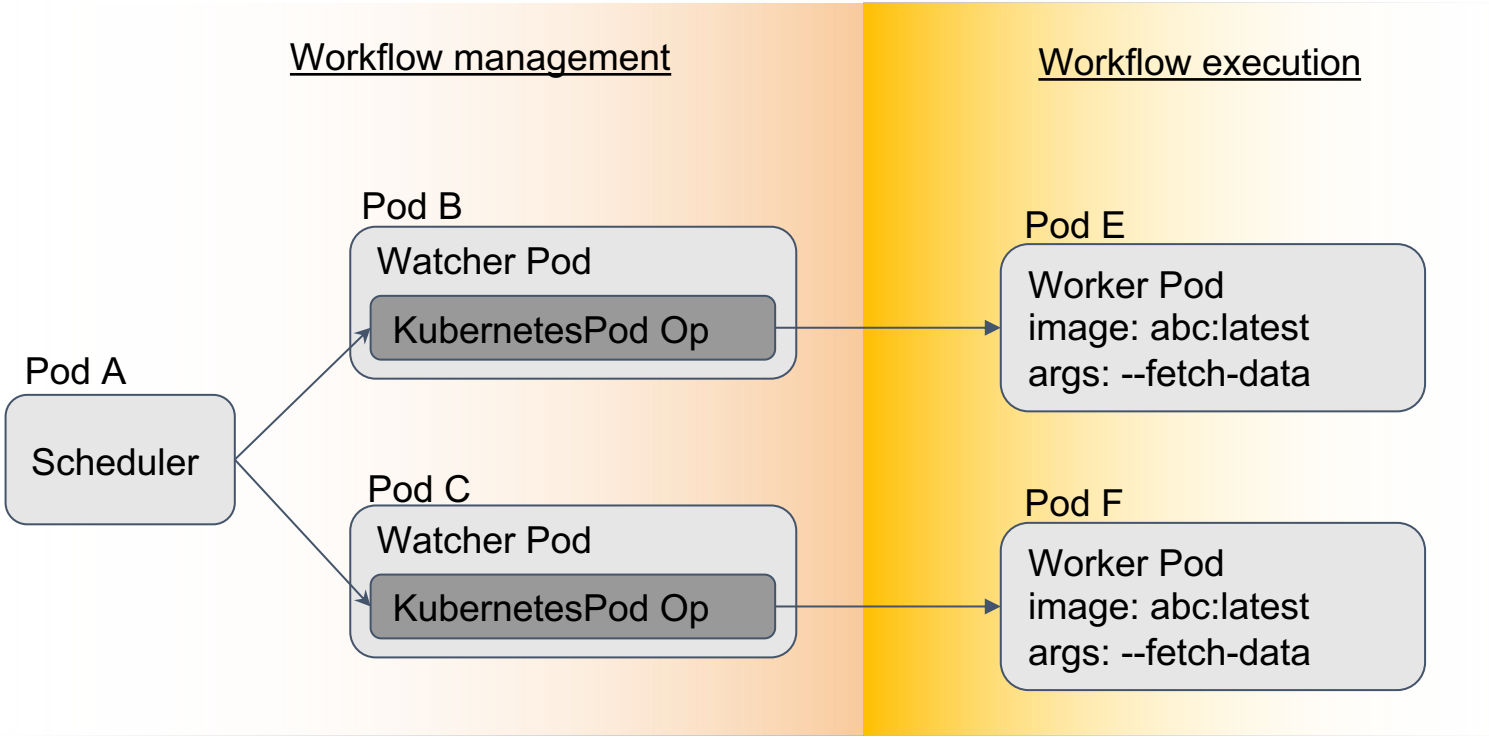


workflow execution

workflow management

Kubernetes Pod Operator

Lets you run docker container on dedicated worker node (pod) in kubernetes cluster



Pros & Cons

Docker Operator

- ✗ resource limited - container running inside workers
- ✓ not populating airflow image with extra packages
- ✓ Tasks packages installed in container image
- ✓ Not bound to python anymore
- ✓ Reuse same image with different args
- ✓ Easy to debug

Kubernetes Pod Operator

- ✗ Kubernetes knowledge required
- ✗ slightly slow - Each task require pair of nodes