Democratized data workflows at scale

Emil Todorov
Mihail Petkov
Our agenda for today

- Why Airflow?
- Architecture
- Security
- Execution environment in Kubernetes
FT is a data driven organization
Time for a change
Why Airflow?
Architecture

- PostgreSQL
- Scheduler Pod
- Web Server Pod
- Worker Pod
Airflow will be used by multiple teams
Airflow requirements

Team 1

Team 2

Team N
Teams will share *Airflow resources*
Airflow shared components
Teams will share Kubernetes resources
Kubernetes shared components
How to evolve this architecture?
Airflow instance per team
One instance components
Instance per team problems

- Adding new team is **hard**
- Maintaining environment per team is **difficult**
- Releasing new features is **slow**
- Resources are **not fully utilised**
- Total **cost increase**
Another way?
Multitenancy
Multiple independent instances in a shared environment
Multi-tenant components
How to make AWS multi-tenant?
IAM Security

Team 1 IAM user

Team 2 IAM user

Team N IAM user
IAM Security

Team 1 IAM user

Team 2 IAM user

Team N IAM user
How to enhance Kubernetes?
How to improve PostgreSQL?
How to extend Airflow?
Redesign Airflow source code
Redesign Airflow source code

- Module per team
Redesign Airflow source code

- Module per team
- Connections per team

class ExtendedConnection(Connection):
    @staticmethod
    def get(conn_id: str) -> str:
        team_id = DAGMetaService.get_team_id_from_dag()
        return team_id + '_' + conn_id
Redesign Airflow source code

- Module per team
- Connections per team
- Extend hooks, operators and sensors
Redesign Airflow source code

- Module per team
- Connections per team
- Extend hooks, operators and sensors
- Use `airflow_local_settings.py`

```python
def policy(task_instance: TaskInstance):
    team_id = get_team_id_from_dag_filepath(task_instance.dag.filepath)
    task_instance.executor_config['KubernetesExecutor']['labels']['team_id'] = team_id

def pod_mutation_hook(pod: Pod):
    team_id = pod.labels.get('team_id')
    pod.namespace = get_team_namespace(team_id)
```
Redesign repository structure

- Airflow system code repository
- Airflow repository
  - Team 1 DAG repository
  - Team 2 DAG repository
  - Team N DAG repository
Execution environment in Kubernetes
ETL

Extract

DATA SOURCE 1

DATA SOURCE 2

Transform

AGGREGATIONS

Load

DATA DESTINATION
Extract

DATA SOURCE 1

DATA SOURCE 2

Transform

AGGREGATIONS

Load

DATA DESTINATION
Load

Extract

DATA SOURCE 1

DATA SOURCE 2

Transform

AGGREGATIONS

Load

DATA DESTINATION
Transform?
Example workflow

Task 1

Task 2

Task 3

Task 4
Our goals

- Language agnostic jobs
- Cross task data access
KubernetesPodOperator
Our goals

- Language agnostic jobs
- Cross task data access
Unique storage pattern

- Unique team name from the multitenancy
- Unique DAG id
- Unique task id per DAG
- Unique execution date per DAG run

/{team}/{dag_id}/{task_id}/{execution_date}
The power of extensibility
ExecutionEnvironmentOperator

KubernetesPodOperator

KUBERNETES
POD OPERATOR
EXECUTE

ExecutionEnvironmentOperator

PRE EXECUTE

KUBERNETES
POD OPERATOR
EXECUTE

POST EXECUTE
Configurable cross task data dependencies
Example input configuration

```python
example_execution_environment_config = {
    "input": {
        "operators": [
            {
                "task_id": "task_1"
            },
            {
                "task_id": "task_2"
            }
        ]
    }
}

example_execution_environment_operator = ExecutionEnvironmentOperator(
    task_id='task_3',
    job_config=example_execution_environment_config,
    image='example_docker_image',
    tag='latest'
)
```
example_execution_environment_config = {
    "output": {
        "default": {
            "destinations": [
                {
                    "type": "s3",
                    "data": {
                        "bucket": "<s3_bucket>",
                        "path": "<s3_path>",
                        "aws_conn_id": "<s3_connection_for_upload>",
                    }
                }
            ]
        }
    }
}

example_execution_environment_operator = ExecutionEnvironmentOperator(
    task_id='task_3',
    job_config=example_execution_environment_config,
    image='example_docker_image',
    tag='latest'
)
Pre-execute

- Bootstrap the environment
- Enrich the configuration
- Export the configuration to the execution environment pod
Post-execute

- Handle the execution
- Clear all bootstraps
- Deal with the output
POC with AWS S3 as intermediate storage

Task 1

Task 2

Task 3

Task 4
Is this efficient?

- Multiple downloads and uploads
- Single processing power
- Always loading the data in memory
How to evolve the execution environment?

- Remove unnecessary data transfers
- Parallelize the processing
- Provide hot data access
Shared file system
Kubernetes persistent volume

Task 1
Task 2
Task 3
Task 4
Kubernetes persistent volume with EFS

Task 1
Task 2
Task 3
Task 4
So far so good

- Remove unnecessary data transfers
- Parallelize the processing
- Provide hot data access
One worker?
Benefits from Spark

- Runs perfectly in Kubernetes
- Supports many distributed storages
- Allows faster data processing
- Supports multiple languages
- Easy to use
SparkExecutionEnvironmentOperator

- PRE EXECUTE
  - SETUP SPARK ENVIRONMENT

- POD OPERATOR EXECUTE
  - RUN SPARK BASED IMAGE

- POST EXECUTE
  - CLEAR SPARK BASED RESOURCES
Spark execution environment

Spark driver

Spark workers
Our current state

- Remove unnecessary data transfers
- Parallelize the processing
- Provide hot data access
Hot & cold data

HOT DATA

COLD DATA

Task 1
Task 2
Task 3
Task 4
Alluxio

Task 1

Task 2

Task 3

Task 4
Thank you!

#apacheairflow