

# 데이터 분산처리 끝내기

- Part4 Airflow

# 이 수업에서 다루고자 하는 것

## 이론

### Chapter 1. Airflow란?

- 워크플로우 오케스트레이션 이해
- Airflow의 역사, 특성
- Airflow의 구성요소
- 싱글 노드 구조, 멀티 노드 구조
- DAG
- Operators 알아보기
- Cloud Managed Service of Airflow
- Luigi, Prefect, Oozie 비교

# 이 수업에서 다루고자 하는 것

## 실습

### Chapter 2. AWS EC2에 Airflow 설치하기

- Airflow 기본적인 설치, 싱글구조 알아보기
- Airflow Webserver 살펴보기
- airflow.cfg 살펴보기
- 클러스터 구성하기1
- 클러스터 구성하기2 master node 설정 끝내기
- 클러스터 구성하기3 DB 설정
- 클러스터 구성하기4 Worker 설정

### Chapter 3. Kubernetes에 Airflow 설치하기

- Rancher란?
- EKS 생성 마무리
- Rancher 둘러보기
- Airflow on K8S 살펴보기
- Sidecar 패턴, Gitsync 설정
- KubernetesPodOperator 사용 설정

# Chapter 1. Airflow란?

# Chapter 1. Airflow란?

## 01. 워크플로우 오케스트레이션

## 01. 워크플로우 오케스트레이션

### Work Flow

- 작업 절차를 통한 정보 또는 업무의 이동
- 작업의 흐름을 의미함
- ETL, ELT 같은 작업이 있을 때 이 작업에 해당하는 순서를 가지는 Task들의 모음
- 이전 워크플로에서 단계는 다음 단계에 연결되어 있음
- 작업의 단계가 개별적이고 단계 간에 데이터 흐름이 있어야 함
- 워크 플로우가 흐른다는 말?

자동화에 대한 기대가 내재되어 있음

## 01. 워크플로우 오케스트레이션

### Work Flow Orchestration

- 작업 흐름의 자동화를 위한 시스템
- 컴퓨터 응용 프로그램 및 서비스를 통해 자동으로 구성, 관리 및 조정
- 오케스트레이션 규칙과 비즈니스 논리를 이해하는 방식으로 데이터 흐름을 관리하는 것을 의미
- 워크플로우 오케스트레이션을 사용하면 코드를 예약, 실행 및 관찰 할 수 있는 워크플로로 전환이 가능함
- 좋은 Workflow Orchestration의 조건?

## 01. 워크플로우 오케스트레이션

### Work Flow Orchestration

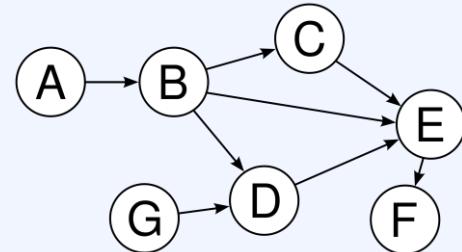
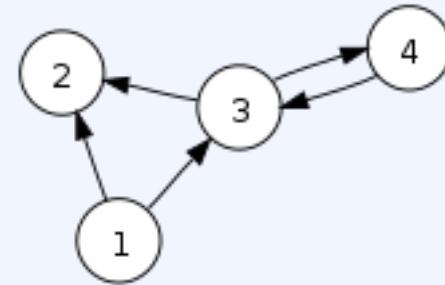
- Task 간 데이터 전달
- ad-hoc 한 실행 트리거
- 사용자 지정 일정 할당
- 알람 - 작업 실패 시 alert
- 실패 시 재시도 및 복구
- 캐싱을 통한 비용이 많이 드는 재계산 방지
- Task가 잘못될 것이라고 가정할 때 올바른 워크플로 도구는 이러한 "잘못된 Task" 를 빠르고 간단하게 처리하고 문제를 해결하는 방법을 안내해야 함(Jeremiah Lowin)

## 01. 워크플로우 오케스트레이션

### Work Flow Orchestration

#### Data Pipeline, Graph

- Work Flow의 태스크 간에는 작업 간 종속성이 발생하기 마련
- 이 종속성을 명확하게 만드는 방법 중 하나는 그래프를 활용하는 것
- 노드와 엣지(방향성) -> Directed Graph
- DAG(Directed Acyclic Graph)를 활용, 한 방향성
- 양방향성을 가진다면 명확한 실행 Path가 없게 됨



## 01. 워크플로우 오케스트레이션

### Work Flow Orchestration

#### 대표적인 툴

- Airflow
- Dagster
- Argo
- Prefect
- Luigi
- Oozie
- ...

# Chapter 1. Airflow란?

## 02. Airflow의 역사, 특성

## 02. Airflow의 역사, 특성

### Airflow

- Airbnb에서 2014년 10월 시작된 오픈소스 프로젝트
- 2015년 6월 발표
- 2016년 3월에 Apache Software Foundation에 인큐베이팅
- 2019년 1월 아파치 재단에서 Top Tier 프로젝트로 발표하면서 급속도로 성장  
(<https://news.apache.org/foundation/entry/the-apache-software-foundation-announces44>)

## 02. Airflow의 역사, 특성

### Airflow

- Python 기반으로 짜여져 있어서 쉽게 사용이 가능함
- Python Library들을 활용해서 Work flow를 구성할 수 있음
- Python 프레임워크를 사용하면 거의 모든 기술과 연결되는 워크플로를 구축 가능
- 배치 지향 워크플로를 개발, 스케줄링 및 모니터링하기 위한 오픈 소스 플랫폼
- 단일 프로세스부터 대규모 워크플로를 지원하는 분산 설정까지 다양한 방식으로 배포가 가능

## 02. Airflow의 역사, 특성

### Airflow 특징

- **Dynamic:**

Airflow 파이프라인은 Python코드로 구성되어 동적 파이프라인 생성이 가능

- **Extensible:**

Operator, Executor를 통해 사용자 환경에 적합하도록 확장해 사용이 가능

- **Elegant:**

파이프라인이 간결하고 명시적, 스크립트 Parameter는 Jinja 템플릿 엔진을 사용

- **Scalable:**

분산구조와 메세지 큐를 이용해 scale out과 워커 간의 오케스트레이션을 지원

- <https://airbnb.io/projects/airflow/>

## 02. Airflow의 역사, 특성

### Airflow 왜 사용해야 할까?

Airflow(**workflow orchestration**)가 없다면?

- 데이터를 불러오고 일련의 처리를 하고 저장을 해야하는 상황
- 세 가지의 다른 태스크를 실행하는데 있어서 각각의 스크립트를 실행해줘야 함
- 작업 간의 의존성 부여 불가
- 실행 시간과 실행 주기 설정, Crontab?
- 실제로 어떻게 처리되고 있는지에 대해서도 파악하기 힘듬
- 이것을 DAG라는 그래프 방식으로 쉽게 보여주고 의존성을 붙일 수도 있음

## 02. Airflow의 역사, 특성

### Airflow 장점과 단점

#### 장점?

- 쉽게 새로운 기술로 확장할 수 있다
- 다양한 Operator를 사용할 수 있음
- Python을 통한 확장성
- 스케줄러를 통한 파이프라인의 정기적인 실행
- 개별 태스크들에 대해 자세히 확인할 수 있고 시간에 따른 파이프라인 상황 확인 가능
- 로그를 확인하고 태스크를 관리 가능
- 커뮤니티가 굉장히 활성화 되어 있음, 문제에 대한 답을 쉽게 찾을 수 있고 토론이 가능
- 빠르게 업데이트가 이루어짐

## 02. Airflow의 역사, 특성

### Airflow 장점과 단점

#### 단점?

- 배치처리에 사용할 수 있지만, 스트리밍에는 사용이 불가능함
- 데이터 파이프라인 코드를 작성하는데 있어서 test case 작성하는 것이 어려움
- Python으로 구성되어 있어서 문법이 익숙하지 않다면, DAG 작성이 어려움
- 설치 후 환경 하나의 변경에도 민감한 경우가 있음
- 대규모 작업 처리를 Airflow로만 처리하는 것은 지양
- 업데이트가 너무 빨라서 DAG작성 법이 달라지는 경우도 발생

# Chapter 1. Airflow란?

## 03. Airflow의 구성요소

## 03. Airflow의 구성요소

### Airflow main components

- Airflow Webserver :

파이프라인, DAG를 시각화해주고 사용자에게 메인 인터페이스를 제공

- Airflow Scheduler :

DAG들을 파싱하고 스케줄을 확인, 워커들에게 작업을 할당

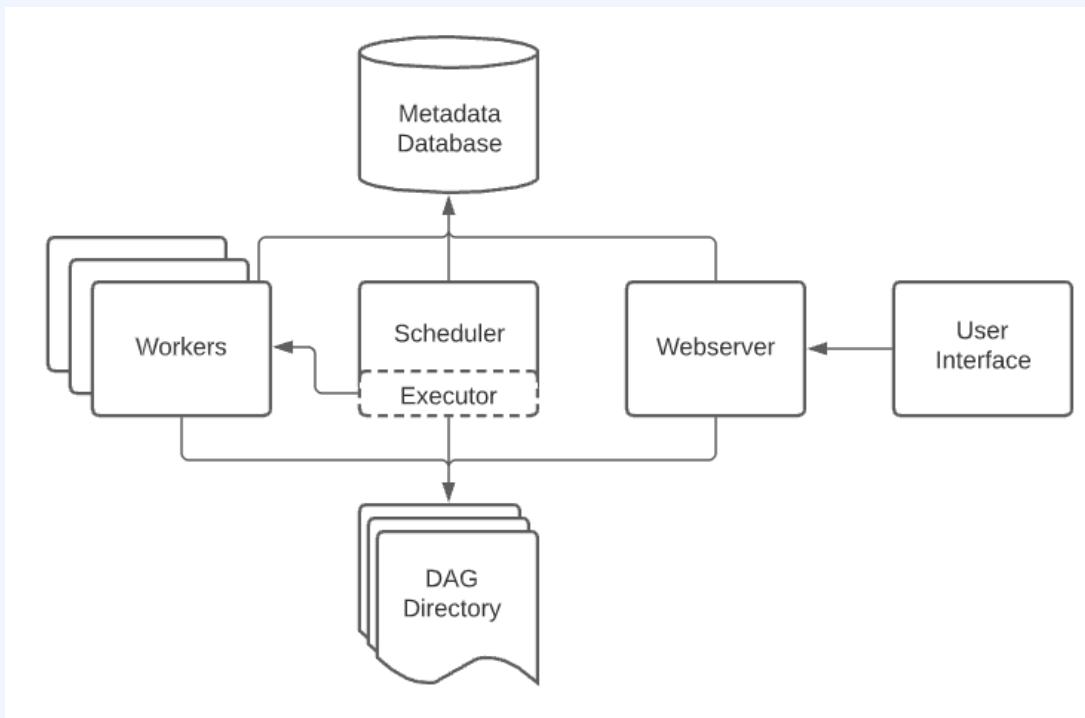
- Airflow Workers :

스케줄링된 작업들을 가져와서 실행

- DB :

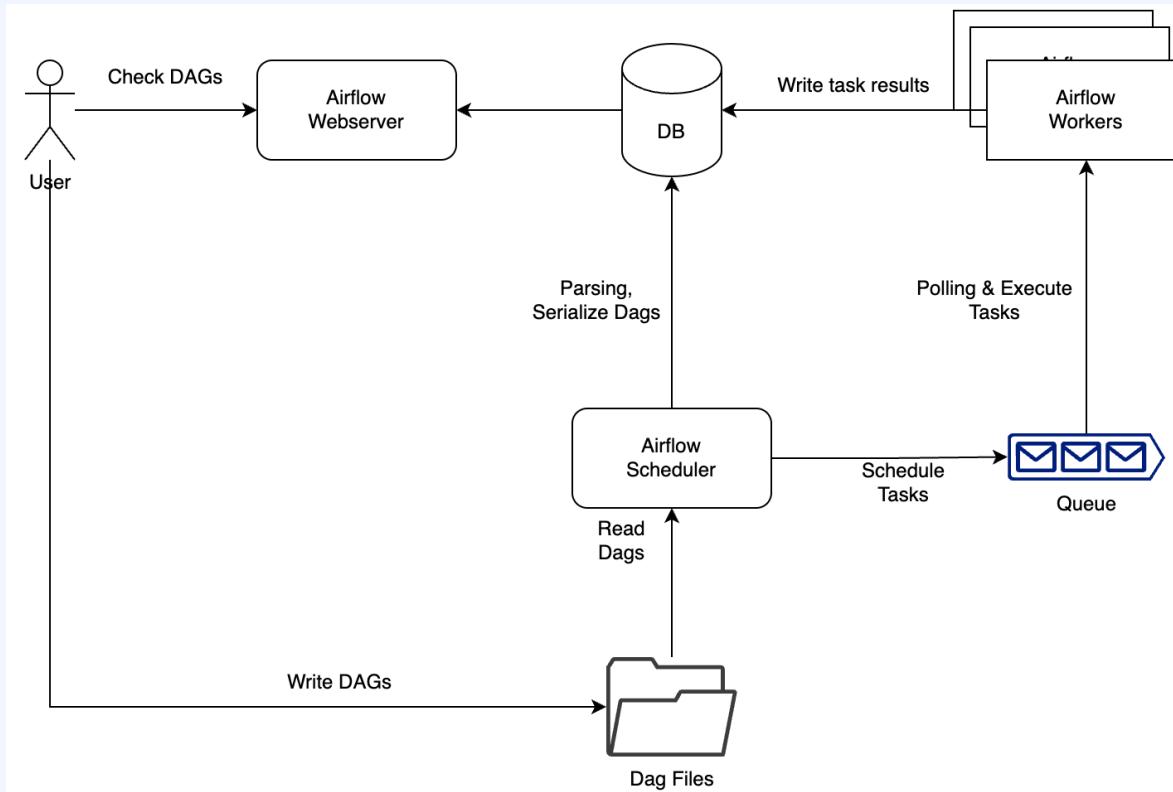
작업 실행 내용들과 기타 Variables, Connections를 저장

## 03. Airflow의 구성요소



<https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>

## 03. Airflow의 구성요소



## 03. Airflow의 구성요소

### Airflow Webserver

- Airflow의 Metastore(DB)에 저장된 로그를 시각화
- 스케줄러에 의해 파싱된 DAG들을 UI를 통해 제공
- DAG들이 실행되는 상황과 결과들을 확인이 가능
- Connections, Variables 설정, DAG 제어

## 03. Airflow의 구성요소

The screenshot shows the Airflow web interface with the title "DAGs". The top navigation bar includes links for "DAGs", "Security", "Browse", "Admin", and "Docs", along with a timestamp "21:11 UTC" and a user icon "RH".

The main content area displays a table of DAGs. The table has columns: "DAG", "Owner", "Runs", "Schedule", "Last Run", "Recent Tasks", "Actions", and "Links".

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator example example2	airflow	2	0 0 * * *	2020-10-26, 21:08:11	6	[Actions]	[Links]
example_branch_dop_operator_v3 example	airflow	0	/1 * * * *		0	[Actions]	[Links]
example_branch_operator example example2	airflow	1	@daily	2020-10-23, 14:09:17	11	[Actions]	[Links]
example_complex example example2 example3	airflow	1	None	2020-10-26, 21:08:04	37	[Actions]	[Links]
example_external_task_marker_child	airflow	1	None	2020-10-26, 21:07:33	2	[Actions]	[Links]
example_external_task_marker_parent	airflow	1	None	2020-10-26, 21:08:34	1	[Actions]	[Links]
example_kubernetes_executor example example2	airflow	0	None		0	[Actions]	[Links]
example_kubernetes_executor_config example3	airflow	1	None	2020-10-26, 21:07:40	5	[Actions]	[Links]
example_nested_branch_dag example	airflow	1	@daily	2020-10-26, 21:07:37	9	[Actions]	[Links]
example_passing_params_via_test_command example	airflow	0	/1 * * * *		0	[Actions]	[Links]

## 03. Airflow의 구성요소

Airflow DAGs Datasets Security Browse Admin Docs 10:12 UTC AA

### DAG: demo

Grid Graph Calendar Task Duration Task Tries Landing Times Gantt Details Code Audit Log

10 / 06 / 2022, 10:12:28 AM 25 All Run Types All Run States Clear Filters

deferred failed queued running scheduled skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

Auto-refresh

Duration: May 14, 00:00 - May 24, 00:00 - Jun 03, 00:00

hello airflow

DAG: demo Run: 2022-05-30, 00:00:00 UTC Task: hello

Task Instance Details Rendered Template Log XCom List Instances, all runs Filter Upstream

Details Logs

Task Actions

Ignore All Dps Ignore Task State Ignore Task Dps Run

Past Future Upstream Downstream Recursive Failed Clear

Past Future Upstream Downstream Mark Failed

Past Future Upstream Downstream Mark Success

Status: success

Task ID: hello

Run ID: scheduled\_2022-05-29T00:00:00+00:00

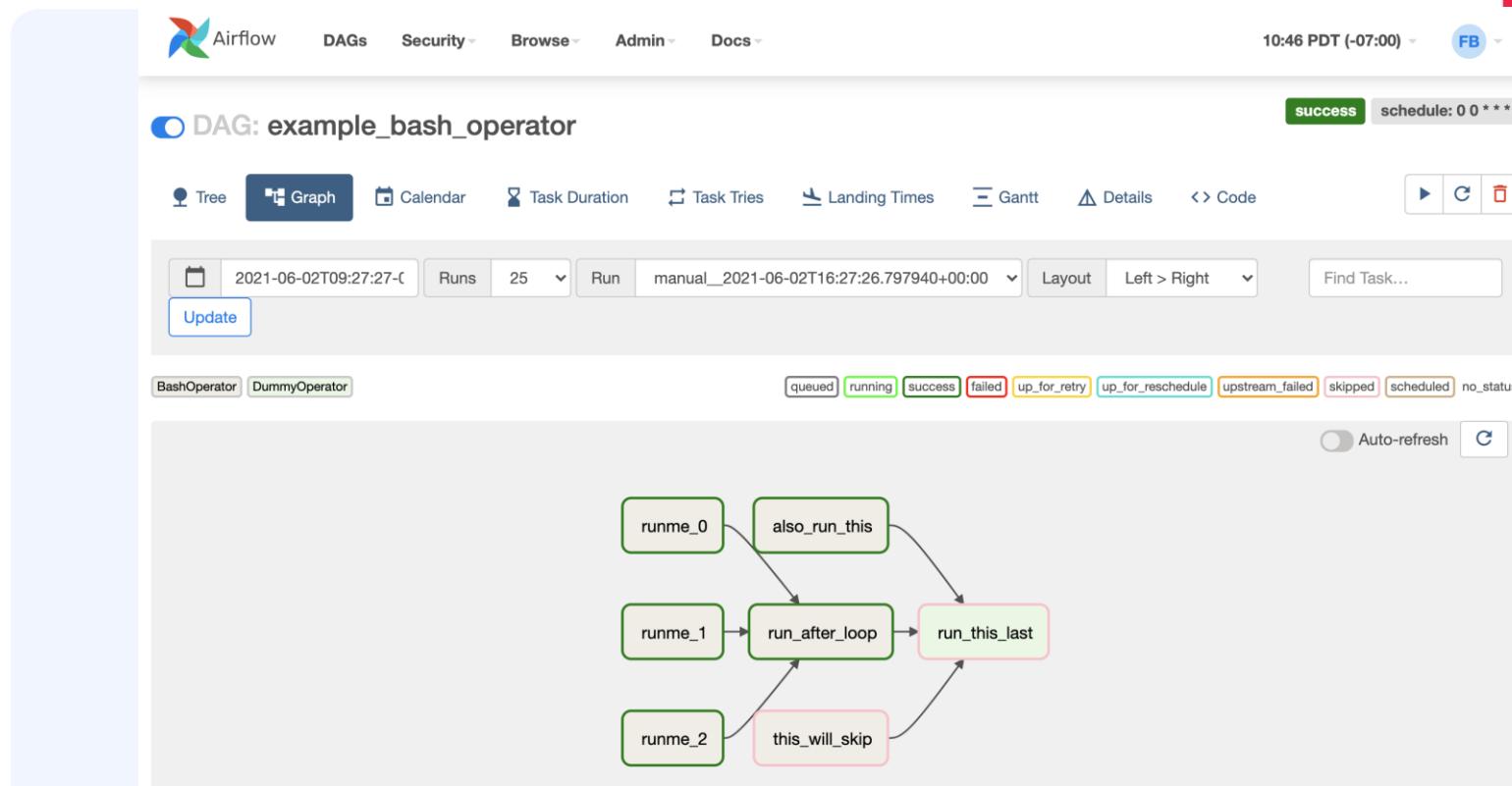
Operator: BashOperator

Duration: 00:00:01

Started: 2022-10-06, 10:12:26 UTC

Ended: 2022-10-06, 10:12:27 UTC

## 03. Airflow의 구성요소



## 03. Airflow의 구성요소

### Airflow Scheduler

- DAG들을 파싱해서 스케줄된 작업들의 작업 간격을 확인
- 실제로 작업 명령을 워커(또는 큐)에 전달
- DAG에 걸려있는 의존성을 확인하고 걸려있다면, 실행 큐에 더해 줌
- 스케줄러는 지정된 DAG 디렉터리의 모든 DAG를 모니터링
- 스케줄러는 Airflow의 심장과 같은 역할을 수행, DAG와 직접적으로 붙어 있는 요소  
지속적으로 DAG 파일을 파싱하고 데이터베이스의 DAG와 동기화  
실행할 작업을 지속적으로 예약

## 03. Airflow의 구성요소

### Airflow Workers

- 다른 요소와는 달리 WorkerS
- 하나 이상의 워커들을 사용할 수 있음
- 실제 운영 상에서는 큐 상태를 확인하고 큐가 밀린다면 워커들을 늘려서 지연을 해소
- 큐에 들어있는 작업에 대해서 큐 파라미터 명을 확인하고 작업을 할당, 실행
- 특정 큐는 특정 워커에서 실행이 가능함
- 워커에서 실행하고 실행 후 나온 로그들을 회수
- 로그들은 각 구성요소 별로 쌓이는데 워커 로그가 삭제되면 태스크 작업내역 확인 불가

# Chapter 1. Airflow란?

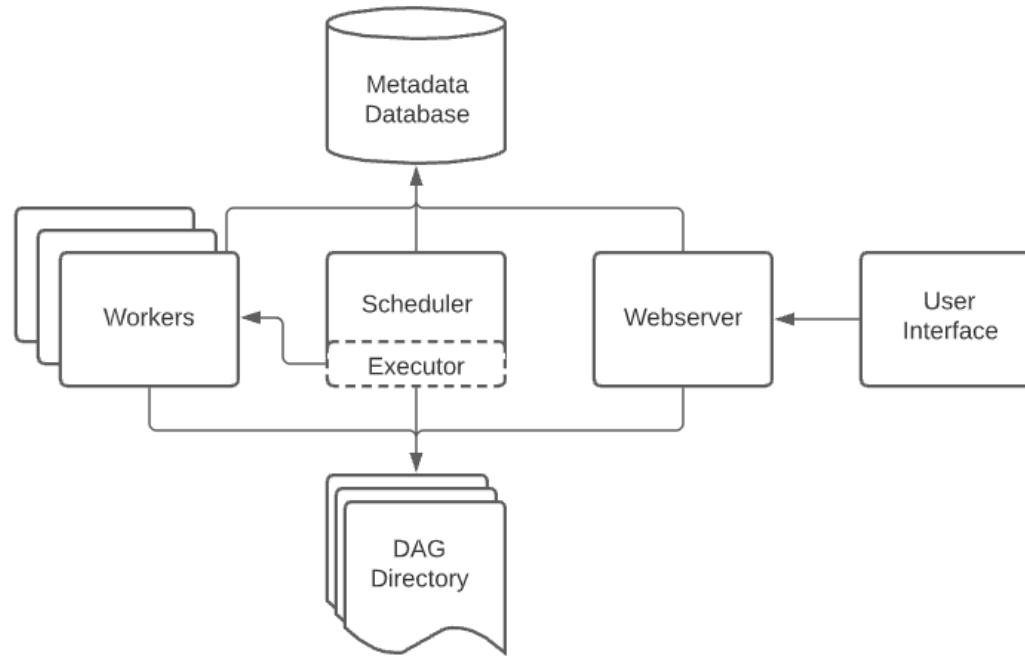
## 04. 싱글 노드 구조, 멀티 노드 구조

## 04. 싱글 노드 구조, 멀티 노드 구조

### Airflow 싱글 노드 구조

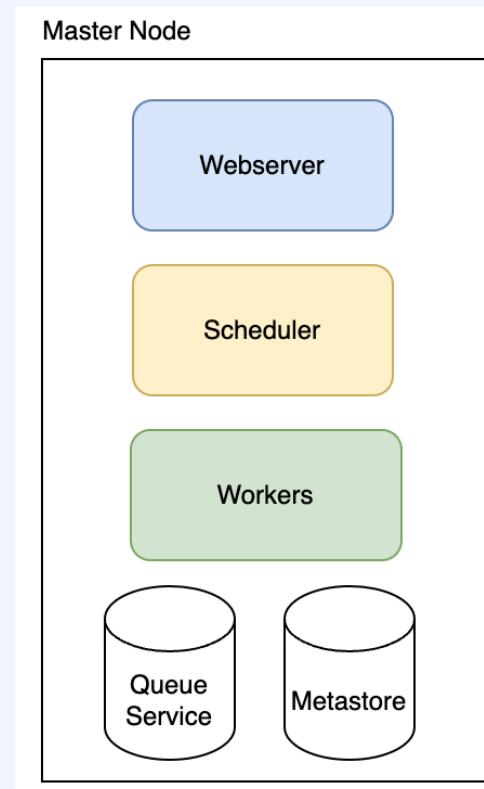
- Airflow를 pip install로 설치했을 때의 구조
- 한 서버 내(마스터 노드)에 Airflow의 구성요소가 모두 있음  
Webserver, Scheduler, Worker, DB
- Airflow의 기능을 간단하게 확인하고자 할 때 사용
- LocalExecutor 사용
- 운영 용으로 사용하기에는 무리가 있음

## 04. 싱글 노드 구조, 멀티 노드 구조



<https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>

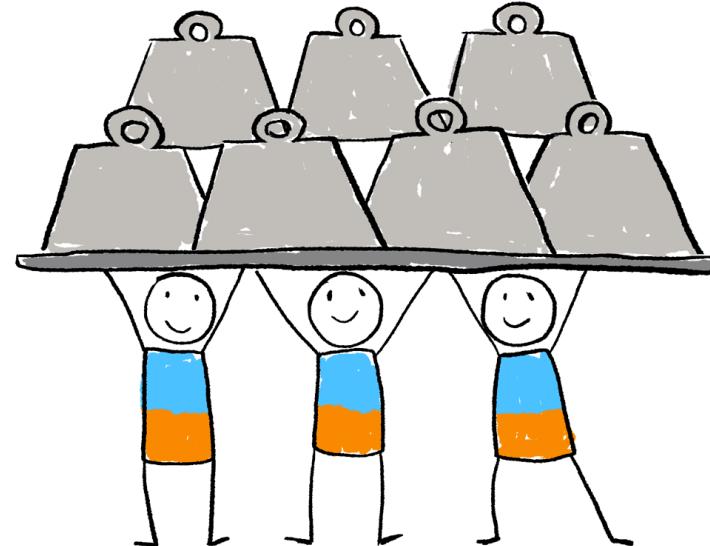
## 04. 싱글 노드 구조, 멀티 노드 구조



## 04. 싱글 노드 구조, 멀티 노드 구조

### Airflow 멀티 노드 구조

- 작업 부하가 점점 커진다면?



## 04. 싱글 노드 구조, 멀티 노드 구조

### Airflow 멀티 노드 구조

- 워커에 오토스케일링을 적용
- 작업 부하가 커지면 워커를 스케일 아웃해 처리, 부하가 줄어들면 스케일 인
- 이 구조를 위해서는 CeleryExecutor를 활성화 해야함
- Celery?

Python으로 작성된 분산 메시지 전달을 기반으로 한 비동기 작업 큐,

Worker의 한 종류

별도로 실행 중인 Worker Process가 Message Broker(Queue)로부터 Message를 전달 받아  
작업을 대신 수행해 주는 라이브러리

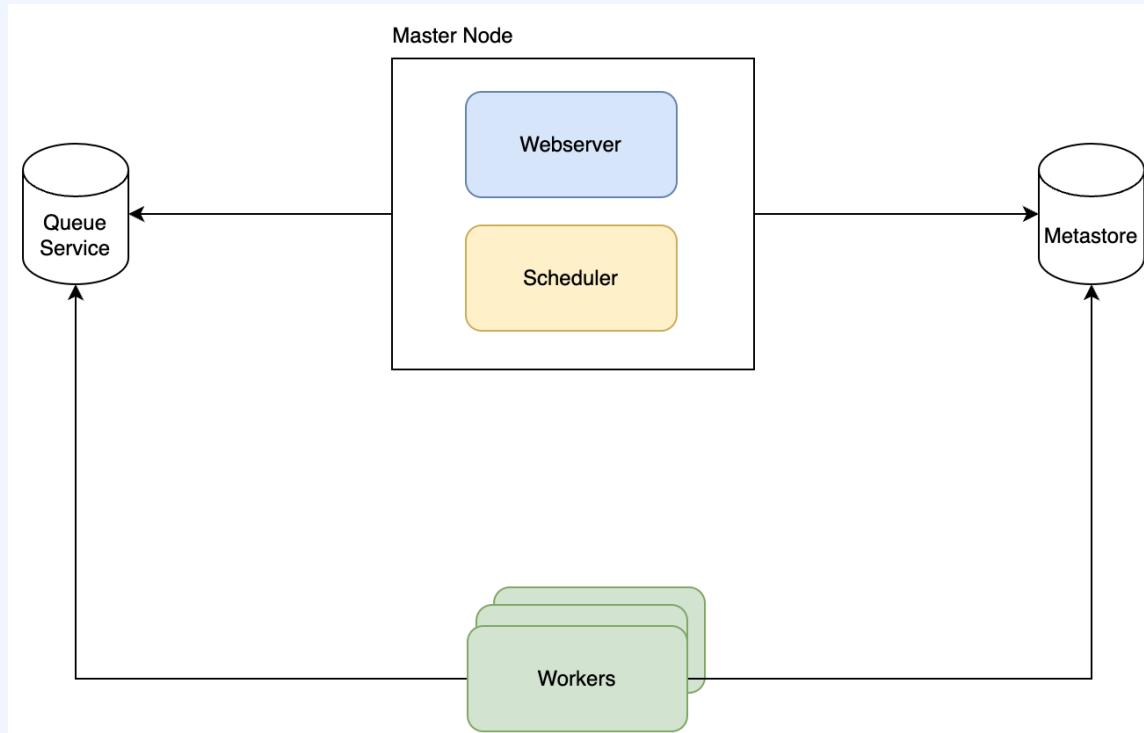
## 04. 싱글 노드 구조, 멀티 노드 구조

### Airflow 멀티 노드 구조

- CeleryExecutor를 활성화 하기 위해서는 Celery 백엔드 구성이 필요
- Redis VS RabbitMQ
- Redis: In Memeory 방식이며 key-value 데이터 구조 스토어  
빠른 Read, Write 성능을 보장
- RabbitMQ: 메세지 브로커로 메시지의 우선순위를 지원 크고 복잡한 메시지를 다룰 때 적합
- Airflow의 브로커로 어떤 걸 사용할지는 현재 서비스 할 비즈니스 프로세스에 따라 판단
- 속도가 더 중요하다고 생각된다면 In Memory 방식의 Redis
- 복잡한 메세지 처리가 필요하면 RabbitMQ  
Redis를 더 많이 사용하는 추세

## 04. 싱글 노드 구조, 멀티 노드 구조

### Airflow 멀티 노드 구조



# Chapter 1. Airflow란?

## 05. DAG

## 05. DAG

### Airflow DAG

- DAG(Directed Acyclic Graph)로 단방향의 그래프
- DAG(Directed Acyclic Graph)는 관계와 의존성을 가진 작업들의 집합
- DAG는 파이썬으로 작성하고 코드를 통해 DAG 구조를 나타냄
- Python을 사용해 DAG를 잘 구성하는 것이 Airflow의 핵심
- 작성한 파일은 DAG\_FOLDER 환경 안에 있어야 스케줄러가 파싱 시작

## 05. DAG

### Airflow Tutorial DAG

```
from datetime import datetime

from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator

# A DAG represents a workflow, a collection of tasks
with DAG(dag_id="demo", start_date=datetime(2022, 1, 1), schedule="0 0 * * *") as dag:

    # Tasks are represented as operators
    hello = BashOperator(task_id="hello", bash_command="echo hello")

    @task()
    def airflow():
        print("airflow")

    # Set dependencies between tasks
    hello >> airflow()
```

## 05. DAG

### Airflow Tutorial DAG

```
from datetime import datetime

from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator

# A DAG represents a workflow, a collection of tasks
1. with DAG()
2. @dag
3. dag=
    # Tasks are represented as operators
    hello = BashOperator(task_id="hello", bash_command="echo hello")

    @task()
    def airflow():
        print("airflow")

# Set dependencies between tasks
    hello >> airflow()
```

## 05. DAG

### Airflow Tutorial DAG

```
import datetime

from airflow import DAG
from airflow.operators.empty import EmptyOperator

my_dag = DAG(
    dag_id="my_dag_name",
    start_date=datetime.datetime(2021, 1, 1),
    schedule="@daily",
)
EmptyOperator(task_id="task", dag=my_dag)
```

```
import datetime

from airflow.decorators import dag
from airflow.operators.empty import EmptyOperator

@dag(start_date=datetime.datetime(2021, 1, 1), schedule="@daily")
def generate_dag():
    EmptyOperator(task_id="task")

generate_dag()
```

05. DAG

# Airflow Tutorial DAG

- dag\_id: UI에서 보여질 DAG ID, 필수적으로 기입, 중복되면 문제 발생
  - start\_date: 실제 시작 전으로 작성
  - schedule\_interval: cron 시간으로 설정 가능

```
#          minute (0 - 59)
#          |       hour (0 - 23)
#          |       |   day of the month (1 - 31)
#          |       |   month (1 - 12)
#          |       |   |   day of the week (0 - 6) (Sunday to Saturday;
#          |       |   |   7 is also Sunday on some systems)
#          |       |   |
#          |       |
#          |       |
#          |       |
# * * * * * <command to execute>
```

## 05. DAG

### Airflow Tutorial DAG

- cron
- \* : 모든 값을 의미
- ? : 특정한 값이 없음을 의미
- - : 범위 지정
- , : 특정한 값일 때만 동작
- / : 시작시간 / 단위 (ex) 0분부터 매 30분 0/30
- L : 일에서 사용하면 마지막 일, 요일에서는 마지막 요일(Saturday)
- W : 가장 가까운 평일 (ex) 12W는 12일에서 가장 가까운 평일 (Mon - Fri)을 찾음
- # : 몇 째 주의 무슨曜일을 표현 (ex) 3#2 : 2번째주 수요일

<https://crontab.guru/>에서 cron표현식의 의미를 확인하고 사용

# crontab guru

The quick and simple editor for cron schedule expressions by [Cronitor](#)

“At 04:05.”

[next](#) at 2023-03-16 04:05:00

[random](#)

5 4 \* \* \*

minute	hour	day	month	day
		(month)		(week)

\* any value

,

value list  
separator

- range of values

/ step values

**@yearly** (non-standard)

**@annually** (non-standard)

**@monthly** (non-standard)

**@weekly** (non-standard)

**@daily** (non-standard)

**@hourly** (non-standard)

**@reboot** (non-standard)

## 05. DAG

### Airflow DAG

- dependency: Task간 의존성을 >> 으로 설정
- default\_args: DAG안에 들어갈 요소들을 Key Value형식으로 지정해 넣을 수 있음  
동일한 기본 argument set에 대해 쉽게 설정 가능
- description: DAG에 대한 설명
- tags: DAG에 tag를 설정해 UI에서 쉽게 구분
- catchup: 정해진 시간에 실행되지 못한 DAG를 나중에 실행하는 것  
보통 False로 설정  
ex)start\_date부터 현재 시간까지 실행 못한 DAG를 모두 실행

# Chapter 1. Airflow란?

## 06. Operator 살펴보기

## 06. Operator 살펴보기

### Airflow Operators

- Operator는 개념적으로 미리 정의된 Task에 대한 템플릿  
DAG 내부에서 선언적으로 정의가 가능
- Airflow의 장점 중 하나는 다양한 Operator들이 있다는 것
- 원하는 대부분의 Operator들이 준비되어 있음
- Python, Bash, Spark, EMR, Bigquery 등등
- Operator들을 가져와서 목적에 맞게 사용하면 됨
- Airflow의 확장성에 기여

## 06. Operator 살펴보기

### 기본적인 Airflow Operators

- PythonOperator
- BashOperator

### AWS, GCP에서 활용할만한 Operators

- BigqueryOperators
- Amazon EMROperators
- KubernetesPodOperator

## 06. Operator 살펴보기

### PythonOperator

```
@task(task_id="print_the_context")      <- Task decorator (@task()) python 태스크를 만듦
def print_context(ds=None, **kwargs):
    """Print the Airflow context and ds variable from the context."""
    pprint(kwargs)
    print(ds)
    return "Whatever you return gets printed in the logs"

run_this = print_context()
```

## 06. Operator 살펴보기

### PythonOperator

```
# Generate 5 sleeping tasks, sleeping from 0.0 to 0.4 seconds respectively
for i in range(5):

    @task(task_id=f"sleep_for_{i}") <-parameter를 전달하고 싶다면
    def my_sleeping_function(random_base):
        """This is a function that will run within the DAG execution"""
        time.sleep(random_base)

    sleeping_task = my_sleeping_function(random_base=float(i) / 10)

    run_this >> log_the_sql >> sleeping_task
```

## 06. Operator 살펴보기

### BashOperator

```
run_this = BashOperator(  
    task_id="run_after_loop",  
    bash_command="echo 1",  
)  
  
t2 = BashOperator(  
    task_id="bash_example",  
    # "scripts" folder is under "/usr/local/airflow/dags"  
    bash_command="scripts/test.sh",  
    dag=dag,  
)
```

## 06. Operator 살펴보기

### BigqueryOperators

```
create_dataset = BigQueryCreateEmptyDatasetOperator(task_id="create_dataset", dataset_id=DATASET_NAME)

get_dataset = BigQueryGetDatasetOperator(task_id="get-dataset", dataset_id=DATASET_NAME)

get_dataset_tables = BigQueryGetDatasetTablesOperator(
    task_id="get_dataset_tables", dataset_id=DATASET_NAME
)
```

## 06. Operator 살펴보기

### BigqueryOperators

```
update_table = BigQueryUpdateTableOperator(  
    task_id="update_table",  
    dataset_id=DATASET_NAME,  
    table_id="test_table",  
    fields=["friendlyName", "description"],  
    table_resource={  
        "friendlyName": "Updated Table",  
        "description": "Updated Table",  
    },  
)
```

## 06. Operator 살펴보기

### BigqueryOperators

```
get_data = BigQueryGetDataOperator(  
    task_id="get_data",  
    dataset_id=DATASET,  
    table_id=TABLE_1,  
    max_results=10,  
    selected_fields="value,name",  
    location=location,  
)
```

## 06. Operator 살펴보기

### BigqueryOperators

```
INSERT_ROWS_QUERY = (
    f"INSERT {DATASET}.{TABLE_1} VALUES "
    f"(42, 'monty python', '{INSERT_DATE}'), "
    f"(42, 'fishy fish', '{INSERT_DATE}');"
)
```

```
insert_query_job = BigQueryInsertJobOperator(
    task_id="insert_query_job",
    configuration={
        "query": {
            "query": INSERT_ROWS_QUERY,
            "useLegacySql": False,
        }
    },
    location=location,
)
```

## 06. Operator 살펴보기

### AWS EMR Operators

```
SPARK_STEPS = [
    {
        'Name': 'calculate_pi',
        'ActionOnFailure': 'CONTINUE',
        'HadoopJarStep': {
            'Jar': 'command-runner.jar',
            'Args': ['/usr/lib/spark/bin/run-example', 'SparkPi', '10'],
        },
    },
]
```

```
JOB_FLOW_OVERRIDES = {
    'Name': 'PiCalc',
    'ReleaseLabel': 'emr-5.29.0',
    'Instances': {
        'InstanceGroups': [
            {
                'Name': 'Master node',
                'Market': 'SPOT',
                'InstanceRole': 'MASTER',
                'InstanceType': 'm1.medium',
                'InstanceCount': 1,
            }
        ],
        'KeepJobFlowAliveWhenNoSteps': False,
        'TerminationProtected': False,
    },
    'Steps': SPARK_STEPS,
    'JobFlowRole': 'EMR_EC2_DefaultRole',
    'ServiceRole': 'EMR_DefaultRole',
}
```

## 06. Operator 살펴보기

### AWS EMR Operators

```
cluster_creator = EmrCreateJobFlowOperator(  
    task_id='create_job_flow',  
    job_flow_overrides=JOB_FLOW_OVERRIDES,  
    aws_conn_id='aws_default',  
    emr_conn_id='emr_default',  
)  
  
step_adder = EmrAddStepsOperator(  
    task_id='add_steps',  
    job_flow_id=cluster_creator.output,  
    aws_conn_id='aws_default',  
    steps=SPARK_STEPS,  
)
```

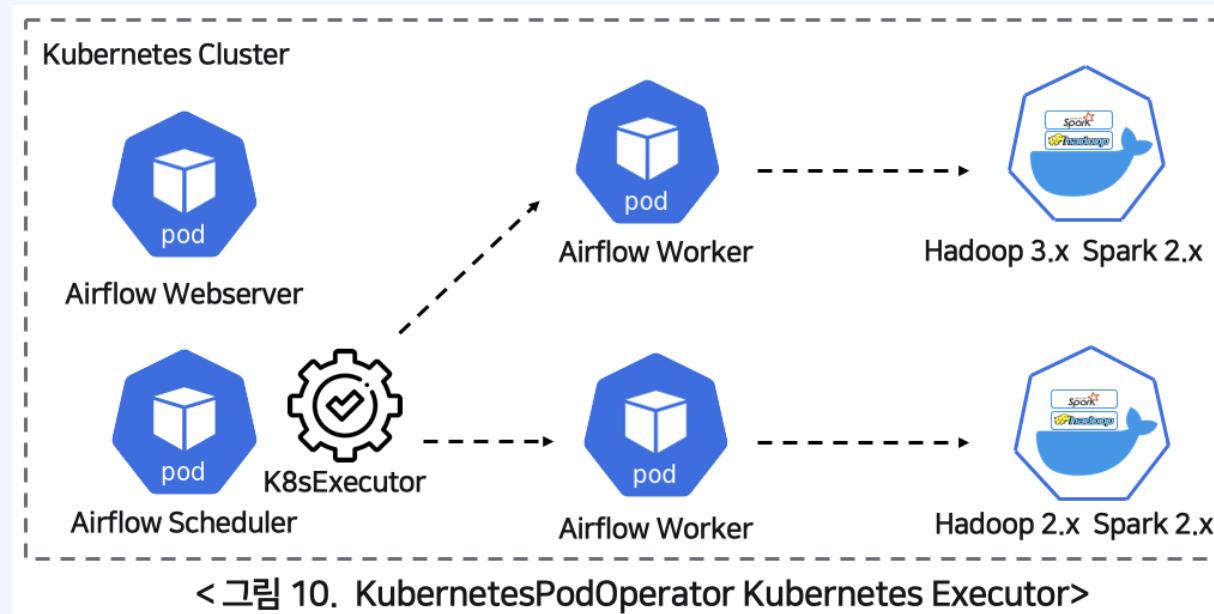
## 06. Operator 살펴보기

### AWS EMR Operators

```
step_checker = EmrStepSensor(  
    task_id='watch_step',  
    job_flow_id=cluster_creator.output,  
    step_id="{{ task_instance.xcom_pull(task_ids='add_steps', key='return_value')[0] }}",  
    aws_conn_id='aws_default',  
)  
  
cluster_remover = EmrTerminateJobFlowOperator(  
    task_id='remove_cluster',  
    job_flow_id=cluster_creator.output,  
    aws_conn_id='aws_default',  
)  
  
step_adder >> step_checker >> cluster_remover
```

## 06. Operator 살펴보기

### KubernetesPodOperator



<https://engineering.linecorp.com/ko/blog/data-engineering-with-airflow-k8s-2/>

## 06. Operator 살펴보기

### KubernetesPodOperator

```
quay_k8s = KubernetesPodOperator(  
    namespace="default",  
    image="quay.io/apache/bash",  
    image_pull_secrets=[k8s.V1LocalObjectReference("testquay")],  
    cmd=["bash", "-cx"],  
    arguments=["echo", "10", "echo pwd"],  
    labels={"foo": "bar"},  
    name="airflow-private-image-pod",  
    is_delete_operator_pod=True,  
    in_cluster=True,  
    task_id="task-two",  
    get_logs=True,  
)
```

# Chapter 1. Airflow란?

## 07. Cloud Managed Service of Airflow

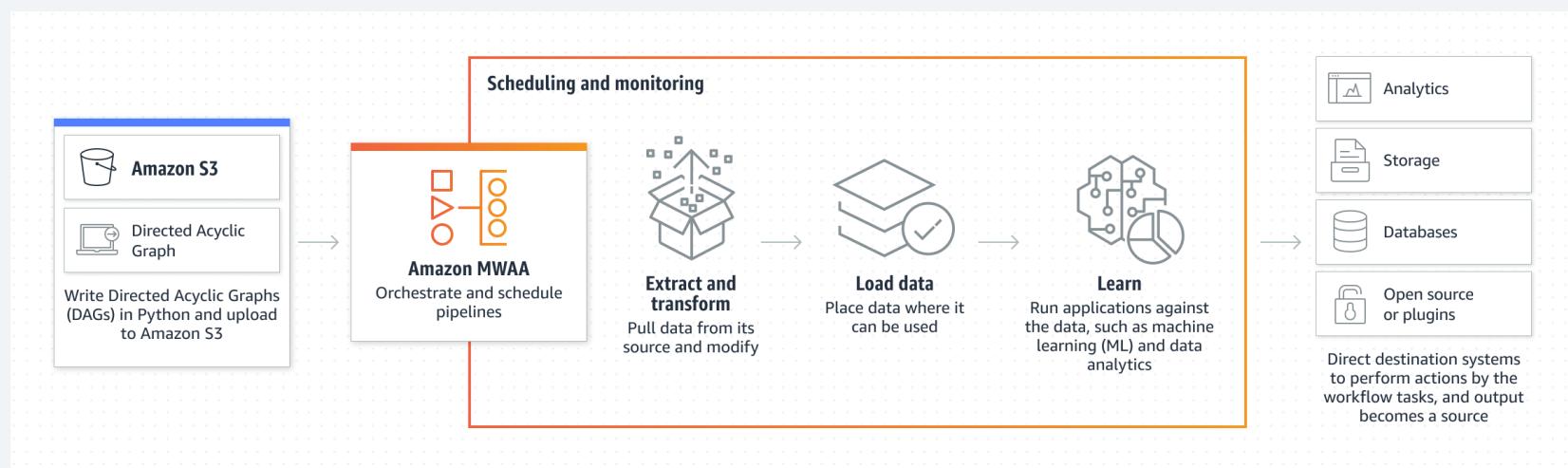
## 07. Cloud Managed Service of Airflow

### Managed Service

- AWS, GCP 같은 클라우드 사에서 제공하는 완전 관리형 서비스
- 복잡한 설정을 클라우드 사에 맡겨버림
- 코드에만 집중할 수 있음, 운영 업무 감소
- 장애가 나더라도 클라우드 사의 문제
- 대표적으로 AWS의 Amazon Managed Workflows for Apache Airflow(MWAA), GCP의 Composer가 있음

## 07. Cloud Managed Service of Airflow

### AWS MWAA



## 07. Cloud Managed Service of Airflow

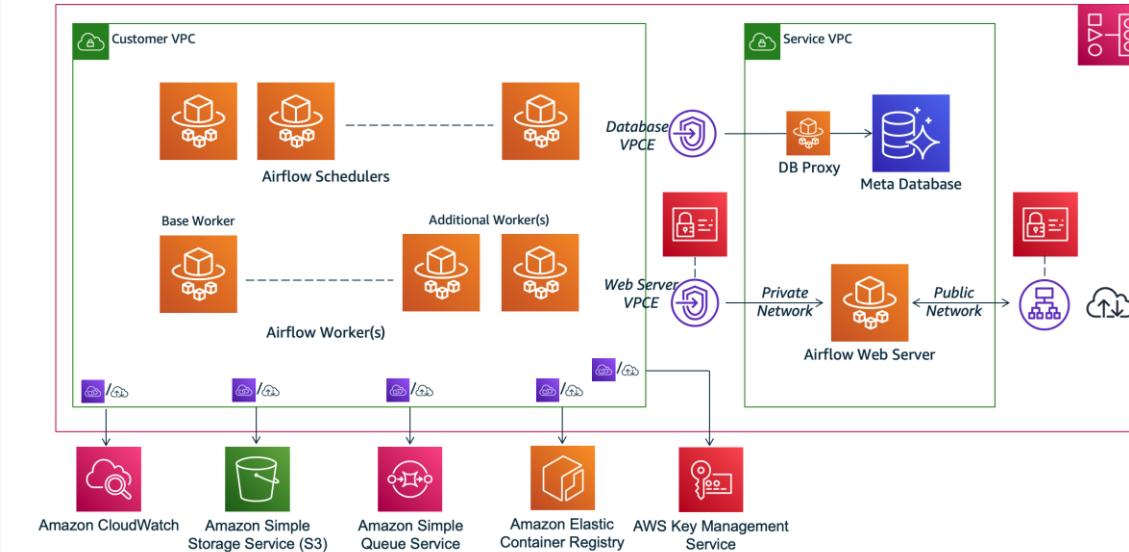
### AWS MWAA

- 간단하게 Airflow를 사용할 수 있음
- Worker가 자동 확장, 오토 스케일링
- AWS 기본 보안
- 워크플로우 모니터링
- 운영 비용 감소
- AWS의 다양한 플러그인 사용 가능  
(Athena, Batch, Cloudwatch, DynamoDB, DataSync, EMR, ECS/Fargate, EKS, Firehose, Glue, Lambda, Redshift, SQS, SNS, Sagemaker, S3)

## 07. Cloud Managed Service of Airflow

### AWS MWAA

#### Amazon MWAA Architecture



<https://docs.aws.amazon.com/mwaa/latest/userguide/what-is-mwaa.html>

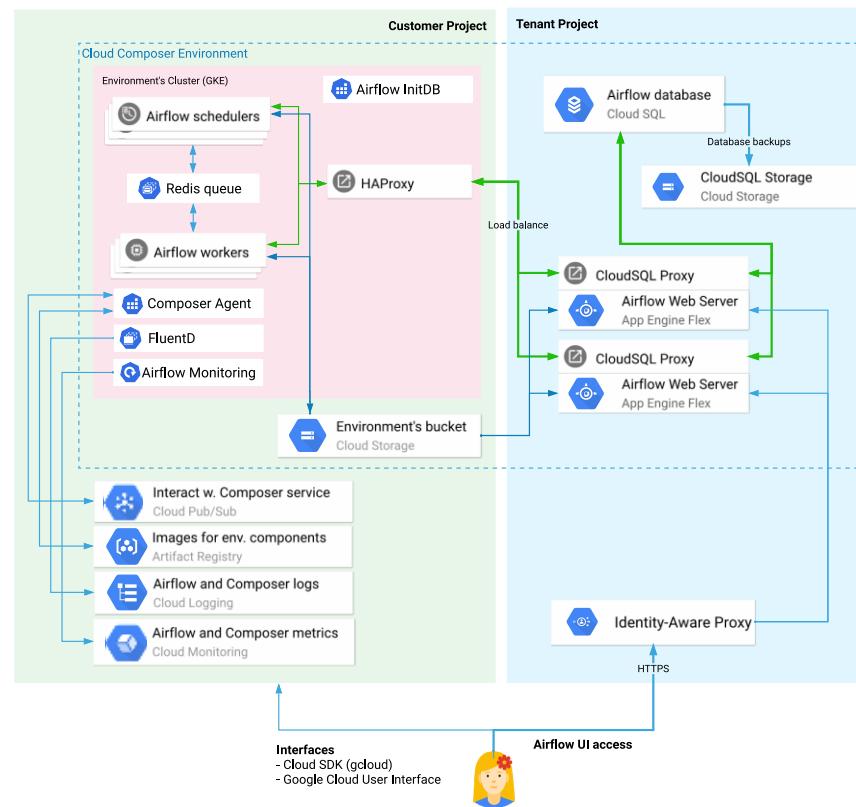
## 07. Cloud Managed Service of Airflow

### GCP Composer

- 간단하게 Airflow를 사용할 수 있음
- 리소스를 프로비저닝하기보다 워크플로 작성, 예약, 모니터링에 집중
- GCP 제품들과 통합가능  
(BigQuery, Dataflow, Dataproc, Datastore, Cloud Storage,)
- 멀티 클라우드 제공
- GCP 네트워킹, 보안 제공
- Kubernetes 기반의 Airflow 구성

## 07. Cloud Managed Service of Airflow

### GCP Composer



## 07. Cloud Managed Service of Airflow

### Managed Service?

- 장애가 나면 클라우드 사의 문제 해결까지 기다려야 함

중요 파이프라인에 장애가 난다면

- 생각보다 비싼 비용

<https://aws.amazon.com/ko/managed-workflows-for-apache-airflow/pricing/>

<https://cloud.google.com/composer/pricing?hl=ko>

- 내부에 쿠버네티스를 사용할 수 있는 인력이 있다면?

- 커스텀 할 수 없음, 제약이 발생

configuration 조정이 원하는대로..?

- 당장 급하면 바로 콘솔에서 생성할 수 있다.

난이도는 MWAA < Composer (개인적 견해)

# Chapter 1. Airflow란?

## 08. Luigi, Prefect, Oozie 비교

## 08. Argo, Luigi, Prefect, Oozie 비교

### 여러 오케스트레이션 툴

- 대표적으로 Luigi, Prefect, Oozie
- 이 툴들의 특징들을 살펴보기
- Airflow와 비교해보기
- Airflow를 선택한 이유?

## 08. Luigi, Prefect, Oozie 비교

### Luigi

- Spotify에서 개발
- Python기반으로 배치 작업 실행 가능
- 작업과 작업 간의 종속성 지정이 가능
- Airflow와 매우 유사함, Central Scheduler
- UI가 최소한으로 제공
- 실행 중인 프로세스와 유저간의 상호작용이 적음
- 분산 실행환경 지원하지 않음
- 자체 트리거링 없음

## 08. Luigi, Prefect, Oozie 비교

### Luigi

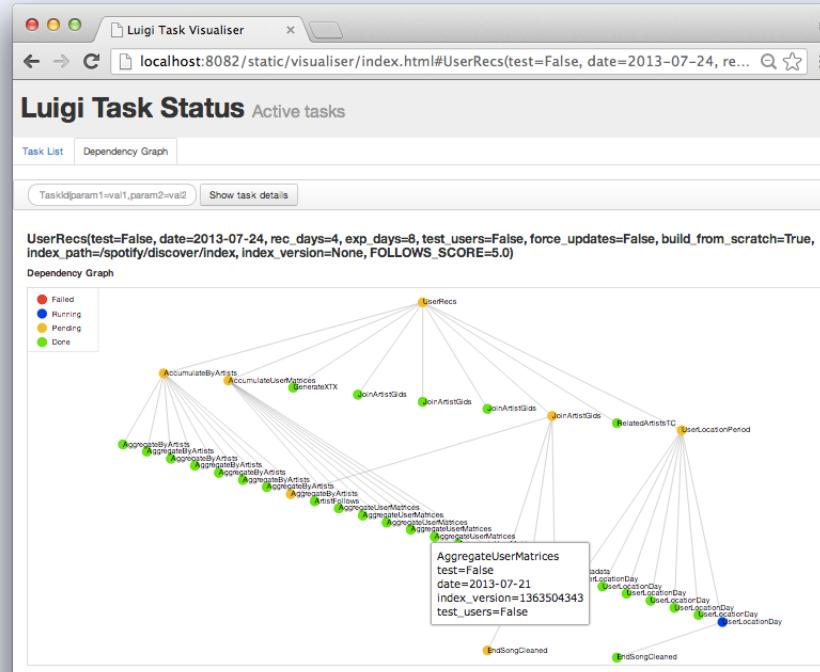
The screenshot shows the Luigi Task Status interface. At the top, there are summary metrics for Pending Tasks (3), Running Tasks (1), Failed Tasks (3), Upstream Failure (1), Batch Running Tasks (0), Disabled Tasks (0), Done Tasks (134), and Upstream Disabled (0). Below this, a table displays a single running task: LoadTable, which is part of the data\_management task family. The table includes columns for Name, Details, Priority, Time, and Actions.

Name	Details	Priority	Time	Actions
LoadTable	table=FLIP, date=2019-05-28	0	5/28/2019, 8:00:17 AM	[View]

On the left side, a sidebar lists various task families: data\_management, PartsReport, PFEPE, BillOfMaterials, and Others. Under 'Others', there is a single entry: LoadTable.

## 08. Luigi, Prefect, Oozie 비교

### Luigi



## 08. Luigi, Prefect, Oozie 비교

### Luigi

[spotify / luigi](#) Public

Watch 482 Fork 2.4k Star 16.4k

Code Issues 84 Pull requests 27 Actions Projects Security Insights

master 1 branch 64 tags Go to file Add file Code

honnix Release 3.2.1 (#3231)	✓ 18bf067 3 days ago	⌚ 4,074 commits
.github support Python 3.10 (#3140)	3 months ago	
bin Issue #838: setup.py: improve packaging of 'bin' scripts with 'entry_...'	9 years ago	
doc Update print statements in code examples to python 3 (#3211)	3 months ago	
examples Improve control over dynamic requirements (#3179)	6 months ago	
luigi Release 3.2.1 (#3231)	3 days ago	
scripts/ci Login to docker hub (#3030)	3 years ago	
test Fix typo in range_test.py (#3228)	3 weeks ago	
.coveragerc add codecov section	2 years ago	
.gitignore Add data files to .gitignore (#2367)	5 years ago	

About

Luigi is a Python module that helps you build complex pipelines of batch jobs. It handles dependency resolution, workflow management, visualization etc. It also comes with Hadoop support built in.

python hadoop scheduling  
orchestration-framework luigi

Readme Apache-2.0 license  
Code of conduct  
Security policy  
16.4k stars  
482 watching

## 08. Luigi, Prefect, Oozie 비교

### Luigi

apache / airflow Public

Code Issues 704 Pull requests 136 Discussions Actions Projects 12 Security Insights

main 52 branches 2,815 tags Go to file Add file Code

ephraimbuddy and eejbyfeldt Revert fix for on\_failure\_callback when t... 869c1e3 2 hours ago 19,183 commits

.devcontainer Configure vscode dev-container (#26958) 5 months ago

.github rename airflow-how-to-pr Slack channel (#30159) 5 hours ago

airflow Revert fix for on\_failure\_callback when task receives a SIGTERM (#3...) 2 hours ago

chart Add Helm chart controller annotations (#30126) 18 hours ago

clients Update content type validation regexp for go client (#29476) last month

constraints Add capability of iterating locally on constraint files for CI image (#...) yesterday

dev Run separate job for latest boto/botocore version when needed (#30...) 15 hours ago

docker-context-files Add capability of iterating locally on constraint files for CI image (#...) yesterday

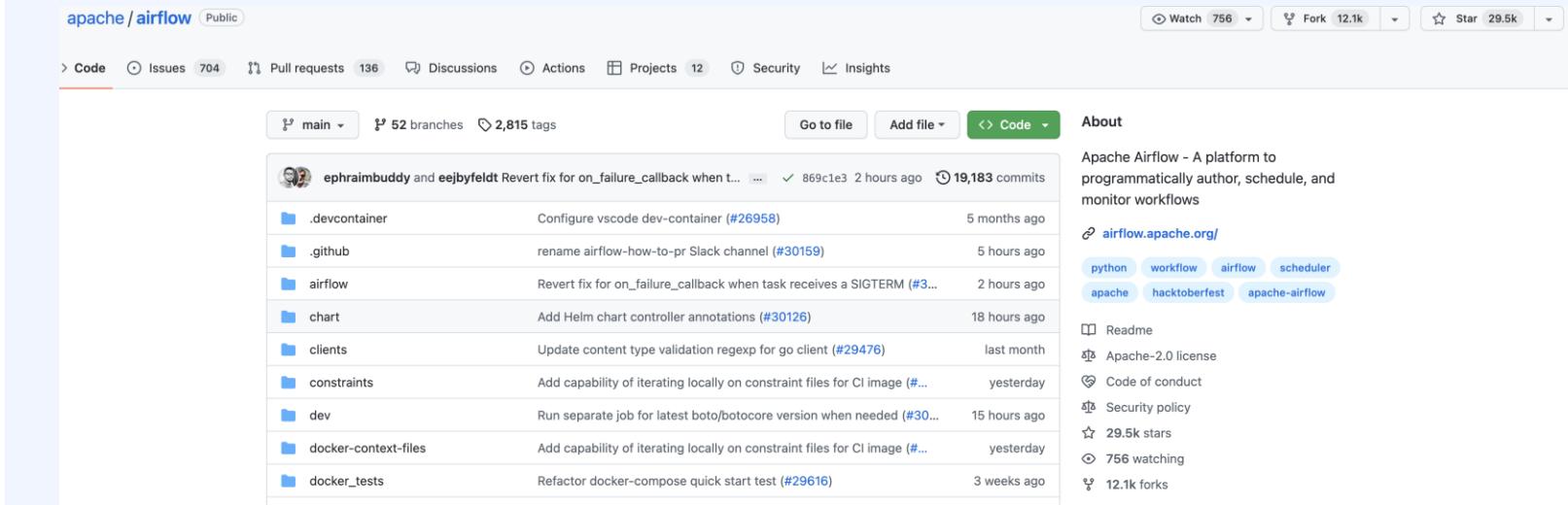
docker\_tests Refactor docker-compose quick start test (#29616) 3 weeks ago

About

Apache Airflow - A platform to programmatically author, schedule, and monitor workflows

airflow.apache.org/ python workflow airflow scheduler apache hacktoberfest apache-airflow

Readme Apache-2.0 license Code of conduct Security policy 29.5k stars 756 watching 12.1k forks



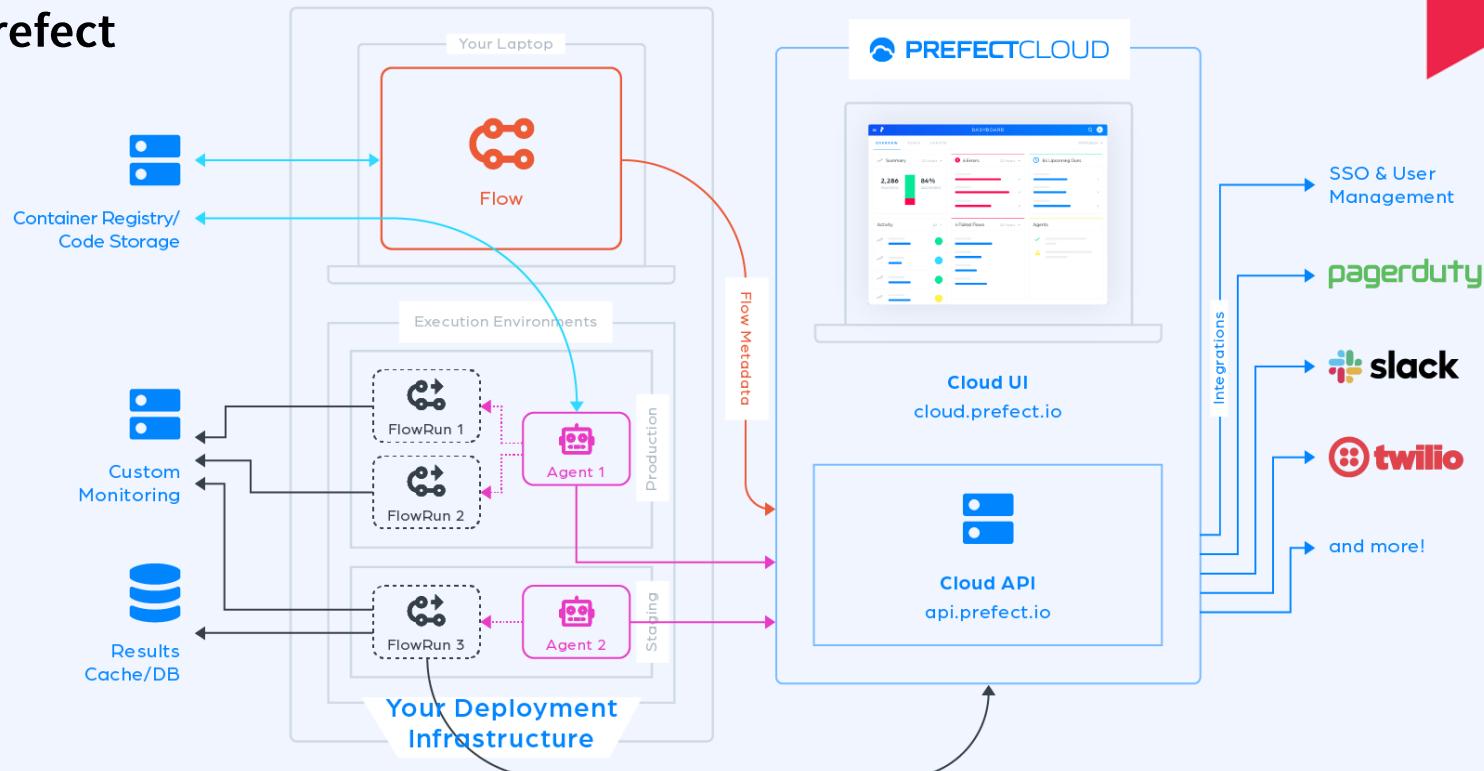
## 08. Luigi, Prefect, Oozie 비교

### Prefect

- 가장 최근에 뜨고 있는 Workflow 오픈소스 툴
- Python기반으로 배치 작업 실행 가능
- 작업과 작업 간의 종속성 지정이 가능, Scheduler 사용 가능
- Task와 Flow로 이루어짐, Flow는 Task간의 관계를 정의
- Flow에 여러 파라미터를 지정할 수 있음
- Prefect 클라우드 서비스를 사용할 수 있다  
UI서버를 자동으로 띄워줌, UI 서버는 Flow 스케줄링, 로그 관리, GUI, Flow 메타 정보 관리
- Flow를 실행하기 위해서는 Agent, Flow 코드(Code Storage)가 필요함
- 코드의 실제 실행은 Agent에서 이루어짐, 유저가 직접 띄운다  
Local, Docker, K8S ...

## 08. Luigi, Prefect, Oozie 비교

### Prefect



## 08. Luigi, Prefect, Oozie 비교

### Prefect

[PrefectHQ / prefect](#) Public

Watch 153 Fork 1.2k Star 11.5k

<> Code Issues 694 Pull requests 44 Discussions Actions Projects Security Insights

main ▾ 199 branches 175 tags Go to file Add file <> Code

Author	Commit Message	Date
madkinsz	Use new concurrency tooling in engine (#8702)	23a@0c5c 16 hours ago
.circleci	Add minimal CircleCI config	last year
.github	Use new concurrency tooling in engine (#8702)	16 hours ago
benches	Use new concurrency tooling in engine (#8702)	16 hours ago
docs	Update announce bar for 2.8.6 (#8823)	17 hours ago
flows	Add coverage for flow run with agent to integration test (#8374)	last month
netlify/edge-functions	Add Netlify Edge function to proxy to Segment (#8657)	2 weeks ago
scripts	Fix incorrect postgres URL in testing script (#8617)	3 weeks ago
src/prefect	Use new concurrency tooling in engine (#8702)	16 hours ago
tests	Use new concurrency tooling in engine (#8702)	16 hours ago
ui	Bump @prefecthq/prefect-ui-library from 1.1.14 to 1.1.15 in /ui (#8815)	yesterday
.dockerignore	Chore: Prefect UI Library (#8492)	last month
.gitattributes	Add versioning and version route	2 years ago
.gitignore	Add back the vscode folder to the ui (#8526)	last month

About

The easiest way to orchestrate and observe your data pipelines

prefect.io

python infrastructure workflow  
data-science data automation  
pipeline workflow-engine  
orchestration data-engineering orion  
observability prefect data-ops  
ml-ops

Readme Apache-2.0 license  
11.5k stars 153 watching  
1.2k forks

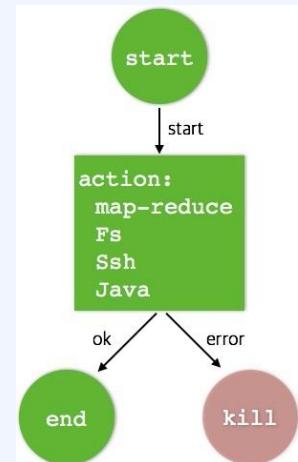
Releases 173

Release 2.8.6 Latest 17 hours ago

## 08. Luigi, Prefect, Oozie 비교

### Oozie

- 가장 오래된 Workflow 스케줄러
- Apache Hadoop의 맵리듀스(MapReduce) 작업 흐름을 관리
- 워크플로는 XML 기반의 hPDL (hadoop Process Definition Language)로 정의 Java로 이루어짐
  - DAG (Direct Acyclic Graph)를 사용함
- 흐름제어와 액션노드로 구성됨
  - 흐름제어: 워크플로의 시작과 끝, 그리고 워크플로 실행 방향
  - 액션노드: 워크플로의 계산, 처리 태스크의 실행, 다양한 액션을 제공
- 워크플로는 workflow.xml에 작성
- 하둡 기반 시스템에서 자주 볼 수 있음



<https://guide.ncloud-docs.com/docs/df-eco-oozie>

## 08. Luigi, Prefect, Oozie 비교

### Oozie

Oozie UI Workflows Coordinators Bundles Oozie Documentation hue

Filter: Search for username, name, etc... Show only 1 7 15 30 days with status Succeeded Running Killed

### Running

Submission	Status	Name	Progress	Submitter	Created	Last modified	Run	Id	Action
Thu, 16 May 2013 19:38:29	RUNNING	map-reduce-wf	50%	hue	Thu, 16 May 2013 19:38:28	Thu, 16 May 2013 19:38:29	0	0000044-130516114910501-oozie-oozi-W	Kill

Showing 1 to 1 of 1 entries

← Previous 1 Next →

### Completed

Completion	Status	Name	Duration	Submitter	Created	Last modified	Run	Id
Thu, 16 May 2013 19:31:10	SUCCEEDED	map-reduce-wf	41s	hue	Thu, 16 May 2013 19:30:29	Thu, 16 May 2013 19:31:10	0	0000043-130516114910501-oozie-oozi-W
Thu, 16 May 2013 19:30:08	SUCCEEDED	map-reduce-wf	40s	hue	Thu, 16 May 2013 19:29:28	Thu, 16 May 2013 19:30:08	0	0000042-130516114910501-oozie-oozi-W
Thu, 16 May 2013 19:05:28	SUCCEEDED	aggregator-wf	43s	hue	Thu, 16 May 2013 19:04:45	Thu, 16 May 2013 19:05:28	0	0000041-130516114910501-oozie-oozi-W
Thu, 16 May 2013 19:01:56	SUCCEEDED	aggregator-wf	53s	hue	Thu, 16 May 2013 19:01:03	Thu, 16 May 2013 19:01:56	0	0000040-130516114910501-oozie-oozi-W
Thu, 16 May 2013 19:00:41	KILLED	aggregator-wf	30s	hue	Thu, 16 May 2013 19:00:11	Thu, 16 May 2013 19:00:42	0	0000037-130516114910501-oozie-oozi-W
Thu, 16 May 2013 19:00:29	SUCCEEDED	map-reduce-wf	46s	hue	Thu, 16 May 2013 19:00:27	Thu, 16 May 2013 19:00:29	1	0000035-130516114910501-oozie-oozi-W

## 08. Luigi, Prefect, Oozie 비교

### Oozie

apache / oozie Public

Code Pull requests 46 Actions Projects Security Insights

master 27 branches 51 tags Go to file Add file Code

dionusos OOZIE-3695 [sharelib-hive2] Fix current SpotBugs discovered issues ... b5d2b90 on Feb 3 2,405 commits

bin	OOZIE-3468 [build] Use modernizer plugin (asalamon74 via kmarton)	4 years ago
client	OOZIE-3674 Add a --insecure like parameter to Oozie client so it can...	4 months ago
core	OOZIE-3689 Remove usage of commons-HttpClient due to EOL (jma...	3 months ago
distro	Changed version to 5.3.0-SNAPSHOT	4 years ago
docs	OOZIE-3606 Extend file system EL functions to use custom file syste...	3 months ago
examples	OOZIE-3693 [examples] Fix current SpotBugs discovered issues in ...	2 months ago
fluent-job	OOZIE-3628 Fix Oozie build errors caused by Maven 3.8.1	2 years ago
minitest	Changed version to 5.3.0-SNAPSHOT	4 years ago
server	OOZIE-3690 [server] Fix current SpotBugs discovered issues in Oozi...	2 months ago
sharelib	OOZIE-3695 [sharelib-hive2] Fix current SpotBugs discovered issue...	last month

About

Mirror of Apache Oozie

javascript java big-data oozie

Readme Apache-2.0 license Code of conduct Security policy 671 stars 77 watching 472 forks

Releases 51 tags

