

1. Apache Airflow란

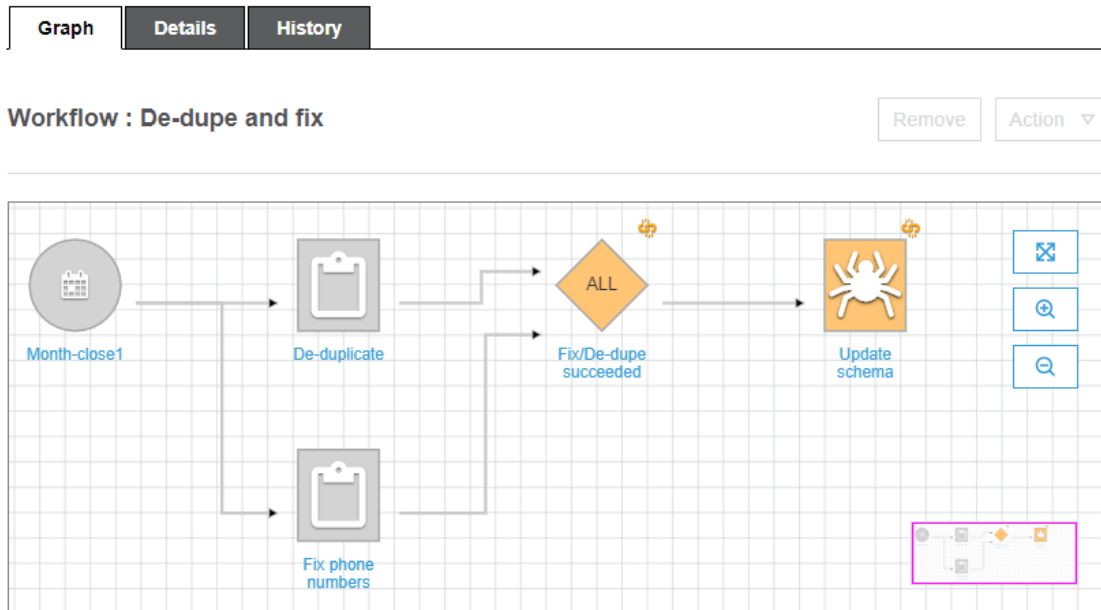
1. [정의](#)
2. [아키텍처](#)
3. [Airflow Single Node 구성](#)

1. 정의

Apache Airflow : Airbnb에서 개발한 워크플로우 스케줄링, 모니터링 플랫폼

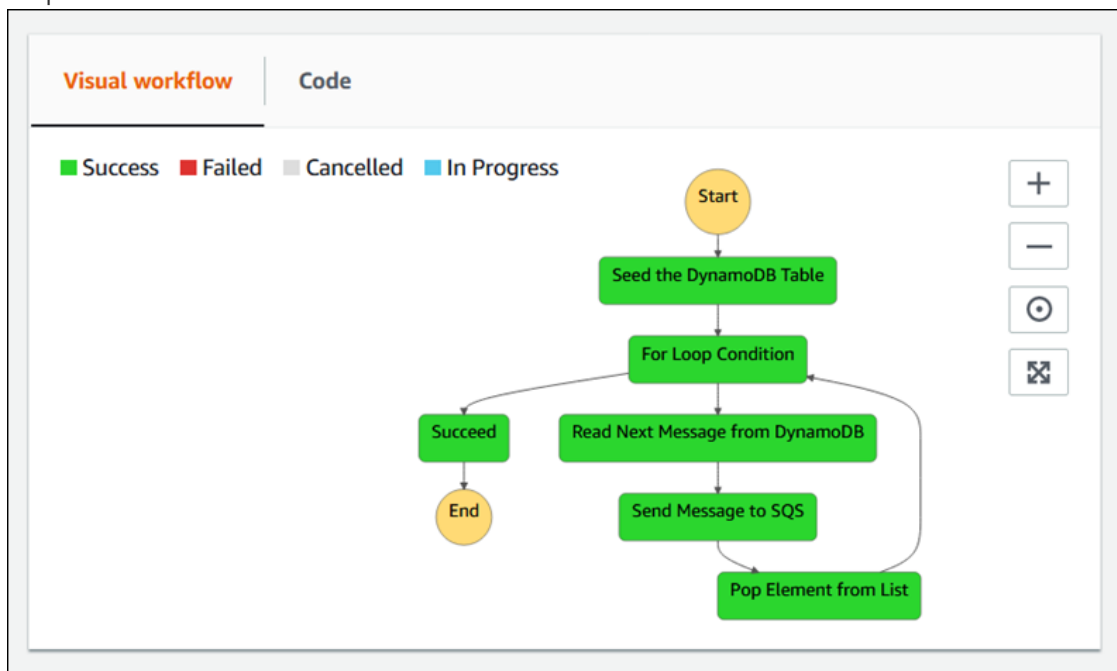
워크플로우 플랫폼 (종류):

- Glue WorkFlow



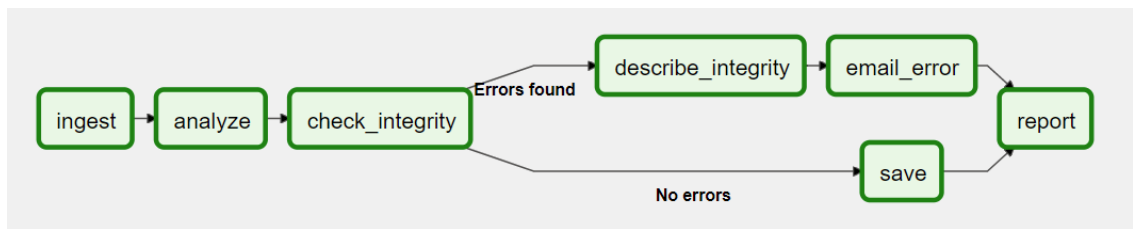
(Image URL : https://docs.aws.amazon.com/ko_kr/glue/latest/dg/workflows_overview.html)

- Step Function



(Image URL : <https://aws.amazon.com/ko/blogs/korea/new-compute-database-messaging-analytics-and-machine-learning-integration-for-aws-step-functions/>)

- Airflow



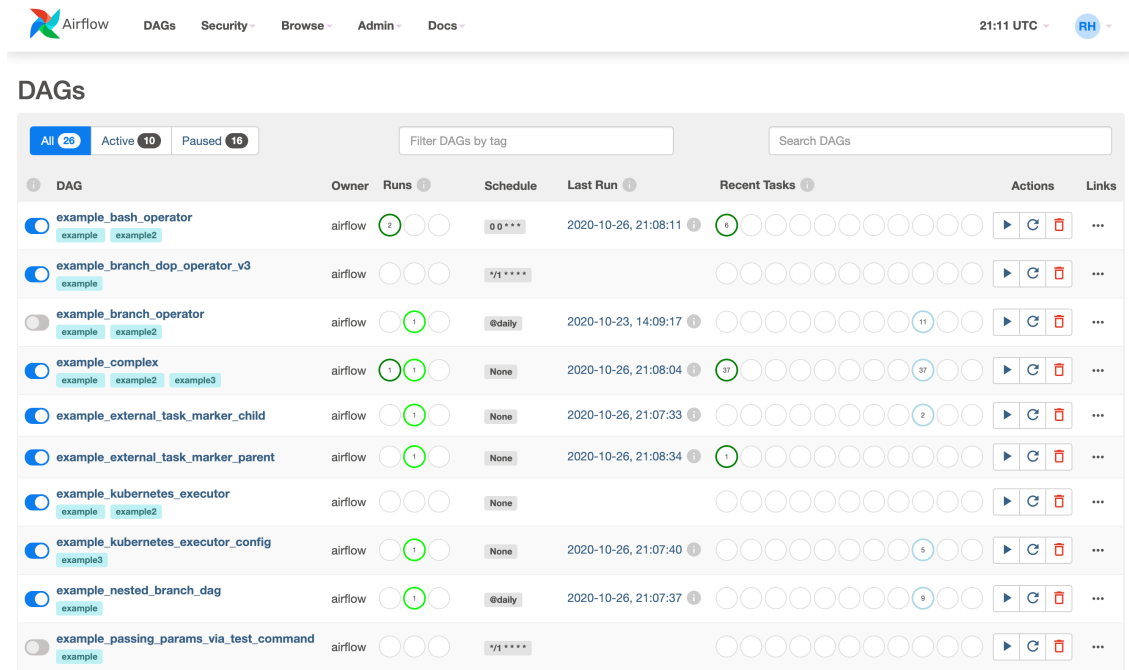
(Image URL : <https://airflow.apache.org/docs/apache-airflow/stable/concepts/overview.html>)

즉, 개별 작업들을 절차에 따라 관리 진행할 수 있는 소프트웨어

2. 아키텍처

1. 구성 요소

- Scheduler : 예약된 워크플로 트리거 및 Executor에 작업 제출
- Executor : Scheduler로부터 전달 받은 작업의 처리를 위해 자원 활용과 작업을 가장 잘 분배하는 중개자 역할. 분배된 작업을 적절하게 Worker에게 제출
- DAG Directory(Dag Bag) : Scheduler, Executor, Worker 가 읽는 DAG 파일들이 있는 폴더
- Metadata DB : Airflow의 상태 및 계정 정보 등의 Data를 저장, Scheduler, Executor 및 Webserver에서 사용
- Webserver : 사용자 인터페이스(UI) 제공, DAG의 검사, 트리거 및 Task의 동작 디버깅

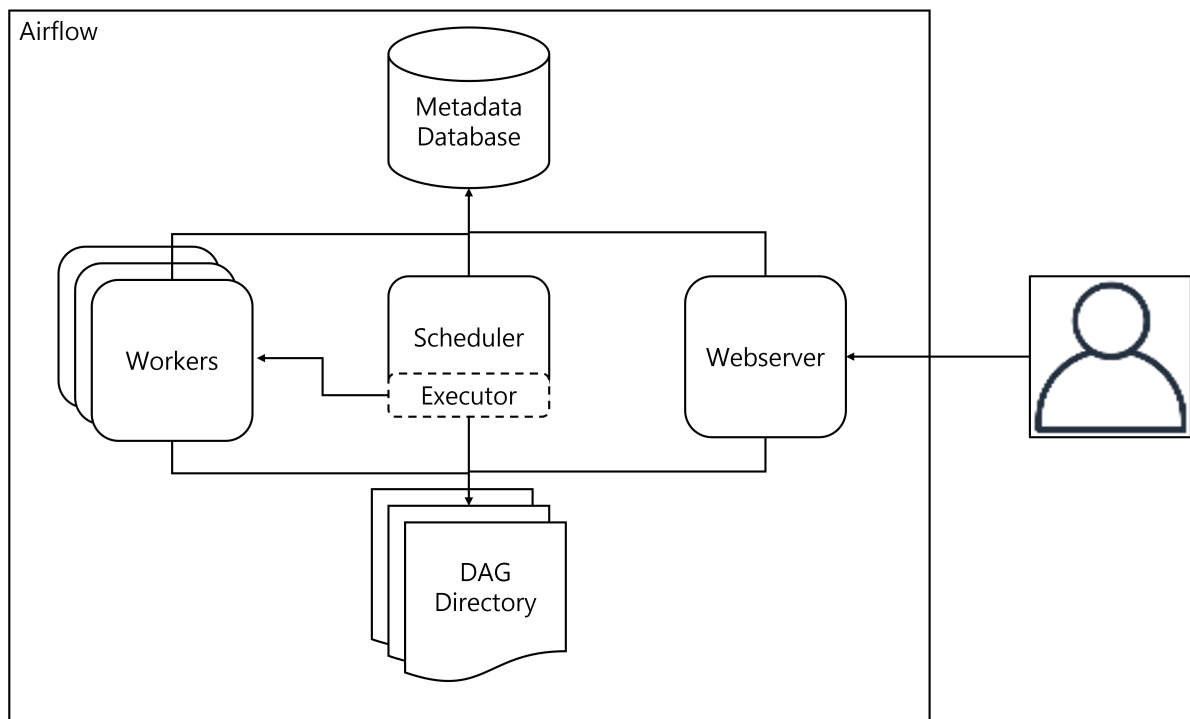


The screenshot shows the Apache Airflow web interface. At the top, there's a navigation bar with links for DAGs, Security, Browse, Admin, and Docs. The main heading is "DAGs". Below it, there are filters for "All" (26), "Active" (10), and "Paused" (16). A search bar is also present. The table lists various DAGs, including "example_bash_operator", "example_branch_dop_operator_v3", "example_branch_operator", "example_complex", "example_external_task_marker_child", "example_external_task_marker_parent", "example_kubernetes_executor", "example_kubernetes_executor_config", "example_nested_branch_dag", and "example_passing_params_via_test_command". Each row shows the DAG name, owner, runs, schedule, last run, and recent tasks.

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator	airflow	2	0 0 ***	2020-10-26, 21:08:11	4	[Refresh] [Reset] [Delete]	...
example_branch_dop_operator_v3	airflow	0	* / 1 * ***		0	[Refresh] [Reset] [Delete]	...
example_branch_operator	airflow	1	@daily	2020-10-23, 14:09:17	11	[Refresh] [Reset] [Delete]	...
example_complex	airflow	1	None	2020-10-26, 21:08:04	37	[Refresh] [Reset] [Delete]	...
example_external_task_marker_child	airflow	1	None	2020-10-26, 21:07:33	2	[Refresh] [Reset] [Delete]	...
example_external_task_marker_parent	airflow	1	None	2020-10-26, 21:08:34	1	[Refresh] [Reset] [Delete]	...
example_kubernetes_executor	airflow	0	None		0	[Refresh] [Reset] [Delete]	...
example_kubernetes_executor_config	airflow	1	None	2020-10-26, 21:07:40	5	[Refresh] [Reset] [Delete]	...
example_nested_branch_dag	airflow	1	@daily	2020-10-26, 21:07:37	9	[Refresh] [Reset] [Delete]	...
example_passing_params_via_test_command	airflow	0	* / 1 * ***		0	[Refresh] [Reset] [Delete]	...

(Image URL : <https://airflow.apache.org/docs/apache-airflow/stable/concepts/overview.html>)

Airflow Architecture



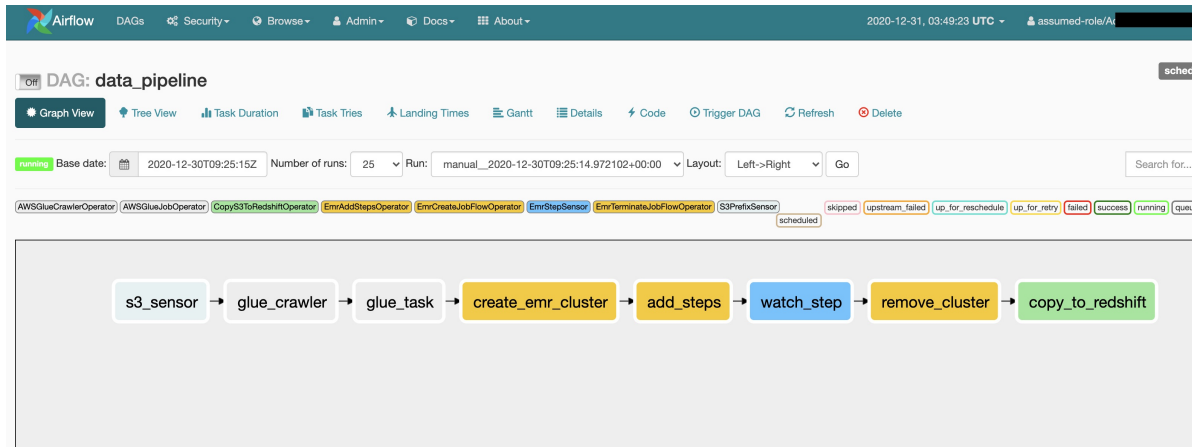
- 사용자는 Webserver에만 접근 가능
- Webserver에서 Workflow 대한 실행 시 Workflow의 실행 정보가 Metadata Database에 저장

- Scheduler는 Metadata DB로 부터 실행 내용확인 및 Workflow를 Executor에 제출
- Executor는 DAG Directory에서 DAG 정보 획득 및 Worker에 작업 할당

2. WorkLaod

DAG는 일련의 작업을 통해 실행되며, 세 가지 일반적인 작업 유형 존재

- Operator : DAG의 대부분 task들을 구성하기 위해 빠르게 조합할 수 있는 사전 정의된 작업.
- Sensors : 외부 이벤트가 발생하기를 기다리고 감지하는 Operator의 특수 하위 클래스.
- TaskFlow : `@task` 데코레이터를 사용하여 python 패키지로 된 사용자 정의 operator 기능(Airflow 2.0.0 신규 기능).



(Image URL : <https://amazon-mwaa-for-analytics.workshop.aws/images/airflow-dag.jpg>)

상단의 이미지와 같이 하나의 DAG는 Operator와 Sensor들로 생성된 Task들의 조합이다. 상단의 DAG는 하단과 같은 순서로 Workflow가 진행된다.

1. S3에서 개체를 사용할 수 있을 때까지 감지
2. Glue Cralwer 호출
3. Glue Job 호출
4. EMR Cluster 생성
5. EMR Cluster에 add step 수행
6. EMR step 수행 여부 감지
7. EMR Cluster 종료
8. S3 Data를 Redshift에 복사

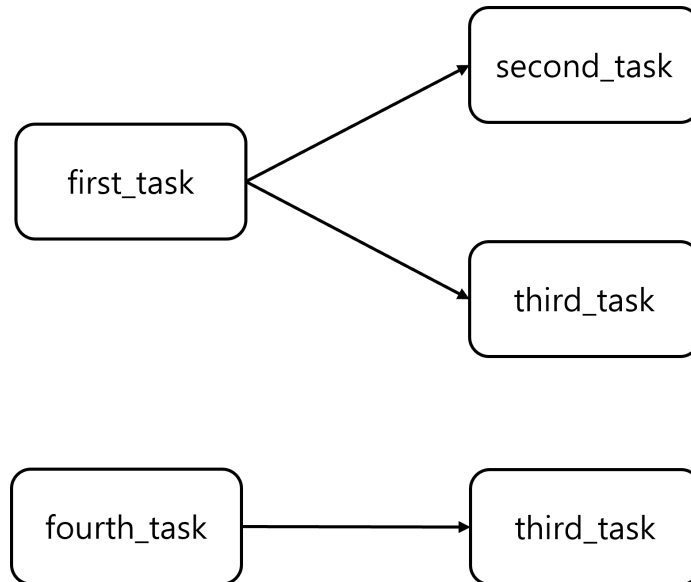
3. Control Flow

DAG는 Code로 구성되어 있기 때문에 재사용 및 동시 사용이 가능합니다.

각각의 task에는 서로에 대해 선언된 종속성이 있습니다.

Control Flow는 각각의 Task들에 대한 종속성 및 순서를 Design하여 Workflow를 구성합니다.

하단의 예시는 [first_task, second_task, third_task, fourth_task] 들로 DAG를 Design



상단의 이미지와 같이 큰 갈래로는 first_task, fourth_task의 workflow가 병렬로 진행되며 second_task, third_task 는 앞선 task가 수행된 후에 수행

- `>>` 및 `<<` 연산자를 사용

```
first_task >> [second_task, third_task]
third_task << fourth_task
```

- `set_upstream` 및 `set_downstream` 메서드를 사용

```
first_task.set_downstream([second_task, third_task])
third_task.set_upstream(fourth_task)
```

3. Airflow Single Node 구성

Airflow를 Single Node로 AWS EC2 위에 구성합니다. (EC2 생성 방법은 숙지했다고 가정)
EC2 Config는 다음과 같으며 Internet Gateway가 부착된 Public Subnet에 설정하도록 합니다.

유형	상세 내용
AMI	Amazon Linux2
Type	t2.large
vCPU	2
메모리	8
스토리지	EBS 전용

1. Airflow 기본 세팅

EC2를 생성하셨다면 기본적으로 환경변수 `AIRFLOW_HOME`을 설정합니다.

Airflow Home 경로를 설정(선택):

Quick Start

The installation is quick and straightforward.

```
# airflow needs a home, ~/airflow is the default,  
# but you can lay foundation somewhere else if you prefer  
# (optional)  
export AIRFLOW_HOME=~/airflow
```

(Image URL : <https://airflow.apache.org/docs/apache-airflow/stable/start/local.html>)

```
# EC2  
export AIRFLOW_HOME=/home/ec2-user/airflow
```


2. Redis 세팅

Airflow는 다양한 Executor를 지원합니다.

기본적으로 제공하는 SequentialExecutor, LocalExecutor 도 있지만 분산 처리에 대한 수요가 늘면서 celeryexecutor, kubernetesexecutor 를 많이 사용합니다.

이 예제에서는 celeryexecutor를 사용하도록 하겠습니다.

celeryexecutor는 분산 처리를 위해 Broker 역할을 할 수 있는 Queue service가 필요합니다.
(RabbitMQ, Redis, Amazon SQS등)

브로커로 사용할 Redis를 설치합니다.

AWS Linux의 패키지 설치 도구인 `yum`을 업데이트하고 필요한 라이브러리를 설치합니다.

```
sudo yum -y update
```

Remi 저장소 활성화합니다.

```
sudo amazon-linux-extras install epel -y
sudo yum install http://rpms.remirepo.net/enterprise/remi-release-7.rpm -y
```

Redis 패키지 yum으로 설치 및 실행합니다.

```
sudo yum install redis -y
sudo service redis start
```

Redis의 configure 파일을 수정합니다.

```
sudo vim /etc/redis.conf
#/etc/redis.conf
[.]
daemonize yes
[.]

[.]
bind 0.0.0.0
[.]
```

부팅시 자동 실행하기 위한 설정

```
sudo chkconfig redis on
sudo chkconfig redis-sentinel on
systemctl list-unit-files | grep redis
```

```
sudo chkconfig redis on
Note: Forwarding request to 'systemctl enable redis.service'.
Created symlink from /etc/systemd/system/multi-user.target.wants/redis.service to /usr/lib/systemd/system/redis.service.
sudo chkconfig redis-sentinel on
Note: Forwarding request to 'systemctl enable redis-sentinel.service'.
Created symlink from /etc/systemd/system/multi-user.target.wants/redis-sentinel.service to /usr/lib/systemd/system/redis-sentinel.service.
systemctl list-unit-files | grep redis
redis-sentinel.service enabled
redis.service enabled
```

서버 실행 및 종료 방법

```
sudo service redis start
```

강제 종료시

```
sudo service redis stop
```

```
sudo service redis start  
Redirecting to /bin/systemctl start redis.service
```

`sudo service redis-server start` 명령이 정상 작동했는지 확인하기 위해 `ps` 명령어를 수행합니다.

```
ps -ef | grep redis
```

```
root      2329      1  0 12월11 ?        00:01:50 /usr/local/bin/redis-server 0.0.0.0:6379
```

브로커 역할을 할 redis에 대한 설정이 완료되었습니다.

3. MySQL 세팅

Airflow의 Metadata Database로는 MySQL을 사용합니다.

Webserver, Scheduler, Worker 등 많은 서비스에서 이용되는 만큼 까다롭고 신중히 설정해야하는 내용입니다.

MySQL 설치

```
sudo yum install mysql -y
```

```
$ sudo yum install mysql -y
Loaded plugins: extras, suggestions, langpacks, priorities, update-motd
amzn2-core | 3.7 KB 00:00:00
222 packages excluded due to repository priority protections
Resolving Dependencies
--> Running transaction check
--> Package mariadb.x86_64 1:5.5.68-1.amzn2 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository Size
=====
Installing:
mariadb x86_64 1:5.5.68-1.amzn2 amzn2-core 8.8 M
Transaction Summary
-----
Install 1 Package
Total download size: 8.8 M
Installed size: 49 M
Downloading packages:
mariadb-5.5.68-1.amzn2.x86_64.rpm | 8.8 MB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : 1:mariadb-5.5.68-1.amzn2.x86_64
Verifying : 1:mariadb-5.5.68-1.amzn2.x86_64
Installed:
mariadb.x86_64 1:5.5.68-1.amzn2
Complete!
```

yum 명령어로 설치시 mariadb가 설치되는데 이는 권장 DB가 아닙니다.

하단의 공식 문서에 따르면 MySQL 5.7 또는 8을 권장하고 있으므로 다른 방법으로 설치해야 합니다.

<https://airflow.apache.org/docs/apache-airflow/stable/installation.html#prerequisites>

MySQL 5.7 설치

```
sudo yum install https://dev.mysql.com/get/mysql57-community-release-e17-11.noarch.rpm -y
sudo yum install mysql-community-server -y
sudo yum install mysql mysql-server mysql-libs mysql-devel -y
```

MySQL 데몬을 실행

```
sudo service mysqld start
```

정상적으로 실행이 된다면 MySQL에 대한 기본적인 설치는 완료되었습니다.

Airflow에서 MySQL을 DB로 사용하기 위해서는 Airflow가 MySQL에 Connection을 만들 수 있는 계정 정보가 필요합니다.

따라서 Airflow가 사용할 Database와 해당 DB에 접근할 권한이 있는 계정을 생성해야 합니다.

Airflow Database가 없다면 Airflow 실행에 필요한 테이블들을 만들지 못합니다.

우선 root로 접속해서 계정 생성 후 데이터베이스를 생성합니다.

MySQL 세부 세팅

```
mysql -u root -p
```

최초 접근 시 `root` User에 대한 비밀번호를 모르기 때문에 접근이 불가능합니다.

```
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
```

MySQL은 설치 시 `root` User에 대한 임시 비밀번호를 생성합니다.

하단 방법에 따라 임시 비밀번호를 획득합니다.

```
sudo cat /var/log/mysqld.log | grep root
```

```
[ec2-user@ip-10-0-1-251 ~]$ sudo cat /var/log/mysqld.log | grep root
2021-04-04T12:05:27.908396Z 1 [Note] A temporary password is generated for root@localhost: Bt&p0lgem7w>
2021-04-04T12:05:56.113939Z 2 [Note] Access denied for user 'root'@'localhost' (using password: NO)
```

비밀번호를 복사해 놓은 후 다시 입력해 봅니다.

```
mysql -u root -p
# Enter password:
```

mysql 프롬프트가 나왔다면 성공입니다.

`root` User는 언제든지 재사용될 수 있기 때문에 비밀번호를 재설정 합니다.

```
UPDATE mysql.user SET Password=PASSWORD('Bespın12!') WHERE User='root';
```

에러 발생

```
ERROR 1820 (HY000): You must reset your password using ALTER USER statement
before executing this statement.
```

mysql에는 비밀번호 정책이 상 중 하로 나누어져 있는데 기본 설정은 MEDIUM입니다. LOW로 변경해 줍니다.

```
set global validate_password_policy='LOW';
```

LOW로 변경 후 다시 `root` User 비밀번호를 변경합니다. 비밀번호는 8자 이상으로만 세팅하면 됩니다.

```
alter user 'root'@'localhost' identified by 'Bespın12!';
use mysql;
SET GLOBAL explicit_defaults_for_timestamp = 1;
flush privileges;
```

Airflow 계정 및 Database를 생성합니다.

```
# 계정 생성
create user 'airflow'@'localhost' identified by 'Bespın12!';

# DB 권한 부여
grant all privileges on *.* to 'airflow'@'localhost';
grant all privileges on airflow.* to 'airflow'@'localhost';

# Database 생성
create database airflow;

flush privileges;
```

MySQL 세팅은 완료되었습니다.

4. Airflow 설치 및 세팅

Airflow를 설치 및 Airflow에서 사용할 Plugin(mysql, aws, celery, redis)을 설치합니다.

```
sudo yum install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel
python3-devel.x86_64 cyrus-sasl-devel.x86_64 -y
sudo yum install gcc -y
sudo yum install libevent-devel -y
sudo pip3 install wheel
sudo pip3 install boto3 PyMySQL celery flask-bcrypt
sudo pip3 install apache-airflow-providers-sqlite
sudo pip3 install "SQLAlchemy==1.3.15"
sudo pip3 install 'apache-airflow[mysql, aws, celery, redis]'
```

설치가 완료되었다면 airflow 폴더에 있는 airflow.cfg를 수정해서 앞서 설치한 redis와 mysql의 Connection을 설정해야 합니다.

최초 airflow 생성시 AIRFLOW_HOME에 해당되는 폴더가 생성되어야 하지만 찾을 수 없습니다.

airflow 명령어가 동작하는지 먼저 확인합니다. 만약 명령어가 작동한다면 아까 설정해둔

AIRFLOW_HOME 경로에 airflow 폴더가 생성될 것입니다. 그 안에 configuration 파일이 있습니다.

```
airflow db init
```

오류가 나도 무시하셔도 됩니다. sqlite는 사용하지 않습니다.

airflow.cfg 설정

```
cd airflow
vim airflow.cfg
```

하단 내용을 찾아 적용합니다.

```
# 사용할 dag 폴더 지정
# subfolder in a code repository. This path must be absolute.
days_folder = /home/ec2-user/airflow/dags

# 사용할 executor 설정
# executor = SequentialExecutor
executor = CeleryExecutor

# MySQL Connection 설정
# sql_alchemy_conn = sqlite:///home/airflow/airflow/airflow.db
sql_alchemy_conn = mysql+pymysql://airflow:Besp1n12!@127.0.0.1:3306/airflow

# 비밀번호 사용
# auth_backend = airflow.api.auth.backend.deny_all
auth_backend = airflow.api.auth.backend.basic_auth

# Redis Connection 설정
# broker_url = sqla+mysql://airflow:airflow@127.0.0.1:3306/airflow
broker_url = redis://localhost:6379/0

# result_backend = db+mysql://airflow:airflow@localhost:3306/airflow
result_backend = db+mysql://airflow:Besp1n12!@127.0.0.1:3306/airflow
```

```
# catchup_by_default = True
catchup_by_default = False
```

다시 Airflow DB를 초기화 합니다.

```
airflow db init
```

Done! 이 나왔다면 성공입니다.

Airflow Webserver에서 사용할 계정을 생성합니다.

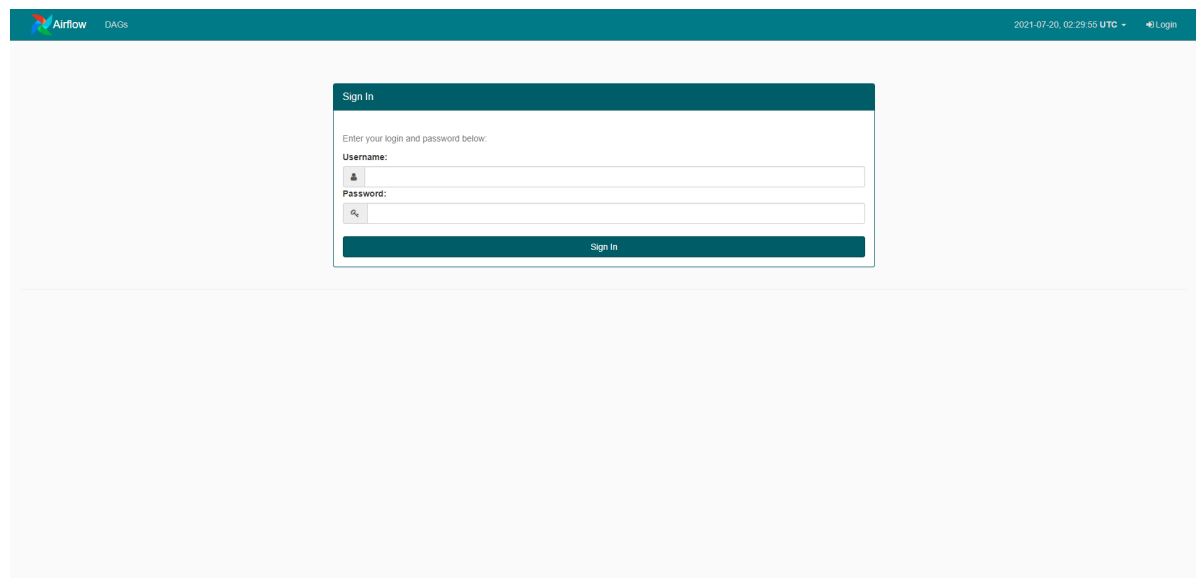
```
# user 생성
airflow users create -r Admin -u jungmin -p Bepin12! -e
candy7manner@gmail.com -f jungmin -l choi
```

Setting이 완료되었습니다.

이제 위에서 보았던 것처럼 Webserver에 접근 가능한지 확인합니다.

```
airflow webserver
```

Browser에 {DNS}:8080 을 입력합니다. 아래와 같이 Sign In 화면이 나타난다면 정상적으로 Setting이 된 것을 확인할 수 있습니다.



위에서 생성한 계정으로 접근 합니다.

The scheduler does not appear to be running.
The DAGs list may not update, and new tasks will not be scheduled.

DAGs

All 22 Active 0 Paused 22

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	example_bash_operator	example 0 0 *	Airflow				
	example_branch_dop_operator_v3	example */* *	Airflow				
	example_branch_operator	example 0 0 *	Airflow				
	example_complex	None	airflow				
	example_external_task_marker_child	None	airflow				
	example_external_task_marker_parent	None	airflow				
	example_http_operator	example 1 day, 0:00:00	Airflow				
	example_kubernetes_executor_config	None	Airflow				
	example_nested_branch_dag	example 0 0 *	airflow				
	example_passing_params_via_test_command	example */* *	airflow				
	example_pig_operator	example None	Airflow				
	example_python_operator	example None	Airflow				
	example_short_circuit_operator	example 1 day, 0:00:00	Airflow				
	example_skip_dag	example 1 day, 0:00:00	Airflow				
	example_subdag_operator	example 0 0 *	Airflow				
	example_trigger_controller_dag	example 0 0 *	airflow				

상단의 노란박스는 Airflow의 Scheduler가 동작하지 않는다는 경고문입니다. 아직 Scheduler를 실행하지 않았기 때문에 발생합니다.

하단에 보이는 DAGs list는 기본적으로 Airflow가 제공하는 Example 입니다.

기본 명령어

```
# db initialize
airflow db init

# user 생성
airflow users create -r {Role-Name} -u {User-Name} -p {Password} -e {Email} -f {First-Name} -l {Last-Name}

# dag 목록 조회
airflow list_dags
```