Overview

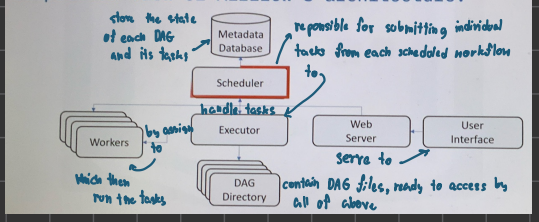
What is Apache Airflow?

- Great open source workflow orchestration tool
- A platform that lets you build and run workflows → batch data pipelines
- A workflow is represented as a DAG
- Note: Airflow is not a data streaming solution it is primary a workflow manager
  - unlike Big Data tool such as Apache Kafka/Storm/Spark

Apache Airflow

What is Apache Airflow? basic components

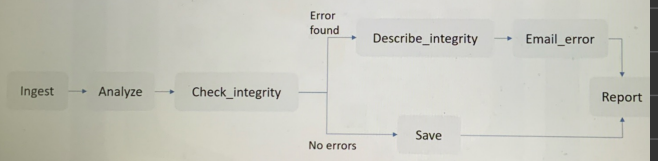
Simplified view of Airflow's architecture:



What is Apache Airflow?

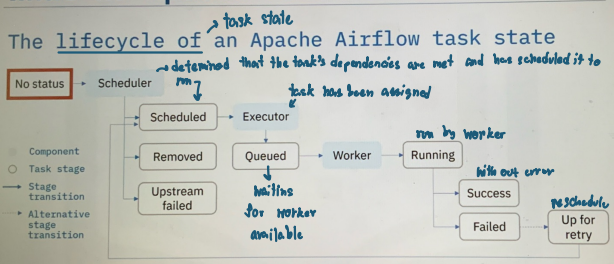
DAG specifies the dependencies between tasks, and the order in which execute them — tasks themselve decide what to do

Sample DAG illustrating the labeling of different branches:



What is Apache Airflow?

The lifecycle of an Apache Airflow task state



Apache Airflow features

- Pure Python = Flexibility
- Useful UI = Full Insight
- Integration = Plug and Play

Apache Airflow features

- Easy to Use = Unlimited Pipeline Scope
- Open Source = Community of Developers

Apache Airflow principles

Airflow pipelines are built on four principles:

- Scalable — ready code to infinity
- Dynamic — can contain multiple simultaneous tasks
- Extensible — extend library
- Lean

Apache Airflow use cases

- Defining and organizing machine learning pipeline dependencies
- Increasing the visibility of batch processes and decoupling them
- Deploying as an enterprise scheduling tool
- Orchestrating SQL transformations in data warehouses
- Sending daily analytics emails

sift
SENIORLINK
EXPERITY
Onefootball

Advantages of using Data pipelines as DAGs in Apache Airflow

Objectives

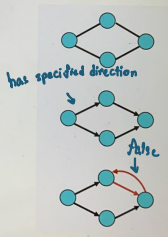
After watching this video, you will be able to:

- Define what a Directed Acyclic Graph (DAG) is
- Describe workflows as DAGs of tasks and dependencies
- Outline the components of a DAG definition file
- Describe how Apache Airflow Scheduler executes tasks on an array of workers
- List key advantages of defining workflows as code

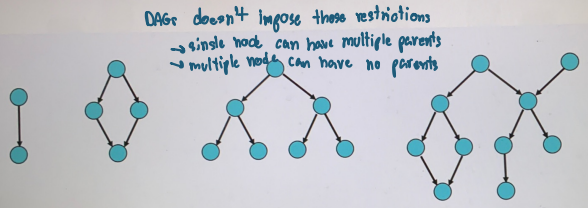
What is a DAG?

Directed Acyclic Graph

- Graph – nodes and edges
- Directed graph – each edge has a direction
- Acyclic – no loops (cycles)



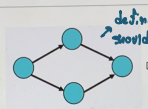
Examples of DAGs



What is a DAG in Airflow?

DAGs represent workflows:

- Nodes are tasks
- Edges are dependencies



Tasks run in order

Defined as Code

Scheduled

Tasks and operators

- Tasks are written in Python
- Tasks implement *operators*, for example, Python, SQL, or Bash operators
- Operators determine what each task does
- Sensor operators poll for a certain time or condition
- Other operators include email and HTTP request operators

DAG definition components

Python script blocks:

- Library imports
- DAG arguments
- DAG definition
- Task definitions
- Task pipeline

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
import datetime as dt

default_args = {
    'owner': 'me',
    'start_date': dt.datetime(2021, 7, 28),
    'retries': 1,
    'retry_delay': dt.timedelta(minutes=5),
}

dag = DAG('simple_example',
          description='A simple example DAG',
          default_args=default_args,
          schedule_interval=dt.timedelta(seconds=5),
)
```

DAG definition components

Python script blocks:

- Library imports
- DAG arguments
- DAG definition
- Task definitions
- Task pipeline

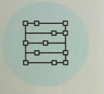
```
task1 = BashOperator(
    task_id='print_hello',
    bash_command='echo \'Greetings. The date and time are \'',
    dag=dag,
)

task2 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag,
)

task1 >> task2
```

bash

bash

depend on

Airflow Scheduler



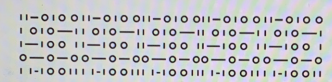
Deploys on Worker Array

Follows your DAG

First DAG Run

Subsequent Runs

Advantages of workflows as code



Defined as Code

Maintainable

Versionable

Collaborative

Testable

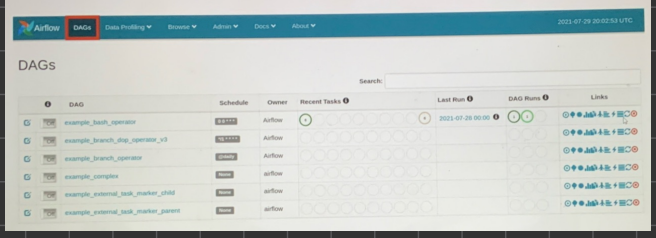
Apache Airflow UI

Objectives

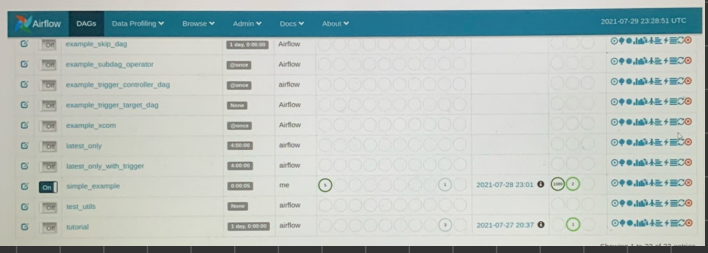
After watching this video, you will be able to:

- Identify current DAGs in your environment
- List different ways to visualize a specific DAG
- Review the code that defines your DAG
- Analyze the duration of each task in your DAG over multiple runs
- Select context metadata for any task instance

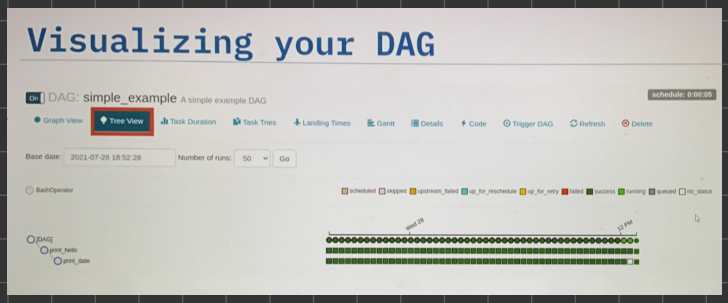
DAGs View



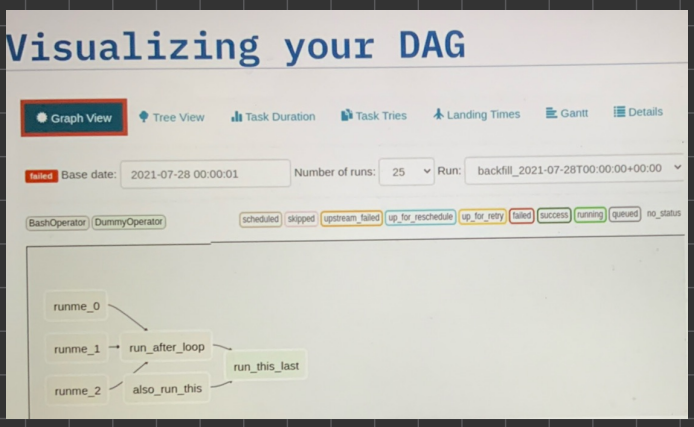
Visualizing your DAG



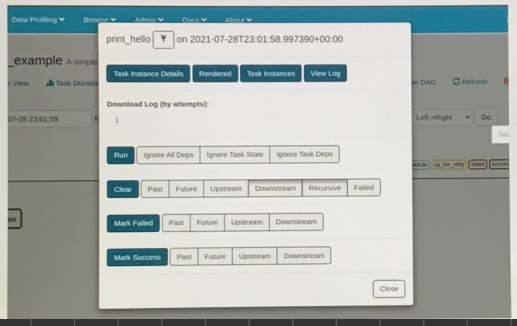
Visualizing your DAG



Visualizing your DAG



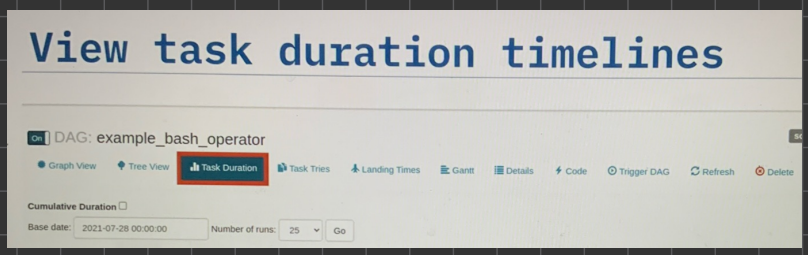
View context metadata for a task



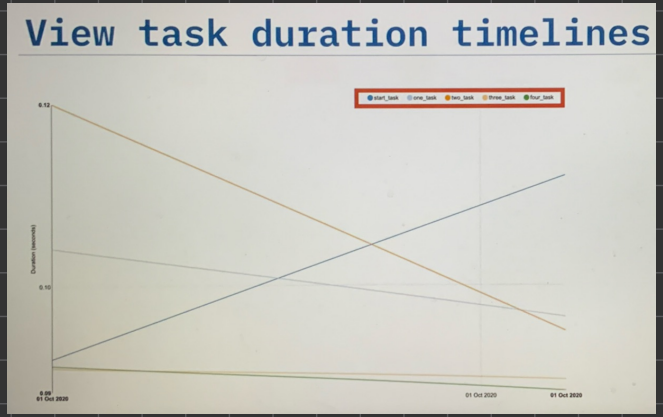
Viewing the code for your DAG

```python
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
import datetime as dt

default_args = {
    'owner': 'me',
    'start_date': dt.datetime(2021, 7, 28),
    'retries': 1,
    'retry_delay': dt.timedelta(minutes=5),
}

dag = DAG('simple_example',
          description='A simple example DAG',
          default_args=default_args,
          schedule_interval=dt.timedelta(seconds=5),
)

task1 = BashOperator(
    task_id='print_hello',
    bash_command='echo \'Greetings. The date and time are \'',
    dag=dag,
)

task2 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag,
)

task1 >> task2
```

View task duration timelines



View task duration timelines



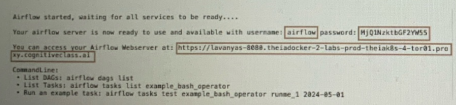
LAB

Run the command below in the terminal to start Apache Airflow.

```
start_airflow
```

Please be patient, it may take upto 5 minutes for airflow to get started.

When airflow starts successfully, you should see an output similar to the one below.



Exercise 3 - List all DAGs

Apache airflow gives us some handy command line options to work with.

Run the command below in the terminal to list out all the existing DAGs.

```
airflow dags list
```

Exercise 4 - List tasks in a DAG

Run the command below in the terminal to list out all the tasks in the DAG named `example_bash_operator` .

```
airflow tasks list example_bash_operator
```

Here `example_bash_operator` is the name of the DAG.

Try listing out the tasks for the DAG `tutorial` .

Exercise 5 - Pause/Unpause a DAG

Run the command below in the terminal to unpause a DAG.

```
airflow dags unpause tutorial
```

Here `tutorial` is the name of the DAG.

Pause a DAG.

```
airflow dags pause tutorial
```

Try to unpause the DAG named `example_branch_operator` .

Build DAG using Airflow

Objectives

After watching this video, you will be able to:

- Interpret an Airflow pipeline as a Python script that defines an Airflow DAG object
- List the key components of a DAG definition file
- Create tasks by instantiating operators in your DAG definition file
- Set up dependencies amongst tasks

Airflow DAG definition script

Python script blocks:

- Library imports
- DAG arguments
- DAG definition
- Task definitions
- Task pipeline

Build an Airflow pipeline

```
$ simple_example_DAG.py
```

↓ contains this

Print Greeting → Print Date & Time

5 sec ← schedule to repeat process every 5 s

Airflow pipeline script

Python script blocks:

- Python library imports

```
from airflow import DAG                                  for DAG
from airflow.operators.bash_operator import BashOperator  for bash
import datetime as dt                                    for time to schedule
```

Airflow pipeline script

Python script containing:

- Python library imports
- DAG arguments

```
default_args = {
    'owner': 'me',                                  ex
    'start_date': dt.datetime(2021, 7, 28),
    'retries': 1,                                   number of times it should keep trying
    'retry_delay': dt.timedelta(minutes=5),
}                                                   time to wait between subsequent tries → 5 min
```

as python dict

if it failing
↓
This ex is one if it does fail

Airflow pipeline script

Python script containing:

- Python library imports
- DAG arguments
- DAG definition

```
dag=DAG('simple_example',                           name of your DAG
        description='A simple example DAG',
        default_args=default_args,                  default arguments to apply to your DAG
        schedule_interval=dt.timedelta(seconds=5),  run repeatedly every 5 s
        )
```

This Ex is from above Block

Airflow pipeline script

Task Definitions

```
task1 = BashOperator(                               respective ID
    task_id='print_hello',
    bash_command='echo \'Greetings. The date and time are \'',   call bash
    dag=dag,                                        Assign DAG Block
)

task2 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag,
)
```

Airflow pipeline script

Python script containing:

- Python library imports
- DAG arguments
- DAG definition
- Task definitions
- Task pipeline

```
task1 >> task2
```

downstream from task one

mean

'print_hello' 'print_date'

This one runs first → then

LAB

Start Apache Airflow in the lab environment.

```
start_airflow
```

Please be patient, it will take a few minutes for airflow to get started.

Exercise 3 - Explore the anatomy of a DAG

An Apache Airflow DAG is a python program. It consists of these logical blocks.

- Imports
- DAG Arguments
- DAG Definition
- Task Definitions
- Task Pipeline

A typical `imports` block looks like this.

```
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow.models import DAG
# Operators; you need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago
```

A typical `DAG Arguments` block looks like this.

```
#defining DAG arguments

# You can override them on a per-task basis during operat
default_args = {
    'owner': 'Lavanya',
    'start_date': days_ago(0),
    'email': ['lavanya@somemail.com'],
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

to retry again

DAG arguments are like settings for the DAG.

The above settings mention

- the owner name,
- when this DAG should run from: days_age(0) means today,
- the email address where the alerts are sent to,
- the number of retries in case of failure, and
- the time delay between retries.

The other options that you can include are:

- 'queue' : The name of the queue the task should be a part of.
- 'pool' : The pool that this task should use.
- 'email_on_failure' : Whether an email should be sent to the owner on failure
- 'email_on_retry' : Whether an email should be sent to the owner on retry
- 'priority_weight' : Priority weight of this task against other tasks.
- 'end_date' : End date for the task
- 'wait_for_downstream' : Boolean value indicating whether it should wait for downtime
- 'sla' : Time by which the task should have succeeded. This can be a timedelta object.
- 'execution_timeout' : Time limit for running the task. This can be a timedelta object.
- 'on_failure_callback' : Some function, or list of functions to call on failure
- 'on_success_callback' : Some function, or list of functions to call on success
- 'on_retry_callback' : Another function, or list of functions to call on retry
- 'sla_miss_callback' : Yet another function, or list of functions when sla is missed
- 'on_skipped_callback' : Some function to call when the task is skipped
- 'trigger_rule' : Defines the rule by which the generated task gets triggered

A typical `DAG definition` block looks like this.

```
# define the DAG
dag = DAG(
    dag_id='sample-etl-dag',
    default_args=default_args,
    description='Sample ETL DAG using Bash',
    schedule_interval=timedelta(days=1),
)
```

Here you are creating a variable named dag by instantiating the DAG class with the following parameters.

`sample-etl-dag` is the ID of the DAG. This is what you see on the web console.

you are passing the dictionary `default_args` , in which all the defaults are defined.

`description` helps us in understanding what this DAG does.

`schedule_interval` tells us how frequently this DAG runs. In this case every day. (`days=1`).

A typical `task definitions` block looks like this:

```
# define the tasks

# define the first task named extract
extract = BashOperator(
    task_id='extract',
    bash_command='echo "extract"',
    dag=dag,
)

# define the second task named transform
transform = BashOperator(
    task_id='transform',
    bash_command='echo "transform"',
    dag=dag,
)

# define the third task named load

load = BashOperator(
    task_id='load',
    bash_command='echo "load"',
    dag=dag,
)
```

A task is defined using:

- A task_id which is a string and helps in identifying the task.
- What bash command it represents.
- Which dag this task belongs to.

A typical `task pipeline` block looks like this:

```
# task pipeline
extract >> transform >> load
```

Task pipeline helps us to organize the order of tasks.

Here the task `extract` must run first, followed by `transform` , followed by the task `load` .

Exercise 4 - Create a DAG

Let us create a DAG that runs daily, and extracts user information from */etc/passwd* file, transforms it, and loads it into a file.

This DAG has two tasks `extract` that extracts fields from `/etc/passwd` file and `transform_and_load` that transforms and loads data into a file.

```
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow.models import DAG
# Operators; you need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago

#defining DAG arguments

# You can override them on a per-task basis during operat
default_args = {
    'owner': 'your_name_here',
    'start_date': days_ago(0),
    'email': ['your email here'],
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# defining the DAG

# define the DAG
dag = DAG(
    'my-first-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)

# define the tasks

# define the first task

extract = BashOperator(
    task_id='extract',
    bash_command='cut -d":" -f1,3,6 /etc/passwd > /home/passwd > /home/project/airflow/dags/extracted-data.txt',
    dag=dag,
)

# define the second task
transform_and_load = BashOperator(
    task_id='transform',
    bash_command='tr ":" "," < /home/project/airflow/dags ect/airflow/dags/extracted-data.txt > /home/project/airflo
    dag=dag,
)

# task pipeline
extract >> transform_and_load
```

Create a new file by choosing File->New File and name it `my_first_dag.py`. Copy the code above and paste it into `my_first_dag.py`.

Exercise 5 - Submit a DAG

Submitting a DAG is as simple as copying the DAG python file into `dags` folder in the `AIRFLOW_HOME` directory.

Airflow searches for Python source files within the specified `DAGS_FOLDER`. The location of `DAGS_FOLDER` can be located in the airflow.cfg file, where it has been configured as `/home/project/airflow/dags`.

airflow > airflow.cfg

```
[core]
# The folder where your airflow pipelines live, most likely a
# subfolder in a code repository. This path must be absolute.
dags_folder = /home/project/airflow/dags
```

Airflow will load the Python source files from this designated location. It will process each file, execute its contents, and subsequently load any DAG objects present in the file.

Therefore, when submitting a `DAG`, it is essential to position it within this directory structure. Alternatively, the `AIRFLOW_HOME` directory, representing the structure `/home/project/airflow`, can also be utilized for DAG submission.

theia@theiadocker-lavanyas: /home/project

```
theia@theiadocker-lavanyas:/home/project$ echo $AIRFLOW_HOME
/home/project/airflow
```

Open a terminal and run the command below to submit the DAG that was created in the previous exercise.

```
cp my_first_dag.py $AIRFLOW_HOME/dags
```

Verify that your DAG actually got submitted.

Run the command below to list out all the existing DAGs.

```
airflow dags list
```

Verify that `my-first-dag` is a part of the output.

```
airflow dags list|grep "my-first-dag"
```

You should see your DAG name in the output.

Run the command below to list out all the tasks in `my-first-dag`.

```
airflow tasks list my-first-dag
```

You should see 2 tasks in the output.

LAB

Exercise 2 - Create a basic shell script

Here we will define a shell script `extract_transform_load.sh` which will define a pipeline of tasks such as

- extract
- transform
- load

For now, let the shell script have basic echo statements for `extract` , `transform` and `load` .

```
#!/bin/bash

echo "Extract data"

echo "Transform data"

echo "Load data"
```

In the next section we will define a `DAG` to call and execute the shell script.

A typical `task definitions` block looks like this:

```
# define the tasks

# define the task named extract_transform_and_load to call the shell script
extract_transform_and_load = BashOperator(
    task_id='extract_transform_and_load',
    bash_command='/home/project/airflow/dags/extract_transform_load.sh "',
    dag=dag,
)
```

A task is defined using:

- A task_id which is a string and helps in identifying the task.
- What bash command it represents. Here we are calling the shell script `extract_transform_load.sh` which we previously defined
- Which dag this task belongs to.

A typical `task pipeline` block looks like this:

```
# task pipeline
extract_transform_and_load
```

When we execute the task `extract_transform_and_load` the code in the shell script gets executed.

Exercise 3 - Explore the anatomy of a DAG

An Apache Airflow DAG is a python program. It consists of these logical blocks.

- Imports
- DAG Arguments
- DAG Definition
- Task Definitions
- Task Pipeline

A typical `imports` block looks like this.

```
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago
```

A typical `DAG Arguments` block looks like this.

```
#defining DAG arguments

# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': True,
    'email_on_retry': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

DAG arguments are like settings for the DAG.

The above settings mention

- the owner name,
- when this DAG should run from: days_age(0) means today,
- the email address where the alerts are sent to,
- whether alert must be sent on failure,
- whether alert must be sent on retry,
- the number of retries in case of failure, and
- the time delay between retries.

A typical `DAG definition` block looks like this.

```
# define the DAG
dag = DAG(
    dag_id='sample-etl-dag',
    default_args=default_args,
    description='Sample ETL DAG using Bash',
    schedule_interval=timedelta(days=1),
)
```

Here we are creating a variable named `dag` by instantiating the DAG class with the following parameters.

`sample-etl-dag` is the ID of the DAG. This is what you see on the web console.

We are passing the dictionary `default_args` , in which all the defaults are defined.

`description` helps us in understanding what this DAG does.

`schedule_interval` tells us how frequently this DAG runs. In this case every day. (`days=1`).

Exercise 4 - ETL process on a `/etc/passwd` file

Here we will first

- Create a new shell script called `my_first_dag.sh` to perform the ETL process.
- Create a `DAG` file `my_first_dag.py` which will run daily and defines a task `etl` to call the shell script `my_first_etl.sh` .
- Submit the `DAG`

Create a new shell script `my_first_dag.sh` by selecting **File->New File**.

- The shell script extracts the first,third and sixth fields from `/etc/passwd` file using the `cut` command and writes to a new file `extracted-data.txt`
- Next the `extracted-data.txt` is transformed to a `csv` file and loaded into a new file called `transformed-data.csv` using `tr` command.

Copy the code below in the shell script.

```
#!/bin/bash
echo "extract_transform_and_load"
cut -d":" -f1,3,6 /etc/passwd > /home/project/airflow/dags/extracted-data.txt

tr ":" "," < /home/project/airflow/dags/extracted-data.txt > /home/project/airflow/dags/transformed-data.csv
```

Click `Save` to save the shell script.

Create a new `DAG` file `my_first_dag.py` by selecting **File->New File**.

This DAG has one task `etl` that calls the shell script `my_first_dag.sh`

```
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago

#defining DAG arguments

# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# defining the DAG

# define the DAG
dag = DAG(
    'my-first-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)

# define the task **extract_transform_and_load** to call shell script

#calling the shell script
extract_transform_load = BashOperator(
    task_id="extract_transform_load",
    bash_command="/home/project/airflow/dags/my_first_dag.sh ",
    dag=dag,
)

# task pipeline
extract_transform_load
```

Exercise 5 - Submit a DAG

Submitting a DAG is as simple as copying the DAG python file into `dags` folder in the `AIRFLOW_HOME` directory.

Open a terminal and run the command below to submit the DAG that was created in the previous exercise.

Note: While submitting the dag that was created in the previous exercise, use **sudo** in the terminal before the command used to submit the dag.

```
cp my_first_dag.py $AIRFLOW_HOME/dags
```

Next, run the command below one by one to submit shell script in the dags folder and to change the permission for reading shell script.

```
cp my_first_dag.sh $AIRFLOW_HOME/dags
cd airflow/dags
chmod 777 my_first_dag.sh
```

Verify that our DAG actually got submitted.

Run the command below to list out all the existing DAGs.

```
airflow dags list
```

Verify that `my-first-dag` is a part of the output.

```
airflow dags list|grep "my-first-dag"
```

You should see your DAG name in the output.

Run the command below to list out all the tasks in `my-first-dag` .

```
airflow tasks list my-first-dag
```

You should see 1 task in the output.

Airflow Monitoring and Logging

Objectives

After watching this video, you will be able to:

- Use logging capabilities to monitor the task status and diagnose problems with DAG runs
- Explain accessing emitted metrics such as counters, gauges, and timers

Logging

by default, Airflow logs are saved to local file system as log files

Airflow log files

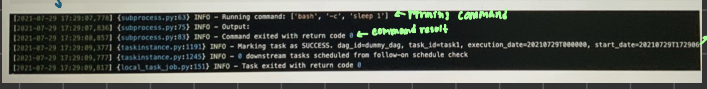
Default log file location:

log file are organized by dag id

```
logs/dag_id/task_id/execution_date/try_number.log
logs/dummy_dag/task1/2021-07-29T00:17:00+00:00/1.log
```

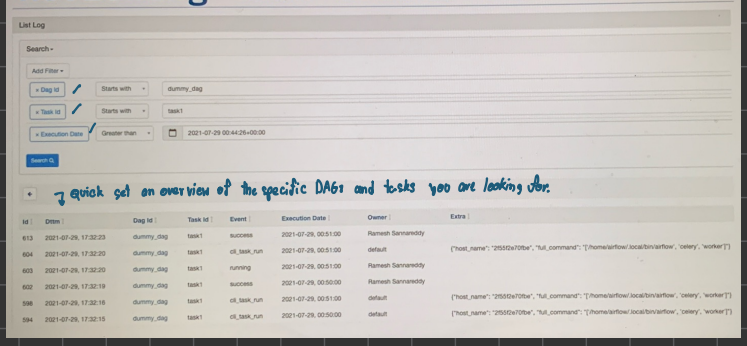
if you want to find the log of the first execution try of task1 in a dummy DAG

in first log file



task result

Reviewing task events via UI



Monitoring metrics

Airflow produces three different types of metrics for checking

- Counters: Metrics that always increase
  - Total count of task instances failures
  - Total count of task instances successes
- Gauges: Metrics that may fluctuate
  - Number of running tasks
  - DAG bag size, or number of DAGs in production
- Timers: Metrics related to time duration
  - Milliseconds to finish a task
  - Milliseconds to reach a success or failed state

Storing and analyzing metrics