APACHE AIRFLOW

Author: Sony Shrestha

Date: 9th October, 2020

Table of Contents

[Introduction to Airflow 4](#_Toc53140238)

[Basic Terminologies Used in Airflow 4](#_Toc53140239)

[Dag 4](#_Toc53140240)

[Operator 4](#_Toc53140241)

[Task 4](#_Toc53140242)

[Workflow 4](#_Toc53140243)

[Airflow Installation 5](#_Toc53140244)

[Configuring airflow.cfg 5](#_Toc53140245)

[Configuring locations 5](#_Toc53140246)

[Setting up database connection 6](#_Toc53140247)

[Configuring Mail 7](#_Toc53140248)

[Load Example DAG 7](#_Toc53140249)

[Executor 9](#_Toc53140250)

[Other configs 9](#_Toc53140251)

[Airflow CLI Comands 9](#_Toc53140252)

[Create user 9](#_Toc53140253)

[Reset Database 9](#_Toc53140254)

[Initialize database 9](#_Toc53140255)

[Run Airflow 10](#_Toc53140256)

[Print list of active dags 10](#_Toc53140257)

[Test Single Task 10](#_Toc53140258)

[Print list of tasks of certain DAG 10](#_Toc53140259)

[Print the hierarchy of tasks 10](#_Toc53140260)

[Create Connection 10](#_Toc53140261)

[Start Airflow as background process 11](#_Toc53140262)

[Airflow State 13](#_Toc53140263)

[Dagrun 13](#_Toc53140264)

[Task 13](#_Toc53140265)

[LifeCycle of Airflow Task State 13](#_Toc53140266)

[Writing DAG Program 16](#_Toc53140267)

[Import module 16](#_Toc53140268)

[Default arguments 17](#_Toc53140269)

[Instantiate a dag 18](#_Toc53140270)

[Tasks 18](#_Toc53140271)

[Setting up Dependencies 19](#_Toc53140272)

[Making config driven 19](#_Toc53140273)

[Operators 23](#_Toc53140274)

[Properties of Operators: 23](#_Toc53140275)

[Different types of Operator 23](#_Toc53140276)

[Importing Operators 23](#_Toc53140277)

[Operatorwise Explanation 24](#_Toc53140278)

[BashOperator 24](#_Toc53140279)

[Python Operator 24](#_Toc53140280)

[MySqlOperator 24](#_Toc53140281)

[Email Operator 26](#_Toc53140282)

[Dummy Operator 26](#_Toc53140283)

[Branch Python Operator 27](#_Toc53140284)

[Sensor 28](#_Toc53140285)

[Definition 28](#_Toc53140286)

[Importing sensor 28](#_Toc53140287)

[Terms associated with Sensor 28](#_Toc53140288)

[Explanation of Different Sensor 28](#_Toc53140289)

[File Sensor 28](#_Toc53140290)

[Sql Sensor 29](#_Toc53140291)

[External Task Sensor 29](#_Toc53140292)

[Additional Details 30](#_Toc53140293)

[Run DAG from failed task onwards 30](#_Toc53140294)

[Sending Custom mails with attachment 32](#_Toc53140295)

[Installation of Oracle Client in Linux 35](#_Toc53140296)

[Installation of Oracle Client in Windows 35](#_Toc53140297)

# Introduction to Airflow

An platform to programmatically author, schedule and monitor workflows.

Features:

1. It is a job scheduler that executes a job based on their dependencies.
2. In Airflow, airflow is written as code in form of DAGs (Directed Acyclic Graphs).
3. Interactive UI Interface

# Basic Terminologies Used in Airflow

## Dag

A DAG (Directed Acyclic Graph) is unidirectional, acyclic graph connecting edges, in which each node refer to a task and edges define dependencies between these tasks.

## Operator

Operator determines what actually needds to be done when a DAG runs.

## Task

When an operator is instantiated, it is called task.

## Workflow

Workflow is sequence of task arranged in a control dependency.

# Airflow Installation

The easiest way to install airflow according to documentation is:

**pip install apache-airflow**

Once airflow is installed, airflow.cfg and dags folder will be created in location where airflow has been installed.

**In server 8:**

/root/airflow/dags

/root/airflow/airflow.cfg

To initialize database

Airflow initdb

sudo yum install python3-devel mysql-devel

sudo yum install python3-devel mariadb-devel

pip install mysqlclient

To create user

Airflow create-user –r \*\*\* -f \*\*\* -l \*\*\*\* -p \*\*\* -email \*\*\*\*

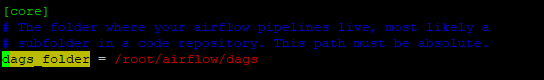
## Configuring airflow.cfg

In the airflow home directory, there is a file named airflow.cfg.

This config file is used to change various airflow settings.

### Configuring locations

First we have to set base locations for logs and DAGS. Find the section [core] and you can set the folders where you want your logs and DAGs to be.



[core]

dags\_folder = /data/airflow/dags

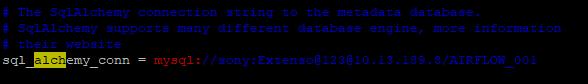
base\_log\_folder = /data/airflow/logs

### Setting up database connection

In the section [core], edit sql\_alchemy\_conn in the following format:

By default, airflow uses SQLite. Follow this document to set MYSQL as the database backend. This database is used as metadata for airflow.

NOTE: Make sure you have installed sqlalchemy and mysqlclient (if you are using mysql as backend) in the same python environment where you have airflow installed.



[core]

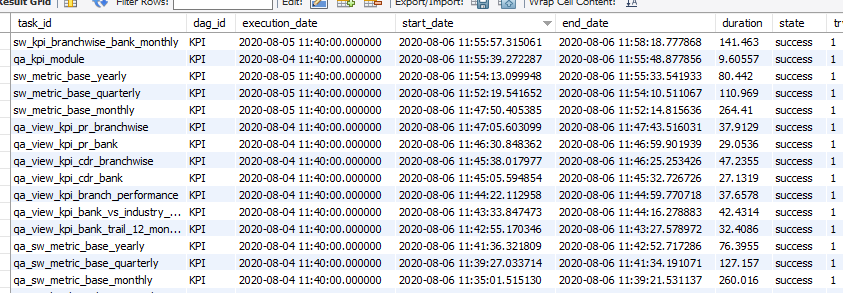
sql\_alchemy\_conn = <driver>://<user>:<password>@<host>/<airflow\_db>

#example

#sql\_alchemy\_conn = mysql://siddhi:9865@localhost/airflow

**Note:**

Once this connection is provided and database is initialized, airflow\_db is created and all information about state of tasks is available in task\_instance table of airflow\_db.



### Configuring Mail

You will have to find a section [smtp], and edit them according to the smtp server used.

The following configuration is used to setup a gmail smtp.



[smtp]

smtp\_host = smtp.gmail.com

smtp\_starttls = True

smtp\_ssl = False

smtp\_user = noreply.extenso@gmail.com

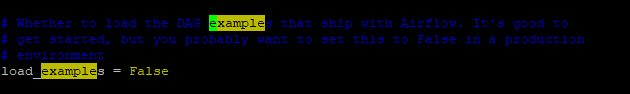
smtp\_password = extenso@123

smtp\_port = 587

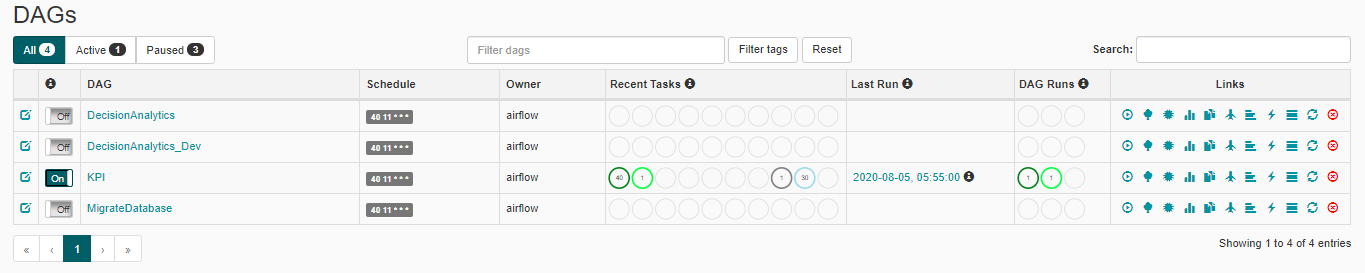
smtp\_mail\_from = noreply.extenso@gmail.com

### Load Example DAG

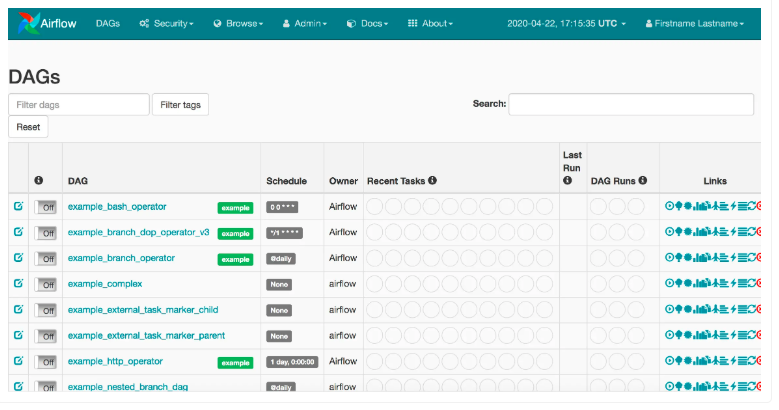
We can choose to whether load sample examples or not by changing following setting in airflow.cfg.



Following is state when load\_examples=False

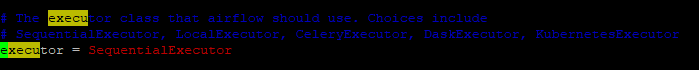


Following is state when load\_examples = True

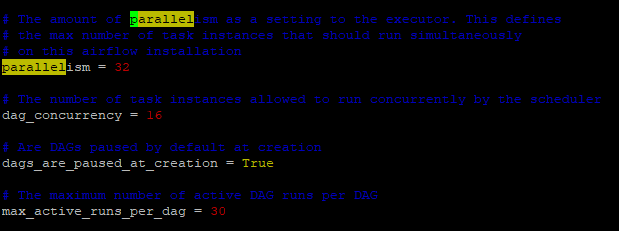


### Executor

We can decide whether tasks in airflow will be executed sequentially or parallel by changing executor in airflow.cfg file.



### Other configs



# Airflow CLI Comands

## Create user

airflow create\_user -u admin123 -r admin -e sony.shrestha@extensodata.com -f sony -l shrestha -p admin123

## Reset Database

airflow resetdb

Note:

After we reset database, we must create user because created user will be removed from our database.

## Initialize database

Initialize the database.

**airflow initdb**

## Run Airflow

1. Start the scheduler

**airflow scheduler**

1. Start the webserver

**airflow webserver -p 8080**

## Print list of active dags

airflow list\_dags

## Test Single Task

**airflow test airflow\_name task\_name date**

airflow test MigrateDatabase create\_database\_bkp 2020-07-28

## Print list of tasks of certain DAG

airflow list\_tasks KPI

## Print the hierarchy of tasks

airflow list\_tasks KPI –tree

## Create Connection

airflow connections --add --conn\_id 'server\_8' --conn\_uri 'mysql://sony:Extenso@123@10.13.189.8'

# Start Airflow as background process

In CLI,

Run

cd /etc/systemd/system

Create two files

vi airflow\_web.service

vi airflow\_scheduler.service

In airflow\_web.service

[Unit]

Description=Airflow webserver.

[Service]

Environment="PATH=/data/airflow/airflow\_venv/bin/"

Type=simple

ExecStart=/data/airflow/airflow\_venv/bin/airflow webserver -p 8081

[Install]

WantedBy=multi-user.target

In airflow\_scheduler.service

[Unit]

Description=Airflow webserver.

[Service]

Environment="PATH=/data/airflow/airflow\_venv/bin/:$PATH:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin"

Type=simple

ExecStart=/data/airflow/airflow\_venv/bin/airflow scheduler

[Install]

WantedBy=multi-user.target

Note: to get airflow path do **which airflow**

Run

sudo systemctl daemon-reload

To start and enable service

Sudo systemctl start airflow\_web

Sudo systemctl start airflow\_scheduler

Sudo systemctl enable airflow\_web

Sudo systemctl enable airflow\_scheduler

To stop service

Sudo systemctl stop airflow\_web

Sudo systemctl stop airflow\_scheduler

# Airflow State

There are two main categories for state in airflow:

1. Dagrun
2. Task

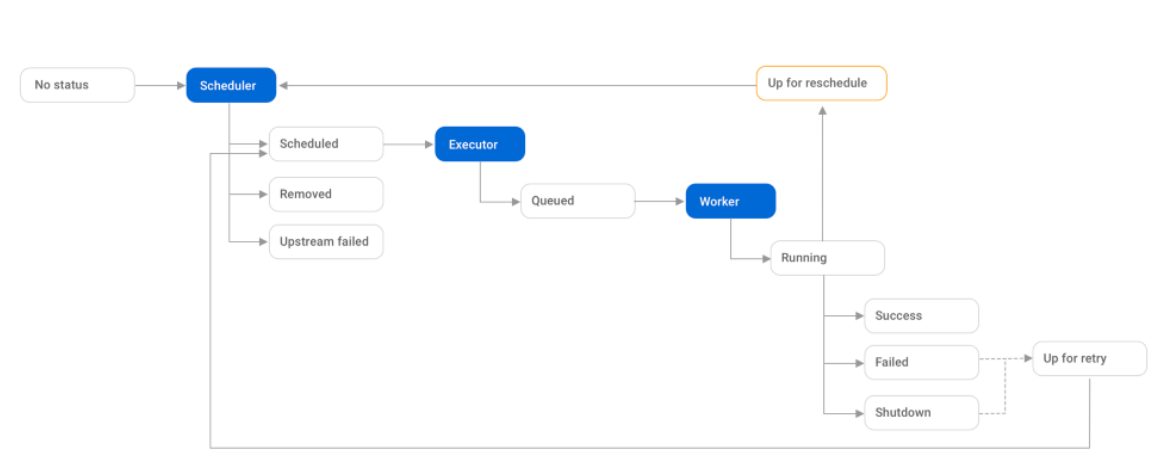
## Dagrun

States included in Dagrun are:

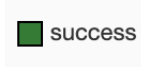
1. Success
2. Running
3. Failed

## Task

### LifeCycle of Airflow Task State



1. Success

This is a state when task runs successfully without any error.

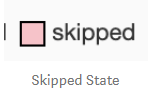
1. Running

This is a state when airflow scheduler has already assigned task to executor.

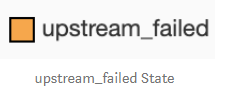
1. Failed

This is a state when there is some error in task.

1. Skipped

This is a state when execution of certain task is skipped like in case of Branch Operator, based on branch condition, task on one end of branch is executed and task in other end of branch is skipped.

1. Upstream\_failed

This state of task refers that t2 was not executed because t1 failed and dependency between t1 and t2 was sent such a way that t2 needs to be executed only when t1 executes successfully.

1. Up\_for\_retry
2. Up\_for\_reschedule
3. Queued

This state states that task is waiting for a slot in an executor.

1. None

This stste states that current state of task is not known.

1. Scheduled

This state states that task has been scheuled by scheduler to sen to executor

# Writing DAG Program

There are only 5 steps you need to remember to write an Airflow DAG or workflow:

* Step 1: Importing modules
* Step 2: Default Arguments
* Step 3: Instantiate a DAG
* Step 4: Tasks
* Step 5: Setting up Dependencies

## Import module

#importing default modules

import os

import shutil

import pendulum

from datetime import timedelta, datetime

#importing airflow modules

from airflow import DAG

from airflow.utils import timezone

from airflow.utils.dates import days\_ago

from airflow.operators.bash\_operator import BashOperator

from airflow.operators.sensors import SqlSensor

#importing custom modules

from utilities.utilities import \*

from utilities.variables import \*

local\_tz = pendulum.timezone('Asia/Kathmandu')

start\_date = datetime(\*\*START\_DATE, tzinfo=local\_tz)

## Default arguments

default\_args = {

'owner': 'airflow',

'depends\_on\_past': False,

'start\_date': start\_date,

'email': EMAIL\_LIST,

'email\_on\_failure': False,

'email\_on\_retry': False,

# 'retries': 1,

# 'retry\_delay': timedelta(minutes=5),

# 'queue': 'bash\_queue',

# 'pool': 'backfill',

# 'priority\_weight': 10,

# 'end\_date': datetime(2016, 1, 1),

# 'wait\_for\_downstream': False,

# 'dag': dag,

# 'sla': timedelta(hours=2),

# 'execution\_timeout': timedelta(seconds=300),

# 'on\_failure\_callback': some\_function,

# 'on\_success\_callback': some\_other\_function,

# 'on\_retry\_callback': another\_function,

# 'sla\_miss\_callback': yet\_another\_function,

# 'trigger\_rule': 'all\_success'

}

## Instantiate a dag

dag = DAG(

'DAProductionPoint',

default\_args=default\_args,

description='Migrate database',

schedule\_interval=CRON\_EXPRESSION\_DA\_Production\_Point,

template\_searchpath=[SQL\_SCRIPT\_LOCATION],

catchup=False

)

## Tasks

Note:

task\_id

dag =dag is must

**Example:**

slave\_task2 = BashOperator(

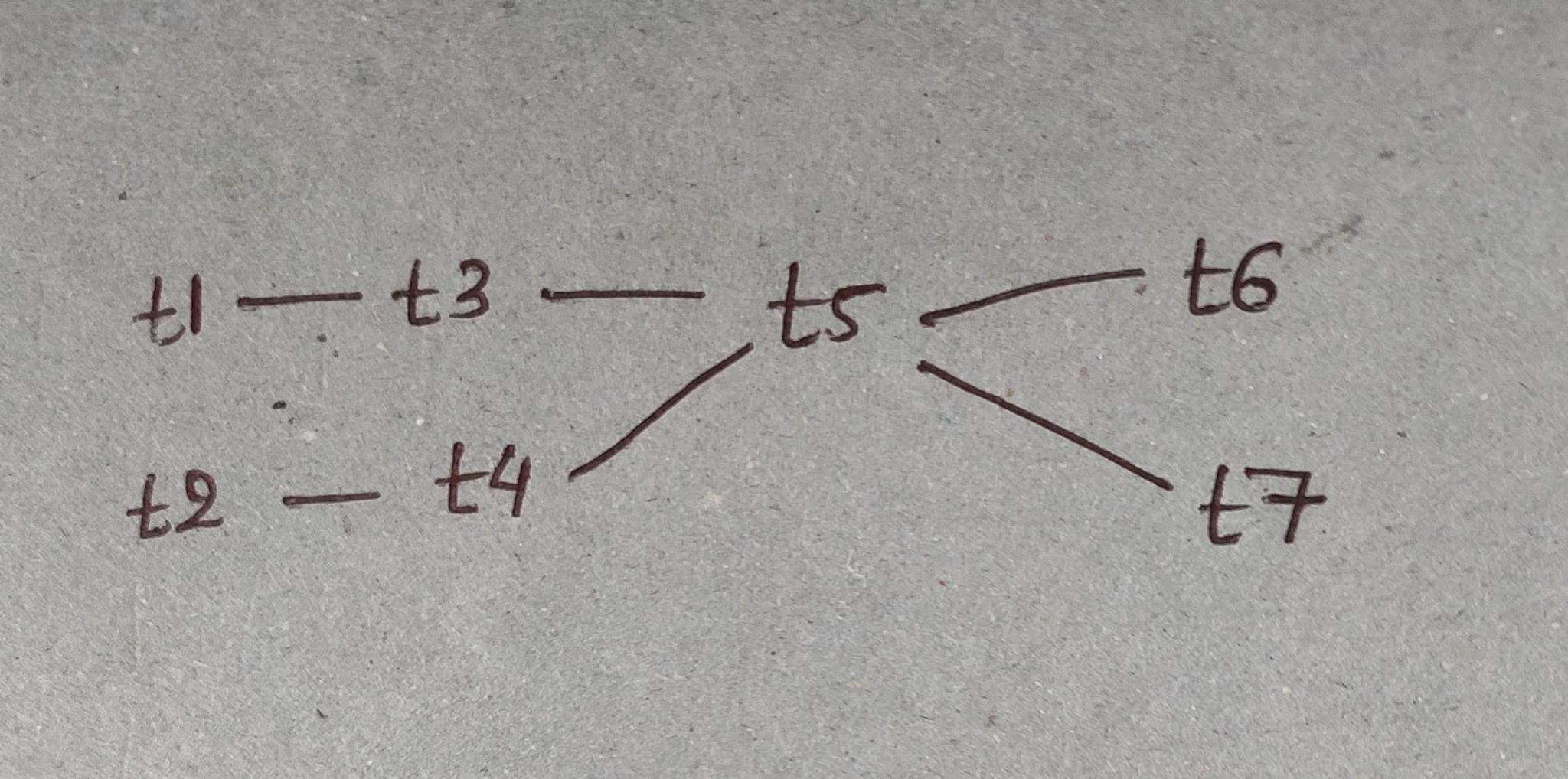
task\_id='slave\_task2',

bash\_command='echo slavetask2 ',

dag=dag

)

## Setting up Dependencies



t1 >> t3

t2 >> t4

[t3, t4] >> t5

T5 >> [t6, t7]

## Making config driven

Importing Custom Modules

Besides this,

We can also import custom module

*#importing custom modules*

**from** **kpi.utilities.utilities** **import** \*

**from** **kpi.utilities.variables** **import** \*

This can be helpful to make variables used in dag program config driven

Suppose, config file is:



In utilities/ variables.py

import os

from sqlalchemy import create\_engine

from utilities.db\_con import \*

file\_path = os.path.dirname(os.path.dirname(os.path.realpath(\_\_file\_\_)))

def initialize\_json\_config():

import json

global PYTHON\_LOCATION

with open(os.path.abspath(file\_path + '/config/config.json'), 'r') as js:

js\_conf = json.load(js)

CLIENT=js\_conf['CLIENT']

mysql\_url=get\_con\_url(\*\*js\_conf["mysql\_config"])

oracle\_url=get\_oracle\_con\_url(\*\*js\_conf["oracle\_config"])

def get\_con\_url(\*\*conf):

return conf["driver"]+"://"+conf["username"]+":"+conf["password"]+"@"+conf["host"]

def get\_oracle\_con\_url(\*\*client\_con):

client\_con\_url = ("{driver}://{username}:{password}@(DESCRIPTION = "

"(LOAD\_BALANCE=on) (FAILOVER=ON) "

"(ADDRESS = (PROTOCOL = TCP)(HOST = {host})(PORT = {port})) "

"(CONNECT\_DATA = (SERVER = DEDICATED) (SERVICE\_NAME = {sid})))")

return client\_con\_url.format(username=client\_con['username'],

password=client\_con['password'],

host=client\_con['host'],

sid=client\_con['sid\_name'],

port=client\_con['port'],

driver=client\_con['driver'])

initialize\_json\_config()

In db\_con.py

from sqlalchemy.orm import sessionmaker

from sqlalchemy import create\_engine

def create\_session(connection\_url):

global session

en\_flag = 0

ses\_flag = 0

try:

if type(connection\_url) is str:

engine = create\_engine(connection\_url)

en\_flag = 1

else:

engine = connection\_url

Session = sessionmaker(bind=engine)

session = Session()

ses\_flag = 1

except Exception as e:

if en\_flag == 1:

engine.dispose()

if ses\_flag == 1:

session.close()

raise e

return session

# Operators

An operator describes a single task in a workflow.

Operator determines what actually needs to be done when a DAG runs.

## Properties of Operators:

Operators are atomic (but not always)

Once an operator is instantiated, it is refered to a task.

Every operator is derived from Airflow’s BaseOperator.

Two operators can communicate with each other using feature called Xcom.

## Different types of Operator

1. Action Operator
   1. Action operators are used to perform an action.
   2. Example: PythonOperator, BashOperator
2. Sensor Operator
   1. Sensor Operator keeps executing at certain time interval.
   2. They succeed when a criteria is met and fail when timeout occurs.
   3. Example: FileSensor, SqlSensor
3. Transfer Operator
   1. Transfer Operator moves/ transfers data from one location to another location.
   2. Example: MySqlToHiveTransfer

## Importing Operators

# importing operator in airflow

from airflow.operators.bash\_operator import BashOperator

from airflow.operators.python\_operator import PythonOperator

from airflow.operators.mysql\_operator import MySqlOperator

from airflow.operators.email\_operator import EmailOperator

from airflow.operators.dummy\_operator import DummyOperator

from airflow.operators.python\_operator import BranchPythonOperator

## Operatorwise Explanation

### BashOperator

Bash operator is used to perform certain bash commands.

**Example:**

t1 = BashOperator(

task\_id='BashOperatorExample',

bash\_command='echo "Hello World"',

dag=dag,

)

### Python Operator

Python Operator is used to call python function.

**Example:**

t2 = PythonOperator(

task\_id='PythonOperatorExample',

python\_callable=python\_function,

dag=dag,

)

Note:

This python callable function can be within current dag file or we can import python function from other location:

# import python functions

from Airflow\_Tutorial.python\_files.main import \*

### MySqlOperator

It is used to execute Mysql commands.

**Example:**

t3 = MySqlOperator(

task\_id='MySqlOperatorExample',

mysql\_conn\_id="server\_8",

sql="create\_table.sql", # sql= “ update……”

dag=dag

)

Note:

It is possible to run single mysql command or commands from a sql file.

For this:

While creating dag object following line should be included:

dag=DAG(

'AirflowTutorial',

default\_args=default\_args,

description='Understanding DAG Program',

schedule\_interval='@daily',

template\_searchpath=['/root/airflow/dags/Airflow\_Tutorial/sql\_files'],

catchup=False )

This line specifies that sql file is present inside this template location.

**Connection needs to be created as follows:**

airflow connections --add --conn\_id 'server\_8' --conn\_uri 'mysql://sony:Extenso@123@10.13.189.8'

### Email Operator

It is used to send mail.

**Example:**

t4 = EmailOperator(

task\_id='EmailOperatorExample',

subject='Test: Email Operator',

cc='sony.sth8@gmail.com',

bcc='pop.son4p@gmail.com',

to='sony.shrestha@extensodata.com',

html\_content=""" <h1> Email Operator Passed </h1> <br> <br> Email sent successfully <br> """,

files=['/root/airflow/dags/Airflow\_Tutorial/sql\_files/create\_table.sql'],

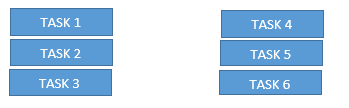
dag=dag

)

### Dummy Operator

It does nothing.

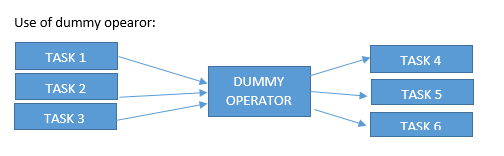
**Use of dummy opearor:**



Suppose TASK 4,5,6 needs to be executed only after TASK1,2,3 are executed successfully.

It is ddifficult estalish link without using Dumy Operator.

Now, using dummy Operator:



**Example:**

t5 = DummyOperator(

task\_id='DummyOperatorExample',

dag=dag

)

### Branch Python Operator

It is used to peform branching operation.

**Example:**

def decide\_branch():

if sth:

return "t7"

else:

return "t8"

t6=BranchPythonOperator(

task\_id='BranchPythonOperatorExample',

python\_callable=decide\_branch,

dag=dag

)

t7=BashOperator(

task\_id='on\_success',

bash\_command='echo "SUCCESS"',

dag=dag

)

t8=BashOperator(

task\_id='on\_failure',

bash\_command='echo "FAILURE"',

dag=dag

)

# Sensor

## Definition

Sensor execute certain task in certain interval of time.

They succeed when a condition is met an fails when timeout occurs.

## Importing sensor

# importing airflow sensors

from airflow.contrib.sensors.file\_sensor import FileSensor

from airflow.operators.sensors import ExternalTaskSensor

from airflow.operators.sensors import SqlSensor

## Terms associated with Sensor

1. **poke\_interval:** time in seconds that the given job should wait in between each tries
2. **timeout:** time in second before task fails or timeout occurs
3. **soft\_fail:** If it is set to True, it will mark tas as SKIPPED on failure

## Explanation of Different Sensor

### File Sensor

It will check for existence of certain file until timeout occurs.

Example:

file\_sensor = FileSensor(

task\_id='file\_sensor',

filepath='/home/fonecredit/airflow/dags/test.txt',

dag=dag

)

### Sql Sensor

It will execute a sql statement until a condition is met.

It will keep trying until success or failure criteria are met of if the first cell is not in (0, ‘0’ , ‘’ , None).

An allow\_null parameter will exclude None/Null results from failure criteria.

**Example:**

check\_signal = SqlSensor(

task\_id='check\_signal',

conn\_id='mysql\_server',

sql='SELECT flag FROM '+master\_db\_name+'.md\_check\_signal e"',

dag=dag,

timeout=300,

poke\_interval=10,

soft\_fail=True

)

### External Task Sensor

It will wait until task in another dag succeeds.

Suppose, there is master dag with dag\_id=master\_dag and last task id as master\_task

Then, to run slave dag only if master ag succeeds, we can use External Task Sensor.

**Example:**

sensor\_task = ExternalTaskSensor(

task\_id = 'sensor',

external\_dag\_id='master\_dag',

external\_task\_id='master\_task',

dag=dag

)

slave\_task1 = BashOperator(

task\_id='slave\_task1',

bash\_command='echo slavetask1 ',

dag=dag,

trigger\_rule = 'all\_success'

)

slave\_task2 = BashOperator(

task\_id='slave\_task2',

bash\_command='echo slavetask2 ',

dag=dag,

trigger\_rule = 'all\_success'

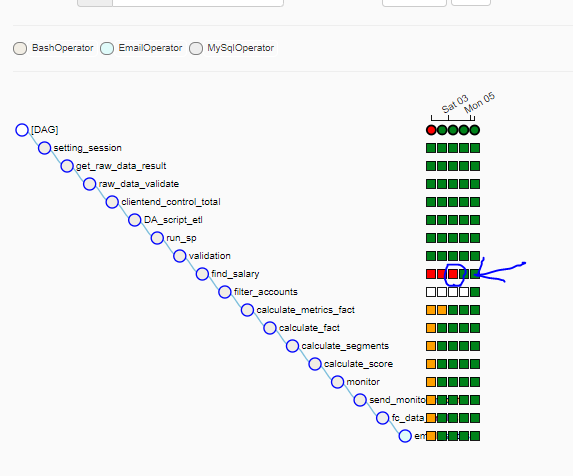
)

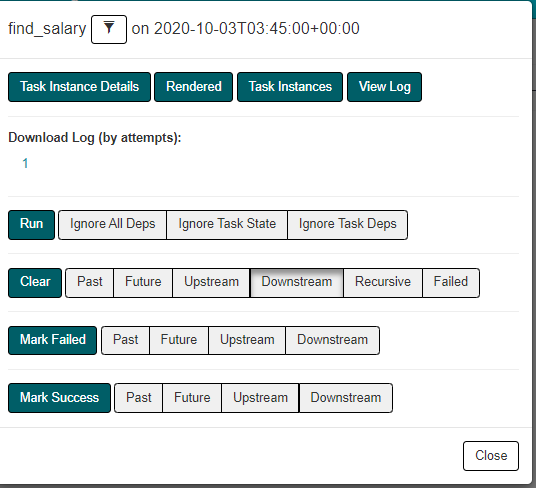
sensor\_task >> slave\_task1 >> slave\_task2

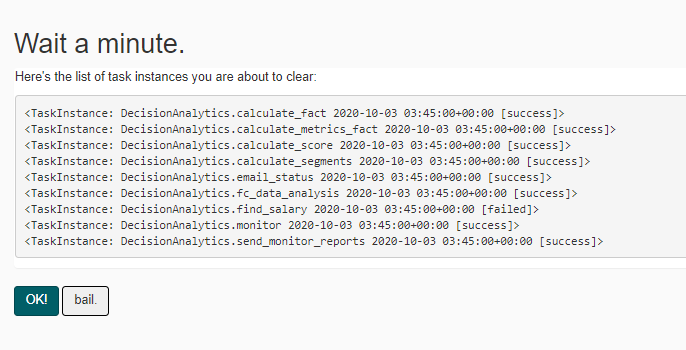
# Additional Details

## Run DAG from failed task onwards

1. Click on failed task instance in tree view.
2. Click on Clear with Downstream
3. Click on OK.







# Sending Custom mails with attachment

# importing default modules

import os

import shutil

# importing airflow modules

from airflow.utils.email import send\_email

# import email template

from html\_template.email\_template import \*

from utilities.variables import \*

# get log file

def get\_log\_location(contextDict):

log\_folder = contextDict['ti'].log\_filepath.split('.')[0]

if os.path.exists(log\_folder):

shutil.make\_archive(log\_folder, 'zip', log\_folder)

return log\_folder+'.zip'

else:

return contextDict['ti'].log\_filepath

# mail on success

def notify\_email\_success(contextDict, \*\*kwargs):

if 'file' in (contextDict['params']):

files = (contextDict['params']['file'])

else:

files = []

files.append(get\_log\_location(contextDict))

email\_list = (contextDict['params']['email\_list'])

title = "{} DA: {} {}".format(CLIENT,contextDict['task'].task\_id,success\_title)

body="{}<br>{}<br><br>{}<br><br>{}".format(

salutation,success\_body,contextDict['params']['task\_success\_msg'],regards)

files = [f for f in files if os.path.exists(f)]

if len(files) == 0:

send\_email(email\_list, title, body)

else:

send\_email(email\_list, title, body, files)

# mail on failure

def notify\_email\_failure(contextDict, \*\*kwargs):

if 'file' in (contextDict['params']):

files = (contextDict['params']['file'])

else:

files = []

files.append(get\_log\_location(contextDict))

email\_list = (contextDict['params']['email\_list'])

# email title.

title = "{} DA: {} {}".format(CLIENT,contextDict['task'].task\_id,failure\_title)

# email contents

body="{}<br>{}<br><br>{}<br><br>{}".format(

salutation,failure\_body,contextDict['params']['task\_failure\_msg'],regards)

files = [f for f in files if os.path.exists(f)]

if len(files) == 0:

send\_email(email\_list, title, body)

else:

send\_email(email\_list, title, body, files)

html\_template.py

global salutation

global success\_title

global success\_body

global failure\_title

global failure\_body

global regards

salutation="""Respected Sir/Madam,

"""

success\_title="""passed"""

success\_body="""Above task has been successfully executed.

"""

failure\_title="""failed"""

failure\_body="""Above task failed to execute. Find attached log file to debug error for respective task.

"""

regards="""With regards,

<br>

Airflow Automated Mail

"""

# Installation of Oracle Client in Linux

Steps:

1. Download the desired Instant Client ZIP files. All installations require a Basic or Basic Light package.
2. Unzip the packages into a single directory such as /opt/oracle/instantclient\_19\_3 that is accessible to your application. For example:
   1. cd /opt/oracle
   2. unzip instantclient-basic-linux.x64-19.3.0.0.0dbru.zip
3. Prior to version 18.3, create the appropriate links for the version of Instant Client. For example:
   1. cd /opt/oracle/instantclient\_12\_2
   2. ln -s libclntsh.so.12.1 libclntsh.so
   3. ln -s libocci.so.12.1 libocci.so
4. Install the operating system libaio package. This is called libaio1 on some Linux distributions. For example, on Oracle Linux, run:
   1. sudo yum install libaio
5. If Instant Client is the only Oracle Software installed on this system then update the runtime link path, for example:
   1. sudo sh -c "echo /opt/oracle/instantclient\_19\_3 > \ /etc/ld.so.conf.d/oracle-instantclient.conf"
   2. sudo ldconfig
6. Alternatively, set the LD\_LIBRARY\_PATH environment variable prior to running applications. For example:
   1. export LD\_LIBRARY\_PATH=/opt/oracle/instantclient\_19\_3:$LD\_LIBRARY\_PATH

# Installation of Oracle Client in Windows

Go to the link:

<https://www.oracle.com/database/technologies/oracle12c-windows-downloads.html>

Download:

Oracle Database 12c Release 2 Client (12.2.0.1.0) for Microsoft Windows (x64)