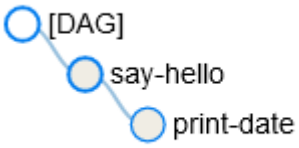
1. Ejemplo de 2 rutinas bash (por ejemplos script de bash o sh) que se llamen de forma secuencial (primero una y tan pronto termine la otra)

Solución

ej1.py

```python
from datetime import timedelta

from airflow import DAG

#from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago

import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}



#definir DAG
dag = DAG(
    'ej1',
    default_args=default_args,
    description = '2 tareas bash secuencial',
    schedule_interval=timedelta(days=1),
    start_date=days_ago(2),
    tags=['exercise1'],
)
#creacion tareas
t1 = BashOperator(task_id="say-hello", bash_command='echo "hello world"', dag=dag)
t2 = BashOperator(task_id="print-date", bash_command='date', dag=dag)


    #especificar dependencias (órden)
t1 >> t2
```

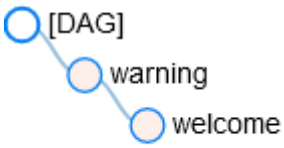


2. Ejemplo de la ejecución de 2 rutinas Python que se llamen de forma secuencial (primero una y tan pronto termine la otra)

Solución

ej2.py

```python
from datetime import timedelta

from airflow import DAG

from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago

import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retr_delay': timedelta(minutes=5),
}

def warning():
    logging.warning('error found')

def welcome():
    print("Bienvenido !")


#definir DAG
with DAG(
    'ej2',
    default_args=default_args,
    description = '2 tareas python secuencial',
    schedule_interval=timedelta(days=3),
    start_date=days_ago(1),
    tags=['example2'],
) as dag:
#creacion tareas
    t1 = PythonOperator(task_id="warning", python_callable=warning)
    t2 = PythonOperator(task_id="welcome", python_callable=welcome)

    #especificar dependencias (órden)
    t1 >> t2
```

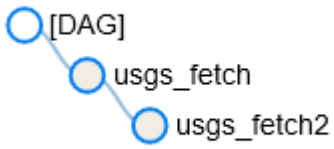


3. Ejemplo de la ejecución de 2 rutinas Java que se llamen de forma secuencial (primero una y tan pronto termine la otra)

Solución

ej3.py

```
#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=1),
}

#definir DAG
dag = DAG(
    'ej3',
    default_args=default_args,
    description = '2 tareas java secuencial',
    schedule_interval='1 * 3 * *',
    start_date=days_ago(2),
    tags=['exercise3'],
)
#creacion tareas
t1 = BashOperator(
  task_id = 'usgs_fetch'
  , dag = dag
  , bash_command = 'java -cp /mnt/c/Users/img/Documents/GitHub/backbone/target/kafkaUSGS-1.0-SNAPSHOT.jar KafkaUSGS'

)
t2 = BashOperator(
  task_id = 'usgs_fetch2'
  , dag = dag
  , bash_command = 'java -cp /mnt/c/Users/img/Documents/GitHub/backbone/target/kafkaUSGS-1.2-SNAPSHOT.jar KafkaUSGS'

)

    #especificar dependencias (órden)
t1 >> t2
```



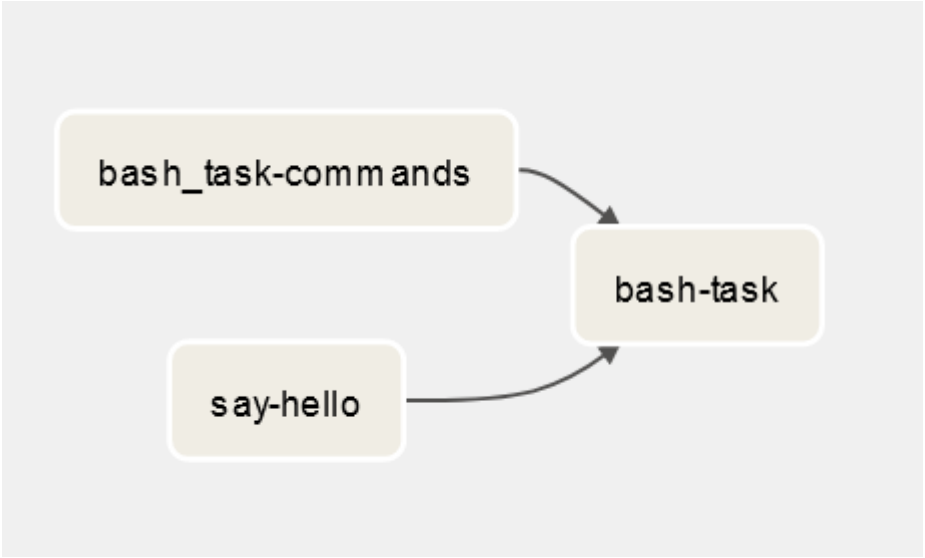
4. Ejemplo de la ejecución de 2 rutinas bash que se ejecuten en paralelo y cuando ambas hayan terminado se ejecute la rutina 3 (bash)

Solución

ej4.py

```python
from datetime import timedelta

from airflow import DAG

#from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago

import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}



#definir DAG
dag = DAG(
    'ej4',
    default_args=default_args,
    description = '2 tareas bash paralelo ',
    schedule_interval=timedelta(days=1),
    start_date=days_ago(2),
    tags=['exercise1'],
)
commands = """
    mkdir -p /usr/local/airflow/dags/{{ ds }};
    cp /usr/local/airflow/dags/command.sh /usr/local/airflow/dags/{{ ds }};
    sh /usr/local/airflow/dags/{{ ds }}/command.sh;
    """
#creacion tareas
t1 = BashOperator(task_id="say-hello", bash_command='echo "hello world"', dag=dag)
t2 = BashOperator(task_id='bash_task-commands', bash_command=commands)
t3 = BashOperator(task_id='bash-task', bash_command="/usr/local/airflow/dags/command.sh ", xcom_push=True)

    #especificar dependencias (órden)
[t1, t2] >> t3
```

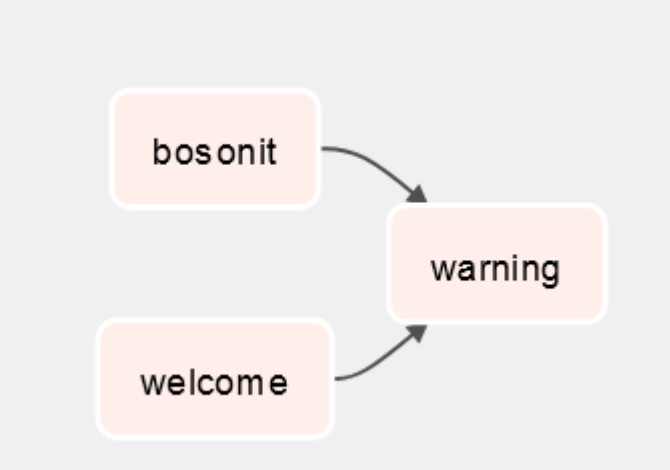
```
$ command.sh
#!/bin/bash
echo 'commands from a script'
ps -T
```



5. Ejemplo de la ejecución de 2 rutinas Python que se ejecuten en paralelo y cuando ambas hayan terminado se ejecute la rutina 3 (Python)

Solución

```python
from datetime import timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retr_delay': timedelta(minutes=5),
}

def warning():
    logging.warning('error found')

def welcome():
    print("Bienvenido !")

def bosonit(x):
    return x + " te ayuda a obtener conocimientos sobre Airflow"

#definir DAG
with DAG(
    'ej5',
    default_args=default_args,
    description = '2 tareas python secuencial',
    schedule_interval=timedelta(days=3),
    start_date=days_ago(1),
    tags=['example5'],
) as dag:
#creacion tareas
    t1 = PythonOperator(task_id="warning", python_callable=warning)
    t2 = PythonOperator(task_id="welcome", python_callable=welcome)
    t3 = PythonOperator(
    task_id='bosonit',
    python_callable= bosonit,
    op_kwargs = {"x" : "Bosonit"},
    dag=dag,
)
    #especificar dependencias (órden)
    [t2, t3] >> t1
```

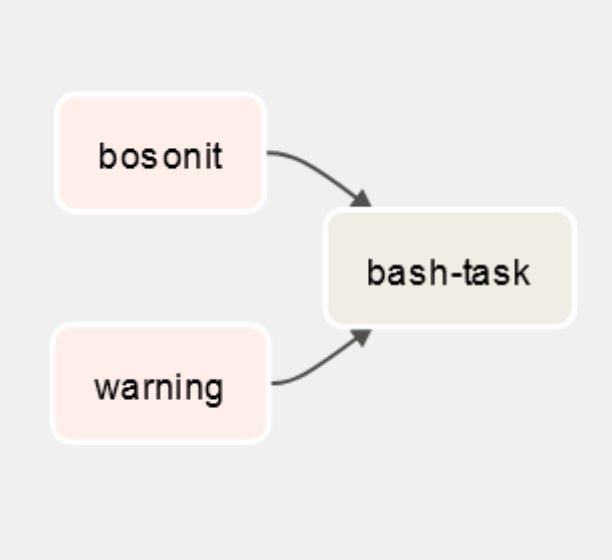


6. Ejemplo de la ejecución de 2 rutinas Python que se ejecuten en paralelo y cuando ambas hayan terminado se ejecute la rutina 3 (Java o Bash)

Solución

ej6.py

```python
from datetime import timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago
import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retr_delay': timedelta(minutes=5),
}

def warning():
    logging.warning('error found')

def bosonit(x):
    return x + " te ayuda a obtener conocimientos sobre Airflow"

#definir DAG
with DAG(
    'ej6',
    default_args=default_args,
    description = '2 tareas python paralelo + 1 bash',
    schedule_interval=timedelta(days=3),
    start_date=days_ago(1),
    tags=['example6'],
) as dag:
#creacion tareas
    t1 = PythonOperator(task_id="warning", python_callable=warning)
    t2 = PythonOperator(
    task_id='bosonit',
    python_callable= bosonit,
    op_kwargs = {"x" : "Bosonit"},
    dag=dag,
    )
    t3 = BashOperator(task_id='bash-task', bash_command="cat /usr/local/airflow/dags/command.sh")
    #especificar dependencias (órden)
    [t2, t1] >> t3
```



7. Verificar la existencia de un fichero en una ruta (carpeta) en HDFS si no existe se sube a hdfs (PUT) en caso contrario ejecutar script de bash (por ejemplo, echo "Fichero ya existe" > file.txt)

Solución

ej7.py

```python
from datetime import timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago
import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retr_delay': timedelta(minutes=5),
}



#definir DAG
with DAG(
    'ej7',
    default_args=default_args,
    description = ' comprobar fichero hdfs',
    schedule_interval=timedelta(days=3),
    start_date=days_ago(1),
    tags=['example7'],
) as dag:
#creacion tareas
    commands = """
    if [ params.t1 -neq 0 ]; then
        hdfs dfs -put /root/Hadoop/solutions.txt hdfs://sandbox.bosonit.com:8020/root/tmp/adrian_moreno
    fi
    """
    t1 = BashOperator(task_id='test', bash_command="hdfs dfs -test -e hdfs://sandbox.hortonworks.com:8020/root/tmp/adrian_moreno/solutions.txt")
    t2 = BashOperator(task_id='nonexist', bash_command=commands, params={'t1': t1})
    t3 = BashOperator(task_id='exist', bash_command="/usr/local/airflow/dags/exists.sh", params={'t1': t1})


    #especificar dependencias (órden)
    t1 >> [t2,t3]
```

exits.sh

```
$ exits.sh
#!/bin/bash
if [ params.t1 -eq 0 ]; then
echo "Fichero ya existe" > file.txt
fi
```

8. Ejemplo: Hacer un DAG que efectúe las siguientes tareas:
    a. Crear Hive Table
    b. Invocar Load data [local] Inpath ..
    c. Ejecutar consulta Hive y guardar el resultado en HDFS como un csv

Solución

ej8.py

```python
from datetime import timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.apache.hive.operators.hive import HiveOperator
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago
import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retr_delay': timedelta(minutes=5),
}

#definir DAG
with DAG(
    'ej8',
    default_args=default_args,
    description = ' DAG 8 ',
    schedule_interval=timedelta(days=1),
    start_date=days_ago(1),
    tags=['example8'],
) as dag:
#creacion tareas
    crea = HiveOperator(
    hql = "hql/hive_create.hql",
    task_id = "create_hql_task",
    hive_cli_conn_id = "hive_local",
    dag = dag
    )
    carga = HiveOperator(
    hql = "hql/hive_load.hql",
    task_id = "load_hql_task",
    hive_cli_conn_id = "hive_local",
    dag = dag
    )
    guardar = BashOperator(task_id="save",
    bash_command='hive -e "SELECT * FROM players WHERE salary > 20000000" > hdfs://sandbox.bosonit.com:8020/root/tmp/adrian_moreno/ejercicio8.csv',
    dag=dag
    )
    #especificar dependencias (órden)
    crea >> carga >> guardar
```

hive_create.hql

```sql
CREATE DATABASE adr_library;
use adr_library;

CREATE TABLE players(
id int,
player_name string ,
country string,
salary int
)
ROW FORMAT DELIMITED
fields terminated by ','
TBLPROPERTIES ("skip.header.line.count"="1");

CREATE TABLE teams(
id int,
team_name string
)
ROW FORMAT DELIMITED
fields terminated by ','
TBLPROPERTIES ("skip.header.line.count"="1");
```

hive_load.hql

```sql
load data local inpath '/home/Downloads/football-players.txt' into table `adr_library`.`players`;

load data local inpath '/home/Downloads/football-teams.txt' into table `adr_library`.`teams`;
```

9. Ejemplo: Hacer un DAG que efectúe las siguientes tareas:
    a. Rutina PIG para transformar un fichero CSV en HDFS
    b. Utilizar el fichero resultante en HDFS y crear una external Table en Hive
    c. Ejecutar consulta Hive y guardar el resultado en HDFS como un csv

Solución

ej9.py

```
#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}


#definir DAG
dag = DAG(
    'ej9',
    default_args=default_args,
    description = 'pig',
    schedule_interval=timedelta(days=2),
    start_date=days_ago(2),
    tags=['exercise9'],
)


pig = PigOperator(
    task_id="pig",
    pig="dfs -copyFromLocal ~/data/datacovid.csv /user/covid;",
    pig_opts="-x local",
    dag=dag,
)
commands = """
hive -e create external table if not exists covid19 (country string, date string, total_cases int, total_deaths int, total_vaccinations int)
    comment 'the importance of vaccination'
    row format delimited
    fields terminated by ','
    stored as textfile
    location '/user/covid';
"""

external = BashOperator(task_id="table",
bash_command= commands,
dag=dag
)

guardar = BashOperator(task_id="save",
bash_command='hive -e "SELECT * FROM covid19 GROUPBY country" > hdfs://sandbox.bosonit.com:8020/root/tmp/adrian_moreno/covid19.csv',
dag=dag
)
pig >> external >> guardar
```

10. Ejemplo: Hacer un DAG que efectúe las siguientes tareas:
    a. Invocar Sqoop Import hacia tabla Hive (crearla si no existe)

b. Ejecutar consulta Hive y guardar el resultado en HDFS como un csv

Solución

ej10.py

```python
from datetime import timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.apache.hive.operators.hive import HiveOperator
from airflow.operators.bash import BashOperator
from airflow.providers.apache.sqoop.operators.sqoop import SqoopOperator
from airflow.utils.dates import days_ago
import logging

#parametros por defecto
default_args ={
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retr_delay': timedelta(minutes=5),
}

#definir DAG
with DAG(
    'ej10',
    default_args=default_args,
    description = ' DAG 10 ',
    schedule_interval=timedelta(days=1),
    start_date=days_ago(1),
    tags=['example10'],
) as dag:
#creacion tareas
    crea = SqoopOperator(conn_id='sqoop',
                         table='food',
                         username='root',
                         password='123456',
                         driver='jdbc:mysql://mysql.example.com/recipes',
                         cmd_type='import')


    guarda = BashOperator(task_id="save",
    bash_command='hive -e "SELECT * FROM food WHERE category === "vegan" " > hdfs://sandbox.bosonit.com:8020/root/tmp/adrian_moreno/myfood.csv',
    dag=dag
    )
    #especificar dependencias (órden)
    crea >> guarda
```

11. Programar (schedule) cualquiera de los DAGs creados anteriormente para que se ejecuten periódicamente (por ejemplo, cada 5 min)

Solución

```
default_args ={
    'depends_on_past': False,
        'email': ['airflow@example.com'],
        '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,
        # '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'
}
```

12. Programar (schedule) cualquiera de los DAGs creados anteriormente para que se ejecuten periódicamente (por ejemplo, cada 5 min) y que haga reintentos 3 reintentos cada 10 min en caso de fallar

Solución

```
default_args ={
    'depends_on_past': False,
        'email': ['airflow@example.com'],
        'email_on_failure': False,
        'email_on_retry': False,
        'retries': 3,
        'retry_delay': timedelta(minutes=10),
        # 'queue': 'bash_queue',
        # 'pool': 'backfill',
        # 'priority_weight': 10,
        # 'end_date': datetime(2016, 1, 1),
        # 'wait_for_downstream': False,
        # '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'
}
```

13. Hacer DAG con el ejemplo del Padrón de Madrid con las siguientes tareas:

    a. Crear tabla Hive cuyo formato de almacenamiento sea texto delimitado

    b. Hacer un CTAS (Create Tablas as Select) que cree la tabla con los mismos campos, pero esta vez almacenando los datos en formato parquet

    c. Invocar rutina en Spark que lea la tabla cuyos datos están en Parquet y genere un único Dataframe/Dataset que se almacene como una tabla en Hive y que calcule por distrito:

        i. Total de españoles hombres

        ii. Total de españoles mujeres

        iii. Total de extranjeros hombres

        iv. Total de extranjeros mujeres

        v. Total de Hombres

        vi. Total de Mujeres

vii. Total de Personas

Solución

```
#definir DAG
with DAG(
    'padron',
    default_args=default_args,
    description = ' padronMadrid ',
    schedule_interval=timedelta(days=1),
    start_date=days_ago(1),
    tags=['Madrid'],
) as dag:
#creacion tareas
    crea = HiveOperator(
    hql = "hql/hive_createMadrid.hql",
    task_id = "create_hql_task",
    hive_cli_conn_id = "hive_local",
    dag = dag
    )
    ctas = HiveOperator(
    hql = "hql/ctas.hql",
    task_id = "ctas",
    hive_cli_conn_id = "hive_local",
    dag = dag
    )

    commands = """
    SELECT SUM(EspanolesHombres) AS Hespanoles, SUM(EspanolesMujeres) AS Mespanoles,
        SUM(ExtranjerosHombres) AS HExtranjeros, SUM(ExtranjerosMujeres) AS MExtranjeros,
        SUM(Hespanoles+HExtranjeros) AS Hombres, SUM(MExtranjeros+Mespanoles) AS Mujeres,
        SUM(Hombres+Mujeres) AS Personas

    FROM padron2
    GROUP BY DESC_DISTRITO
    """
    sql_job = SparkSqlOperator(sql=commands, master="local", task_id="sql_job")

    #especificar dependencias (órden)
    crea >> ctas >> sql_job
```