Projekt 1 – orkiestracja – Apache Airflow

Apache Airflow to platforma open source do opracowywania, harmonogramowania i monitorowania wsadowych przepływów pracy. Funkcjonalność, którą ona oferuje, często określa się mianem orkiestracja.

Niektórzy uważają, że jest to jedno z najlepszych narzędzi w swojej klasie.

Apache Airflow jest oparte na Pythonie. Prawie w każdym elemencie systemu wykorzystywany jest właśnie ten język programowania. Począwszy od sposobu definiowania przepływów pracy przez użytkownika, a skończywszy na implementacji wtyczek. Dzięki wykorzystywaniu tak uniwersalnego języka, Apache Airflow pozwala na tworzenie przepływów, które mogą wykorzystywać praktycznie dowolną technologię.

Istotnym elementem systemu jest jego interfejs sieciowy. Pozwala on nie tylko podglądać zdefiniowane przepływy, oraz je uruchamiać, ale także w pełnym zakresie zarządzać stanem przepływów.

Airflow można wdrożyć na wiele sposobów, od pojedynczego procesu na laptopie po konfigurację rozproszoną, która obsługuje potężne przepływy. My wykorzystamy bardzo prostą konfigurację uruchomioną na maszynie master klastra *Dataproc*.

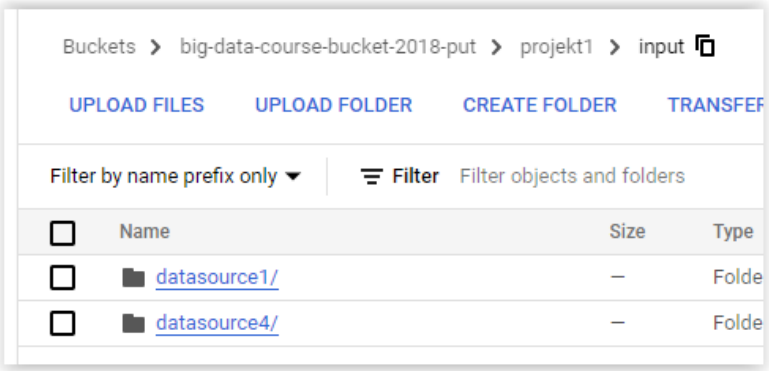
Naszym celem ostatecznym będzie przygotowanie przepływu, który będzie zarządzał uruchamianiem (orkiestrował) elementów składowych projektu 1.

Elementy wyjściowe

W założeniu wszystkie elementy składowe dla projektu 1 są już gotowe. W przypadku niniejszego warsztatu będzie to wariant: *Hadoop Streaming, Pig, Zestaw danych nr 3*

Powyższe oznacza, że są już gotowe:

- Program MapReduce:
  - skrypt mappera: `mapper.py`
  - skrypt reduktora: `reducer.py`
  - skrypt combinera: `combiner.py`
- Skrypt Piga
  - skrypt `transform5.pig`
- Dane wejściowe
  - znajdują się one w zasobniku w katalogu `gs://`*`nazwa_zasobnika`*`/projekt1/input` (w podkatalogach `datasource1` i `datasource4`)



Ponadto jest gotowa pobrana źródłowa definicja przepływu, plik `projekt1.py`

Oczywiście każdy projekt może dotyczyć innego wariantu, a zatem elementy składowe projektu mogą być inne. Należy to uwzględnić realizując ten warsztat. W razie potrzeby szukaj wsparcia u prowadzącego.

Wszystkie elementy składowe powinny być przetestowane i działające. Polecenia uruchamiające powinny być opracowane i znane. Skrypt Pig/Hive powinien być parametryzowany:

- `input_dir3` – katalogiem wejściowym dla przetworzonego pierwszego zbioru danych (wynik przetwarzania MapReduce)
- `input_dir4` – katalogiem wejściowym dla drugiego zbioru danych
- `output_dir6` – katalogiem wyjściowym, który będzie zawierał ostateczny wynik całości przetwarzania

W razie potrzeby wprowadź odpowiednie poprawki oraz przetestuj jeszcze raz wszystkie składowe projektu 1

Uruchomienie klastra Dataproc

Apache Airflow, nie należy do komponentów dostępnych w ramach klastra *Dataproc*, o czym możemy się przekonać: `https://cloud.google.com/dataproc/docs/concepts/versioning/dataproc-release-2.0`

Oznacza to, że będziemy go musieli sobie zainstalować niezależnie na uruchomionym uprzednio klastrze.

Tak na marginesie, w ramach usług dostarczanych przez platformę GCP istnieje narzędzie *Cloud Composer*.
Jest to usługa pozwalająca na orkiestrację przetwarzania (danych). Opiera się ona całkowicie na *Apache Airflow*.
Nie będziemy jednak z niej korzystali w ramach tego warsztatu.

1. Utwórz klaster *Dataproc* za pomocą poniższego polecenia

```
gcloud beta dataproc clusters create ${CLUSTER_NAME} \
--enable-component-gateway --bucket ${BUCKET_NAME} \
--region ${REGION} --subnet default --zone ${ZONE} \
--master-machine-type n1-standard-4 --master-boot-disk-size 50 \
--num-workers 2 \
--worker-machine-type n1-standard-2 --worker-boot-disk-size 50 \
--image-version 2.0-debian10 \
--optional-components DOCKER \
--project ${PROJECT_ID} --max-age=3h
```

Instalacja i uruchomienie Apache Airflow

Zgodnie z dokumentacją, *Apache Airflow* wymaga zainstalowanego Pythona w wersji 3.X. Sama instalacja natomiast powinna się odbywać za pomocą instalatora pakietów Pythona `pip`. Postąpimy zatem zgodnie z tymi regułami.

2. Uruchom terminal SSH do maszyny master naszego klastra. Sprawdź za pomocą poniższego polecenia wersję zainstalowanego Pythona

```
python --version
```

```
jankiewicz_krzysztof@cluster-0999-m:~$ python --version
Python 3.8.13
```

3. Zanim zainstalujemy samego *Apache Airflow*, ustaw zmienną `AIRFLOW_HOME`

```
export AIRFLOW_HOME=~/airflow
```

4. Nie będziemy instalowali żadnego z setek pakietów wykorzystywanych w ramach tego narzędzia. W zupełności wystarczą nam podstawowe tzw. operatory, z których buduje się definicje przepływów. Zainstaluj *Apache Airflow*

```
pip install apache-airflow
```

5. Dodaj do zmiennej środowiskowej ścieżkę do narzędzi Apache Airflow

```
export PATH=$PATH:~/.local/bin
```

6. Zainicjuj bazę danych, z której będzie korzystał *Apache Airflow*

```
airflow db init
```

7. Czas na finał – uruchom naszego *Apache Airflow*. Pozostaw ten terminal otwarty. Zwróć uwagę na dane uwierzytelniające dla administratora naszego środowiska oraz port, pod którym dostępny jest *Apache Airflow*.

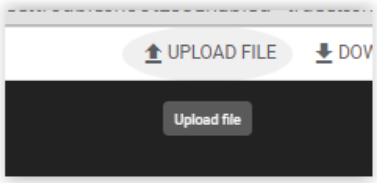
```
airflow standalone
```

```
 triggerer | ____________       _____________
 triggerer | ____    |__( )_________  __/__  /________      __
 triggerer | ____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
 triggerer | ___  ___ |  / _  /    _  __/ _  / / /_/ /_ |/ |/ /
 triggerer | _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
 triggerer | [2022-09-20 10:01:47,268] {triggerer_job.py:101} INFO - Starting the triggerer
 webserver | [2022-09-20 10:01:47 +0000] [8970] [INFO] Starting gunicorn 20.1.0
 webserver | [2022-09-20 10:01:48 +0000] [8970] [INFO] Listening at: http://0.0.0.0:8080 (8970)
 webserver | [2022-09-20 10:01:48 +0000] [8970] [INFO] Using worker: sync
 webserver | [2022-09-20 10:01:48 +0000] [8999] [INFO] Booting worker with pid: 8999
 webserver | [2022-09-20 10:01:48 +0000] [9000] [INFO] Booting worker with pid: 9000
 webserver | [2022-09-20 10:01:48 +0000] [9001] [INFO] Booting worker with pid: 9001
 webserver | [2022-09-20 10:01:48 +0000] [9002] [INFO] Booting worker with pid: 9002
standalone |
standalone | Airflow is ready
standalone | Login with username: admin  password: F5G5xSdZyuNnenWp
standalone | Airflow Standalone is for development purposes only. Do not use this in production!
```

Instalacja wzorcowej wersji przepływu

8. Otwórz nowy terminal SSH do maszyny master naszego klastra.

9. Załaduj do domowego katalogu wszystkie pliki Twojego projektu, a także plik z definicją przepływu



10. Sprawdź czy wszystkie pliki znalazły się na serwerze master.

```
jankiewicz_krzysztof@cluster-0999-m:~$ ls
airflow  combiner.py  mapper.py  projekt1.py  reducer.py  transform5.pig
```

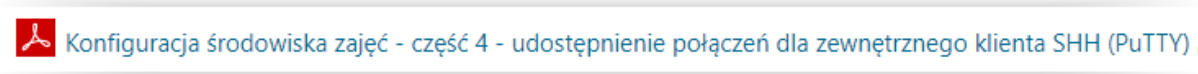
11. Utwórz w katalogu `~/airflow` podkatalog `dags`, z którego będą załadowywane definicje przepływów. Przy okazji, utwórz w nim podkatalog `project_files`, który będzie zawierał wszystkie składowe naszego projektu

```
mkdir -p ~/airflow/dags/project_files
```

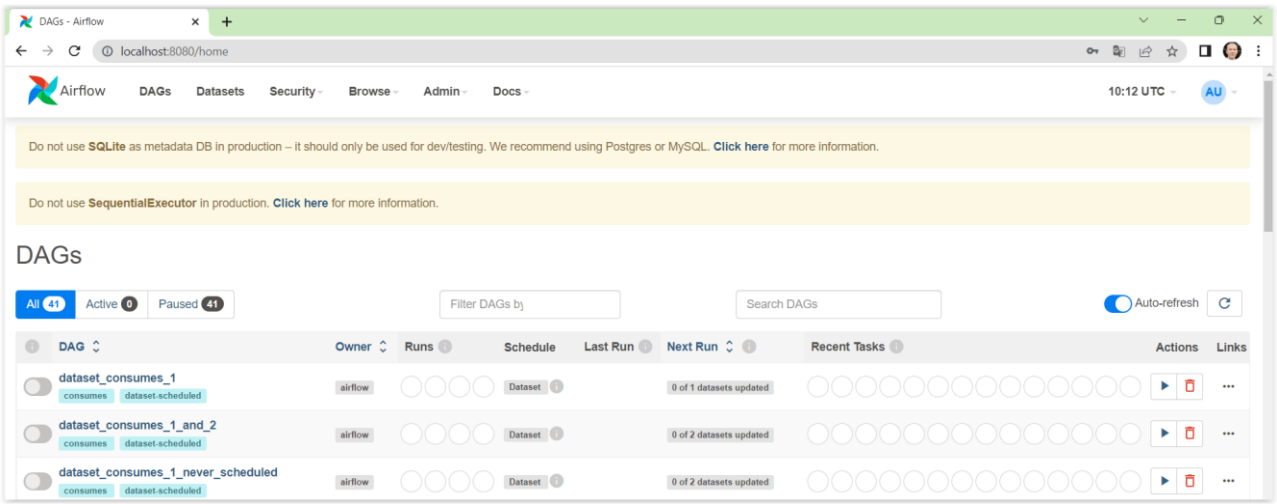
12. Przenieś plik z definicją przepływu do katalogu dags. Pozostałe pliki przenieś do katalogu `project_files`

```
mv projekt1.py ~/airflow/dags/
mv *.* ~/airflow/dags/project_files
```

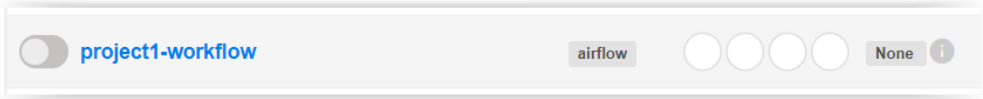
13. Utwórz tunel do portu 8080 naszego serwera master. Sposób definicji tunelu został opisany na stronie kursu



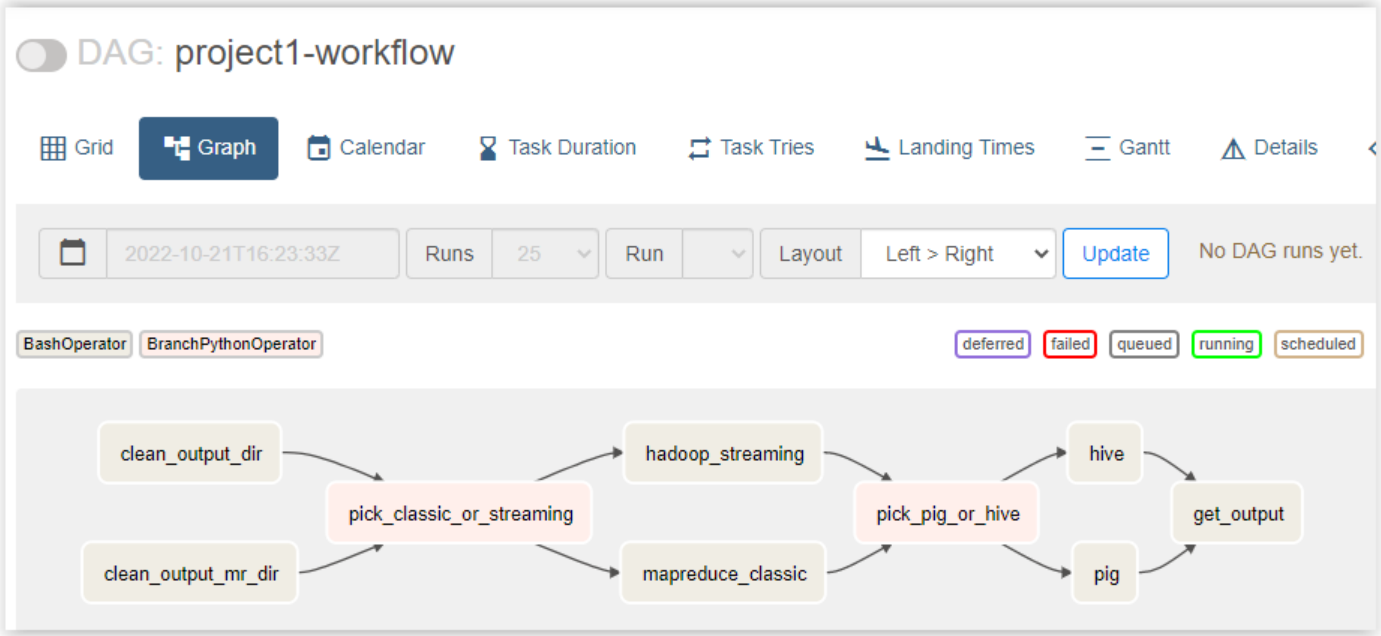
14. Otwórz przeglądarkę pod adresem `http://localhost:8080`. Zaloguj się zgodnie z poznanymi danymi uwierzytelniającymi.



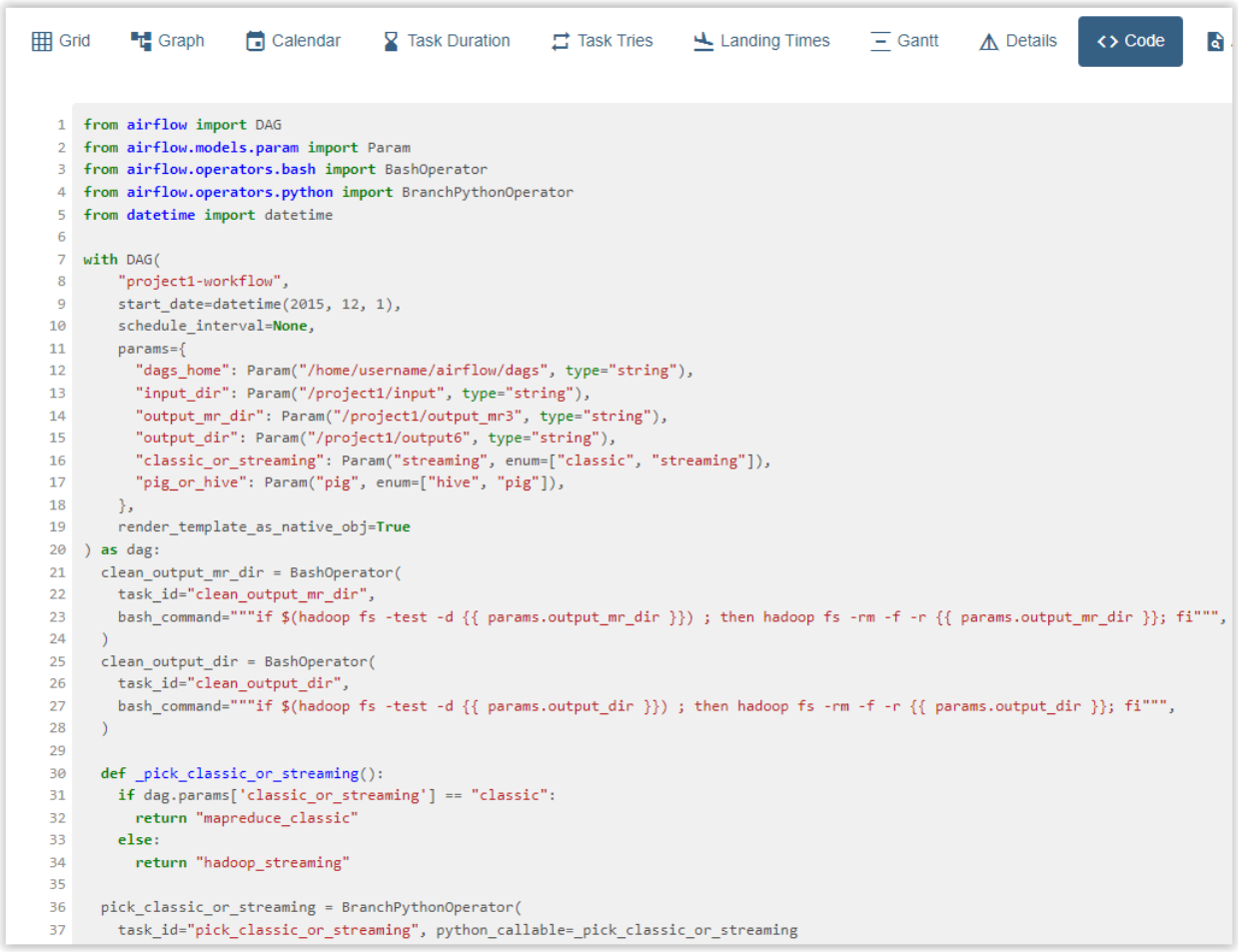
15. Znajdź wśród wielu przykładowych przepływów przepływ przeznaczony dla naszego projektu



16. Wybierz go, a następnie przyglądnij się jego graficznej reprezentacji



17. Zaglądnij także do kodu tego przepływu i zapoznaj się z jego budową.




```
from airflow import DAG
from airflow.models.param import Param
from airflow.operators.bash import BashOperator
from airflow.operators.python import BranchPythonOperator
from datetime import datetime

with DAG(
    "project1-workflow",
    start_date=datetime(2015, 12, 1),
    schedule_interval=None,
    params={
      "dags_home": Param("/home/username/airflow/dags", type="string"),
      "input_dir": Param("/project1/input", type="string"),
      "output_mr_dir": Param("/project1/output_mr3", type="string"),
      "output_dir": Param("/project1/output6", type="string"),
      "classic_or_streaming": Param("streaming", enum=["classic", "streaming"]),
      "pig_or_hive": Param("pig", enum=["hive", "pig"]),
    },
    render_template_as_native_obj=True
) as dag:
  clean_output_mr_dir = BashOperator(
    task_id="clean_output_mr_dir",
    bash_command="""if $(hadoop fs -test -d {{ params.output_mr_dir }}) ; then hadoop fs -rm -f -r {{ params.output_mr_dir }}; fi""",
  )
  clean_output_dir = BashOperator(
    task_id="clean_output_dir",
    bash_command="""if $(hadoop fs -test -d {{ params.output_dir }}) ; then hadoop fs -rm -f -r {{ params.output_dir }}; fi""",
  )

  def _pick_classic_or_streaming():
    if dag.params['classic_or_streaming'] == "classic":
      return "mapreduce_classic"
    else:
      return "hadoop_streaming"

  pick_classic_or_streaming = BranchPythonOperator(
    task_id="pick_classic_or_streaming", python_callable=_pick_classic_or_streaming
```

18. Zwróć uwagę na:
    a. to, że jest on oparty o dwa typy operatorów
        i. `BashOperator` – który uruchamia polecenia systemu operacyjnego, oraz
        ii. `BranchPythonOperator`, stosowany do warunkowego uruchamiania kolejnych operatorów
    b. parametry oraz ich wartości domyślne,
    c. polecenia zawarte w operatorach `BashOperator` oraz wykorzystanie w nich wartości parametrów.

19. Aby nasz przepływ mógł zadziałać konieczne są poprawki. W terminalu SSH wywołaj edytor tekstowy

```
nano ~/airflow/dags/projekt1.py
```

20. A następnie:
    a. zmień domyślne parametry `classic_or_streaming` oraz `pig_or_hive` na zgodne z Twoim wariantem projektu. Dla przykładu:

```
    "output_dir": Param("/project1/output6", type="string"),
    "classic_or_streaming": Param("streaming", enum=["classic", "streaming"]),
    "pig_or_hive": Param("pig", enum=["hive", "pig"]),
},
```

    b. nie zmieniaj domyślnych katalogów wejściowych oraz wyjściowych – mogłyby one zdradzić Twoje personalia
    c. popraw polecenia wywołujące elementy składowe Twojego projektu. Wykorzystaj te polecenia, które były przez Ciebie wykorzystane podczas testów. Uzupełnij je o odwołania do odpowiednich parametrów. W przypadku wariantu wykorzystywanego w tym warsztacie konieczna będzie modyfikacja:
        i. polecenia w operatorze `hadoop_streaming`. Przykładowe niekompletne polecenie wywołujące przetwarzanie *Hadoop Streaming*:

```
  hadoop_streaming = BashOperator(
    task_id="hadoop_streaming",
    bash_command="""mapred streaming \
-files {{ params.dags_home }}/project_files/mapper.py,\
{{ params.dags_home }}/project_files/combiner.py,\
{{ params.dags_home }}/project_files/reducer.py \
-input {{ params.input_dir }}/datasource1 \
-mapper  mapper.py \
-combiner combiner.py \
-reducer reducer.py \
-output {{ params.output_mr_dir }} \
```

        ii. polecenia w operatorze `pig`. Przykładowe (również niekompletnie) polecenie uruchamiające skrypt obsługiwany za pomocą *Piga*:

```
  pig = BashOperator(
    task_id="pig",
    bash_command="""pig -f {{ params.dags_home }}/project_files/transform5.pig \
-param input_dir4={{ params.input_dir }}/datasource4 \
-param input_dir3={{ params.output_mr_dir }} \
-param output_dir6={{ params.output_dir }}""",
  )
```

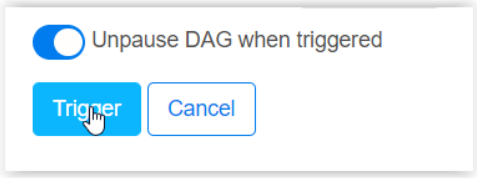
    Pamiętaj, aby dokonać modyfikacji zgodnych z Twoim wariantem projektu i Twoimi składowymi. Powyższe fragmenty to tylko przykład dla wariantu wykorzystywanego w tym warsztacie.

21. Po zakończeniu wszystkich poprawek zapisz zmiany w pliku definicji przepływu.

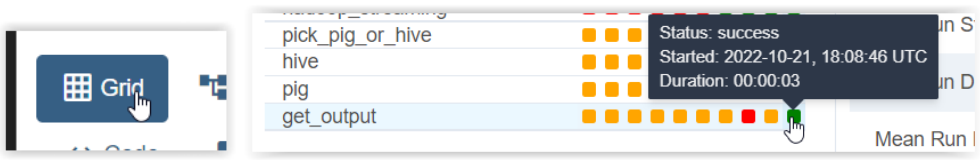
22. Powróć do interfejsu sieciowego *Apache Airflow*. Odśwież kod przepływu, sprawdź czy wszystkie wprowadzone zmiany są poprawne. Jeśli nie, dokonaj odpowiednich poprawek.

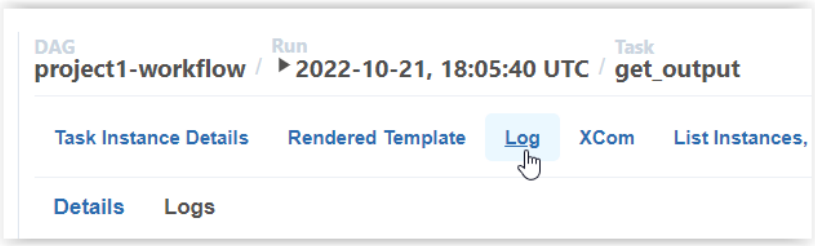
23. Jeśli Twój przepływ jest zgodny z oczekiwaniami. Czas na jego uruchomienie. Nasze domyślne ścieżki nie odpowiadają rzeczywistości, dlatego uruchomimy nasz przepływ w taki sposób, który pozwali nam skorygować domyślną konfigurację. Popraw odpowiednie ścieżki.

24. Uruchom przepływ za pomocą przycisku *Trigger*.



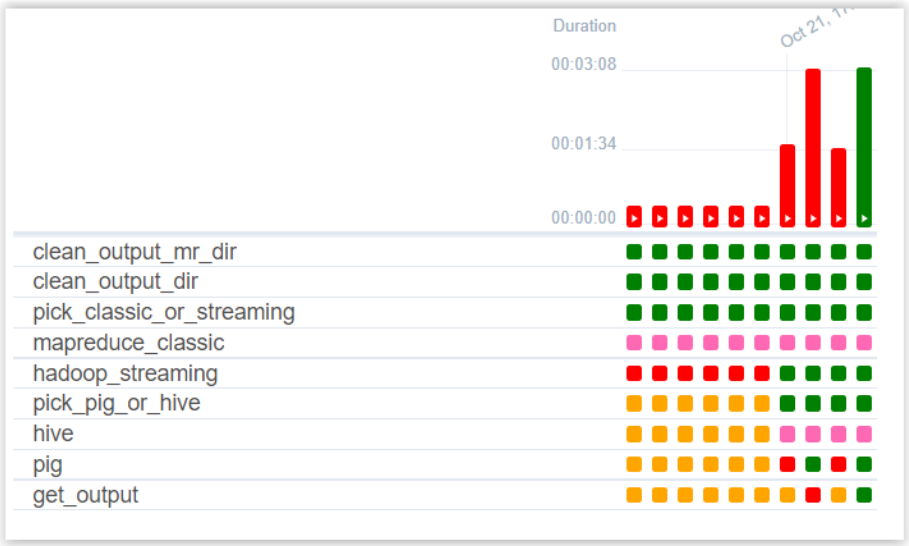
25. Poczekaj aż Twój przepływ zakończy swoje przetwarzanie. Podglądaj przebieg przepływu na zakładce *Grid*. Jeśli Twój przepływ za pierwszym zakończy się sukcesem… no cóż, przyjmij gratulacje. To zazwyczaj się nie zdarza. Zaglądnij do logów ostatniego z zadań (`get_ouput`) – znajdziesz w nim zaprezentowany końcowy wynik Twojego przetwarzania. Jeśli wszystko jest analogiczne jak podczas Twoich testów, mamy gotową całość. Wystarczy ją przygotować i zarejestrować jako rozwiązanie projektu.



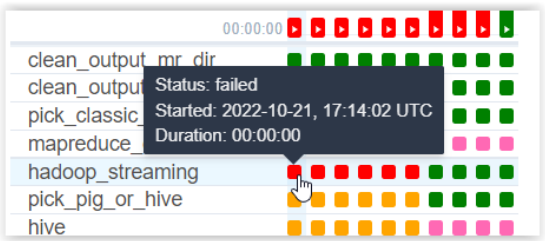


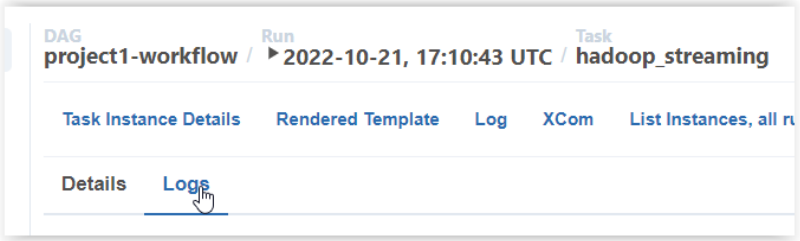

```
[2022-10-21, 18:08:47 UTC] {subprocess.py:75} INFO - Running command: ['/usr/bin/bash', '-c', 'h
[2022-10-21, 18:08:47 UTC] {subprocess.py:86} INFO - Output:
[2022-10-21, 18:08:49 UTC] {subprocess.py:93} INFO - street     person_type     killed  injured
[2022-10-21, 18:08:49 UTC] {subprocess.py:93} INFO - {"fields":[{"name":"street","type":55,"desc
```

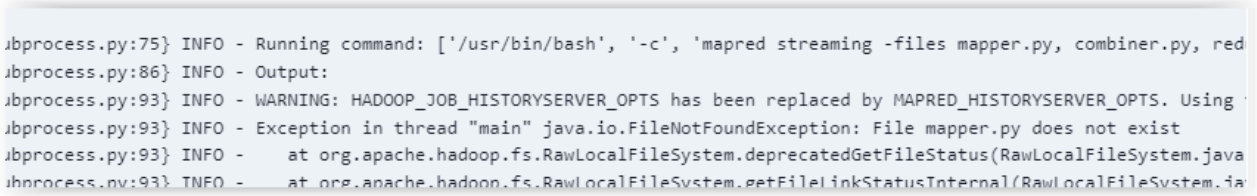
26. Jeśli jednak coś poszło nie tak (nie każdy jest nieomylny), przykładowo wielokrotnie poniżej…



…wówczas *Apache Airflow* jest nieoceniony. Każde z niepoprawnie wykonanych zadań można zaznaczyć, a następnie zaglądnąć do wygenerowanych logów zapisanych przez *Apache Airflow*.
Z nich można wyczytać wszystko.







27. Napraw kolejno wszystkie błędy, doprowadzając swój przepływ do etapu, w którym będzie działał on bez zarzutu i powtarzalnie.

Przygotowanie projektu do rejestracji

28. Jeśli Twój projekt jest gotowy, należy przygotować go do rejestracji. Przejdź do katalogu `~/airflow`, a następnie spakuj katalog `dags` wraz z jego zawartością do pliku `projekt1.zip`

```
cd ~/airflow
rm -r dags/__pycache__
zip -r ~/projekt1.zip dags
```

29. Skopiuj tak utworzony plik do swojego zasobnika. Dzięki temu przeżyje on usunięcie klastra.

```
hadoop fs -copyFromLocal ~/projekt1.zip gs://nazwa_zasobnika/projekt1
```

30. Pobierz plik z zasobnika i zarejestruj go jako rozwiązanie swojego projektu. Osoba, która będzie go oceniała wystarczy, że rozpakuje Twój plik w katalogu `~/airflow` i już będzie mogła go przetestować.