Ministry of Higher Education and Scientific Research
Higher School of Computer Science, May 8, 1945, Sidi Bel Abbes
Academic year: 2024-2025 **Semester 2**
2 SC: IASD **Module: Big Data Technologies**

Lab 8: Apache Airflow

In this lab, we will help a rocket enthusiast named John, who closely follows every rocket launch.

News about rocket launches is available from many information sources that John tracks, and ideally, he would like to have all his rocket-related news aggregated in one place.

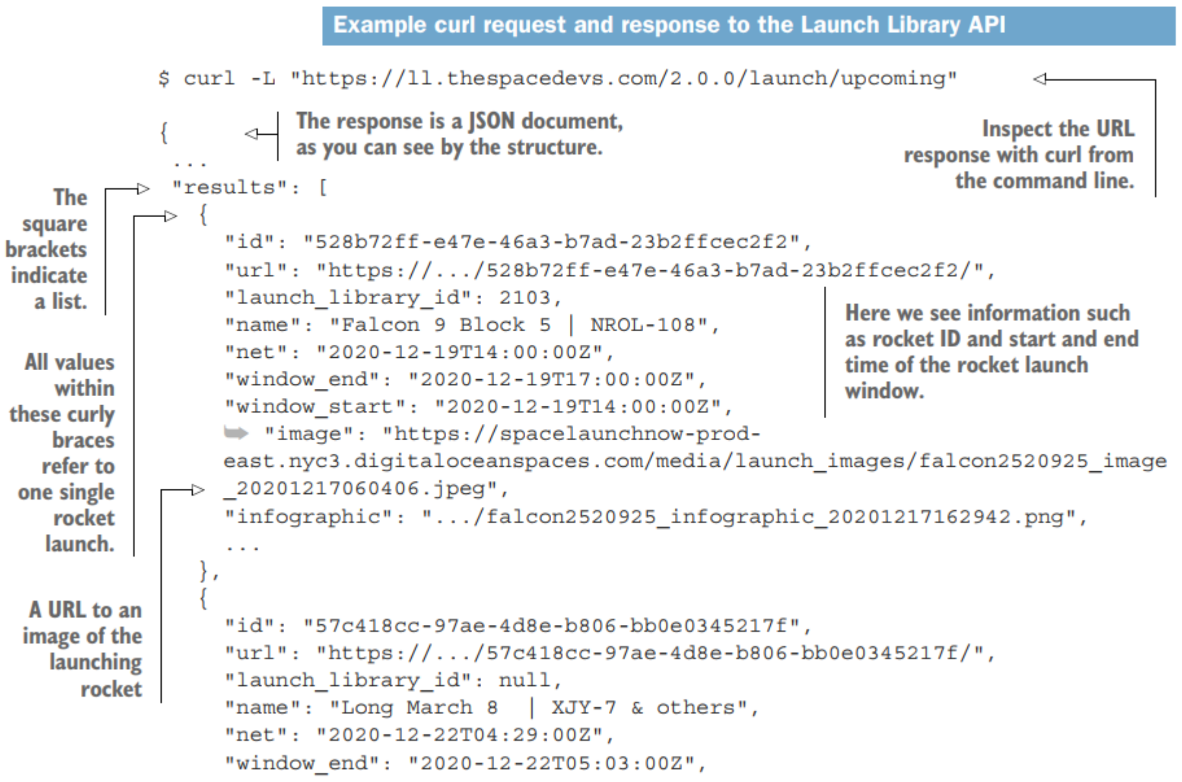
John would like an automated way to collect information about all rocket launches, as well as a personal overview of the latest rocket news.

To begin, John has decided to first collect images of rockets.

Exploring the Data

For the data, we are using **Launch Library 2** (https://thespacedevs.com/llapi), an online repository of past and upcoming rocket launch data from various sources.

John is currently interested only in upcoming rocket launches. Fortunately, the Launch Library provides exactly the data he is looking for (https://ll.thespacedevs.com/2.0.0/launch/upcoming). It offers information on upcoming rocket launches, including URLs where the respective rocket images can be found. Here is a snippet of the data returned by this URL:



Example curl request and response to the Launch Library API

```
$ curl -L "https://ll.thespacedevs.com/2.0.0/launch/upcoming"

{
  ...
 "results": [
   {
     "id": "528b72ff-e47e-46a3-b7ad-23b2ffcec2f2",
     "url": "https://.../528b72ff-e47e-46a3-b7ad-23b2ffcec2f2/",
     "launch_library_id": 2103,
     "name": "Falcon 9 Block 5 | NROL-108",
     "net": "2020-12-19T14:00:00Z",
     "window_end": "2020-12-19T17:00:00Z",
     "window_start": "2020-12-19T14:00:00Z",
        "image": "https://spacelaunchnow-prod-
     east.nyc3.digitaloceanspaces.com/media/launch_images/falcon2520925_image
     _20201217060406.jpeg",
     "infographic": ".../falcon2520925_infographic_20201217162942.png",
     ...
   },
   {
     "id": "57c418cc-97ae-4d8e-b806-bb0e0345217f",
     "url": "https://.../57c418cc-97ae-4d8e-b806-bb0e0345217f/",
     "launch_library_id": null,
     "name": "Long March 8  | XJY-7 & others",
     "net": "2020-12-22T04:29:00Z",
     "window_end": "2020-12-22T05:03:00Z",
```

Inspect the URL response with curl from the command line.

The response is a JSON document, as you can see by the structure.

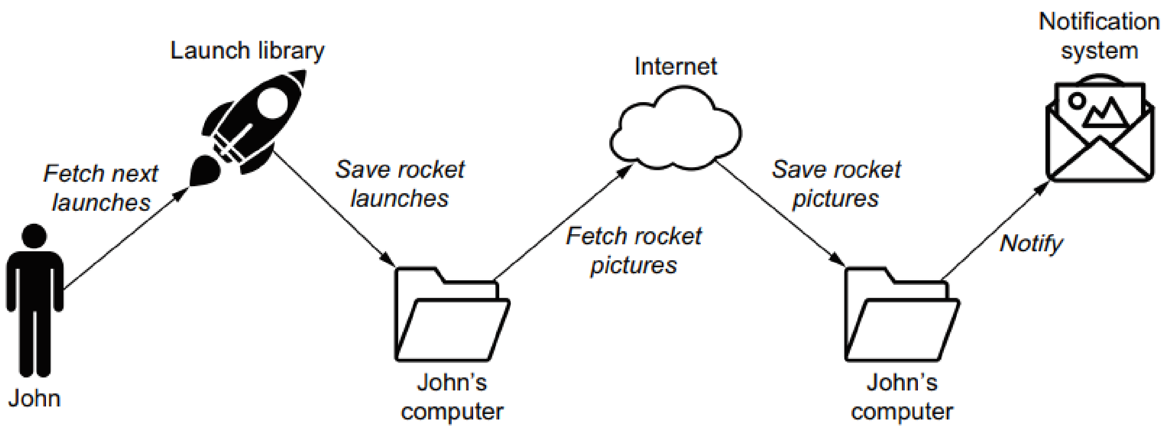
The square brackets indicate a list.

All values within these curly braces refer to one single rocket launch.

Here we see information such as rocket ID and start and end time of the rocket launch window.

A URL to an image of the launching rocket

As you can see, the data is in JSON format, providing information about rocket launches. For each launch, there are details about each rocket, such as its ID, name, and the URL of its image. This is exactly what John needs, and he initially develops the plan illustrated in the following figure to collect images of upcoming rocket launches. At the end of the day, John's goal is to have a directory filled with rocket images.



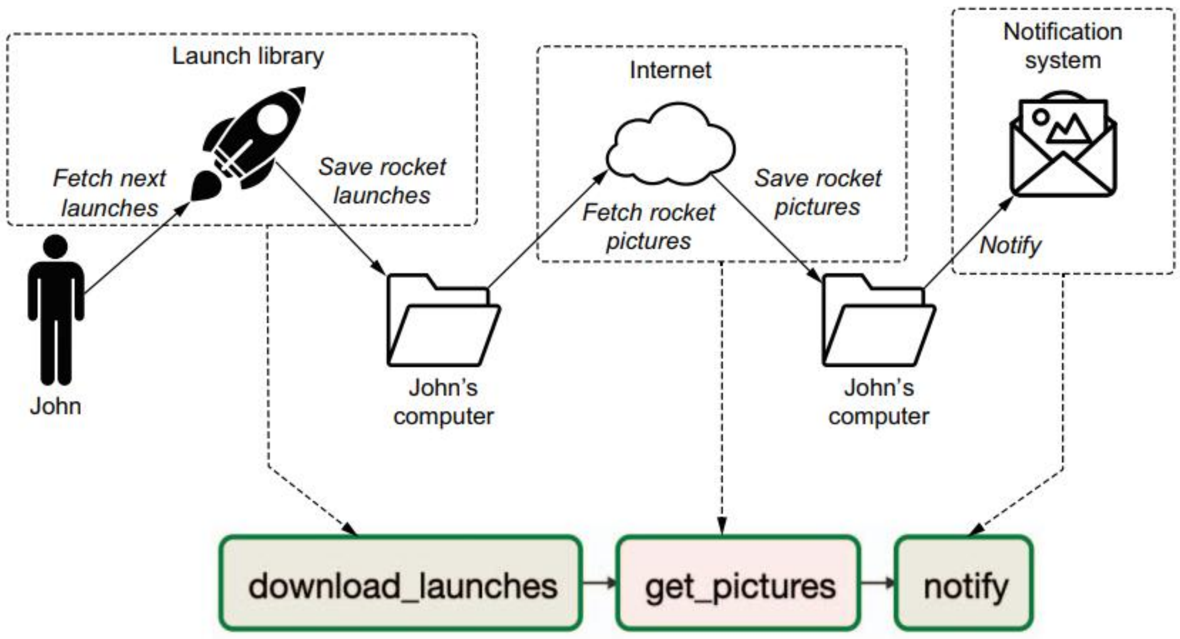
The Airflow DAG

John's use case is well-defined, so let's see how to program his plan.

One question arises: why would we need a system like Airflow for this job?

The advantage of Airflow is that we can divide a large job, consisting of one or more steps, into individual tasks that collectively form a DAG. Multiple tasks can run in parallel, and they can use different technologies. For example, we could first execute a Bash script and then run a Python script.

We have broken down John's workflow into three tasks in Airflow, as shown in the following figure:



To start **docker-compose**:

```
cmd> docker-compose up
```

1) Write the Python code (File: **tp8.py**) corresponding to John's workflow, with the following details:

- The schedule_interval is daily.
- Task 1: download_launches: Stores the result of the command

  curl -Lk "https://ll.thespacedevs.com/2.0.0/launch/upcoming"

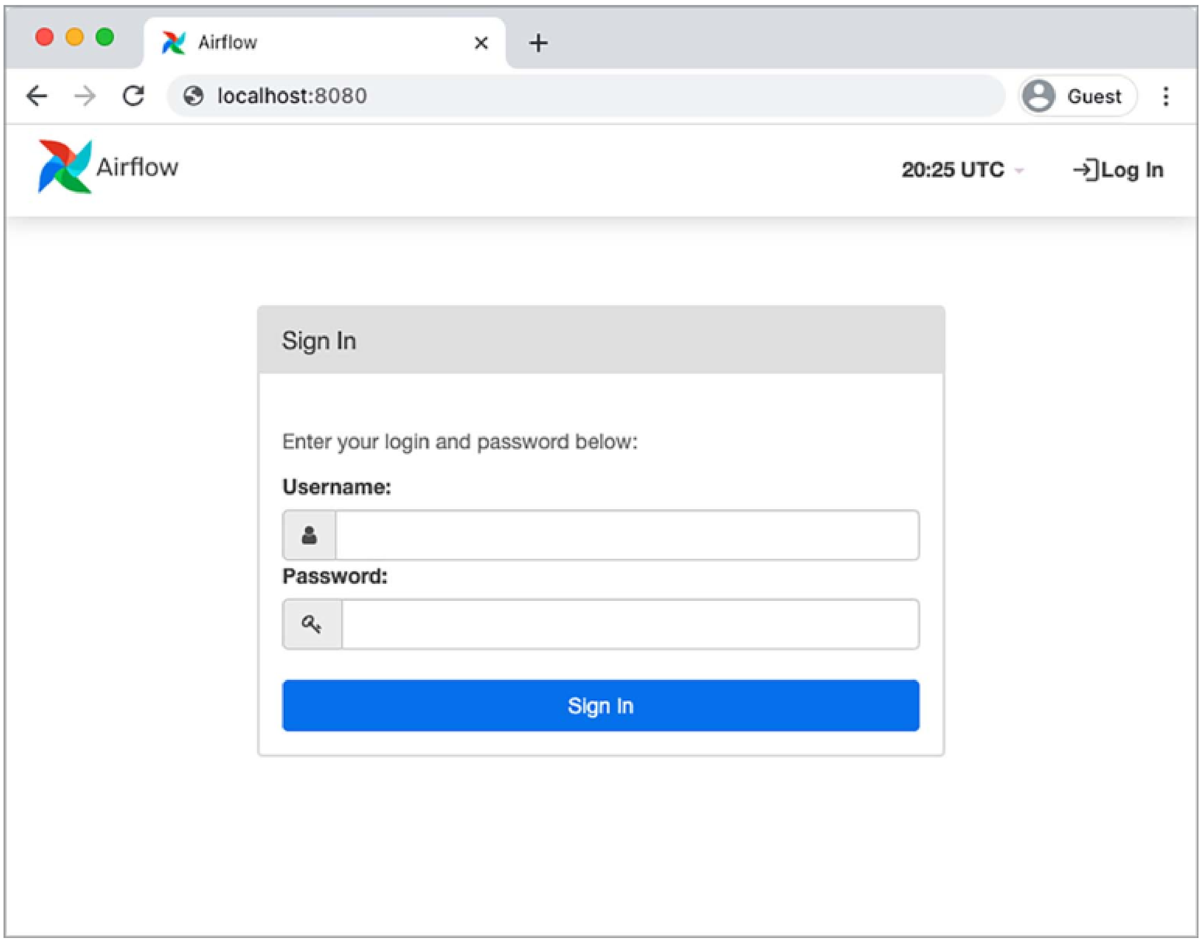
  into the file "/opt/airflow/dags/tmp/launches.json".

- Task 2: get_pictures: (1) Creates a directory "/opt/airflow/dags/tmp/images", (2) Reads the content of the file "/opt/airflow/dags/tmp/launches.json" to retrieve the image URLs, (3) Sends an HTTP GET request to each image URL, and (4) Stores the images returned by the HTTP requests in the "/opt/airflow/dags/tmp/images" directory.
- Task 3: notify: A BashOperator that displays the number of images in the "/opt/airflow/dags/tmp/images" directory.

2) Place the **tp8.py** file in the directory: **/opt/airflow/dags**

Airflow User Interface

The first interface of Airflow at http://localhost:8080 is the login screen:

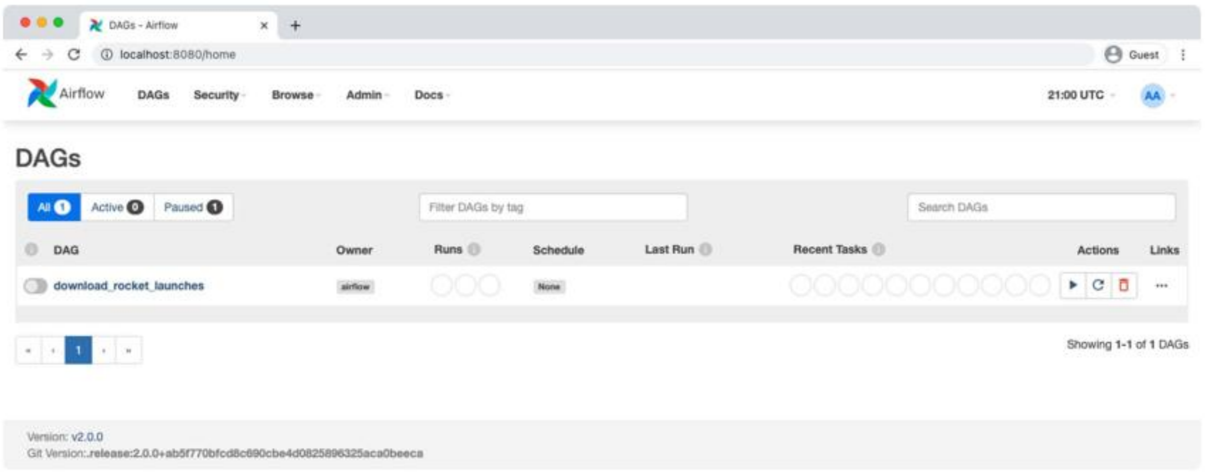


Enter your login and password below:

Username:

Password:

After authentication (**Username:** airflow and **Password:** airflow), you will see the list of DAGs, as shown in the following figure:



DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
download_rocket_launches	airflow		None				···

Showing 1-1 of 1 DAGs

Version: v2.0.0
Git Version:.release:2.0.0+ab5f770bfcd8c690cbe4d0825896325aca0beeca

Figure 1 Airflow home screen

Click on the name of your DAG to open it, then click on the icon named **Graph**:

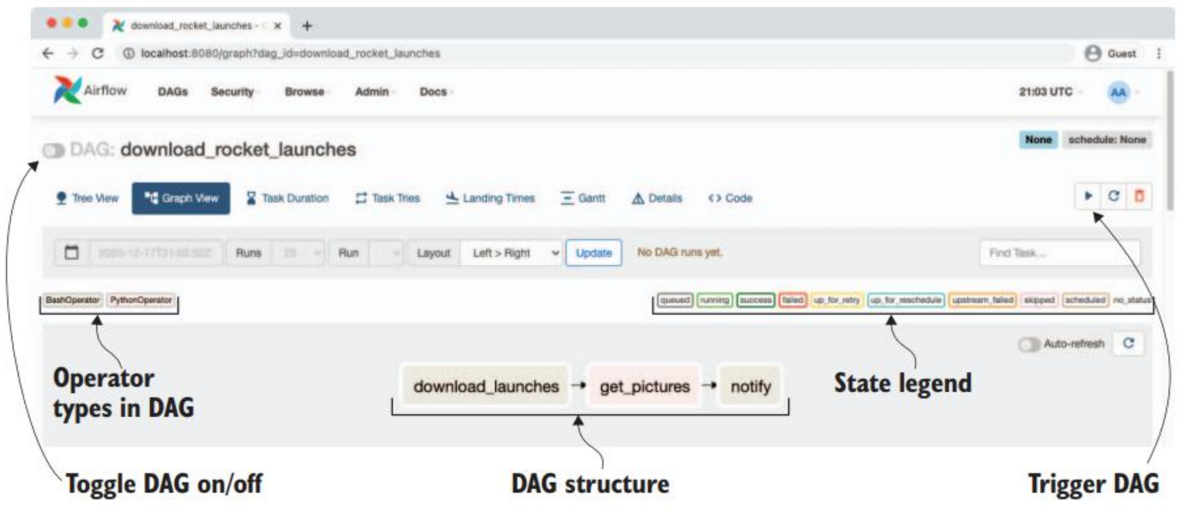


Figure 2 Airflow graph view

The Graph View shows the structure of the script provided to Airflow. Once placed in the **dags** directory, Airflow reads the script and extracts the elements that together form a DAG so that it can be visualized in the user interface. The Graph View displays the structure of the DAG, how the tasks are connected, and the order in which they will be executed. This is one of the views you will likely use most often when developing your workflows.

The state legend shows all the colors you might see during execution. Let's see what happens and run the DAG. First, the DAG needs to be enabled (on) to execute; toggle the button next to the DAG's name for this. Then, click the Play button (Trigger DAG) to execute it.

After triggering the DAG, it will start running, and you will see the current state of the workflow represented by colors (Figure 3).

Since we have defined dependencies between our tasks, consecutive tasks will only start executing once the previous tasks have completed.

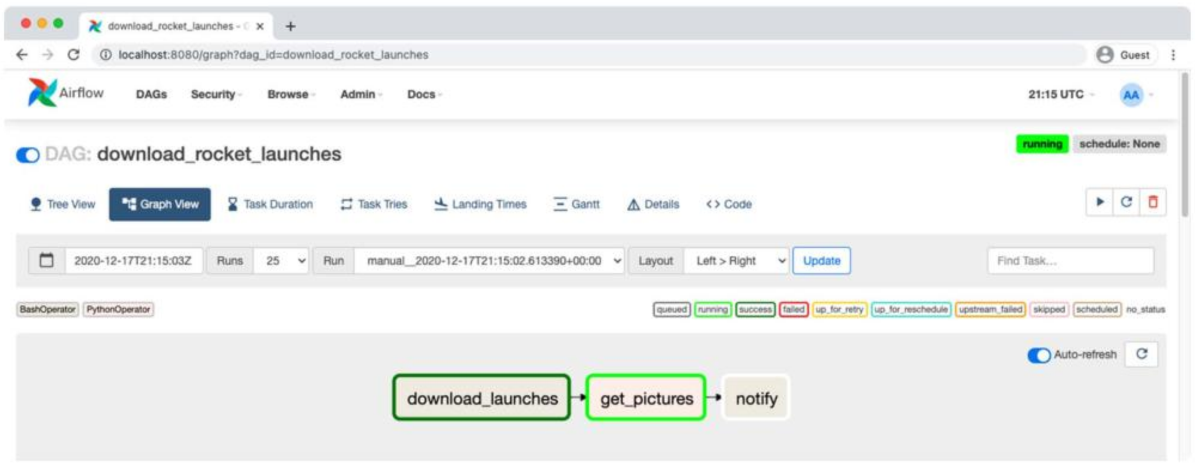
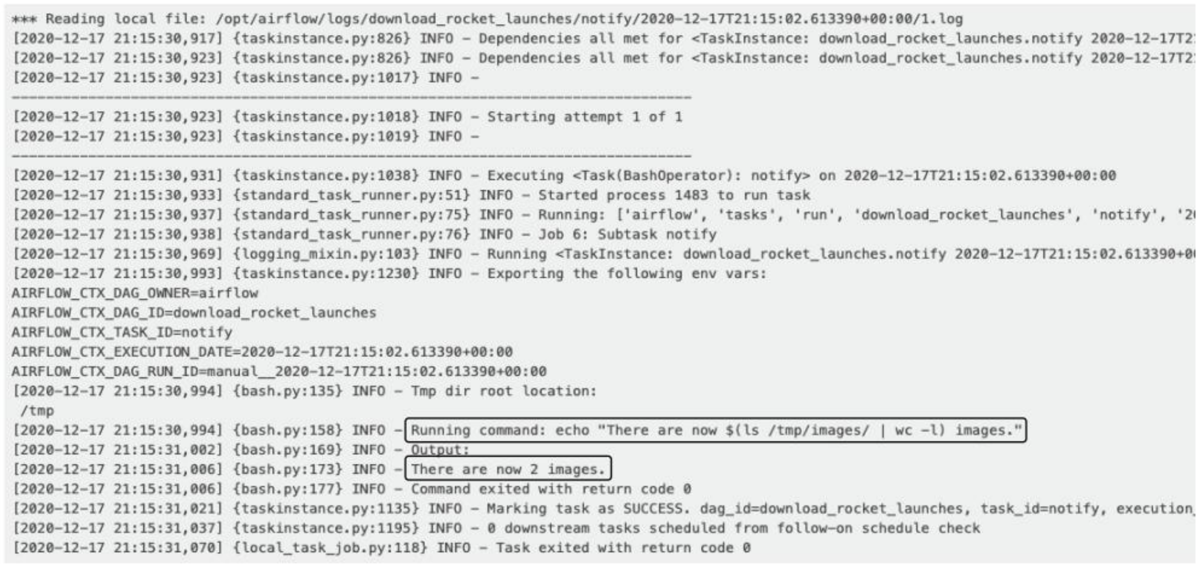


Figure 3 Graph view displaying a running DAG

Logs

All task logs are collected in Airflow, allowing you to search the user interface for outputs or potential issues in case of failure.

For example, click on the **notify** task, and you will see a pop-up window with several options. Click the **Log** button to inspect the logs, as shown in the following figure:



```
*** Reading local file: /opt/airflow/logs/download_rocket_launches/notify/2020-12-17T21:15:02.613390+00:00/1.log
[2020-12-17 21:15:30,917] {taskinstance.py:826} INFO - Dependencies all met for <TaskInstance: download_rocket_launches.notify 2020-12-17T2
[2020-12-17 21:15:30,923] {taskinstance.py:826} INFO - Dependencies all met for <TaskInstance: download_rocket_launches.notify 2020-12-17T2
[2020-12-17 21:15:30,923] {taskinstance.py:1017} INFO -
--------------------------------------------------------------------------------
[2020-12-17 21:15:30,923] {taskinstance.py:1018} INFO - Starting attempt 1 of 1
[2020-12-17 21:15:30,923] {taskinstance.py:1019} INFO -
--------------------------------------------------------------------------------
[2020-12-17 21:15:30,931] {taskinstance.py:1038} INFO - Executing <Task(BashOperator): notify> on 2020-12-17T21:15:02.613390+00:00
[2020-12-17 21:15:30,933] {standard_task_runner.py:51} INFO - Started process 1483 to run task
[2020-12-17 21:15:30,937] {standard_task_runner.py:75} INFO - Running: ['airflow', 'tasks', 'run', 'download_rocket_launches', 'notify', '2
[2020-12-17 21:15:30,938] {standard_task_runner.py:76} INFO - Job 6: Subtask notify
[2020-12-17 21:15:30,969] {logging_mixin.py:103} INFO - Running <TaskInstance: download_rocket_launches.notify 2020-12-17T21:15:02.613390+0
[2020-12-17 21:15:30,993] {taskinstance.py:1230} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=download_rocket_launches
AIRFLOW_CTX_TASK_ID=notify
AIRFLOW_CTX_EXECUTION_DATE=2020-12-17T21:15:02.613390+00:00
AIRFLOW_CTX_DAG_RUN_ID=manual__2020-12-17T21:15:02.613390+00:00
[2020-12-17 21:15:30,994] {bash.py:135} INFO - Tmp dir root location:
 /tmp
[2020-12-17 21:15:30,994] {bash.py:158} INFO - Running command: echo "There are now $(ls /tmp/images/ | wc -l) images."
[2020-12-17 21:15:31,002] {bash.py:169} INFO - Output:
[2020-12-17 21:15:31,006] {bash.py:173} INFO - There are now 2 images.
[2020-12-17 21:15:31,006] {bash.py:177} INFO - Command exited with return code 0
[2020-12-17 21:15:31,021] {taskinstance.py:1135} INFO - Marking task as SUCCESS. dag_id=download_rocket_launches, task_id=notify, execution
[2020-12-17 21:15:31,037] {taskinstance.py:1195} INFO - 0 downstream tasks scheduled from follow-on schedule check
[2020-12-17 21:15:31,070] {local_task_job.py:118} INFO - Task exited with return code 0
```