ECOLE
POLYTECHNIQUE
DE BRUXELLES

2023-2024

INFO-H420 - Management of Data Science and Business Workflows

Assignment 3:Workflows with Apache Airflow

Jintao Ma (ID 000586557)

Xianyun Zhuang (ID 000586733)

Professor

Dimitris Sacharidis

November 2023

Contents

Abstract **1**

1 Exercise 1 **2**

1.1 Environment Configuration: 2

1.2 Define the DAG: 4

1.3 Create a Task to Scan for a Log: 5

1.4 Create a Task to Extract Data: 5

1.5 Create a Task to Transform Data: 6

1.5.1 awk: 6

1.5.2 sed: 7

1.6 Create a Task to Load the Data: 7

1.7 Define the Workflow: 8

2 Exercise 2 **9**

2.1 Test run for each of the tasks: 9

2.2 Test run for the workflow: 12

2.3 Trigger the workflow and monitor a few runs: 13

2.4 Findings 15

2.4.1 Computational Power and Speed: 15

2.4.2 Memory and Storage: 16

2.4.3 Resource Competition: 16

2.4.4 System Configuration and Environment: 16

3 Exercise 3 **17**

3.1 Create an Incoming Webhook in Slack 17

3.2 Create a connection to Slack 18
3.3 Create a new hook within Airflow 18
3.3.1 init: 18
3.3.2 get_conn: 18
3.3.3 send_message: 18
3.4 Result of sending message to slack 20

Abstract

In this report, we have implemented the Assignment 3.

(i) In Exercise1, we finished the process_web_log data pipeline process with 4 tasks.

(ii) In Exercise2, we did test runs for each of the tasks we defined and a test run for the workflow. Additionally, we triggered the workflow and monitored a few runs. Document the test runs with our findings or observations from the runs.

(iii) In Exercise3, we added to the workflow a task that implemented sending a message to Slack after the last task.

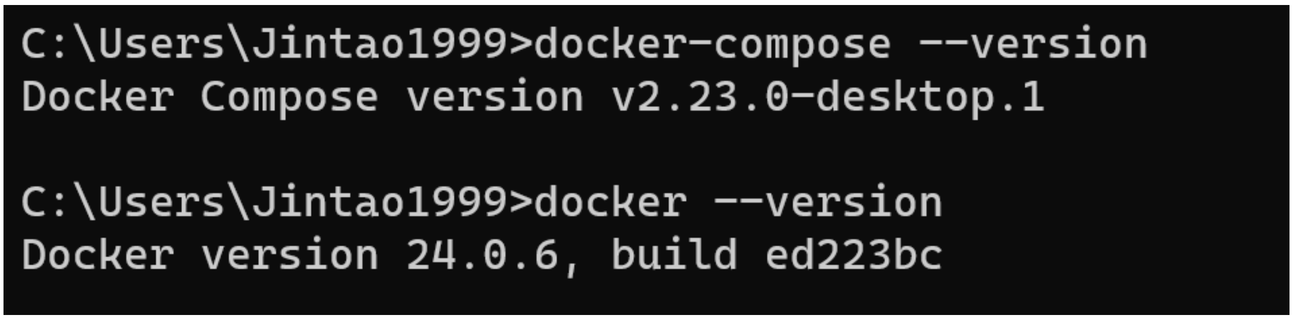
At last, we uploaded the code to Github:
`https://github.com/woshimajintao/ETL-Pipeline-with-Apache-Airflow`

1 Exercise 1

Question: Define the DAG and workflow. Create 4 tasks to scan file, extract data, transform data, and load data
The following is our implementation process and code snippet of process_web_log.

1.1 Environment Configuration:

1. **Docker:** Because both of us used the Windows system we had to install the Docker Desktop and Docker compose as the Figure1.1:



```
C:\Users\Jintao1999>docker-compose --version
Docker Compose version v2.23.0-desktop.1

C:\Users\Jintao1999>docker --version
Docker version 24.0.6, build ed223bc
```

Figure 1.1: Docker Set

2. **YAML File:** Docker-compose.yaml is a configuration file used by Docker Compose to define and run multi-container Docker applications. In this YAML file, we specified all the containers (services) required for an application, along with our configurations such as the image to be used, ports to be mapped, environment variables, volumes, and inter-container dependencies. Docker Compose utilizes this file to orchestrate the creation, configuration, and startup of all the specified services, making the management of complex applications with multiple components more manageable and efficient.

Here we used the airflow 2.7.3.[1]because it was a stable version.

We thought one noticed part of the docker-compose file was the volume. These configurations map the local directory to /opt/airflow/ inside the container. It was related to the path setting of the ETL pipeline.

```
volumes:
  - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
  - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
  - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
  - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
```

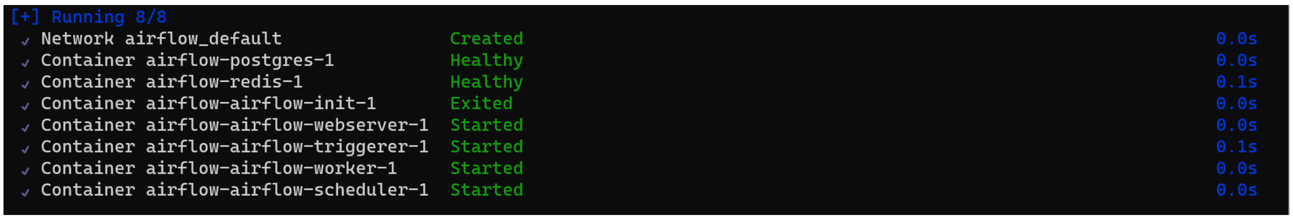
Listing 1.1: Volume Setting

3. **Make the Python File:** We found the Bashoperator was easy to use and implement 3 tasks of the whole ETL process.[2]And we used the PythonOperator to implement the message task. After we finish the Python file, we need to put it in the dag folder of Airflow.

4. **Run the Airflow:** We can use the Windows cmd command to open the airflow on Docker1.2:

```
D:\Apache\Airflow>docker compose up -d
```

Listing 1.2: Open Airflow



```
[+] Running 8/8
 ✓ Network airflow_default                   Created                                                     0.0s
 ✓ Container airflow-postgres-1              Healthy                                                     0.0s
 ✓ Container airflow-redis-1                 Healthy                                                     0.1s
 ✓ Container airflow-airflow-init-1          Exited                                                      0.0s
 ✓ Container airflow-airflow-webserver-1     Started                                                     0.0s
 ✓ Container airflow-airflow-triggerer-1     Started                                                     0.1s
 ✓ Container airflow-airflow-worker-1        Started                                                     0.0s
 ✓ Container airflow-airflow-scheduler-1     Started                                                     0.0s
```

Figure 1.2: open Airflow

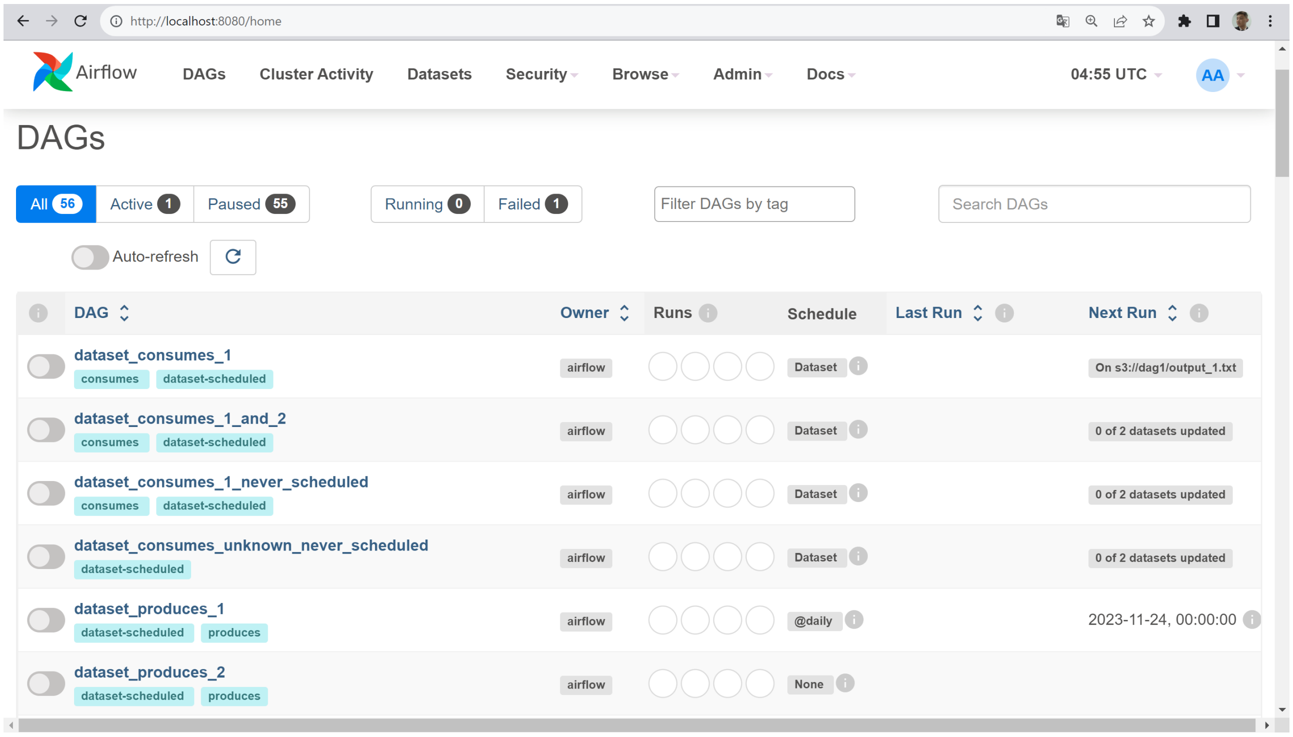
Then we clicked the `http://localhost:8080/` to access the Web UI of Airflow1.3.

Also, we can use the Windows cmd command to close the airflow on Docker1.4:

```
D:\Apache\Airflow>docker compose down
```

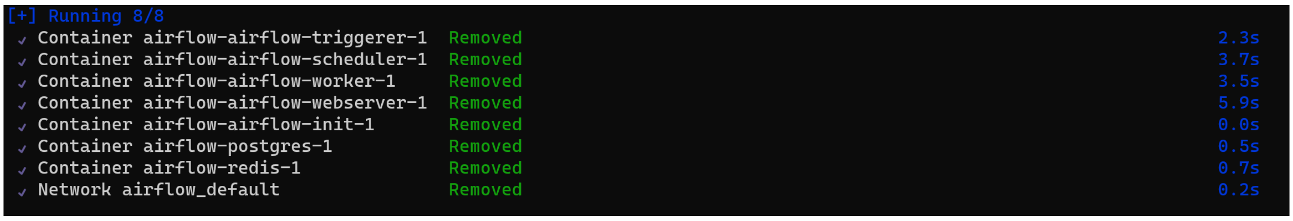
Listing 1.3: close Airflow

[1]https://airflow.apache.org/docs/apache-airflow/2.7.3/docker-compose.yaml
[2]https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/bash.html



DAG	Owner	Runs	Schedule	Last Run	Next Run
dataset_consumes_1 (consumes, dataset-scheduled)	airflow		Dataset		On s3://dag1/output_1.txt
dataset_consumes_1_and_2 (consumes, dataset-scheduled)	airflow		Dataset		0 of 2 datasets updated
dataset_consumes_1_never_scheduled (consumes, dataset-scheduled)	airflow		Dataset		0 of 2 datasets updated
dataset_consumes_unknown_never_scheduled (dataset-scheduled)	airflow		Dataset		0 of 2 datasets updated
dataset_produces_1 (dataset-scheduled, produces)	airflow		@daily		2023-11-24, 00:00:00
dataset_produces_2 (dataset-scheduled, produces)	airflow		None		

Figure 1.3: Airflow Web UI



```
[+] Running 8/8
 ✓ Container airflow-airflow-triggerer-1  Removed                                                         2.3s
 ✓ Container airflow-airflow-scheduler-1  Removed                                                         3.7s
 ✓ Container airflow-airflow-worker-1     Removed                                                         3.5s
 ✓ Container airflow-airflow-webserver-1  Removed                                                         5.9s
 ✓ Container airflow-airflow-init-1       Removed                                                         0.0s
 ✓ Container airflow-postgres-1           Removed                                                         0.5s
 ✓ Container airflow-redis-1              Removed                                                         0.7s
 ✓ Network airflow_default                Removed                                                         0.2s
```

Figure 1.4: Close Airflow

1.2 Define the DAG:

The assignment is related to data science workflow monitor[Dum18]. All graphs were directed acyclic graphs (DAGs). This is our DAG definition block:

```
with DAG(
    dag_id='process_web_log',
    start_date=datetime(2023, 11, 18),
    default_args=default_args,
    schedule_interval=timedelta(days=1),
    catchup=False,
) as dag:
```

Listing 1.4: DAG definition

The dag_id was the name of the DAG. The schedule_interval was to set the scheduling interval of the DAG to once a day.

1.3 Create a Task to Scan for a Log:

We used the FileSensor[3] to implement the scan_for_log task. The FileSensor is a built-in sensor in Apache Airflow that waits for a file or a group of files to be present. The sensor checks for the file's presence in the filesystem and ensures that the file is not being written to before returning.

```
scan_for_file = FileSensor(
        task_id='scan_for_file',
        filepath='/opt/airflow/dags/the_logs/log.txt',
        fs_conn_id='my_file_system',  #connection ID of the filesystem
        timeout=300,
        mode='poke',
        poke_interval=60,
)
```

Listing 1.5: scan_for_log

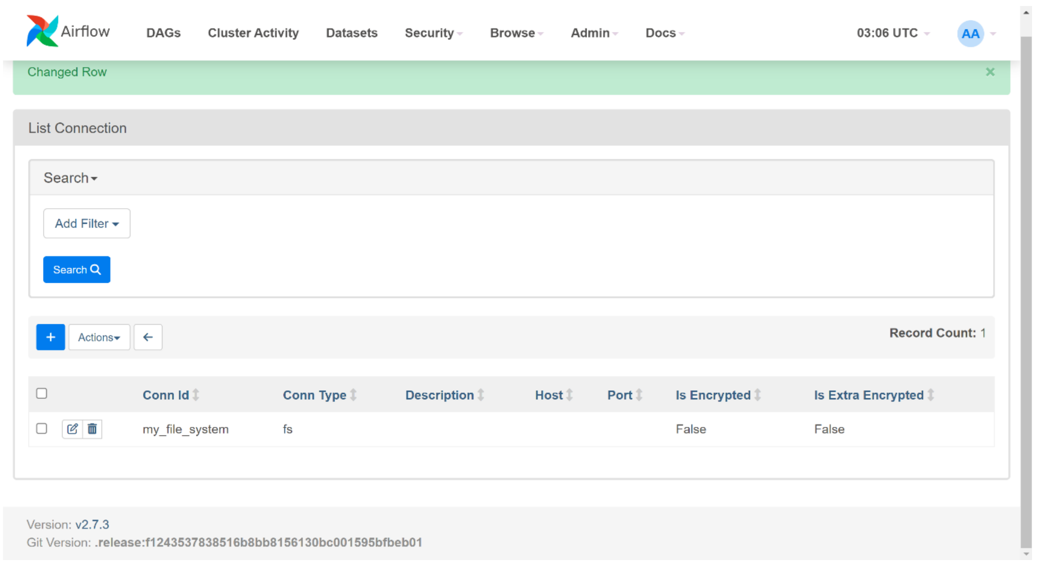
Please note that the fs_conn_id parameter[4] is the connection ID of the filesystem where the file will be located. We needed to set up this connection in Airflow's UI before running the DAG 1.5.

1.4 Create a Task to Extract Data:

We found the Bashoperator[5] was easy to use and implement the whole ETL process because it fits well with docker. So the three tasks were implemented by Bashoperaor.
Grep [6]is a useful command to search for matching patterns in a file. This command would extract all IP addresses from log.txt and write them into extracted_data.txt, separated by a newline.

```
extract_data = BashOperator(
        task_id='extract_data',
```

[3]https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/file.html
[4]https://www.restack.io/docs/airflow-faq-howto-operator-file-01
[5]https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/bash.html
[6]https://www.freecodecamp.org/news/grep-command-in-linux-usage-options-and-syntax-examples/



Changed Row

List Connection

Search

Add Filter

Search

Actions

Record Count: 1

	Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
	my_file_system	fs				False	False

Version: v2.7.3
Git Version: .release:f1243537838516b8bb8156130bc001595bfbeb01

Figure 1.5: fs_conn_id setting

```
        bash_command='grep -E -o "([0-9]{1,3}[\.]){3}[0-9]{1,3}" /opt/airflow/dags/the_logs/log.txt > /opt/airflow/dags/the_logs/extracted_data.txt',
        dag=dag
)
```

Listing 1.6: extract_data

1.5 Create a Task to Transform Data:

Here we used two kinds of Command to implement the transform task[7]. Both of them could filter out all the occurrences of IP address 198.46.149.143 from "extracted_data.txt" and save the output to a file named "transformed_data.txt".

1.5.1 awk:

The awk is a widely used command for text processing.[8] When using awk, we are able to select data – one or more pieces of individual text – based on a pattern we provide.

```
transform_data = BashOperator(
        task_id='transform_data',
        bash_command='awk \'$1 != "198.46.149.143"\' /opt/airflow/dags/the_logs/extracted_data.txt > /opt/airflow/dags/the_logs/transformed_data.txt',
```

[7]https://stackoverflow.com/questions/13579929/how-to-delete-lines-from-file-with-sed-awk
[8]https://www.freecodecamp.org/news/the-linux-awk-command-linux-and-unix-usage-syntax-examples/

```
        dag=dag
)
```

Listing 1.7: transform_data

1.5.2 sed:

Sed stands for stream editor. It is a non-interactive command-line editor. It is primarily used for editing and filtering files. Sed is most often used in scripting, where it is used for editing either a single file or multiple files sequentially. Sed can be used to perform a sequence of editing actions on a single file or a group of files. It can be used to delete lines and words in a file, replace words, and even add text at specific locations in a file [9].

```
transform_data = BashOperator(
        task_id='transform_data',
        bash_command="sed '/198.46.149.143/d' /opt/airflow/dags/the_logs/
    extracted_data.txt> /opt/airflow/dags/the_logs/transformed_data.txt",
        dag=dag
)
```

Listing 1.8: transform_data

1.6 Create a Task to Load the Data:

Here we used the Tar Command to implement the load task[10]. The ‘tar’ stands for tape archive, which is used to create Archive and extract the Archive files. tar command in Linux is one of the important commands that provides archiving functionality in Linux. We can use the Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

The tar command uses the following flags to customize the command input[11]:

‘-c’: Creates a new archive.

‘-z’: Reduces the size of a given file.

‘-v’: Displays verbose output, showing the progress of the archiving process.

‘-f’: Specifies the filename of the archive.

```
load_data = BashOperator(
```

[9]https://tecadmin.net/sed-command-to-delete-line-in-file/google_vignette

[10]https://www.geeksforgeeks.org/tar-command-linux-examples/

[11]https://www.freecodecamp.org/news/how-to-compress-files-in-linux-with-tar-command/

```
        task_id='load_data',
        bash_command='tar -czvf weblog.tar /opt/airflow/dags/the_logs/
    transformed_data.txt',
        dag=dag
)
```

Listing 1.9: load_data

1.7 Define the Workflow:

At last, we defined the workflow that executes the aforementioned tasks in sequence[12]. The load_data task depends on the transform_data task, the transform_data task depends on the extract_data task, and the extract_data task depends on the scan_for_file task.

```
scan_for_file >> extract_data >> transform_data >>load_data
```

Listing 1.10: Dependency Relationship

Putting all of the pieces together, we had our completed DAG.

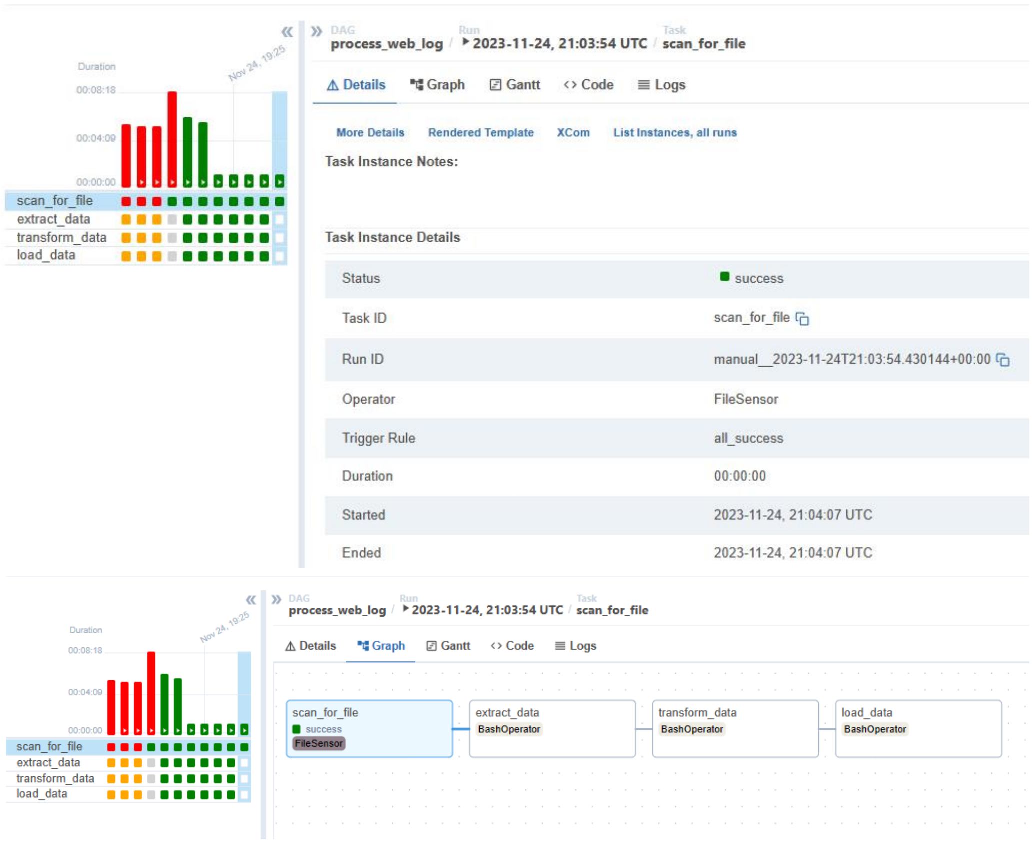
[12]https://airflow.apache.org/docs/apache-airflow/stable/tutorial/pipeline.html

2
Exercise 2

Question: Do a test run for each of the tasks you defined. Once all work as expected, do a test run for the workflow. Finally, trigger/run the workflow and monitor a few runs. In the report, document the test runs, and include any findings or observations that you may have from the runs.

2.1 Test run for each of the tasks:

First, comment out all tasks in the Python file except for the task currently needed for the test run. The Figure2.1 below shows the result for running only the 'scan' task.



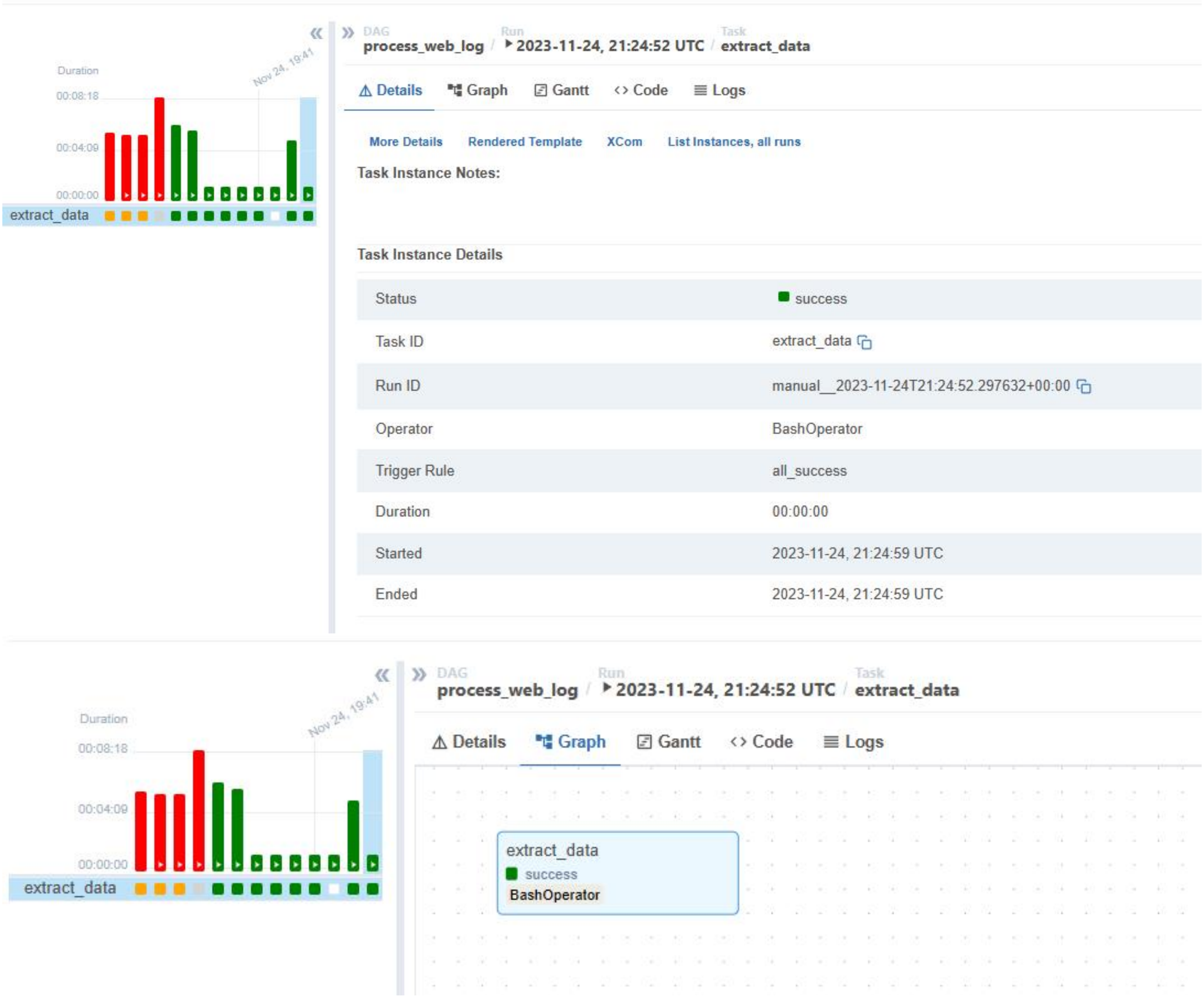
Task Instance Notes:

Task Instance Details

Status	success
Task ID	scan_for_file
Run ID	manual__2023-11-24T21:03:54.430144+00:00
Operator	FileSensor
Trigger Rule	all_success
Duration	00:00:00
Started	2023-11-24, 21:04:07 UTC
Ended	2023-11-24, 21:04:07 UTC

Figure 2.1: Test run result of 'scan' task

We can see from the Details and Graph that the 'scan' task ran successfully. Therefore, we performed the same operation on the other three tasks. The results of the test runs for the three tasks are as follows from the Figure2.2 to the Figure2.4:

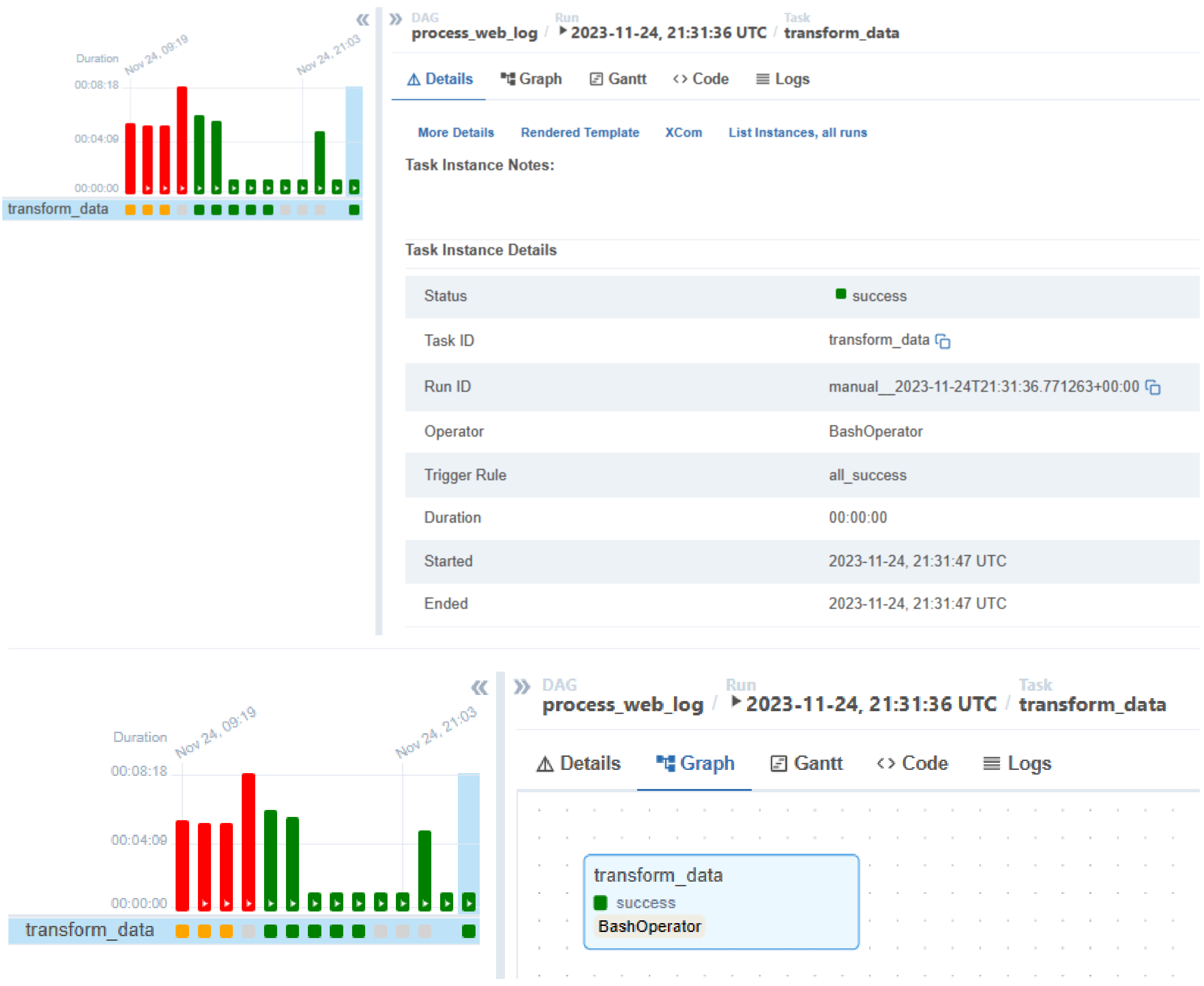


Task Instance Notes:

Task Instance Details

Status	🟩 success
Task ID	extract_data
Run ID	manual__2023-11-24T21:24:52.297632+00:00
Operator	BashOperator
Trigger Rule	all_success
Duration	00:00:00
Started	2023-11-24, 21:24:59 UTC
Ended	2023-11-24, 21:24:59 UTC

Figure 2.2: Test run result of 'extract' task



DAG process_web_log / Run 2023-11-24, 21:31:36 UTC / Task transform_data

Task Instance Notes:

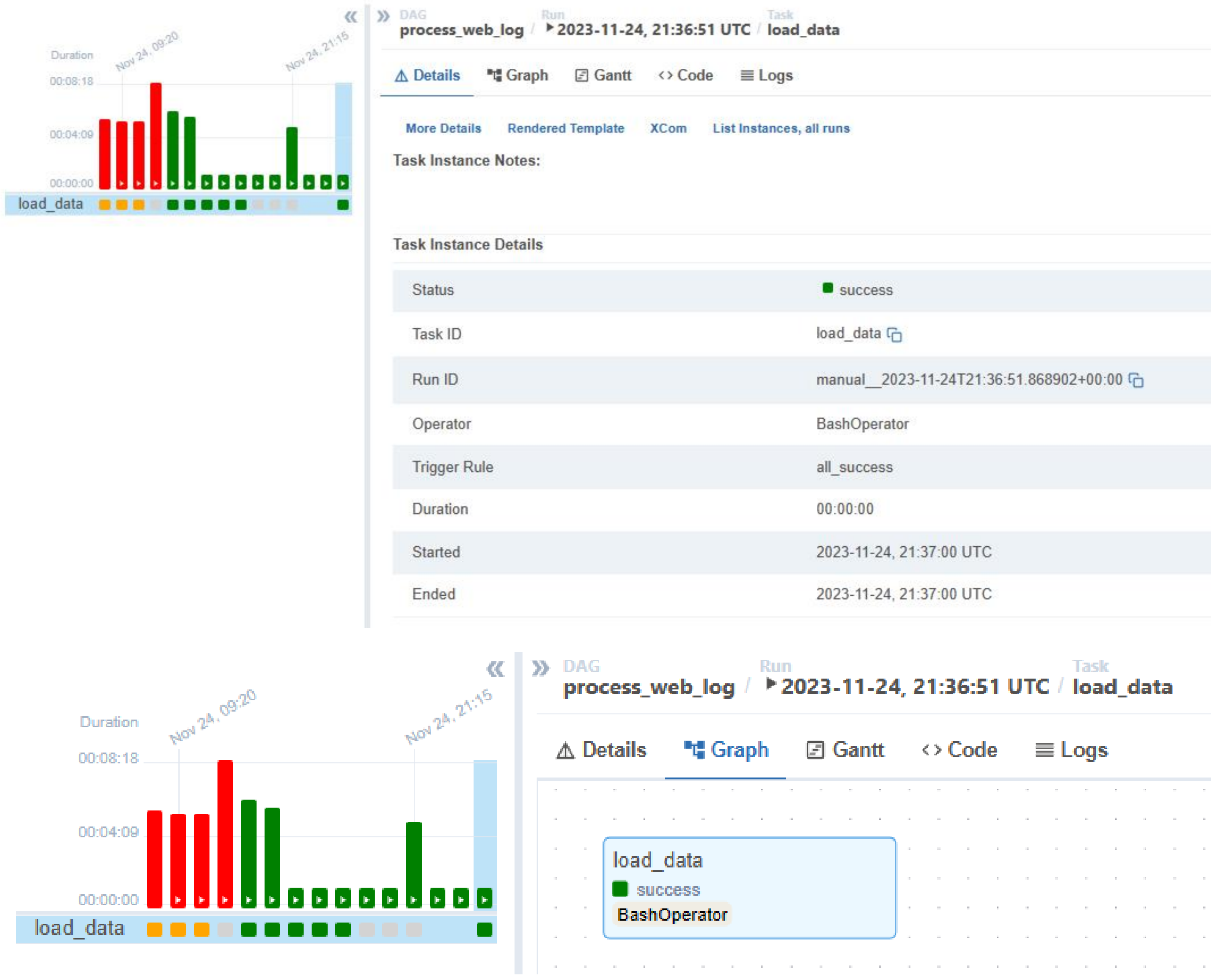
Task Instance Details

Status	🟩 success
Task ID	transform_data
Run ID	manual__2023-11-24T21:31:36.771263+00:00
Operator	BashOperator
Trigger Rule	all_success
Duration	00:00:00
Started	2023-11-24, 21:31:47 UTC
Ended	2023-11-24, 21:31:47 UTC

DAG process_web_log / Run 2023-11-24, 21:31:36 UTC / Task transform_data

transform_data
🟩 success
BashOperator

Figure 2.3: Test run result of ’transform’ task



DAG process_web_log / Run 2023-11-24, 21:36:51 UTC / Task load_data

Task Instance Notes:

Task Instance Details

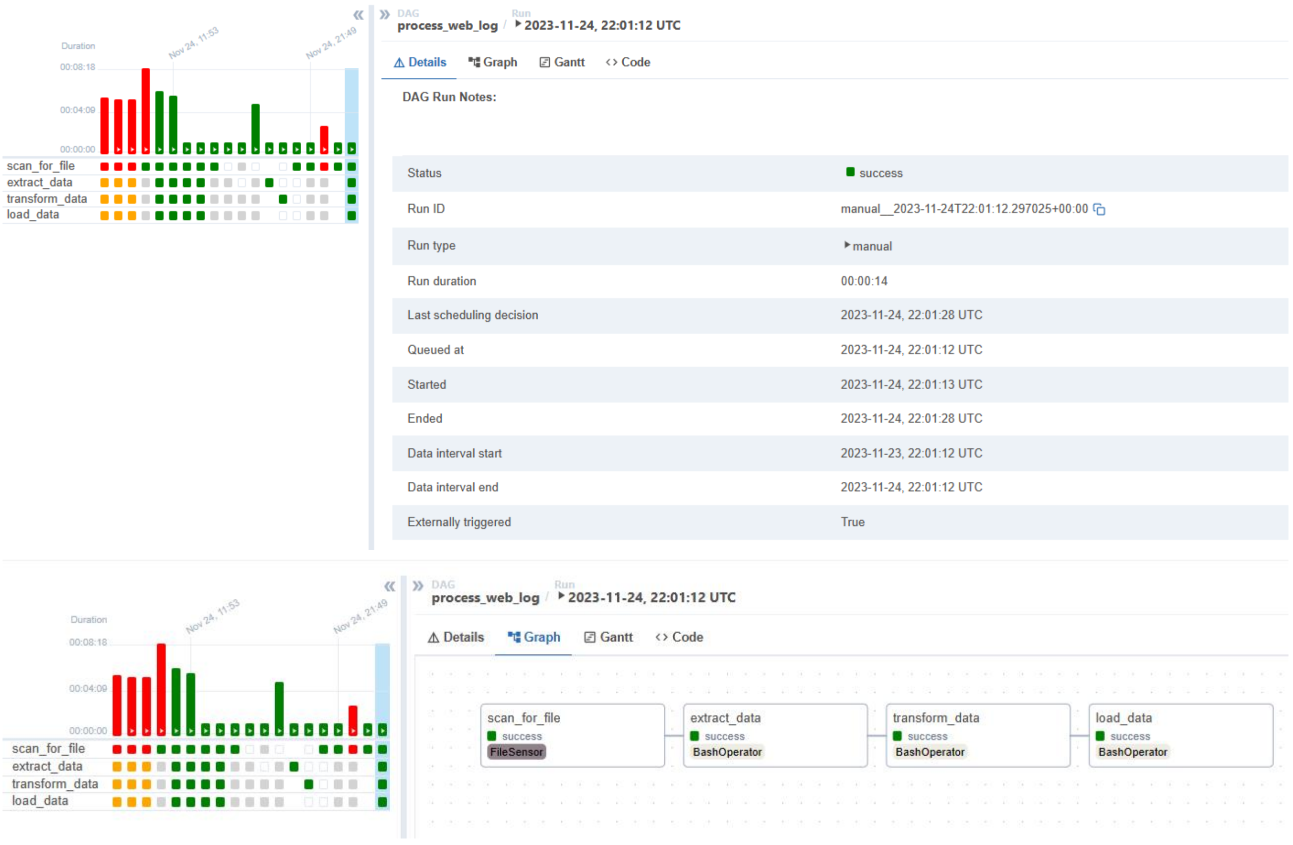
Status	🟩 success
Task ID	load_data
Run ID	manual__2023-11-24T21:36:51.868902+00:00
Operator	BashOperator
Trigger Rule	all_success
Duration	00:00:00
Started	2023-11-24, 21:37:00 UTC
Ended	2023-11-24, 21:37:00 UTC

load_data
🟩 success
BashOperator

Figure 2.4: Test run result of 'load data' task

2.2 Test run for the workflow:

We added all task codes to the DAG for a test run, which was executed successfully. The results are as shown in Figure2.5.


DAG process_web_log / Run 2023-11-24, 22:01:12 UTC

Details Graph Gantt Code

DAG Run Notes:

Status	🟩 success
Run ID	manual__2023-11-24T22:01:12.297025+00:00
Run type	manual
Run duration	00:00:14
Last scheduling decision	2023-11-24, 22:01:28 UTC
Queued at	2023-11-24, 22:01:12 UTC
Started	2023-11-24, 22:01:13 UTC
Ended	2023-11-24, 22:01:28 UTC
Data interval start	2023-11-23, 22:01:12 UTC
Data interval end	2023-11-24, 22:01:12 UTC
Externally triggered	True

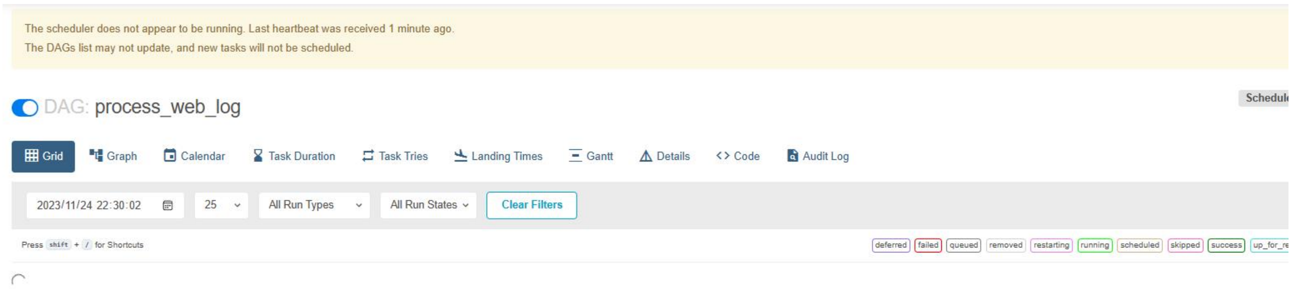
Duration
00:08:18
00:04:09
00:00:00
Nov 24, 11:53
Nov 24, 21:49
scan_for_file
extract_data
transform_data
load_data
scan_for_file success FileSensor
extract_data success BashOperator
transform_data success BashOperator
load_data success BashOperator

Figure 2.5: Test run result of the workflow

2.3 Trigger the workflow and monitor a few runs:

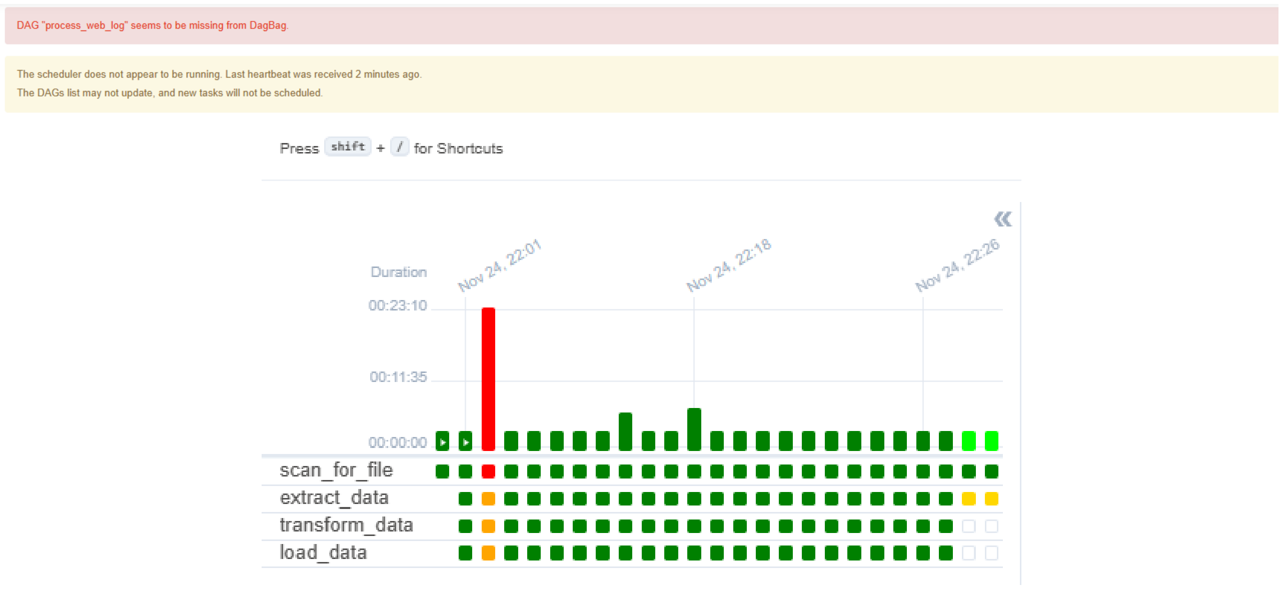
For testing purposes, we conducted two sets of tests by adjusting the execution intervals to 1 minute and 30 seconds, respectively.

When the interval was set to 1 minute, the DAG ran successfully but it took a long time to open the DAG and some of the tasks encountered ’up for retry’ situations, resulting in increased processing time as shown in Figure2.6.



The scheduler does not appear to be running. Last heartbeat was received 1 minute ago.
The DAGs list may not update, and new tasks will not be scheduled.

DAG: process_web_log

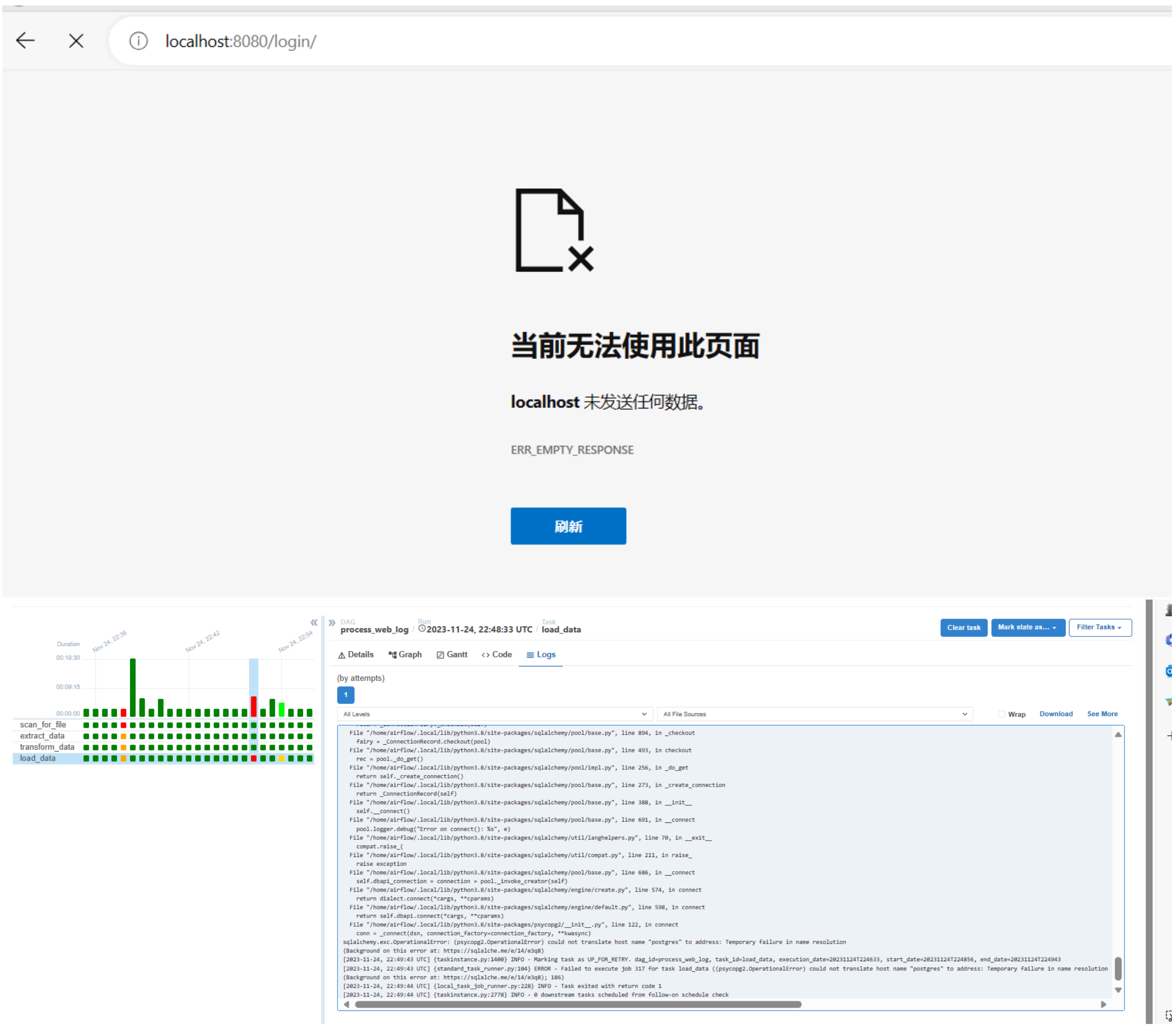


DAG "process_web_log" seems to be missing from DagBag.

The scheduler does not appear to be running. Last heartbeat was received 2 minutes ago.
The DAGs list may not update, and new tasks will not be scheduled.

Figure 2.6: Test run result of triggering the workflow with 1-minute interval

When the interval was set to 30 seconds, the localhost page became inaccessible, and there were also task failures as shown in Figure2.7.



当前无法使用此页面

localhost 未发送任何数据。

ERR_EMPTY_RESPONSE

DAG process_web_log / Run 2023-11-24, 22:48:33 UTC / Task load_data

(by attempts)

```
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 894, in _checkout
    fairy = _ConnectionRecord.checkout(pool)
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 493, in checkout
    rec = pool._do_get()
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/pool/impl.py", line 256, in _do_get
    return self._create_connection()
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 273, in _create_connection
    return _ConnectionRecord(self)
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 388, in __init__
    self.__connect()
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 691, in __connect
    pool.logger.debug("Error on connect(): %s", e)
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/util/langhelpers.py", line 70, in __exit__
    compat.raise_(
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/util/compat.py", line 211, in raise_
    raise exception
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 686, in __connect
    self.dbapi_connection = connection = pool._invoke_creator(self)
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/engine/create.py", line 574, in connect
    return dialect.connect(*cargs, **cparams)
  File "/home/airflow/.local/lib/python3.8/site-packages/sqlalchemy/engine/default.py", line 598, in connect
    return self.dbapi.connect(*cargs, **cparams)
  File "/home/airflow/.local/lib/python3.8/site-packages/psycopg2/__init__.py", line 122, in connect
    conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
sqlalchemy.exc.OperationalError: (psycopg2.OperationalError) could not translate host name "postgres" to address: Temporary failure in name resolution
(Background on this error at: https://sqlalche.me/e/14/e3q8)
[2023-11-24, 22:49:43 UTC] {taskinstance.py:1400} INFO - Marking task as UP_FOR_RETRY. dag_id=process_web_log, task_id=load_data, execution_date=20231124T224633, start_date=20231124T224856, end_date=20231124T224943
[2023-11-24, 22:49:43 UTC] {standard_task_runner.py:104} ERROR - Failed to execute job 317 for task load_data ((psycopg2.OperationalError) could not translate host name "postgres" to address: Temporary failure in name resolution
(Background on this error at: https://sqlalche.me/e/14/e3q8); 186)
[2023-11-24, 22:49:44 UTC] {local_task_job_runner.py:228} INFO - Task exited with return code 1
[2023-11-24, 22:49:44 UTC] {taskinstance.py:2778} INFO - 0 downstream tasks scheduled from follow-on schedule check
```

Figure 2.7: Test run result of triggering the workflow with 30-seconds interval

2.4 Findings

During the test runs, we've noticed a significant impact on the success or failure of task execution based on computer performance and resource utilization. This could be attributed to several factors:

2.4.1 Computational Power and Speed:

Computational Power and Speed: The stronger the computer performance and faster the processor speed, the higher the efficiency in executing tasks. Some tasks require substantial computational power and speed, and inadequate computer performance might lead to slow or failed task execution.

2.4.2 Memory and Storage:

Certain tasks demand significant memory or disk space for data storage and processing. Insufficient memory or limited storage space on the computer can hinder task execution.

2.4.3 Resource Competition:

If other programs or processes are consuming a considerable amount of resources (e.g., large programs, background updates, or system processes), they might compete for the required resources, impacting the execution of tasks.

2.4.4 System Configuration and Environment:

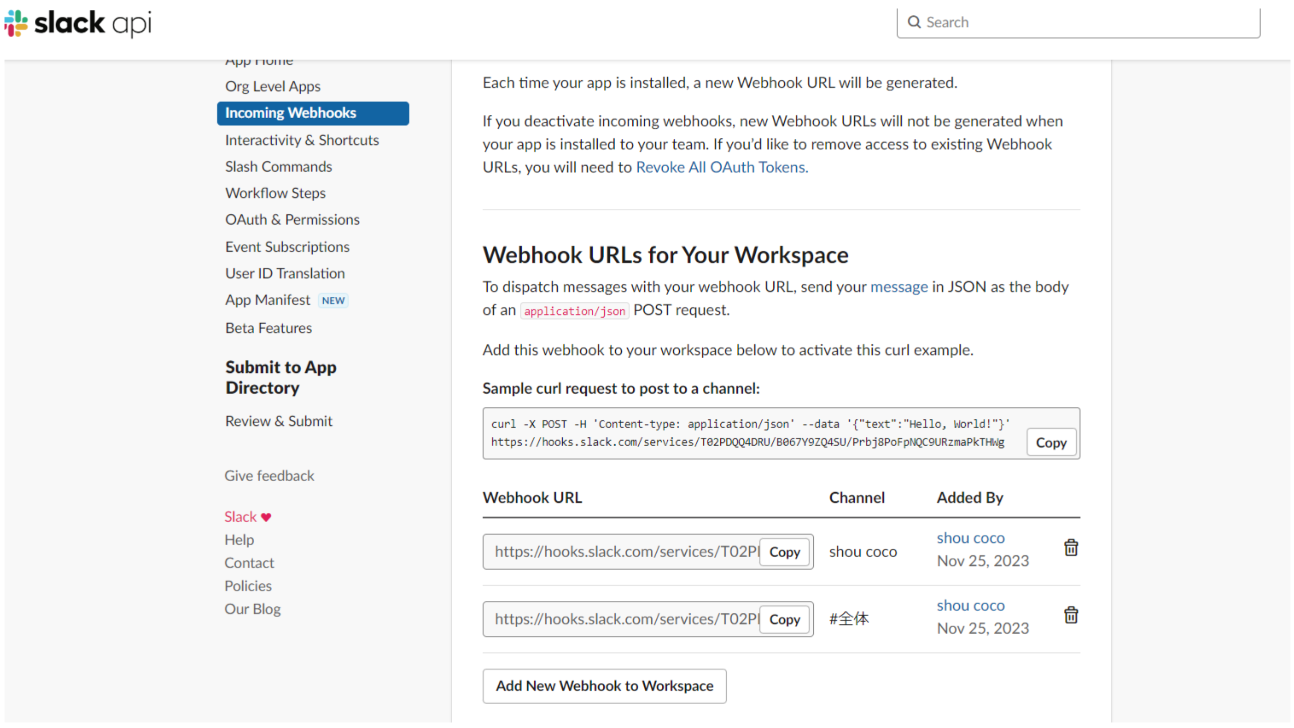
System Configuration and Environment: Occasionally, task success also correlates with the operating system, software versions, and configurations of dependencies. Different system environments can influence task execution.
Therefore, optimizing computer performance and resource utilization by ensuring adequate computational power, memory, and storage, along with optimal system configurations and environments, can potentially enhance the success rate of task execution.

3 Exercise 3

Question: Add to the workflow an additional task after the last task. That task should send a message that the workflow was executed. You are free to define how and where this message is sent. For example, you can post to a Discord or Slack channel, write the message to a text file uploaded to Google Drive or Dropbox. In the report, document how you implemented the message task, and provide evidence that it executes.

3.1 Create an Incoming Webhook in Slack

We started by setting up an Incoming Webhook in Slack.[1] To do this, create a new application on the Slack API website and enable the Incoming Webhook feature as shown in Figure3.1.



Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to Revoke All OAuth Tokens.

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your message in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"}'
https://hooks.slack.com/services/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
```

Webhook URL	Channel	Added By
https://hooks.slack.com/services/T02Pl	shou coco	shou coco Nov 25, 2023
https://hooks.slack.com/services/T02Pl	#全体	shou coco Nov 25, 2023

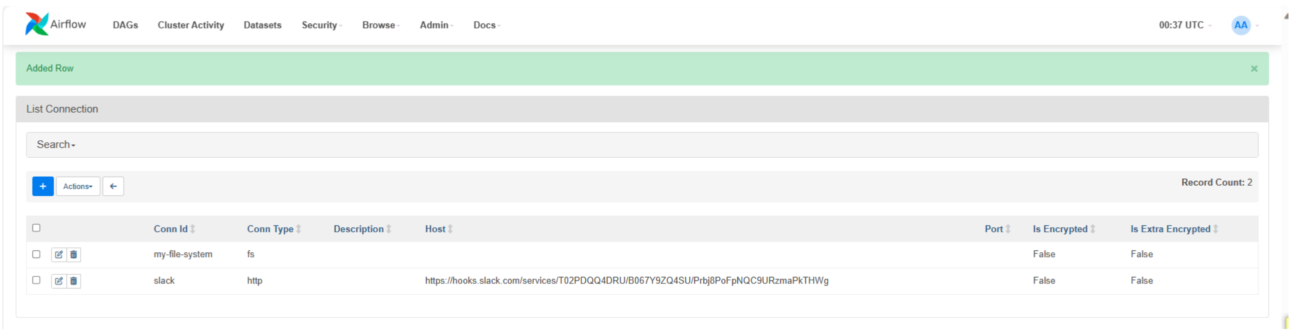
Add New Webhook to Workspace

Figure 3.1: Test run result of triggering the workflow with 30-seconds interval

[1]https://www.restack.io/docs/airflow-knowledge-apache-providers-slack-webhook-http-pypi-operator

3.2 Create a connection to Slack

We used the Incoming Webhook URL obtained from Slack as the host while adding a new connection[2] in Airflow as shown in Figure3.2.



Added Row

List Connection

Record Count: 2

Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
my-file-system	fs				False	False
slack	http		https://hooks.slack.com/services/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		False	False

Figure 3.2: Test run result of triggering the workflow with 30-seconds interval

3.3 Create a new hook within Airflow

Firstly, we've defined a class called SlackHook, which inherits from Airflow's BaseHook class. This class includes methods for communicating with Slack [3].

3.3.1 init:

This is the initialization function that takes a parameter, slack_hook_conn_id, defaulted to 'slack_hook'. It calls the initialization function of the parent class and sets the passed parameter as the class attribute self.slack_hook_conn_id.

3.3.2 get_conn:

This method retrieves Slack's connection information. It uses the get_connection method, passing slack_hook_conn_id, and returns an object conn representing the Slack connection.

3.3.3 send_message:

This method is for sending messages to Slack. It first obtains the Slack connection's URL, constructs the data structure of the message to be sent, and uses requests. post method to send the message to Slack, and return the response result, response.

[2]https://airflow.apache.org/docs/apache-airflow-providers-slack/stable/connections/slack-incoming-webhook.html

[3]https://airflow.apache.org/docs/apacheairflow/stable/_api/airflow/hooks/base/index.html

```python
class SlackHook(BaseHook):
    def __init__(self, slack_hook_conn_id='slack_hook', *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.slack_hook_conn_id = slack_hook_conn_id

    def get_conn(self):
        conn = self.get_connection(self.slack_hook_conn_id)
        return conn

    def send_message(self, message):
        conn = self.get_conn()
        slack_webhook_url = conn.host
        slack_data = {'text': message}
        response = requests.post(
            slack_webhook_url, json=slack_data,
            headers={'Content-Type': 'application/json'}
        )
        return response
```

Listing 3.1: SlackHook

The send_slack_message function stands alone; it instantiates the SlackHook class and sends a message to Slack using the send_message method. The message content is 'Workflow was executed successfully!'.

```python
def send_slack_message():
    slack_hook = SlackHook(slack_hook_conn_id='slack')
    slack_hook.send_message('Workflow was executed successfully!')
```

Listing 3.2: send_slack_message

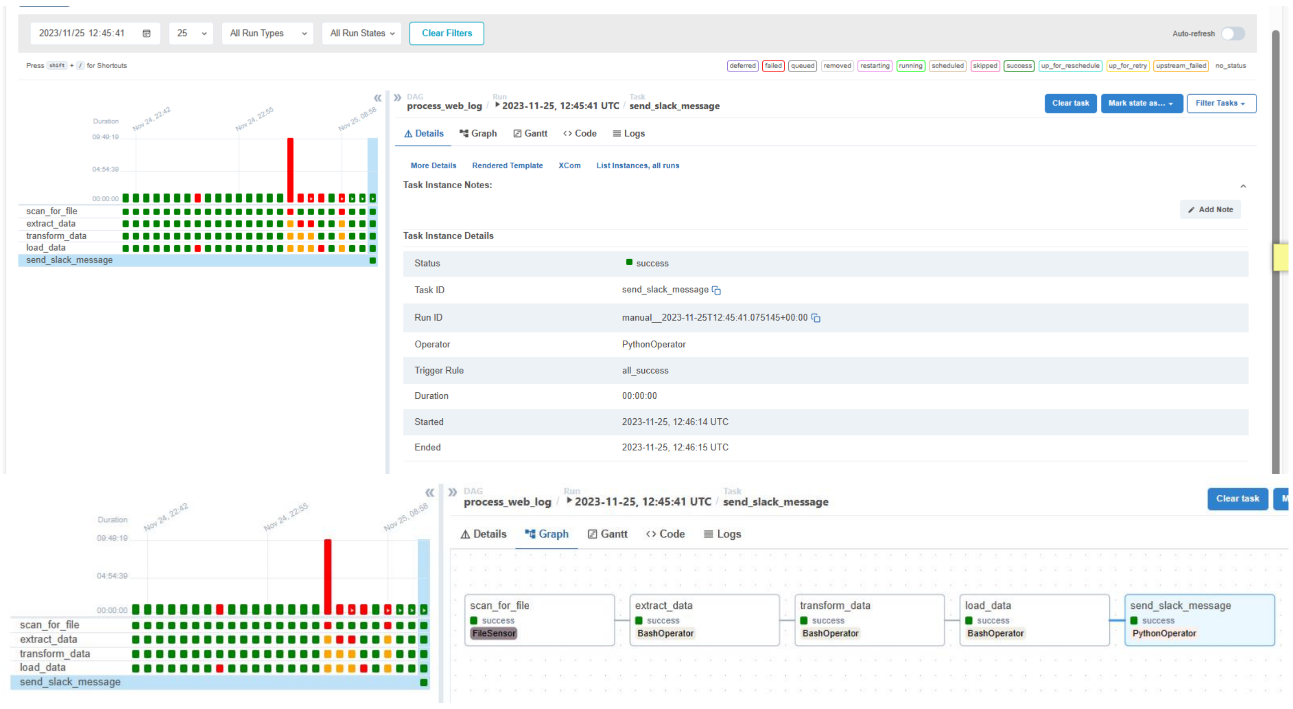
Finally, we defined a task within an Airflow DAG. It uses a PythonOperator named send_slack_message, setting the send_slack_message function as the python_callable to be executed when this task runs.

```python
send_message_task = PythonOperator(
        task_id='send_slack_message',
        python_callable=send_slack_message,
)
```

Listing 3.3: send_message_task

3.4 Result of sending message to slack

After running the DAG, as shown in Figure3.3, we received the message 'Workflow was executed successfully!' in Slack, as shown in Figure3.4.

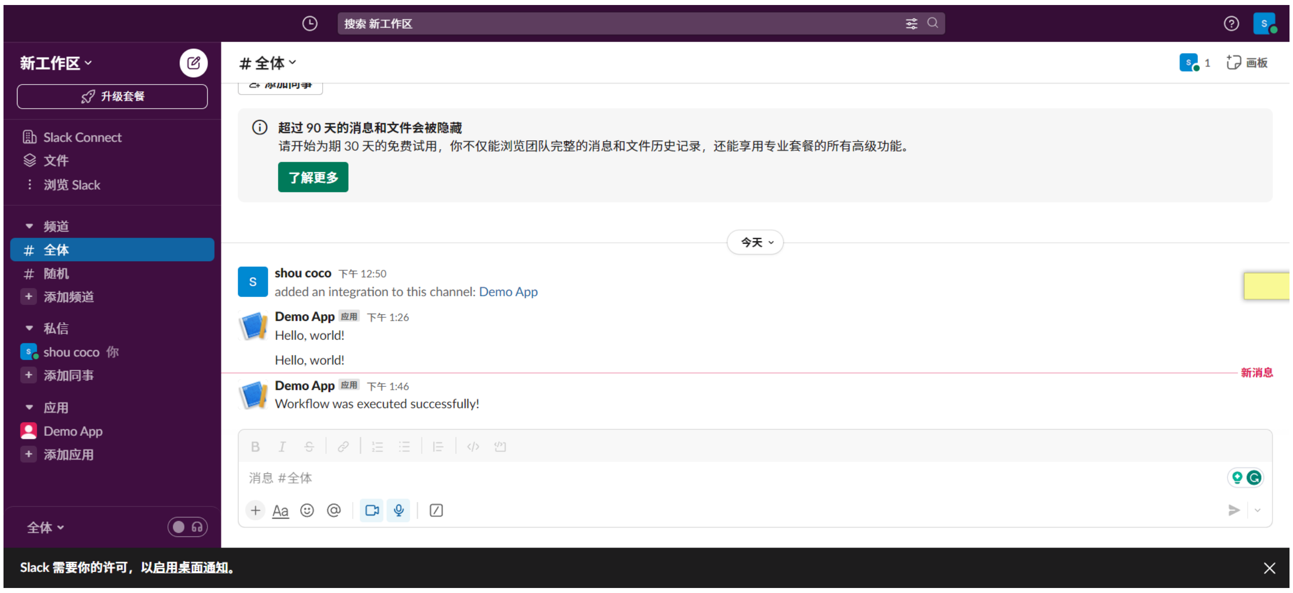


Task Instance Notes:

Task Instance Details

Status	success
Task ID	send_slack_message
Run ID	manual__2023-11-25T12:45:41.075145+00:00
Operator	PythonOperator
Trigger Rule	all_success
Duration	00:00:00
Started	2023-11-25, 12:46:14 UTC
Ended	2023-11-25, 12:46:15 UTC

Figure 3.3: running the DAG with send message to slack task



超过 90 天的消息和文件会被隐藏
请开始为期 30 天的免费试用，你不仅能浏览团队完整的消息和文件历史记录，还能享用专业套餐的所有高级功能。

shou coco 下午 12:50
added an integration to this channel: Demo App

Demo App 应用 下午 1:26
Hello, world!

Hello, world!

Demo App 应用 下午 1:46
Workflow was executed successfully!

Slack 需要你的许可，以启用桌面通知。

Figure 3.4: Test run result of triggering the workflow with 30-seconds interval

Articles

[Dum18] Dumas, Marlon (2018). “Fundamentals of Business Process Management”. In: URL: https://link.springer.com/book/10.1007/978-3-662-56509-4.