Configure a DAG with Celery Executor, PostgreSQL and RabbitMQ

Time to practice!

Instructions

- If your airflow webserver and airflow scheduler are running, stop them by typing `ctrl-C` in their respective terminal.
- `vim ~/airflow/airflow.cfg`
- In the configuration change the following:
  - `executor = CeleryExecutor`
  - `sql_alchemy_conn = postgresql+psycopg2://airflow@localhost:5432/airflow_mdb`
  - `broker_url = pyamqp://admin:rabbitmq@localhost/`
  - `result_backend = db+postgresql://airflow@localhost:5432/airflow_mdb`
  - `worker_log_server_port = 8794`
- Restart airflow
  - `airflow initdb`
  - `airflow webserver`
  - `airflow scheduler`

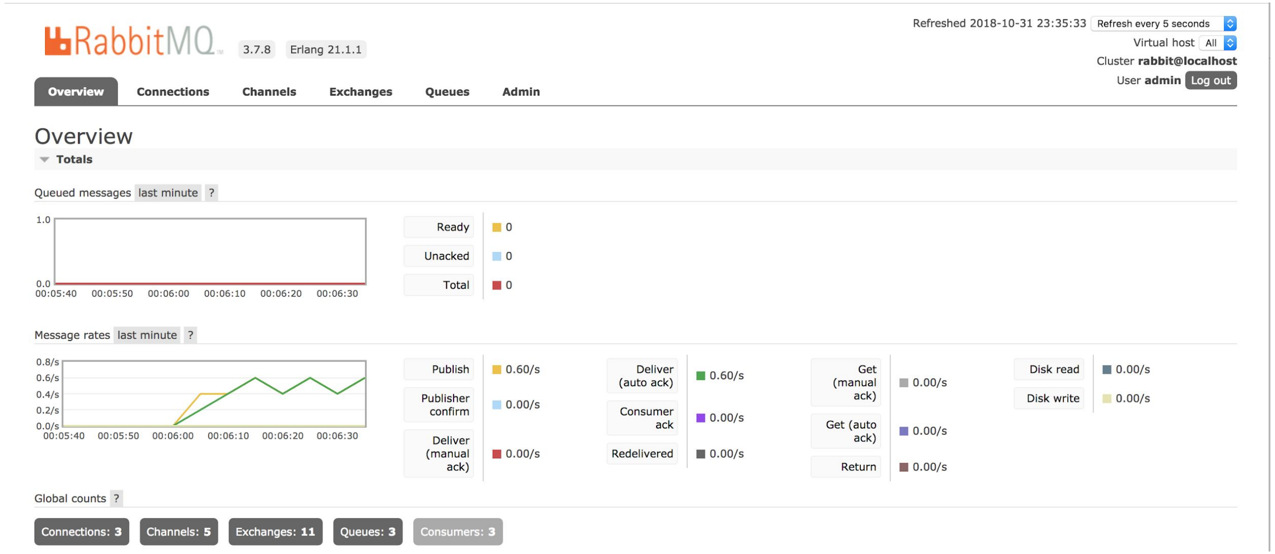
Important Note

- When we type `airflow worker` we actually run a Celery Worker Node.
- When a celery worker is running, it creates one parent process to manage the running tasks. This process handles features like sending/receiving queue messages, tracking status, registering and killing tasks, etc.
- This process then spawns N number of child worker processes that actually execute the individual tasks.
- The number of child worker processes can be determined by typing: `airflow worker -c 2 (for 2 child worker processes)` or by changing the `worker_concurrency` parameter in airflow.cfg

Instructions

- When we start an airflow worker, airflow starts a tiny web server subprocess to serve the workers local log files to airflow main web server. This tiny web server use the parameter `worker_log_server_port`
- Now your Airflow worker is started, let’s take a look at the RabbitMQ interface.

RabbitMQ UI



RabbitMQ 3.7.8 Erlang 21.1.1

Refreshed 2018-10-31 23:35:33 Refresh every 5 seconds
Virtual host All
Cluster **rabbit@localhost**
User **admin** Log out

Overview

Totals

Queued messages last minute ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last minute ?

Publish	🟨 0.60/s	Deliver (auto ack)	🟩 0.60/s	Get (manual ack)	⬜ 0.00/s	Disk read	⬛ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟪 0.00/s	Disk write	🟨 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Return	🟫 0.00/s		

Global counts ?

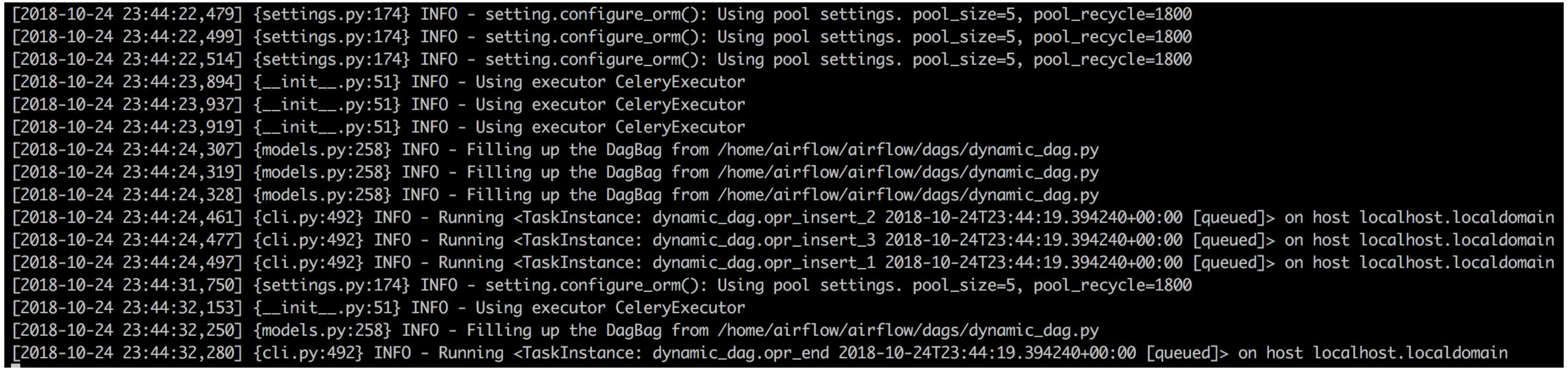
Connections: **3** Channels: **5** Exchanges: **11** Queues: **3** Consumers: **3**

Instructions

- Now we are going to actually reuse the DAG 'dynamic_dag' but now with Celery Executors as we have configured into airflow.cgf
- Go on the Airflow UI and turn ON the dynamic_dag's toggle. Then click on the 'trigger run' icon on the right into the box links.
- After refreshing a couple of times the Airflow UI you should see that your DAG Run ended well. So how can I see if I'm actually running my DAG in a distributed mode ?

Instructions

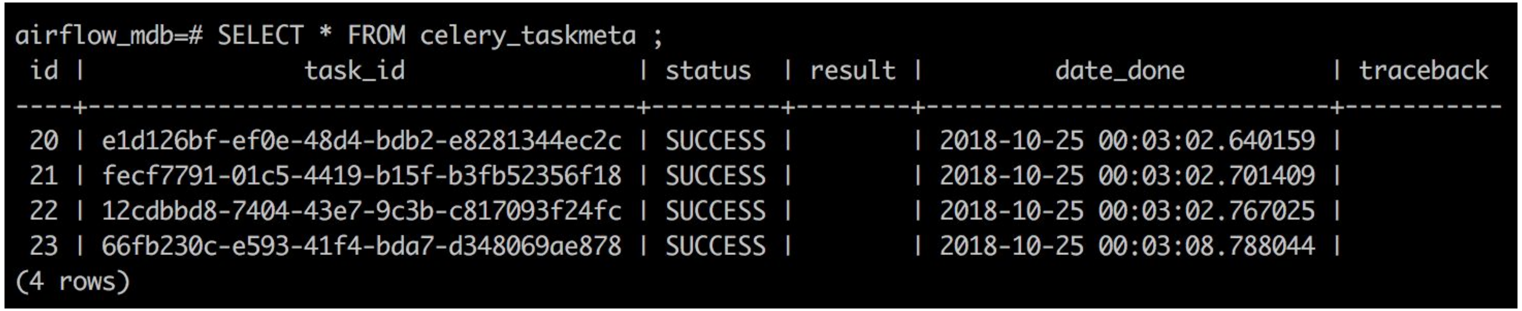
- There are three spots to watch, the first one is actually the worker logs:



```
[2018-10-24 23:44:22,479] {settings.py:174} INFO - setting.configure_orm(): Using pool settings. pool_size=5, pool_recycle=1800
[2018-10-24 23:44:22,499] {settings.py:174} INFO - setting.configure_orm(): Using pool settings. pool_size=5, pool_recycle=1800
[2018-10-24 23:44:22,514] {settings.py:174} INFO - setting.configure_orm(): Using pool settings. pool_size=5, pool_recycle=1800
[2018-10-24 23:44:23,894] {__init__.py:51} INFO - Using executor CeleryExecutor
[2018-10-24 23:44:23,937] {__init__.py:51} INFO - Using executor CeleryExecutor
[2018-10-24 23:44:23,919] {__init__.py:51} INFO - Using executor CeleryExecutor
[2018-10-24 23:44:24,307] {models.py:258} INFO - Filling up the DagBag from /home/airflow/airflow/dags/dynamic_dag.py
[2018-10-24 23:44:24,319] {models.py:258} INFO - Filling up the DagBag from /home/airflow/airflow/dags/dynamic_dag.py
[2018-10-24 23:44:24,328] {models.py:258} INFO - Filling up the DagBag from /home/airflow/airflow/dags/dynamic_dag.py
[2018-10-24 23:44:24,461] {cli.py:492} INFO - Running <TaskInstance: dynamic_dag.opr_insert_2 2018-10-24T23:44:19.394240+00:00 [queued]> on host localhost.localdomain
[2018-10-24 23:44:24,477] {cli.py:492} INFO - Running <TaskInstance: dynamic_dag.opr_insert_3 2018-10-24T23:44:19.394240+00:00 [queued]> on host localhost.localdomain
[2018-10-24 23:44:24,497] {cli.py:492} INFO - Running <TaskInstance: dynamic_dag.opr_insert_1 2018-10-24T23:44:19.394240+00:00 [queued]> on host localhost.localdomain
[2018-10-24 23:44:31,750] {settings.py:174} INFO - setting.configure_orm(): Using pool settings. pool_size=5, pool_recycle=1800
[2018-10-24 23:44:32,153] {__init__.py:51} INFO - Using executor CeleryExecutor
[2018-10-24 23:44:32,250] {models.py:258} INFO - Filling up the DagBag from /home/airflow/airflow/dags/dynamic_dag.py
[2018-10-24 23:44:32,280] {cli.py:492} INFO - Running <TaskInstance: dynamic_dag.opr_end 2018-10-24T23:44:19.394240+00:00 [queued]> on host localhost.localdomain
```

Instructions

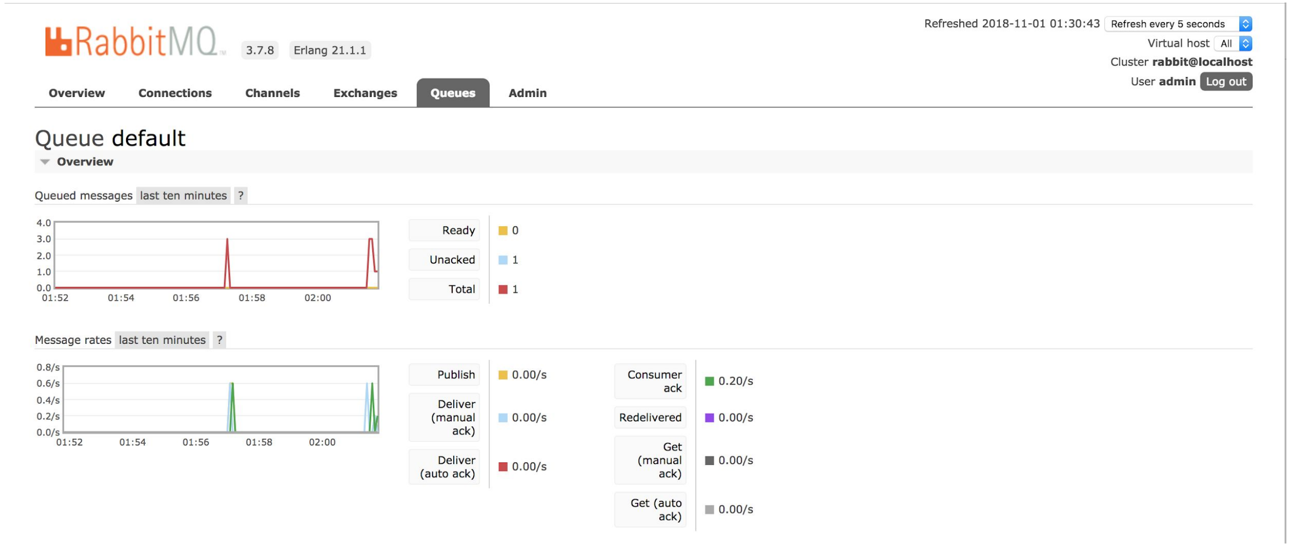
- The second spot is into the PostgreSQL database:



```
airflow_mdb=# SELECT * FROM celery_taskmeta ;
 id |               task_id                | status  | result |          date_done          | traceback
----+--------------------------------------+---------+--------+----------------------------+-----------
 20 | e1d126bf-ef0e-48d4-bdb2-e8281344ec2c | SUCCESS |        | 2018-10-25 00:03:02.640159 |
 21 | fecf7791-01c5-4419-b15f-b3fb52356f18 | SUCCESS |        | 2018-10-25 00:03:02.701409 |
 22 | 12cdbbd8-7404-43e7-9c3b-c817093f24fc | SUCCESS |        | 2018-10-25 00:03:02.767025 |
 23 | 66fb230c-e593-41f4-bda7-d348069ae878 | SUCCESS |        | 2018-10-25 00:03:08.788044 |
(4 rows)
```

Instructions

- And the third spot is from the RabbitMQ UI:



RabbitMQ 3.7.8 Erlang 21.1.1

Refreshed 2018-11-01 01:30:43 Refresh every 5 seconds
Virtual host All
Cluster **rabbit@localhost**
User **admin** Log out

Queue default

Overview

Queued messages last ten minutes ?

Ready	0
Unacked	1
Total	1

Message rates last ten minutes ?

Publish	0.00/s	Consumer ack	0.20/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s
		Get (auto ack)	0.00/s