







Hybrid Executors Have Your Cake and Eat it Too

Niko Oliveira











Who Am I?

- Apache Airflow committer
- Sr. software engineer at Amazon
 - Amazon Managed Workflows for Apache Airflow (MWAA)
 - Founding member of the Amazon Apache Airflow Open Source Team
- Spent much of the last year working on Airflow executors (again)







PIRFLO PER

- Executors facilitate the running of Airflow tasks (Task Instances)
- The Airflow scheduler decides when a task should run and the executor decides where and how
- Examples:
 - CeleryExecutor, KubernetesExecutor, LocalExecutor, ECS Executor
- Runs within the Airflow scheduler process
- Pluggable and extensible, you can write your very own!

- There are many types of Airflow executors, but some major ones include:
 - Local Executors: Airflow tasks are executed on the same host that the executor (i.e. scheduler) is running on. E.g.: LocalExecutor

- There are many types of Airflow executors, but some major ones include:
 - Local Executors: Airflow tasks are executed on the same host that the executor (i.e. scheduler) is running on. E.g.: LocalExecutor
 - Remote Queued/Batched Executors: Airflow tasks are sent to a central queue where remote workers pull tasks to execute. Often workers are persistent and run multiple tasks at once: E.g.: CeleryExecutor, AwsBatchExecutor

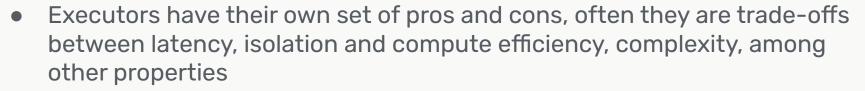
- There are many types of Airflow executors, but some major ones include:
 - Local Executors: Airflow tasks are executed on the same host that the executor (i.e. scheduler) is running on. E.g.: LocalExecutor
 - Remote Queued/Batched Executors: Airflow tasks are sent to a central queue where remote workers pull tasks to execute. Often workers are persistent and run multiple tasks at once: E.g.: CeleryExecutor, AwsBatchExecutor
 - Remote Containerized Executors: Airflow tasks are executed ad hoc inside containers/pods. Each task is isolated in its own environment. E.g.:
 KubernetesExecutor, AwsEcsExecutor







Executor Compromises











Executor Compromises

- Executors have their own set of pros and cons, often they are trade-offs between latency, isolation and compute efficiency, complexity, among other properties
- Running multiple executors would allow you to make better use of the strengths of all the available executors and avoid their weaknesses







Executor Compromises

- Executors have their own set of pros and cons, often they are trade-offs between latency, isolation and compute efficiency, complexity, among other properties
- Running multiple executors would allow you to make better use of the strengths of all the available executors and avoid their weaknesses
- Starting with version 2.10.0, Airflow can now operate with Multiple Executor Configuration (formerly Hybrid Executors)!









How to Use it: Configuration

[core]

executor = 'LocalExecutor, KubernetesExecutor, my.custom.module.ExecutorClass:ShortName'

- The same **core.executor** Airflow configuration is used for multiple executors
- The first executor in the list is the default, it behaves in the same way a single executor configuration did <=2.9
- Executors can be given aliases (e.g. ShortName). This allows easier specification in the DAG code since custom modules can be guite long
- Airflow core executors are still referenced by their short names (e.g. LocalExecutor)









How to Use it: Writing DAGs

[core]

executor = 'LocalExecutor, KubernetesExecutor, my.custom.module.ExecutorClass:ShortName'

```
BashOperator(
    task_id="hello_world_1",
    # Will use the custom executor class
    executor="ShortName",
    bash_command="echo 'hello world!'",
# Will use the KubernetesExecutor
@task(executor="KubernetesExecutor")
def hello_world_2():
    print("hello world!")
# Will use the default LocalExecutor
@task()
def hello_world_3():
    print("hello world!")
```

- Use the `executor` field on tasks/operators to specify which Executor (from configuration) should run each task
- Specify no executor at all to use the default executor







How to Use it: Writing DAGs

[core]

executor = 'LocalExecutor, KubernetesExecutor, my.custom.module.ExecutorClass:ShortName'

```
def hello_world():
    print("hello world!")
def hello_world_again():
    print("hello world again!")
with DAG(
    dag_id="hello_worlds",
    # Applies to all tasks in the DAG
    default_args={"executor": "KubernetesExecutor"},
  as dag:
    # All tasks will use the executor from default args
    hw = hello_world()
    hw_again = hello_world_again()
```

- You can specify an Executor to use for every task in a DAG by leveraging `default_args`
- Individual tasks/operators may still override the default if the 'executor' field is explicitly set









Keeping an Eye on Things

Metrics

- If only a single executor is configured in Airflow configuration, executor metrics behave the same as they did before (for Airflow version <= 3.0)
 - E.g.: executor.open_slots
- If multiple executors are configured then metrics are emitted for each executor explicitly (Executor classname in the metric name)
 - E.g.: executor.open_slots.<executor_class_name>









Keeping an Eye on Things

Metrics

- If only a single executor is configured in Airflow configuration, executor metrics behave the same as they did before (for Airflow version <= 3.0)
 - E.g.: executor.open_slots
- o If multiple executors are configured then metrics are emitted for each executor explicitly
 - E.g.: executor.open_slots.<executor_class_name>

Logging

- Logging works the same when using multiple Executors
- Executor logs are emitted in the Airflow Scheduler log output.









Keeping an Eye on Things

- Metrics
 - If only a single executor is configured in Airflow configuration, executor metrics behave the same as they did before (for Airflow version <= 3.0)
 - E.g.: executor.open_slots
 - o If multiple executors are configured then metrics are emitted for each executor explicitly
 - E.g.: executor.open_slots.<executor_class_name>
- Logging
 - Logging works the same when using multiple Executors
 - Executor logs are emitted in the Airflow Scheduler log output.
- Tasks Instances have a record of the Executor they ran with in the DB







PIRF LOI

FAQs: What if I...?

- ...configure my task to use an executor that isn't present?
 - o Airflow will detect this mismatch and fail to parse the DAG and show a banner in the UI









FAQs: What if I...?

- ...configure my task to use an executor that isn't present?
 - Airflow will detect this mismatch and fail to parse the DAG and show a banner in the UI
- ...add an executor to core.executor but no tasks make use of it?
 - Airflow will not be dramatically affected, this is fine!









FAQs: What if I...?

- ... configure my task to use an executor that isn't present?
 - o Airflow will detect this mismatch and fail to parse the DAG and show a banner in the UI
- ... add an executor to core.executor but no tasks use it?
 - Airflow will not be dramatically affected, this is fine!
- ... change my core.executor configuration during or between DAG runs?
 - o Tasks will still run with whichever executor they are specified to run with
 - Tasks using the default executor will run on whatever the default is at the time of execution







RIRELON PLANT

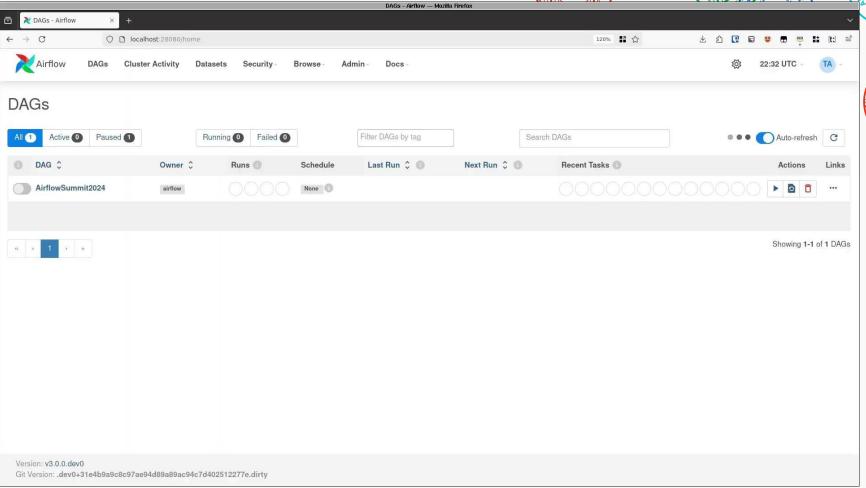
Out With The Old, In With The New

- A note on the existing "statically coded" hybrid executors:
 LocalKubernetesExecutor and CeleryKubernetesExecutor
 - Handcrafted/static combinations of executors, does not scale well
 - Make use of the queue field to direct tasks, misuse of the field
 - Creating all possible combinations is completely unreasonable
 - o Do not use the public base executor interface, changes often not propagated correctly
 - Using these hybrid Executors is no longer recommended!

















Questions?



in linkedin.com/in/niko-oliveira-aws