

AIRFLOW NOTES

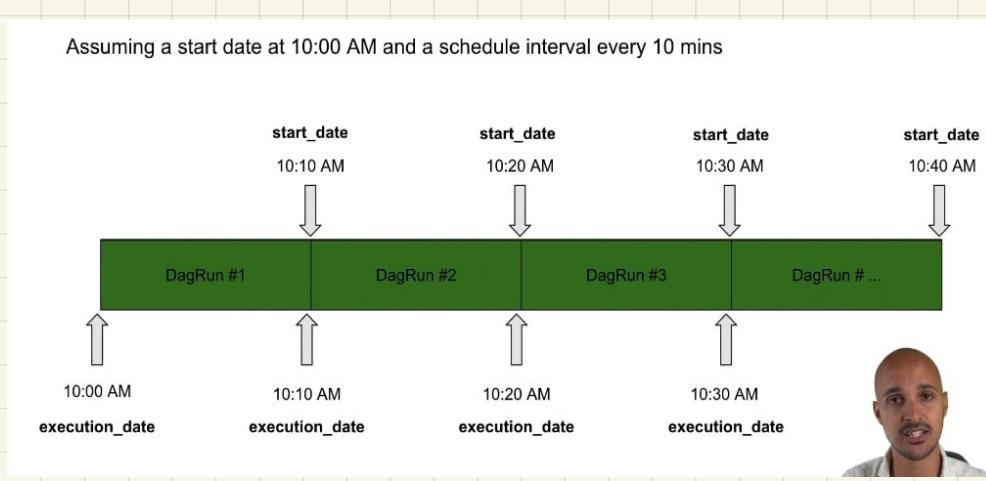


Hannah



- `airflow ignore` → to ignore folder in logs (like gitignore) utility
- `with clause` → permette di non passare dag=dag ad ogni task
- `Dag with same id` → non abbiamo errori ma non sappiamo quale viene caricato
- `description` → buone convenzioni aggiungere i tagli
- `start_date` → necessarie per i task, ma per il dag → buone norme nel dag
- `schedule_interval` → ogni quanto viene caricato
- `dagrun_timeout` → tempo massimo di creazione di un dag → NO DEFAULT
- `tags` ⇒ si può usare per filtrare i dag
- `catchup` ⇒ set always to false
- `max_active_runs` → numero massimo di dag in esecuzione

Assuming a start date at 10:00 AM and a schedule interval every 10 mins



SCHEDULE

DAG : processor - customer ↵ start_date: 01/01 10:00 AM
`schedule_interval = "@daily" → 0 0 * * *`

① 01/01 00:00 ② 01/02 00:00 ③

`schedule_interval = timedelta(days=1)`

① 01/01 10:00 AM ② 01/02 10:00 AM

CRON

√

TIMEDELTA

We can use **Variable** to share common variable between **tasks**.
Esse possono essere integrati e nasce - quindi anche per dati sensibili (also_ssn)

Some Additional Notes

1: There are 6 different ways of creating variables in Airflow 🎉

- Airflow UI 🎯
- Airflow CLI 🎯
- REST API 🎯
- Environment Variables ❤️
- Secret Backend ❤️
- Programmatically 😊

Whichever one you choose depends on both use case and personal preference.

2: By creating a variable with an environment variable you:

- avoid making a connection to your DB
- hide sensitive values (the variable can only be fetched within a DAG)

3: It's worth noting that it is possible to create connections with environment variables. You just have to export the env variable...

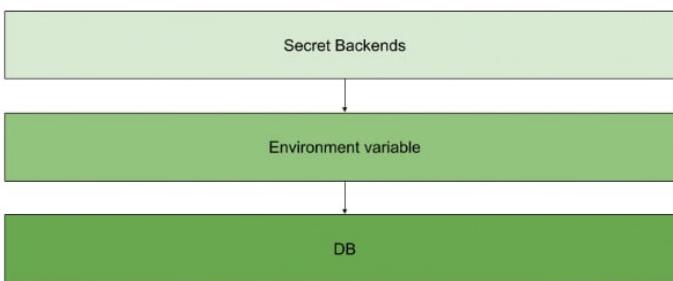
`AIRFLOW_CONN_NAME_OF_CONNECTION=your_connection`

...and get the same benefits as stated above.

4: Ultimately, if you really want a secure way of storing your variables or connections, use a Secret Backend.

To learn more about this, click on this [link](#)

5: Finally, here is the order in which Airflow checks if your variable/connection exists:



Another way to share variable between logs is XCOM:

- `xcom_push (chave, valores)`
- `xcom_pull (chave, task_id)`

In `pythonOperator` or other with `RETURN`, the return is automatically pushed.

V
A
R
I
A
B
L
E

DECORATORS → Si possono usare per estendere i task
i task con gli operatori. Si possono creare legan
dipendenze ecc tramite l'uso di funzioni.

TASK GROUP → Si può usare per raggruppare più task in un subdag

BRANCH OPERATOR → Permette di selezionare quali task eseguire in base a condizioni

TRIGGER RULE → Indica le condizioni per cui un task viene eseguito (attualmente) es. all failed → email

CROSS-DOWNSTREAM → Dipendenze tra liste di task

CHAIN → Sequenza di task o downstream task

CONCURRENCY → Number of tasks in execution

TASK CONCURRENCY → Number of task in creation

POOL → Definisce la coda di esecuzione del task e per gestire le priorità

DEPENDENCIES → Permette di dire a un task di dipendere dal risultato di esecuzione passata rispetto.

WAIT FOR DOWNSTREAM → Esegue solamente se lo stesso task è in successo nel precedente

ON-SUCCESS-CALLBACK → Esegue una funzione se non c'è stato errore

ON-FAILURE-CALLBACK → Esegue una funzione se c'è stato errore

ON-RETRY-CALLBACK → Esegue una funzione quando ritenta l'esecuzione

RETRY → Numero delle volte che il task viene rieseguito

RETRY-DELAY → Dopo quanto riprovare

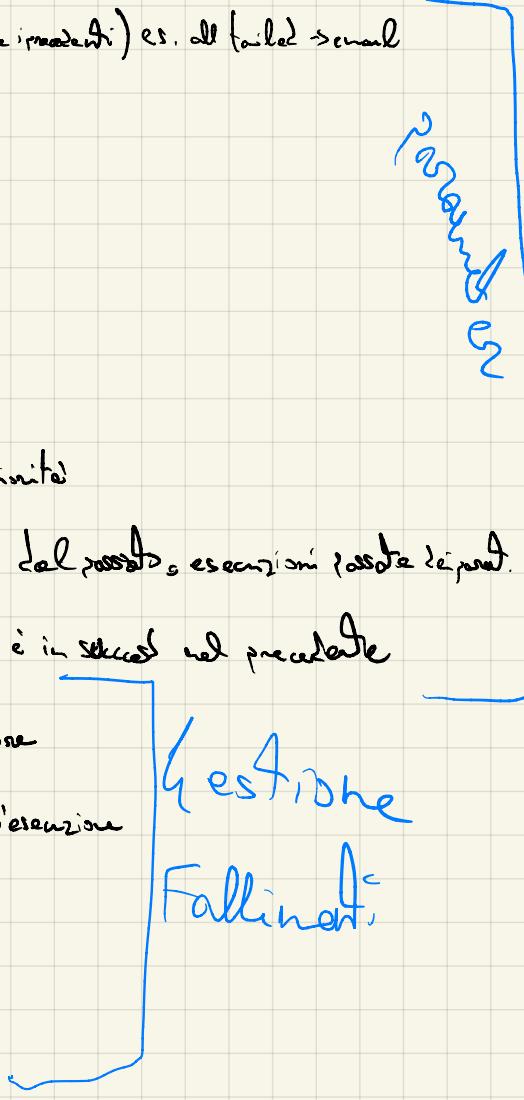
RETRY-EXPONENTIAL-BACKOFF → Crescita in modo esponenziale dell'attesa

MAX-RETRY-DELAY → Massimo tempo d'attesa

SLA → Dopo un tempo prefissato indica lo stato dei task (inviabili di oggi)

<https://academy.astronomer.io/astronomer-certification-apache-airflow-dag-authoring-preparation/899677>

EXTERNAL TASK → Aspetta Task di altri dag (usato in pipeline)



DAG dinamici