

Apache Airflow



Apache
Airflow

BIG DATA – BUSINESS INTELLIGENCE – MACHINE LEARNING

stratebi
open business intelligence



CONTENIDO

| | | |
|----|--|----|
| 1. | INTRODUCCIÓN | 4 |
| 2. | REQUISITOS DE INSTALACIÓN | 4 |
| 3. | INSTALACIÓN..... | 4 |
| 1. | PIP | 4 |
| 2. | DOCKER..... | 6 |
| 4. | INTERFAZ WEB..... | 7 |
| 1. | VISTA DAGs..... | 7 |
| 2. | DATA PROFILING | 14 |
| 3. | BROWSE..... | 15 |
| 5. | CONCEPTOS..... | 18 |
| 1. | DAGs..... | 18 |
| 2. | OPERADORES..... | 18 |
| 3. | TAREAS..... | 19 |
| 4. | CONEXIONES..... | 19 |
| 5. | HOOKS..... | 20 |
| 6. | VARIABLES..... | 20 |
| 7. | XCOMS | 21 |
| 8. | BRANCHING | 22 |
| 9. | DEPENDENCIAS..... | 22 |
| 6. | ORQUESTACIÓN DE TRABAJOS DE TALEND Y PDI..... | 23 |
| 1. | TALEND..... | 23 |
| 2. | PDI..... | 23 |
| 7. | EJEMPLO PIPELINE..... | 26 |
| 1. | INICIALIZACIÓN DEL DAG | 26 |
| 2. | CREACIÓN DE TABLAS CON POSTGRESOPERATOR..... | 27 |
| 3. | EJECUCIÓN DE TRABAJOS TALEND Y PDI CON BASHOPERATOR..... | 31 |
| 4. | EJECUCIÓN DE TRABAJO DE PDI CON KITCHENOPERATOR | 32 |
| 5. | DEPENDENCIAS..... | 35 |
| 6. | EJECUCIÓN DEL DAG | 35 |
| 8. | GOOGLE CLOUD COMPOSER..... | 37 |
| 1. | REQUISITOS | 37 |
| 2. | CREACIÓN DEL ENTORNO..... | 40 |
| 3. | EJECUCIÓN DE DAGs..... | 43 |

| | | |
|-----|---|----|
| 9. | INTEGRACIÓN Y DESPLIEGUE CONTINUO..... | 46 |
| 1. | PRUEBAS DE VALIDACIÓN..... | 48 |
| 2. | PRUEBAS DE DEFINICIÓN..... | 49 |
| 3. | PRUEBAS DE LÓGICA..... | 50 |
| 4. | PRUEBAS DE EXTREMO A EXTREMO..... | 51 |
| 5. | RECOMENDACIONES..... | 51 |
| 10. | CASOS DE USO Y BENEFICIOS..... | 52 |
| 11. | ANEXO..... | 53 |
| 1. | CÓDIGO DEL PRIMER EJEMPLO | 53 |
| 12. | POWER BI..... | 55 |
| 13. | TECNOLOGÍAS..... | 62 |
| 14. | INFORMACIÓN SOBRE STRATEBI..... | 64 |
| 15. | OTROS..... | 66 |
| 16. | EJEMPLOS DE DESARROLLOS ANALYTICS | 67 |

1. INTRODUCCIÓN

[Apache Airflow](#) es una herramienta de orquestación que permite crear, programar y monitorizar flujos de trabajo mediante programación en Python. Estos flujos de trabajos se representan como Grafos Acíclicos Dirigidos o DAGs (del inglés Directed Acyclic Graph) de tareas / procesos, es decir grafos donde los datos fluyen en una sola dirección entre los procesos, por lo que si algún trabajo X falla, los trabajos que dependen del trabajo X no se ejecutan.

2. REQUISITOS DE INSTALACIÓN

Para poder usar Apache Airflow, es necesario que la máquina cuente con una versión de Python igual o superior a 2.7.

Aunque, por defecto, Apache Airflow usa SQLite, por motivos de rendimiento en un entorno de producción se tiene que usar otra base de datos (p.e. PostgreSQL).

Para verificar que se tiene Python, se puede emplear el siguiente comando:

```
python --version
```

3. INSTALACIÓN

1. PIP

El directorio de inicio predeterminado es `~/airflow`. Se puede cambiar definiendo la variable de entorno `AIRFLOW_HOME`.

```
export AIRFLOW_HOME=~/airflow
```

Para instalar Apache Airflow, hay que ejecutar el siguiente comando:

```
pip install apache-airflow
```

También se puede instalar con paquetes extras:

```
pip install apache-airflow[paq1,paq2,...]
```

- all – todos los paquetes
- all_dbs – todos los paquetes necesarios para BBDD
- async – worker asíncrono par Gunicorn
- aws – Amazon Web Services
- azure – Microsoft Azure

- celery – CeleryExecutor
- cloudant – IBM Cloudant
- crypto – para cifrar las credenciales
- devel – herramientas de desarrollo
- devel_hadoop – lo anterior + dependencias Hadoop
- druid
- gcp – Google Cloud Platform
- github_enterprise
- google_auth
- hashicorp
- hdfs
- hive
- jdbc
- kerberos
- kubernetes
- ldap
- mssql
- mysql
- oracle
- password - autenticación
- postgres
- presto
- qds - Qubole Data Service
- rabbitmq
- redis
- samba
- slack
- ssh
- vertica

Después de instalar Apache Airflow, hay que inicializar la base de datos:

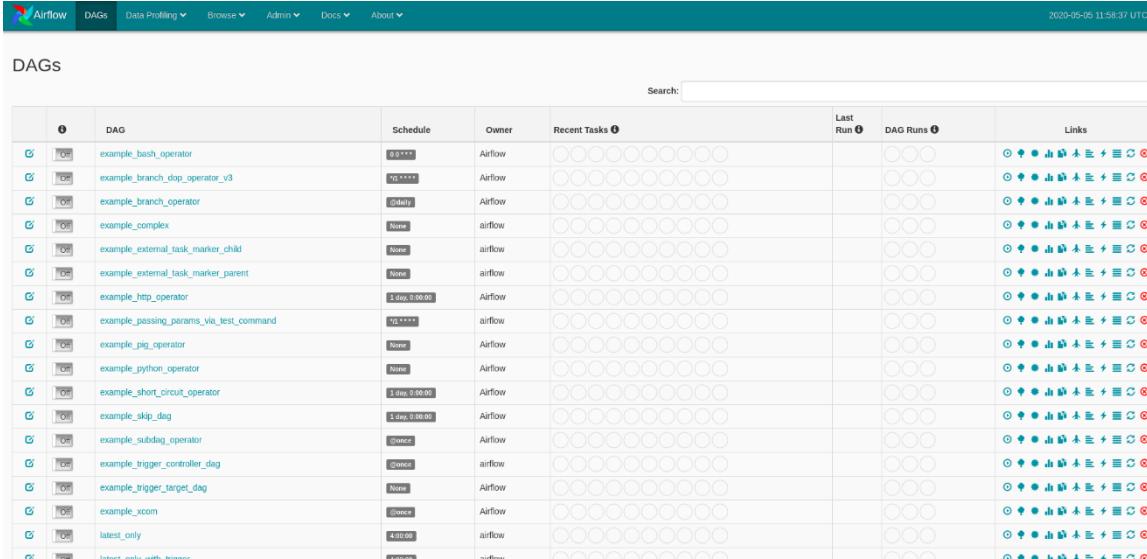
```
airflow initdb
```

Por último, hay que arrancar el servidor web (opcionalmente especificar el puerto) y el planificador:

```
# servidor web, puerto 8080  
airflow webserver -p 8080
```

```
# planificador  
airflow scheduler
```

Para acceder a la interfaz web hay que usar la IP de la maquina y el puerto (p.e. localhost:8080).



| | DAG | Schedule | Owner | Recent Tasks | Last Run | DAG Runs | Links |
|----|-------------------------------------|----------------|---------|--------------------|-------------------------|--------------------|-------|
| 1 | example_bash_operator | @0 * * * * | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 2 | example_branch_operator_v3 | @0 * * * * | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 3 | example_branch_operator | @daily | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 4 | example_complex | None | airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 5 | example_external_task_marker_child | None | airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 6 | example_external_task_marker_parent | None | airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 7 | example_http_operator | 1 day, 0:00:00 | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 8 | example_passing_params_via_xcom | @0 * * * * | airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 9 | example_pig_operator | None | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 10 | example_python_operator | None | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 11 | example_short_circuit_operator | 1 day, 0:00:00 | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 12 | example_skip_dag | 1 day, 0:00:00 | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 13 | example_subdag_operator | Once | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 14 | example_trigger_controller_dag | Once | airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 15 | example_trigger_target_dag | None | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 16 | example_xcom | Once | Airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 17 | latest_only | 4:00:00 | airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |
| 18 | Subdag example_dag | 4:00:00 | airflow | ██████████████████ | 2020-05-05 11:58:37 UTC | ██████████████████ | 🔗 |

Podemos modificar la configuración de Apache Airflow en el archivo `$(AIRFLOW_HOME)/airflow.cfg`

También es posible definir opciones de configuración usando variables de entorno con el patrón

`AIRFLOW_<sección>_<clave>`, donde `<sección>` es el nombre de la sección en el archivo de configuración, y `<clave>` es la opción de configuración.

2. Docker

Es posible ejecutar Apache Airflow sobre Docker usando la imagen [puckel/docker-airflow](#):

```
docker run -d -p 8080:8080 -v airflow_home:/usr/local/airflow puckel/docker-airflow
```

Podemos modificar la configuración de Apache Airflow editando el archivo de configuración `/usr/local/airflow/airflow.cfg` ejecutando el contenedor con las variables de entorno necesarias para la configuración deseada.

4. INTERFAZ WEB

1. Vista DAGs

Una lista de DAGs. Podemos activar los DAGs, ver el estado de las tareas y del DAG o ir a diferentes páginas (columna Links) para ver más información de un DAG en concreto.

| | DAG | Schedule | Owner | Recent Tasks | Last Run | DAG Runs | Links |
|-------------------------------------|--------------------------------|----------------|---------|---|------------------|---|-------|
| <input checked="" type="checkbox"/> | ejemplo_covid_pdi_talend | 1 day, 0:00:00 | airflow | 1 2 | 2020-05-20 00:00 | 1 2 | |
| <input checked="" type="checkbox"/> | example_bash_operator | 0 * * * * | Airflow | 1 2 | | 1 2 | |
| <input checked="" type="checkbox"/> | example_branch_dop_operator_v3 | *1 * * * * | Airflow | 1 2 | | 1 2 | |
| <input checked="" type="checkbox"/> | example_branch_operator | @daily | Airflow | 1 2 | | 1 2 | |
| <input checked="" type="checkbox"/> | example_complex | None | airflow | 1 2 | | 1 2 | |

La columna Links tiene varios botones:

- Trigger Dag – ejecuta el DAG
- Tree View – representación en formato árbol del DAG

DAG: ejemplo_covid_pdi_talend

schedule: 1 day, 0:00:00

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code

Trigger DAG Refresh Delete

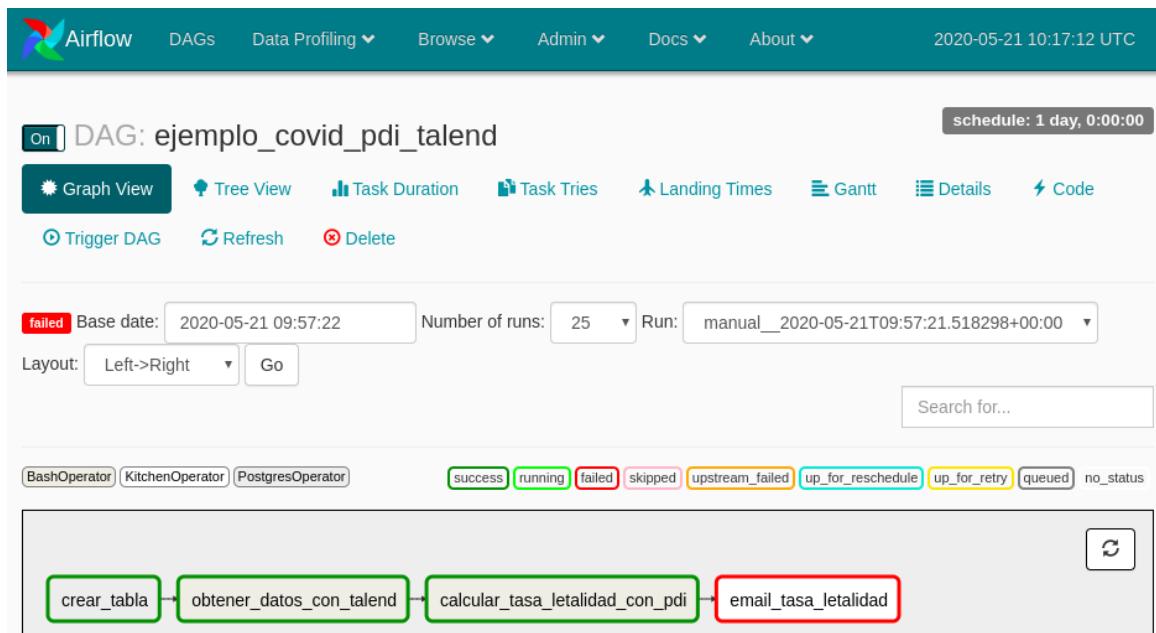
Base date: 2020-05-21 09:57:21 Number of runs: 25 Go

BashOperator KitchenOperator PostgresOperator success running failed skipped upstream_failed up_for_reschedule up_for_retry queued no_status

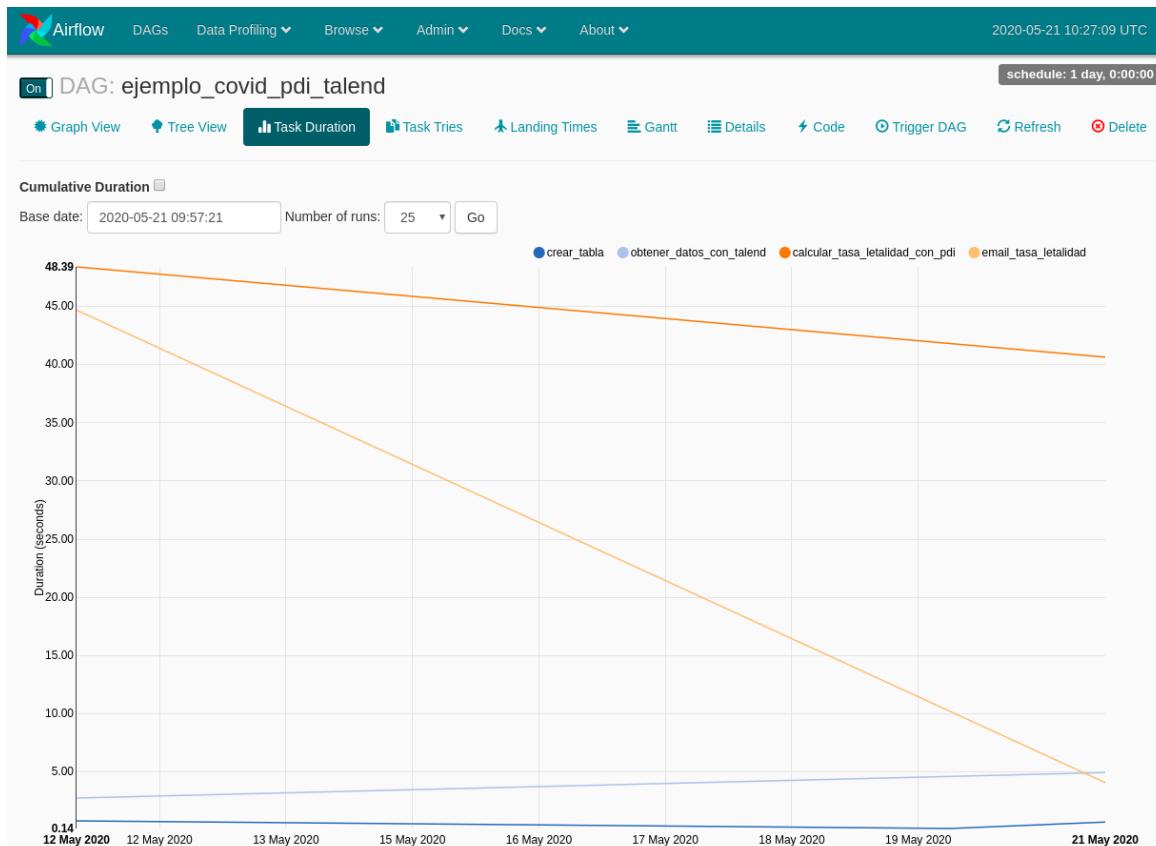
[DAG] -> crear_tabla -> obtener_datos_con_talend -> calcular_tasa_letalidad_con_pdi -> email_tasa_letalidad

May 17

- Graph View – visualizar las dependencias entre las tareas y los estados de las tareas.



- Task Duration – representación gráfica la duración de las tareas, útil para encontrar valores atípicos.

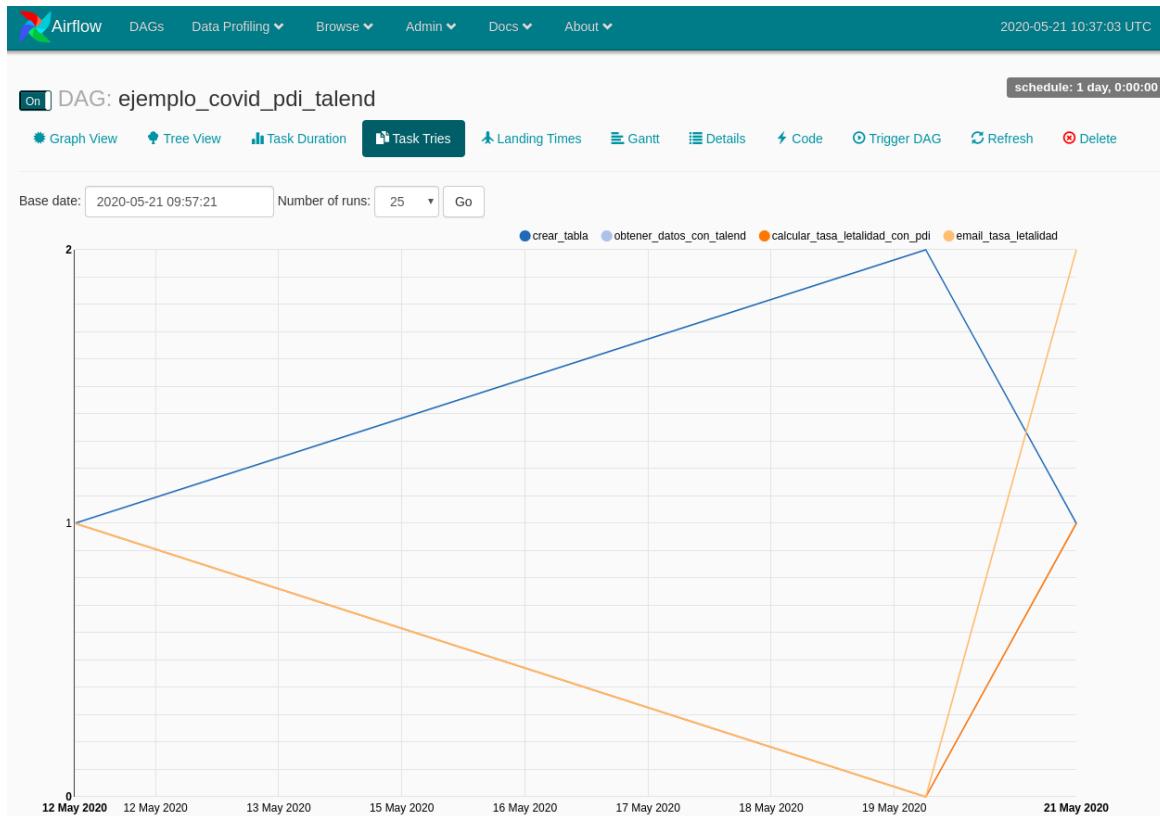


www.stratebi.com 91.788.34.10

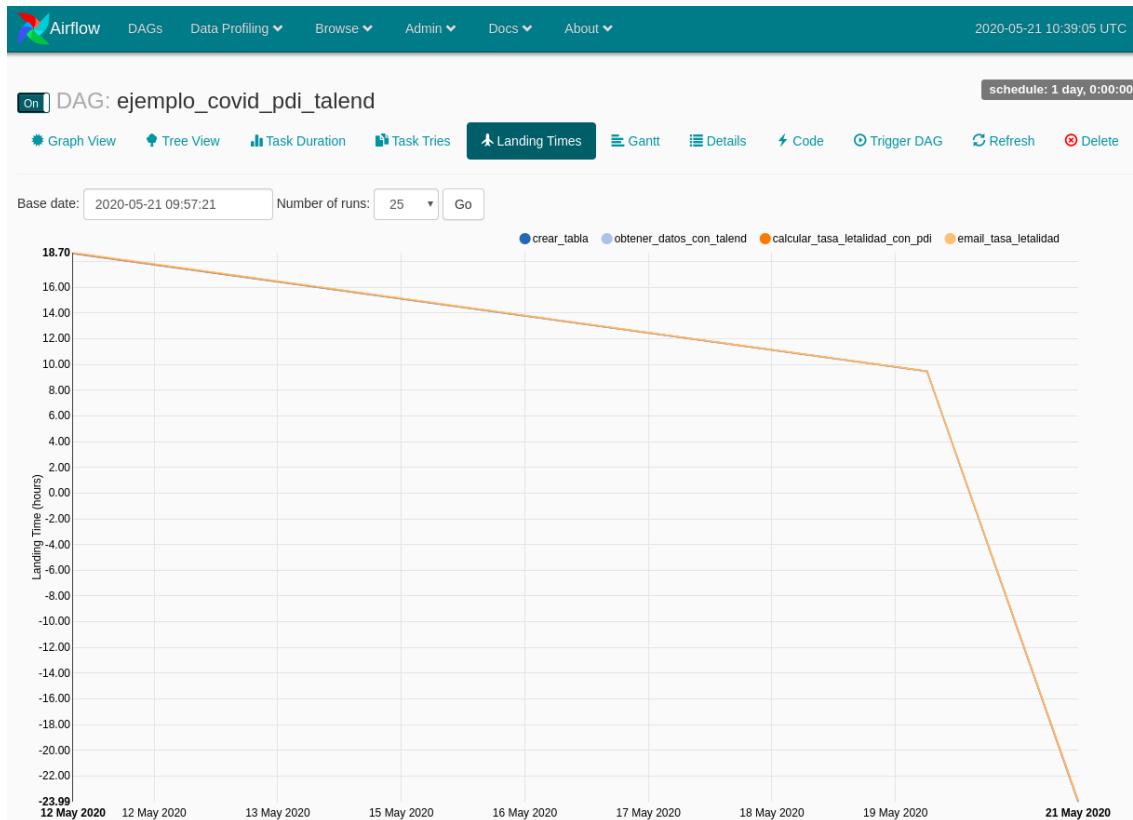
info@stratebi.com



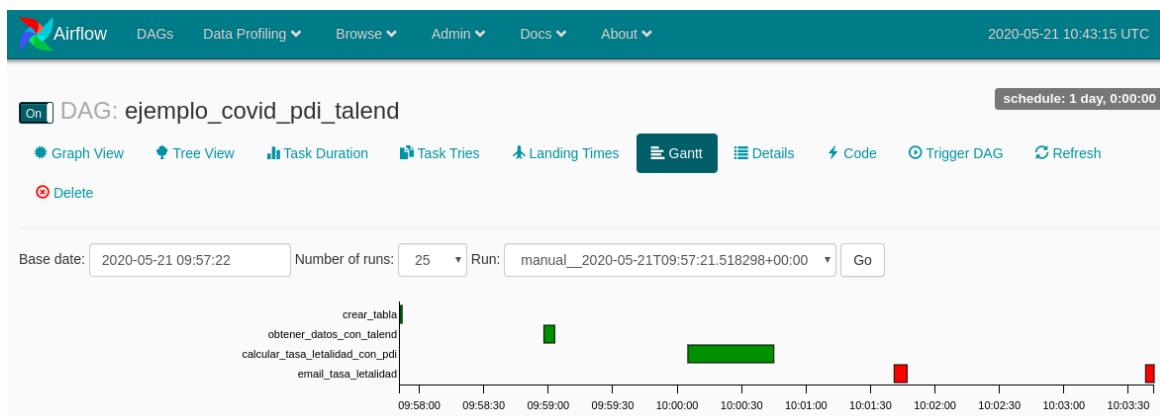
- Task Tries – representación gráfica de los reintentos de las tareas.



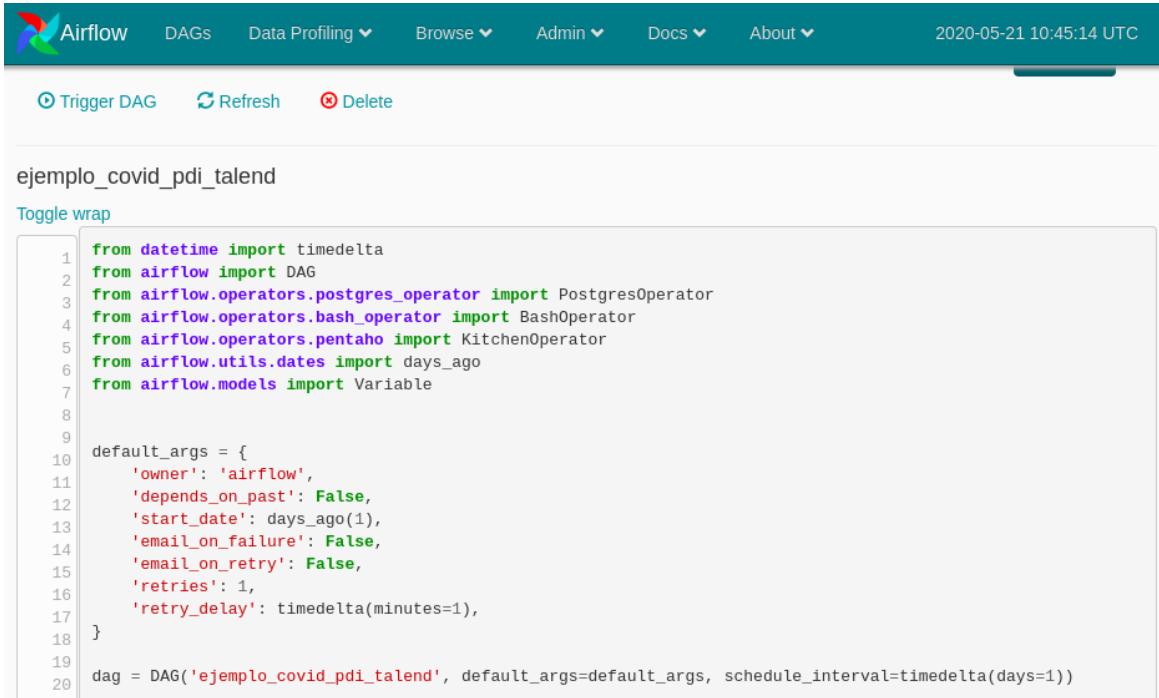
- Landing Times – comparación de las tareas a lo largo del tiempo.



- Gantt Chart – permite analizar la duración de las tareas, útil para identificar cuellos de botella.



- Code View – vista con el código fuente del DAG.



```

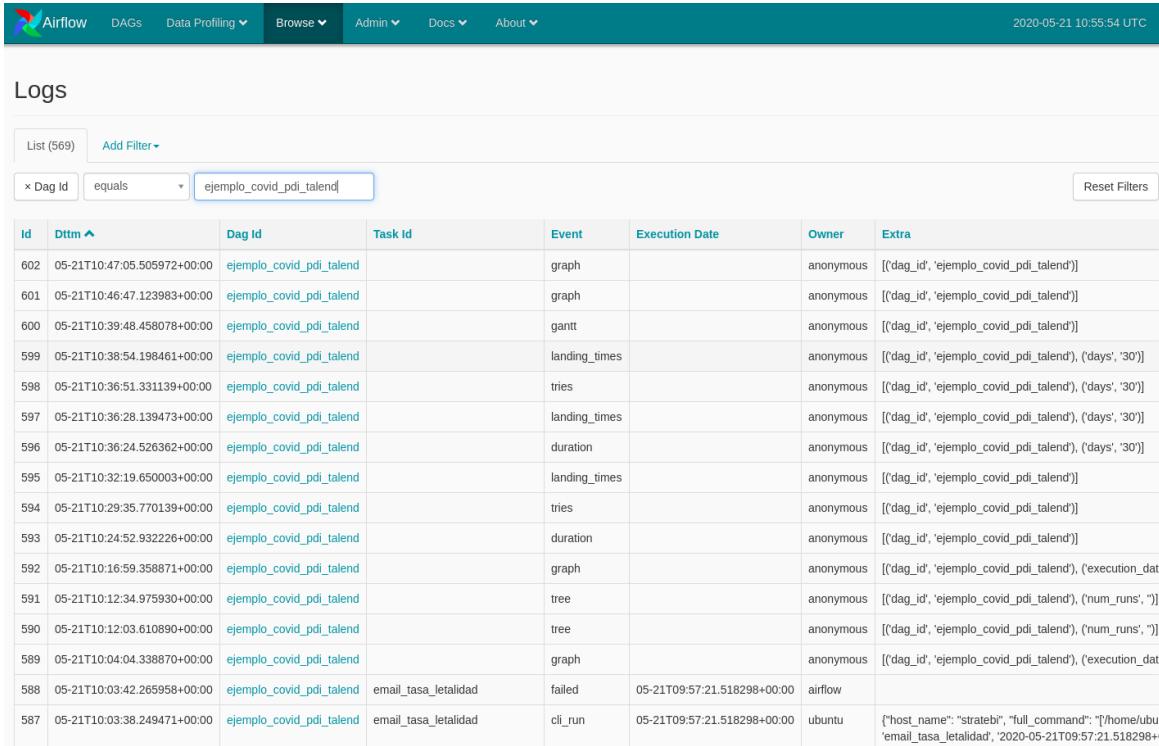
from datetime import timedelta
from airflow import DAG
from airflow.operators.postgres_operator import PostgresOperator
from airflow.operators.bash_operator import BashOperator
from airflow.operators.pentaho import KitchenOperator
from airflow.utils.dates import days_ago
from airflow.models import Variable

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=1),
}

dag = DAG('ejemplo_covid_pdi_talend', default_args=default_args, schedule_interval=timedelta(days=1))

```

- Logs – lista de los logs del DAG



| Logs | | | | | | | |
|------------|-----------------------------|--------------------------|----------------------|----------------|-----------------------------|---|---|
| List (569) | | Add Filter ▾ | | | | | |
| | Dag Id | Task Id | Event | Execution Date | Owner | Extra | Reset Filters |
| 602 | 05-21T10:47:05.505972+00:00 | ejemplo_covid_pdi_talend | graph | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}] | |
| 601 | 05-21T10:46:47.123983+00:00 | ejemplo_covid_pdi_talend | graph | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}] | |
| 600 | 05-21T10:39:48.458078+00:00 | ejemplo_covid_pdi_talend | gantt | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}] | |
| 599 | 05-21T10:38:54.198461+00:00 | ejemplo_covid_pdi_talend | landing_times | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"days": "30"}] | |
| 598 | 05-21T10:36:51.331139+00:00 | ejemplo_covid_pdi_talend | tries | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"days": "30"}] | |
| 597 | 05-21T10:36:28.139473+00:00 | ejemplo_covid_pdi_talend | landing_times | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"days": "30"}] | |
| 596 | 05-21T10:36:24.526362+00:00 | ejemplo_covid_pdi_talend | duration | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"days": "30"}] | |
| 595 | 05-21T10:32:19.650003+00:00 | ejemplo_covid_pdi_talend | landing_times | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}] | |
| 594 | 05-21T10:29:35.770139+00:00 | ejemplo_covid_pdi_talend | tries | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}] | |
| 593 | 05-21T10:24:52.932226+00:00 | ejemplo_covid_pdi_talend | duration | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}] | |
| 592 | 05-21T10:16:59.358871+00:00 | ejemplo_covid_pdi_talend | graph | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"execution_dat | |
| 591 | 05-21T10:12:34.975930+00:00 | ejemplo_covid_pdi_talend | tree | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"num_runs": "0"}] | |
| 590 | 05-21T10:12:03.610890+00:00 | ejemplo_covid_pdi_talend | tree | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"num_runs": "0"}] | |
| 589 | 05-21T10:04:04.338870+00:00 | ejemplo_covid_pdi_talend | graph | | anonymous | [{"dag_id": "ejemplo_covid_pdi_talend"}, {"execution_dat | |
| 588 | 05-21T10:03:42.265958+00:00 | ejemplo_covid_pdi_talend | email_tasa_letalidad | failed | 05-21T09:57:21.518298+00:00 | airflow | |
| 587 | 05-21T10:03:38.249471+00:00 | ejemplo_covid_pdi_talend | email_tasa_letalidad | cli_run | 05-21T09:57:21.518298+00:00 | ubuntu | {"host_name": "stratebi", "full_command": "/bin/echo 'email_tasa_letalidad' > /tmp/test.log", "start_time": "2020-05-21T09:57:21.518298+00:00", "end_time": "2020-05-21T09:57:21.518298+00:00", "status": "success", "log": "email_tasa_letalidad"} |

- Refresh – recarga la pagina

- Delete Dag – borra el DAG

2. Data Profiling

- Ad Hoc Query – permite ejecutar consultas SQL.

Screenshot of the Apache Airflow Data Profiling interface showing an Ad Hoc Query results page.

The top navigation bar includes links for Airflow, DAGs, Data Profiling (selected), Browse, Admin, Docs, and About, along with the timestamp 2020-05-21 11:16:19 UTC.

Ad Hoc Query

Query parameters: con_postgres, Run!, .csv

```
1 select * from public.covid
```

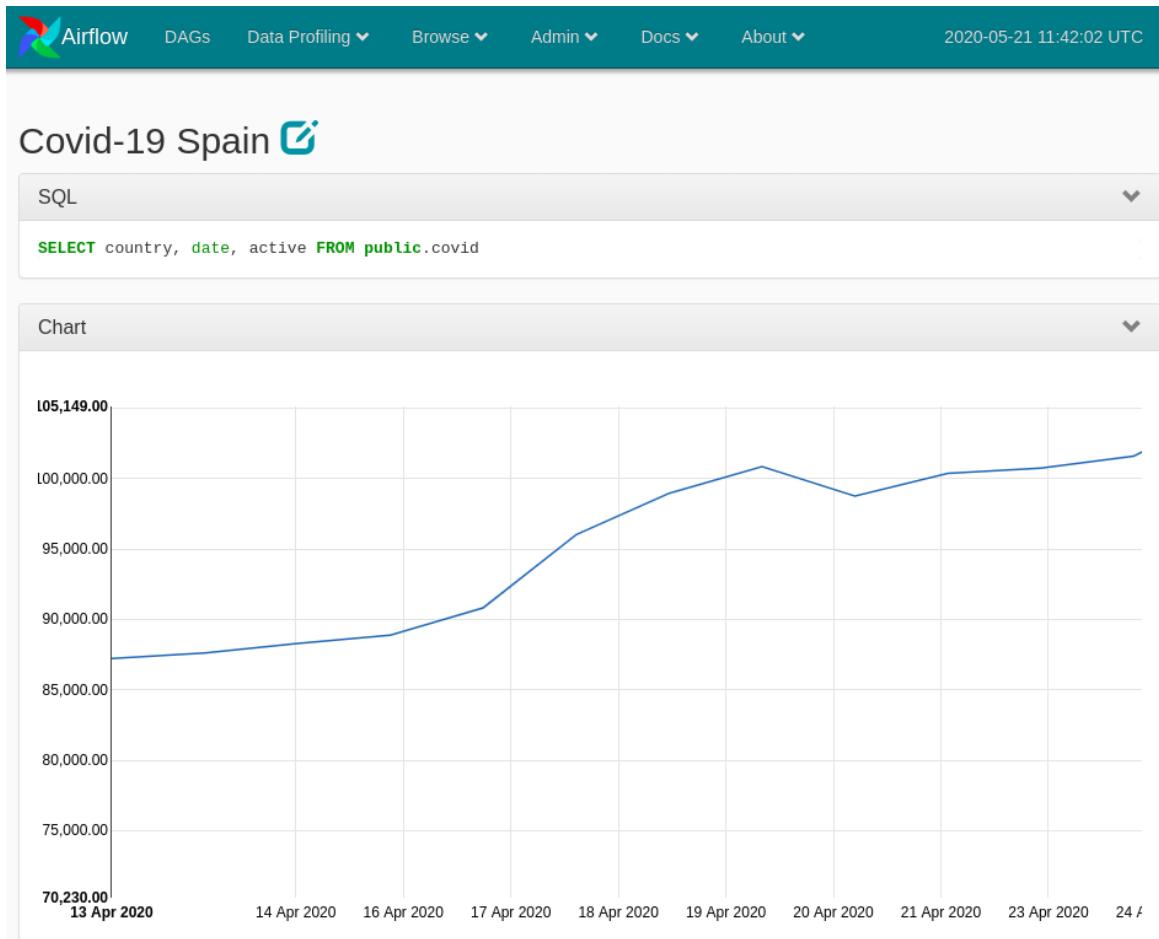
Show 100 entries Search:

| country | countrycode | province | city | citycode | lat | lon | confirmed | deaths | recovered | active | date |
|---------|-------------|----------|------|----------|-------|-------|-----------|--------|-----------|--------|------------|
| Spain | ES | | | | 40.46 | -3.75 | 166831 | 17209 | 62391 | 87231 | 2020-04-13 |
| Spain | ES | | | | 40.46 | -3.75 | 170099 | 17756 | 64727 | 87616 | 2020-04-14 |
| Spain | ES | | | | 40.46 | -3.75 | 174060 | 18255 | 67504 | 88301 | 2020-04-15 |
| Spain | ES | | | | 40.46 | -3.75 | 182816 | 19130 | 74797 | 88889 | 2020-04-16 |
| Spain | ES | | | | 40.46 | -3.75 | 184948 | 19315 | 74797 | 90836 | 2020-04-17 |
| Spain | ES | | | | 40.46 | -3.75 | 190839 | 20002 | 74797 | 96040 | 2020-04-18 |
| Spain | ES | | | | 40.46 | -3.75 | 194416 | 20639 | 74797 | 98980 | 2020-04-19 |
| Spain | ES | | | | 40.46 | -3.75 | 198674 | 20453 | 77357 | 100864 | 2020-04-20 |
| Spain | ES | | | | 40.46 | -3.75 | 200210 | 20852 | 80587 | 98771 | 2020-04-21 |
| Spain | ES | | | | 40.46 | -3.75 | 204178 | 21282 | 82514 | 100382 | 2020-04-22 |
| Spain | ES | | | | 40.46 | -3.75 | 208389 | 21717 | 85915 | 100757 | 2020-04-23 |
| Spain | ES | | | | 40.46 | -3.75 | 213024 | 22157 | 89250 | 101617 | 2020-04-24 |
| Spain | ES | | | | 40.46 | -3.75 | 219764 | 22524 | 92355 | 104885 | 2020-04-25 |
| Spain | ES | | | | 40.46 | -3.75 | 223759 | 22902 | 95708 | 105149 | 2020-04-26 |
| Spain | ES | | | | 40.46 | -3.75 | 226629 | 23190 | 117727 | 85712 | 2020-04-27 |
| Spain | ES | | | | 40.46 | -3.75 | 229422 | 23521 | 120832 | 85069 | 2020-04-28 |
| Spain | ES | | | | 40.46 | -3.75 | 217466 | 25264 | 118902 | 73300 | 2020-05-04 |
| Spain | ES | | | | 40.46 | -3.75 | 218011 | 25428 | 121343 | 71240 | 2020-05-05 |
| Spain | ES | | | | 40.46 | -3.75 | 219329 | 25613 | 123486 | 70230 | 2020-05-06 |

Showing 1 to 19 of 19 entries

Previous 1 Next

- Charts – permite crear visualizaciones de datos y gráficos.



3. Browse

- SLA Misses – en el caso de que una tarea exceda el tiempo esperado desde el inicio de la ejecución del DAG, esta queda registrada en SLA Misses.
- Task Instances – una vista histórica de las tareas.



Task Instances

| List (12) | | Add Filter ▾ | With selected ▾ | Search: dag_id, task_id, state | | | | | | | | | | 2020-05-21 13:57:16 UTC | |
|-----------|--------------|-------------------------|----------------------------------|--------------------------------|------------------|-----------------------------|-----------------------------|----------------|--------|-------------|----------|----------|---------|-------------------------|-------------|
| | State | Dag Id | Task Id | Execution Date | Operator | Start Date | End Date | Duration | Job Id | Hostname | Username | Priority | Weight | Queue | Queued Dttm |
| 0 | failed | ejemplo_covid_pd_talend | email_tasa_leletalidad | 05-21T09:57:21.518298+00:00 | KitchenOperator | 05-21T10:03:38.184184+00:00 | 05-21T10:03:42.265883+00:00 | 0:00:04.081699 | 43 | stratebi.bx | ubuntu | 1 | default | 05-21T10:03:3- | |
| 1 | success | ejemplo_covid_pd_talend | calcular_tasa_leletalidad_con_pd | 05-21T09:57:21.518298+00:00 | BashOperator | 05-21T10:00:05.065605+00:00 | 05-21T10:00:45.702959+00:00 | 0:00:40.637353 | 41 | stratebi.bx | ubuntu | 2 | default | 05-21T10:00:0- | |
| 2 | success | ejemplo_covid_pd_talend | obtener_datos_con_talend | 05-21T09:57:21.518298+00:00 | BashOperator | 05-21T09:58.851136+00:00 | 05-21T09:59:03.793896+00:00 | 0:00:04.942780 | 40 | stratebi.bx | ubuntu | 3 | default | 05-21T09:58:5- | |
| 3 | success | ejemplo_covid_pd_talend | crear_tabla | 05-21T09:57:21.518298+00:00 | PostgresOperator | 05-21T09:57:51.913331+00:00 | 05-21T09:57:52.603044+00:00 | 0:00:00.688673 | 39 | stratebi.bx | ubuntu | 4 | default | 05-21T09:57:4- | |
| 4 | failed | ejemplo_covid_pd_talend | crear_tabla | 05-20T00:00:00+00:00 | PostgresOperator | 05-21T09:27:58.433431+00:00 | 05-21T09:27:58.573873+00:00 | 0:00:00.140442 | 38 | stratebi.bx | ubuntu | 4 | default | 05-21T09:27:5- | |
| 5 | success | ejemplo_covid_pd_talend | email_tasa_leletalidad | 05-12T00:00:00+00:00 | KitchenOperator | 05-13T18:41:06.272541+00:00 | 05-13T18:41:50.950852+00:00 | 0:00:44.678311 | 35 | stratebi.bx | ubuntu | 1 | default | 05-13T18:41:0- | |
| 6 | success | ejemplo_covid_pd_talend | calcular_tasa_leletalidad_con_pd | 05-12T00:00:00+00:00 | BashOperator | 05-13T18:39:17.665354+00:00 | 05-13T18:40:08.059693+00:00 | 0:00:48.393339 | 34 | stratebi.bx | ubuntu | 2 | default | 05-13T18:39:1- | |
| 7 | success | ejemplo_covid_pd_talend | obtener_datos_con_talend | 05-12T00:00:00+00:00 | BashOperator | 05-13T18:38:13.951260+00:00 | 05-13T18:38:18.708892+00:00 | 0:00:02.757632 | 33 | stratebi.bx | ubuntu | 3 | default | 05-13T18:38:1- | |
| 8 | success | ejemplo_covid_pd_talend | crear_tabla | 05-12T00:00:00+00:00 | PostgresOperator | 05-13T18:37:05.829001+00:00 | 05-13T18:37:06.633226+00:00 | 0:00:00.004225 | 32 | stratebi.bx | ubuntu | 4 | default | 05-13T18:37:0- | |
| 9 | unknown/None | ejemplo_covid_pd_talend | email_tasa_leletalidad | 05-20T00:00:00+00:00 | KitchenOperator | 05-21T09:29:02.751885+00:00 | 05-21T09:29:02.751904+00:00 | | | | | 1 | default | | |
| 10 | unknown/None | ejemplo_covid_pd_talend | calcular_tasa_leletalidad_con_pd | 05-20T00:00:00+00:00 | BashOperator | 05-21T09:29:01.328863+00:00 | 05-21T09:29:01.328890+00:00 | | | | | 2 | default | | |
| 11 | unknown/None | ejemplo_covid_pd_talend | obtener_datos_con_talend | 05-20T00:00:00+00:00 | BashOperator | 05-21T09:28:59.920687+00:00 | 05-21T09:28:59.920701+00:00 | | | | | 3 | default | | |

- Logs – lista de logs.



Logs

| List (605) | | Add Filter ▾ | | | | | | | 2020-05-21 13:58:12 UTC | | | |
|------------|-----------------------------|-------------------------|------------------------|---------------|-----------------------------|-----------|--|--|-------------------------|--|--|--|
| ID | Dttm | Dag Id | Task Id | Event | Execution Date | Owner | Extra | | | | | |
| 605 | 05-21T13:31:48.526371+00:00 | ejemplo_covid_pd_talend | | graph | 05-21T09:57:21.518298+00:00 | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}, {"root": ""}, {"execution_date": "2020-05-21T09:57:21.518298+00:00"}] | | | | | |
| 604 | 05-21T13:31:40.432229+00:00 | ejemplo_covid_pd_talend | email_tasa_leletalidad | task | 05-21T09:57:21.518298+00:00 | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}, {"task_id": "email_tasa_leletalidad"}, {"execution_date": "2020-05-21T09:57:21.518298+00:00"}] | | | | | |
| 603 | 05-21T10:58:07.807484+00:00 | ejemplo_covid_pd_talend | | refresh | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}, {"refresh": true}] | | | | | |
| 602 | 05-21T10:47:05.505972+00:00 | ejemplo_covid_pd_talend | | graph | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}] | | | | | |
| 601 | 05-21T10:46:47.123983+00:00 | ejemplo_covid_pd_talend | | graph | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}] | | | | | |
| 600 | 05-21T10:39:48.458078+00:00 | ejemplo_covid_pd_talend | | garant | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}] | | | | | |
| 599 | 05-21T10:38:54.188461+00:00 | ejemplo_covid_pd_talend | | landing_times | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}, {"days": "30"}] | | | | | |
| 598 | 05-21T10:30:51.331139+00:00 | ejemplo_covid_pd_talend | | tries | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}, {"days": "30"}] | | | | | |
| 597 | 05-21T10:36:28.139473+00:00 | ejemplo_covid_pd_talend | | landing_times | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}, {"days": "30"}] | | | | | |
| 596 | 05-21T10:36:24.526362+00:00 | ejemplo_covid_pd_talend | | duration | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}, {"days": "30"}] | | | | | |
| 595 | 05-21T10:32:19.650003+00:00 | ejemplo_covid_pd_talend | | landing_times | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}] | | | | | |
| 594 | 05-21T10:29:35.770139+00:00 | ejemplo_covid_pd_talend | | tries | | anonymous | [{"dag_id": "ejemplo_covid_pd_talend"}] | | | | | |

- Jobs – lista de los flujos de trabajo



Jobs

| List (17) | | Add Filter ▾ | | | | | | | 2020-05-21 13:58:49 UTC | | |
|-----------|-------------------------|--------------|--------------|-----------------------------|-----------------------------|-----------------------------|----------------|-------------|-------------------------|--|--|
| ID | Dag Id | State | Job Type | Start Date | End Date | Latest Heartbeat | Executor Class | Hostname | Username | | |
| 43 | ejemplo_covid_pd_talend | success | LocalTaskJob | 05-21T10:03:38.146376+00:00 | 05-21T10:03:48.160564+00:00 | 05-21T10:03:43.156034+00:00 | NoneType | stratebi.bx | ubuntu | | |
| 42 | ejemplo_covid_pd_talend | success | LocalTaskJob | 05-21T10:01:40.955680+00:00 | 05-21T10:01:50.970988+00:00 | 05-21T10:01:45.965605+00:00 | NoneType | stratebi.bx | ubuntu | | |
| 41 | ejemplo_covid_pd_talend | success | LocalTaskJob | 05-21T10:00:04.940359+00:00 | 05-21T10:00:46.047502+00:00 | 05-21T10:00:41.041835+00:00 | NoneType | stratebi.bx | ubuntu | | |
| 40 | ejemplo_covid_pd_talend | success | LocalTaskJob | 05-21T09:58.58.674400+00:00 | 05-21T09:59.08.687422+00:00 | 05-21T09:59:03.682497+00:00 | NoneType | stratebi.bx | ubuntu | | |
| 39 | ejemplo_covid_pd_talend | success | LocalTaskJob | 05-21T09:57.51.840754+00:00 | 05-21T09:58.01.854762+00:00 | 05-21T09:57.56.849053+00:00 | NoneType | stratebi.bx | ubuntu | | |
| 38 | ejemplo_covid_pd_talend | success | LocalTaskJob | 05-21T09:27.58.391313+00:00 | 05-21T09:28.06.403620+00:00 | 05-21T09:28:03.403625+00:00 | NoneType | stratebi.bx | ubuntu | | |
| 37 | ejemplo_covid_pd_talend | success | LocalTaskJob | 05-21T09:26.47.599245+00:00 | 05-21T09:26.51.64811+00:00 | 05-21T09:26:52.610090+00:00 | NoneType | stratebi.bx | ubuntu | | |

- DAG Runs – lista con información de ejecución de los DAGS

The screenshot shows the Apache Airflow interface for managing DAG runs. At the top, there's a navigation bar with links for Airflow, DAGs, Data Profiling, Browse, Admin, Docs, and About. The current page is 'Browse'. On the right, it shows the date '2020-05-21 13:59:52 UTC'. Below the navigation is a search bar with placeholder text 'Search: dag_id, state, run_id'. A button labeled 'Create' is also present. The main content area is titled 'Dag Runs' and displays a table with 10 rows. The columns are: State, Dag ID, Execution Date, Run ID, and External Trigger. The 'State' column uses color-coded icons: red for failed, green for success, and grey for other states. The 'Dag ID' column lists 'ejemplo_covid_pdi_talend' repeated 10 times. The 'Execution Date' column shows various dates from May 17, 2020, to May 21, 2020. The 'Run ID' column contains identifiers like 'manual_...' and 'scheduled_...'. The 'External Trigger' column has small circular icons.

| List (10) | Create | Add Filter | With selected | Search: dag_id, state, run_id |
|--------------------------|-------------------------------------|------------|--------------------------|-------------------------------|
| <input type="checkbox"/> | | | | |
| <input type="checkbox"/> | ✗ | failed | ejemplo_covid_pdi_talend | 05-21T09:57:21.518Z98+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | failed | ejemplo_covid_pdi_talend | 05-20T00:00:00+00:00 |
| <input type="checkbox"/> | <input checked="" type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-19T00:00:00+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-18T00:00:00+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-17T00:00:00+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-16T00:00:00+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-15T00:00:00+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-14T00:00:00+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-13T00:00:00+00:00 |
| <input type="checkbox"/> | <input type="checkbox"/> | success | ejemplo_covid_pdi_talend | 05-12T00:00:00+00:00 |

5. CONCEPTOS

1. DAGs

El DAG describe el orden de las tareas, cómo se lleva a cabo el flujo de trabajo y cómo se manejan los fallos. Por ejemplo, en un DAG se puede especificar que una tarea B depende de otra tarea A, que el flujo de trabajo se debe ejecutar cada día a las 12h y que la tarea A puede fallar hasta 5 veces.

Ai DAG se le puede pasar un diccionario de argumentos por defecto, *default_args*, y este a su vez los aplicara a todos sus operadores. Ejemplo de creación de un DAG:

```
from datetime import datetime
from airflow import DAG

default_args = {
    'start_date': datetime(2020, 5, 13),
    'owner': 'airflow'
}

dag = DAG('my_dag', default_args=default_args)
```

2. Operadores

El operador sabe cómo realizar una tarea y tiene todas las "herramientas" para realizarla. En general, un operador no tiene que compartir recursos con otros operadores, pero se puede hacer usando XCom.

Apache Airflow proporciona operadores para muchas tareas comunes, por ejemplo:

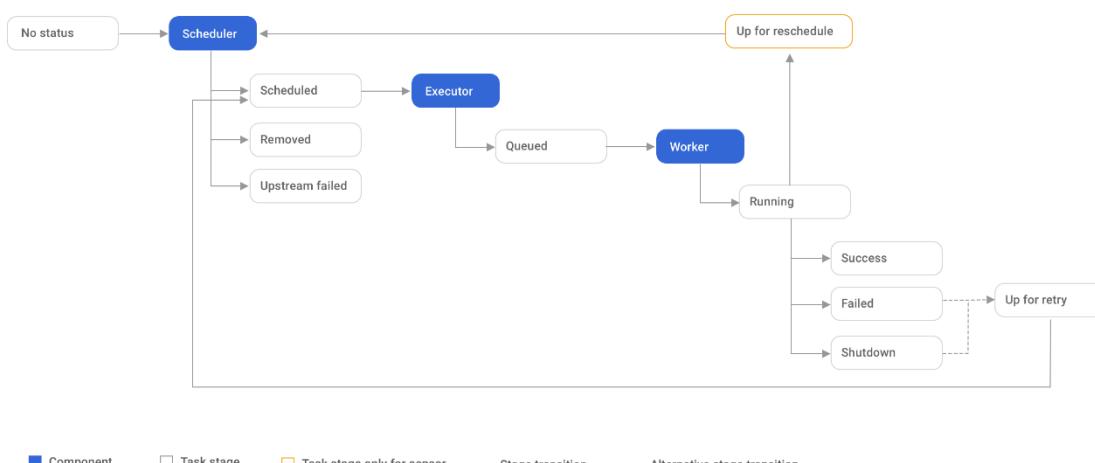
- BashOperator – sabe cómo ejecutar comandos en Bash
- PythonOperator – ejecuta código Python
- PostgresOperator – sabe cómo conectar con una base de datos Postgres y ejecutar SQL
- EmailOperator – envía un correo
- SimpleHTTPOperator – envía una petición HTTP

3. Tareas

Una tarea es el trabajo realizado por un operador. Pasa por varias etapas / estados:

- Success – la tarea ha terminado con éxito
- Running – la tarea se está ejecutando
- Failed – la tarea ha fallado
- Skipped – se ha omitido la tarea
- Rescheduled – se ha reprogramado la tarea
- Retry – se puede reintentar ejecutar la tarea
- Queued - se ha enviado la tarea al ejecutor para que se ejecute
- No status – sin estado

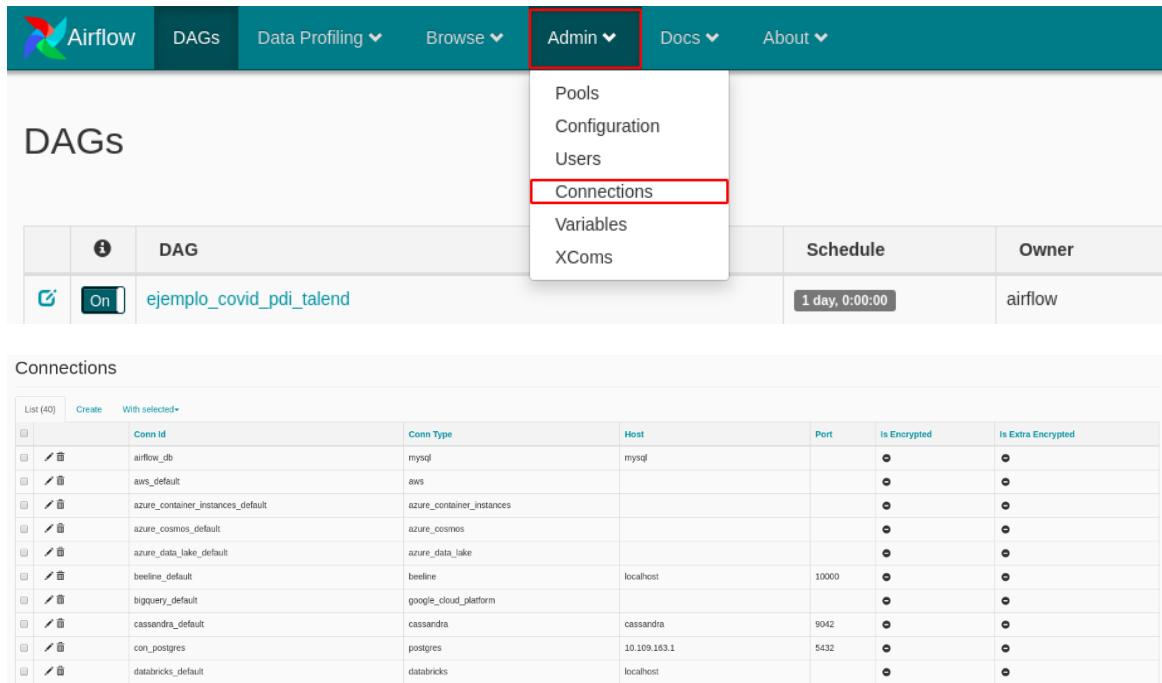
El ciclo de vida completo de una tarea es este:



4. Conexiones

Una conexión representa la información necesaria para conectar con sistemas externos, por ejemplo: una base de datos, Amazon Web Services...

Para configurar conexiones hay que acceder a la interfaz web de Apache Airflow y pinchar en *Admin > Connections*.



The screenshot shows the Apache Airflow web interface. At the top, there is a navigation bar with links: Airflow, DAGs, Data Profiling, Browse, Admin (which is highlighted with a red box), Docs, and About. Below the navigation bar, there is a section titled "DAGs" with a table. The table has columns: DAG, Schedule, and Owner. One row in the table is highlighted with a red box and contains the values "ejemplo_covid_pdi_talend", "1 day, 0:00:00", and "airflow". To the right of the table, a dropdown menu is open under the Admin link, showing options: Pools, Configuration, Users, Connections (which is highlighted with a red box), Variables, and XComs.

Connections

| List (40) | | | | | | |
|-----------|-----------------------------------|---------------------------|--------------|-------|--------------|--------------------|
| | Conn Id | Conn Type | Host | Port | Is Encrypted | Is Extra Encrypted |
| ✓ | airflow_db | mysql | mysql | | ● | ● |
| ✓ | aws_default | aws | | | ● | ● |
| ✓ | azure_container_instances_default | azure_container_instances | | | ● | ● |
| ✓ | azure_cosmos_default | azure_cosmos | | | ● | ● |
| ✓ | azure_data_lake_default | azure_data_lake | | | ● | ● |
| ✓ | beeline_default | beeline | localhost | 10000 | ● | ● |
| ✓ | bq_default | google_cloud_platform | | | ● | ● |
| ✓ | cassandra_default | cassandra | cassandra | 9042 | ● | ● |
| ✓ | con_postgres | postgres | 10.109.183.1 | 5432 | ● | ● |
| ✓ | databricks_default | databricks | localhost | | ● | ● |

5. Hooks

Los hooks son interfaces a plataformas externas o bases de datos. Usan las conexiones para obtener la información de autenticación. En general implementan una interfaz común (los varios hooks son muy similares) y actúan como un bloque de construcción para los operadores.

6. Variables

Son parecidas a las variables de entorno. Las variables pueden almacenar información arbitraria y se pueden usar en las tareas. Se pueden enumerar, crear, actualizar y eliminar con código, línea de comandos o interfaz web.

Para acceder al listado de variables hay que acceder a la interfaz web y pinchar en *Admin > Variables*.

The screenshot shows the Airflow Admin interface. The top navigation bar includes links for Airflow, DAGs, Data Profiling, Browse, Admin (which is highlighted with a red box), Docs, and About. A dropdown menu from the Admin link contains options: Pools, Configuration, Users, Connections, Variables (which is highlighted with a red box), and XComs. Below the navigation, there's a file upload section with 'Choose File' and 'No file chosen', and a 'Import Variable' button. A search bar at the top right is set to 'Search: key, val, is_encrypted'. The main content area has tabs for List, Create, Add Filter, and With selected. A table with three columns (Key, Val, Is Encrypted) is shown, with a note below it stating 'There are no items in the table.'

Podemos obtener el valor de una variable con `Variable.get('nombre_var')`

```
from airflow.models import Variable

foo = Variable.get('foo')
bar = Variable.get('bar', deserialize_json=True)
baz = Variable.get('baz', default_var='baz test')
```

Si la variable es de tipo JSON, se puede deserializar especificando el argumento `deserialize_json=True`. También se puede especificar un valor por defecto por si la variable no existe.

7. XComs

Permite el envío de mensajes entre tareas para dar soporte a la compartición de recursos entre los operadores. Una tarea puede enviar XComs con la función `xcom_push(...)` o recibir XComs con la función `xcom_pull(...)`.

8. Branching

Permite seguir un camino determinado en función de una condición arbitraria. Hay varias formas de hacer esto, pero la más usada es el branching mediante el operador *BranchPythonOperator*. Se le pasa un argumento *python_callable*, que debe ser una función que devuelve una tarea o una lista de tareas.

```
from airflow.operators.python import BranchPythonOperator

def branch_func(**kwargs):
    ti = kwargs['ti']
    xcom_value = int(ti.xcom_pull(task_ids='start_task'))
    if xcom_value >= 5:
        return 'tarea1'
    else:
        return 'tarea2'

branch = BranchPythonOperator(
    task_id='branch',
    provide_context=True,
    python_callable=branch_func,
    dag=dag
)
```

9. Dependencias

Las dependencias entre tareas se definen usando los operadores >> y <<.

Ejemplos:

```
tarea1 >> tarea2 # tarea2 depende de la tarea1
tarea1 >> [tarea2, tarea3] # tarea2 y tarea3 dependen de la tarea1
Tarea1 >> tarea2 << tarea3 # tarea2 depende de las tareas 1 y 3
```

6. ORQUESTACIÓN DE TRABAJOS DE TALEND Y PDI

1. Talend

Puesto que podemos compilar los trabajos y ejecutarlos en cualquier máquina que tenga Java, podemos usar *BashOperator* para la ejecución de trabajos de Talend especificando los siguientes argumentos:

- task_id – id de la tarea
- bash_command – el comando para ejecutar el trabajo
- dag – instancia DAG

```
tarea_talend = BashOperator(  
    task_id='tarea_talend',  
    bash_command='/path/to/tarea_talend_run.sh ',  
    dag=dag  
)
```

2. PDI

Para ejecutar trabajos locales de PDI podemos usar *BashOperator*. Igual que en el punto anterior debemos especificar los argumentos: *task_id*, *bash_command* y *dag*.

```
tarea_pdi= BashOperator(  
    task_id='tarea_pdi',  
    bash_command='kitchen.sh -file="/path/to/job.kjb" ',  
    dag=dag  
)
```

También existe un plugin de Pentaho para Apache Airflow que permite ejecutar trabajos o transformaciones guardadas en un repositorio de Pentaho.

Para instalar el plugin hay que ejecutar el siguiente comando:

```
pip install airflow-pentaho-plugin
```

El plugin consiste en cuatro operadores:

- CarteJobOperator – permite ejecutar trabajos en servidores esclavos.
- KitchenOperator – permite ejecutar trabajos.

- `CarteTransOperator` – permite ejecutar transformaciones en servidores esclavos.
- `PanOperator` – permite ejecutar transformaciones.

Para que se pueda utilizar hay que configurar una conexión, pinchando en *Admin > Connections*. Es obligatorio especificar:

- `Conn id` – el id de la conexión.
- `Login` – usuario de Pentaho.
- `Password` – la contraseña del usuario.
- `Extra` – un JSON con los siguientes campos:
 - `pentaho_home` – la ruta de PDI (opcional si no hay que ejecutar con Kettle / Pan).
 - `rep` – el nombre del repositorio.
 - `carte_username` – usuario `Carte` (opcional si no hay que ejecutar con `Carte`).
 - `carte_password` – contraseña del usuario de `Carte` (opcional si no hay que ejecutar con `Carte`).

Todos los operadores admiten los siguientes argumentos:

- `conn_id` – el id de la conexión.
- `task_id` – el id de la tarea.
- `params` – un diccionario de parámetros
- `dag` – la instancia del DAG

Además, hay argumentos específicos:

- `KitchenOperator`:
 - `job` – el nombre del trabajo.
 - `directory` – el directorio (ruta absoluta) del trabajo.
- `PanOperator`:
 - `trans` – el nombre de la transformación.
 - `directory` – el directorio (ruta absoluta) de la transformación.

- `CarteJobOperator`:
 - `job` – la ruta absoluta del trabajo.
- `CarteTransOperator`:
 - `trans` – la ruta absoluta de la transformación.

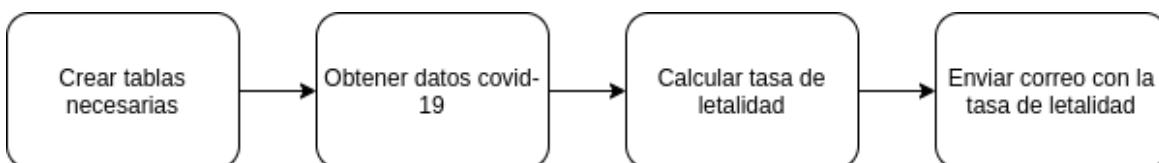
Ejemplo *CarteJobOperator*:

```
from airflow.operators.pentaho import CarteJobOperator

carte_job = CarteJobOperator(
    conn_id='pdi_default',
    task_id='carte_job',
    job='/home/bi/carte_job',
    params={'date': '{{ ds }}'},
    dag=dag
)
```

7. EJEMPLO PIPELINE

En este ejemplo obtenemos los datos de la COVID-19 usando Talend, calculamos la tasa de letalidad con PDI y enviamos un correo con la tasa de letalidad del ultimo día usando PDI y un repositorio de Pentaho.



1. Inicialización del DAG

Importamos los paquetes necesarios e inicializamos un DAG con los siguientes argumentos:

- dag_id
- default_args:
 - owner – airflow, dueño del DAG.
 - depends_on_past – False, para que se ejecute la tarea independientemente del estado de la misma en el pasado.
 - start_date – el día anterior, para que se ejecute una vez antes de lo programado (concepto de Catchup).

- email_on_failure, email_on_retry – False, para que no envíe correo en caso de fallos o reintentos.
- retries – 1 reintento.
- retry_delay – 1 minuto retraso entre reintentos.

```
from datetime import timedelta
from airflow import DAG
from airflow.utils.dates import days_ago

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=1),}

dag = DAG('ejemplo_covid_pdi_talend',
           default_args=default_args,
           schedule_interval=timedelta(days=1))
```

2. Creación de tablas con PostgresOperator

Para que el ejemplo funcione, es necesario tener una base de datos de Postgres llamada *ejemplo*. Antes de crear la tarea, tenemos que definir una conexión a la base de datos yendo a la interfaz web de Apache Airflow y luego pinchando en *Admin > Connections > Create*.

Warning: Connection passwords are stored in plaintext until you enable "Encryption at Rest" in the "Encryption" section of the "Configuration" page.

Connections

| | Conn Id | Conn Type |
|--------------------------|------------|-----------|
| <input type="checkbox"/> | airflow_db | mysql |

Creamos la conexión.

Connection [create]

List Create

| | |
|-----------|--------------|
| Conn Id * | con_postgres |
| Conn Type | Postgres |
| Host | 10.109.163.1 |
| Schema | public |
| Login | airflow |
| Password | ***** |
| Port | 5432 |
| Extra | |

Save Save and Add Another Save and Continue Editing Cancel

En el archivo del código fuente importamos *PostgresOperator* lo inicializamos con los siguientes argumentos:

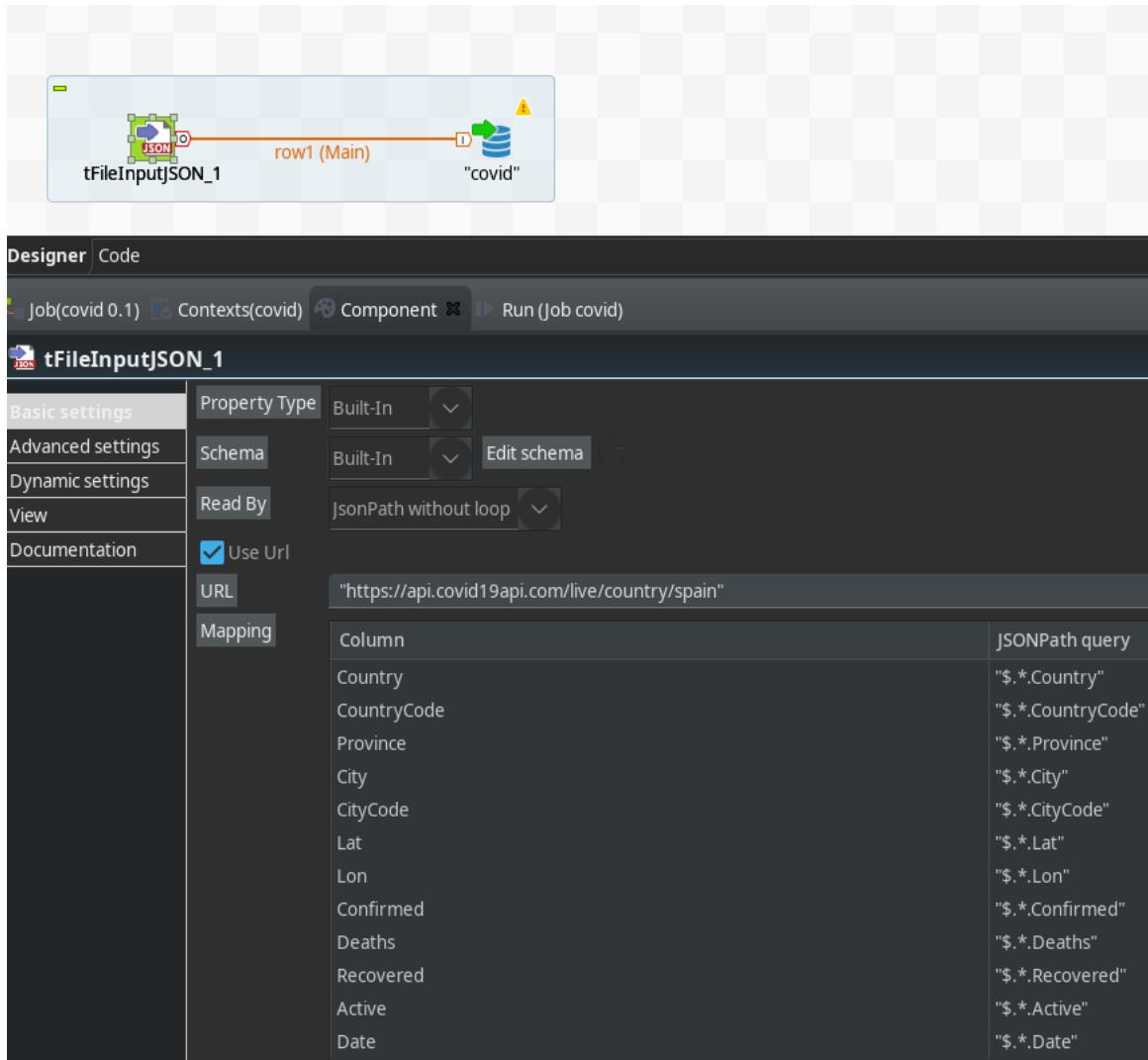
- task_id – el id de la tarea.
- postgres_conn_id – el id de la conexión definida en el paso anterior.
- database – el nombre de la base de datos.
- sql – las sentencias sql para crear las tablas necesarias.
- dag – el dag definido anteriormente.

```
from airflow.operators.postgres_operator import PostgresOperator

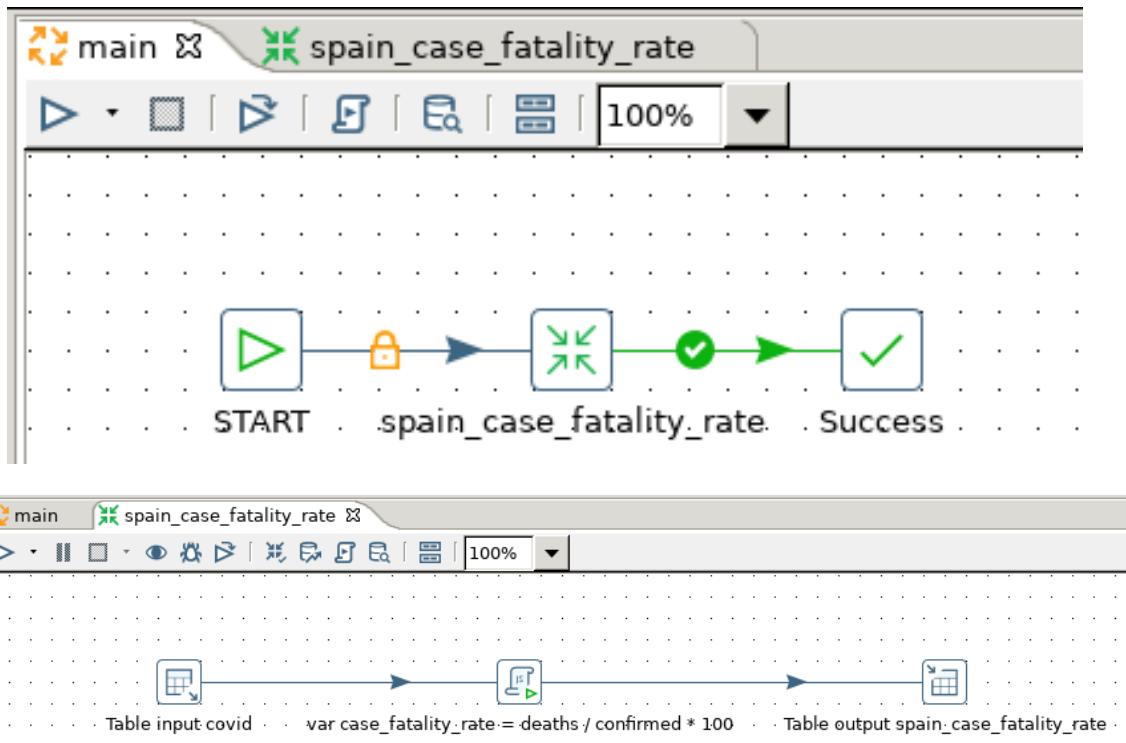
crear_tabla = PostgresOperator(
    task_id='crear_tabla',
    postgres_conn_id='con_postgres',
    database='ejemplo',
    sql="""
CREATE TABLE IF NOT EXISTS public.covid
(
    Country text,
    CountryCode text,
    Province text,
    City text,
    CityCode text,
    Lat double precision,
    Lon double precision,
    Confirmed integer,
    Deaths integer,
    Recovered integer,
    Active integer,
    Date date
);
TRUNCATE TABLE public.covid;
CREATE TABLE IF NOT EXISTS public.spain_case_fatality_rate
(
    date timestamp,
    case_fatality_rate numeric(18,2)
);
TRUNCATE TABLE public.spain_case_fatality_rate;
"""
,
dag=dag,
)
```

3. Ejecución de trabajos Talend y PDI con BashOperator

En el trabajo de Talend nos conectamos a una API (<https://api.covid19api.com/live/country/spain>) y guardamos el resultado en la tabla *covid* de la base de datos *ejemplo*.



En el trabajo de PDI ejecutamos una transformación que calcula la tasa de fatalidad y la insertamos en la tabla *public.spain_case_fatality_rate*.



En el archivo del código fuente importamos *BashOperator* creamos una tarea para la ejecución del trabajo de Talend y otra para la ejecución del trabajo de PDI. Igual que en las tareas anteriores le pasamos los argumentos *task_id* y *dag* y un argumento *bash_command* para ejecutar los comandos de bash.

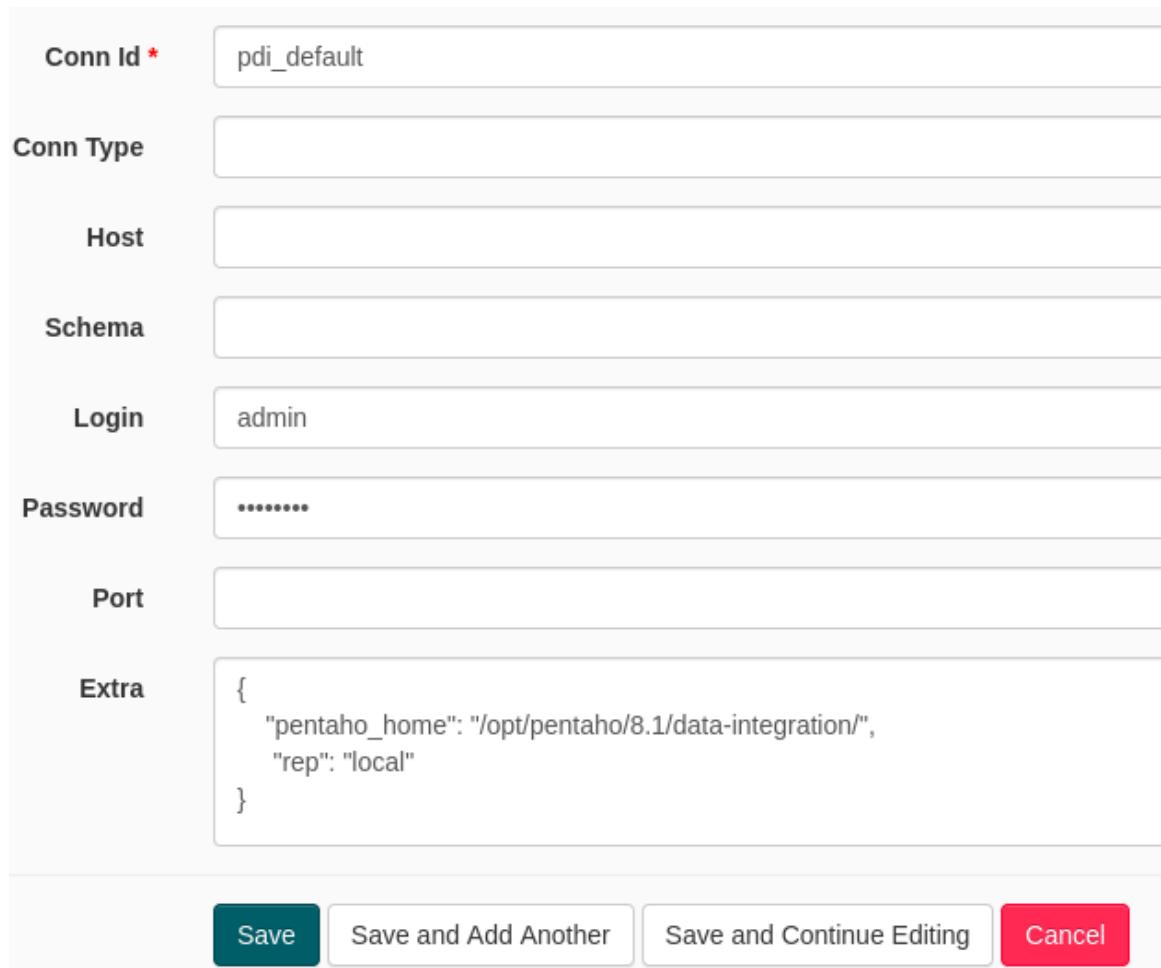
```
from airflow.operators.bash_operator import BashOperator

obtener_datos_con_talend = BashOperator(
    task_id='obtener_datos_con_talend',
    bash_command='/home/ubuntu/talend/covid/covid_run.sh ',
    dag=dag,
)
calcular_tasa_letalidad_con_pdi = BashOperator(
    task_id='calcular_tasa_lethalidad_con_pdi',
    bash_command='/opt/pentaho/8.1/data-integration/kitchen.sh -
level="Basic" -file="/home/ubuntu/pdi/calc_case_fatality_rate
/main.kjb" ',
    dag=dag,
)
```

4. Ejecución de trabajo de PDI con KitchenOperator

En este trabajo obtenemos la tasa de fatalidad del ultimo día y enviamos un correo. El trabajo admite un parámetro *EMAIL*.

Para que funcione es necesario crear una conexión con PDI pinchando en *Admin > Connections > Create*.



| | |
|-----------|---|
| Conn Id * | pdi_default |
| Conn Type | |
| Host | |
| Schema | |
| Login | admin |
| Password | ***** |
| Port | |
| Extra | { "pentaho_home": "/opt/pentaho/8.1/data-integration/", "rep": "local" } |

Save **Save and Add Another** **Save and Continue Editing** **Cancel**

También hay que crear la variable *EMAIL* pinchando en *Admin > Variables > Create*.

The screenshot shows the Apache Airflow Admin Variables page. At the top, there is a navigation bar with links for Airflow, DAGs, Data Profiling, Browse, Admin (which is highlighted with a red box), Docs, and About. A dropdown menu for Admin is open, showing options like Pools, Configuration, Users, Connections, Variables (which is also highlighted with a red box), and XComs. Below the navigation, the main content area has a title "Variable [create]" and two buttons: "List" and "Create". The "Create" button is selected. The form fields are "Key *" (set to "EMAIL") and "Val" (set to "mihai.botezatu@stratebi.com"). At the bottom of the form are four buttons: "Save" (in a dark green box), "Save and Add Another", "Save and Continue Editing", and "Cancel" (in a pink box).

Definimos la tarea iniciando *KitchenOperator* con los siguientes argumentos:

- task_id – id de la tarea.
- conn_id – id de la conexión definida en el paso anterior.
- direcotry – el directorio de Pentaho en el que se encuentra el trabajo.
- job – el nombre del trabajo.
- params – en este caso no le pasamos parámetros, pero el argumento es obligatorio.
- dag – el dag.

```
from airflow.operators.pentaho import KitchenOperator
from airflow.models import Variable

email_tasa_letalidad = KitchenOperator(
    task_id='email_tasa_letalidad',
    conn_id='pdi_default',
    directory='/home/admin/case_fatality_rate_email',
    job='main',
    params={'EMAIL': Variable.get('EMAIL')},
    dag=dag,)
```

5. Dependencias

Por último, tenemos que definir las dependencias entre las tareas.

```
crear_tabla >> obtener_datos_con_talend >> calcular_tasa_lethalidad_con_pdi
>> email_tasa_lethalidad
```

6. Ejecución del DAG

Para poder ejecutar el DAG hay que meter el archivo py con el código fuente en la carpeta *dags* del directorio de inicio de Apache Airflow. Si la carpeta no existe, hay que crearla.

Podemos obtener un listado de todos los DAGs ejecutando el siguiente comando:

```
airflow list_dags
```

Podemos obtener un listado de las tareas de un DAG empleando el comando:

```
airflow list_tasks <id_dag>
```

Ejemplo:

```
$ airflow list_tasks ejemplo_covid_pdi_talend
calcular_tasa_letalidad_con_pdi
crear_tabla
email_tasa_letalidad
obtener_datos_con_talend
```

Podemos probar una tarea del DAG con el comando:

```
airflow test <id_dag> <id_tarea> <fecha_ejecucion>
```

Ejemplo:

```
airflow test ejemplo_covid_pdi_talend email_tasa_letalidad 2020-05-13
```

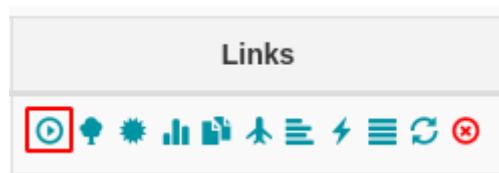
Para activar el DAG tenemos que acceder a la interfaz web de Apache Airflow. En la página de inicio hay un listado con todos los DAGs, buscamos nuestro ejemplo y marcamos la casilla *Off* para que se ponga en *On*.



A screenshot of the Apache Airflow web interface. The top navigation bar includes links for 'DAGs', 'Data Profiling', 'Browse', 'Admin', 'Docs', and 'About'. The main title 'DAGs' is highlighted. A search bar labeled 'Search:' is present. Below is a table listing four DAGs:

| | DAG | Schedule | Owner | Recent Tasks | Last Run | DAG Runs | Links |
|--|--|----------------|---------|--------------|------------------|----------|-------|
| | example_covid_pdi_talend | 1 day, 0:00:00 | airflow | | 2020-05-12 00:00 | | |
| | example_bash_operator | 8 2 * * * | Airflow | | | | |
| | example_branch_dop_operator_v3 | 17 4 * * * | Airflow | | | | |
| | example_branch_operator | @daily | Airflow | | | | |

Una vez activo, Apache Airflow ejecutará el DAG, pero también lo podemos ejecutar manualmente pinchando en el primer ícono de la columna *Links*.



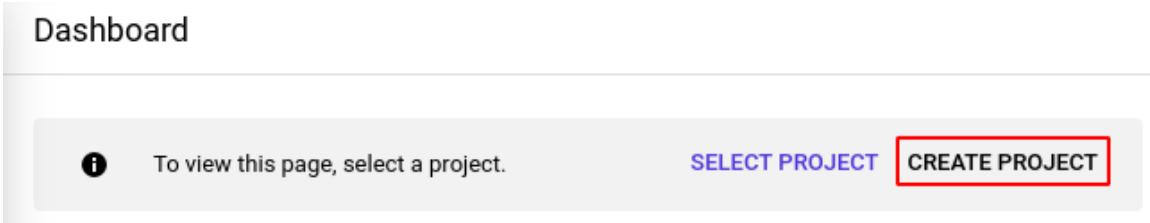
En la columna *Recent Tasks* podemos ver el estado de las tareas y en la columna *DAG Runs* podemos ver el estado de las ejecuciones del DAG.

8. GOOGLE CLOUD COMPOSER

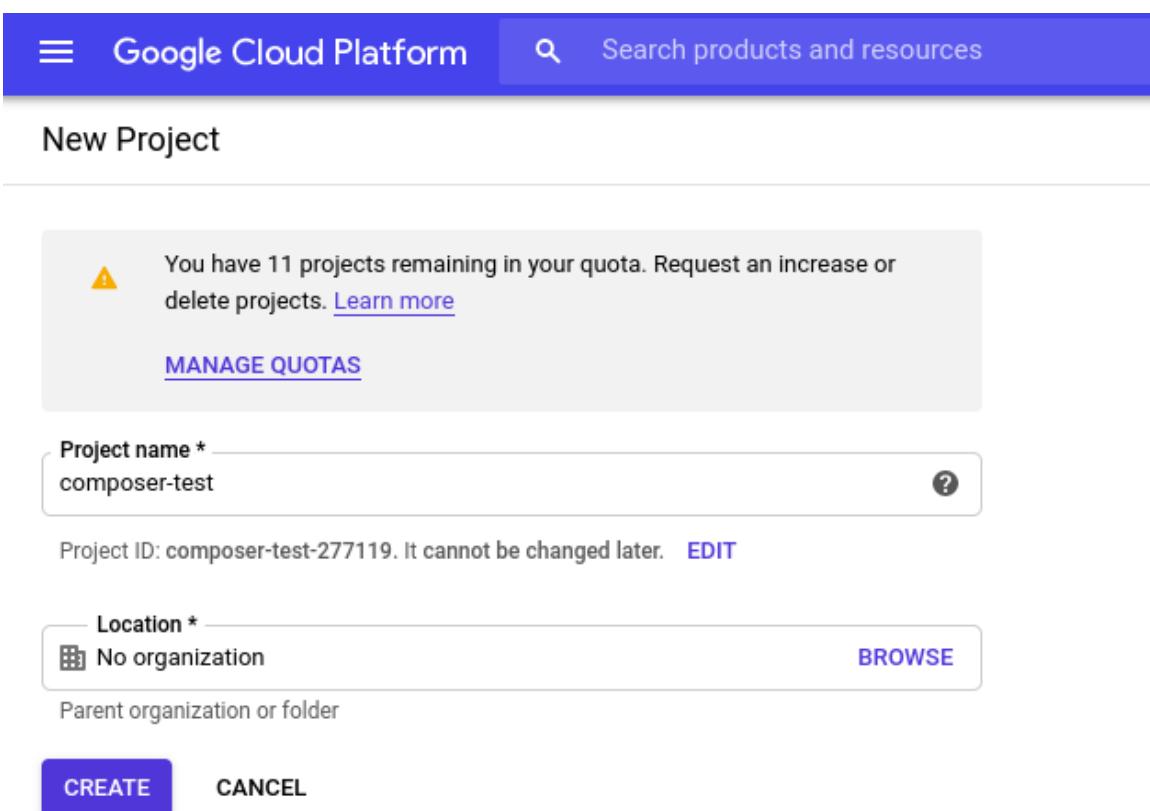
Según la web de [Google Cloud Composer](#): "Cloud Composer es un servicio totalmente gestionado de orquestación de flujos de trabajo que permite crear, programar y supervisar los flujos de procesamiento que se extienden por las nubes y los centros de datos on-premise. Cloud Composer se basa en el popular proyecto de código abierto Apache Airflow y utiliza el lenguaje de programación Python, por lo que es fácil de usar y no exige ninguna dependencia."

1. Requisitos

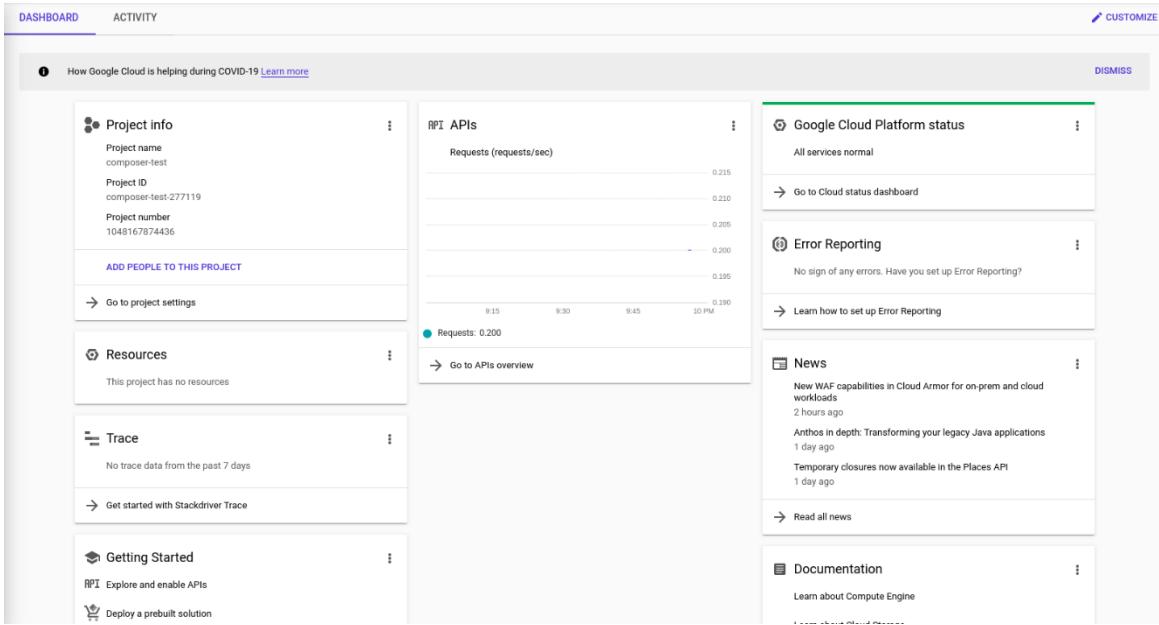
Antes de empezar hay acceder al [dashboard](#) de los proyectos de Google Cloud y seleccionar o crear un proyecto.



The screenshot shows the Google Cloud Platform Dashboard. At the top, there is a message: "To view this page, select a project." Below it are two buttons: "SELECT PROJECT" and "CREATE PROJECT". The "CREATE PROJECT" button is highlighted with a red border.



The screenshot shows the "New Project" creation form. It includes a warning message: "⚠ You have 11 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)". Below this is a "MANAGE QUOTAS" link. The "Project name *" field contains "composer-test". The "Location *" field shows "No organization" and has a "BROWSE" button. A note below says "Parent organization or folder". At the bottom are "CREATE" and "CANCEL" buttons.



The screenshot shows the Google Cloud Platform Dashboard. At the top, there's a banner with the text "How Google Cloud is helping during COVID-19" and a "Learn more" link, along with a "DISMISS" button and a "CUSTOMIZE" link.

The dashboard is divided into several sections:

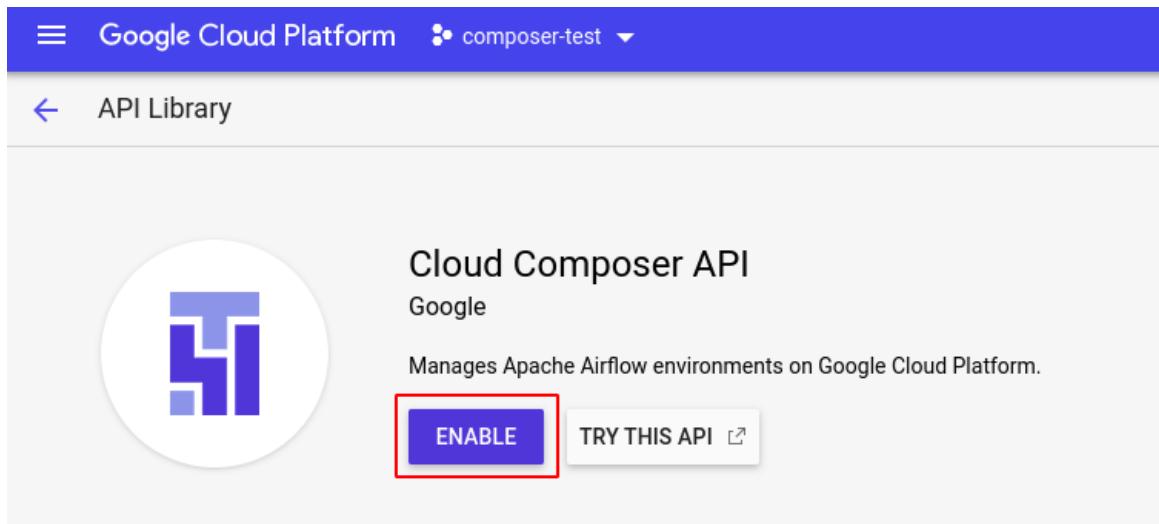
- Project info:** Shows the project name (composer-test), Project ID (composer-test-277119), and Project number (1048167874436). It also has a "ADD PEOPLE TO THIS PROJECT" button and a "Go to project settings" link.
- RPC APIs:** A chart titled "Requests (requests/sec)" showing data from 8:15 to 10 PM. The values fluctuate between 0.190 and 0.215. A callout says "Requests: 0.200". Below the chart is a link to "Go to APIs overview".
- Google Cloud Platform status:** Shows "All services normal" and a link to "Go to Cloud status dashboard".
- Error Reporting:** States "No sign of any errors. Have you set up Error Reporting?" and a link to "Learn how to set up Error Reporting".
- News:** Lists recent news items:
 - New WAF capabilities in Cloud Armor for on-prem and cloud workloads (2 hours ago)
 - Anthos in depth: Transforming your legacy Java applications (1 day ago)
 - Temporary closures now available in the Places API (1 day ago)A link "Read all news" is provided.
- Documentation:** A link to "Learn about Compute Engine".
- Getting Started:** Links to "Explore and enable APIs" and "Deploy a prebuilt solution".
- Trace:** States "No trace data from the past 7 days" and a link to "Get started with Stackdriver Trace".

Luego, hay que habilitar la API de Cloud Composer en *APIs & Services > Dashboard*.

The screenshot shows the Google Cloud Platform dashboard for a project named "composer-test". The left sidebar has several options: Home, Marketplace, Billing, API, Support, IAM & Admin, Getting started, Security, and Anthos. The "API" option is highlighted with a red box. A dropdown menu is open under "API", showing "Dashboard" and other options like Library, Credentials, OAuth consent screen, Domain verification, and Page usage agreements. The "Dashboard" option is also highlighted with a red box.

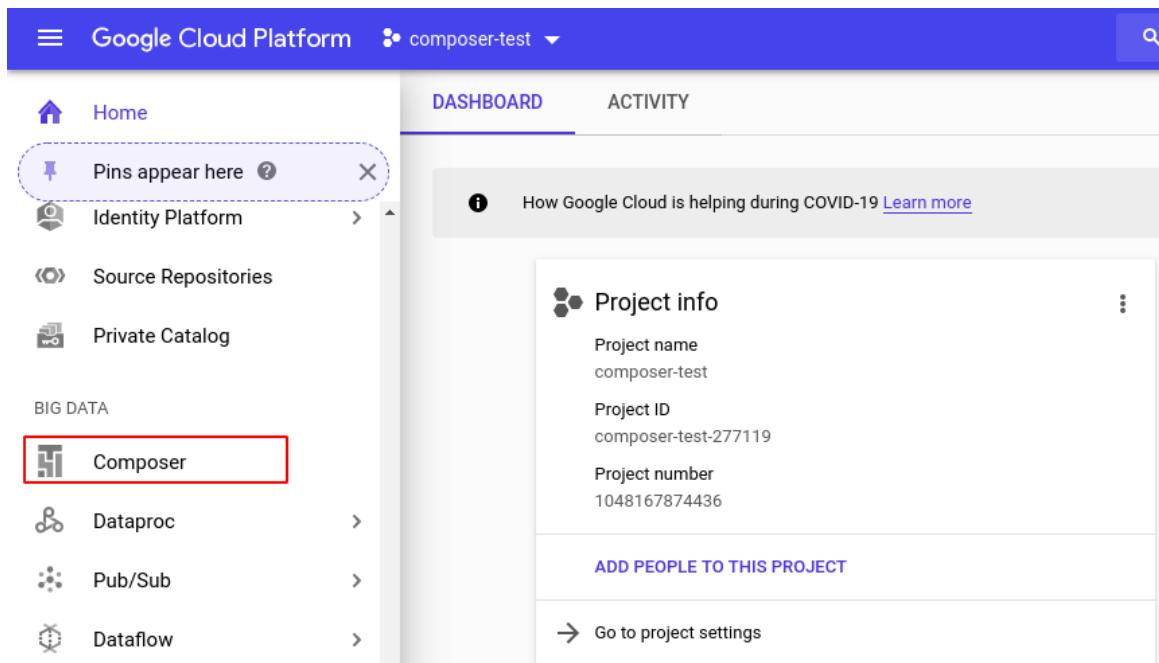
The screenshot shows the "APIs & Services" page. It has two tabs: "API" and "Dashboard". The "API" tab is selected. On the right side, there is a button labeled "+ ENABLE APIS AND SERVICES" with a red box around it. The "Dashboard" tab is also visible.

Buscamos "composer" y hacemos click en el botón *Enable*.



2. Creación del entorno

Para crear el entorno tenemos que hacer click en *Composer*.



Nos lleva a otra página donde podemos hacer click en *CREATE ENVIRONMENT*.

The screenshot shows the Google Cloud Platform interface for Cloud Composer environments. At the top, there's a blue header bar with the Google Cloud logo, the text "Google Cloud Platform", a dropdown menu showing "composer-test", a search bar with the placeholder "Search products", and a dropdown arrow. Below the header, the navigation bar has two items: "Composer" (with a logo) and "Environments". On the right side of the navigation bar is a square icon. The main content area has a light gray background. It features the heading "Cloud Composer Environments" in bold. Below the heading is a descriptive text: "With Composer, you can easily create and manage Airflow environments. Get started by creating your first environment." At the bottom of the main content area are two buttons: a red-bordered "CREATE ENVIRONMENT" button and a "LEARN MORE" button.

Para la configuración del entorno utilizamos los valores por defecto, pero hay que especificar el nombre (*Name*) y la ubicación (*Location*).

Google Cloud Platform composer-test Search products

Composer Create environment

Name * entorno-test

Node configuration

The configuration information for the Google Kubernetes Engine nodes running the Airflow software.

Node count * 3

The number of nodes in the Google Kubernetes Engine cluster that will be used to run this environment.

Location * europe-west1

The Google Compute Engine region where the environment will be created.

Zone europe-west1-b

The Compute Engine zone in which to deploy the VMs used to run the Apache Airflow software. If unspecified, the service will pick a zone in the Compute Engine region corresponding to the selected location. Must specify location before selecting zone. [Learn more](#).

Tardará unos minutos en crear el entorno.

Google Cloud Platform composer-test Search products

Composer Environments CREATE DELETE Enable Beta Features

Filter environments

| <input type="checkbox"/> | <input checked="" type="checkbox"/> | Name ↑ | Location | Creation time | Update time | Airflow webserver | Logs | DAGs folder | Labels |
|--------------------------|-------------------------------------|---------------------|--------------|-------------------|-------------------|----------------------|-------------------|-------------------|--------|
| <input type="checkbox"/> | <input checked="" type="checkbox"/> | entorno-test | europe-west1 | 5/13/20, 10:19 PM | 5/13/20, 10:33 PM | Airflow | Logs | DAGs | None |

3. Ejecución de DAGs

Creamos un DAG de ejemplo para ejecutarlo en Google Cloud Composer. La tarea muestra en los logs el id del DAG.

```

import datetime
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

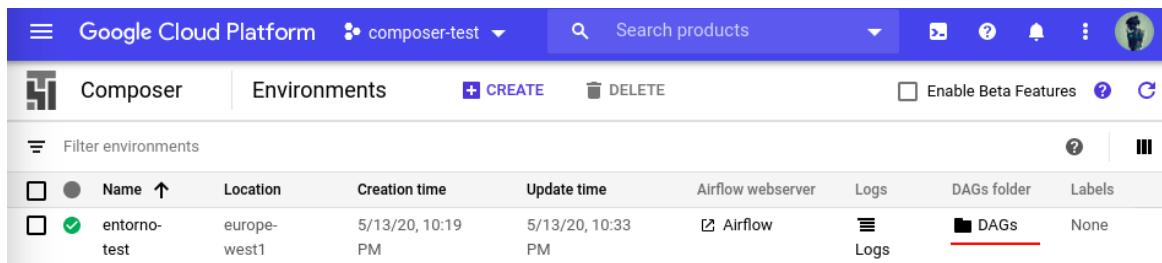
default_args = {
    'owner': 'Composer',
    'depends_on_past': False,
    'email': [],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': datetime.timedelta(minutes=1),
    'start_date': days_ago(1),
}

with DAG(
    'ejemplo_composer',
    default_args=default_args,
    schedule_interval=datetime.timedelta(days=1)) as dag:

    print_dag_run_conf = BashOperator(
        task_id='print_conf',
        bash_command='echo {{ dag_run.id }}')

```

Google Cloud Composer ejecuta solo los DAGs que se encuentran en la carpeta *DAGs* del Google Cloud Storage del entorno. Hacemos click en *DAGs*.



The screenshot shows the Google Cloud Platform Composer interface. At the top, there's a navigation bar with the Google Cloud logo, 'Google Cloud Platform', a dropdown for 'composer-test', a search bar 'Search products', and various icons for help, notifications, and user profile.

The main area is titled 'Composer' and shows a table of environments. There is one environment listed:

| Name | Location | Creation time | Update time | Airflow webserver | Logs | DAGs folder | Labels |
|--|--------------|-------------------|-------------------|-------------------|------|-------------|--------|
| <input checked="" type="checkbox"/> entorno-test | europe-west1 | 5/13/20, 10:19 PM | 5/13/20, 10:33 PM | Airflow | Logs | DAGs | None |

Subimos el archivo py del ejemplo.

The screenshot shows the 'Bucket details' page for 'europe-west1-entorno-test-10539e7a-bucket'. The left sidebar has 'Browser' selected. The main area shows two uploaded files: 'airflow_monitoring.py' and 'ejemplo_composer.py'. The 'Upload files' button is highlighted with a red box.

| Name | Size | Type | Storage class | Last modified | Public access | Encryption |
|-----------------------|-------|---------------|---------------|----------------------------|---------------|--------------------|
| airflow_monitoring.py | 729 B | text/x-python | Standard | 5/13/20, 10:29:43 PM UTC+2 | Not public | Google-managed key |
| ejemplo_composer.py | 666 B | text/x-python | Standard | 5/13/20, 11:05:05 PM UTC+2 | Not public | Google-managed key |

Accedemos a la interfaz web de Airflow.

The screenshot shows the 'Composer' environments page. It lists one environment named 'entorno-test' located in 'europe-west1'. The 'Airflow' link in the row for 'entorno-test' is highlighted with a red box.

| Name | Location | Creation time | Update time | Airflow webserver | Logs | DAGs folder | Labels |
|--------------|--------------|-------------------|-------------------|-------------------|------|-------------|--------|
| entorno-test | europe-west1 | 5/13/20, 10:19 PM | 5/13/20, 10:33 PM | Airflow | Logs | DAGs | None |

Por defecto, Google Cloud Composer activa todos los DAGs subidos.

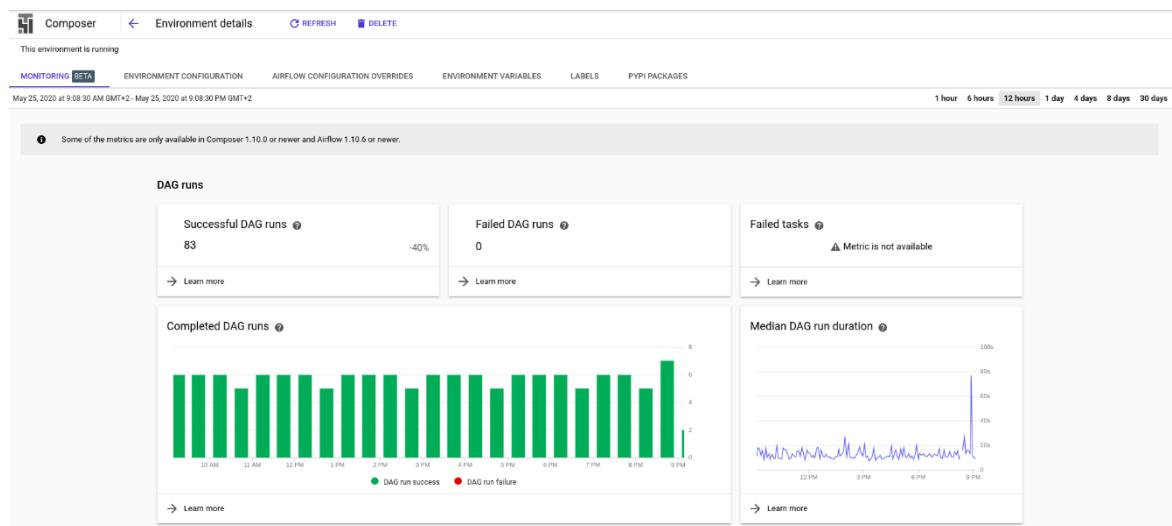
The screenshot shows the 'DAGs' page in the Apache Airflow UI. It displays two active DAGs: 'airflow_monitoring' and 'ejemplo_composer'. Both DAGs have a green status icon indicating they are running. The 'airflow_monitoring' DAG was last run on 2020-05-25 at 18:50, and the 'ejemplo_composer' DAG was last run on 2020-05-24 at 00:00. The 'DAG Runs' column shows the number of runs for each DAG.

| DAG | Schedule | Owner | Recent Tasks | Last Run | DAG Runs | Links |
|--------------------|----------------|----------|------------------|------------------|----------|---------|
| airflow_monitoring | None | Airflow | 100% (100 tasks) | 2020-05-25 18:50 | 273 | Details |
| ejemplo_composer | 1 day, 0:00:00 | Composer | 100% (100 tasks) | 2020-05-24 00:00 | 1 | Details |

Podemos monitorizar el entorno haciendo click en el nombre.

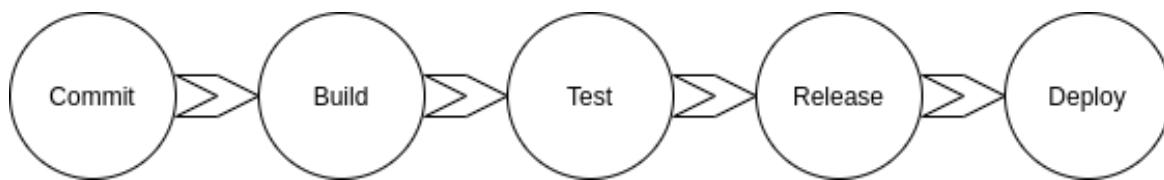
| Name | Location | Creation time | Update time | Airflow webserver | Logs | DAGs folder | Labels |
|--------------|--------------|------------------|------------------|-------------------|------|-------------|--------|
| compose-test | europe-west1 | 5/24/20, 8:56 PM | 5/24/20, 9:11 PM | Airflow | Logs | DAGs | None |

Se puede encontrar más información en la documentación de [Google Cloud Composer](#).



9. INTEGRACIÓN Y DESPLIEGUE CONTINUO

La integración y el despliegue continuo es una parte integral de cualquier sistema software para aumentar la confiabilidad del sistema. El objetivo de la integración continua es establecer una forma automatizada para compilar y probar aplicaciones cuando hay cambios en el repositorio. El despliegue continuo automatiza la publicación de aplicaciones en los entornos de la infraestructura.



Como esta parte depende de la infraestructura que ya tenemos montada, nos vamos a centrar en la parte de las pruebas.

Hay varios tipos de pruebas para los DAGs, pero los más importantes son:

- Pruebas de validación de los DAGs.
- Pruebas de definición de los DAGs.
- Pruebas de lógica.
- Pruebas de extremo a extremo.

Vamos a usar el siguiente ejemplo para las pruebas:

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator

def hola():
    return 'Hola Mundo'

def incrementar(variable):
    return variable + 1

dag = DAG(
    'ejemplo_ci',
    schedule_interval=timedelta(days=1),
    start_date=datetime(2020, 5, 20),
    catchup=False
)

dummy = DummyOperator(
    task_id='dummy',
    dag=dag
)

hola_mundo = PythonOperator(
    task_id='hola_mundo',
    python_callable=hola,
    dag=dag
)

incrementar_x = PythonOperator(
    task_id='incrementar_variable',
    python_callable=incrementar,
    op_kwargs={'variable': 1},
    dag=dag
)
```

```
)  
  
dummy >> [hola_mundo, incrementar_x]
```

1. Pruebas de validación

Las pruebas de validación son comunes para todos los DAGs. Lo mínimo que debe hacer una prueba de validación es verificar la correctitud del DAG y si contiene o no ciclos. Además, si queremos seguir una estructura en concreto para los DAGs, podemos especificar pruebas para esto.

Ejemplo de pruebas de validación:

```
import unittest
from airflow.models import DagBag

class PruebaValidacionDag(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.dagbag = DagBag()

    def test_import(self):
        self.assertDictEqual(self.dagbag.import_errors, {})

    def test_email_on_failure(self):
        for dag_id, dag in self.dagbag.dags.items():
            ok = dag.default_args.get('email_on_failure', False)
            msg = f'email_on_failure no configurado para {dag_id}'
            self.assertTrue(email_on_failure, msg=msg)

suite = unittest.TestLoader()
.loadTestsFromTestCase(PruebaValidacionDag)
unittest.TextTestRunner(verbosity=2).run(suite)
```

En la función `test_import` probamos si hay errores semánticos o sintácticos en el código, así como si el DAG es cíclico u otros errores de Apache Airflow. La función `test_email_on_failure` define una prueba que falla si el argumento `email_on_failure` del DAG no tiene el valor `True`.

En este caso, el output de las pruebas es el siguiente:

```
test_email_on_failure (ejemplo_ci_test.PruebaValidacionDag) ... FAIL
test_import (ejemplo_ci_test.PruebaValidacionDag) ... ok
```

2. Pruebas de definición

Las pruebas de definición nos ayudan a verificar la definición del pipeline y son específicas por cada DAG. En esta parte podemos verificar el número de tareas, el ID de cada tarea, las dependencias entre las tareas...

Ejemplo de pruebas de definición:

```

import unittest
from airflow.models import DagBag

class PruebasDefinicionDAG(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.dagbag = DagBag()

    def test_num_tareas(self):
        dag = self.dagbag.get_dag('ejemplo_ci')
        self.assertEqual(len(dag.tasks), 3)

    def test_tareas(self):
        dag = self.dagbag.get_dag('ejemplo_ci')
        ids = set([task.task_id for task in dag.tasks])

        self.assertSetEqual(ids, {'dummy', 'hola_mundo',
                                'incrementar_variable'})

    def test_dependencias_dummy(self):
        dag = self.dagbag.get_dag('ejemplo_ci')
        dummy = dag.get_task('dummy')

        upstream_ids = set([task.task_id for task in dummy.upstream_list])
        self.assertSetEqual(upstream_ids, set())

        downstream_ids = set([task.task_id for task in
                             dummy.downstream_list])
        self.assertSetEqual(downstream_ids, {'hola_mundo',
                                            'incrementar_variable'})

    def test_dependencias_hola_mundo(self):
        dag = self.dagbag.get_dag('ejemplo_ci')
        tarea = dag.get_task('hola_mundo')

        upstream_ids = set([task.task_id for task in tarea.upstream_list])
        self.assertSetEqual(upstream_ids, {'dummy'})

```

```

        downstream_ids = set([task.task_id for task in
tarea.downstream_list])
        self.assertSetEqual(downstream_ids, set())

    def test_dependencias_incrementar_variable(self):
        dag = self.dagbag.get_dag('ejemplo_ci')
        tarea = dag.get_task('incrementar_variable')

        upstream_ids = set([task.task_id for task in tarea.upstream_list])
        self.assertSetEqual(upstream_ids, {'dummy'})

        downstream_ids = set([task.task_id for task in
tarea.downstream_list])
        self.assertSetEqual(downstream_ids, set())

suite = unittest.TestLoader()
    .loadTestsFromTestCase(PruebasDefinicionDAG)
unittest.TextTestRunner(verbosity=2).run(suite)

```

El output de las pruebas:

```

test_dependencias_dummy (ejemplo_ci_test2.PruebasDefinicionDAG) ... ok
test_dependencias_hola_mundo (ejemplo_ci_test2.PruebasDefinicionDAG) ... ok
test_dependencias_incrementar_variable
(ejemplo_ci_test2.PruebasDefinicionDAG) ... ok
test_num_tareas (ejemplo_ci_test2.PruebasDefinicionDAG) ... ok
test_tareas (ejemplo_ci_test2.PruebasDefinicionDAG) ... ok

```

3. Pruebas de lógica

Las pruebas de lógica nos permiten verificar la lógica de los DAGs o de las tareas.

Ejemplo de pruebas de lógica:

```

import unittest
from airflow.models import DagBag

class PruebasLogicaDAG(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.dagbag = DagBag()

```

```

def test_hola_mundo(self):
    res = self.dagbag \
        .get_dag('ejemplo_ci') \
        .get_task('hola_mundo') \
        .execute_callable()
    self.assertEqual(res, 'Hola Mundo')

def test_incrementar_variable(self):
    res = self.dagbag \
        .get_dag('ejemplo_ci') \
        .get_task('incrementar_variable') \
        .execute_callable()
    self.assertEqual(res, 2)

suite = unittest.TestLoader()
    .loadTestsFromTestCase(PruebasLogicaDAG)
unittest.TextTestRunner(verbosity=2).run(suite)

```

Output de las pruebas:

```

test_hola_mundo (ejemplo_ci_test3.PruebasLogicaDAG) ... ok
test_incrementar_variable (ejemplo_ci_test3.PruebasLogicaDAG) ... ok

```

4. Pruebas de extremo a extremo

En estas pruebas ejecutamos el DAG completo, con una muestra de datos o datos mock-up, para verificar que el resultado es el esperado. Esta parte también depende mucho de la infraestructura que tenemos, pues hay que crear un entorno de test para hacer las pruebas. Este entorno debe comportarse igual que los demás entornos (dev, staging, prod).

5. Recomendaciones

- Para no introducir duplicados el nombre del archivo py puede ser igual al ID del DAG.
- Considerar usar Docker o Kubernetes para las pruebas de extremo a extremo.

10.

CASOS DE USO Y BENEFICIOS

- Permite la creación, programación y monitorización centralizada de flujos de trabajos complejos que se conectan a varios backends.
- Código personalizado para la lógica de reintentos cuando una tarea falla.
- Orquestación estandarizada de ETLs.

11. ANEXO

1. Código del primer ejemplo

```
from datetime import timedelta
from airflow import DAG
from airflow.utils.dates import days_ago
from airflow.operators.postgres_operator import PostgresOperator
from airflow.operators.bash_operator import BashOperator
from airflow.operators.pentaho import KitchenOperator
from airflow.models import Variable

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=1),}

dag = DAG('ejemplo_covid_pdi_talend',
           default_args=default_args,
           schedule_interval=timedelta(days=1))

crear_tabla = PostgresOperator(
    task_id='crear_tabla',
    postgres_conn_id='con_postgres',
    database='ejemplo',
    sql="""
CREATE TABLE IF NOT EXISTS public.covid
(
    Country text,
    CountryCode text,
    Province text,
    City text,
    CityCode text,
    Lat double precision,
    Lon double precision,
    Confirmed integer,
    Deaths integer,
    Recovered integer,
    Active integer,
    Date date
```

```
);

TRUNCATE TABLE public.covid;
CREATE TABLE IF NOT EXISTS public.spain_case_fatality_rate
(
    date timestamp,
    case_fatality_rate numeric(18,2)
);
TRUNCATE TABLE public.spain_case_fatality_rate;

""",
dag=dag,
)

obtener_datos_con_talend = BashOperator(
    task_id='obtener_datos_con_talend',
    bash_command='/home/ubuntu/talend/covid/covid_run.sh ',
    dag=dag,
)

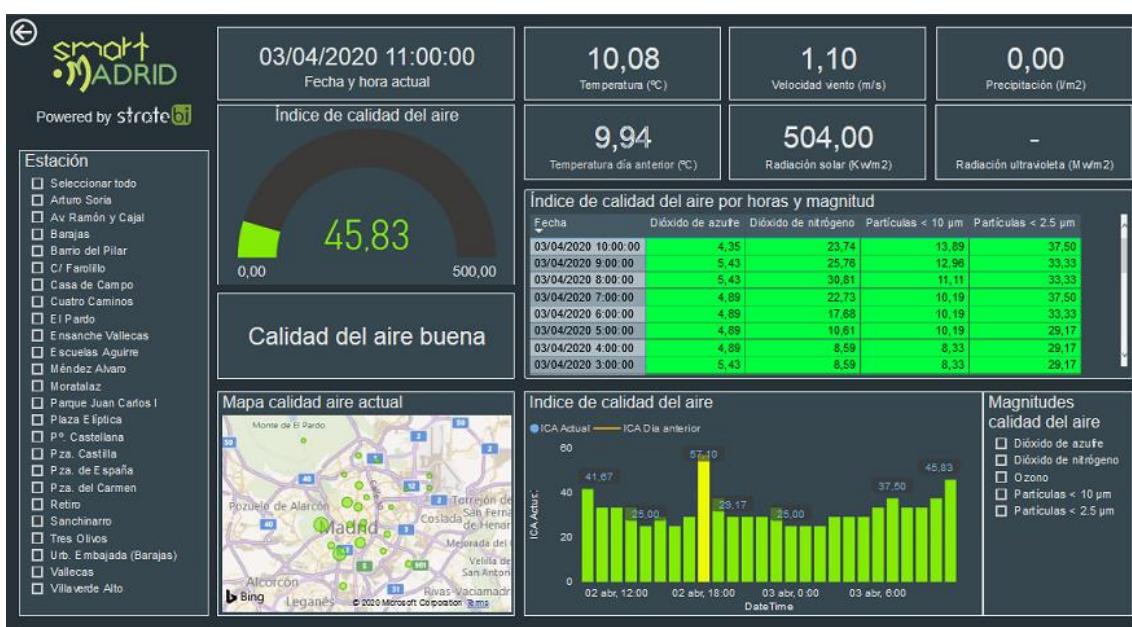
calcular_tasa_letalidad_con_pdi = BashOperator(
    task_id='calcular_tasa_letalidad_con_pdi',
    bash_command='/opt/pentaho/8.1/data-integration/kitchen.sh -
level="Basic" -file="/home/ubuntu/pdi/calc_case_fatality_rate
/main.kjb" ',
    dag=dag,
)

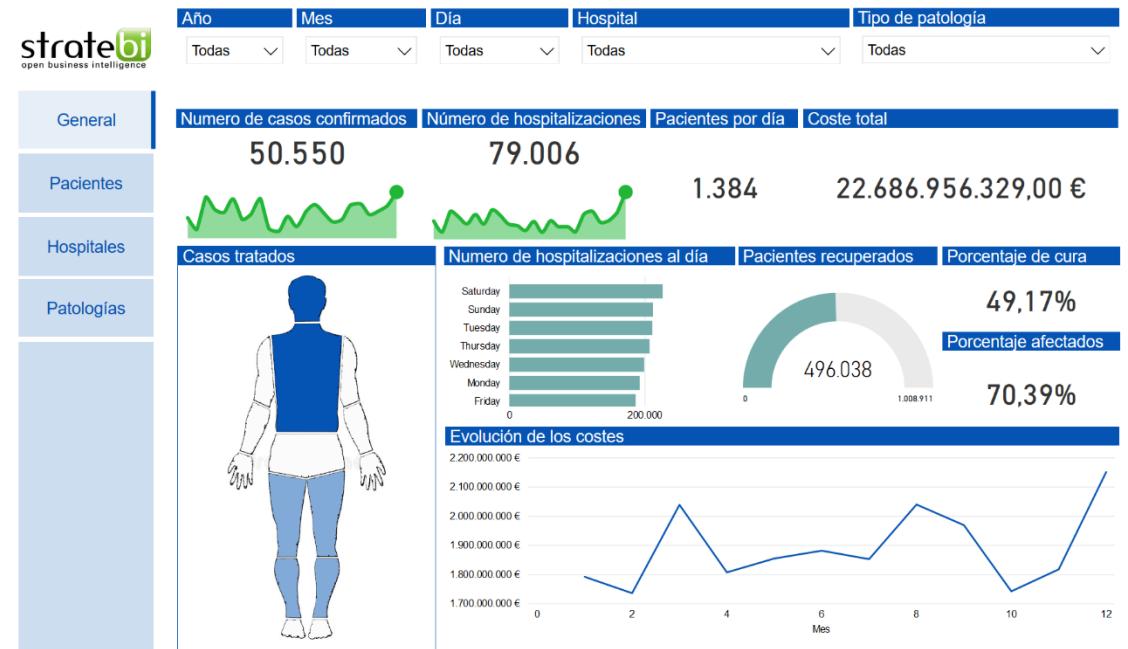
email_tasa_letalidad = KitchenOperator(
    task_id='email_tasa_letalidad',
    conn_id='pdi_default',
    directory='/home/admin/case_fatality_rate_email',
    job='main',
    params={'EMAIL': Variable.get('EMAIL')},
    dag=dag,
)

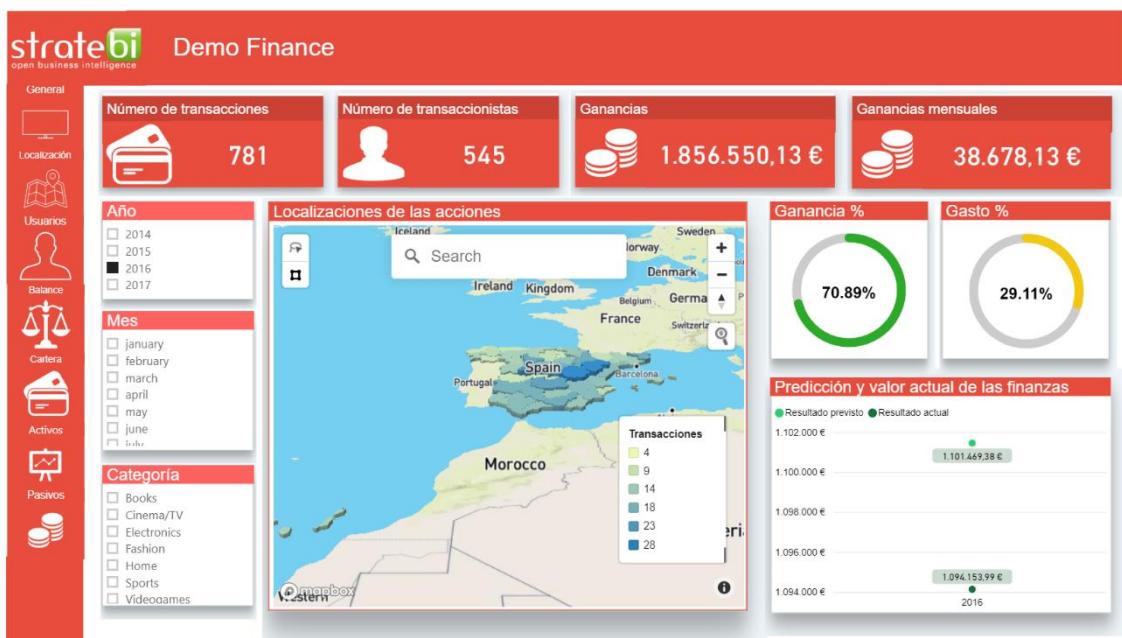
crear_tabla >> obtener_datos_con_talend >> calcular_tasa_letalidad_con_pdi
>> email_tasa_letalidad
```

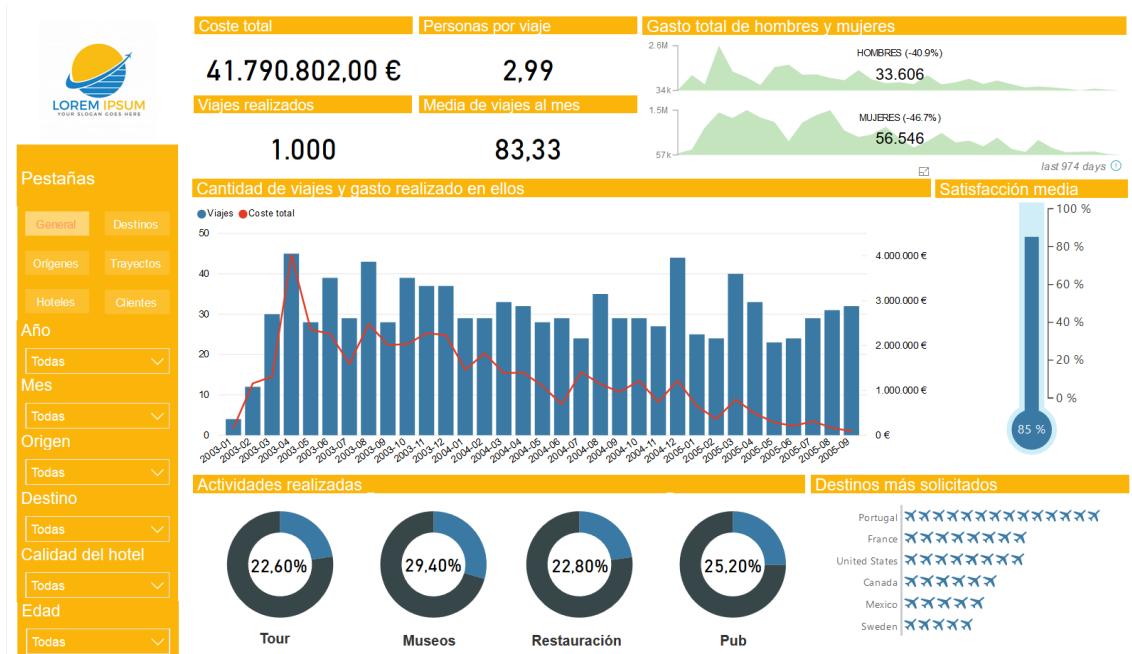
12. POWER BI

Stratebi es Partner Certificado en Microsoft Power BI. En esta sección puedes consultar algunas Demos Online en donde ver el potencial de la herramienta, así como algunos videotutoriales

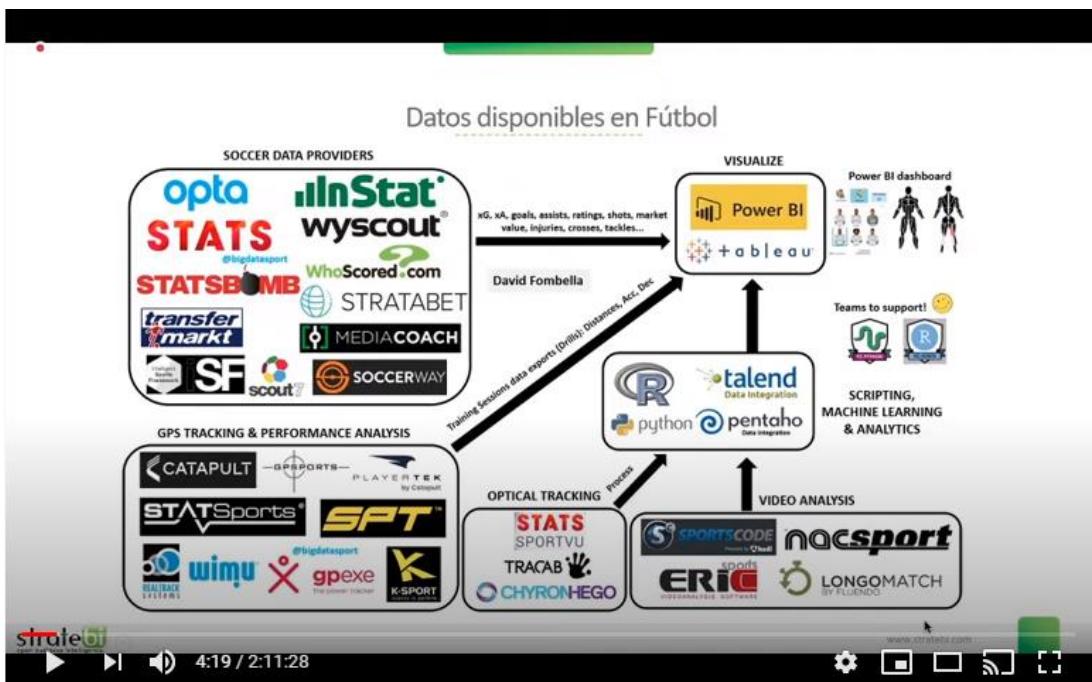
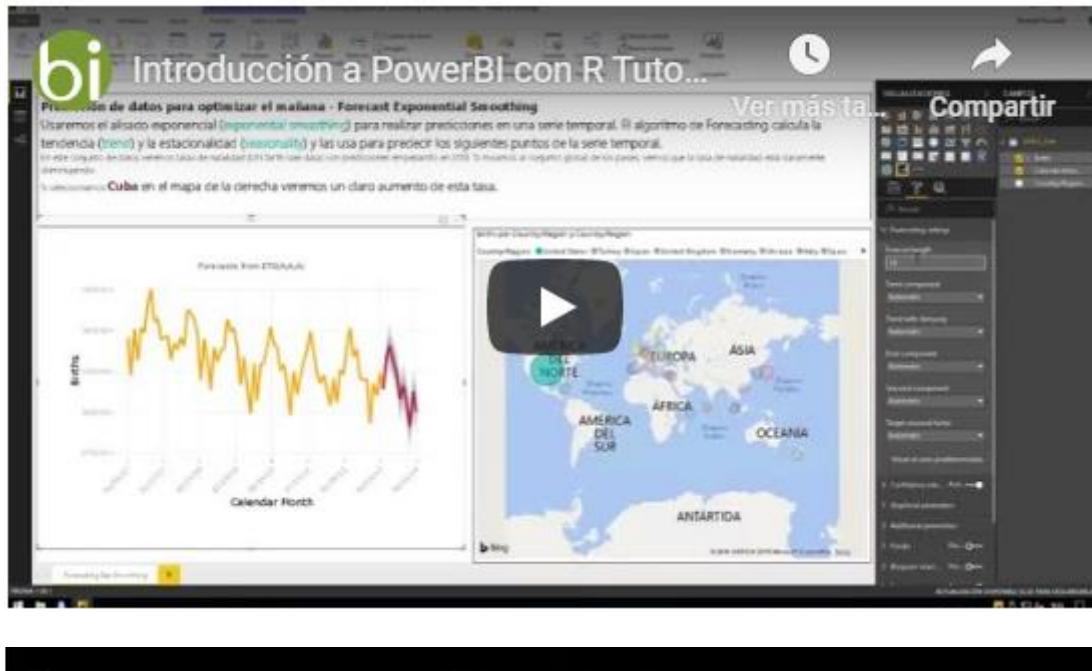












[Sports Analytics con PowerBI](#)

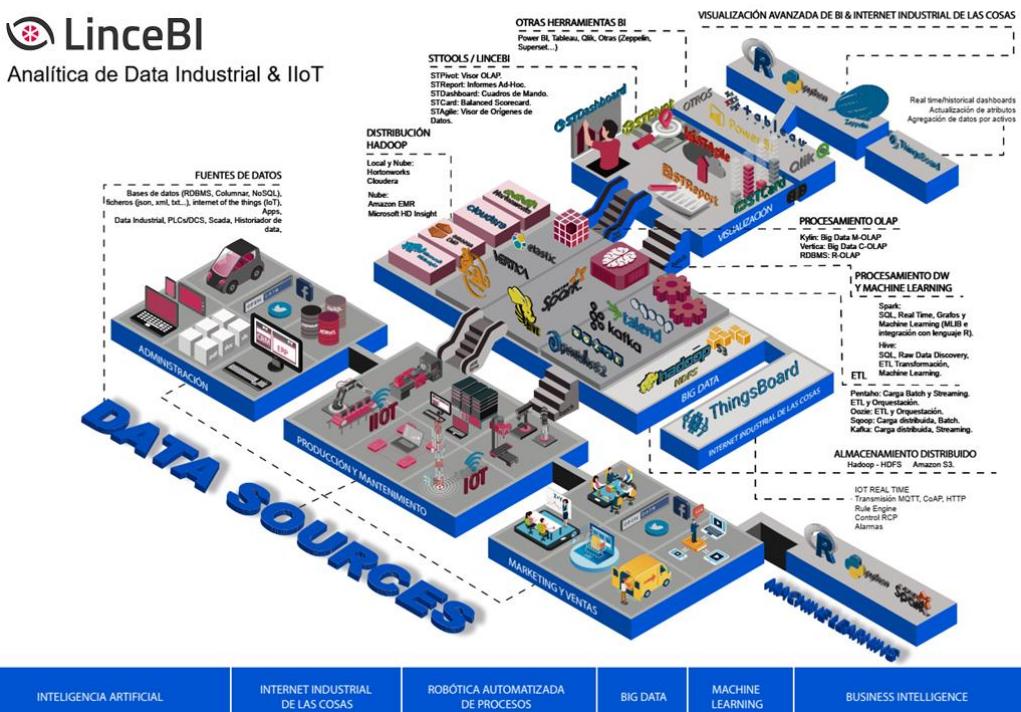
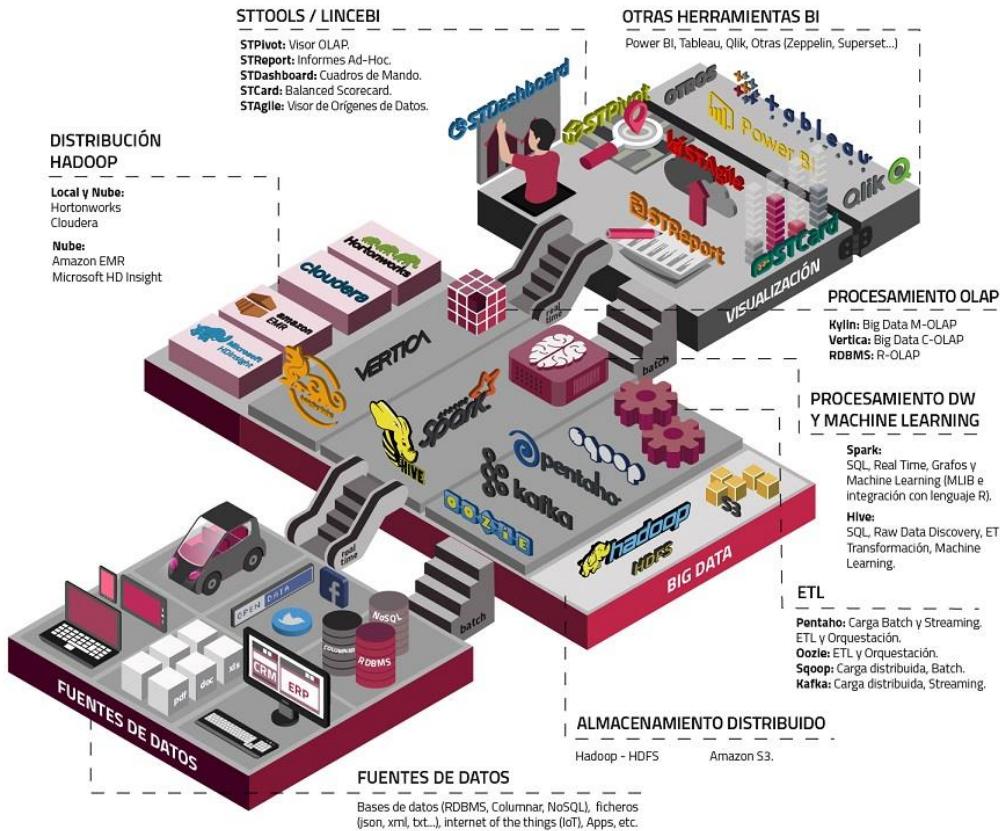
Recursos imprescindibles sobre PowerBI:

1. [Integracion SAP - PowerBI](#)
2. [Futbol Analytics, lo que hay que saber](#)
3. [Dashboard de medicion de la calidad del aire en Madrid](#)
4. [Como funciona Microsoft Power BI? Videotutorial de Introducción](#)
5. [Big Data para PowerBI](#)
6. [Como integrar Salesforce y PowerBI](#)
7. [Videotutorial: Usando R para Machine Learning con PowerBI](#)
8. [Las 50 claves para aprender y conocer PowerBI](#)
9. [PowerBI: Arquitectura End to End](#)
10. [Usando Python con PowerBI](#)
11. [PowerBI + Open Source = Sports Analytics](#)
12. [Comparativa de herramientas Business Intelligence](#)
13. [Use Case Big Data "Dashboards with Hadoop and Power BI"](#)
14. [Todas las presentaciones del Workshop 'El Business Intelligence del Futuro'](#)
15. [Descarga Paper gratuito: Zero to beautiful \(Data visualization\)](#)

13. TECNOLOGÍAS

Recientemente, hemos sido nombrados Partners Certificados de Vertica, Talend, Microsoft, Snowflake, Kylligence, Pentaho, etc.





14. INFORMACIÓN SOBRE STRATEBI



Stratebi es una empresa española, con sede en Madrid y oficinas en Barcelona, Alicante y Sevilla, creada por un grupo de profesionales con amplia experiencia en sistemas de información, soluciones tecnológicas y procesos relacionados con soluciones de Open Source y de inteligencia de Negocio.

Esta experiencia, adquirida durante la participación en proyectos estratégicos en compañías de reconocido prestigio a nivel internacional, se ha puesto a disposición de nuestros clientes.

Somos Partners Certificados en Microsoft PowerBI con una dilatada experiencia

Stratebi es la única empresa española que ha estado presente todos los Pentaho Developers celebrados en Europa habiendo organizado el de España.

En Stratebi nos planteamos como **objetivo** dotar a las compañías e instituciones, de herramientas escalables y adaptadas a sus necesidades, que conformen una estrategia Business Intelligence capaz de rentabilizar la información disponible. Para ello, nos basamos en el desarrollo de soluciones de Inteligencia de Negocio, mediante tecnología Open Source.

Stratebi son profesores y responsables de proyectos del Master en Business Intelligence de la Universidad UOC, UCAM, EOI...

Los profesionales de Stratebi son los creadores y autores del primer weblog en español sobre el mundo del Business Intelligence, Data Warehouse, CRM, Dashboards, Scorecard y Open Source. Todobi.com

Stratebi es partner de las principales soluciones Analytics: Microsoft Power BI, Talend, Pentaho, Vertica, Snowflake, Kyligence, Cloudera...

Todo Bi, se ha convertido en una referencia para el conocimiento y divulgación del Business Intelligence en español.

www.stratebi.com 91.788.34.10

info@stratebi.com



15. OTROS

Trabajamos en los principales sectores y con algunas de las compañías y organizaciones más importantes de España.

SECTOR PRIVADO



SECTOR PÚBLICO



16. EJEMPLOS DE DESARROLLOS ANALYTICS

A continuación, se presentan **ejemplos de algunos screenshots** de cuadros de mando diseñados por Stratebi, con el fin de dar a conocer lo que se puede llegar a obtener, así como Demos Online en la web de Stratebi:

