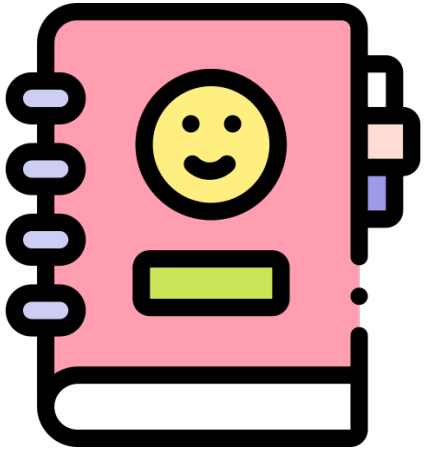
Orquestación de datos con Apache Airflow

Fiorella Piriz Sapio
Solution Architect en aws

Itinerario

- Introducción a la orquestación de datos
- Introducción a Apache Airflow
- Configuración de DAGs
- Caso Práctico
- Conclusión



01

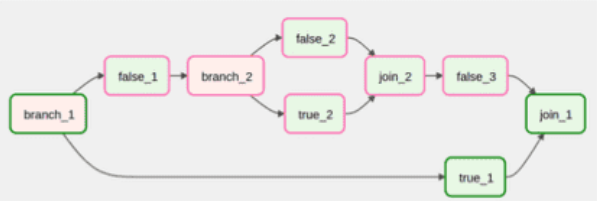
¿Qué es la orquestación de datos?

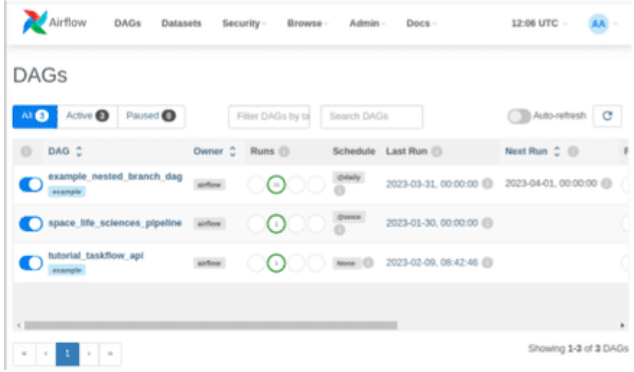
Orquestación de datos

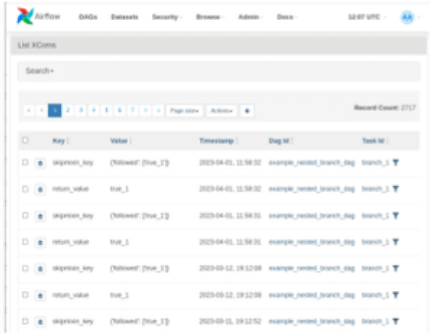
02

¿Qué es Apache Airflow?

Apache Airflow





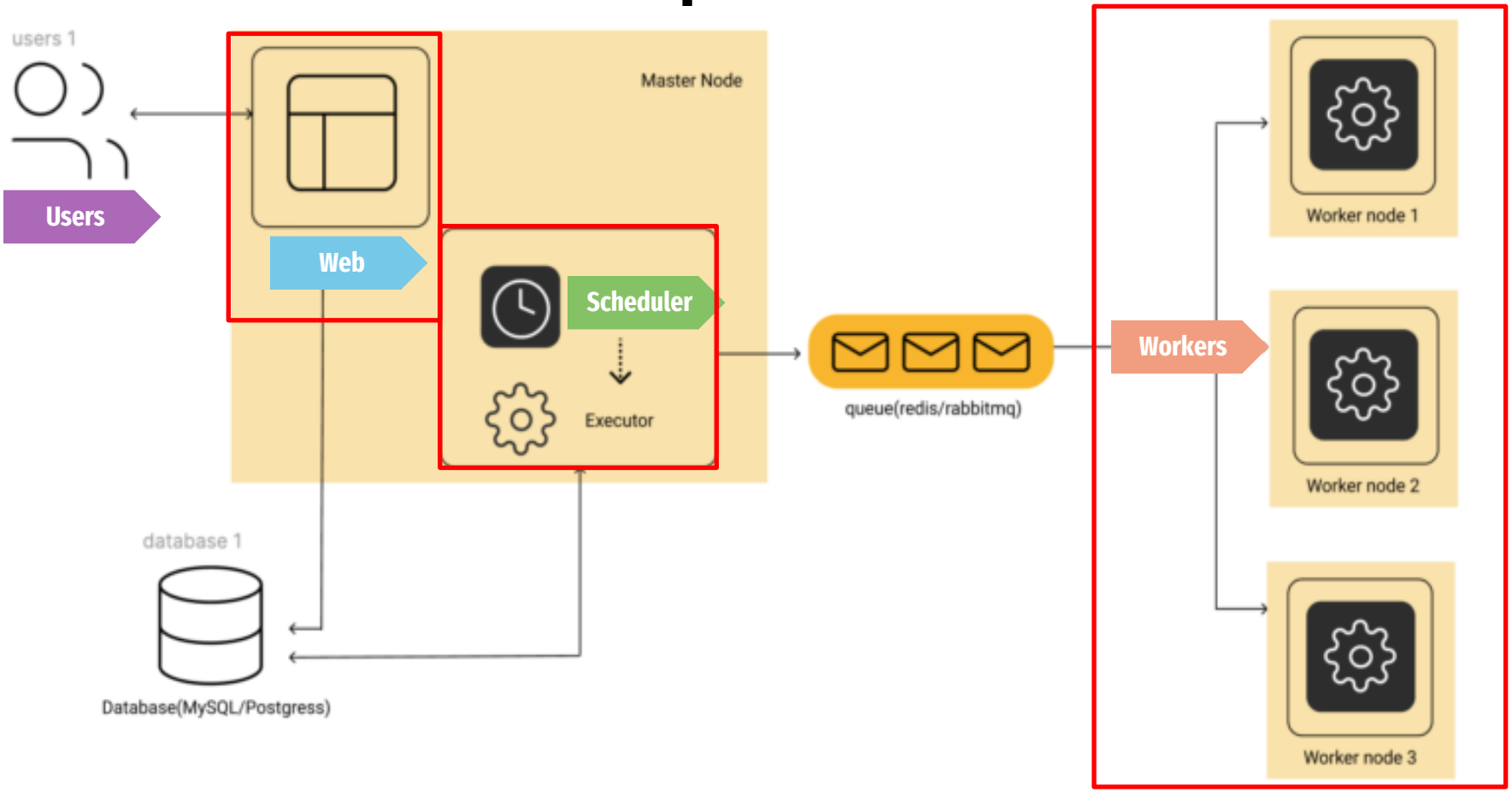


- Opensource
- Fácil de usar
- Basada en Python
- Múltiples conectores

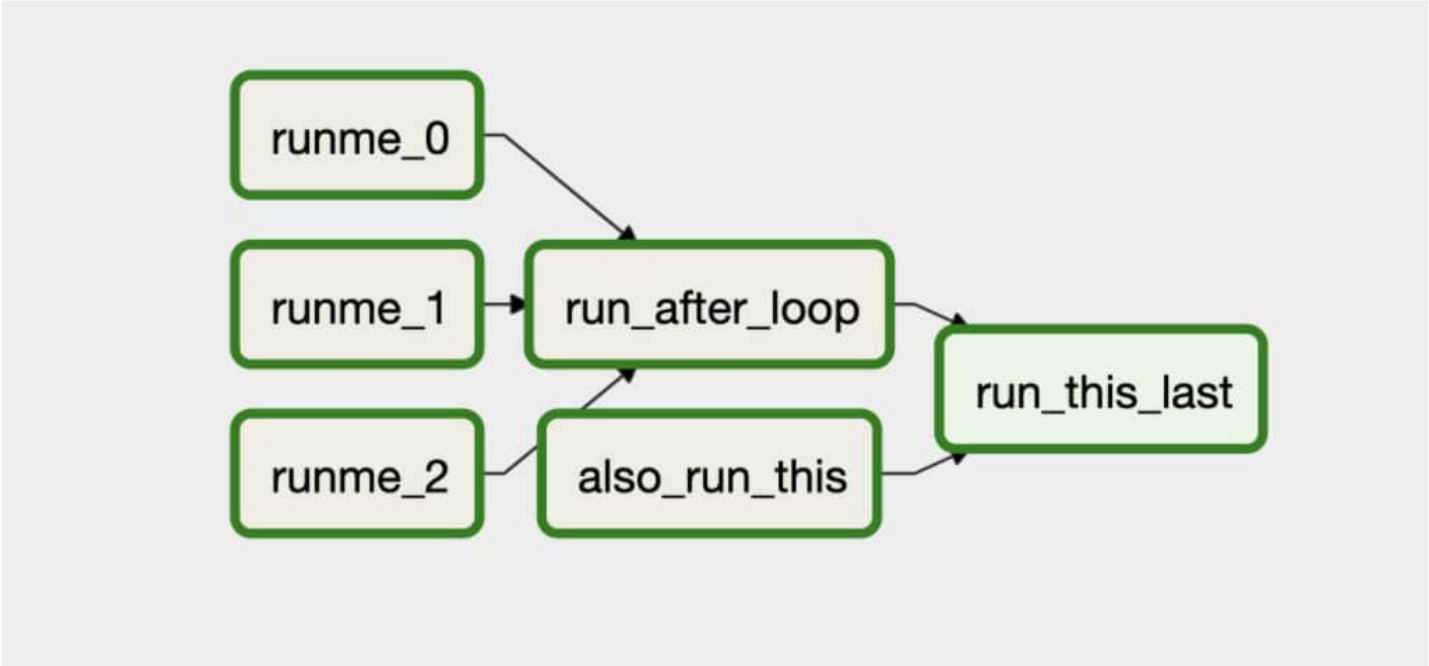
VS

Oozie | **Azkaban** | **Luigi**

Componentes



DAGS


runme_0
runme_1
runme_2
run_after_loop
also_run_this
run_this_last

03

Instalación y preparación del entorno

Instalación de Airflow en Linux

https://airflow.apache.org/docs/apache-airflow/stable/installation/index.html

Pasos		
01	Instalar Apache Airflow	pip install "apache-airflow[gcp]"
02	Configurar variable de entorno	export AIRFLOW_HOME=/home/airflow
03	Configuración	nano /root/airflow/airflow.cfg
04	Comprobar la versión	airflow version
05	Inicializar la base de datos	airflow db init
06	Crear usuario	airflow users create
07	Arrancar webserver	airflow webserver
08	Arrancar scheduler	airflow scheduler

Terminal Shell Edit View Window Help

Sun Sep 15 19:0

Downloads — root@ip-172-31-6-15:/home/ec2-user — ssh -i demo-airflow.pem ec2-...

```
(airflow_env) [root@ip-172-31-6-15 ec2-user]#
```

15.237.255.120

15.237.255.120:9090/home

Relaunch to update

This site can't be reached

15.237.255.120 refused to connect.

Try:

- Checking the connection
- Checking the proxy and the firewall

ERR_CONNECTION_REFUSED

Details

Reload

04

Configuración de un DAG

```python
from airflow import DAG
from airflow.decorators import task
from airflow.operators.dummy import DummyOperator
from airflow.operators.bash import BashOperator
from datetime import datetime

default_args = {
    'owner': 'ADMIN', # Usuario al que pretenece el DAG
    'start_date': datetime(2022, 12, 1) # date.today(); days_ago(6) # Fecha de comienzo de la programación
    #'email': ['airflow@example.com'],
    #'email_on_failure': False,
    #'email_on_retry': False,
    #'retries': 1,
    #'sla': timedelta(hours=3) # Tiempo máximo de duración del dag
    #'end_date': datetime(2016, 1, 1),
}

dag_args = {
    'dag_id': '01-check-file-dag', # Identificador único
    'schedule_interval': '@daily', # 0 * * * *; None
    'catchup': False, # Si se pone al día con la ejecuciones o no
    'default_args': default_args,
    "doc_md":(
    """
    # 01-check_file_dag
    """),
}
```

Operators: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/operators.html
Args: https://airflow.apache.org/docs/apache-airflow/1.10.10/tutorial.html

```python
with DAG(**dag_args) as dag:

    # Definir una función de tarea usando la TaskFlow API
    @task
    def taskflow_function():
        print("Este es un ejemplo de TaskFlow API")
        return "TaskFlow completado"

    # DummyOperator para iniciar el flujo
    start = DummyOperator(
        task_id='start'
    )

    # BashOperator para ejecutar un comando bash
    bash_task = BashOperator(
        task_id='run_bash_command',
        bash_command='echo "Hello, Airflow from Bash!"'
    )

    # Definir dependencias
    start >> taskflow_function() >> bash_task
```

Task dependencies: https://www.astronomer.io/docs/learn/managing-dependencies

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 17:56 UTC AA

DAG: 01-first-dag

Schedule: @daily Next Run ID: 2024-09-14, 00:00:00 UTC

DAG Docs

01-check_file_dag

15/09/2024 17:56:20 All Run Types All Run States Clear Filters Auto-refresh 25

Press shift + / for Shortcuts

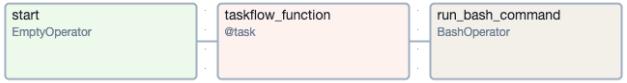
deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status

DAG
01-first-dag

Details Graph Gantt Code Event Log Run Duration Task Duration Calendar

Layout:
Left -> Right

start
taskflow_function
run_bash_command



05

Caso práctico

Caso Práctico Juegos Olímpicos

ETL

01

get_data

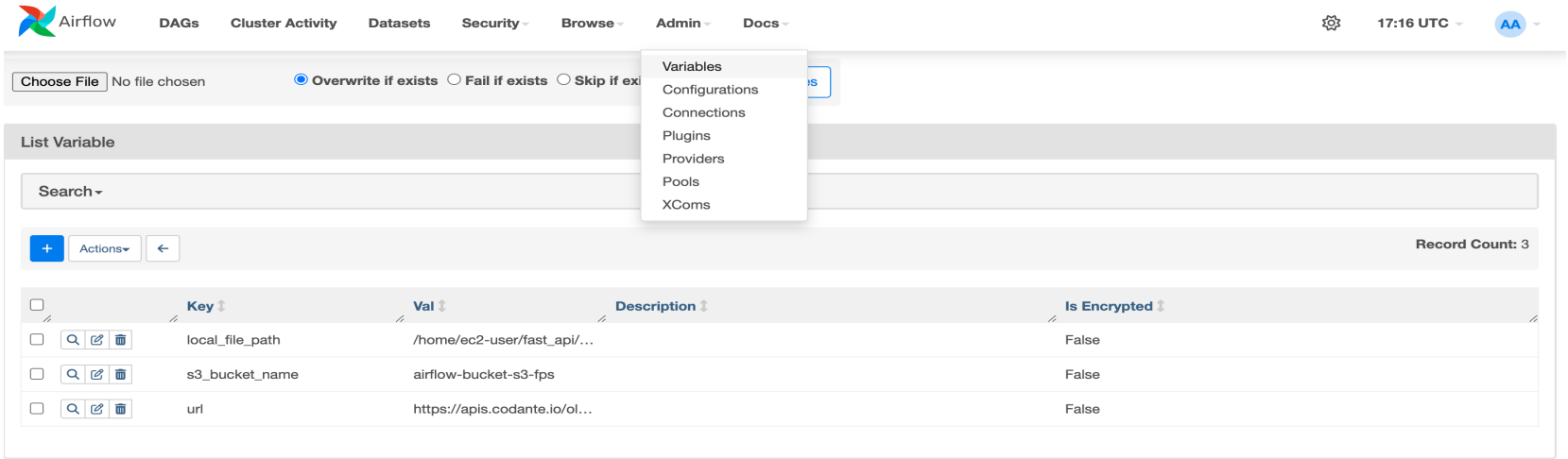
API: https://apis.codante.io/olympic-games/events

Generar diccionario con los datos de la API

- PythonOperator
- Variables
- **XcomPull**

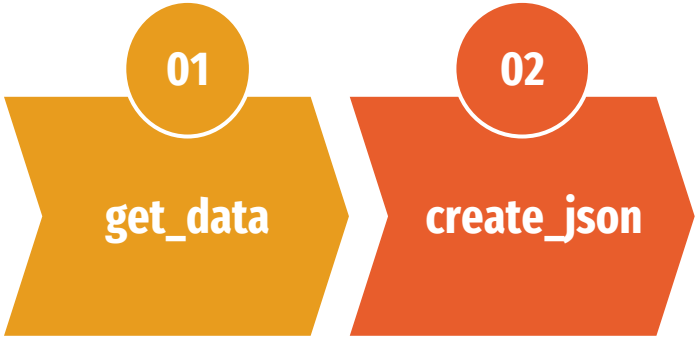
```python
# Function to get data from the URL
def request_url():
    url = Variable.get("url")  # Use Airflow Variable for URL
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
        raise ValueError(f"Failed to retrieve data. Status code: {response.status_code}")


# Task 1: Get Data from the URL
    get_data = PythonOperator(
        task_id='get_data',
        python_callable=request_url,
        provide_context=False
    )
```



Caso Práctico Juegos Olímpicos

ETL



Generar diccionario con los dato de la API

- PythonOperator
- Variables
- **XcomPush**

Transformar datos

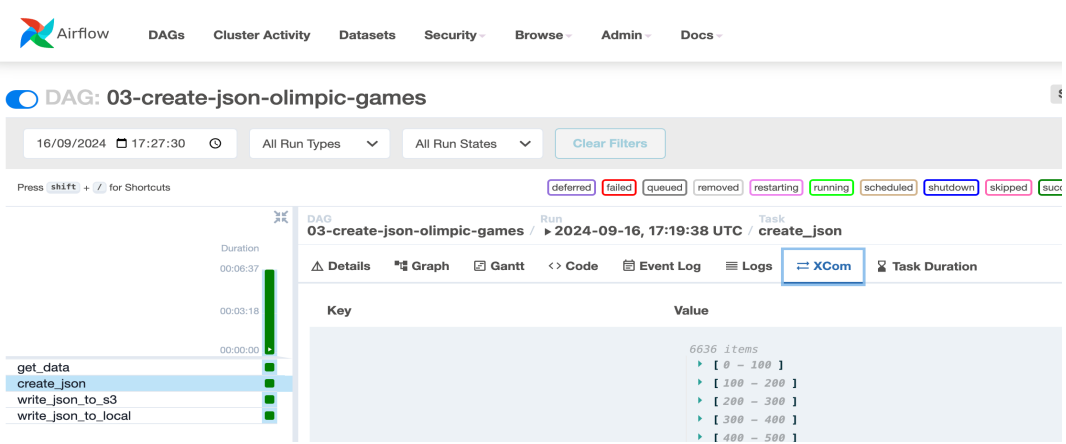
- **XcomPull**

```python
# Function to create JSON from the data
def create_json_olympic_games(**kwargs):
    data = kwargs['ti'].xcom_pull(task_ids='get_data')
    total_pages = data['meta']['last_page']
    results_dict = {}

    for page in range(1, total_pages + 1):
        new_url = Variable.get("url").replace("{page_id}", str(page))
        page_data = requests.get(new_url).json()
        events_list = page_data['data']

        for event in events_list:
            ....
            ....
            ....

# Task 2: Create JSON from the Data
    create_json = PythonOperator(
        task_id='create_json',
        python_callable=create_json_olympic_games,
        provide_context=True
    )
```



Caso Práctico Juegos Olímpicos

ETL

01

get_data

Generar diccionario con los dato de la API

- PythonOperator
- Variables
- **XcomPush**

02

create_json

Transformar datos

- **XcomPull**

03

write_json_to_local

Escribir json en local

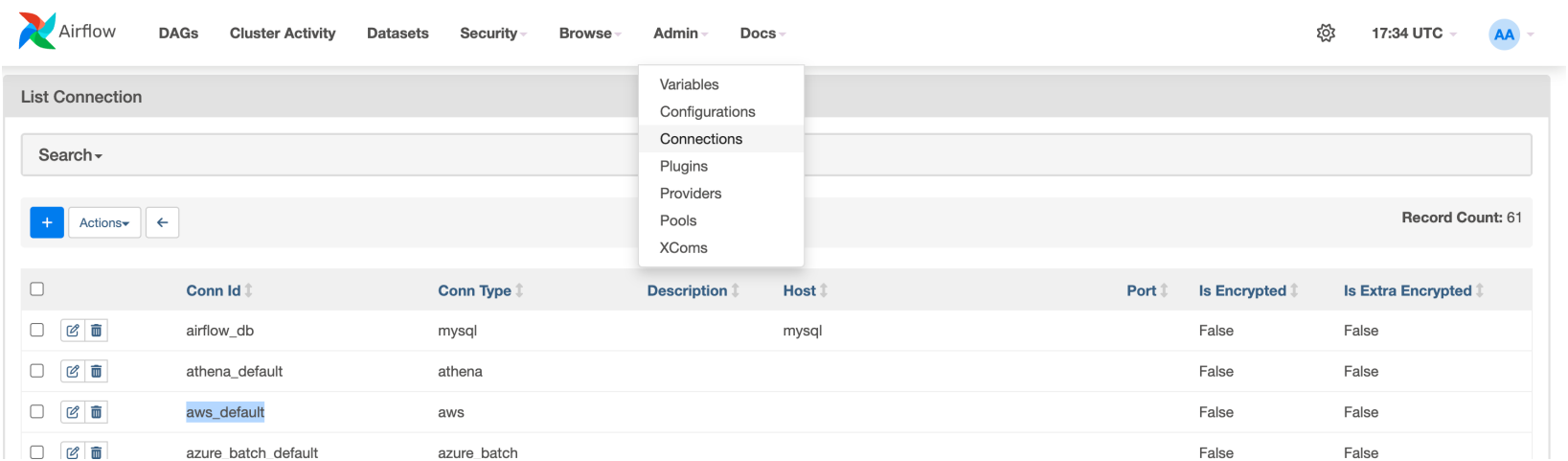
- PythonOperator
- Variables
- **XcomPush**

04

write_json_to_s3

Conectar a Amazon S3 y escribir los datos en un bucket

- **Conexiones**
- **S3CreateObjectOperator**




```python
# Task 3: Write the JSON to S3
write_json_to_s3 = S3CreateObjectOperator(
    task_id="write_json_to_s3",
    aws_conn_id='aws_default',
    s3_bucket=Variable.get("s3_bucket_name"),  # Use Airflow Variable for S3 bucket name
    s3_key='data/olympic_results.json',
    data="{{ ti.xcom_pull(task_ids='create_json') }}",
    replace=True
)
```

Caso Práctico Juegos Olímpicos

ETL

01 — get_data

Generar diccionario con los dato de la API

- PythonOperator
- Variables
- **XcomPush**

02 — create_json

Transformar datos

- **XcomPull**

03 — write_json_to_local

Escribir json en local

- PythonOperator
- Variables
- **XcomPush**

04 — write_json_to_s3

Conectar a Amazon S3 y escribir los datos en un bucket

- **Conexiones**
- **S3CreateObjectOperator**

05 — launch_fastapi

Arracar FastAPI

- **BashOperator**

```python
# Task: Launch FastAPI using uvicorn
launch_fastapi = BashOperator(
    task_id='launch_fastapi',
    bash_command='cd /home/ec2-user/fast_api && nohup uvicorn main:app --host 0.0.0.0 --port 8000 --reload &',
    execution_timeout=None  # Disable timeout to keep FastAPI running
)
```

Juegos Olímpicos 2024

PARIS 2024

País: Argentina Deporte: Football Categoria: Men Filtrar

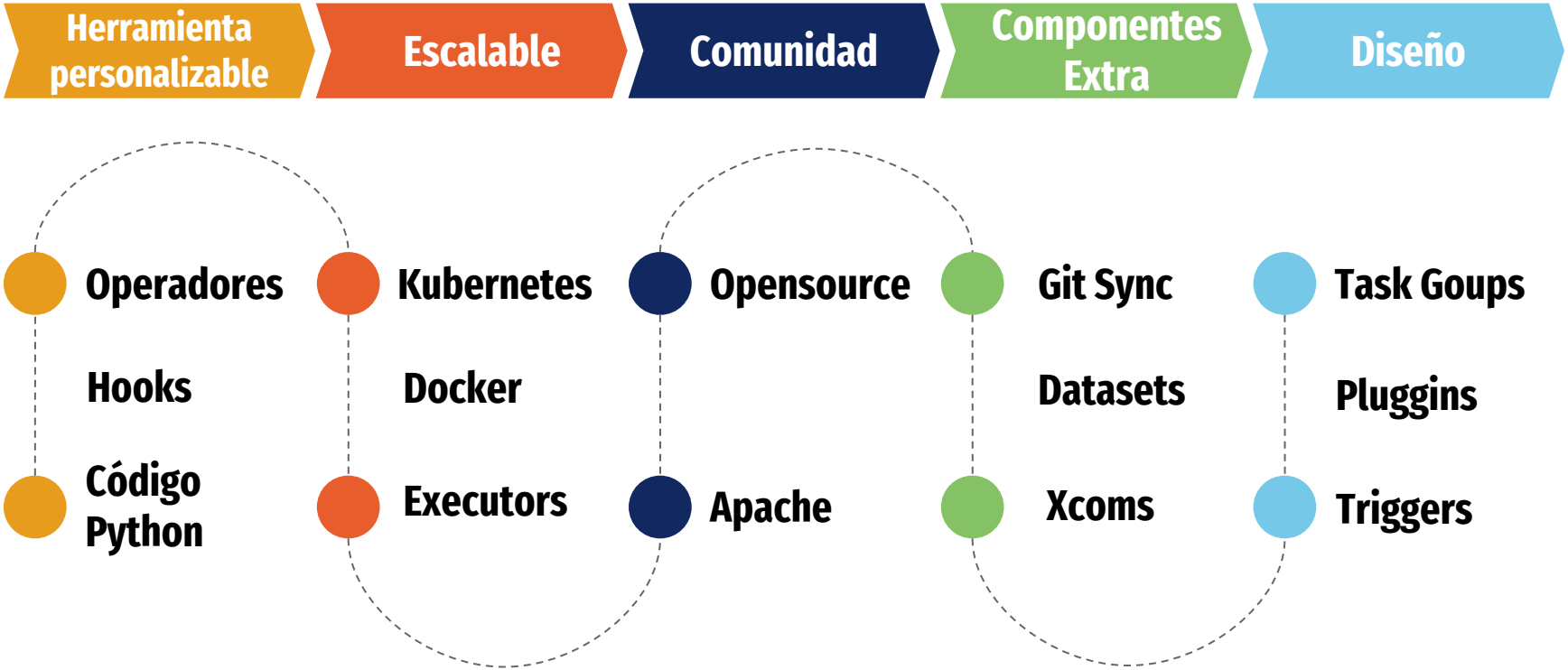
Filtered Events

Event Name	Venue	Competitors	Result
Men's Group B	Geoffroy-Guichard Stadium	Argentina-Marrocos	1-2
Men's Group B	Lyon Stadium	Argentina-Iraque	3-1
Men's Group B	Lyon Stadium	Ucrânia-Argentina	0-2
Men's Quarter-final	Bordeaux Stadium	-	-

05

Conclusión

¿Qué hemos aprendido?



Q&A

Fin →

Muchas gracias