Escuela de Datos Vivos

Curso Data Engineer: Creando un pipeline de datos

MÓDULO E - Clase 7

Agenda

- Transformaciones con SCALA
- Transformaciones en cloud (Spark)
- Orquestación
- Ejercitación

Transformaciones SCALA

Pyspark - Scala

Para ingresar a Spark Python

pyspark

Para ingresar a Spark Scala

spark-shell

Para salir de Spark Python

exit()

Para salir de Spark Scala

:q

Pyspark - Scala

Extract

```
scala> val df = spark.read.option("header", "true").csv("/ingest/yellow_tripdata_2021-01.csv")
df: org.apache.spark.sql.DataFrame = [VendorID: string, tpep_pickup_datetime: string ... 16 more fields]
```

Pyspark - Scala

Transform

```
scala> val df_cast = df.select( df("tpep_pickup_datetime").cast("date"), df("passenger_count").cast("int"), df("trip_distance").cast("float") )
df_cast: org.apache.spark.sql.DataFrame = [tpep_pickup_datetime: date, passenger_count: int ... 1 more field]

scala> val df_trans = df_cast.filter( (df_cast("passenger_count") > 1) && (df("trip_distance") > 10) )
df_trans: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [tpep_pickup_datetime: date, passenger_count: int ... 1 more field]
```

Pyspark - Scala

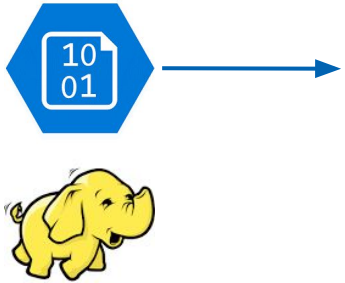
Load

```
scala> df_trans.write.insertInto("tripsdb.trips")
```

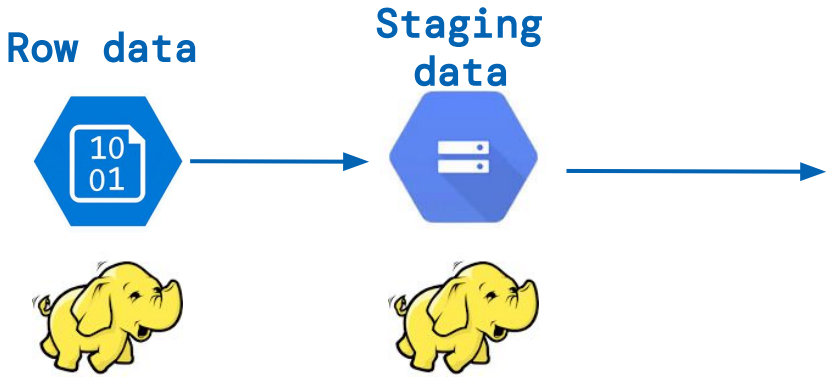
Transformaciones en cloud

Data Proc

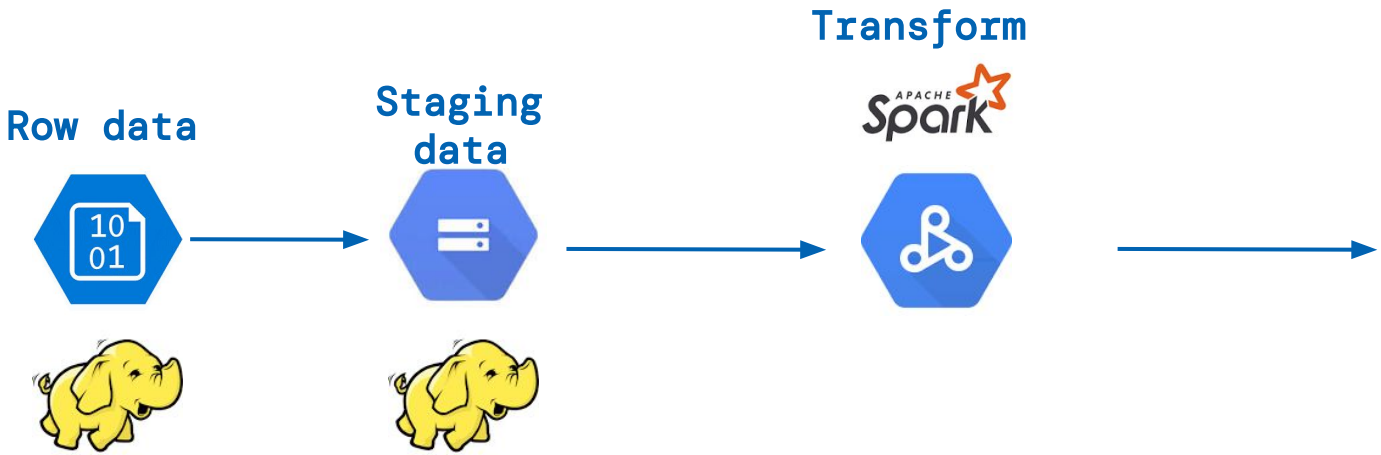
Row data



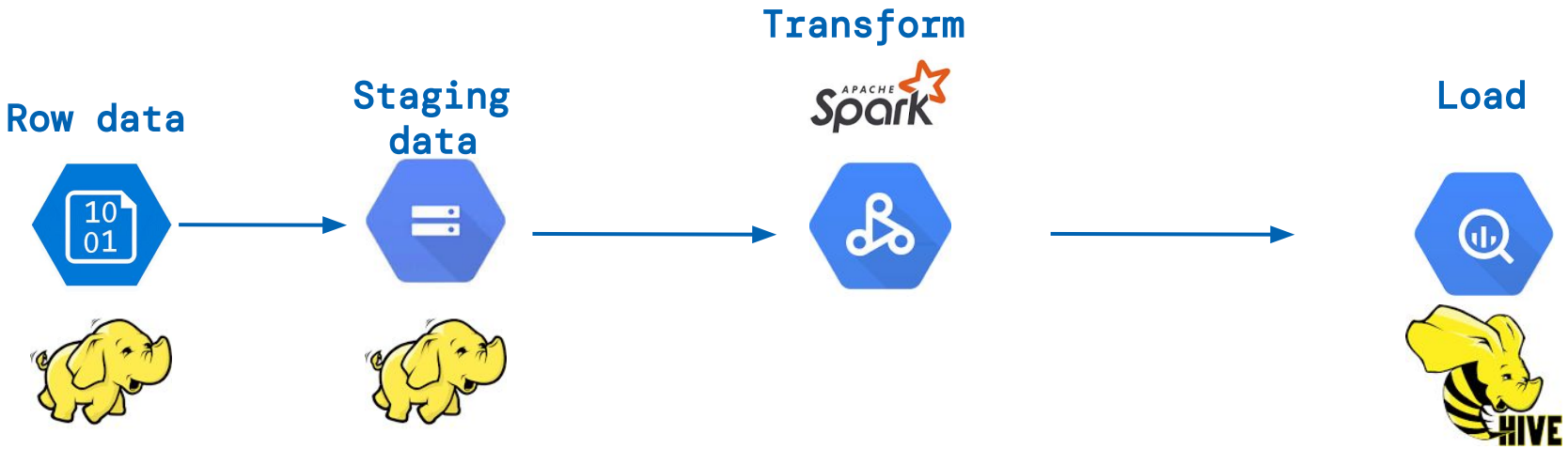
Data Proc



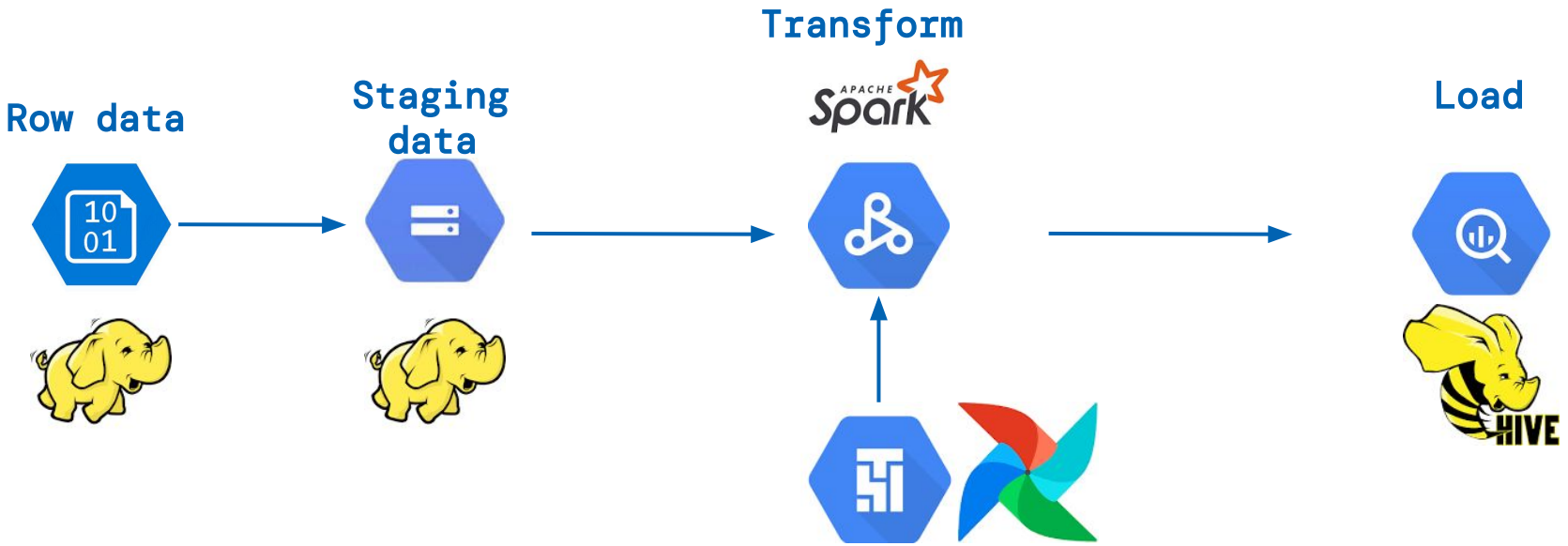
Data Proc



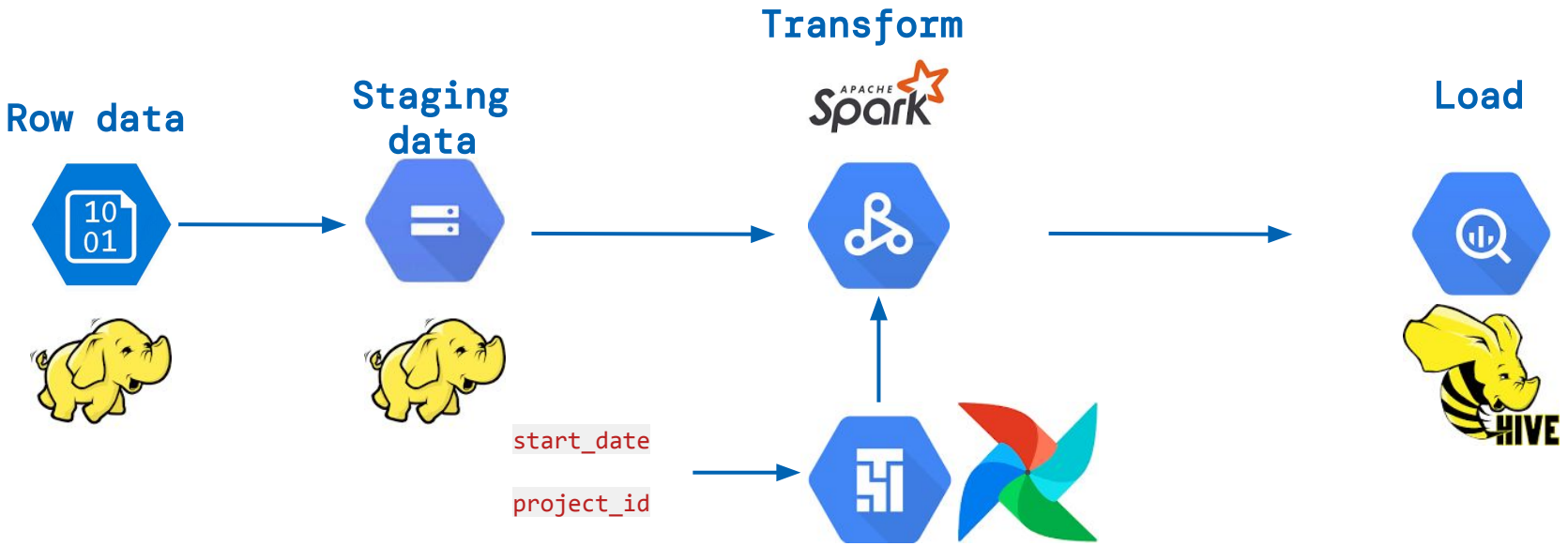
Data Proc


Row data
Staging data
Transform
Load
10
01
APACHE
Spark
HIVE

Data Proc

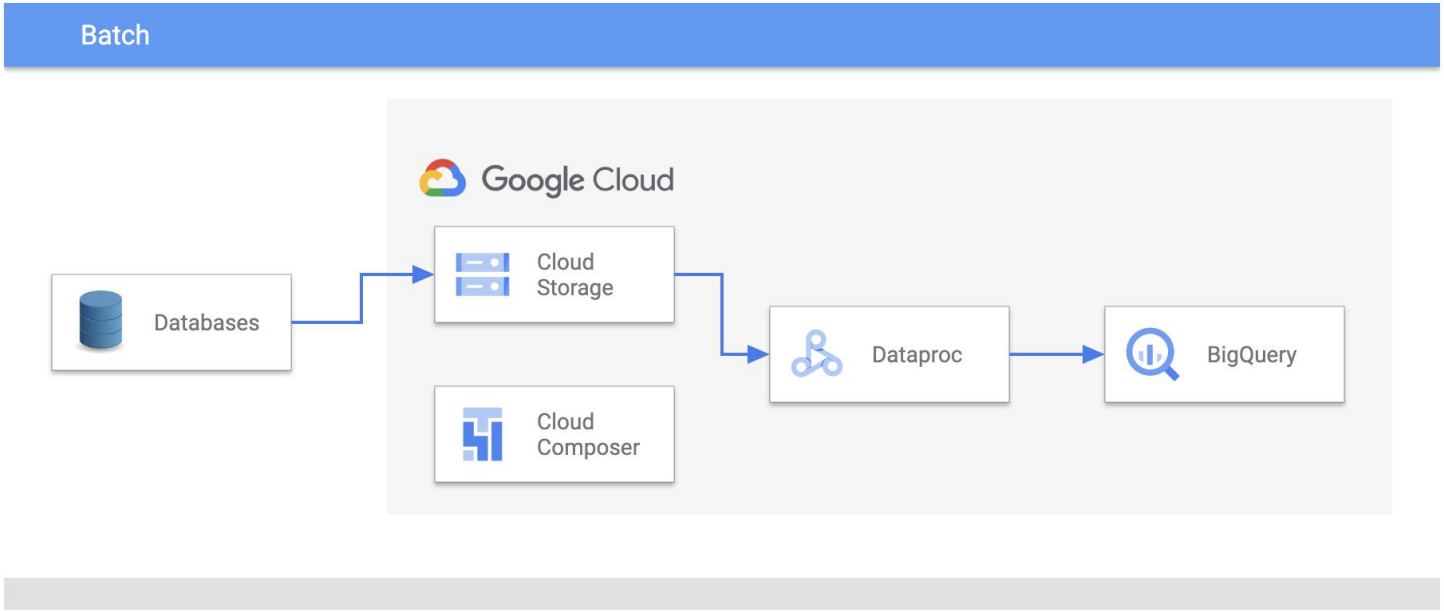

Row data
Staging data
Transform
Load
APACHE Spark
10
01
HIVE

Data Proc

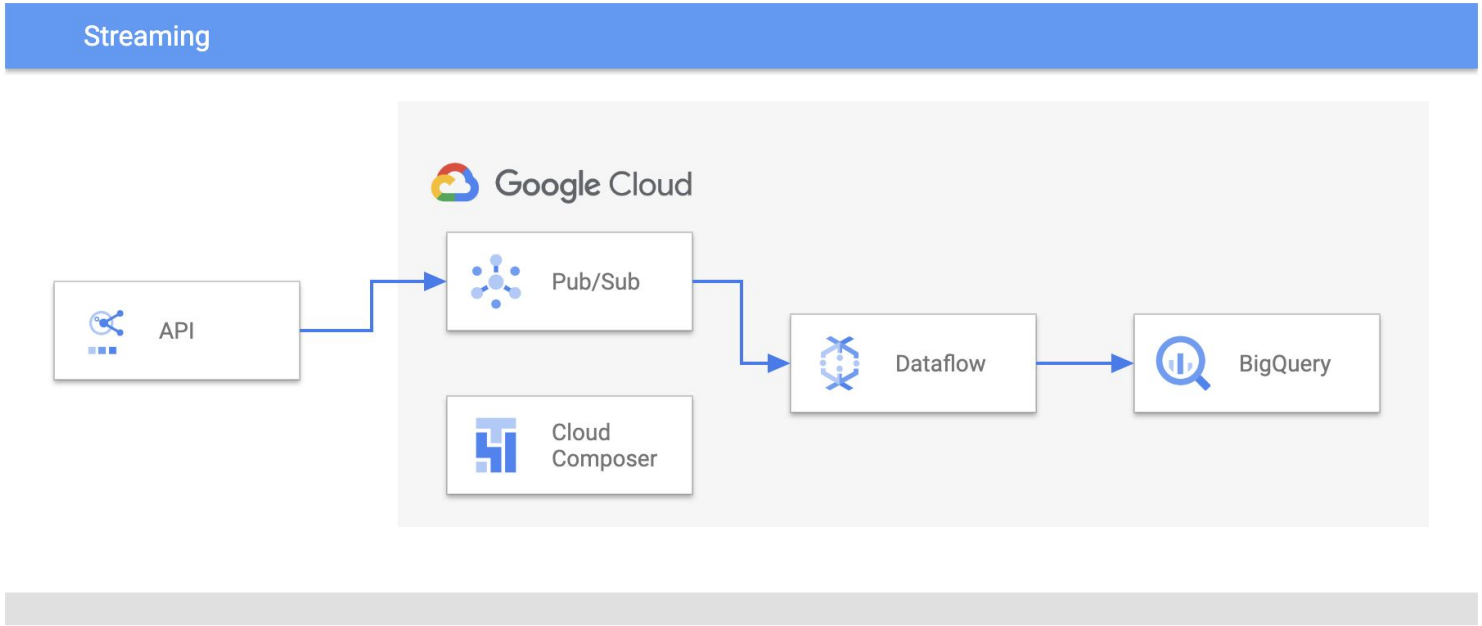

Transform
Row data
Staging data
Load
APACHE Spark
10
01
HIVE
start_date
project_id

Orquestación

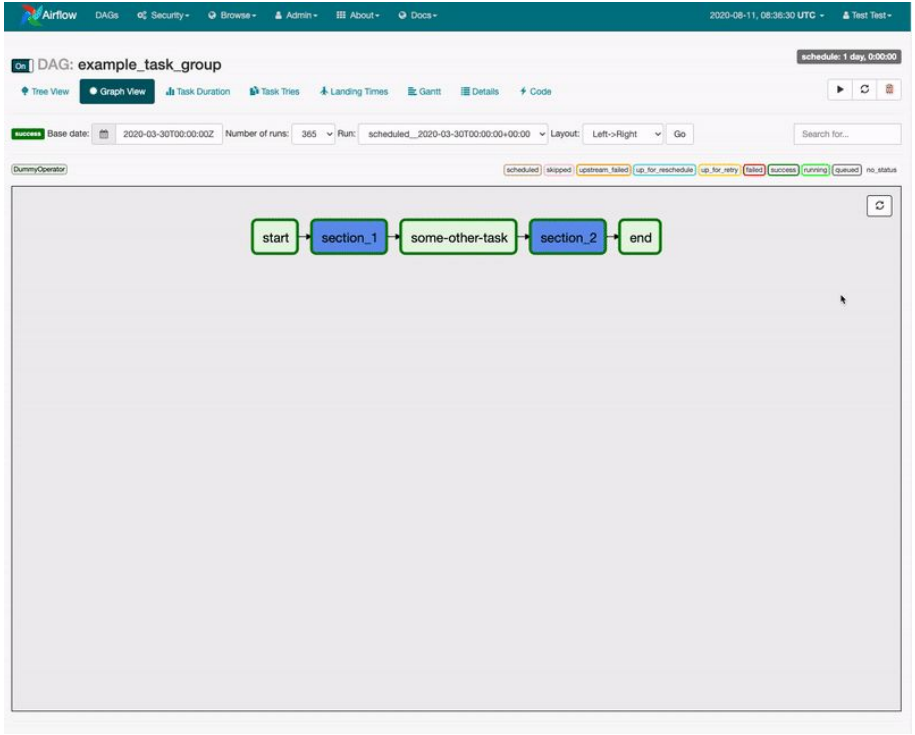
Batch process



Streaming process



Apache Airflow



Apache Airflow

Importing Modules

An Airflow pipeline is just a Python script that happens to define an Airflow DAG object. Let's start by importing the libraries we will need.

airflow/example_dags/tutorial.py [source]

```
from datetime import datetime, timedelta
from textwrap import dedent

# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG

# Operators; we need this to operate!
from airflow.operators.bash import BashOperator
```

Apache Airflow

Default Arguments

We're about to create a DAG and some tasks, and we have the choice to explicitly pass a set of arguments to each task's constructor (which would become redundant), or (better!) we can define a dictionary of default parameters that we can use when creating tasks.

airflow/example_dags/tutorial.py [source]

```
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
default_args={
    "depends_on_past": False,
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'sla_miss_callback': yet_another_function,
    # 'trigger_rule': 'all_success'
},
```

Apache Airflow

Instantiate a DAG

We'll need a DAG object to nest our tasks into. Here we pass a string that defines the `dag_id`, which serves as a unique identifier for your DAG. We also pass the default argument dictionary that we just defined and define a `schedule` of 1 day for the DAG.

airflow/example_dags/tutorial.py [source]

```
with DAG(
    "tutorial",
    # These args will get passed on to each operator
    # You can override them on a per-task basis during operator initialization
    default_args={
        "depends_on_past": False,
        "email": ["airflow@example.com"],
        "email_on_failure": False,
        "email_on_retry": False,
        "retries": 1,
        "retry_delay": timedelta(minutes=5),
        # 'queue': 'bash_queue',
        # 'pool': 'backfill',
        # 'priority_weight': 10,
        # 'end_date': datetime(2016, 1, 1),
        # 'wait_for_downstream': False,
        # 'sla': timedelta(hours=2),
        # 'execution_timeout': timedelta(seconds=300),
        # 'on_failure_callback': some_function,
        # 'on_success_callback': some_other_function,
        # 'on_retry_callback': another_function,
        # 'sla_miss_callback': yet_another_function,
        # 'trigger_rule': 'all_success'
    },
    description="A simple tutorial DAG",
    schedule=timedelta(days=1),
    start_date=datetime(2021, 1, 1),
    catchup=False,
    tags=["example"],
) as dag:
```

Apache Airflow

Operators

An Operator is conceptually a template for a predefined Task, that you can just define declaratively inside your DAG:

```
with DAG("my-dag") as dag:
    ping = SimpleHttpOperator(endpoint="http://example.com/update/")
    email = EmailOperator(to="admin@example.com", subject="Update complete")

    ping >> email
```

Airflow has a very extensive set of operators available, with some built-in to the core or pre-installed providers. Some popular operators from core include:

- **`BashOperator`** - executes a bash command
- **`PythonOperator`** - calls an arbitrary Python function
- **`EmailOperator`** - sends an email

Apache Airflow

If the operator you need isn't installed with Airflow by default, you can probably find it as part of our huge set of community provider packages. Some popular operators from here include:

- `SimpleHttpOperator`
- `MySqlOperator`
- `PostgresOperator`
- `MsSqlOperator`
- `OracleOperator`
- `JdbcOperator`
- `DockerOperator`
- `HiveOperator`
- `S3FileTransformOperator`
- `PrestoToMySqlOperator`
- `SlackAPIOperator`

Apache Airflow

Tasks

To use an operator in a DAG, you have to instantiate it as a task. Tasks determine how to execute your operator's work within the context of a DAG.

In the following example, we instantiate the BashOperator as two separate tasks in order to run two separate bash scripts. The first argument for each instantiation, `task_id`, acts as a unique identifier for the task.

airflow/example_dags/tutorial.py [source]

```
t1 = BashOperator(
    task_id="print_date",
    bash_command="date",
)

t2 = BashOperator(
    task_id="sleep",
    depends_on_past=False,
    bash_command="sleep 5",
    retries=3,
)
```

Ejercicio

Apache Airflow

http://<direccion ip>:8010

Username: airflow
Password: airflow

Apache Airflow

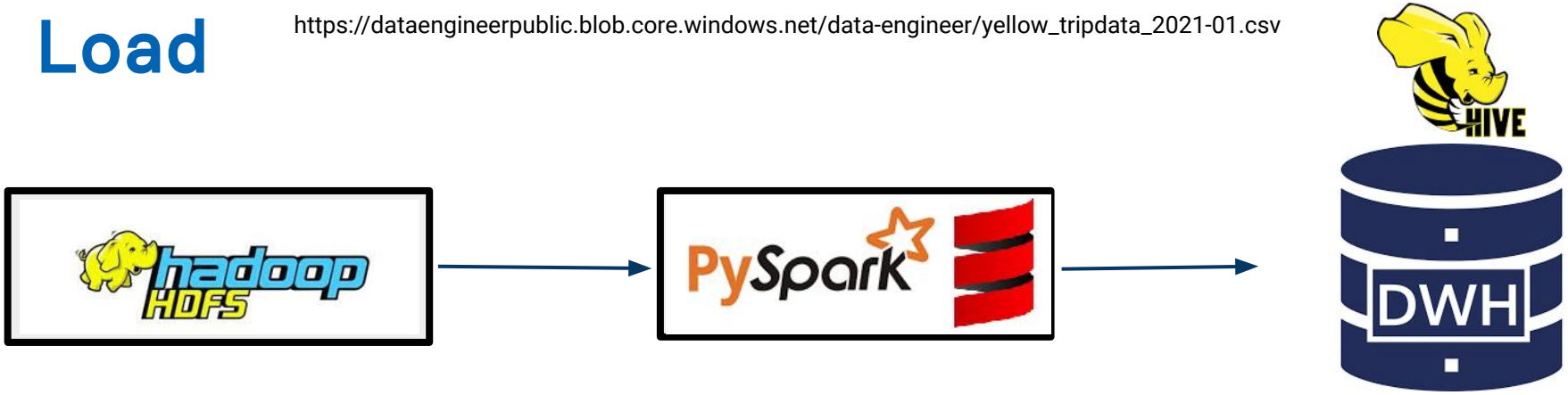
ejercicio

Crear un pipeline a través de un DAG, creando las tareas:

- Ingest
- Transform
- Load

Load

https://dataengineerpublic.blob.core.windows.net/data-engineer/yellow_tripdata_2021-01.csv



Extract **Transform** **Load**

Orchestration

Preguntas?

Gracias