HandsOn – **Apache Airflow**

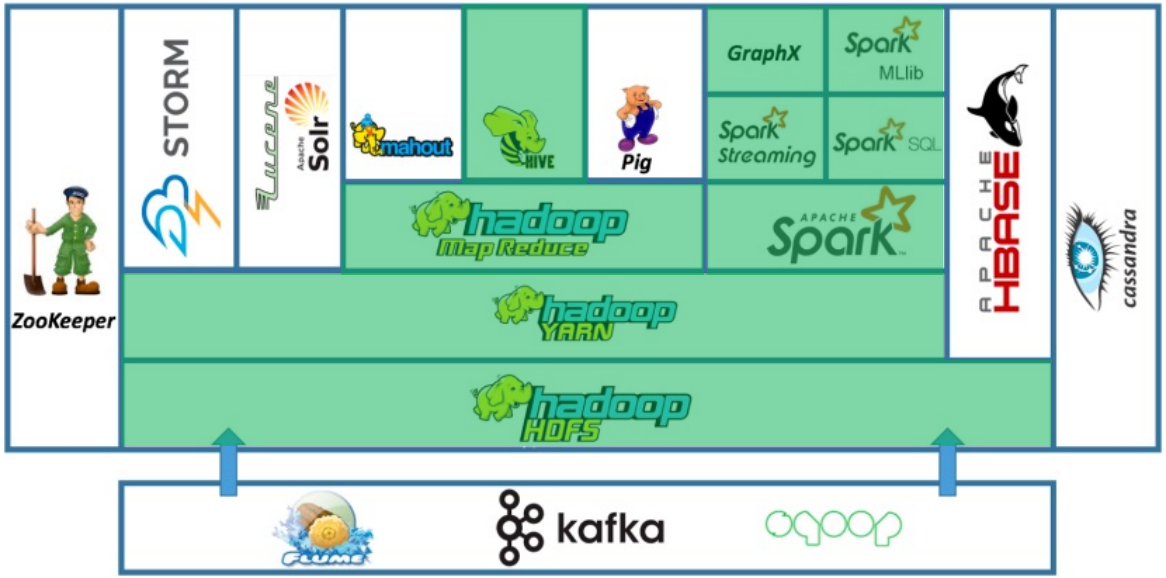
A quick Introduction to ETL Workflow with Apache Airflow

The Hadoop Ecosystem

pentaho

Airflow

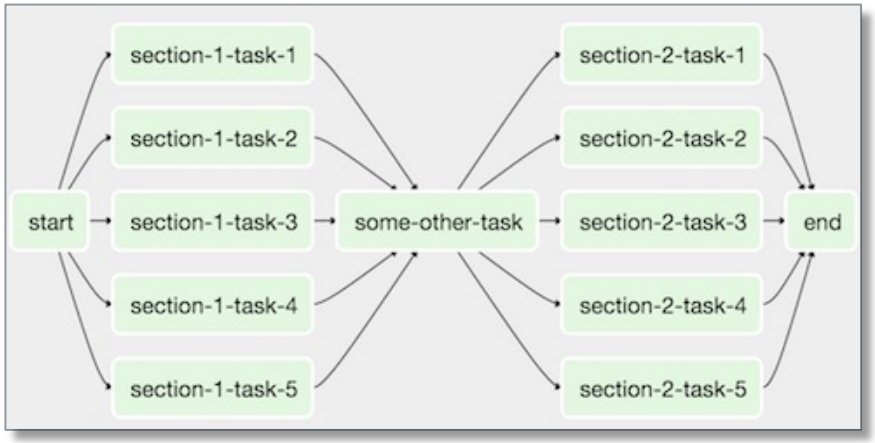
Today's (exercise) focus



Airflow

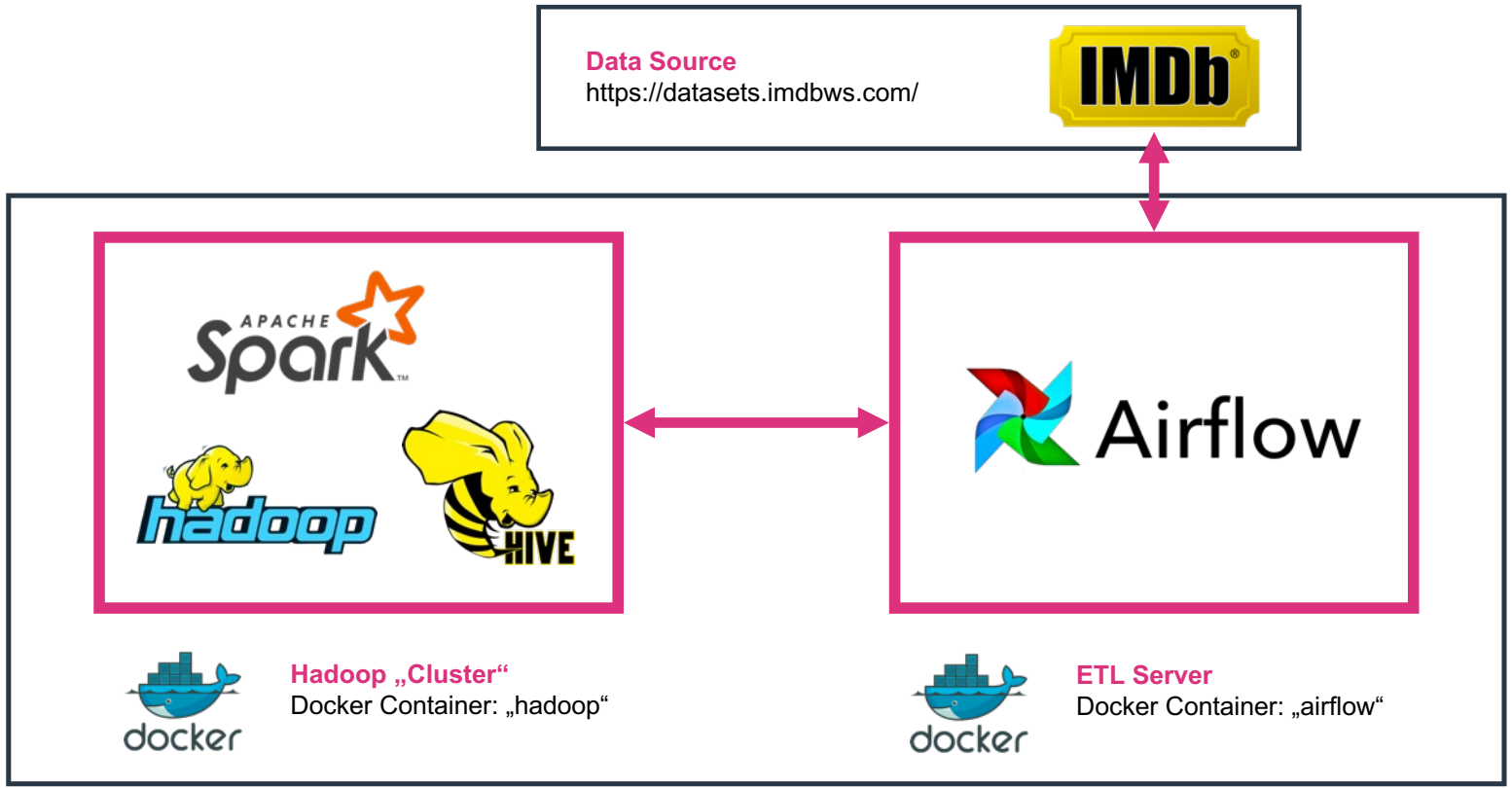
- Apache Open Source Project
- Model Task (e.g. ETL) Workflows
- Python Code Base
- Scheduling, Queues and Pools
- Cluster Ready
- Web UI

- DAG = **Directed Acyclic Graph**



→ a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies.

What do we want to do?



Remove Previously created Docker Container

1. Stop and Remove Images:

(This will delete all files you created within previous exercise.
Save them somewhere outside the docker container, if you haven‘t done yet.)

```
docker stop hadoop
docker rm hadoop
```

```
docker stop pentaho
docker rm pentaho
```


Start Hadoop/Hive/Spark Docker Container

1. Pull Docker Image:

```
docker pull marcelmittelstaedt/spark_base:latest
```

2. Start Docker Image:

```
docker run -dit --name hadoop \
          -p 8088:8088 -p 9870:9870 -p 9864:9864 -p 10000:10000 \
          -p 8032:8032 -p 8030:8030 -p 8031:8031 -p 9000:9000 \
          -p 8888:8888 --net bigdatanet \
          marcelmittelstaedt/spark_base:latest
```

3. Wait till first Container Initialization finished:

```
docker logs hadoop

[…]
Stopping nodemanagers
Stopping resourcemanager
Container Startup finished.
```


Start Hadoop/Hive/Spark Docker Container

4. Get into Docker container:

```
docker exec -it hadoop bash
```

5. Switch to hadoop user:

```
sudo su hadoop
```

```
cd
```

6. Start Hadoop Cluster:

```
start-all.sh
```

7. Start HiveServer2:

```
hiveserver2
```


Start Hadoop/Hive/Spark Docker Container

8. Start HiveServer2:

```
hive/bin/hiveserver2

2018-10-02 16:19:08: Starting HiveServer2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4
j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/hadoop/share/hadoop/common/lib/slf4j-log4j12-1
.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = b8d1efb3-fc8c-4ec8-bdf0-6a9a41e2ddaa
Hive Session ID = 32503981-a5fd-497e-b887-faf3ec1e686e
Hive Session ID = 00f7eab4-5a29-4ce4-ad97-e90904d9206f
Hive Session ID = 100e54c5-14c6-4acc-b398-040152b08ebf
[…]
```

Start ETL (Airflow) Docker Container

1. Pull Docker Image:

```
docker pull marcelmittelstaedt/airflow:latest
```

2. Start Docker Image:

```
docker run -dit --name airflow \
           -p 8080:8080 \
           --net bigdatanet \
           marcelmittelstaedt/airflow:latest
```

3. Wait till first Container Initialization finished:

```
docker logs airflow

[…]
Successfully added `conn_id`=spark : spark://:@yarn:

Container Startup finished.
```


Start ETL (Airflow) Docker Container

4. Get into Docker container:

```
docker exec -it airflow bash
```

5. Switch to airflow user:

```
sudo su airflow
```

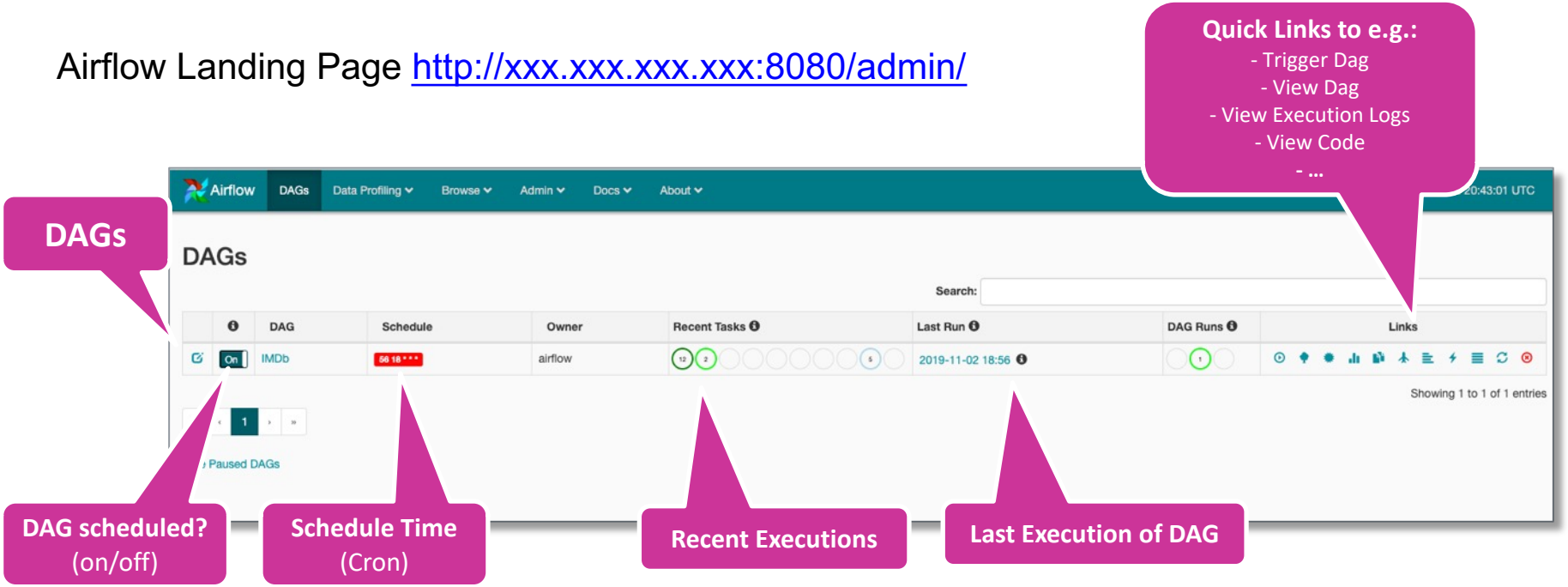
```
cd
```


Exercises Preparation II

Airflow First Steps/Dag

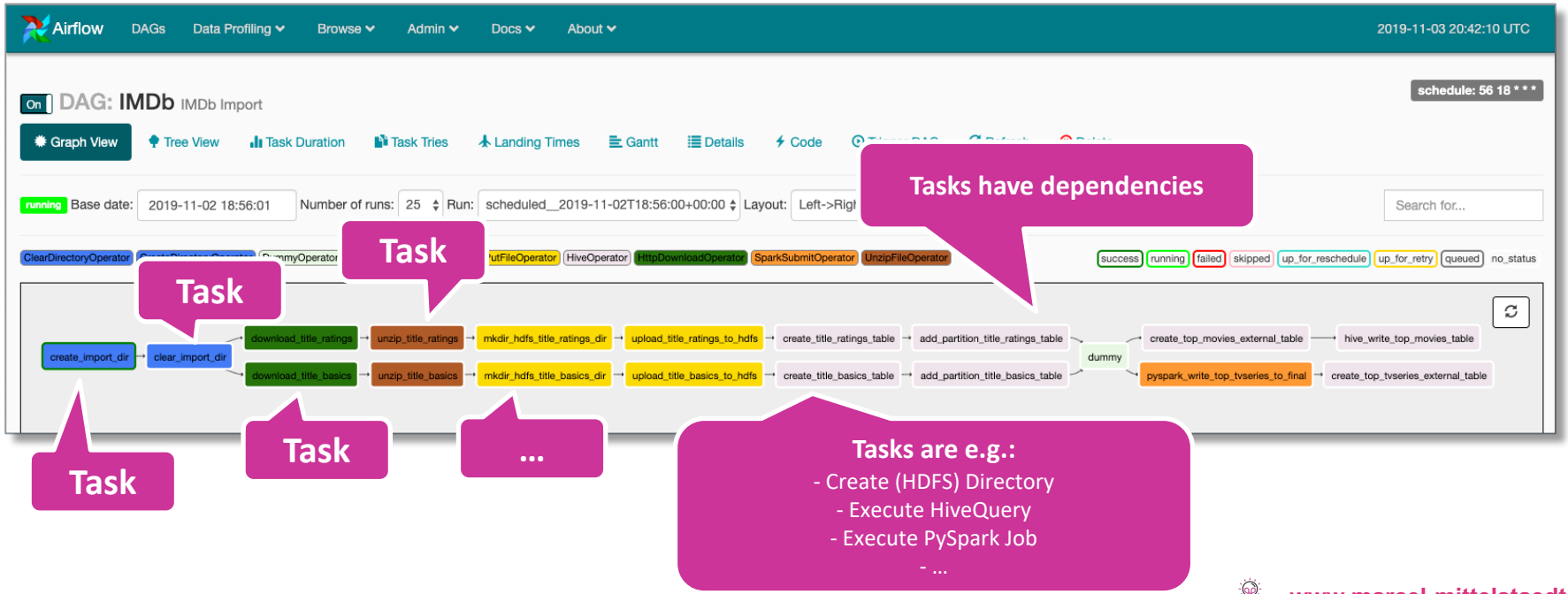
Spoon Interface

Airflow Landing Page http://xxx.xxx.xxx.xxx:8080/admin/



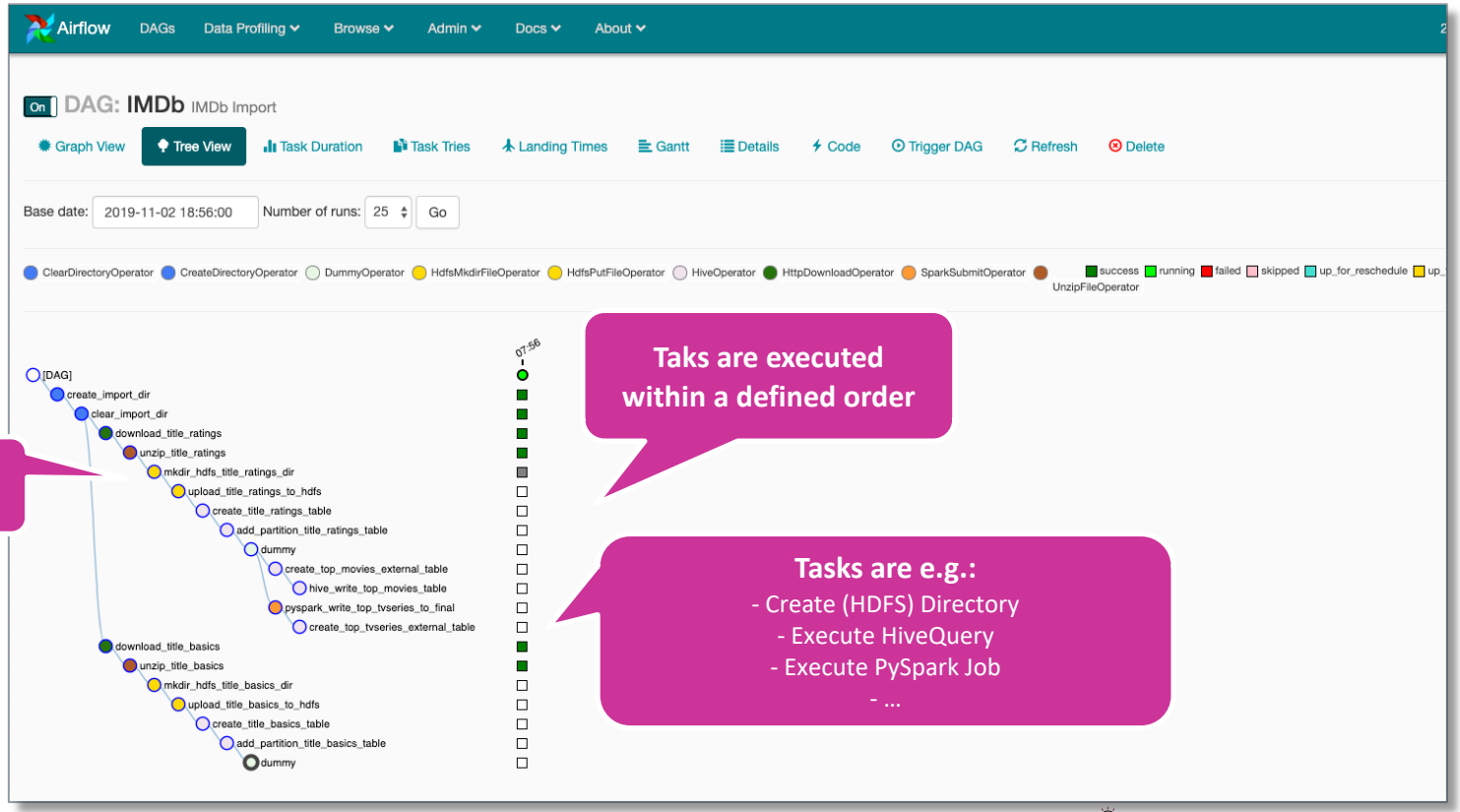
Spoon Interface

Graph View:



Spoon Interface

Tree View:



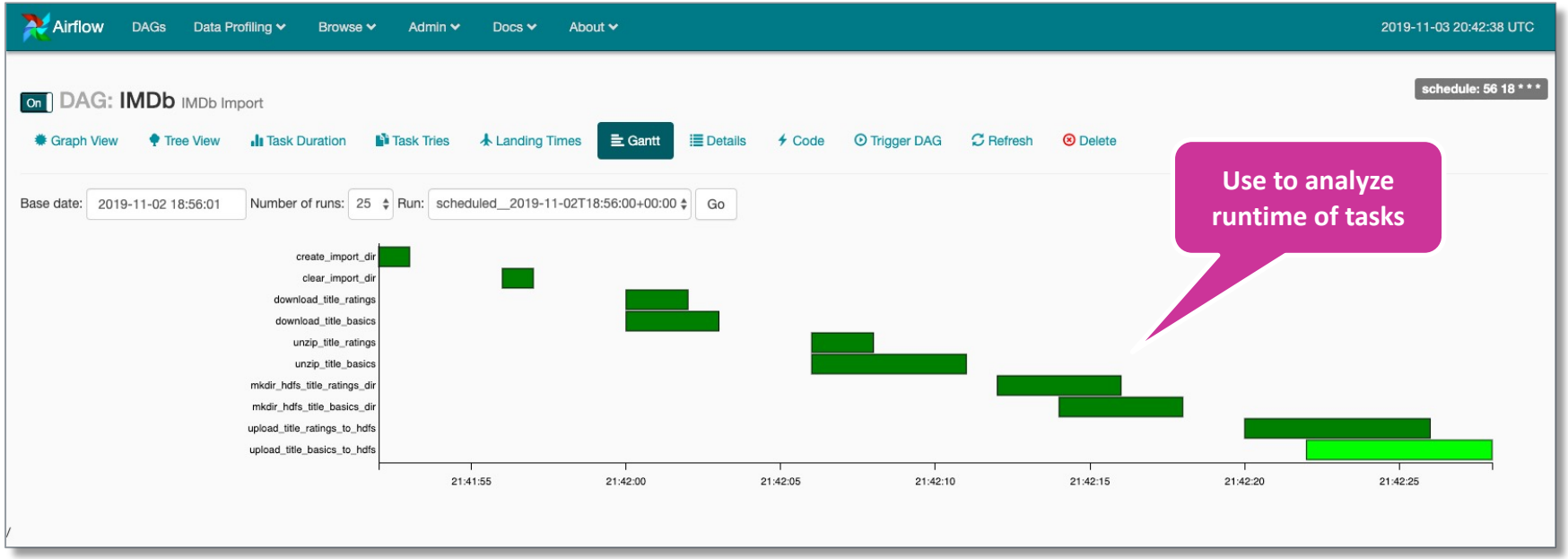
Tasks have dependencies

Taks are executed within a defined order

Tasks are e.g.:
- Create (HDFS) Directory
- Execute HiveQuery
- Execute PySpark Job
- ...

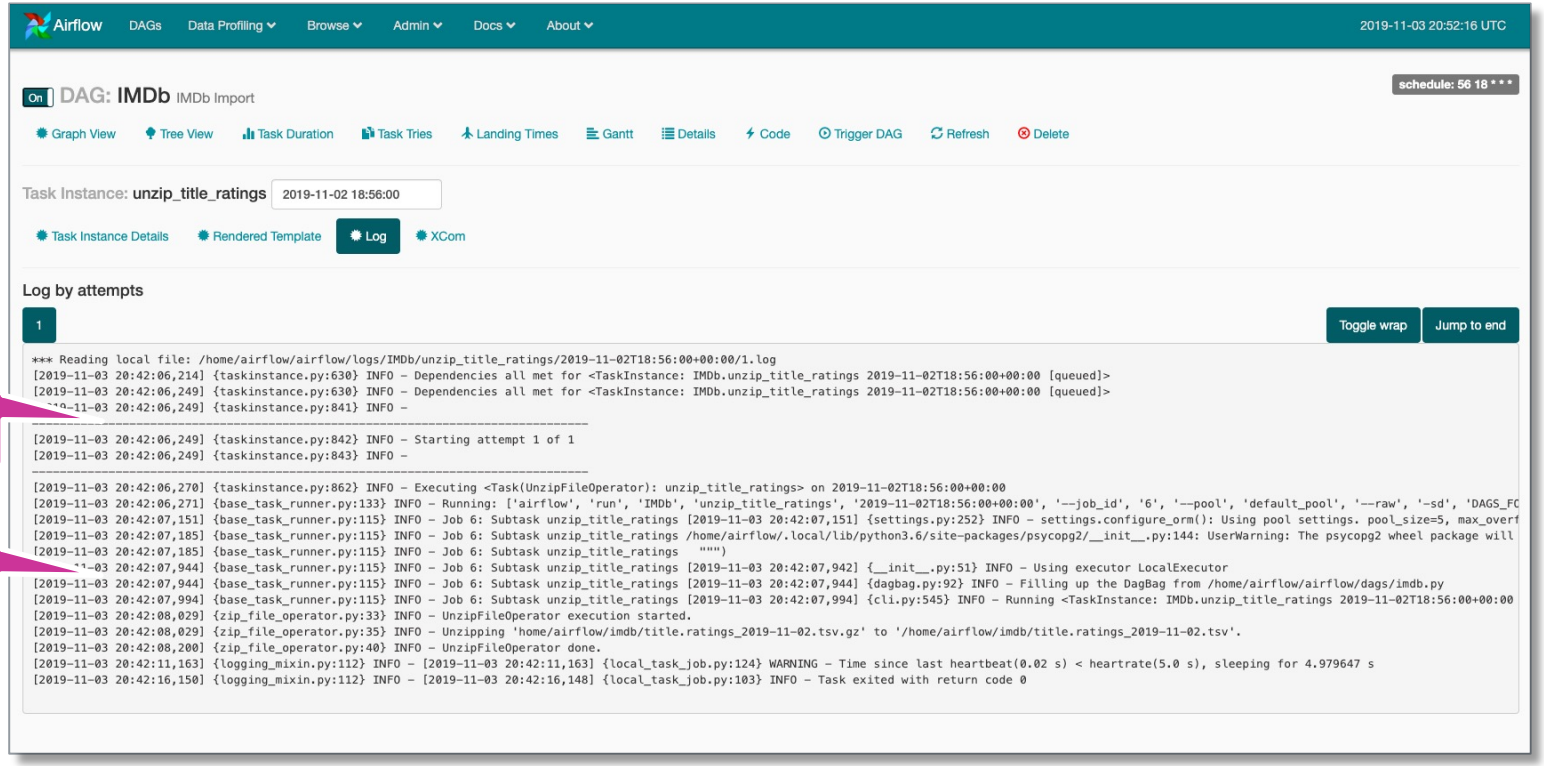
Spoon Interface

Gantt View:



Spoon Interface

Logs View:



Use for Debugging

Each Task has it's own Log

Create Simple Example DAG

1. Open Dag File:

```
vi /home/airflow/airflow/dags/example_dag.py
```

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta

args = {
    'owner': 'airflow'
}

dag = DAG('ExampleDAG', default_args=args, description='Simple Example DAG',
          schedule_interval='56 18 * * *',
          start_date=datetime(2019, 10, 16), catchup=False, max_active_runs=1)

task_1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

task_2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)

task_1 >> task_2
```

DAG Definition, e.g.
- Name
- Schedule Interval (Cron)
- Description
- Start date
- …

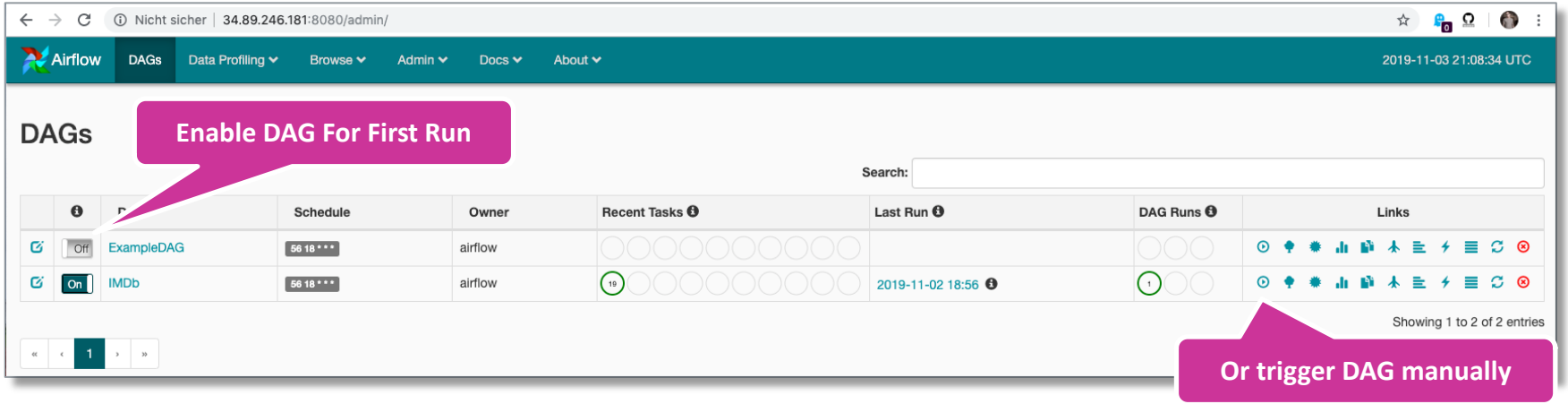
Task 1

Task 2

Task Execution Order
task_2 is dependent on **task_1**

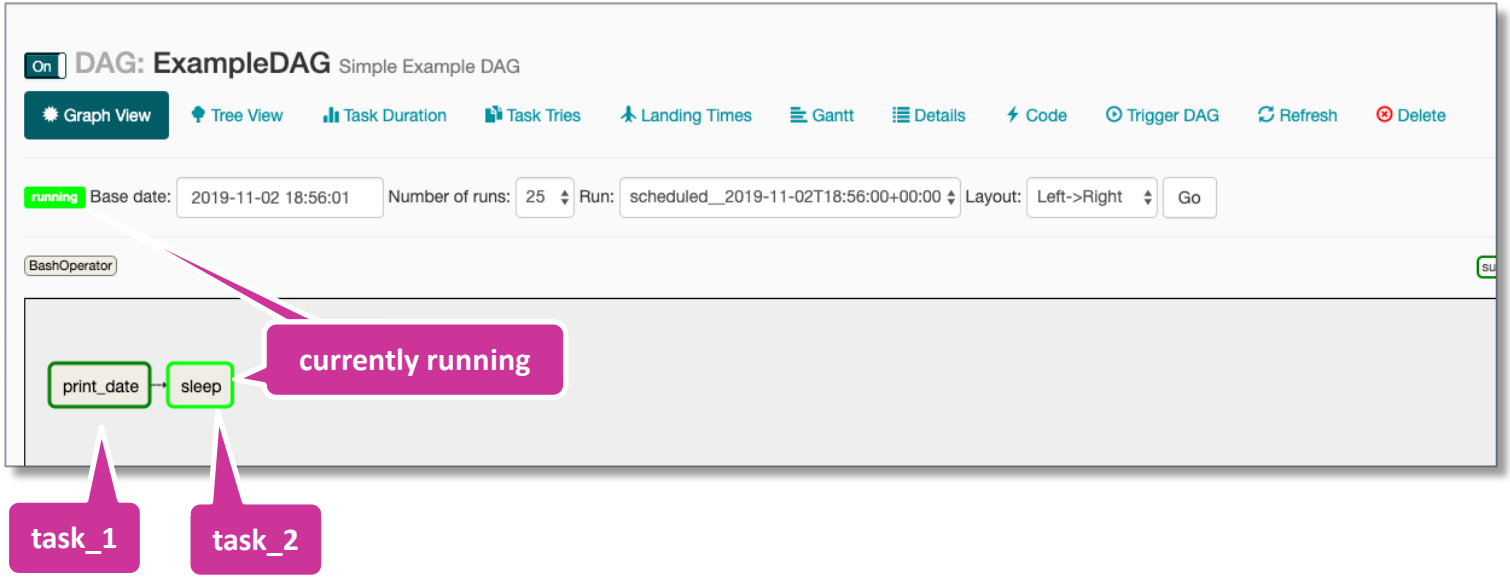
Spoon Interface

Execute DAG:



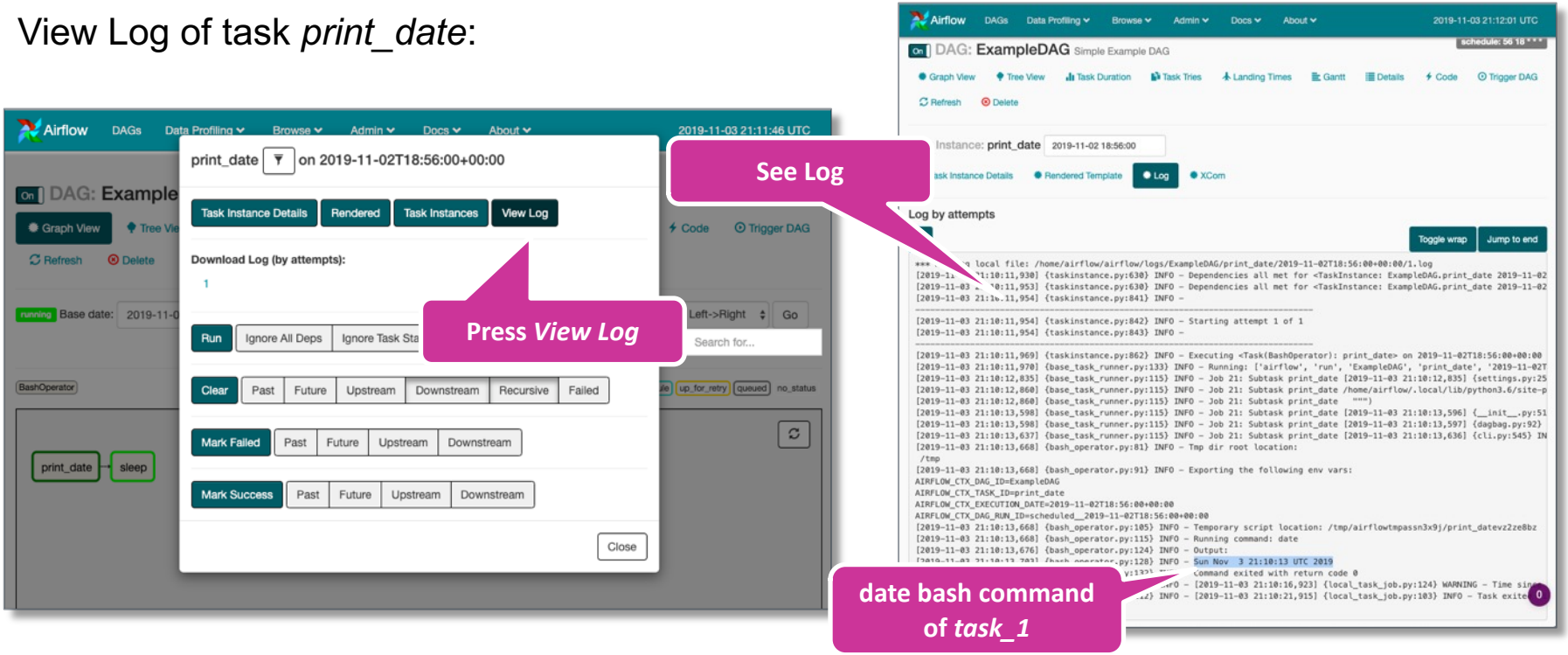
Spoon Interface

See executing DAG:



Spoon Interface

View Log of task *print_date*:

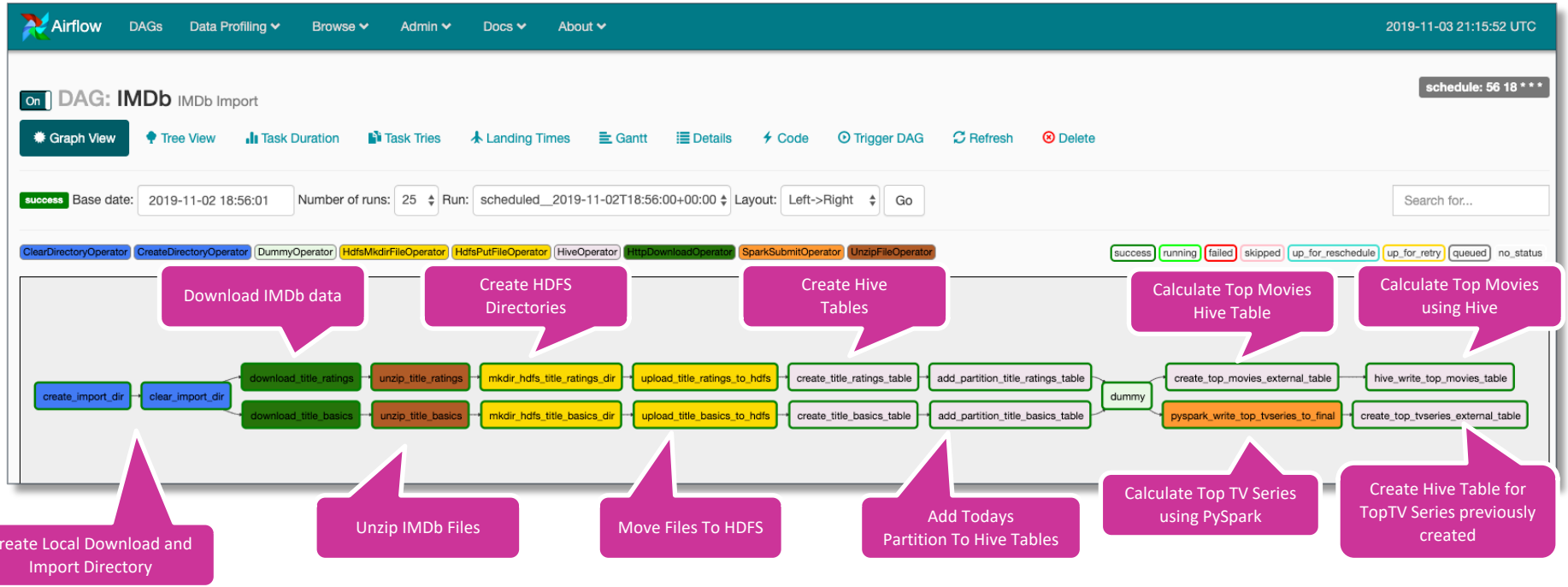


Exercises II

Use Apache Airflow to solve exercises based on IMDb data

Spoon Interface

See executing DAG:



Pentaho Data Integration Exercises – **IMDB**

1. Execute IMDb DAG

2. Use Airflow and previous IMDb DAG to do following changes (`vi /home/airflow/airflow/dags/imdb.py`):

    a) **Extend Airflow IMDb DAG** to also download ***name.basics.tsv.gz***

    b) **Extend Airflow IMDb DAG** to also import ***name.basics.tsv*** to HDFS raw layer.

    c) **Create Hive table *name_basics*** for ***name.basics.tsv*** in raw layer within DAG. Table should be partitioned by year, month and day of load date like the other tables.

    d) **Create table *actors*** and **extend IMDb Airflow DAG** to fill table using Hive or PySpark:
        - make use of all columns within table name_basics
        - add column ***alive*** which contains alive if actor is alive or dead if actor is dead
        - add column ***age*** which contains current age of actor (calculated by using birth and death year)

    e) Run DAG

Well Done

WE'RE DONE FOR ...TODAY

Stop Your VM Instances

DON'T FORGET TO STOP YOUR VM INSTANCE!


```
gcloud compute instances stop big-data
```