L’evento inizierà alle 18:05

develer 20

Vita da develeriani

Equilibrio tra vita lavorativa e privata

- *Sede aperta 24h su 24*
- *Flessibilità oraria*
- *Banca ore*

L’evento inizierà alle 18:05

develer | 20

Vita da develeriani

AI passo con i tempi

Incoraggiamento alla formazione annuale:

- *Budget 800€*
- *Budget 80 ore*

L’evento inizierà alle 18:05

develer 20

Vita in Develer

L'evento inizierà alle 18:05

develer 20

Lavora in Develer

- Web Full Stack Developer
- C++ Developer
- Python Developer

Claudia Cimino
Responsabile risorse umane

jobs@develer.com

Nadir Sampaoli
Developer

Pipeline con Python e Luigi

Develer webinar

2022-02-23

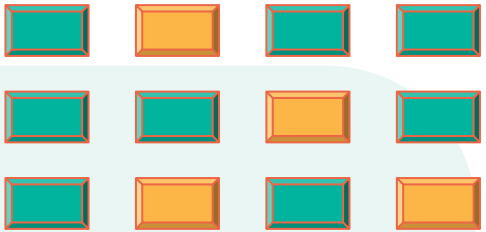
develer

- Cos’è Luigi
- Possibili utilizzi
- Esempio pratico

COS'È LUIGI

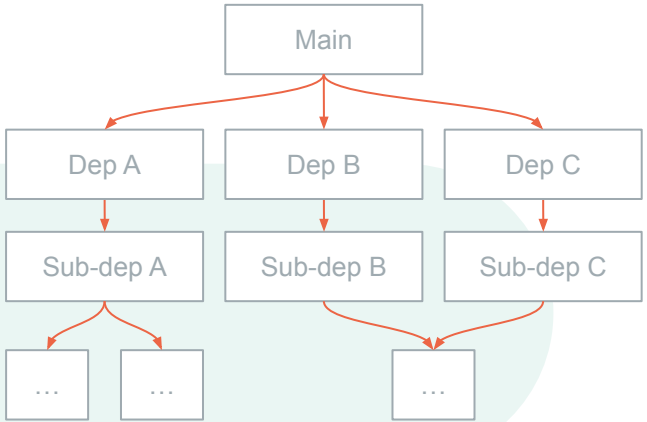
- pacchetto **Python**
- **pipeline** formata da **task**
- coordinati da uno **scheduler**
- per gestire **batch job**

BATCH JOB



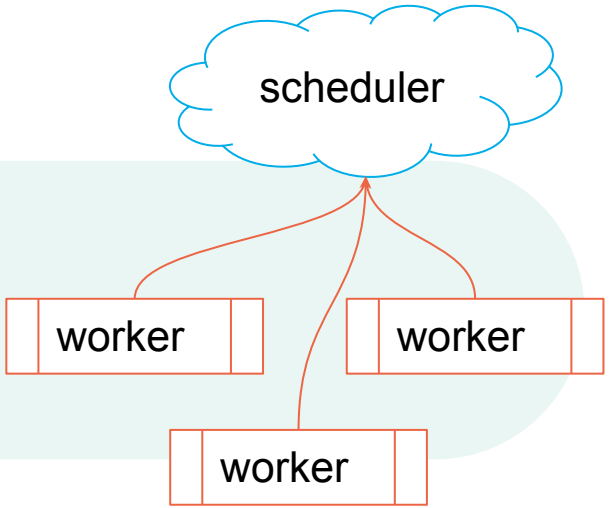
- programma **non interattivo**
- sequenza di operazioni
- esecuzione **automatica**
- **processamento** dati

PIPELINE



- **pipeline** = DAG
  - suddivide lavoro in **task**
  - definisce **dipendenze** tra task
- **task** = unità di lavoro
  - riceve **input** da dipendenze
  - legge/trasforma/usa dati
  - produce **output**
- **dipendenze** = relazioni tra task

SCHEDULER



- **scheduler**
  - server
  - coordina i task
  - mantiene lo stato della pipeline
- **luigi worker**
  - client
  - chiede lo stato della pipeline
  - esegue i task

INSTALLAZIONE

```
$ pip install luigi
```

luigi.Task

```python
class MyTask(luigi.Task):
    def requires(self):
        return OtherTask()

    def output(self):
        return luigi.LocalTarget("/some/file.txt")

    def run(self):
        print("do something")
        with self.output().open("w") as output:
            output.write("done!")
```

luigi.Task

```python
class MyTask(luigi.Task):
    def requires(self):
        return OtherTask()

    def output(self):
        return luigi.LocalTarget("/some/file.txt")

    def run(self):
        print("do something")
        with self.output().open("w") as output:
            output.write("done!")
```

classe derivata da luigi.Task

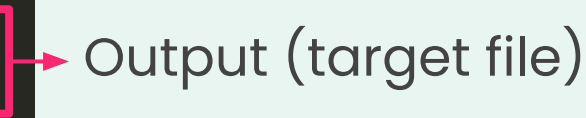
Task.requires()

```python
class MyTask(luigi.Task):
    def requires(self):
        return OtherTask()

    def output(self):
        return luigi.LocalTarget("/some/file.txt")

    def run(self):
        print("do something")
        with self.output().open("w") as output:
            output.write("done!")
```

Dipendenze del task

Task.output()

```python
class MyTask(luigi.Task):
    def requires(self):
        return OtherTask()

    def output(self):
        return luigi.LocalTarget("/some/file.txt")

    def run(self):
        print("do something")
        with self.output().open("w") as output:
            output.write("done!")
```



Task.run()

```python
class MyTask(luigi.Task):
    def requires(self):
        return OtherTask()

    def output(self):
        return luigi.LocalTarget("/some/file.txt")

    def run(self):
        print("do something")
        with self.output().open("w") as output:
            output.write("done!")
```

Cosa fa effettivamente il task

Task.input() ← Task.requires()

```python
class MyTask(luigi.Task):
    def requires(self) -> List[luigi.Task]:
        return [
            TaskA(),
            TaskB(),
        ]

    def run(self):
        with self.output().open("w") as output:
            for input in self.input():
                with input.open("r") as input_data:
                    output.write(input_data)
```

luigi.Parameter

```python
class DailyTask(luigi.Task):
    date = luigi.DateParameter()

    def requires(self):
        return OtherTask(date=self.date)

    def output(self):
        date = self.date.isoformat()
        return luigi.LocalTarget(f"/{date}/file.txt")

    def run(self):
        # ...
```

USE CASE:
Sincronizzare dati tra sistemi

- Gestionale -> Shop online
- Form contatti -> CRM
- Backup dati off-site

USE CASE:
Aggregare e trasformare stream di dati

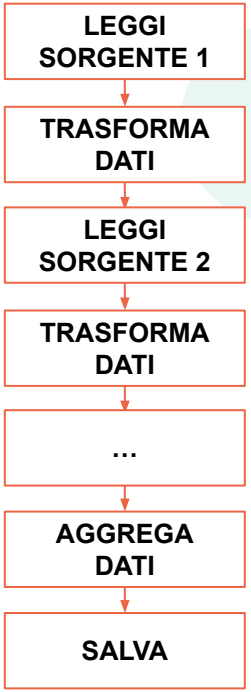
- Log eventi sistema real-time
- aggregazione metriche e statistiche

USE CASE:

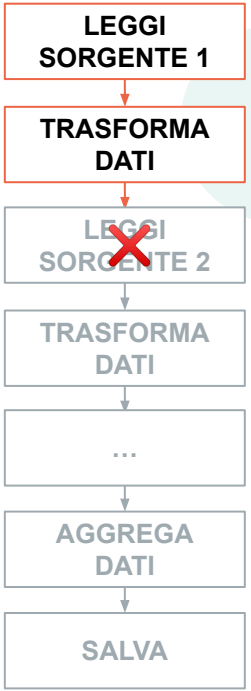
Aggregare e trasformare stream di dati

- Raccogliere dati da più servizi remoti
- trasformare/filtrare/unire i dati
- salvare i risultati

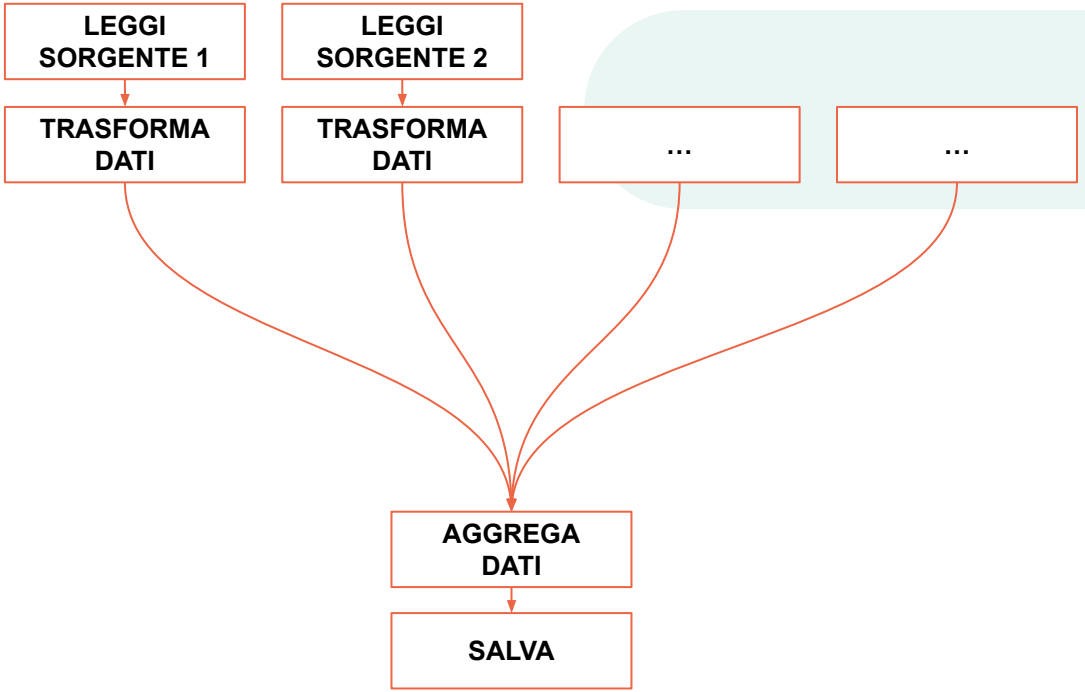
Procedura lineare



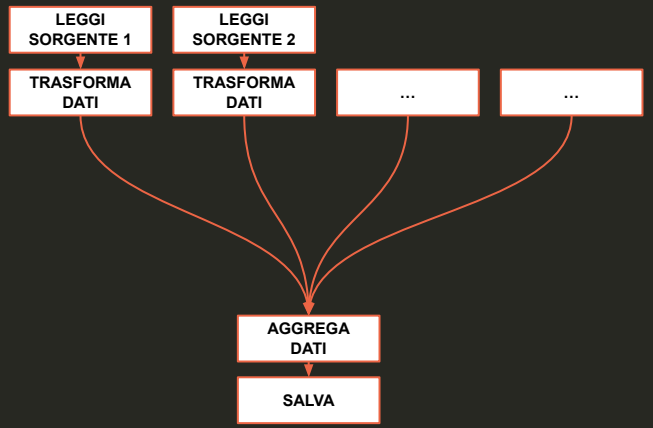
Procedura lineare



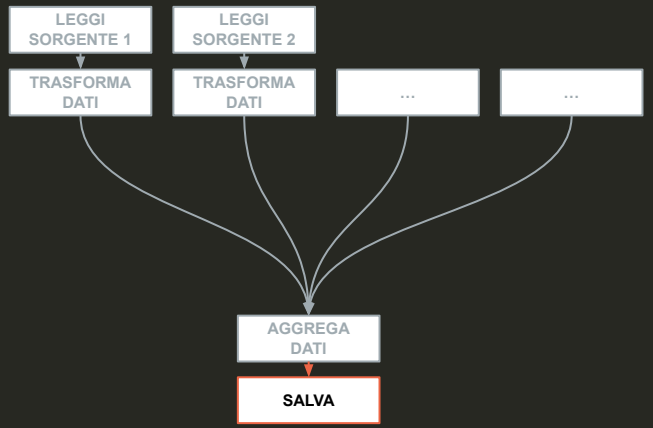
Pipeline


LEGGI SORGENTE 1
LEGGI SORGENTE 2
TRASFORMA DATI
TRASFORMA DATI
...
...
AGGREGA DATI
SALVA

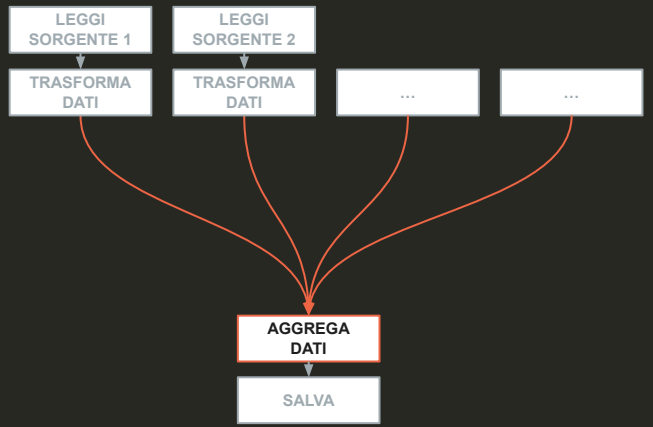
```python
class Pipeline(luigi.Task):
    date_hour = luigi.DateHourParameter()

    def requires(self):
        return SaveTask(date_hour=self.date_hour)

    def output(self):
        date = self.date_hour.isoformat()
        return luigi.LocalTarget(f"/pipeline/{date}/success.txt")

    def run(self):
        write_success(self.output())

def write_success(target: luigi.Target):
    with target.open("w") as f:
        f.write("DONE!")
```



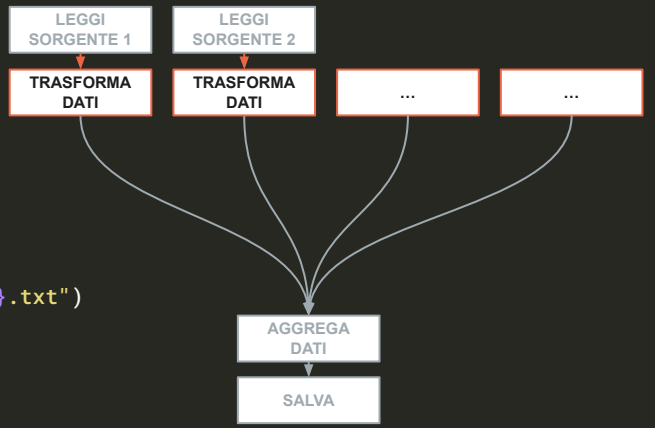
```python
class SaveTask(luigi.Task):
    date_hour = luigi.DateHourParameter()

    def requires(self):
        return MergeTask(date_hour=self.date_hour)

    def output(self):
        date = self.date_hour.isoformat()
        return luigi.LocalTarget(f"/pipeline/{date}/save.txt")

    def run(self):
        with self.input().open("r") as in_file:
            save_data_in_some_database(in_file)

        write_success(self.output())
```



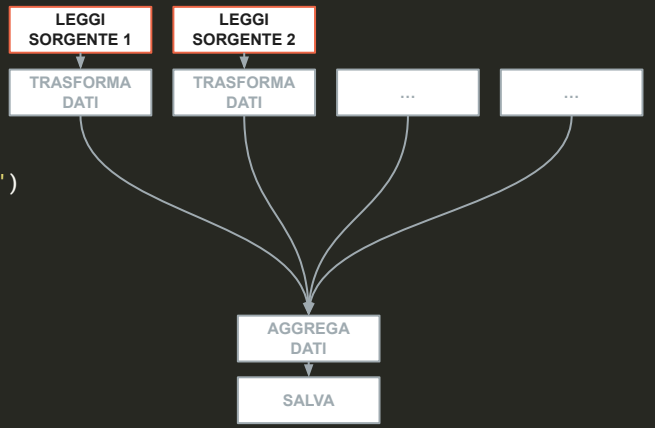
```python
class MergeTask(luigi.Task):
    date_hour = luigi.DateHourParameter()

    def requires(self):
        for source in ["Cassini", "Galileo", "Hubble", "Messenger"]:
            yield TransformTask(date_hour=self.date_hour, source=source)

    def output(self):
        date = self.date_hour.isoformat()
        return luigi.LocalTarget(f"/pipeline/{date}/merge.csv")

    def run(self):
        data = merge_data(self.input())
        with self.output().open("w") as out_file:
            out_file.write(data)
```



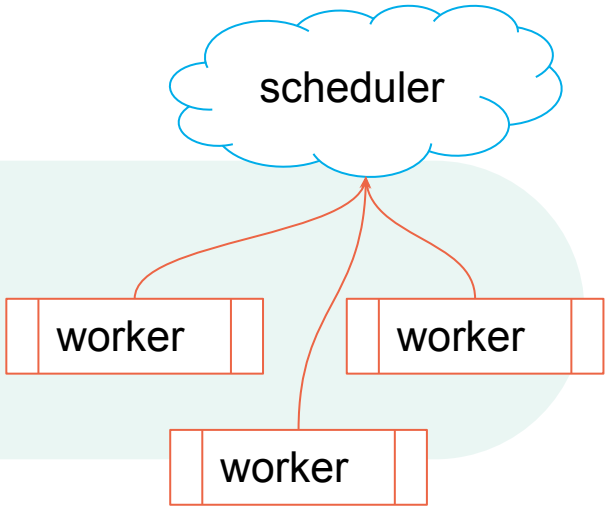
```python
class TransformTask(luigi.Task):
    date_hour = luigi.DateHourParameter()
    source = luigi.Parameter()

    def requires(self):
        return ReadTask(self.date_hour, source=self.source)

    def output(self):
        date = self.date_hour.isoformat()
        return luigi.LocalTarget(f"/pipeline/{date}/transform_{self.source}.txt")

    def run(self):
        data = transform_data(self.input(), self.source)
        with self.output().open("w") as out_file:
            out_file.write(data)
```



```python
class ReadTask(luigi.Task):
    date_hour = luigi.DateHourParameter()
    source = luigi.Parameter()

    def output(self):
        date = self.date_hour.isoformat()
        return luigi.LocalTarget(f"/pipeline/{date}/read_{self.source}.txt")

    def run(self):
        data = read_data_from_some_source(self.source)
        with self.output().open("w") as out_file:
            out_file.write(data)
```

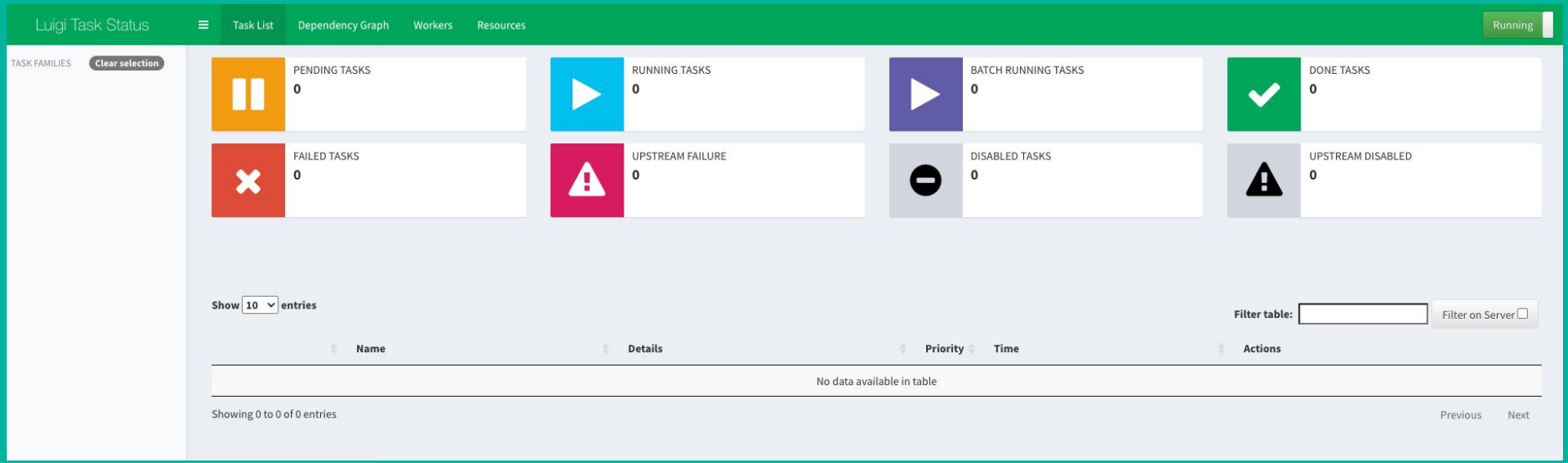


SCHEDULER



- **scheduler**
  - server
  - coordina i task
  - mantiene lo stato della pipeline
- **luigi worker**
  - client
  - chiede lo stato della pipeline
  - esegue i task

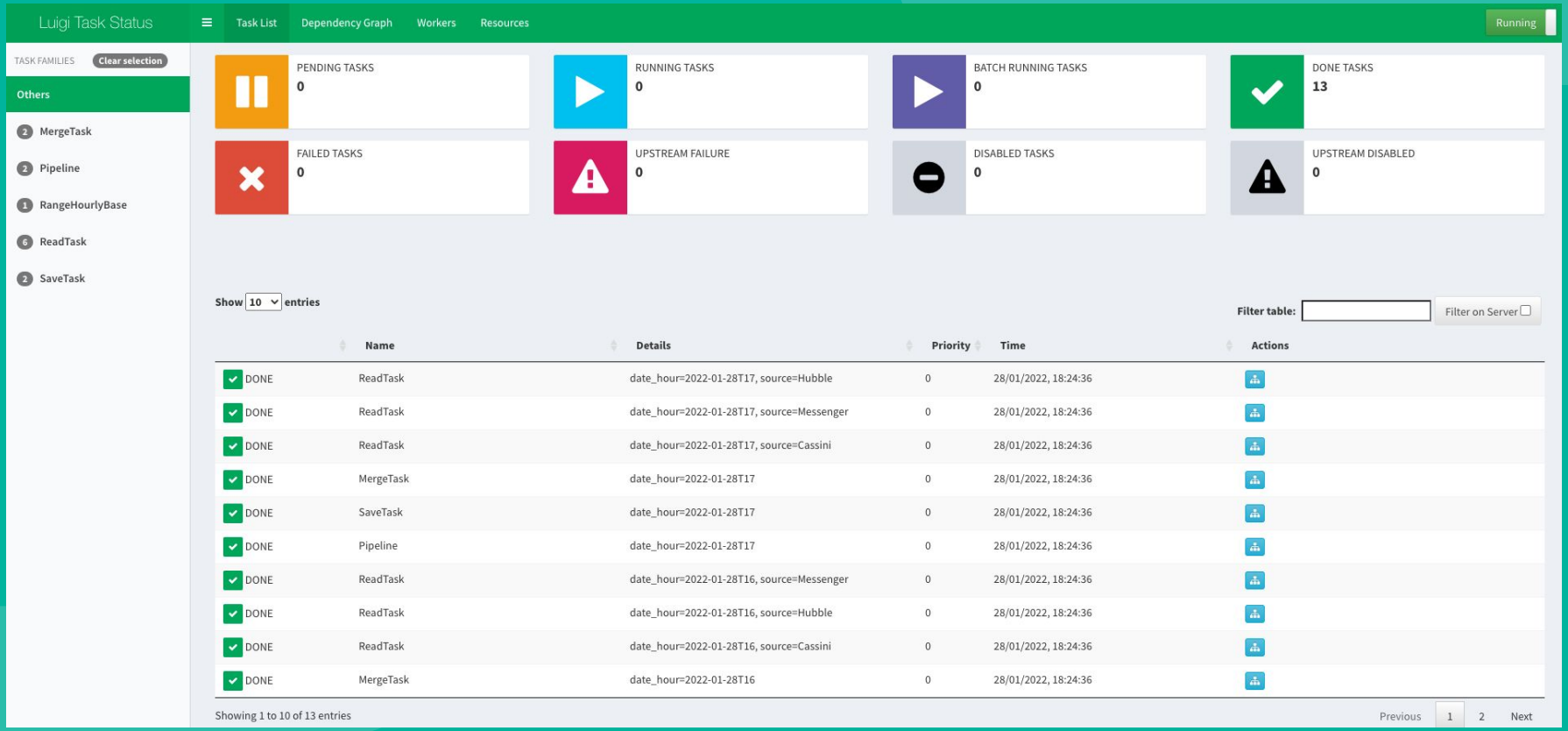
Scheduler




```
$ luigid --port 8082
```

Worker

```
python -m luigi \
       --module backend.pipeline \
       RangeHourlyBase \
       --RangeHourlyBase-of Pipeline \
       --RangeHourlyBase-start 2022-02-23T18 \
       --scheduler-url "http://localhost:8082"
```


Luigi Task Status
Task List
Dependency Graph
Workers
Resources
Running
TASK FAMILIES
Clear selection
Others
2 MergeTask
2 Pipeline
1 RangeHourlyBase
6 ReadTask
2 SaveTask
PENDING TASKS
0
RUNNING TASKS
0
BATCH RUNNING TASKS
0
DONE TASKS
13
FAILED TASKS
0
UPSTREAM FAILURE
0
DISABLED TASKS
0
UPSTREAM DISABLED
0
Show 10 entries
Filter table:
Filter on Server
Name
Details
Priority
Time
Actions
DONE ReadTask date_hour=2022-01-28T17, source=Hubble 0 28/01/2022, 18:24:36
DONE ReadTask date_hour=2022-01-28T17, source=Messenger 0 28/01/2022, 18:24:36
DONE ReadTask date_hour=2022-01-28T17, source=Cassini 0 28/01/2022, 18:24:36
DONE MergeTask date_hour=2022-01-28T17 0 28/01/2022, 18:24:36
DONE SaveTask date_hour=2022-01-28T17 0 28/01/2022, 18:24:36
DONE Pipeline date_hour=2022-01-28T17 0 28/01/2022, 18:24:36
DONE ReadTask date_hour=2022-01-28T16, source=Messenger 0 28/01/2022, 18:24:36
DONE ReadTask date_hour=2022-01-28T16, source=Hubble 0 28/01/2022, 18:24:36
DONE ReadTask date_hour=2022-01-28T16, source=Cassini 0 28/01/2022, 18:24:36
DONE MergeTask date_hour=2022-01-28T16 0 28/01/2022, 18:24:36
Showing 1 to 10 of 13 entries
Previous
1
2
Next

- ALTERNATIVE: AIRFLOW

Airflow

- scritto in Python
- DAG-based
- ricca UI
- non richiede cron esterno

- https://luigi.readthedocs.io/

workshop@develer.com

Vuoi rimanere aggiornato sugli eventi Develer?
Seguici nei nostri canali social:

develer

www.develer.com