TA Section Week 5

10/4/2018 8-9PM

Zhenyu Zhao

DevOps Engineer

Infrastructure Technology Services

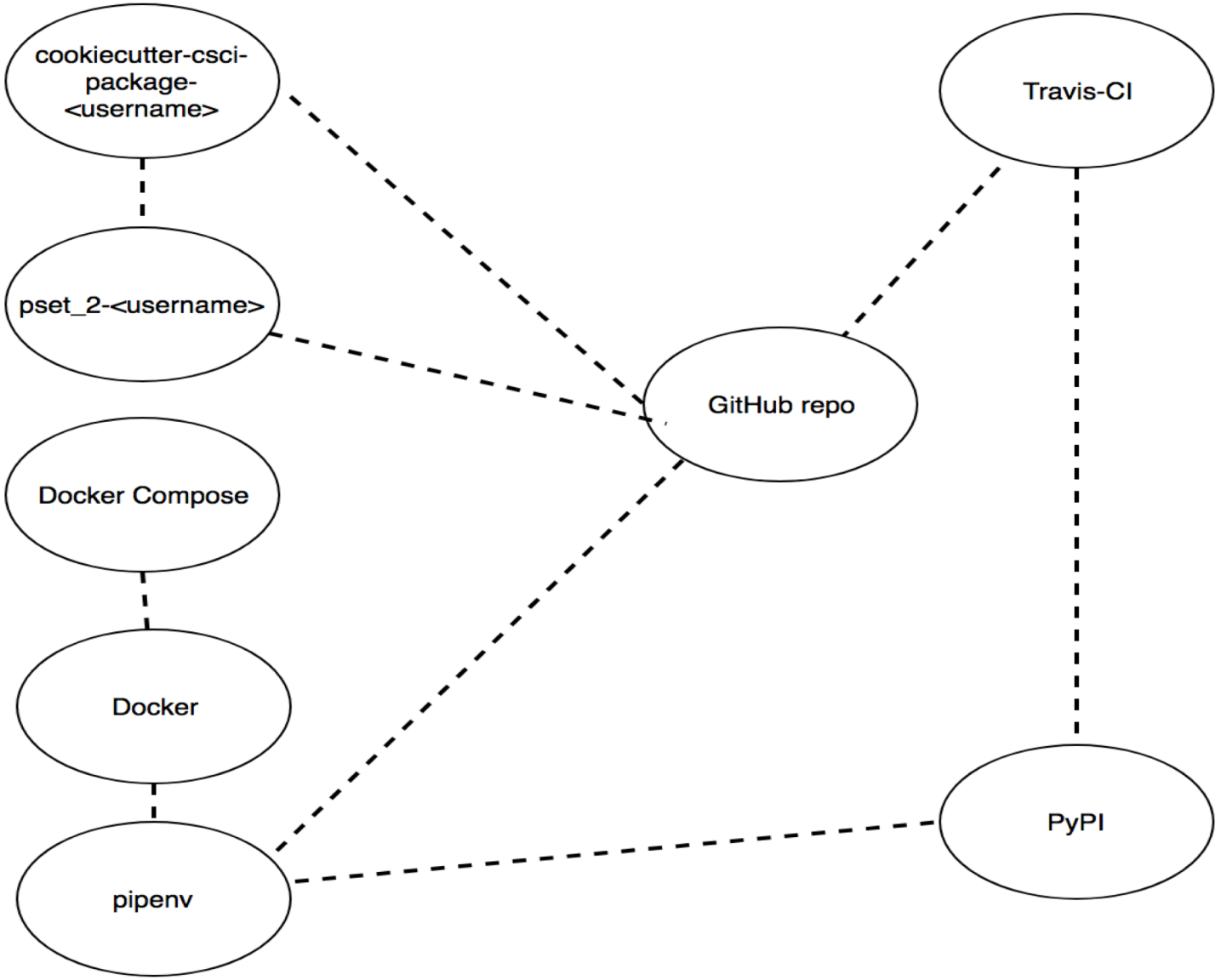
Harvard University IT

Email: Zhenyu_zhao@Harvard.edu

Agenda

- ❑Pset2 Afterthought
- ❑Python Package from init to more
- ❑Luigi modue

Pset2 Afterthought



Pset2 Afterthought

- There are 3 Python environments here:
  - ✓ Local file system
  - ✓ Docker image/container
  - ✓ Travis CI

Pset2 Afterthought

- Local file system
  - $ cd /path/to/project
  - $ pipenv install
  - $ pipenv check
  - $ pipenv run python

Pset2 Afterthought

- Docker image/container

  $ docker-compose run app pipenv run python

  $ docker-compose run app pipenv run pytest

Pset2 Afterthought

- Travis CI

```
[zzhao@rhel72 pset_2-zhenyuzhao (develop)]$ cat .travis.yml
sudo: required
language: minimal
services:
- docker
install: docker-compose build
jobs:
  include:
  - stage: test
    if: branch = develop
    script: ./drun_app pytest
  - stage: deploy
    if: branch = master
    script: ./drun_app python main.py
env:
  global:
    secure: C04ZnzpOpGoRDAgpnAfMgVxBmrroXx9t70xyvHFEmi9D524VZqSPZicZDOea8NXatZPZuTYFL/
```

```
1  Uploading script
2  Worker information
7  Build system information
414
415 Network availability confirmed.
416
417 Setting APT mirror in /etc/apt/sources.list: http://us-central1.gce.archive.ubuntu.com/ubuntu/
418 $ sudo service docker start
421 Installing SSH key from: default repository key
423 Using /home/travis/.netrc to clone repository.
425
426 $ git clone --depth=50 --branch=master https://github.com/csci-e-29/pset_utils-manish.git csci-
437 $ source ~/virtualenv/python3.6/bin/activate
438
439 $ python --version
440 Python 3.6.3
441 $ pip --version
442 pip 9.0.1 from /home/travis/virtualenv/python3.6.3/lib/python3.6/site-packages (python 3.6)
443 $ docker-compose build
490 $ ./drun_app pytest
491 Creating network "psetutilsmanish_default" with the default driver
492 ============================= test session starts ==============================
493 platform linux -- Python 3.6.6, pytest-3.8.2, py-1.6.0, pluggy-0.7.1
494 rootdir: /app, inifile: setup.cfg
495 plugins: cov-2.6.0
496 collected 6 items
497
498 pset_utils_manish/hashing/tests/test__HashStringTestCase.py ..           [ 33%]
499 pset_utils_manish/io/tests/test__AtomicWriteTestCase.py ...              [ 83%]
500 tests/test_pset_utils_manish.py .                                        [100%]
501
502 ---------- coverage: platform linux, python 3.6.6-final-0 ----------
```

Python Packaging from init to Deploy

- Watch a video presentation:

https://www.youtube.com/watch?v=4fzAMdLKC5k

BTW Pay attention to three different package distribution methods:

1. sdist: source distribution - bleeding edge version

   $ pipenv install -e git+https://github.com/csci-e-29/pset_utils-zhenyuzhao#egg=pset_utils)
2. sdist: source distribution - setup.py
3. bdist_wheel: build distribution (wheel format)
4. ~~bdist_egg: build distribution (egg format, obselete)~~

Luigi #1

In the early days of a prototype, the data pipeline often looks like this:

```
$ python get_some_data.py
$ python clean_some_data.py
$ python join_other_data.py
$ python do_stuff_with_data.py
```

Luigi #2

The obvious hacky solution seems to be: let's put everything in one script called do_everything.py

```
if __name__ == '__main__':
    get_some_data()
    clean_some_data()
    join_other_data()
    do_stuff_with_data()
```

Then, run:

```
$ python do_everything.py
```

Luigi #3

Problem:

1. When moving towards a production-ready pipeline, there are a few more aspects to consider besides the run-everything code. In particular, error handling should be taken into account:

```
try:
    get_some_data()
except GetSomeDataError as e:
    # handle this
```

2. But if we chain all the individual tasks together, we end up with a Christmas tree of try/except:

```
try:
    get_some_data()
    try:
        clean_some_data()
        try:
            # you see where this is going...
        except EvenMoreErrors:
            # ...
    except CleanSomeDataError as e:
        # handle CleanSomeDataError
except GetSomeDataError as e:
    # handle GetSomeDataError
```

Luigi #4

Problem:

3. How to resume a pipeline?

```
# check if the task was already successful
if not i_got_the_data_already():
    # if not, run it
    try:
        get_some_date()
    except GetSomeDataError as e:
        # handle the error
```

Luigi #5

- What is Luigi?

  https://github.com/spotify/luigi

Luigi is a Python module that helps you build complex pipelines of batch jobs. It helps you manage workflow. It handles dependency resolution, workflow management, visualization etc.

Some useful features:

- Dependency management
- Checkpoints/failure recovery
- CLI integration/parametrization
- Dependency graph visualization

Luigi #6

- Two core concepts
  - ✓ Task: a task is a unit of work, designed by extending the class luigi.
  - ✓ Target: the output of a task is a target

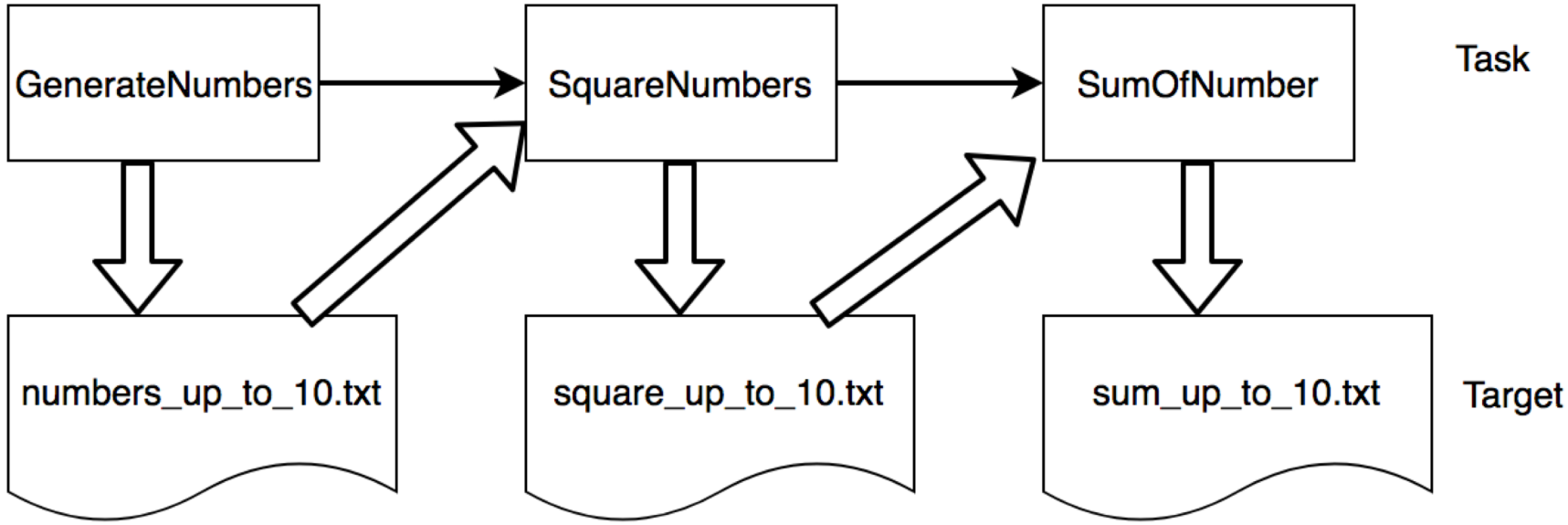
Luigi #7

- Dependency

Dependencies are defined in terms of inputs and outputs, i.e. if Task B depends on Task A, it means that the output pf Task A is the input of Task B.

Luigi #8

Example:



Luigi #9

Task#1: GenerateNumbers

Target: numbers_up_to_xxx.txt

```
class GenerateNumbers(luigi.Task):
    n = luigi.IntParameter(default=10)

    def requires(self):
        return []

    def output(self):
        return luigi.LocalTarget("numbers_up_to_{}.txt".format(self.n))

    def run(self):
        with self.output().open('w') as f:
            for i in range(1, self.n+1):
                #time.sleep(5)
                f.write("{}\n".format(i))
```

Luigi #10

Task#2: GenerateNumbers

Target: squares_up_to_xxx.txt

```
class SquaredNumbers(luigi.Task):
    n = luigi.IntParameter(default=10)

    def requires(self):
        return [GenerateNumbers(n=self.n)]

    def output(self):
        return luigi.LocalTarget("squares_up_to_{}.txt".format(self.n))

    def run(self):
        with self.input()[0].open() as fin, self.output().open('w') as fout:
            for line in fin:
                #time.sleep(5)
                n = int(line.strip())
                out = n * n
                fout.write("{}:{}\n".format(n, out))
```

Luigi #11

Task#3: SumOfNumbers

Target: sum_up_to_xxx.txt

```
class SumOfNumbers(luigi.Task):
    n = luigi.IntParameter(default=10)

    def requires(self):
        return [SquaredNumbers(n=self.n)]

    def output(self):
        return luigi.LocalTarget("sum_up_to_{}.txt".format(self.n))

    def run(self):
        with self.input()[0].open() as fin, self.output().open('w') as fout:
            out = 0
            for line in fin:
                #time.sleep(5)
                out += int(line.strip())

            fout.write("SUM:{}\n".format(out))
```

Luigi #12

You can launch the pipeline using the local scheduler for testing

```
$ python run_luigi_2.py SumOfNumbers --local-scheduler
```

Luigi #13

You can also launch the pipeline using the central schedule.

Firstly, start up the central scheduler:

```
# luigid --background
```

By default, the daemon is listening on 8082/tcp

```
$ python run_luigi_3.py SumOfNumbers
```

Luigi #14

Then, launch the pipeline

```
$ python run_luigi_3.py SumOfNumbers
$ python run_luigi_3.py SumOfNumbers  --n 20
```

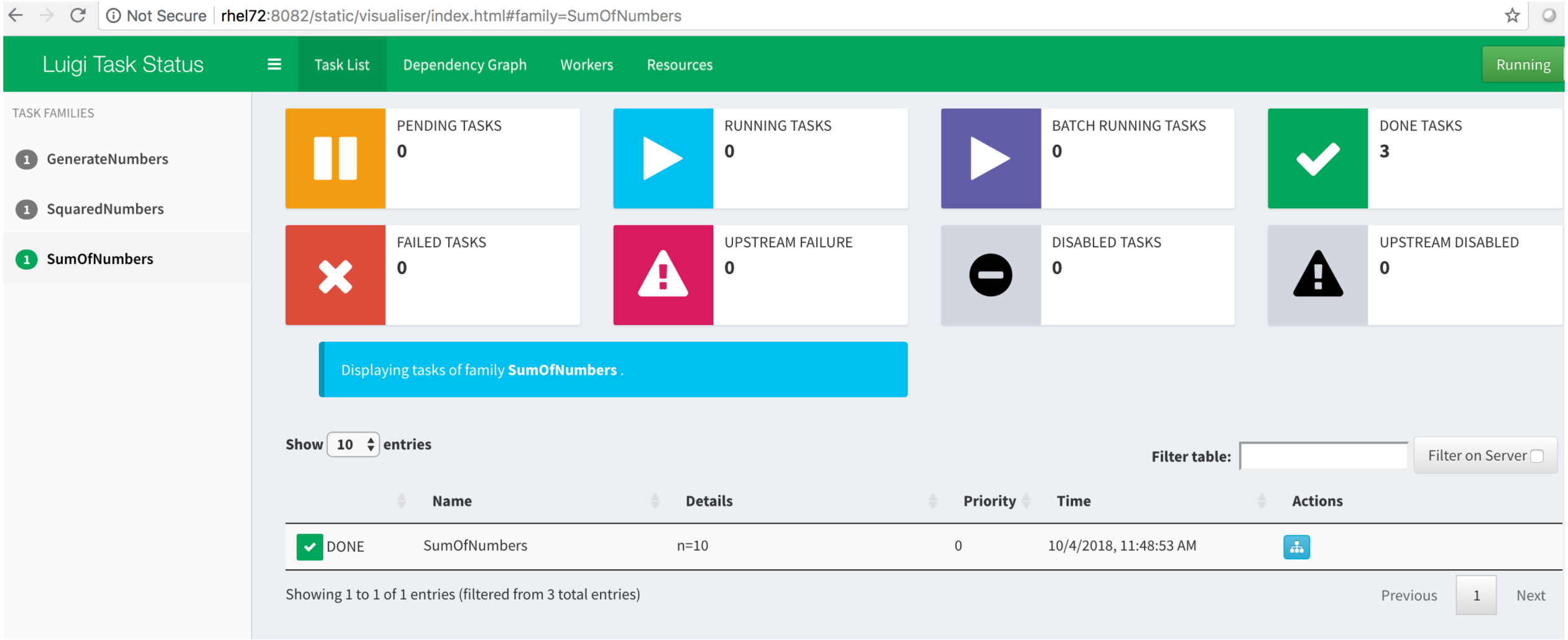
Luigi #15

Output:

```
DEBUG: Checking if SumOfNumbers(n=10) is complete
DEBUG: Checking if SquaredNumbers(n=10) is complete
INFO: Informed scheduler that task   SumOfNumbers_10_afdc16722f   has status   PENDING
DEBUG: Checking if GenerateNumbers(n=10) is complete
INFO: Informed scheduler that task   SquaredNumbers_10_afdc16722f   has status   PENDING
INFO: Informed scheduler that task   GenerateNumbers_10_afdc16722f   has status   PENDING
INFO: Done scheduling tasks
INFO: Running Worker with 1 processes
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 3
INFO: [pid 67865] Worker Worker(salt=587366431, workers=1, host=rhel72.linuxtoys.net, username=zzhao, pid=67865) running   GenerateNumbers(n=10)
INFO: [pid 67865] Worker Worker(salt=587366431, workers=1, host=rhel72.linuxtoys.net, username=zzhao, pid=67865) done      GenerateNumbers(n=10)
DEBUG: 1 running tasks, waiting for next task to finish
INFO: Informed scheduler that task   GenerateNumbers_10_afdc16722f   has status   DONE
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 2
INFO: [pid 67865] Worker Worker(salt=587366431, workers=1, host=rhel72.linuxtoys.net, username=zzhao, pid=67865) running   SquaredNumbers(n=10)
INFO: [pid 67865] Worker Worker(salt=587366431, workers=1, host=rhel72.linuxtoys.net, username=zzhao, pid=67865) done      SquaredNumbers(n=10)
DEBUG: 1 running tasks, waiting for next task to finish
INFO: Informed scheduler that task   SquaredNumbers_10_afdc16722f   has status   DONE
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 1
INFO: [pid 67865] Worker Worker(salt=587366431, workers=1, host=rhel72.linuxtoys.net, username=zzhao, pid=67865) running   SumOfNumbers(n=10)
INFO: [pid 67865] Worker Worker(salt=587366431, workers=1, host=rhel72.linuxtoys.net, username=zzhao, pid=67865) done      SumOfNumbers(n=10)
DEBUG: 1 running tasks, waiting for next task to finish
INFO: Informed scheduler that task   SumOfNumbers_10_afdc16722f   has status   DONE
DEBUG: Asking scheduler for work...
DEBUG: Done
DEBUG: There are no more tasks to run at this time
INFO: Worker Worker(salt=587366431, workers=1, host=rhel72.linuxtoys.net, username=zzhao, pid=67865) was stopped. Shutting down Keep-Alive thread
INFO:
===== Luigi Execution Summary =====

Scheduled 3 tasks of which:
* 3 ran successfully:
    - 1 GenerateNumbers(n=10)
    - 1 SquaredNumbers(n=10)
    - 1 SumOfNumbers(n=10)

This progress looks :) because there were no failed tasks or missing dependencies

===== Luigi Execution Summary =====
```

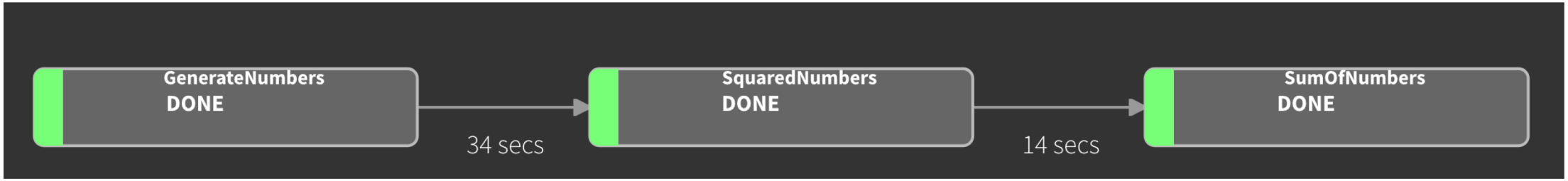
Luigi #16

Go to the central scheduler web site:



Luigi #17

See task dependency graph:



Luigi #18

The centralized scheduler serves two purposes:

- Make sure two instances of the same task are not running simultaneously
- Provide visualization of everything that's going on.

The central scheduler does not execute anything for you or help you with job parallelization. For running tasks periodically, the easiest thing to do is to trigger a Python script from cron or from a continuously running process. There is no central process that automatically triggers jobs.