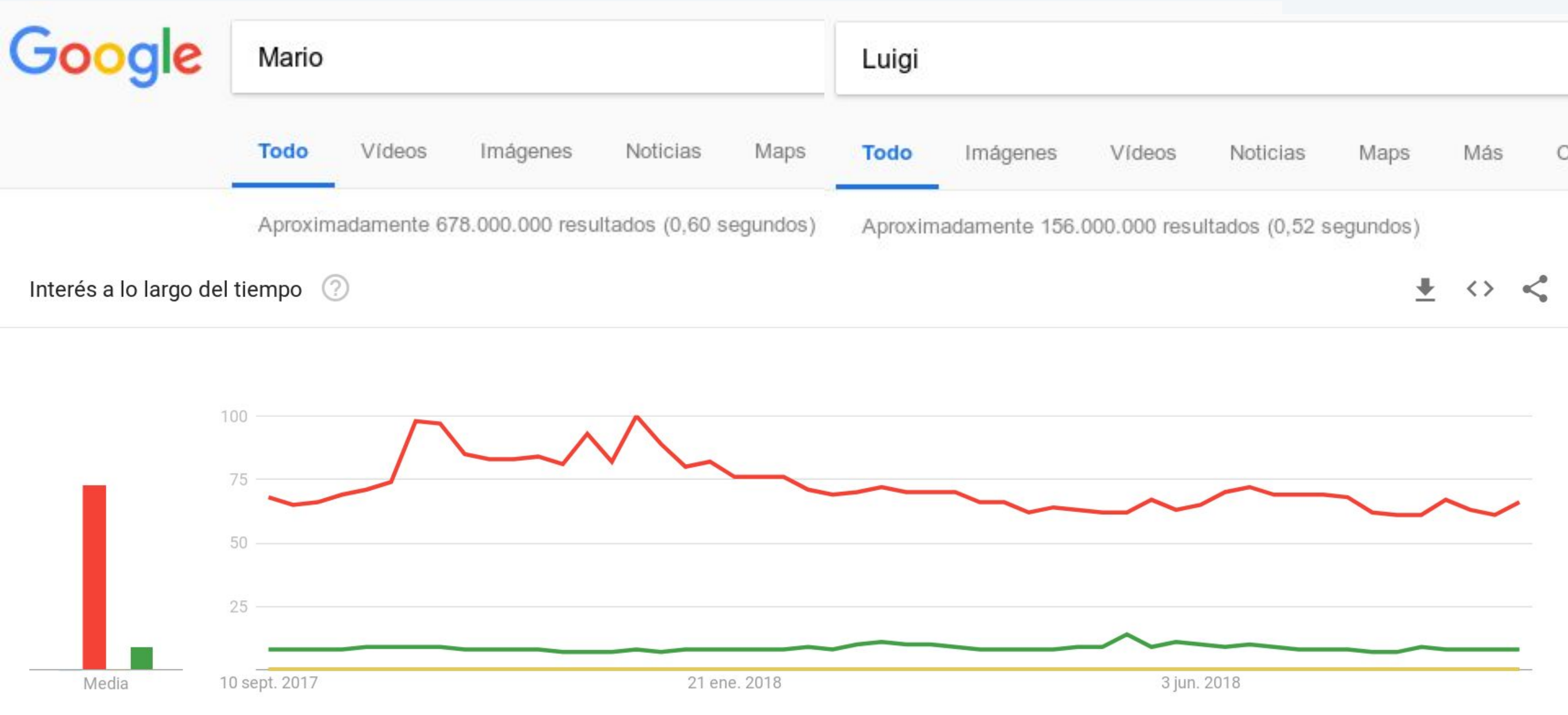
CARTO

2018

Sustituyendo Makefiles con Python y Luigi

Javier Torres

Backend Lead @ CARTO



Mario

Todo Vídeos Imágenes Noticias Maps

Aproximadamente 678.000.000 resultados (0,60 segundos)

Luigi

Todo Imágenes Vídeos Noticias Maps Más

Aproximadamente 156.000.000 resultados (0,52 segundos)

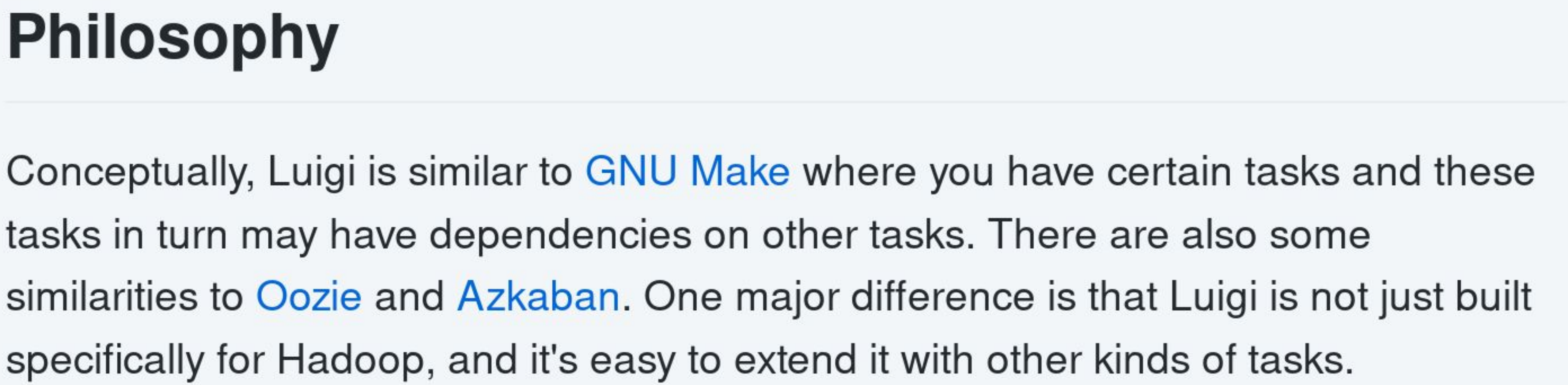
```python
import imageio, numpy, matplotlib.colors

for frame in imageio.mimread('mario.gif'):
    rgb = frame[:,:,0:3] / 256
    hsv = matplotlib.colors.rgb_to_hsv(rgb)
    hsv[..., 0] = numpy.where(
        numpy.absolute(hsv[...,0] - 0.5) < 0.48,
        hsv[...,0],
        (hsv[...,0] + 0.3) % 1
    )
    rgb = matplotlib.colors.hsv_to_rgb(hsv) * 256
    numpy.copyto(frame[...,0:3], rgb, casting='unsafe')

durations = [f.meta.duration/1000 for f in frames]
imageio.mimwrite('luigi.gif', frames, duration=durations)
```

Luigi

De Makefile a Luigi con un ejemplo



Philosophy

Conceptually, Luigi is similar to GNU Make where you have certain tasks and these tasks in turn may have dependencies on other tasks. There are also some similarities to Oozie and Azkaban. One major difference is that Luigi is not just built specifically for Hadoop, and it's easy to extend it with other kinds of tasks.

```
hello: hello.c
    gcc -o $@ $^
```

```
hello: hello.c

     gcc -o $@ $^
```

```
class InputTask(ExternalTask):
    filename = Parameter()

    def output(self):
        return LocalTarget(self.filename)

class Compile(Task):
    def requires(self):
        return InputTask('hello.c')

    def run(self):
        subprocess.run(['gcc', '-o',
                        self.output().path,
                        self.input().path])

    def output(self):
        return LocalTarget('hello')
```

```
hello: hello.c
	gcc -o $@ $^
```

```python
class InputTask(ExternalTask):
    filename = Parameter()

    def output(self):
        return LocalTarget(self.filename)

class Compile(Task):
    def requires(self):
        return InputTask('hello.c')

    def run(self):
        subprocess.run(['gcc', '-o',
                        self.output().path,
                        self.input().path])

    def output(self):
        return LocalTarget('hello')
```

```
hello: hello.c
    gcc -o $@ $^
```

```python
class InputTask(ExternalTask):
    filename = Parameter()

    def output(self):
        return LocalTarget(self.filename)
```

```python
class Compile(Task):
    def requires(self):
        return InputTask('hello.c')

    def run(self):
        subprocess.run(['gcc', '-o',
                        self.output().path,
                        self.input().path])

    def output(self):
        return LocalTarget('hello')
```

```
hello: hello.c
    gcc -o $@ $^
```

```python
class InputTask(ExternalTask):
    filename = Parameter()

    def output(self):
        return LocalTarget(self.filename)

class Compile(Task):
    def requires(self):
        return InputTask('hello.c')

    def run(self):
        subprocess.run(['gcc', '-o',
                        self.output().path,
                        self.input().path])

    def output(self):
        return LocalTarget('hello')
```

```
hello: hello.c
    gcc -o $@ $^
```

```python
class InputTask(ExternalTask):
    filename = Parameter()

    def output(self):
        return LocalTarget(self.filename)

class Compile(Task):
    def requires(self):
        return InputTask('hello.c')

    def run(self):
        subprocess.run(['gcc', '-o',
                        self.output().path,
                        self.input().path])

    def output(self):
        return LocalTarget('hello')
```

```
%.o: %.c
    gcc -c -o $@ $<

hello: hello.o hello2.o
    gcc -o $@ $^
```

```
%.o: %.c
    gcc -c -o $@ $<

hello: hello.o hello2.o
    gcc -o $@ $^
```

```
class CompileObject(Task):
    obj = Parameter()

    def requires(self):
        return InputTask(self.obj+ '.c')

    def run(self):
        subprocess.run(['gcc', '-c', '-o',
                        self.output().path,
                        self.input().path])

    def output(self):
        return LocalTarget(self.obj+ '.o')
```

```
%.o: %.c
	gcc -c -o $@ $<

hello: hello.o hello2.o
	gcc -o $@ $^
```

```python
class CompileObject(Task):
    obj = Parameter()

    def requires(self):
        return InputTask(self.obj+ '.c')

    def run(self):
        subprocess.run(['gcc', '-c', '-o',
                        self.output().path,
                        self.input().path])

    def output(self):
        return LocalTarget(self.obj+ '.o')
```

```
%.o: %.c
    gcc -c -o $@ $<

hello: hello.o hello2.o
    gcc -o $@ $^
```

```
class Compile(Task):
    def requires(self):
        return CompileObject('hello'),
               CompileObject('hello2')

    def run(self):
        subprocess.run(
            ['gcc', '-o', self.output().path] +
            [i.path for i in self.input()])

    def output(self):
        return LocalTarget('hello')
```

```
%.o: %.c
    gcc -c -o $@ $<

hello: hello.o hello2.o
    gcc -o $@ $^
```

```
class Compile(Task):
    def requires(self):
        return CompileObject('hello'),
               CompileObject('hello2')

    def run(self):
        subprocess.run(
            ['gcc', '-o', self.output().path] +
            [i.path for i in self.input()])

    def output(self):
        return LocalTarget('hello')
```

¿Cuándo usar Luigi?

Extensibilidad

Targets

El objetivo puede ser cualquier cosa, no sólo ficheros.

Tasks

Las tareas son código Python. Mucho más que con comandos de shell.

```
class IssueTarget(luigi.Target):
  def __init__(self, repo, number):
    self.repo = repo
    self.number = number

  def exists(self):
    issue = self.repo.issue_set.where(
        models.Issue.number == self.number
    ).first()
    return issue is not None
```

Extensibilidad

Targets

El objetivo puede ser cualquier cosa, no sólo ficheros.

Tasks

Las tareas son código Python. Mucho más que con comandos de shell.

```
class GetPullRequest(luigi.Task):
  repo = luigi.Parameter()
  number = luigi.IntParameter()

  def run(self):
    user = Github().get_user(self.repo.user)
    repo = user.get_repo(self.repo.name)
    issue = repo.get_issue(self.number)
    models.Issue.create(
      repository=self.repo,
      number=self.number,
      name=issue.title
    )
```

Batteries included

Planificador multi-nodo

Tareas largas y pesadas.

Visualizador web

Estado de ejecución, dependencias

Hadoop / HDFS

Trabajar con grandes volúmenes de datos.

Bases de datos / URIs

Soporte para objetivos en MySQL, MSSQL, Postgres, Redis, FTP, SSHFS...

ETL!

La letra pequeña

Parámetros no serializables

(multiprocessing)

```python
class GetPullRequest(luigi.Task):
    repo = luigi.Parameter()
    number = luigi.IntParameter()

    def run(self):
        Issue.create(repository=self.repo, number=self.number, name="")

    def output(self):
        return IssueTarget(self.repo, self.number)


class DoStuff(luigi.WrapperTask):
    def requires(self):
      repo = Repository.get(1)
      return [GetPullRequest(repo, i) for i in range(10)]
```

```
$ PYTHONPATH=. luigi --module tasks.test DoStuff --local-scheduler
$ PYTHONPATH=. luigi --module tasks.test DoStuff --local-scheduler

INFO: Informed scheduler that task   DoStuff__99914b932b   has status   PENDING
…
INFO: Running Worker with 1 processes
…
INFO: [pid 12476] Worker Worker(salt=873079442, workers=1, host=archie,
username=javier, pid=12476) done  DoStuff()
===== Luigi Execution Summary =====

Scheduled 11 tasks of which:
* 11 ran successfully:
    - 1 DoStuff()
    - 10 GetPullRequest(repo=1, number=0...9)

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

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

$
```

```
$ PYTHONPATH=. luigi --module tasks.test DoStuff --local-scheduler --workers 4
$ PYTHONPATH=. luigi --module tasks.test DoStuff --local-scheduler --workers 4

INFO: Informed scheduler that task   DoStuff__99914b932b   has status   PENDING
…
INFO: Running Worker with 4 processes
INFO: [pid 12634] Worker Worker(salt=728263954, workers=4, host=archie,
username=javier, pid=12621) running   GetPullRequest(repo=1, number=1)
ERROR: [pid 12633] Worker Worker(salt=728263954, workers=4, host=archie,
username=javier, pid=12621) failed GetPullRequest(repo=1, number=9)
Traceback (most recent call last):
  File "peewee.py", line 2653, in execute_sql
    cursor.execute(sql, params or ())
psycopg2.DatabaseError: error with status PGRES_TUPLES_OK and no message from
the libpq
```

```
class GetPullRequest(luigi.Task):
    repo_id = luigi.IntParameter()
    number = luigi.IntParameter()

    def run(self):
        Issue.create(repository_id=self.repo_id, number=self.number, name="")

    def output(self):
        return IssueTarget(self.repo_id self.number)


class DoStuff(luigi.WrapperTask):
    def requires(self):
      return [GetPullRequest(1, i) for i in range(10)]
```

```
class GetPullRequest(luigi.Task):
    repo = luigi.Parameter()
    number = luigi.IntParameter()

    @models.db.connection_context()
    def run(self):
        Issue.create(repository=self.repo, number=self.number, name="")

    def output(self):
        return IssueTarget(self.repo, self.number)


class DoStuff(luigi.WrapperTask):
    def requires(self):
      repo = Repository.get(1)
      return [GetPullRequest(repo, i) for i in range(10)]
```

Tareas que no siempre devuelven lo mismo

p.ej: actualizaciones

```
class PullRepository(luigi.Task):
    repo = luigi.Parameter()

    def run(self):
        git.Repo(self.repo.path()).remote().pull(progress=CloneProgress(self))

    def output(self):
        return ???
```

```
class PullRepository(luigi.Task):
    repo = luigi.Parameter()
    executed = False

    def run(self):
        git.Repo(self.repo.path()).remote().pull(progress=CloneProgress(self))
        self.executed = True

    def complete(self):
        return self.executed
```

```python
class PullRepository(luigi.Task):
    repo = luigi.Parameter()

    def run(self):
        git.Repo(self.repo.path()).remote().pull(progress=CloneProgress(self))
        os.utime(self.repo.path())

    def output(self):
        return RecentlyUpdatedTarget(self.repo.path())

class RecentlyUpdatedTarget(luigi.Target):
    def __init__(self, path):
        self.path = path

    def exists(self):
        return os.path.getmtime(self.path) + 10 > time.time()
```

```python
class PullRepository(luigi.Task):
    repo = luigi.Parameter()
    date = luigi.DateParameter(default=datetime.date.today())

    def run(self):
        git.Repo(self.repo.path()).remote().pull(progress=CloneProgress(self))
        os.utime(self.repo.path(), self.date)

    def output(self):
        return RecentlyUpdatedTarget(self.repo.path(), self.date)

class RecentlyUpdatedTarget(luigi.Target):
    def __init__(self, path, date):
        self.path = path
        self.date = date

    def exists(self):
        return os.path.getmtime(self.path) + 10 > self.date
```

¿ Preguntas ?

CARTO

¡ Gracias !