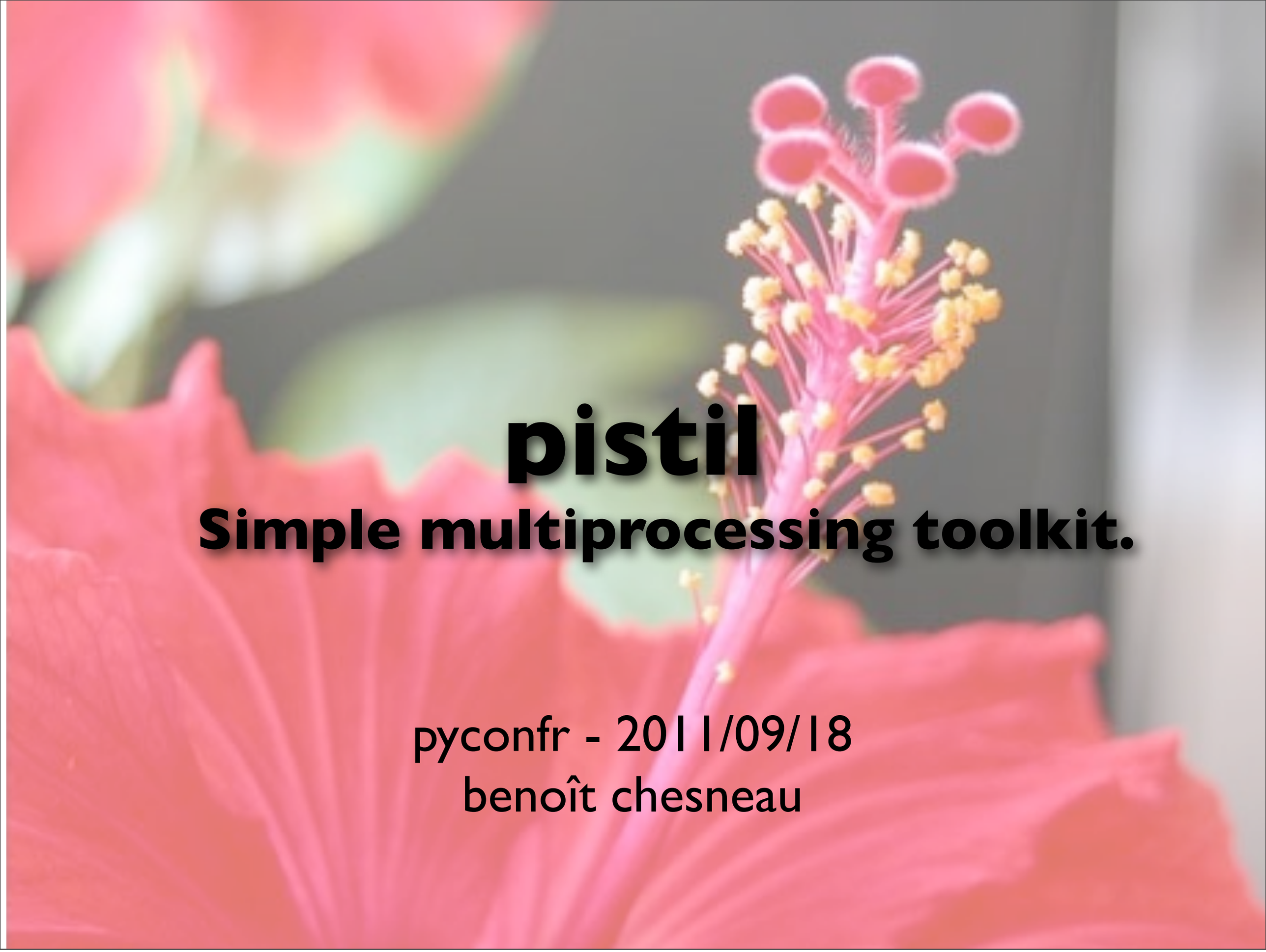

pistil

Simple multiprocessing toolkit.

pyconfr - 2011/09/18

benoît chesneau

- multiprocessing toolkit
- réutiliser le moteur de gunicorn
- version 0.2
- https://github.com/benoitc/pistil

- fork & pre-fork
- plusieurs types de workers (supervisor, kill, worker)
- supervision
- gestion des signaux (HUP, KILL, ...)

- Permet de créer des serveurs TCP
- ou n'importe quel autre type de services
- compatible avec gevent, eventlet & ...

```
from pistil.arbiter import Arbiter
from pistil.worker import Worker

class MyWorker(Worker):

    def handle(self):
        print "hello from worker n°%s" % self.pid

if __name__ == "__main__":
    conf = {}
    specs = [(MyWorker, 30, "worker", {}, "test")]
    a = Arbiter(conf, specs)
    a.run()
```

spec: definit le type de worker

configuration

arbiter - supervise tous les “workers”

Pool de workers

```
from pistil.pool import PoolArbiter
from pistil.worker import Worker

class MyWorker(Worker):

    def handle(self):
        print "hello from worker n°%s" % self.pid

if __name__ == "__main__":
    conf = {"num_workers": 3 }
    spec = (MyWorker, 30, "worker", {}, "test",)
    a = PoolArbiter(conf, spec)
    a.run()
```

Arbitre une pool de “workers”

Multi worker

```
class MyWorker(Worker):

    def handle(self):
        print "hello worker 1 from %s" % self.name
```

Multi worker

```
class MyWorker2(Worker):

    def handle(self):
        print "hello worker 2 from %s" % self.name
```

Multi worker

specification des workers

```
if __name__ == '__main__':
    conf = {}

    specs = [
        (MyWorker, 30, "worker", {}, "w1"),
        (MyWorker2, 30, "worker", {}, "w2"),
        (MyWorker2, 30, "kill", {}, "w3")
    ]
    # launchh the arbiter
    arbiter = Arbiter(conf, specs)
    arbiter.run()
```

TCP worker

socket

addresse du client

```
class MyTcpWorker(TcpSyncWorker):

    def handle(self, sock, addr):
        p = HttpStream(SocketReader(sock))

        path = p.path()
        data = "hello world"
        sock.send("".join(["HTTP/1.1 200 OK\r\n",
                           "Content-Type: text/html\r\n",
                           "Content-Length:" + str(len(data)) + "\r\n",
                            "Connection: close\r\n\r\n",
                            data]))
```

http-parser
https://github.com/benoitc/http-parser

TCP worker

```
if __name__ == '__main__':
    conf = {"num_workers": 3}
    spec = (MyTcpWorker, 30, "worker", {}, "worker",)

    arbiter = TcpArbiter(conf, spec)

    arbiter.run()
```

arbitre TCP

masturbation (multi worker tcp)

```
class UrlWorker(Worker):

    def run(self):
        print "ici"
        while self.alive:
            time.sleep(0.1)
            f = urllib2.urlopen("http://localhost:5000")
            print f.read()
            self.notify()
```

masturbation (multi worker tcp)

arbiter TCP

```
if __name__ == '__main__':
    conf = {"num_workers": 3, "address": ("127.0.0.1", 5000)}

    specs = [
        (MyPoolArbiter, 30, "supervisor", {}, "tcp_pool"),
        (UrlWorker, 30, "worker", {}, "grabber")
    ]

    arbiter = Arbiter(conf, specs)
    arbiter.run()
```

worker

- créer un serveur
- Ex: fserve: serveur de fichiers statiques
  https://github.com/benoitc/fserve

hooks

- on_init
- pre/post fork
- ...

Todo

- Queues multiprocess (named pipe)
- Messaging entre workers et nodes: **flower**

https://github.com/benoitc/pistil