

# *NoSQL adatbáziskezelők*

## *A Redis kulcs-érték adattároló rendszer*

Szathmáry László  
Debreceni Egyetem  
Informatikai Kar

(utolsó módosítás: 2018. dec. 6.)

# Redis

<http://redis.io>

A Redis egy nyílt forráskódú, kulcs-érték adattároló rendszer, ahol az adatok a memóriában helyezkednek el. A kulcs-érték adattároló rendszerek között a Redis a legnépszerűbb.

További jellemzők:

1. C-ben lett implementálva
2. az első változat 2009-ben jött ki
3. több platformra is létezik
4. nagyon sok programozási nyelvhez készült interfész (pl. C, C++, C#, Erlang, Go, Haskell, Java, Lua, Perl, PHP, Python, R, Ruby, Scala, stb.)
5. gyorsan megtanulható

Aktívan fejlesztik. BSD licenc alatt érhető el.

Leegyszerűsítve, a Redis egy hash szótár, ahol a kulcsokhoz értéket lehet rendelni. Viszont, és itt tér el más hasonló adattárolóktól, a Redis esetében az érték nem csak sztring lehet!

Az érték lehet:

- sztring
- lista
- halmaz
- rendezett halmaz (az elemek rendezve lesznek egy *score* érték alapján)
- hash

A szerver oldalon magas szintű műveletek is elvégezhetők, pl. lista rendezése, halmazműveletek (unió, metszet, különbség), stb.

## Perzisztencia

A Redis a teljes adathalmazt a memóriában kezeli. Viszont:

1. Kilépéskor lemezre lehet menteni az adatbázis tartalmát.
2. Indításkor a lementett adatbázist be lehet tölteni.

Továbbá a rendszer bizonyos időközönként automatikusan mentéseket végez (*snapshotting*). Ez lehet teljes vagy inkrementális.

## Teljesítmény

C-ben lett implementálva (nagyon gyors); a memóriában dolgozik (kevés az I/O művelet, lásd fent).

Egyetlen processzben fut, ill. egyetlen szálat használ (*single threaded*)

=> a műveletek atomi műveletek.

## Tranzakció

Minden művelet atomi. Ha több műveletet szeretnénk egy atomi csoportba összefogni, akkor a tranzakciók segítségével ez megoldható.

# Telepítés forrásból

<http://redis.io/download>

```
# keressük meg a legfrissebb verziót
$ wget http://download.redis.io/releases/redis-2.8.19.tar.gz
$ tar xvf redis-2.8.19.tar.gz
$ cd redis-2.8.19
$ make
$ make test
# opcionális, Ubuntu alatt a /usr/local/bin -be telepíti:
$ sudo make install
```

# Telepítés csomagkezelővel #1

Telepítés **Ubuntu** Linux rendszeren

```
$ sudo apt install redis
```

A „redis-server” csomag tartalmazza a szerveret, míg a parancssoros kliens a „redis-tools” csomagban található. A fenti parancs hatására mindkettő feltelepül.

# Telepítés csomagkezelővel #2

Telepítés **Manjaro** Linux rendszeren:

```
$ sudo pacman -S redis
```

A Manjaro egy Arch Linuxra épülő disztribúció, mely az utóbbi időben egyre nagyobb népszerűsége tett szert. Jól összeállított rendszer, mely az Ubuntu-hoz hasonlóan könnyen telepíthető, könnyen használható.

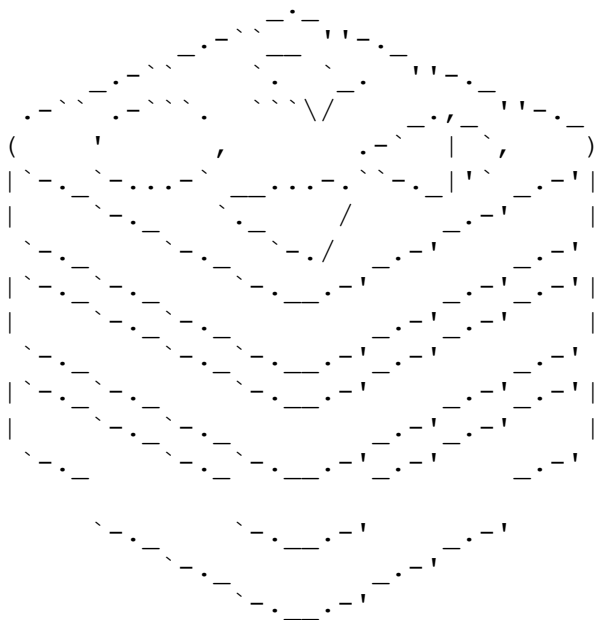
A Manjaro repository-kba nagyon gyorsan bekerülnek az új verziók, így itt valószínűleg nem szükséges a forrásból való telepítés.

Itt csak egyetlen csomagot szükséges telepíteni, mely mind a szerveret, mind pedig a parancssoros klienst tartalmazza.

# Manuális indítás

A szervert manuálisan a `redis-server` parancs kiadásával tudjuk elindítani:

```
$ redis-server
[11650] 29 Sep 13:44:31.595 # Warning: no config file specified, using the default config. In
order to specify a config file use redis-server /path/to/redis.conf
[11650] 29 Sep 13:44:31.596 # Unable to set the max number of files limit to 10032 (Operation
not permitted), setting the max clients configuration to 3984.
```



**Redis 2.6.16** (00000000/0) **64 bit**

← verzió

Running in stand alone mode

**Port: 6379**

PID: 11650

← alapértelmezett port

<http://redis.io>

```
[11650] 29 Sep 13:44:31.596 # Server started, Redis version 2.6.16
[11650] 29 Sep 13:44:31.599 * DB loaded from disk: 0.003 seconds
[11650] 29 Sep 13:44:31.599 * The server is now ready to accept connections on port 6379
```



# Automatikus indítás #1

## Ubuntu rendszeren

Ubuntu esetén a telepítő beregisztrálja, hogy a redis szerver bootolás után automatikusan elinduljon. Forrásból való telepítés esetén az itt leírtak szerint járjunk el:

<https://www.digitalocean.com/community/articles/how-to-install-and-use-redis>

Arról, hogy fut-e a szerver, a legegyszerűbben így tudunk meggyőződni:

```
$ redis-cli  
127.0.0.1:6379>
```

Ha a kliens indításakor nem kapunk hibaüzenetet, akkor fut a szerver.

# Automatikus indítás #2

## Manjaro rendszeren

Manjaro Linuxon a bootolás utáni automatikus indítást engedélyezni kell:

```
$ systemctl enable redis.service
```

Ha nem akarjuk újraindítani a rendszert, akkor a köv. paranccsal tudjuk elindítani a redis szolgáltatást:

```
$ systemctl start redis.service
```

Itt is a „redis-cli” paranccsal tudjuk a legegyszerűbben leellenőrizni, hogy fut-e a szerver:

```
$ redis-cli  
127.0.0.1:6379>
```

# Adatszerkezetek

A Redis **öt** különböző adatszerkezetet használ. Ebből csak egy tekinthető tipikusan kulcs-érték struktúrának.

**Kulcs:** tetszőleges sztring, pl.: `users:jabba`

A kettőspontnak nincs speciális jelentése, elhatárolónak szokták használni.

**Érték:** a kulcshoz rendelt érték

A legtöbb esetben a Redis ezt egy byte-sorozatnak veszi s nem foglalkozik vele, hogy mi van benne.

```
$ redis-cli
redis 127.0.0.1:6379> set users:jabba "Jabba Laci"
OK
redis 127.0.0.1:6379> get users
(nil)
redis 127.0.0.1:6379> get users:jabba
"Jabba Laci"
redis 127.0.0.1:6379>
```



parancssoros kliens

# Adatszerkezetek

*„A kulcs minden, az érték semmi.”*

Azaz, a Redis-ben az értékekre nem lehet lekérdezéseket írni. Pl. egy JSON érték a Redis számára csupán egy közösleges sztring.

A Redis nagyszerűen használható bizonyos feladatokra, de nem nyújt általános megoldást mindenre. Nem arra lett kitalálva, hogy RDBMS rendszereket váltson le.

A Redis-ben minden parancs egy bizonyos adatszerkezetre vonatkozik. A parancsok teljes listája itt tekinthető meg: <http://redis.io/commands>.

1. sztring
2. hash
3. lista
4. halmaz
5. rendezett halmaz

példák később


# Webes adminisztrációs felület

A *Redis Commander* egy node.js-ben írt webes adminisztrációs felület a Redis-hez. Szerepét tekintve a PHPMyAdmin-hoz lehetne hasonlítani.

<http://nearinfinity.github.io/redis-commander/>

## Telepítése

```
$ sudo npm install -g redis-commander
$ redis-commander
path.existsSync is now called `fs.existsSync`.
listening on 8081
Redis Connection 127.0.0.1:6379 Using Redis DB #0
```

 **redis Commander**

Refresh

Commands

Add Server

127.0.0.1:6379:0

users:\* (1)

abc jabba

Add New Key...

Disconnect

Name	Value
# server	
Redis version	2.6.16
Redis git sha1	00000000
Redis git dirty	0
Redis mode	standalone
Os	Linux 3.8.0-31-generic x86_64
Arch bits	64
Multiplexing api	epoll
Gcc version	4.7.3
Process	11650
Run	6d06dab587abe60085d9f5c5e38e378336bea373
Tcp port	6379
Uptime in seconds	1161

127.0.0.1:6379:0

Current Instance: 127.0.0.1:6379:0

redis>

A Redis Commander webes felülete (localhost:8081)

# GUI adminisztrációs felület

Egy másik adminisztrációs felület a Redis Desktop Manager, mely egy C++ -ban írt natív alkalmazás.

## Telepítése

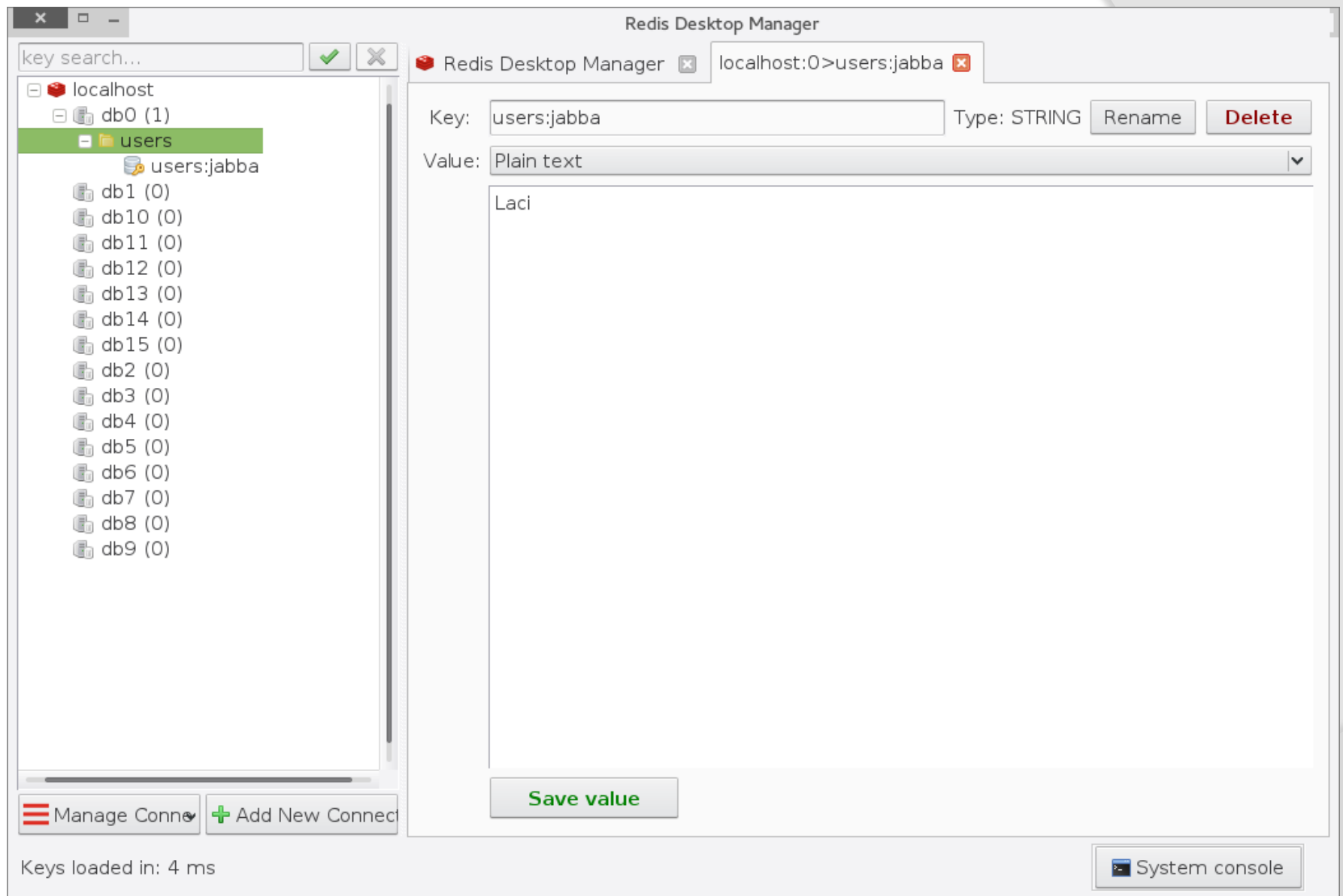
<http://redisdesktop.com/>

```
$ sudo snap install redis-desktop-manager
```

Indítása:

```
$ redis-desktop-manager.rdm
```

Ehhez az kell, hogy a `/snap/bin` mappa benne legyen a PATH-ban.



A Redis Desktop Manager felülete



# A Redis használata alkalmazásokból

A Redis-t számos programozási nyelvből tudjuk használni. Nézzünk meg néhány példát a Python programozási nyelv segítségével.

Először is telepítsük fel a `redis` csomagot:

```
sudo pip3 install redis -U
```

Ellenőrizzük le, hogy sikeres volt-e a telepítés:

```
[08:34:09] ~ $ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>> █
```

Rendben, az importálás nem jelzett hibát.

# Kapcsolódás a szerverhez

```
1 #!/usr/bin/env python
2
3 import sys
4 import redis
5
6 r = redis.Redis("localhost", 6379)
7 #r = redis.Redis() # use default values
8
9 def check_server():
10     try:
11         r.info()
12     except redis.exceptions.ConnectionError:
13         print >>sys.stderr, "Error: cannot connect to redis server. Is the server running?"
14         sys.exit(1)
15
16
17 def main():
18     check_server()
19     print "The server is up and running."
20
21 #####
22
23 if __name__ == "__main__":
24     main()
```



# Példák #1

```
>>> import redis
>>> r = redis.Redis()      # default: localhost, port 6379
>>> r.set("name", "jabba")
True
>>> r.get("name")
'jabba'

# the list is called "test"
# rpush: right push, i.e. put an element on its right side (tail)
>>> r.rpush("test", 24)
1L
>>> r.rpush("test", 67)
2L
>>> r.rpush("test", 9)
3L
# list all the elements (-1 is the index of the last element)
>>> r.lrange("test", 0, -1)
['24', '67', '9']
# number of elements
>>> r.llen("test")
3
# delete the list if you don't need it anymore
>>> r.delete("test")
1
```

## Példák #2

```
>>> import redis
>>> r = redis.Redis()    # default: localhost, port 6379
>>> r.incr("goku:powerlevel")
1
>>> r.incr("goku:powerlevel")
2
```

Gyakori művelet valamilyen számláló tárolása Redis-ben. Mivel minden művelet atomi, illetve a Redis egyetlen szálát használ, ezért egy Redis adattárolót több processz is használhat konkurrens módon.

Egy termelő-fogyasztó alkalmazás Redis-szel például triviálisan megoldható.

## Példák #3

```
>>> import redis
>>> r = redis.Redis()
>>> r.set("name", "jabba")
True
>>> r.get("name")
'jabba'
>>> r.expire("name", 10)
True
>>> r.ttl("name")
6L
>>> r.ttl("name")
4L
>>> r.ttl("name")
>>> r.get("name")
>>> █
```

automatikusan törlődjön a  
kulcs-érték pár 10 mp. múlva

ttl: time to live

Unix időbélyeg is használható, pl. `r.expireat("name", 1356933600)`

A Redis ideális választás lehet adatok cache-elésére.

# Autentikáció #1

Ha a Redis-t az otthoni gépünkön használjuk, akkor nincs szükség azonosításra. Ha viszont egy szerverre telepítjük fel, akkor már le kell védeni az illetéktelenek elől.

A jelszavas védelemhez a következő sort kell betenni a `/etc/redis.conf` állományba:

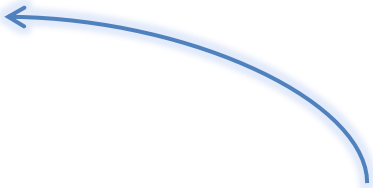
```
requirepass "secret password"
```

Mivel a Redis másodpercenként rengeteg kérést ki tud szolgálni, ezért a Redis szerzője szerint mindenképpen hosszú, nehezen kitalálható jelszót válasszunk. Ellenkező esetben egy egyszerű brute force támadással is hozzá tudnak férni a jelszóhoz.

# Autentikáció #2

Azonosítás parancssorból:

```
$ redis-cli
127.0.0.1:6379> info
NOAUTH Authentication required.
127.0.0.1:6379> auth "secret password"
OK
127.0.0.1:6379> info
# Server
redis_version:2.8.4
redis_git_sha1:00000000
...
```




azonosítás után már  
használható

# Autentikáció #3

Azonosítás Python alkalmazásban:

```
>>> import redis
>>> r = redis.Redis()
>>> r.info()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "/usr/local/lib/python2.7/dist-packages/redis/client.py", line 114, in info
    return self.execute_command('INFO')
  File "/usr/local/lib/python2.7/dist-packages/redis/client.py", line 114, in info
    return self.parse_response(connection, command_name,
  File "/usr/local/lib/python2.7/dist-packages/redis/client.py", line 114, in info
    response = connection.read_response()
  File "/usr/local/lib/python2.7/dist-packages/redis/connection.py", line 114, in read_response
    raise response
ResponseError: NOAUTH Authentication required.
>>> r = redis.Redis(password="secret password")
>>> len(r.info())
67
>>> █
```





# Linkek

- Redis HQ  
<http://redis.io>
- Redis parancsok  
<http://redis.io/commands>
- Karl Seguin: The Little Redis Book  
<http://openmymind.net/2012/1/23/The-Little-Redis-Book/>
- Getting Started: Redis and Python  
<http://degizmo.com/2010/03/22/getting-started-redis-and-python/>
- Redis Python client: redis-py  
<https://github.com/andymccurdy/redis-py>
- Python wrapper for Redis datatypes  
[https://github.com/Doist/redis\\_wrap](https://github.com/Doist/redis_wrap)

# Alternatívák

<http://www.memcached.org/>

Az alternatívák közül a legismertebb a *Memcached*.

A Memcached egy nagy méretű hash táblázatot biztosít, melyben kulcs / érték párokat tudunk tárolni. Általában dinamikus weboldalak használják adatok és objektumok tárolására, lecsökkentve ezzel az adatbázisból való olvasások számát.

Példa:

```
function get_foo(foo_id)
  foo = memcached_get("foo:" . foo_id)
  return foo if defined foo

  foo = fetch_foo_from_database(foo_id)
  memcached_set("foo:" . foo_id, foo)
  return foo
end
```

Előbb a cache-ből próbáljuk kiolvasni az elemet. Ha nincs benne, akkor kiolvassuk az adatbázisból, majd betesszük a cache-be. A 2. hívás esetén már a cache-ből tudjuk kiolvasni az elemet.

# A Redis használata cache-ként

Ha a Redis-t cache-ként akarjuk használni, két lehetőségünk is van.

- 1) A betett elemeknek beállítunk egy élettartamot (expire), így azok egy bizonyos idő múlva törlődnek. Így elejét vehetjük annak, hogy a memória szép lassan beteljen.
- 2) Direkt cache módban használjuk a Redis-t. Ekkor beállíthatjuk, hogy maximum mennyi memóriát használhat. Ha elértük ezt a limitet, akkor a legrégebben használt elemet törli, s így be tudja tenni az új elemet. Vagyis a cache tartalma folyamatosan változik, a régi elemek törlődnek, helyet adva az újaknak. Ehhez a következőt kell beírni a `/etc/redis.conf` állományba:

```
maxmemory 2mb  
maxmemory-policy allkeys-lru
```

2 MB-os memórialimit használata.  
A Redis ekkor a Memcached-hez hasonlóan működik.

# RQ -- Redis Queue

<http://python-rq.org/>



Mivel a Redis-ben minden művelet atomi, illetve egyetlen szálat használ, ezért a termelő - fogyasztó alkalmazásokban nagyszerűen alkalmazható **sorként** is.

A Redis Queue (röviden RQ), mely egy ún. *job queue* rendszer, ezt a feladatot valósítja meg. Vagyis lehetővé teszi *job*-ok sorba való felfűzését, majd ezen *job*-ok háttérben való feldolgozását *worker* processzek segítségével.

A *job*-ok tárolására a Redis-t használja. Előnye a rendkívül egyszerű telepítés és egyszerű használat.

*Job*: tetszőleges Python függvény, melyet a worker processz(ek) aszinkron módon hív(nak) meg. Mivel a *job*-ok serializációjára a Python *pickle* formátumát használja, ezért az RQ csak Python környezetben használható.

Alternatív job queue rendszerek: [Celery](#), [Resque](#), [Huey](#), stb.

Áttekintés: [queues.io](http://queues.io).

# Az RQ telepítése

## Telepítés

```
$ sudo pip install rq
```

## RQ dashboard

Az „rq-dashboard” telepítése opcionális. Ez egy webes felülettel rendelkező monitorozó alkalmazás, mellyel az RQ futását lehet követni.

```
$ sudo pip install rq-dashboard
```

Az „rq-dashboard” paranccsal tudjuk elindítani, majd a <http://0.0.0.0:9181/> címen lehet elérni.

# RQ esettanulmány #1

Nézzük meg az RQ használatát egy konkrét példán keresztül.

Az alábbi címen 38 157 db háttérkép található:

<https://gist.github.com/jabbalaci/10009101>

Egy-egy bejegyzés csupán a file nevét tartalmazza, pl. 95yWLrG.jpg.  
Átalakítás teljes URL-lé:

95yWLrG.jpg => `http://i.imgur.com/95yWLrG.jpg`

Töltsük le a képeket egy adott könyvtárba (pl. `/trash/wallpapers`).  
A letöltésen kívül még felbontás alapján is válogassuk szét a képeket. Ha pl. egy kép 1024x768-as méretű, akkor a kép a  
`/trash/wallpapers/1024x768` nevű könyvtárba kerüljön.

# RQ esettanulmány #2

processor.py

```
8 import os
9 from unipath import Path
10 from PIL import Image
11 import shutil
12
13
14 BASE_DIR = "/trash/wallpapers"
15
16
17 def download(url):
18     fname = Path(url).name
19     fullname = "{folder}/{fname}".format(folder=BASE_DIR, fname=fname)
20     cmd = 'wget --quiet {url} -O {fullname}'.format(url=url, fullname=fullname)
21     os.system(cmd)
22     img = Image.open(fullname)
23     width, height = img.size
24     to_dir = "{base}/{w}x{h}".format(base=BASE_DIR, w=width, h=height)
25     Path(to_dir).mkdir()
26     shutil.move(fullname, to_dir)
```

# RQ esettanulmány #3

main.py

```
4 from redis import Redis
5 from rq import Queue
6
7 from processor import download
8
9 q = Queue(connection=Redis())
10
11
12 def main():
13     with open("list.txt") as f:
14         for line in f:
15             line = line.rstrip("\n")
16             url = "http://i.imgur.com/{img}".format(img=line)
17             q.enqueue(download, url)
```



# RQ esettanulmány #4

## Magyarázat

Egy külön modulba (jelen esetben a `processor.py` fájlba) kitesszük azt a függvényt, aminek a végrehajtása több időt vesz igénybe, s emiatt a háttérben, aszinkron módon szeretnénk majd végrehajtani. A `download` függvény megkapja egy kép URL-jét. Ezt letölti egy adott könyvtárba, majd megnézi a kép dimenzióját (szélesség, magasság). A dimenzióknak megfelelő könyvtárat létrehozza, s ide áthelyezi a képet.

A `main.py` fájlban létrehozunk egy Redis sort. A képek listáját tartalmazó fájlt bejárjuk; minden egyes bejegyzést átalakítunk rendes URL-lé, majd a sorba betesszük a `download` függvényt az URL paraméterrel együtt.

## Worker processz(ek) indítása

Már csak a worker processzt kell elindítani. Ehhez lépünk be a projekt könyvtárába (ahol a `processor.py` található), majd adjuk ki az „`rqworker`” parancsot. Akár több `rqworker`-t is indíthatunk a gyorsabb feldolgozás érdekében.

# RQ esettanulmány #5

## Queues

This list below contains all the registered queues with the number of jobs currently in the queue. Select a queue from above to view all jobs currently pending on the queue.

Queue	Jobs
 default	36730
 failed	0

## Workers

This list below contains all the registered workers.

State	Worker	Queues
▶	vostro.7822	default
▶	vostro.7757	default
▶	vostro.7857	default
▶	vostro.7787	default



## Jobs on default

This list below contains all the registered jobs on queue **default**, sorted by age (oldest on top).

 Requeue All

 Compact

 Empty

Name	Age	Actions
 processor.download("http://i.imgur.com/eUjJ4Yb.jpg") 40935494-fc0c-4dfc-b48f-e7392c9462dc	12 hours ago	 Cancel

Az rq-dashboard webes felülete (localhost:9181).

Bal oldalt: feldolgozandó job-ok száma. Jobb oldalt: jelenleg 4 worker processz fut.

# Celery

<http://www.celeryproject.org/>



Az RQ „nagytestvére” a Celery, mely Python környezetben a legismertebb, legelterjedtebben használt *task queue* / *job queue* rendszer.

A job-ok sorban való feldolgozásán kívül támogatja a crontab-szerű időzített / ismételt feladatvégrehajtást is.

Első lépésben meg kell adnunk egy üzenetküldő rendszert (*message broker*), ami a job-ok sorban való tárolására fog szolgálni. A Celery számos ilyen rendszert támogat:

- [RabbitMQ](#) (Erlang-ban implementált, stabil, éles környezetre ajánlott broker)
- Redis (egyszerűen telepíthető, gyors, de éles környezetben talán kevésbé megbízható mint az előző)
- adatbázis (nem erre lett kitalálva, nem ajánlott)

A továbbiakban az egyszerűség kedvéért a Redis-t fogjuk használni.

# Celery

A Celery az eredmények tárolására egy másik rendszert használ (bár ez megegyezhet az előzővel). A következők közül választhatunk:

- memcached
- Redis
- RabbitMQ
- MongoDB
- stb.

Az egyszerűség kedvéért ismét a Redis-t fogjuk felhasználni erre a célra.

## Telepítés

```
$ sudo pip install celery
```

## Flower

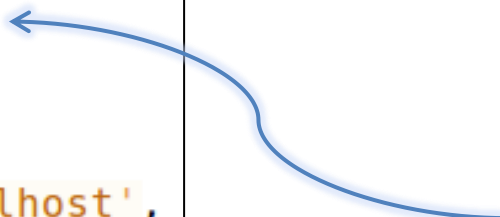
A Celery-hez is létezik egy webes felülettel rendelkező monitorozó alkalmazás, mellyel a Celery futását lehet követni. Ehhez a [flower](#) csomagot kell telepíteni (`sudo pip install flower`), majd a `flower` paranccsal tudjuk elindítani.

# Celery esettanulmány #1

Nézzük meg a Celery használatát egy konkrét példán keresztül. A feladat legyen ugyanaz, mint az RQ esetében, azaz töltsük le a <https://gist.github.com/jabbalaci/10009101> címen található képeket s felbontás szerint válogassuk szét őket külön könyvtárakba.

```
1  from celery import Celery
2
3  celery = Celery(include=['processor'])
4
5  celery.conf.update(
6      BROKER_URL='redis://localhost',
7      CELERY_RESULT_BACKEND='redis://localhost',
8
9      CELERY_TASK_SERIALIZER='json',
10     CELERY_RESULT_SERIALIZER='json',
11     CELERY_ACCEPT_CONTENT=['json'],
12
13     CELERY_TASK_RESULT_EXPIRES=60,
14
15     CELERY_TIMEZONE='Europe/Budapest',
16     CELERY_ENABLE_UTC=True,
17 )
```

config.py



# Celery esettanulmány #2

A projekt konfigurációs részét érdemes kitenni egy külön modulba (config.py állomány). Itt létrehozunk egy celery nevű objektumot, s a későbbiekben ezen keresztül tudjuk majd elérni a Celery rendszert.

A celery objektum konstruktorában egy listában fel kell tüntetni azokat a modulokat, melyek a háttérben végrehajtandó függvényeket tartalmazzák.

A továbbiakban beregisztráljuk, hogy mind az üzenetek küldésére, mind az eredmények tárolására a Redis-t akarjuk használni. A könnyebb olvashatóság kedvéért mindent JSON formátumban teszünk be a Redis-be.

A végén jelezzük, hogy a Redis-ben az eredmények 60 másodperc múlva automatikusan törölődjenek. Ezzel elejét vehetjük annak, hogy a Redis szép lassan megtöltse a memóriát.

# Celery esettanulmány #3

processor.py

```
3 import os
4 import shutil
5
6 from PIL import Image
7 from unipath import Path
8
9 from config import celery
10
11 BASE_DIR = "/trash/wallpapers"
12
13
14 @celery.task
15 def download(url):
16     fname = Path(url).name
17     fullname = "{folder}/{fname}".format(folder=BASE_DIR, fname=fname)
18     cmd = 'wget --quiet {url} -O {fullname}'.format(url=url, fullname=fullname)
19     os.system(cmd)
20     img = Image.open(fullname)
21     width, height = img.size
22     to_dir = "{base}/{w}x{h}".format(base=BASE_DIR, w=width, h=height)
23     Path(to_dir).mkdir()
24     shutil.move(fullname, to_dir)
```

celery objektum behozása

jelezzük, hogy a függvényt egy háttérben végrehajtandó taszknak szánjuk

# Celery esettanulmány #4

main.py

```
3  from processor import download
4
5
6  def main():
7      with open("list.txt") as f:
8          for line in f:
9              line = line.rstrip("\n")
10             url = "http://i.imgur.com/{img}".format(img=line)
11             download.delay(url)
```



A `download` függvényt nem közvetlenül hívjuk meg. A `delay` hívással jelezzük, hogy ezt a függvényt aszinkron módon akarjuk meghívni, amiről majd a Celery fog gondoskodni.



# Celery esettanulmány #5

## Worker processz(ek) indítása

Már csak a worker processzt kell elindítani. Ehhez lépünk be a projekt könyvtárába (ahol a `config.py` található), majd adjuk ki az alábbi parancsot:

```
$ celery -A config worker -l info
```

-A config:	a celery objektum a <code>config.py</code> állományban van definiálva
worker:	task queue indítása
-l info:	naplózás beállítása

Nagy előnye a Celery-nek, hogy egyszerre több szálon is indíthatunk worker processzeket. Például így indíthatunk egyszerre 4 worker processzt:

```
$ celery -A config worker -c 4 -l info
```

# Celery esettanulmány #6

A Flower indítása (maradva az előző példánál):

```
$ flower -A config
```

innen tudja majd, hogy  
az eredmények a Redis-ben  
jelennek meg

Celery Flower



## Workers

☐

	Name	Status	Concurrency	Completed Tasks	Running Tasks	Queues
<input type="checkbox"/>	celery@jabba-nancy	Online	4	1803	4	celery

**Broker:** redis://localhost:6379//

A Flower webes felülete (localhost:5555).  
Mint látható, jelenleg 4 worker processz fut.