DISTRIBUTED LOCKS

WITH REDIS AND PYTHON

Prepared by Sebastian Buczyński

WHO AM I

Software developer
Using mainly gevent, Twisted & Celery.

Working for Focus Telecom Poland

WHAT WE DO @ FOCUS

B2B telco solutions in Software as Service model teleconferences and pbx, but our main product is a Contact Center.

CONTACT CENTER?

A platform for building call centers, hotlines or helpdesks.

GOALS OF THIS TALK

Implementation of distributed locks using Redis and Python

Use in a real life application

NoSQL database - key -> value storage

key ∈ {string}

value ∈ {string, list, map, set, ...}

+ message passing

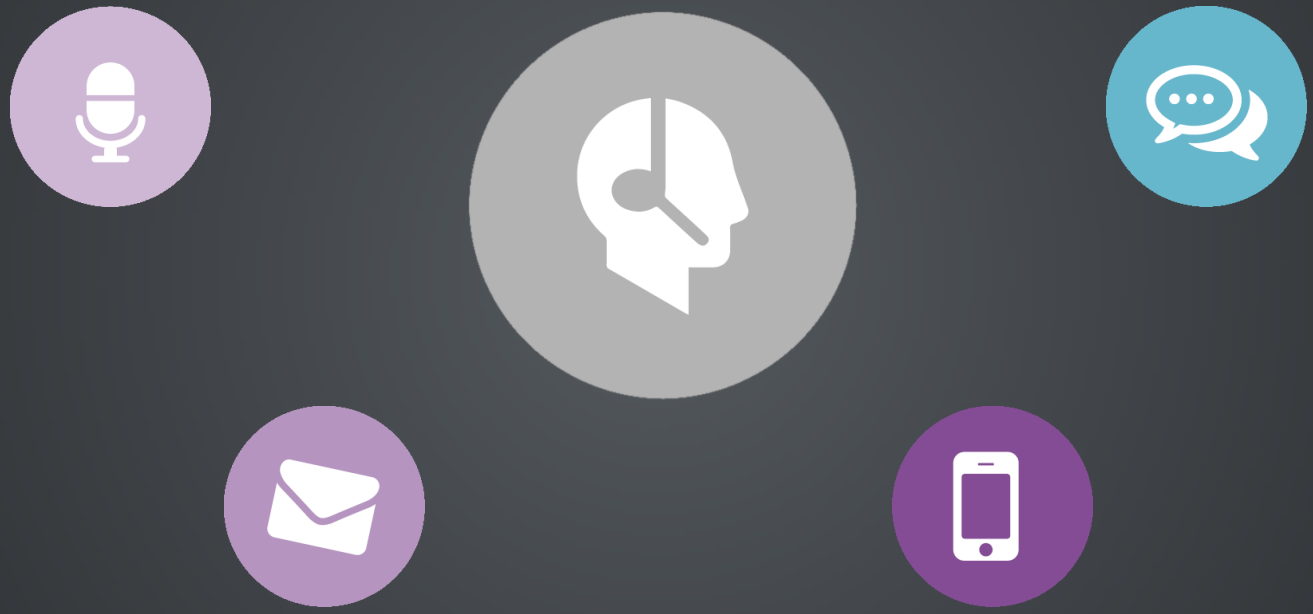
redis.io

LOCK

Software primitive that allows to exclusively access a resource in a way, that nobody else can use it

CASE STUDY - CONTACT CENTER

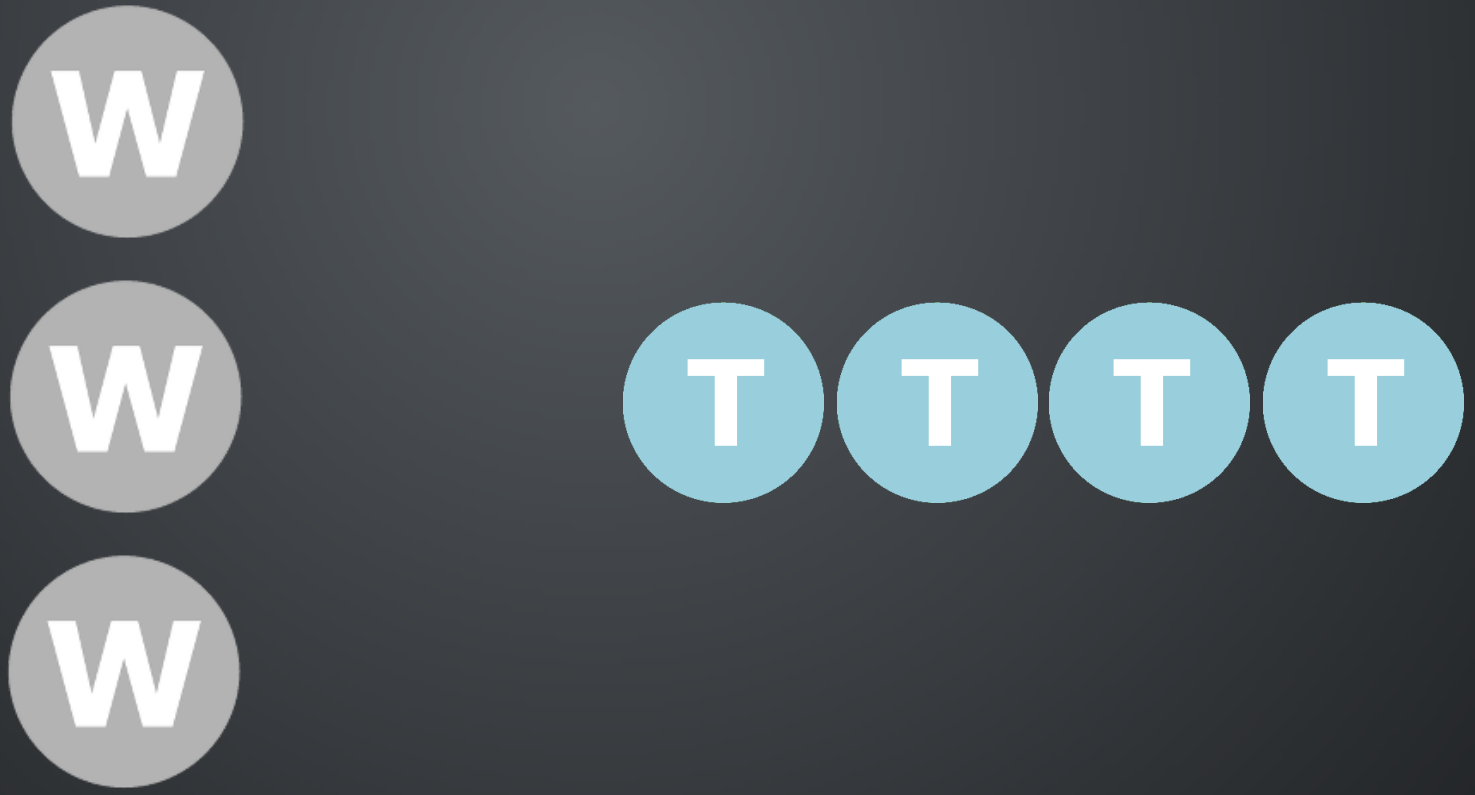
CONTACT CENTER IS ...



Work automation

CONTACT CENTER IS ...

CONTACT CENTER IS LIKE A COMMON TASK QUEUE



NOPE, IT'S NOT

- Worker is not passive
- Tasks are prioritized
- Worker can handle multiple tasks simultaneously

SOLUTION - THE BIG PICTURE

- Event-driven task manager - Twisted based
- Redis - locks server

DISTRIBUTED MUTEX/BINARY SEMAPHORE

- Either locked or unlocked
- Stored as a Redis string
- Non - blocking calls

DISTRIBUTED MUTEX - USE CASE 1

- Keeping state - one lock for each worker
- Locked while working on a task, released afterwards

DISTRIBUTED MUTEX - USE CASE 2

- Choosing task on one's own
- Discontinue work

IMPLEMENTATION DETAILS

Stored in Redis as a string

Need for atomicity

IMPLEMENTATION DETAILS 2

```
GET lock_key
if lock_key == 'unlocked':
        SET lock_key 'locked'
```

IMPLEMENTATION DETAILS 2

```
WATCH lock_key
GET lock_key
if lock_key == 'unlocked':
        SET lock_key 'locked'
else:
        UNWATCH
```

IMPLEMENTATION DETAILS 2

```
WATCH lock_key
GET lock_key
if lock_key == 'unlocked':
        MULTI
        SET lock_key 'locked'
        EXEC
else:
        UNWATCH
```

PYTHON PART - TASK MANAGER

Should be notified

```
PUBLISH locks_changes_channel lock_key_locked
```

To receive notifications

```
SUBSCRIBE locks_changes_channel
```

PYTHON PART - TWISTED POWERED

```
from txredis.client import RedisSubscriber

class LockSubscriber(RedisSubscriber):
    def messageReceived(self, channel, messa
        # do some stuff
```

DISTRIBUTED SEMAPHORE

- threading.Semaphore
- can be acquired/released few times

```
s = Semaphore(2)

s.acquire()
s.acquire()
# s.acquire() # Exception

s.release()
s.release()
# s.release() # that too
```

- Stored as a Redis list
- Blocking calls

DISTRIBUTED SEMAPHORE - USE CASE

- Multiple tasks at the same time

IMPLEMENTATION DETAILS

Stored in Redis as a list

IMPLEMENTATION DETAILS 2

Acquiring

```
BRPOP semaphore_key some_timeout
```

Releasing

```
RPUSH semaphore_key some_val
```

PYTHON PART - WITHOUT CHANGES

...but personally never needed notifications on these.

SEMAPHORE'S STATE AFTER CHANGING IT

Approach one - wrap with MULTI - EXEC

Simpler in this case - write a lua script

```
redis.call('RPUSH', 'semaphore_key', 'some_va
local count = redis.call('LLEN', 'semaphore_k
return count
```

Evaluate this:

```
redis-cli EVAL "$(cat semaphore_release.lua)"
```

WARNING!

Making BRPOP inside MULTI/EXEC will return nil

BRPOP inside lua script will result in an error

ALTERNATIVE?

Non blocking - RPOP

FINAL REMARKS

Care about starting conditions

Study carefully control flow in your application

Work up a restoring state procedure

QUESTIONS?

@EnforcerPL