EUROPEAN SPALLATION SOURCE

# Redis and the Configuration Stub

Afonso Mukai
Data Management Group – DMSC

www.europeanspallationsource.se
7 December 2016

# Agenda

EUROPEAN SPALLATION SOURCE

- Environment setup and check
- Introduction to Redis
- Redis data types
- Additional Redis functionality
- Python and Redis
- Nomenclature: stubs and fakes
- The stub configuration service
- Final remarks

# Environment setup

- ECW-DM VirtualBox virtual machine provided
- CentOS 7, based on ICS development machine image
- In VirtualBox:
  - **File**, **Import Appliance…**
- If it fails, extract the contents of the .ova file with tar:

```
$ tar xf ECW-DM.ova
```

  - Import the extracted file
- Start VM

# Redis

EUROPEAN SPALLATION SOURCE

redis Commands Clients Documentation Community Download Support License

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. Learn more →

## Try it

Ready for a test drive? Check this interactive tutorial that will walk you through the most important features of Redis.

## Download it

Redis 3.2.5 is the latest stable version. Interested in release candidates or unstable versions? Check the downloads page.

## Quick links

Follow day-to-day Redis on Twitter and GitHub. Get help or help others by subscribing to our mailing list, we are 5,000 and counting!

## Redis News

- Redis 4.0 RC1 is out! My blog post about it is here: https://t.co/sv37Um6Pgb

# Redis: overview

EUROPEAN SPALLATION SOURCE

- REmote DIctionary Server
- A key-value store
- Values can have more complex data types
- Easy to install and run (available from EPEL)
- Clients for many languages
- Includes publish-subscribe functionality
- Built-in Lua interpreter

# Redis: persistence and configuration

EUROPEAN SPALLATION SOURCE

- In-memory database, with configurable persistence
  - By default, snapshots saved to disk after an interval dependent on the number of keys that changed
  - Alternatively, can update an append-only file on disk at every change
- These and other configurations can be changed in the /etc/redis.conf file

# Hands-on activity: Installing Redis

EUROPEAN SPALLATION SOURCE

- On CentOS:

```
$ sudo yum install epel-release
$ sudo yum install redis
```

- From source code:
  - Download released package from https://redis.io

```
$ tar xf redis-3.2.5.tar.gz
$ cd redis-3.2.5
$ make
$ sudo make install
```

# Hands-on activity: Checking installation

EUROPEAN SPALLATION SOURCE

- Starting server: on one terminal window or tab:

```
$ redis-server
```

- Command-line client: on another terminal window or tab:

```
$ redis-cli
> set mykey value
> get mykey
> keys *
> flushdb
```

# Redis data types

EUROPEAN SPALLATION SOURCE

- **String**
- List
- **Set**
- Sorted set
- **Hash**
- Using string and special commands:
  - Bit array
  - HyperLogLog (set cardinality estimation)

# Redis: keys

EUROPEAN SPALLATION SOURCE

- Each value is identified by a key
- Keys are strings
- Values can be anything
- Commands operate on keys and are specific to a data type
- Maximum allowed size is 512 MB
- Can be set to expire after a certain time elapses

# Redis: strings

EUROPEAN SPALLATION SOURCE

- Simplest data type
- Can be a string of any type
- Maximum value size is 512 MB
- Can be used as a bitmap (`setbit`, `getbit`)
- Supports some additional operations on specialised value types:
  - Atomic increase and decrease on integers
  - HyperLogLogs are encoded as strings

# Redis: sets and hashes

EUROPEAN SPALLATION SOURCE

- Sets
  - Store unique strings
  - Support set operations such as membership test, union, intersection and difference
  - Values are not ordered; ordered set type attaches a floating point value to each member, allowing for easy ranking
- Hashes
  - Store field-value pairs under each key

# Hands-on activity: Adding data to Redis

EUROPEAN SPALLATION SOURCE

- On the command-line interface:

```
> set instrument nmx
> get instrument
> set instrument:scan 1
> set instrument:user afonso
```

- Keys:

```
> keys instrument:*
> exists instrument:user
> type instrument:user
```

# Hands-on activity: Adding data to Redis

EUROPEAN SPALLATION SOURCE

- Commands for integer values:

```
> incr instrument:scan
> incrby instrument:scan 10
> decr instrument:scan
> get instrument:scan
```

- Using strings as bitmaps

```
> setbit instrument:interlocks 0 1
> setbit instrument:interlocks 7 1
> getbit instrument:interlocks 7
```

# Hands-on activity: Adding data to Redis

EUROPEAN SPALLATION SOURCE

- Sets:

```
> sadd instrument:motors ma
> sadd instrument:motors mb
> smembers instrument:motors
> sismember instrument:motors mc
> sadd newinstrument:motors mc md me
> sunion instrument newinstrument
> sinter instrument newinstrument
> spop instrument:motors
```

# Hands-on activity: Adding data to Redis

EUROPEAN SPALLATION SOURCE

- Hashes

```
> hset motor:ma type nice
> hset motor:ma protocol ca
> hmset motor:mb type nice protocol pva
> hgetall motor:ma
> hget motor:mb type
> hmget motor:mb type protocol
> hlen motor:ma
```

# Hands-on activity: Adding data to Redis

EUROPEAN SPALLATION SOURCE

- Key expiration:

```
> del instrument:user
> expire instrument 15
> ttl instrument
> get instrument
```

- Delete keys of current database:

```
> flushdb
```

- Selecting database

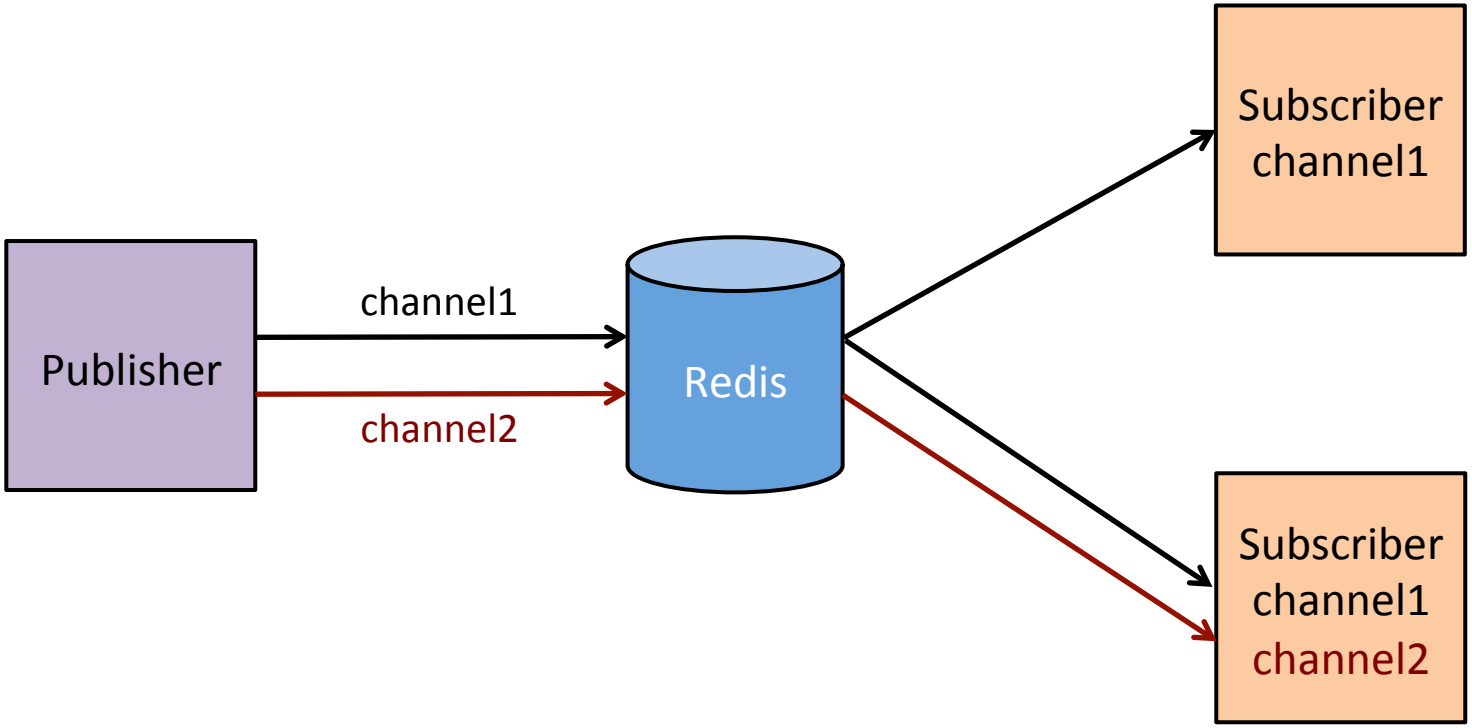
```
> select 1
```

# Redis: transactions

EUROPEAN SPALLATION SOURCE

- Redis commands are atomic (Redis is single-threaded)
- Transactions: execute a group of commands in a single step, sequentially and atomically:
  - Commands are executed in order
  - No other client's commands are executed during it
  - All or none executed
- Scripts are transactional

# Redis: publish/subscribe



# Redis: publish/subscribe

EUROPEAN SPALLATION SOURCE

- Clients subscribe to channels identified by a name
- Publishers write messages to a given channel
- Subscribers will then receive messages from the channels they subscribed to
  - Messages sent before the subscription start or while client is disconnected are missed
- Pattern-matching can be used when subscribing
- Messages are sent to all databases in the server

# Redis: keyspace notifications

EUROPEAN SPALLATION SOURCE

- Clients can subscribe to notifications of changes to the data set
- Feature has to be activated in configuration (either in redis.conf or user the CONFIG SET command)
- Notifications can be activated for a subset of the event types
- Different commands generate different messages, clients can filter them using the subscription string

# Hands-on activity: Publish/subscribe

EUROPEAN SPALLATION SOURCE

- Open three command-line clients
- Subscriber:

```
> subscribe channel1
```

- Pattern-matching subscriber:

```
> psubscribe channel*
```

- Publisher:

```
> publish channel1 hello
> publish channel1 “Welcome to ESS”
> publish channel2 “no receiver”
```

# Hands-on activity: Keyspace notifications

EUROPEAN SPALLATION SOURCE

- Enable all possible keyspace notifications:

```
> config set notify-keyspace-events KEA
```

- Subscriber:

```
> psubscribe __key*__:*
```

- In another client session:

```
> set newkey newvalue
> expire newkey 5
> set anotherkey anothervalue
> del anotherkey
```

# Redis: clients

EUROPEAN SPALLATION SOURCE

## Clients

The recommended client(s) for a language are marked with a ⭐.

Clients with some activity in the official repository within the latest six months are marked with a ☺.

Want **your client listed here?** Please fork the redis-doc repository and edit the clients.json file. **Submit a pull request** and you are done.

Browse by language:

| | | | | | |
|---|---|---|---|---|---|
| ActionScript | Bash | C | C# | C++ | Clojure |
| Common Lisp | Crystal | D | Dart | Delphi | Elixir |
| emacs lisp | Erlang | Fancy | gawk | GNU Prolog | Go |
| Haskell | Haxe | Io | Java | Julia | Lasso |
| Lua | Matlab | mruby | Nim | Node.js | Objective-C |
| OCaml | Pascal | Perl | PHP | Pure Data | Python |
| R | Racket | Rebol | Ruby | Rust | Scala |
| Scheme | Smalltalk | Swift | Tcl | VB | VCL |

# Hands-on activity: redis-py

EUROPEAN SPALLATION SOURCE

- redis-py is the recommended Python client; install it using pip:

```
$ sudo pip install redis
```

- Import it:

```
$ ipython
> import redis
> print(redis.VERSION)
```

# Hands-on activity: redis-py

EUROPEAN SPALLATION SOURCE

- Basic commands:

```
> r = redis.StrictlyRedis()
> r.keys('*')
> r.set('instrument', 'loki')
> r.hset('motor:mf', 'type', 'nice')
> r.hset('motor:mf', 'protocol',
  'custom')
> r.hgetall('motor:mf')
```

# Hands-on activity: redis-py

EUROPEAN SPALLATION SOURCE

- Transactions with pipelines:

```
> p = r.pipeline()
> p.set('index', 2)
> p.incr('index')
> p.incrby('index', 10)
> p.execute()
> r.get('index')
```

# Hands-on activity: redis-py

EUROPEAN SPALLATION SOURCE

- Publish-subscribe:

```
> r.publish('channel3', 'message 1')
> ps = r.pubsub()
> ps.subscribe('channel3')
> r.publish('channel3', 'message 2')
> r.publish('channel3', 'message 3')
> ps.get_message()
> ps.get_message()
```

# Hands-on activity: redis-py

EUROPEAN SPALLATION SOURCE

- Adding a message handler

```
> def handler(message):
      print('inside handler', message)
> ps.subscribe(**{'channel4': handler})
> r.publish('channel4', 'message 1')
> r.publish('channel4', 'message 2')
> ps.get_message()
> ps.get_message()
```

# Test doubles and nomenclature

EUROPEAN SPALLATION SOURCE

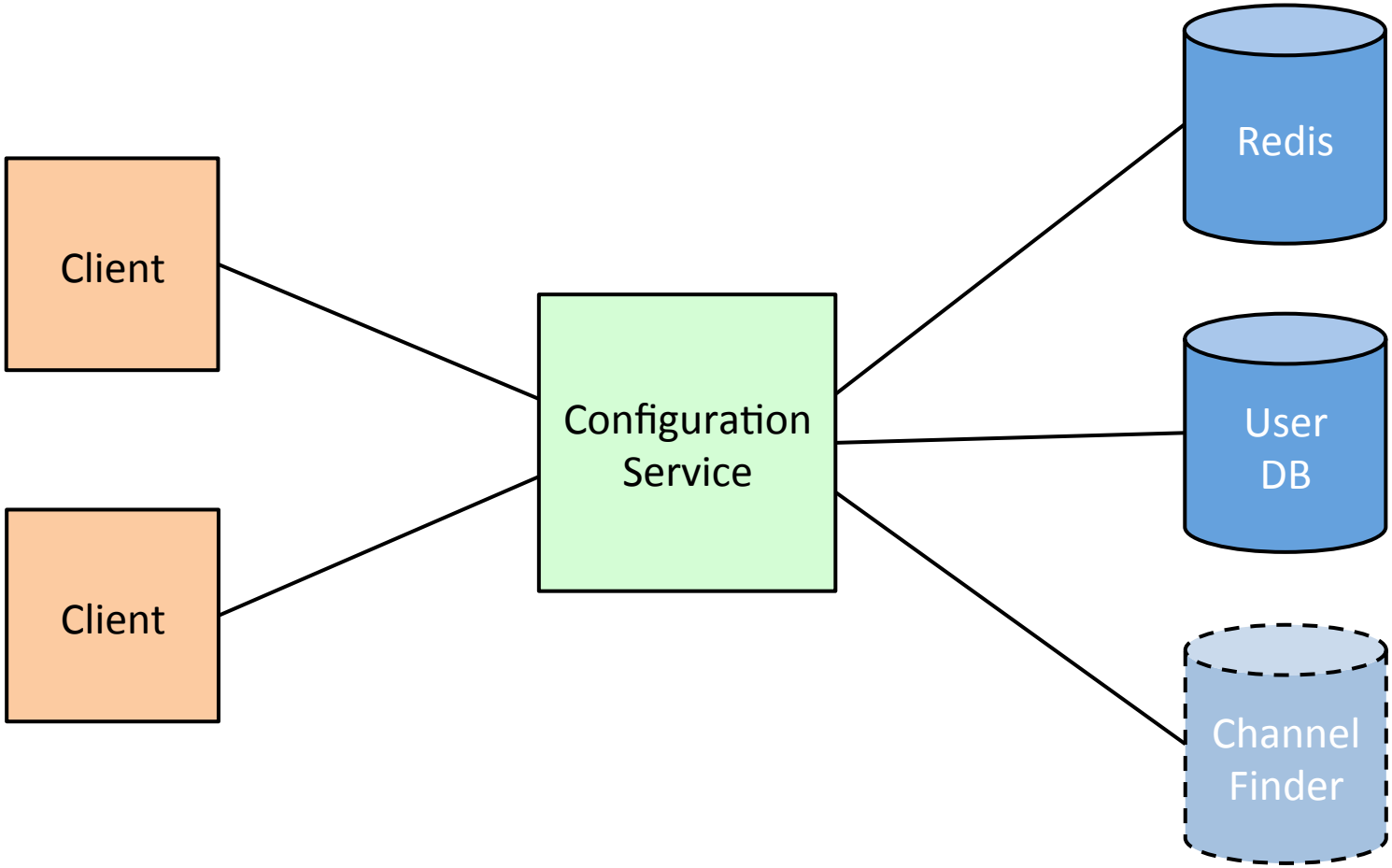
- Test doubles substitute a real object during testing
- Meszaros classification:
  - Stubs provided predefined answers to requests, and will usually not respond to anything else
  - Fake objects have a working implementation with shortcuts
- The stub configuration service:
  - Not really a unit test double
  - Probably more a fake than a stub

# The stub configuration service

EUROPEAN SPALLATION SOURCE

- Where client software gets configuration and metadata, such as:
  - What detector data should be aggregated, protocols to use, addresses
  - What PVs should be aggregated
  - User and experiment metadata: what experiment is running at the instrument, who the current user is
- As we have not made a decision about it yet, a place to put the required configuration for now, gathering requirements in the process

# Stub configuration service: architecture



# Stub configuration service: clients

EUROPEAN SPALLATION SOURCE

- Use the service to get configuration information, without having to hard-code it into software
- Identify what configuration information is needed and where in the client code
- Access to configuration should be kept isolated in order to be easily changed
- The service allows multiple clients to get the same configuration

# Stub configuration service: sources

EUROPEAN SPALLATION SOURCE

- Currently reading data from Redis
  - Arbitrarily structured keys
  - Arbitrary data
- Notification experiment with Kafka
- Could add ChannelFinder for PV information, using properties or tags, for example.

# Hands-on activity: Configuration service

- Clone the Git repository:

```
$ git clone https://bitbucket.org/europeanspallationsource/stub-config-service
$ cd stub-config-service
$ ls
$ cd service/sample_config
$ python add_data.py localhost
```

# Hands-on activity: Configuration service

- Start the service:

```
$ cd ..
$ python configservice.py
```

- Open another terminal tab or window:

```
$ cd stub-config-service/client
$ ipython
```

# Hands-on activity: Configuration service

EUROPEAN SPALLATION SOURCE

- Getting configurations:
  - `import configclient`
  - `c=configclient.ConfigClient('localhost')`
  - `c.get_config('instrument1')`
  - `config = c.get_config('instrument2')`
- The subscription functions are an Apache Kafka notification experiment

# Hands-on activity: Using Kafka in the VM

EUROPEAN SPALLATION SOURCE

- Start ZooKeeper and Kafka:

```
$ sudo systemctl start zookeeper
$ sudo systemctl start kafka
```

- Installed in /opt/dm_group

# Final remarks and discussion

EUROPEAN SPALLATION SOURCE

- Current project is very simple, no changes for some time
- PSI developed a C++ library to get configuration data from Redis: https://bitbucket.org/europeanspallationsource/configuration-manager
- Should we get PV data from ChannelFinder?
- What kind of configuration is required for experiment control?

# Final remarks and discussion

EUROPEAN SPALLATION SOURCE

- What protocol should be used for communication with the service (currently uses ZeroMQ)? REST?
- Do we need automatic configuration change notifications?

# References


- https://redis.io
- http://openmymind.net/redis.pdf
- https://github.com/andymccurdy/redis-py
- https://bitbucket.org/europeanspallationsource/stub-config-service
- Humble, J. and Farley, D. Continuous Delivery. Addison-Wesley Professional, 2010.
- http://martinfowler.com/articles/mocksArentStubs.html