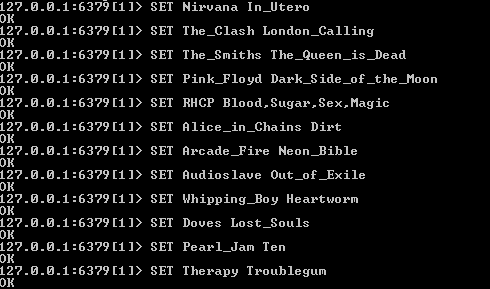
**Portfolio Part 3: Redis**

**T00155775**

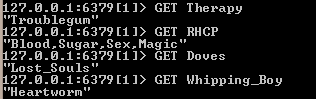
**Joe O Flaherty**

**BSc (Hons) Computing with Multimedia**

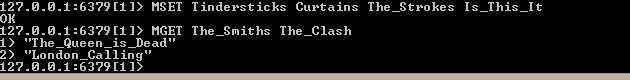
## Implement your own Redis database (showing examples of CRUD)



**C**reating a database/store of bands and favourite albums.



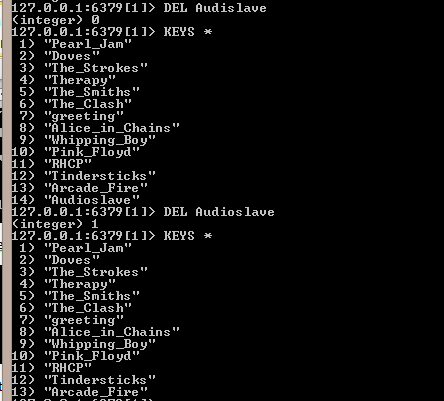
**R**eading values from the database using keys, in this example band names



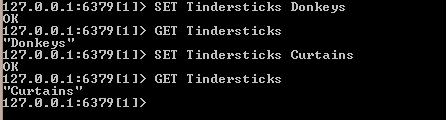
Create and read multiple values using MSET and MGET.



**D**eleting values from the data store, mistake made in second DEL command purposely to show that 1 indicates success, 0 indicates failure.



Using KEY \* command to check list after failed DEL operation, correct command then given and KEYS \* used to confirm the deletion – 14. Audioslave is not visible in the second list.



Example of update operation in Redis, initially Tindersticks is given a value of ‘Donkeys’ confirmed using GET command. Tindersticks is subsequently ‘updated’ to ‘Curtains’ and confirmed again using GET.

## Download, install and implement the same database in memcache.

Memcache is a caching system used to speed up web applications by holding small pieces of arbitrary data, it could be broadly compared to the $\_SESSION variable in PHP, but memcache works across the whole application not just on a per user basis and has been successfully proven to make great gains in speed increases on web applications.

Memcache was designed for Linux and not windows, so it has posed some installation issues because memcache does not come with any installation software and code has to be compiled from source.

Followed installation instructions on <http://zurmo.org/wiki/installing-memcache-on-windows>

Installation proved extremely buggy and problematic with support for the problems encountered lacking. Resulting from this difficulty it was decided to run Memcache on a Linux VM running Ubuntu. Installation on Ubuntu also meant installing MySQL and PHP using **sudo apt-get install mysql-server php5-mysql php5 php5-memcache**

Followed by installing memcached

sudo apt-get install memcached

Then install php-pear, the repository that stores memcache.

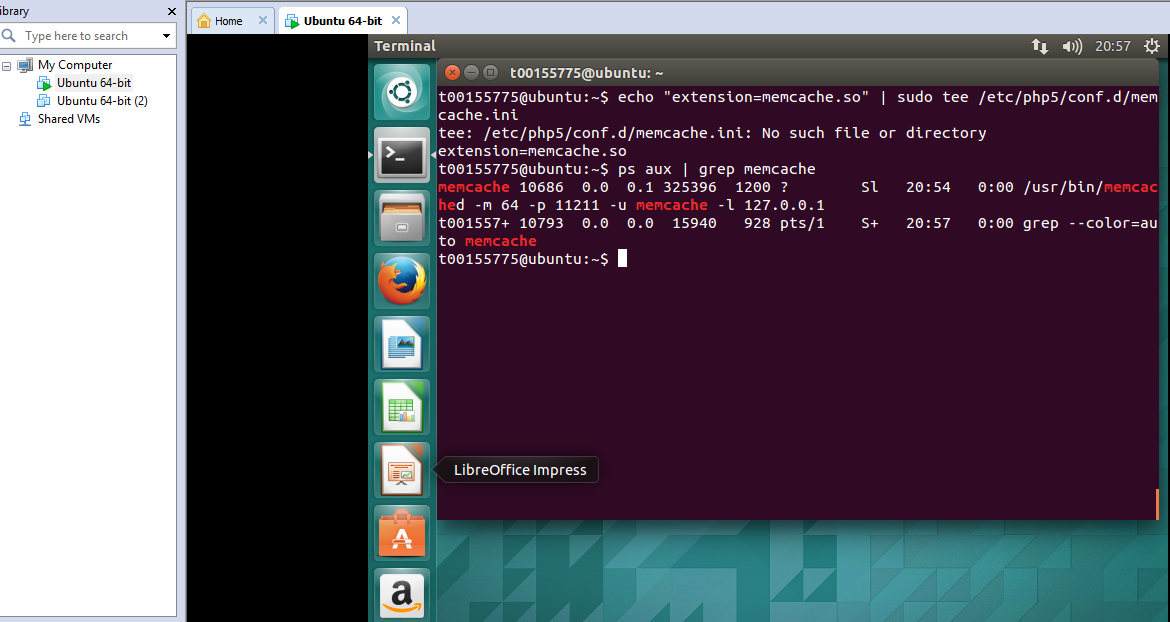
sudo apt-get install php-pear

Next a compiler

sudo apt-get install build-essential

Finally using PECL (PHP Extension Community Library) install memcache:

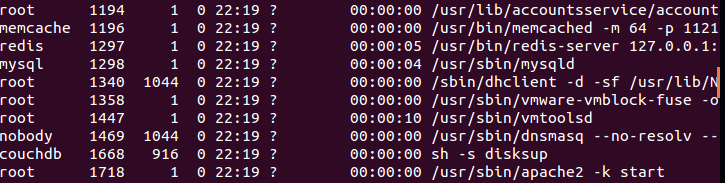
sudo pecl install memcache



Confirming Memcache installation using $ ps aux| grep memcache.

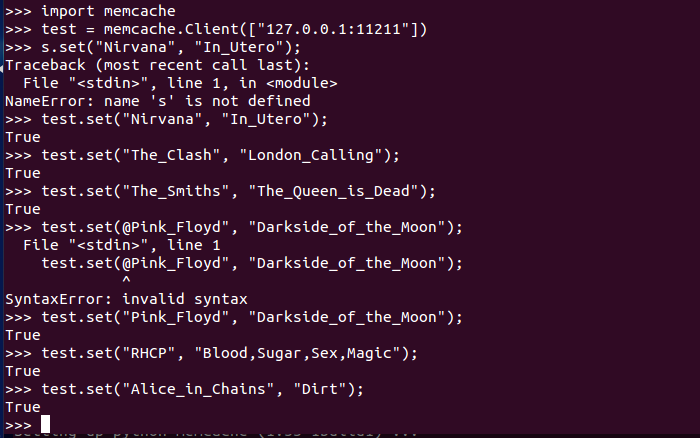


The command ps –ef is used to confirm the port memcache is listening on.

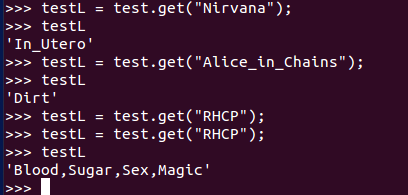


Port 1196 – which proved incorrect.

Having spent a lot of time attempting to install memcache/memcached and use it as a datastore without success, it was decided to access it via a programming language (Python). This proved a far simpler option.



Implementing a database/store in memcache using Python including some typos.

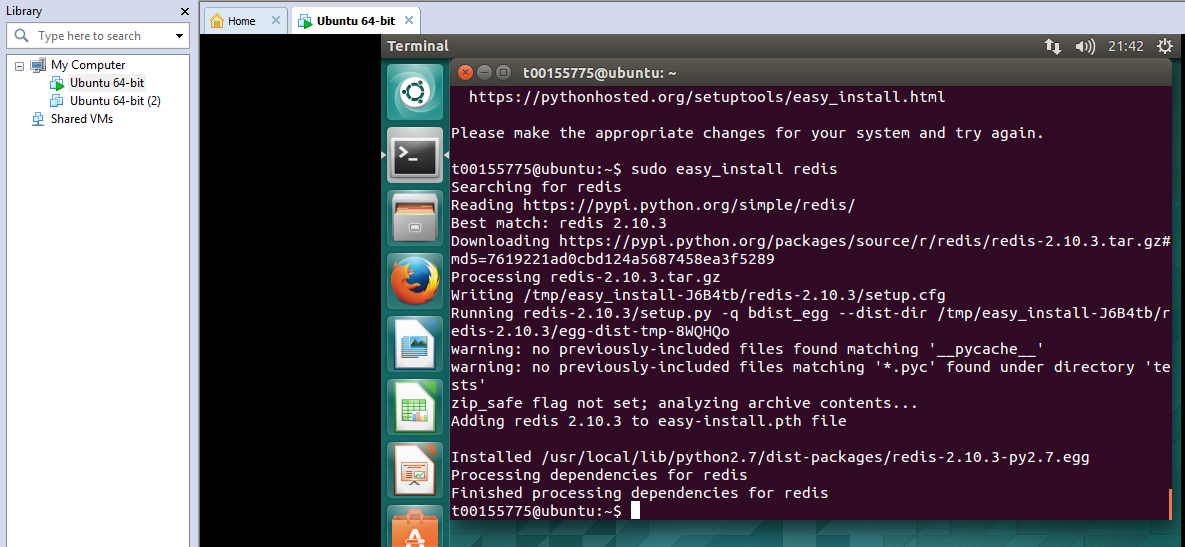


A number of ‘gets’ were performed against memcache via python to confirm that the data was successfully added to memcache.

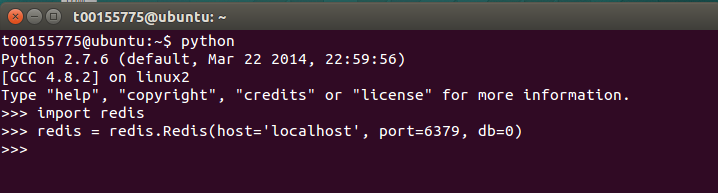
## Compare and contrast memcache with Redis

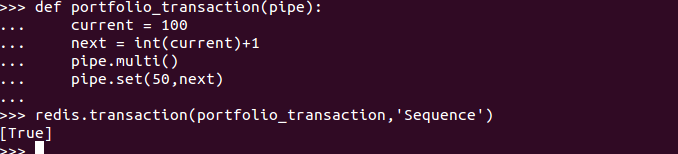
Memcache’s strength is that it is a cache, not a data store, it only holds values for a set length of time for a given key and thus performance can be extremely quick. However Memcache does not provide for persistence, information required by a user may not always be cached which is the major weakness of Memcache. Redis is persistent or at least provides for certain persistence by default, this can be enhanced using provided Save methods and Snapshotting. Memcache stores exclusively strings, Redis can store complex data structures like lists, sets and sorted sets. Redis is also adaptable it can be used as a caching system like Memcache, Memcache cannot be used as a data store like Redis. Memcache is as its name implies a cache and is not really suitable as s data store, Redis can be used with little or no persistence set enabling it to run very much like a cache i.e. extremely quickly, saving can be forced using SAVE or BGSAVE settings.

## Download a driver for a programming language of your choice and connect to a Redis Server. Insert and increment a value using a transaction.



Installing Redis and the python packages required from Linux (Ubuntu) terminal, a very straight forward process.

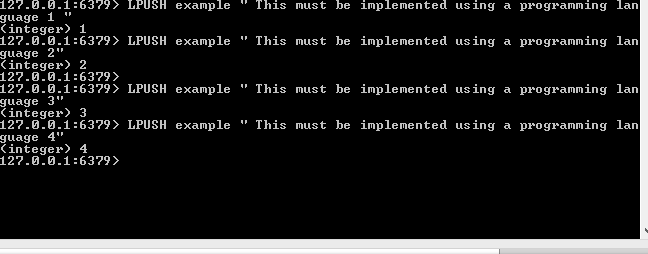


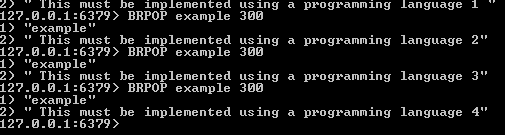


Defining a Python function portfolio\_transaction that simply adds 1 to a variable ‘current’ each time the function is called as a parameter in a Redis transaction. True is returned if the function transaction has executed correctly.

## Using your driver of choice, create a program that reads a blocking list and outputs somewhere (console, file, Socket.io, and so on) and another that writes to the same list. Back it up with a Redis master-slave replicated cluster across multiple nodes as your back end.

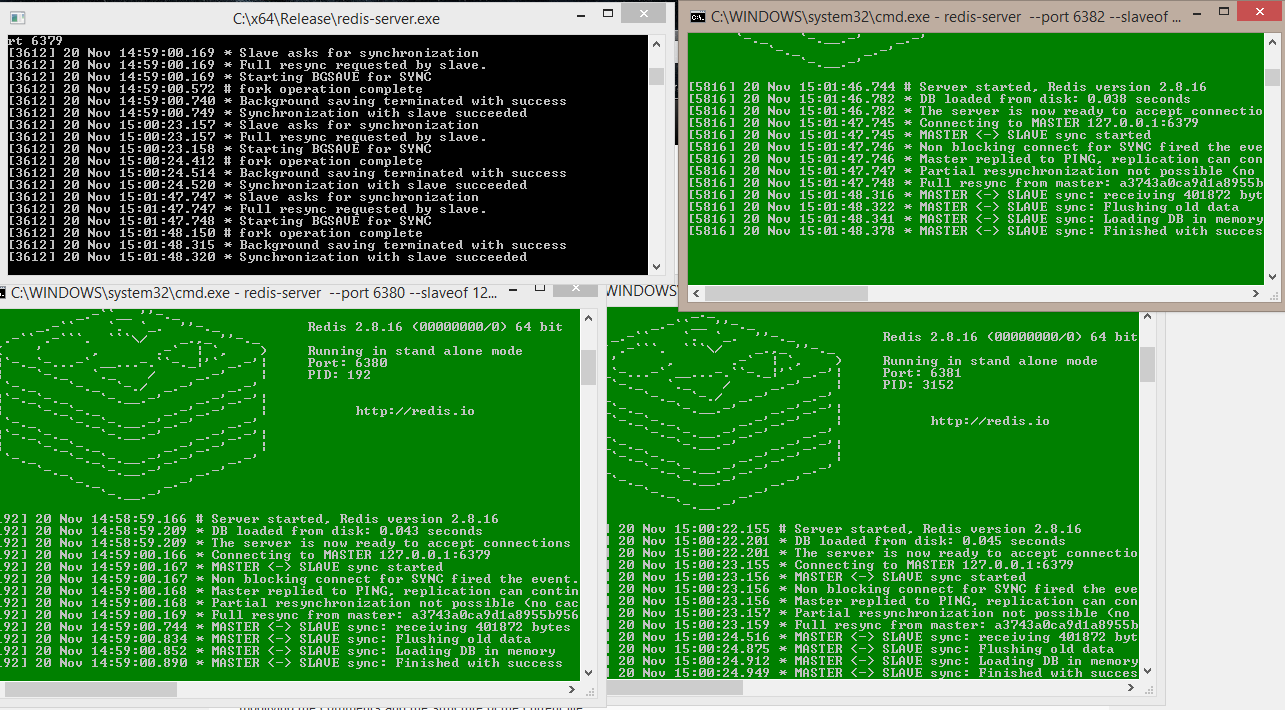
Based on the concept of a client or multiple clients having the ability to write to a blocking list and one client acting as a consumer (digester) of the contents of the list as shown.





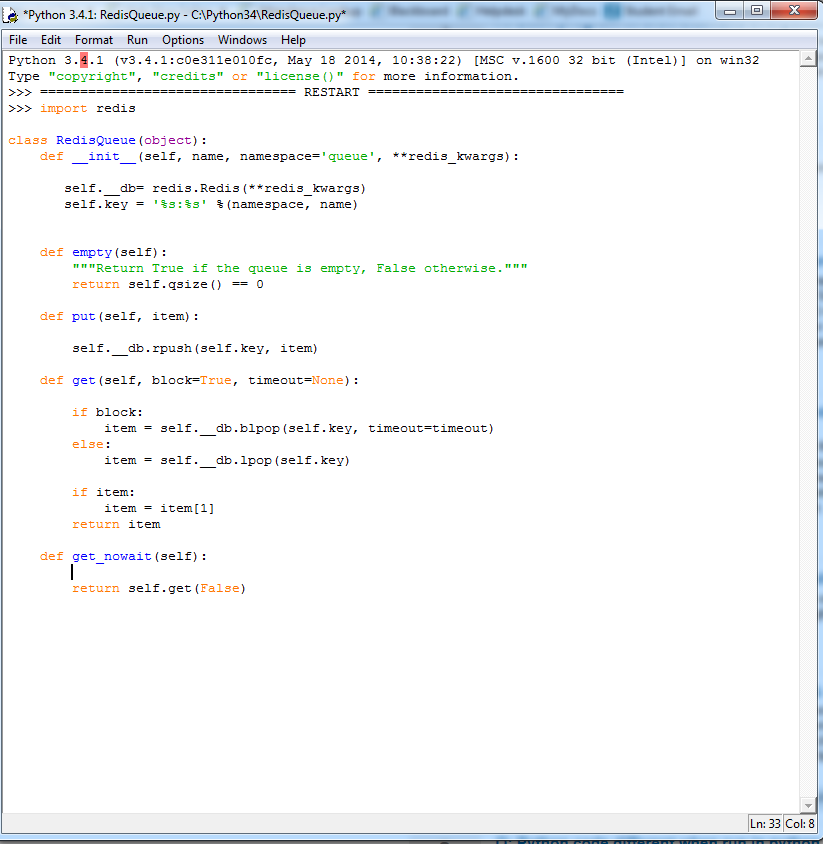
The example above shows 2 clients the first pushing to a queue called ‘examples’ using the LPUSH command and the second the ‘digester’ popping messages from the tail of the list using BRPOP, if we required items from the head of the list LRPOP would be used.

This question has proven quite challenging, following a lot of trial and error it was decided to create the cluster in advance of implementing the other sections.



A master server in black with 3 slave servers all created via the CLI using the command

**redis-server --port 6380 --slaveof 127.0.0.1 6379** and incrementing the port number of each subsequent slave as required – 6380, 6381, 6382



Python script written following a tutorial that should implement a blocking list, failed to run successfully. <http://peter-hoffmann.com/2012/python-simple-queue-redis-queue.html>

The question was left as is at this stage as implementation was proving excessively frustrating and time consuming.

## Conclusion

This portfolio entry was a very mixed experience, firstly Redis is so different from other databases encountered it took a while to become familiar with it, secondly Memcache/Memcached was hugely frustrating and difficult to implement. CRUD operations in Redis are probably easier than with any other database and this part of the portfolio was completed quickly. Memcache then ate up a huge amount of time, installing on Windows was so error prone it was abandoned eventually, thankfully installation on Ubuntu was easier but still far from straightforward. Having eventually completed installation getting data into Memcache proved difficult to the extent that Python was used to interact with it. Running a transaction from a programming language posed no problems. The last question is only half completed as too much time was wasted with Memcache to dedicate a lot of time to another question. Implementing the master – server replicated cluster section of the question was a quick operation and no problems were encountered.