Engineering Redis Challenge

# Objective

[Redis](http://redis.io) is an in-memory NoSQL data store that supports operations or “commands” on data structures such as sets, lists and hashes. Your objective is to implement a service that supports a subset of the Redis command set. That is, you are to build a “mini redis”.

This has two parts -- first, the implementation of Redis commands and the underlying data structure to support them, and second, support for calling this “mini redis” over the network.

As you work on this challenge, try to complete the first part in its entirety before moving on to the second part.

# Part One: Core

## Command Requirements

You are to implement the following set of commands. The command definitions are all available at <http://redis.io/commands>.

1. [SET](http://redis.io/commands/set) key value
2. [SET](http://redis.io/commands/set) key value EX seconds *(need not implement other SET options)*
3. [GET](http://redis.io/commands/get) key
4. [DEL](http://redis.io/commands/del) key
5. [DBSIZE](http://redis.io/commands/dbsize)
6. [INCR](http://redis.io/commands/incr) key
7. [ZADD](http://redis.io/commands/zadd) key score member
8. [ZCARD](http://redis.io/commands/zcard) key
9. [ZRANK](http://redis.io/commands/zrank) key member
10. [ZRANGE](http://redis.io/commands/zrange) key start stop

## Atomicity

One of the key benefits of Redis is that it guarantees atomic, ordered access to data. Your implementation should offer the same guarantee, so that, for example, access from multiple threads is handled safely.

## Deliverable

When this part of the challenge is complete, you should have a working implementation of the specified commands, and there should be one or methods or other entry point that can invoke those commands.

This should include some sort of test harness or set of test cases that allows the developer to demonstrate the functionality of the implemented commands.

# Part Two: Networking

In this part of the challenge, you will add a layer handling network connections, enabling access to the Redis datastore over the network.

## Networking Requirements

When launched, your application should listen on port 5555 and serve inbound connections. You should **not** be concerned with following the Redis wire protocol; instead, we will make some radical simplifications:

1. Keys and values can only be the characters from the set [a-zA-Z0-9-\_].
2. Commands are ASCII strings with space-delimited parameters.
3. Responses are ASCII strings with space-delimited values (where appropriate).

Other networking requirements:

1. Your server should support **multiple** simultaneous connections, just like real Redis
2. It should be possible to connect to the service via telnet. See example below.

## Example

|  |
| --- |
| telnet localhost 5555  SET mykey cool-value  OK  GET mykey  cool-value  DEL mykey  OK  GET mykey  (nil) |

# General Guidance

## Languages and Frameworks

You are welcome to use whatever languages and frameworks you are most comfortable with as you complete this challenge.

## Library Support

In order to allow you to focus on implementing the core Redis features effectively, you can use a performant existing TCP server library to bootstrap your development of the network layer.

## Outside Resources

You are welcome to consult any external, online resources that you find helpful as you work on this challenge, but be prepared to explain your work and ensure you understand your solution thoroughly.

## Submission

When you have completed your implementation, archive it and email it to the Scopely interviewer who requested you complete the challenge.