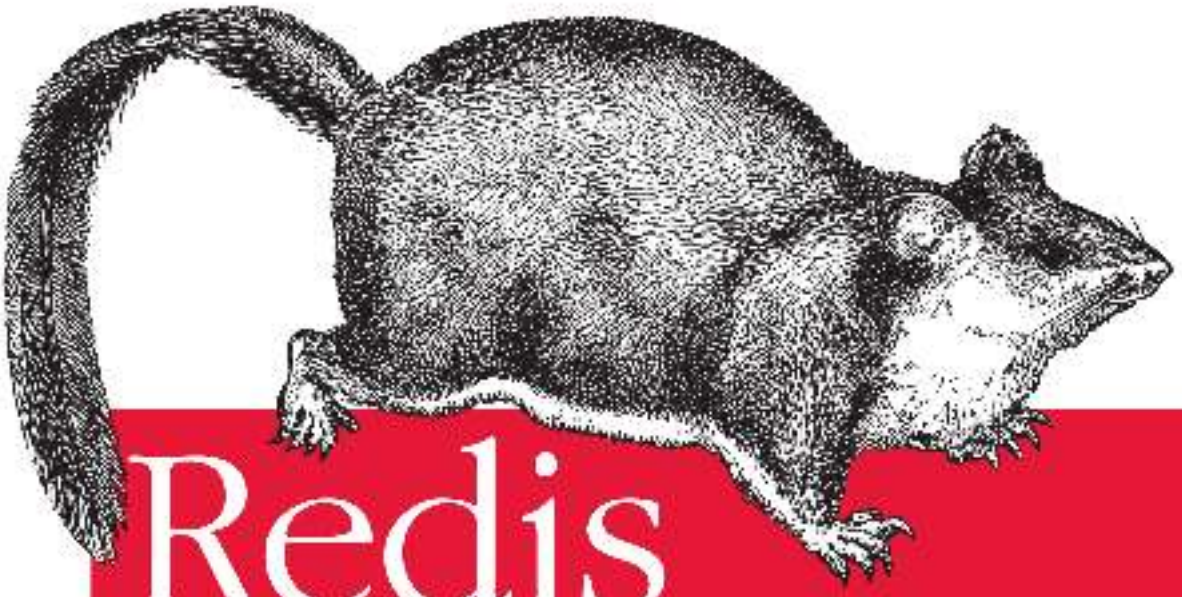
Practical Techniques for Fast Data Manipulation



Redis Cookbook

O'REILLY®

Tiago Macedo & Fred Oliveira

Redis Cookbook

Two years since its initial release, Redis already has an impressive list of adopters, including Engine Yard, GitHub, Craigslist, and Digg. This open source data structure server is built for speed and flexibility, making it ideal for many applications. If you're using Redis, or considering it, this concise cookbook provides recipes for a variety of issues you're likely to face.

Each recipe solves a specific problem, and provides an in-depth discussion of how the solution works. You'll discover that Redis, while simple in nature, offers extensive functionality for manipulating and storing data.

- Learn when it makes sense to use Redis
- Explore several methods for installing Redis
- Connect to Redis in a number of ways, ranging from the command line to popular languages such as Python and Ruby
- Solve a range of needs, from linked datasets to analytics
- Handle backups, sharding, datasets larger than available memory, and many other tasks

Strata
Making Data Work

Strata is the emerging ecosystem of people, tools, and technologies that turn big data into smart decisions. Find information and resources at oreilly.com/data.

Twitter: @oreillymedia
facebook.com/oreilly

US $19.99 CAN $22.99
ISBN: 978-1-449-30504-8

O'REILLY®
oreilly.com

Redis Cookbook

Redis Cookbook

Tiago Macedo and Fred Oliveira

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Redis Cookbook
by Tiago Macedo and Fred Oliveira

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editors: Andy Oram and Mike Hendrickson
Production Editor: Jasmine Perez
Proofreader: O'Reilly Production Services
Cover Designer: Karen Montgomery
Interior Designer: David Futato

Printing History:
August 2011: First Edition.

ISBN: 978-1-449-30504-8

[LSI]

1311195806

Table of Contents

Preface

Introduction

Redis is a data structure server with an in-memory dataset for speed. It is called a data structure server and not simply a key value store because Redis implements data structures allowing keys to contain binary safe strings, hashes, sets and sorted sets, as well as lists. This combination of flexibility and speed makes Redis the ideal tool for many applications.

Redis first started in early 2009 as a key value store developed by Salvatore Sanfilippo in order to improve the performance of his own LLOOGG, an analytics product. Redis grew in popularity after getting support from people and companies in the developer world and has since been supported by VMware, who hired Salvatore and Pieter Noordhuis to work full-time on the project.

Today, Redis is used by companies large and small doing both large and small tasks. Companies like Engine Yard, Github, Craigslist, Disqus, Digg, and Blizzard are part of the growing list of Redis adopters. An extended list of people working with Redis is available on the project's official site at *http://redis.io*.

There are often several ways to solve problems using Redis. This book, while not a tutorial on Redis, key value stores, or data structures, gives you recipes for solving specific problems with Redis that you can then adapt to your own problem set. Many of these recipes have come up because we've used them in our own jobs, solving our own problems.

Each of these recipes solves a specific problem using Redis, including a quick introduction to the problem, the solution, and a longer discussion with insight into how the solution works. Redis is, while simple in nature, quite extensive when it comes to functionality to manipulate and store data. This volume will thus not cover every single command extensively. It will, however, give you the basics on solving specific problems with it, in hopes that our solutions guide you to your own.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Redis Cookbook* by Tiago Macedo and Fred Oliveira (O'Reilly). Copyright 2011 Tiago Macedo and Fred Oliveira, 978-1-449-30504-8."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9781449305048

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgements

We thank Pieter Noordhuis for thoroughly reviewing several chapters of our book, our editor Andy Oram for his work on making us look good, Salvatore Sanfilippo for his words of encouragement, and our respective companies for the extra free time to write this book.

CHAPTER 1

An Introduction to Redis

This chapter discusses some of Redis's basic concepts. We'll look into when Redis is a great fit, how to install the server and command-line client on your machines, and Redis's data types.

When to use Redis

Problem

Nearly every application has to store data, and often lots of fast-changing data. Until recently, most applications stored their data using relational database management systems (RDBMS for short) like Oracle, MySQL, or PostgreSQL. Recently, however, a new paradigm of data storage has emerged from the need to store schema-less data in a more effective way—NoSQL. Choosing whether to use SQL or NoSQL is often an important first step in the design of a successful application.

Solution

There are two important thing to consider when choosing whether to use SQL or NoSQL to store your data: its nature and your usage pattern. Some data is a great fit for a relational storage engine, while other data benefits from the schema-free nature of a NoSQL engine like Redis or its alternatives. If you don't rely on a particular RDBMS feature and need the performance or scalability of a NoSQL database, that might in fact be the ideal choice. So in order to decide whether your data should be stored in a RDBMS or NoSQL engine, you need to look into a few specific things that will help you make a decision. Also bear in mind that quite often the ideal solution will be to use both.

Are your application and data a good fit for NoSQL?

When working on the web, chances are your data and data model keep changing with added functionality and business updates. Evolving the schema to support these changes in a relational database is a painful process, especially if you can't really afford downtime—which people most often can't these days, because applications are expected to run 24/7. As a case in point, in a recent presentation on MongoDB, Jeremy Zawodny of Craigslist mentioned how changing the schema on their database typically takes a two month-long toll on their post archival service.

Examples of data that are a particularly good fit for nonrelation storage are transactional details, historical data, and server logs. These are normally highly dynamic, changing quite often, and their storage tends to grow quite quickly, further compounding the problem of adjusting the schema to store them. They also don't typically feel "relational"—that is, the data in them doesn't tend to fan out in relationships to other types of data. That's a good indication that they can use something other than a RDBMS.

Another way to gauge the fit for NoSQL is to look at whether you find yourself denormalizing your data for performance reasons, and no longer benefit from some of the advantages of a relational system, such as consistency and redundancy checks.

One thing to keep in mind is that NoSQL databases generally don't provide ACID (atomicity, consistency, isolation, durability), or do it only partially. This allows them to make a few tradeoffs that wouldn't be possible otherwise. Redis provides partial ACID compliance by design due to the fact that it is single threaded (which guarantees consistency and isolation), and full compliance if configured with `appendfsync always`, providing durability as well.

Performance can also be a key factor. NoSQL databases are generally faster, particularly for write operations, making them a good fit for applications that are write-heavy.

All this being said, and even though NoSQL feels more flexible, there are also great arguments to be made for storing relational data in a RDBMS. If you have predictable data that is a great fit for normalization, you can reap the benefits of using a relational data storage engine. Always look at the data before making a decision.

Don't believe the hype

NoSQL databases such as Redis are fast, scale easily, and are a great fit for many modern problems. But as with everything else, it is important to always choose the right tool for the job. Play to the strengths of your tools by looking at what you're storing, how often you'll access it, and how data (and its schema) might change over time.

Once you've weighted all the options, picking between SQL (for stable, predictable, relational data) and NoSQL (for temporary, highly dynamic data) should be an easy task. Doing this kind of thinking in advance will save you many headaches in future data migration efforts.

There are also big differences between NoSQL databases that you should account for. For example, MongoDB (a popular NoSQL database) is a feature-heavy document database that allows you to perform range queries, regular expression searches, indexing, and MapReduce. You should weigh all the factors when choosing your database. As we said earlier, the questions boil down to what your data looks like and what your usage pattern is.

For example, Redis is extremely fast, making it perfectly suited for applications that are write-heavy, data that changes often, and data that naturally fits one of Redis's data structures (for instance, analytics data). A scenario where you probably *shouldn't* use Redis is if you have a very large dataset of which only a small part is "hot" (accessed often) or a case where your dataset doesn't fit in memory.

Installing Redis

Problem

You want to install Redis on your computer.

Solution

There are several ways to install Redis on your computer or server, but the best and most flexible option is to compile it yourself. Nevertheless, depending on your distribution or operating system, there are other options.

Discussion

Compiling From Source

Redis evolves very quickly and package maintainers have a hard time keeping up with the latest developments. Since Redis doesn't have any external dependencies, compilation and installation are very straightforward, so we recommend you do it yourself. Redis should build cleanly in most Linux distributions, Mac OS X, Solaris, and Cygwin on Windows.

1. Downloading the source

   You can download Redis from the official site or directly from the Github project, either using Git or your browser to fetch a snapshot of one the branches or tags. This allows you get to get development versions, release candidates, etc.

2. Compiling

   Redis compilation is straightforward. The only required tools should be a C compiler (normally GCC) and Make. If you want to run the test suite, you also need Tcl 8.5.

After unpacking the source and changing your terminal path to the source directory, just type:

```
make
```

This will compile Redis, which on a modern computer should take less than 10 seconds. If you're using a x86_64 system but would like an x86 build (which uses less memory but also has a much lower memory limit), you can do so by passing along `32bit` to Make:

```
make 32bit
```

After compiling Redis, particularly if you're building a development version, you should run the test suite to ensure that the server is behaving as expected.

```
make test
```

3. Installing

   After compiling Redis, you can go ahead and run it:

   ```
   cd src && ./redis-server
   ```

   However, you might find it more convenient to install it to another location in your system. The Makefile wlll also help you do that:

   ```
   make install
   ```

   This will install Redis binaries to */usr/local/bin*. If you wish to install to another location, you can pass it to *make*. For instance:

   ```
   make install /opt/local
   ```

   This will install the binaries in */opt/local/bin*.

   After installating the Redis server, you should also copy the configuration file (*redis.conf*) to a path of your choice, the default being */etc/redis.conf*. If your configuration file is in a different path from the default, you can pass it along as a parameter to redis-server:

   ```
   /usr/local/bin/redis-server /alternate-location-for-redis-config.conf
   ```

Installing on Linux

Most modern Linux distributions have Redis packages available for installation, but keep in mind that these are normally not up-to-date. However, if you prefer to use these, the installation procedure is much simpler:

Debian/Ubuntu

```
sudo apt-get install redis-server
```

Fedora/Redhat/CentOS

```
sudo yum install redis
```

Gentoo

```
sudo emerge redis
```

This approach has a few advantages: by using your package management system, you can more easily keep software up-to-date, and you'll most likely get at least security and stability updates. Besides that, you'll also get startup scripts and an environment more suited to your distribution (user accounts, log files, database location, etc).

Installing on Windows

Although Redis is not officially supported on Windows for several reasons—notably the lack of a copy-on-write `fork()`—the community provides a few ports. Due to Windows' limitations, Redis MinGW builds execute operations such as `BGSAVE` and `BGREWRITEAOF` in the foreground (thus blocking the Redis process) and Cygwin builds don't use CoW, which makes background operations very slow, particularly for large database sizes. Be sure to disable automatic saving in the config file (especially for MinGW builds) and use `SAVE` only when needed.

Beware that for performance and stability reasons, the Windows versions of Redis are not recommended for production use. Consider using a native or virtualized Linux/UNIX environment instead. Despite that, you might find these versions useful for development or testing.

Cygwin

Cygwin is a UNIX-like environment for Windows that implements most of the POSIX API, thereby enabling you to build and run Redis on Windows. Cygwin is the easiest way to compile Redis on Windows. From the Cygwin environment, you can download the Redis source and build it following the steps described in "Compiling From Source" on page 3. Some users might have licensing issues (due to Cygwin's use of the GPL).

ServiceStack maintains a page with a few builds (both x86 and x86_64) along with their open source C# client.

MinGW

MinGW is a Windows port of the GNU CC compiler (GCC) and the GNU binutils. Because it builds native Windows binaries, it doesn't suffer from the same licensing issues as Cygwin.

Since MinGW doesn't provide a POSIX API, the Redis source code needs to be modified in order to build. Dušan Majkić's fork on Github is regularly updated and also adds a Windows Service for easier management.

Installing on Mac OS X

There are several ways to install Redis on Mac OS X. They all require you to have the XCode developer tools installed, which includes libraries and compilers. If you are a developer on a Mac, chances are you already have this package installed. If you don't, you can either download it from Apple's developers website or run "Install Developer Tools" on your Mac's installation DVDs.

You can manually compile Redis from source by following the steps earlier in this chapter. Most people, however, prefer the convenience of a package manager such as Fink, MacPorts, or Homebrew. A Redis package isn't available on Fink, so we'll cover the other two.

Installing through MacPorts. MacPorts defines itself as "an easy to use system for compiling, installing, and managing open source software." It is based on the FreeBSD Ports system, and to a large extent can be used in the exact same way.

In order to install Redis through MacPorts, you need to first install the package management system. There's an extensive guide on how to do that at guide.macports.org. Once you've installed MacPorts, installing the Redis package is as simple as:

```
port install redis
```

Since Redis has no direct dependencies, the actual compilation and installation process is quite speedy. You will then be ready to start using Redis.

Installing through Homebrew. Homebrew is the latest entrant in the Mac package management scene. Being relatively new means that not every package you might be looking for is available on it—even though they make contributions very easy—but if you're looking for a tool that developers use often, chances are that it's going to be available through a Homebrew recipe.

You can install Homebrew by following the detailed instructions available over at Github, but it is usually as simple as running the following command:

```
ruby -e "$(curl -fsSLk https://gist.github.com/raw/323731/install_homebrew.rb)"
```

Once that's done, you'll be ready to install packages using the Homebrew recipes system. Installing Redis is just a matter of typing:

```
brew install redis
```

You can then run `redis-server` manually or install it into the Mac's own LaunchServices so that it starts when you reboot your computer. You can edit the configuration file */usr/local/etc/redis.conf* to tweak it to your liking, and then start the server:

```
redis-server /usr/local/etc/redis.conf
```

Using Redis Data Types

Problem

You need to understand Redis data types in order to make better use of them for specific applications.

Solution

Unlike most other NoSQL solutions and key-value storage engines, Redis includes several built-in data types, allowing developers to structure their data in meaningful semantic ways. Predefined data types add the benefit of being able to perform data-type specific operations inside Redis, which is typically faster than processing the data externally. In this section, we will look at the data types Redis supports, and some of the thinking behind them.

Discussion

Unlike most other NoSQL solutions and key-value storage engines, Redis includes several built-in data types, allowing developers to structure their data in meaningful semantic ways—with the added benefit of being able to perform data-type specific operations inside Redis, which is typically faster than processing the data externally. In this section, we will look at the data types Redis supports, and some of the thinking behind them.

Before we dive into the specific data types, it is important to look at a few things you should keep in mind when designing the key structure that holds your data:

- Be consistent when defining your key space. Because a key can contain any characters, you can use separators to define a namespace with a semantic value for your business. An example might be using `cache:project:319:tasks`, where the colon acts as a namespace separator.
- When defining your keys, try to limit them to a reasonable size. Retrieving a key from storage requires comparison operations, so keeping keys as small as possible is a good idea. Additionally, smaller keys are more effective in terms of memory usage.
- Even though keys shouldn't be exceptionally large, there are no big performance improvements for extremely small keys. This means you should design your keys in such a way that combines readability (to help you) and regular key sizes (to help Redis).

With this in mind, keys like `c:p:319:t` or `user 123` would be bad—the first because it is semantically crude, and the latter because it includes whitespace. On the other hand, keys like `cache:project:319:tasks`, `lastchatmessage`, or

`464A1E96B2D217EBE87449FA8B70E6C7D112560C` are good, because they're semantically meaningful. Note that the last example of an SHA1 hash is, while hard to guess and predict, semantically meaningful and quite useful if you are storing data related to an object for which you can consistently calculate a hash.

Strings

The simplest data type in Redis is a string. Strings are also the typical (and frequently the sole) data type in other key-value storage engines. You can store strings of any kind, including binary data. You might, for example, want to cache image data for avatars in a social network. The only thing you need to keep in mind is that a specific value inside Redis shouldn't go beyond 512MB of data.

Lists

Lists in Redis are ordered lists of binary safe strings, implemented on the idea of a linked list. This means that while getting an element by a specific index is a slow operation, adding to the head or tail of the data structure is extremely fast, as it should be in a database. You might want to use lists in order to implement structures such as queues, a recipe for which we'll look into later in the book.

Hashes

Much like traditional hashtables, hashes in Redis store several fields and their values inside a specific key. Hashes are a perfect option to map complex objects inside Redis, by using fields for object attributes (example fields for a car object might be "color", "brand", "license plate").

Sets and Sorted Sets

Sets in Redis are an unordered collection of binary-safe strings. Elements in a given set can have no duplicates. For instance, if you try to add an element `wheel` to a set twice, Redis will ignore the second operation. Sets allow you to perform typical set operations such as intersections and unions.

While these might look similar to lists, their implementation is quite different and they are suited to different needs due to the different operations they make available. Memory usage should be higher than when using lists.

Sorted sets are a particular case of the set implementation that are defined by a score in addition to the typical binary-safe string. This score allows you to retrieve an ordered list of elements by using the `ZRANGE` command. We'll look at some example applications for both sets and sorted sets later in this book.

CHAPTER 2
Clients

In this chapter, we'll look into some of the ways you can connect to Redis. We'll begin with the most basic option: Redis's command-line client, the `redis-cli` command. Then we'll look at ways to integrate Redis with common programming languages such as Ruby and Python.

Using Redis from the Command Line

Problem

Often you might find yourself in need of firing a simple Redis query, either to set or change a variable, flush a database, or perhaps take a look at your data. With Redis you can achieve this directly from the command line.

Solution

Redis ships with a command line client: *redis-cli*. Redis-cli is a fully featured interactive client, supporting line editing, history, and tab completion. By using `help` followed by a Redis command, you can also get help on how each command works.

You can use *redis-cli* to connect to a local or remote host Redis server and call commands by passing them as arguments (or piping them in) or by using its interactive mode.

Discussion

You can get a list of the command line options by typing:

```
redis-cli -h
```

The most typical usage scenarios would be something like the following, to connect to a remote server in interactive mode:

```
redis-cli -h serverip
```

The following connects to a local server running on a nondefault port in interactive mode:

```
redis-cli -p 6380
```

The following connects to a local server on the default port (6379), executes the *INFO* command, and returns you to your original shell:

```
redis-cli INFO
```

You can also use pipes and output redirection for a more powerful interaction:

```
cat command_list.txt | redis-cli > command_output.txt
```

Using Redis from Python with redis-py

Problem

You want to access and manipulate data in your Redis server with Python.

Solution

Install and use Andy McCurdy's redis-py using *pip*, *easy_install*, or from the source code.

Discussion

Python's package index tool (*pip*) and *easy_install* make it trivial to install and start using redis-py. A couple of commands will get you going. Let's start by looking at how you install redis-py using *pip*:

```
pip install redis-py
```

Alternatively, if you're using *easy_install*, the installation command would be:

```
easy_install redis
```

From this point on, connecting to Redis in Python is as simple as issuing `import redis`, connecting to the server, and executing regular Redis commands. Here's an example:

```
>>> import redis
>>> redis = redis.Redis(host='localhost', port=6379, db=0)
>>> redis.smembers('circle:jdoe:soccer')
set(['users:toby', 'users:adam', 'users:apollo', 'users:mike'])
>>> redis.sadd('circle:jdoe:soccer', 'users:fred')
True
>>> redis.smembers('circle:jdoe:soccer')
set(['users:toby', 'users:adam', 'users:apollo', 'users:mike', 'users:fred'])
```