10 quick tips for Redis

Redis is hot in the tech community right now. It's come a long way from being a small personal project from Antirez, to being an industry standard for in memory data storage. With that comes a set of best practices that most people can agree upon for using Redis properly. Below we'll explore 10 quick tips on using Redis correctly.

1. STOP USING KEYS *

Okay, so maybe shouting at you isn't a great way to start this article. But it's quite possibly the most important point. Too often do I look at an redis instance, pull up a quick commandstats, and see a glaring `KEYS` staring right back at me. In all fairness, coming from a programmatical standpoint it would make sense to have a psuedocode that does something like:

```
for key in 'keys *':
  doAllTheThings()
```

But when you have, say, 13 million keys, things are going to slowdown. Since `KEYS` is O(n) where n is the number of keys returned, your complexity is bound by the dbsize. Also, during this whole operation, nothing else can be run against your instance.

As a substitute, check out `SCAN` which allows you to well... scan through your database in increments instead. This operates on an interator so you can stop and go as you see fit.

2. Find Out What's Slowing Down Redis

Since Redis doesn't have the most verbose of logs, it's often hard to trackdown what exactly is going on inside your instance. Luckily Redis provides you with the commandstat utility to show you this:

```
127.0.0.1:6379> INFO commandstats
# Commandstats
cmdstat_get:calls=78,usec=608,usec_per_call=7.79
cmdstat_setex:calls=5,usec=71,usec_per_call=14.20
cmdstat_keys:calls=2,usec=42,usec_per_call=21.00
cmdstat_info:calls=10,usec=1931,usec_per_call=193.10
```

This gives you a breakdown of all the commands, how many times they've been run, the number of microseconds it took to execute (total and avg per call)

To reset this simply run `CONFIG RESETSTAT`, and you've got a brand new slate.

3. Use Redis-Benchmark as a Baseline, Not the Gospel Truth

Salvatore, the creator of Redis put it geniously: "To test Redis doing GET/SET is like testing a Ferrari checking how good it is at cleaning the mirror when it rains." A lot of times people come to me wondering why their Redis-Benchmark results are less than optimal. But we have to take into account many different factors, such as:

- What client-side limitations could we have run into?
- Was there a difference in versioning?
- Are the tests being performed relevant to what the application will be performing?

Redis-Benchmark provides an awesome baseline to make sure your redis-server isn't behaving abnormally, but it should never be taken as a true "load test". Load tests need to be reflective of how your application behaves, and from an environment as close to production as possible.

4. Hashes Are Your Best Friend

Invite hashes over for dinner. Wine and dine hashes. You'll be amazed at what happiness they can bring if you just give them the chance. I've seen one too many key structures like this before:

```
foo:first_name
foo:last_name
foo:address
```

In the above example, foo would be maybe a username for a user, and each one of those is a separate key. This adds room for errors, and adds unnecessary keys to the fold. Instead, consider a hash. Suddenly you've only got one key:

```
127.0.0.1:6379> HSET foo first_name "Joe"
(integer) 1
127.0.0.1:6379> HSET foo last_name "Engel"
(integer) 1
127.0.0.1:6379> HSET foo address "1 Fanatical Pl"
(integer) 1
127.0.0.1:6379> HGETALL foo
1) "first_name"
2) "Joe"
3) "last_name"
4) "Engel"
5) "address"
```

```
6) "1 Fanatical Pl"
127.0.0.1:6379> HGET foo first_name
"Joe"
```

5. Set That TTL!

Whenever possible, take advantage of expiring keys. A perfect example is storing something like temporary authentication keys. When you retrieve the auth key—let's use OAUTH as an example—you often are given an expiration time. When you set the key, set it with the same expiration, and Redis will clean up for you! No more need for `KEYS *` to iterate through all those keys, eh?

6. Choosing the Proper Eviction Policy

While we're on the topic of cleaning up keys, let's touch on eviction. When your Redis instance fills up, Redis will attempt to evict keys. Depending on your use case, I highly recommend `volatile-lru`—assuming you have expiring keys. If you're running something like a cache and don't have an expiry set, you could consider `allkeys-lru`. I'd recommend checking out the available options here.

7. If Your Data is Important, Try/Except

If it's absolutely critical for data to make it to your Redis instance, I heavily recommend putting in a try/except. Since almost all Redis clients are configured to "fire-and-forget," there should always be consideration for when a key doesn't *actually* make it to the database. The complexity added to your redis call is next to nothing in this case, and you can ensure your important data makes it to where it should be.

8. Don't Flood One Instance

Whenever possible, split up the workload amongst multiple Redis instances. As of version 3.0.0, Redis Cluster is now available. Redis Cluster allows you to break apart keys amongst sets of given masters/slaves based on key ranges. A full breakdown of the magic behind Cluster can be found here. And if you're looking for a tutorial, then look no further. If clustering is not an option, consider namespacing and distributing your keys among multiple instances. An amazing write-up on partitioning your data can be found on the redis.io website here.

9. More Cores = More Better, Right?!

Wrong. Redis is a single threaded process and will, at most, consume two cores if you have persistence enabled. Unless you plan on running multiple instances on the same host—hopefully only for dev testing in that case!—you shouldn't need more than two cores for a Redis instance.

10. HA All the Things!

Redis Sentinel is now very well tested, and many users have it running in production (ObjectRocket included!). If you're relying heavily on Redis for your application, then you need to consider an HA (high availability) solution to keep you online. Of course, if you don't want to manage all of that yourself, ObjectRocket offers our HA platform with 24x7 support for your consumption, give it a shot.