Simeon Yordanov Grancharov

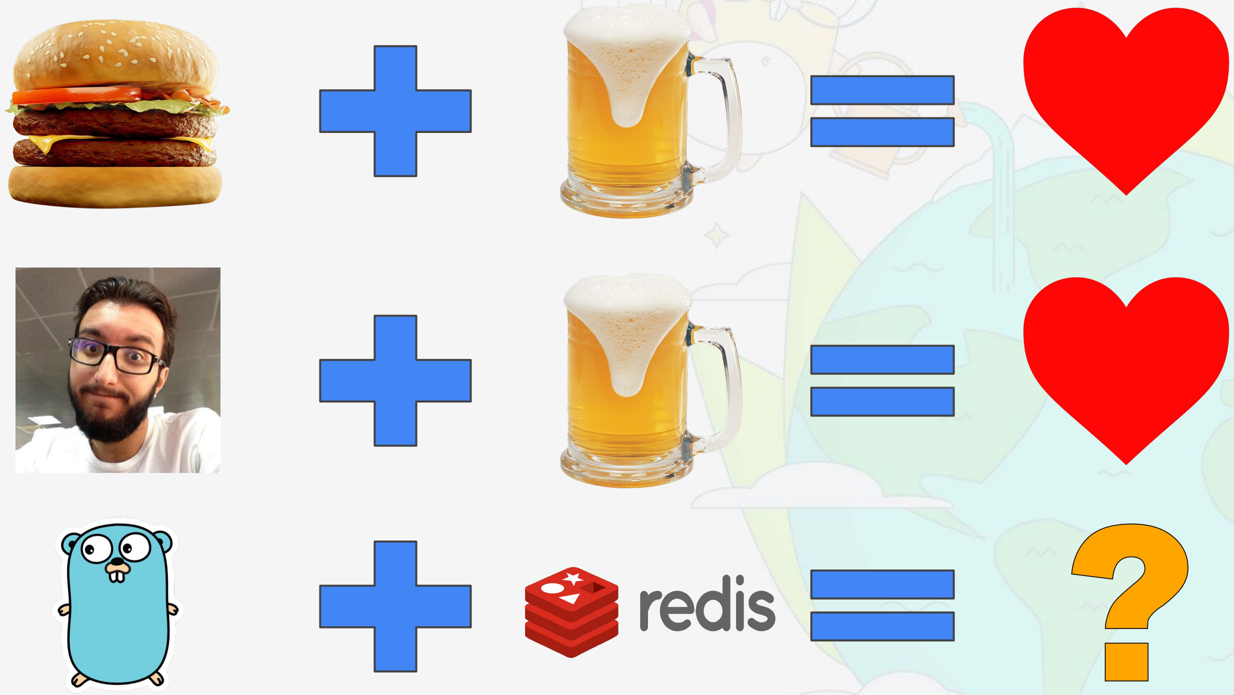
Senior Software Engineer

Go & Redis: More than a love story

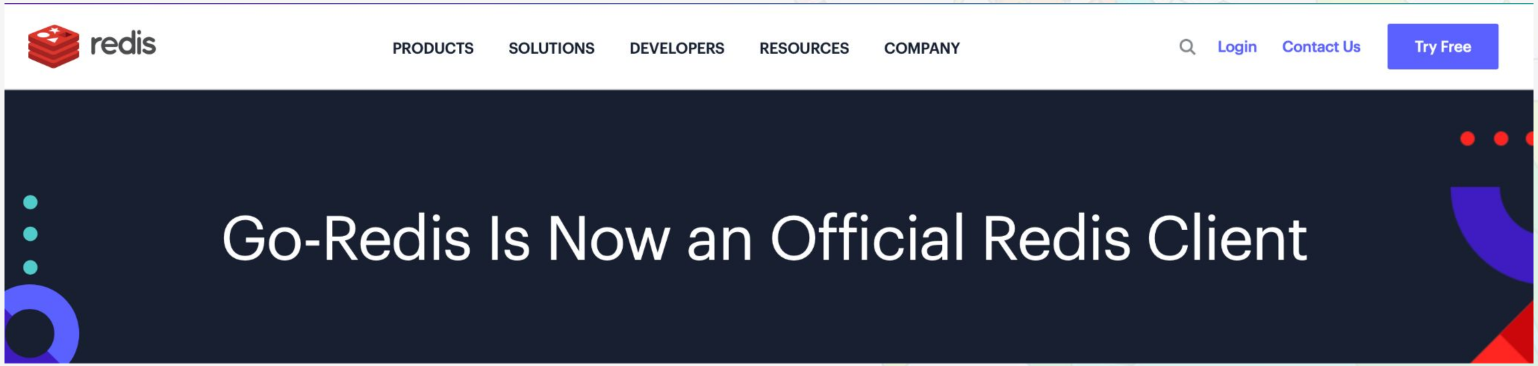
A little bit about me

- I'm a backend engineer currently working at EF
- I have been working with Golang for the past 3 years
- I have been coding for around 10 years using C#, Python and Java.
- Non professional runner, football lover and gym rat.
- I also enjoy reading fantasy, mainly Brandon Sanderson and George RR Martin
- When I'm not coding, I blog about coding

Some things just fit together....



What “spiked” my curiosity?



Go-Redis Is Now an Official Redis Client

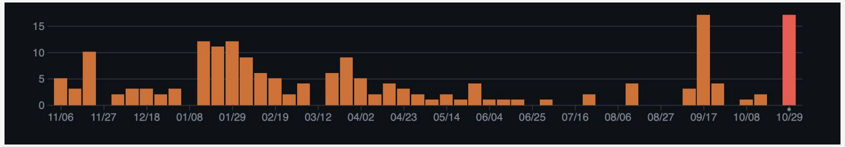
- What is Redis?
  - Open source
  - NonSql
  - Key - Value storage
- Why use Redis?
  - It’s faaaaaaast
  - Data is organized in simple DS
  - Simple set of features
- What’s the catch?
  - no secondary storage = less capacity
  - It makes you think a little more

Go-Redis

https://github.com/redis/go-redis

https://redis.uptrace.dev/

- Officially promoted by Redis itself
- +18k stars on Github



- Automatic connection pooling
- Redis cluster and sentinel clients
- Type safe
- Allows custom commands

How to create a GoRedis client?

```go
import (
	"context"
	"time"

	goRedis "github.com/redis/go-redis/v9"
)

type RedisRepository struct {
	client goRedis.Client
}

func NewRedisRepository(address string) RedisRepository {
	return RedisRepository{
		client: *goRedis.NewClient(&goRedis.Options{
		Addr: address,
		}),
	}
}
```

How to call a Redis command

- GoRedis contains specific functions (type safe API).
- Context is required.
- We need to read the result

```
func (repo *RedisRepository) HashGetAll(key string) (map[string]string, error) {
        ctx := context.Background()
        val, err := repo.client.HGetAll(ctx, key).Result()
        return val, err
}
```

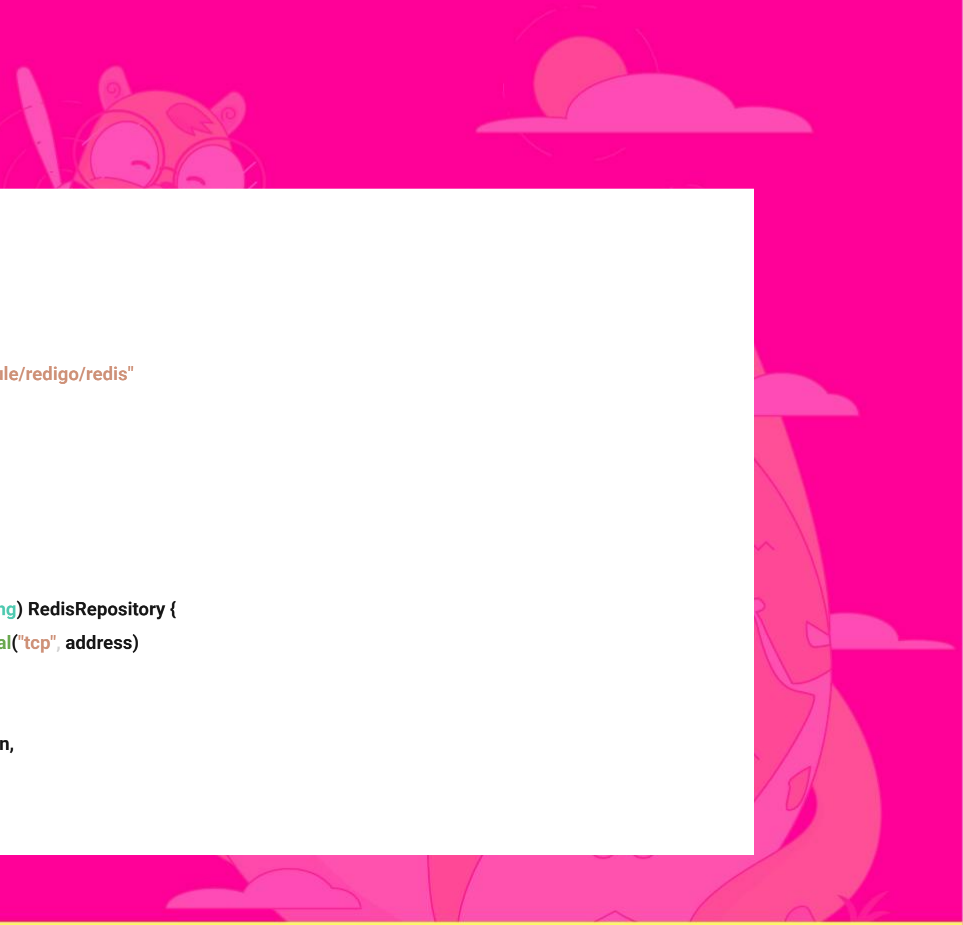
RediGo

https://github.com/gomodule/redigo

- It came first
- Print like API
- Allows custom commands
- Manual connection pooling
- Redis sentinel and cluster

How to create a Redigo client?

```go
package redisgo

import (
	redigo "github.com/gomodule/redigo/redis"
)

type RedisRepository struct {
	conn redigo.Conn
}

func NewRedisRepository(address string) RedisRepository {
	connection, err := redigo.Dial("tcp", address)
	if err != nil { panic(err) }
	return RedisRepository{
		conn: connection,
	}
}
```



How to call a Redis command

- Redigo uses one specific function (print like API).
- No context is required.
- We need to read the result

```
func (repo *RedisRepository) HashGetAll(key string) (map[string]string, error) {
		val, err := redigo.StringMap(repo.conn.Do("HGETALL", key))
		return val, err
}
```

Package Comparing

- We will use the Go Benchmarks from the common library
- Functions that will be compared
  - SET, GET and combined
  - HGETALL, HSET and combined
  - LRANGE, LPUSH and combined
- Both operation execution time and memory storage will be evaluated

How do the benchmarks look like?

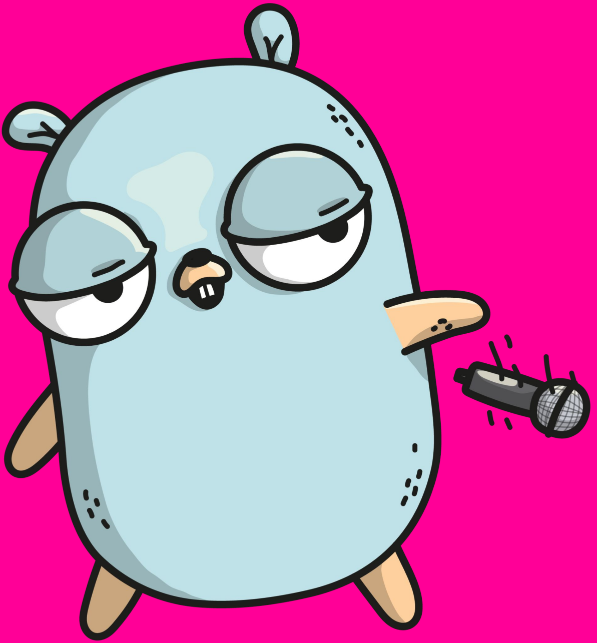
```
import (
        "testing"
)

var redisRepo = NewRedisRepository("0.0.0.0:20003")

func BenchmarkGoRedisGet(b *testing.B) {
        for i := 0; i < b.N; i++ {
        _, err := redisRepo.Get(testKey)
        if err != nil {
                panic(err)
                }
        }
}
```

- Enough talking, let’s run the benchmarks and see the results….

In conclusion

- Prod may differ
- Not all functionalities were tested
- RediGo offers a slightly better performance
- I would use Go redis

Thank you all!



Simeon Yordanov Grancharov