Rewriting the Parse API in Go

GopherCon 2015, Denver
8 July 2015

Abhishek Kona
Software Engineer at Parse and Facebook

What is this talk about?

- Why we rewrote the Parse API in Go
- Tools and libraries we built

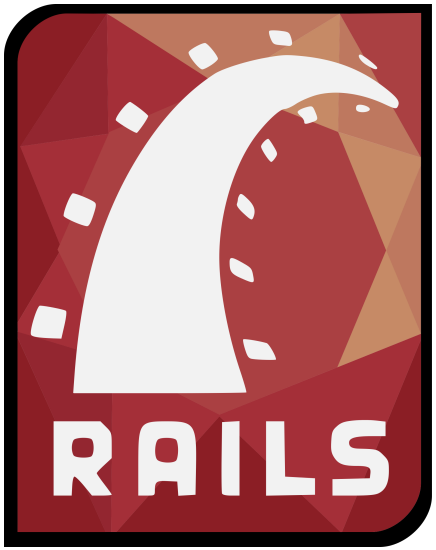
What is Parse?

- Backend as a service
- SDKs for iOS, Android, JS, React, Windows, PHP
- Acquired by Facebook in 2013

Rome was not built in a day - they did not use Parse

Parse circa 2013

- ~60K apps
- 10 engineers
- Ruby on Rails app



Scalability Issues in 2013

- Single popular app could take down Parse
- Fixed-size unicorn pool
- Lengthy deploy times
- Spooky action at a distance

Solution: Rewrite in Go!

Why Rewrite?

- Huge estimated reliability win
- Wanted easier deploys
- Needed faster tests
- Hard to evolve to existing Ruby codebase

Why Go?

- Statically typed
- Good concurrency support
- Dynamic number of worker goroutines per HTTP server
- Easy to attract engineers

Rules of the Rewrite

- Don't break backwards compatibility
- No downtime

Initial Ports

- Hosting server
- Parse Push Notification Service (PPNS)
- Maintains long-lived push sockets with Android clients
- Concurrent conns per node increased from 250K to 1.5M

Mongo Proxy

github.com/facebookgo/dvara (https://github.com/facebookgo/dvara)

- Mongo used to limit max number of connections to 20000
- We wrote our own proxy for Mongo in Go
- Made easy by Go runtime's use of non-blocking I/O

Rollout

- Migrate endpoints one by one
- Diffed responses between old and new code using a shadow cluster
- Started with low-traffic read only endpoints
- Graduated to write endpoints

Comment Goldmine

```
// Note: an unset cache version is treated by ruby as “”.
// Because of this, dirtying this isn’t as simple as deleting it – we need to
// actually set a new value.

// This byte sequence is what ruby expects.
// yes that’s a paren after the second 180, per ruby.

// Inserting and having an op is kinda weird: We already know
// state zero. But ruby supports it, so go does too.

// single geo query, don’t do anything. stupid and does not make sense
// but ruby does it. Changing this will break a lot of client tests.
// just be nice and fix it here.

// Ruby sets various defaults directly in the structure and expects them to appear in cache.
// For consistency, we’ll do the same thing.
```

A Young Language

- Some good libraries: mgo, memcache, etc.
- Some missing libraries

Libraries / Tools

Dependency Injection

- Helps instantiate implementations for test and production
- Easy to miss passing a dependency to a struct

Introducing Inject

github.com/facebookgo/inject (http://github.com/facebookgo/inject)

- Only occurs at process startup for singletons
- Dependencies declared using struct tags
- Fail instead of guessing

Dependency Injection Example

```
type Handler struct {
    Scribe *scribe.Client `inject:""`
    Log    logger.Logger  `inject:""`
}

// ServeHTTP a sample implementation
func (h *Handler) ServeHTTP(w ResponseWriter, r *Request) {
    params := extractParams(r)
    h.Scribe.Log(params)
    h.Log("everything ok")
    w.Write(res)
}
```

Main for Inject

```
func main() {

    var g inject.Graph
    err := g.Provide(
        &inject.Object{Value: scribe.NewHTTPScribeClient()},
        &inject.Object{Value: parse.NewLogger()},
    )
    if err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }

    if err := g.Populate(); err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }
    // rest of main
}
```

Initializing and Destroying Injected Objects

github.com/facebookgo/startstop (http://github.com/facebookgo/startstop)

- Traverses object graph
- At startup: calls `Start` on each injected object in dependency order
- At shutdown: calls `Stop` on each injected object in reverse dependency order
- Fails on cycles

Start-Stop Example

```
type ScribeClient {
    Thrift    *ThriftPool `inject:""`
}

// ScribeClient start will be called after ThriftPool.Start
func (s *ScribeClient) Start() error {
    fmt.Println("starting scribe client")
    return nil
}

type ThriftPool struct {
    //
}

// ThriftPool start will be called first.
func (t *ThriftPool) Start() error {
    fmt.Println("starting thrift pool")
    return t.tcpDial()
}

func (t *ThriftPool) Stop() error {
    fmt.Println("stopping thrift pool")
    return t.tcpCloseAll()
}
```

Graceful Restarts

github.com/facebookgo/grace (https://github.com/facebookgo/grace)

- Restart servers gracefully on deploys
- On USR2, spawns new process and hands off listening socket

Error Reporting

github.com/facebookgo/stackerr (https://github.com/facebookgo/stackerr)

- Wrap error calls with stackerr
- Aggregate errors based on stack trace in an in-house system called Logview

Stackerr Example

```
func main() {
    err := err2()
    fmt.Println(err)
}

func err2() error {
    err := err1()
    if err != nil {
        return stackerr.Wrap(err)
    }
    return nil
}

func err1() error {
    return stackerr.Wrap(errors.New("failure"))
}
```

Stackerr Output

```
failure
/private/tmp/stackerr.go:25                                  err1
/private/tmp/stackerr.go:17                                  err2
/private/tmp/stackerr.go:12                                  main
/usr/local/Cellar/go/1.4.2/libexec/src/runtime/proc.go:72    main
/usr/local/Cellar/go/1.4.2/libexec/src/runtime/asm_amd64.s:2233 goexit
(Stack 2)
/private/tmp/stackerr.go:19                                  err2
/private/tmp/stackerr.go:12                                  main
/usr/local/Cellar/go/1.4.2/libexec/src/runtime/proc.go:72    main
/usr/local/Cellar/go/1.4.2/libexec/src/runtime/asm_amd64.s:2233 goexit
```

Muster

github.com/facebookgo/muster (https://github.com/facebookgo/muster)

- A library to perform operations in a batch
- Two tunables: `MaxBatchSize` and `BatchTimeout`

Generics

github.com/facebookgo/generics (https://github.com/facebookgo/generics)

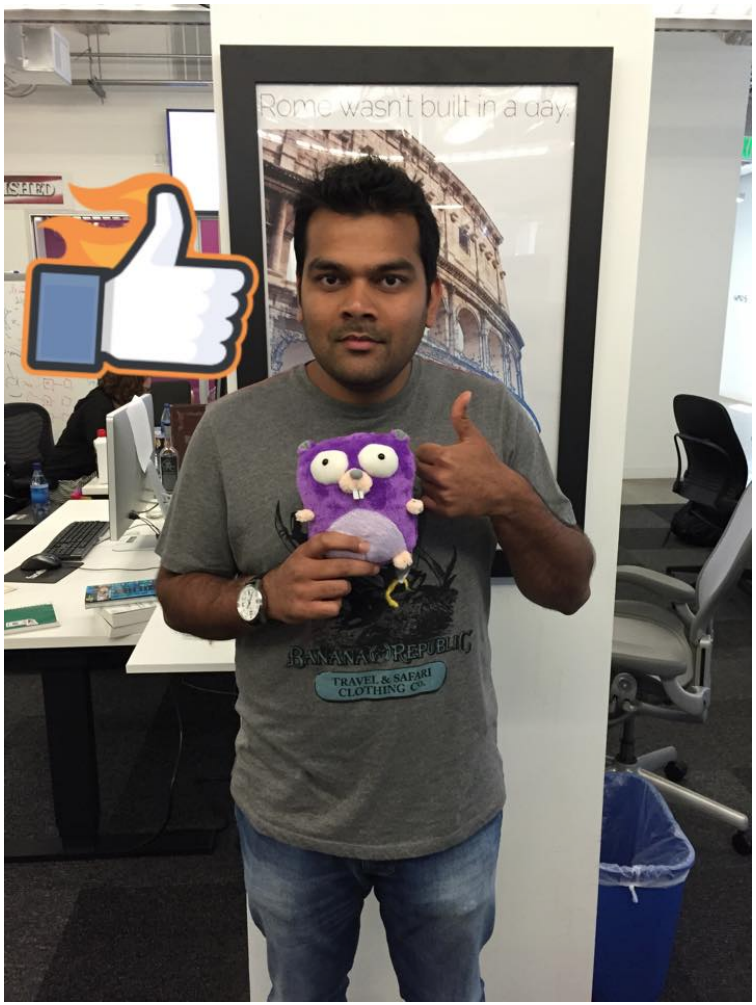
More Libraries

github.com/facebookgo (https://github.com/facebookgo)

- Many more small libraries
- httpcontrol, ensure, stack

We Love Go

github.com/daaku (github.com/daaku)



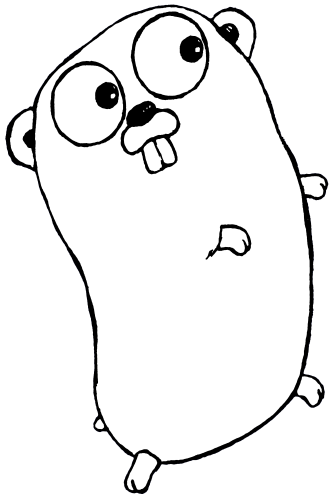
Results

Results

- ~175k LOC in Go vs ~130k LOC in Ruby
- ~3 minutes to run all the unit tests (down from 25min)
- Apps start in seconds instead of minutes
- Downsized API server pool by 90%
- Rolling restarts dropped from 30 minutes to 3 minutes

Parse circa 2015

- >500K apps built on Parse
- 2-3x YoY traffic growth
- Primarily a Go stack



Observations

- Rewrites are hard
- ~4 engineers over 2 years

Go Side Effects

- Deploying with static binaries is easy
- Developers are responsible for deploys, not ops

Recap

http://tiny.cc/parsego

Thank you

Abhishek Kona
Software Engineer at Parse and Facebook
abhishekk@fb.com (mailto:abhishekk@fb.com)
http://sheki.in (http://sheki.in)
@sheki (http://twitter.com/sheki)