Go打造亿级实时分布式平台

高超

- 东南亚最大的出行平台
- 成立于2011年
- 7个国家
- 39个城市
- 710000位司机
- 36000000次App下载

Grab

新加坡

北京

西雅图

越南

印尼

Grab从前的技术栈

Grab现在的技术栈


github
MySQL
redis
amazon web services
PHABRICATOR
etcd
Apache Kafka
Spark
amazon web services
KINESIS
DATADOG
SCALYR
LIGHTSTEP
presto

Why Go at Grab?

- 简洁的语言规范
  - 上手轻松
  - 提升生产效率
- 完整的工具链
  - go test, go build, go vet...
- 方便的部署流程
  - 直接部署打包好的二进制文件
- 优秀的性能
  - 弹性云机器数量骤减90%
  - 响应延迟骤降80%

Grab在用Go做什么

- 全部的后台服务都是用Go打造
  - 50+ 微服务
  - 300+ Gophers（今年目标扩张至800）
- Go打造的流式数据系统支撑每天亿万级事件处理
- API Gateway
- RPC & RESTful framework
- ORM
- CI system
- Machine Learning Platform
- Serverless platform
- ...

Grab的Go实践

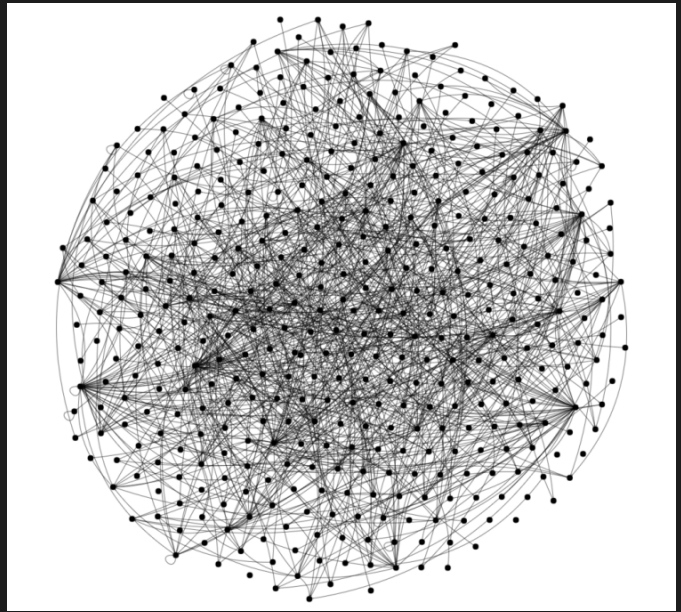
- Monolithic Code Repository
- Distributed Tracing
- Testing
- Code Quality Control
- Bugs

Monolithic Code Repository

- 所有的Go代码都被放置在同一个Repository里面
- 一致的版本，one source of truth
- 极致的代码分享，复用
- 简单的依赖管理
- 原子化的代码更改
- 更好的支撑大规模的重构，代码库的更新
- 团队之间更好的合作
- 灵活的团队界限以及代码归属权
- 最大化的代码透明度，以及自然而然的按团队划分的命名空间

Distributed Tracing

- 架构演变
- 单体应用 -> 大规模的微服务架构
  - 函数变成了服务
  - 越来越多的vendor libs
- 如何快速发现及诊断服务存在的问题？

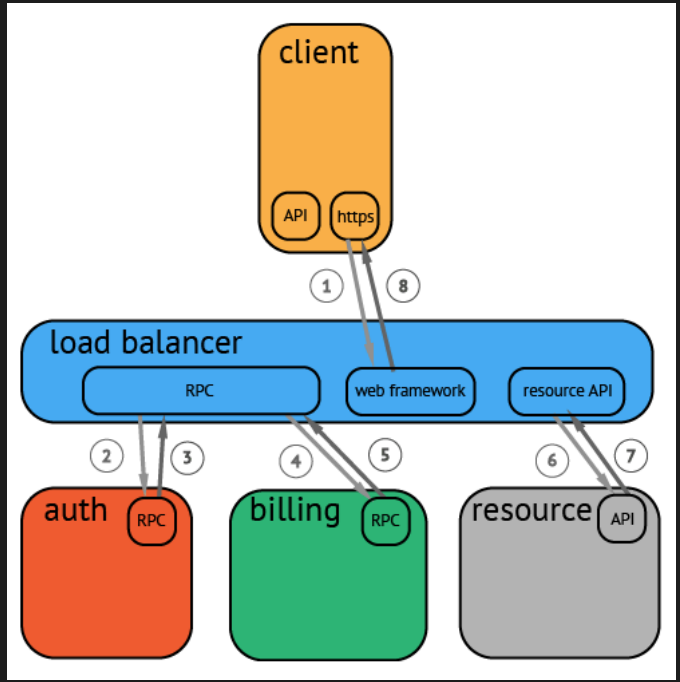


Distributed Tracing

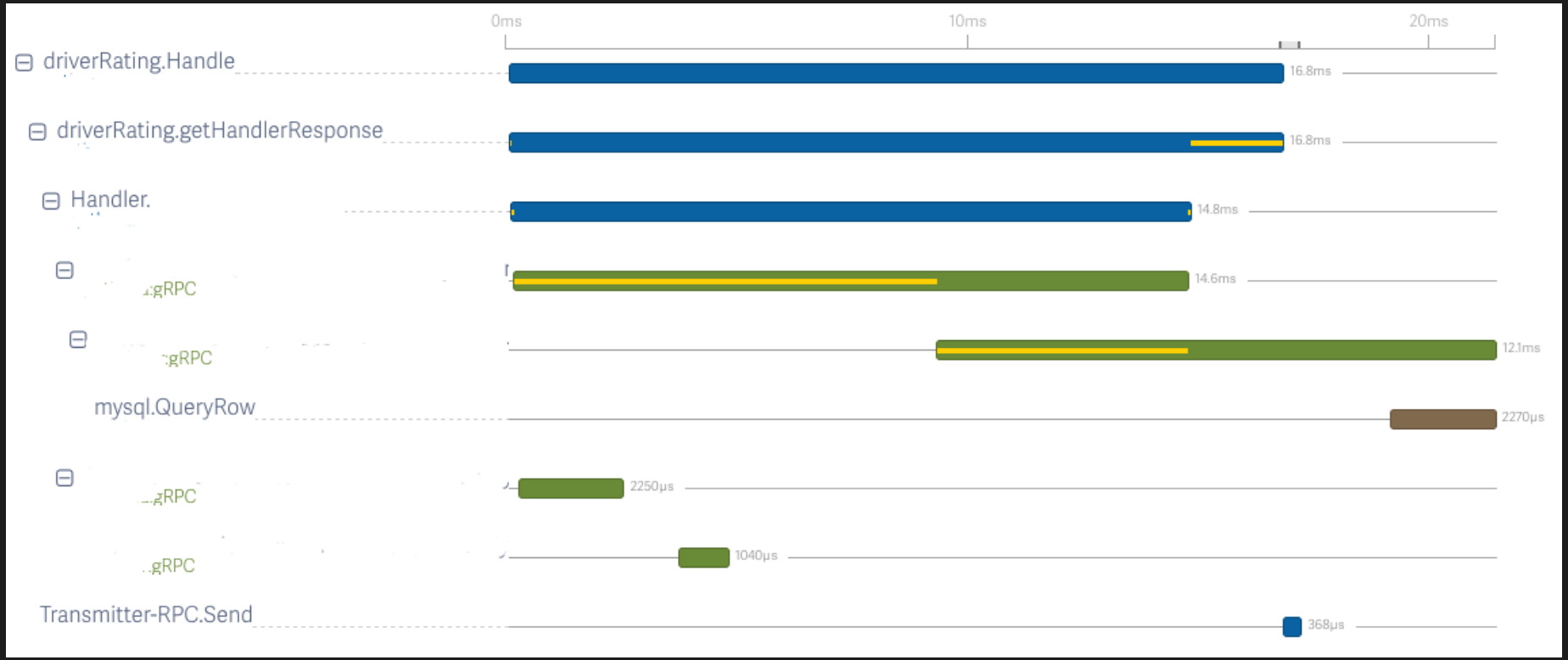
- 应用场景
  - 一个请求耗时三秒才能完成，如何诊断何处耗时最多？
  - 如何定位Single Point of Failure？
  - 如何检测并避免循环依赖关系？
  - 如何定位Fan In，Fan Out？
- 实现原理
  - 在API Gateway生成一个全局唯一的traceID，并将其注入请求的Header里。
  - 在该请求的每个耗时节点生成一个spanID，以traceID+spanID为索引计时，并记录其他元数据
  - 将tracing信息自动传入每个耗时操作
  - 最后以traceID为key来聚合所有诊断信息

Distributed Tracing

- 一个基本的Tracing



Distributed Tracing



Distributed Tracing

- context.Context

```
func (s Server) Handler(ctx context.Context, req Request) error {
        // ...
}
```

- Context的生命周期 == Request在系统中的生命周期
- 多用，善用Context
  - context.WithCancel
  - context.WithTimeout
  - context.WithDeadline
  - context.WithValue
- OpenTracing http://opentracing.io/

Testing

- 单元测试
- 端到端测试

单元测试

```
func TestMyFunction(t *testing.T) {
        scenarios := []struct {
                desc     string
                input    int64
                expected error
        }{
                {
                        "Happy Path",
                        1,
                        nil,
                },
                {
                        "Happy Path",
                        1,
                        ErrNotFound,
                },
        }
        for _, scenario := range scenarios {
                t.Run(scenario.desc, func(t *testing.T) {
                        err := MyFunction(scenario.input)
                        assert.Equal(t, scenario.expected, err)
                })
        }
}
```

单元测试 - Mock - Bad Version

```
var globalDB DB

type Server struct{}

func (s Server) Handler(ctx context.Context, req Request) error {
                    // ...
                    return globalDB.Write(1)
}

func TestHandler(t *testing.T) {
                    s := Server{}
                    err := s.Handler(context.Background(), Request{})
                    assert.Nil(t, err)
}
```

单元测试 - Mock - Better Version

```
type DB interface {
        Write(interface{}) error
        Read() (interface{}, error)
}

type Server struct {
        db DB
}

func TestHandler(t *testing.T) {
        s := &Server{
                db: NewMockDB(),
        }
        err := s.Handler(context.Background(), Request{})
        assert.Nil(t, err)
}
```

Testing - 端到端测试

- Staging集群
- Production集群
- 端到端测试发生在Staging集群上
  - 真实的环境 - DB, Redis, Etcd, DynamoDB
  - Postman
  - Go Test

Code Quality Control - Code Review

- Code Review非常重要
- 但是Code Review的重要性经常会被忽视
- 好的工具能够提高Code Review的效率
- 我们所使用的工具
  - Phabricator
  - Jenkins
  - Slackbot

Code Quality Control - Code Review

当提交代码到服务器请求review时

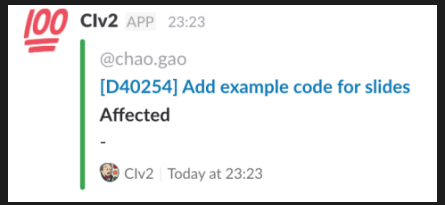
- Phabricator + Arcanist
- 自动运行Linter和单元测试引擎


```
Linting...
Using lint cache, use '--cache 0' to disable it.
passed
LINT OKAY No lint problems.
Running unit tests...
  PASS  <1ms★ GoTest
  PASS  <1ms★ coverage
  PASS  <1ms★ GoTest
  PASS  <1ms★ GoTest
  PASS  <1ms★ GoTest
  PASS  <1ms★ GoTest
  PASS  <1ms★ coverage
  PASS  <1ms★ GoTest

COVERAGE REPORT
    0%
    0%
    0%
   68%
   70%
UNIT OKAY No unit test failures.
SKIP STAGING No staging area is configured for this repository.
Updated an existing Differential revision:
       Revision URI:
```

Code Quality Control - Build Passed

当CI Build成功时

- 代码作者会收到Slackbot的祝贺消息
- Reviewer可以开始代码审查
- Phabricator页面上会清晰的标注测试覆盖和未覆盖到的代码块

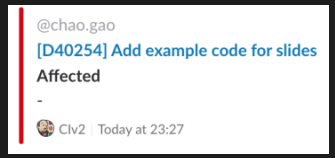


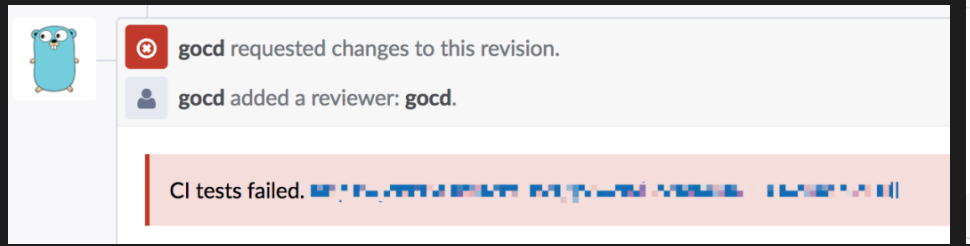

```go
package modules

import "strings"

// SplitString split string by supplied separator
func SplitString(s string, sep string) []string {
  if s == "" {
    return nil
  }
  return strings.Split(s, sep)
}
```

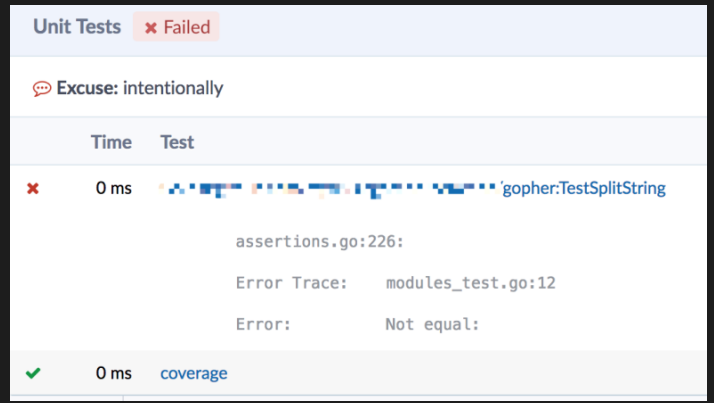
Code Quality Control - Build Failed

当CI Build失败后，会发生三件事情

- Slackbot发送消息给代码作者通知build failure
- Phabricator会自动Reject Diff
- Phabricator Code Review工具会显示哪个测试没通过







Code Quality Control - Test Coverage

- 当单元测试覆盖率低于一定百分比时，CI Build会主动失败

```
04:43:08   %                Statements
04:43:08 79.20                 125
04:43:08 95.12                  82
04:43:08 82.67                 531
04:43:08 81.74                 241
04:43:08 81.82                  33
04:43:08 100.00                 20
04:43:08 94.12                  17
04:43:08 87.28                 574
04:43:08 80.00                  25
04:43:08 80.95                  21
04:43:08 100.00                  1
04:43:08 92.31                  13
04:43:08
04:43:08
04:43:08 ~~~ COVERAGE CHECK COMPLETE ~~~
04:43:08 Finished: SUCCESS
```

```
03:14:39   %                Statements
03:14:39 80.00                  5
03:14:39 100.00                 1
03:14:39 58.33                 48
03:14:39 100.00                 7
03:14:39 53.33                 30
03:14:39 100.00                 1
03:14:39 100.00                 1
03:14:39 50.00                  2
03:14:39 100.00                 4
03:14:39 100.00                 1
03:14:39 100.00                 1
03:14:39 100.00                 1
03:14:39 100.00                 1
03:14:39 100.00                 4
03:14:39 100.00                 4
03:14:39 100.00                 1
03:14:39 100.00                 1
03:14:39 88.46                 26
03:14:39 55.56                  9
03:14:39 100.00                 4
```

Bugs

- 我们曾经遇到的与Go相关的问题
  - Nil Pointer
  - DNS Resolution

Nil Pointer

```
7  type A struct {
8          value string
9  }
10
11 func (a *A) Test() string {
12         log.Print("Test
Called")
13     return a.value
14 }
15
16 func getA() *A {
17         return nil
18 }
19
20 func main() {
21         a := getA()
22         a.Test()
23 }
```

```
2009/11/10 23:00:00 Test Called
panic: runtime error: invalid memory address or nil pointer
dereference
[signal SIGSEGV: segmentation violation code=0xffffffff
addr=0x0 pc=0xd1566]

goroutine 1 [running]:
main.(*A).Test(0x0, 0x10434080, 0x104000f0, 0x104000f0)
        /tmp/sandbox547930359/main.go:13 +0xa6
main.main()
        /tmp/sandbox547930359/main.go:22 +0x20
```

Make zero values useful

```
7  type A struct {
8            value string
9  }
10
11 func (a *A) Test() string {
12     if a == nil {
13         return ""
14     }
15           log.Print("Test
Called")
16     return a.value
17 }
```

- https://www.youtube.com/watch?v=PAAkCSZUG1c&t=6m25s

DNS Resolution

- AWS ELB
  - 请求没有被均衡的分布在不同的机器上
  - 10.0.1.* + 10.0.2.*
- RFC 6724
  - IP地址选择
  - IPv4 or IPv6?
- CGo and Go
  - C library skips sorting when it sees only IPv4 addresses on the outgoing interfaces
  - Go library sorts IP addresses with a subset of RFC rules
- Fix in Go 1.9
  - Sort only ipv6 address and rely on DNS results for the rest
- https://github.com/golang/go/issues/18518

Grab还在用Golang做什么

- Stream processing
  - 整合数以亿计的乘客，司机以及交通信息
- Serverless Architecture
  - 函数式计算
- 机器学习平台
  - 打造工程师与数据科学家之间的桥梁
  - 将人工智能融入平台的每一部分

We’re hiring

Q & A

References

- https://techcrunch.com/2017/03/21/grab-uber-myanmar/
- http://opentracing.io/
- https://lightstep.com/