Building a Debuggable Server in Go

Keeley Erhardt
keeley@improbable.io
@KeeleyErhardt

SREcon EMEA 2018, Dusseldorf
What should I get out of this talk?

- Building production services is hard
- Existing solutions
- Intro to radicle
  - Overview
  - Demo
  - Benefits
- Key takeaways
What should I get out of this talk?

- Building production services is hard
- Existing solutions
- Intro to radicle
  - Overview
  - Demo
  - Benefits
- Key takeaways
Service

- A collection of actions the server can perform at the client’s request

```go
package main

import "github.com/keelerh/demo-radicle"

service Greeter {
    rpc SayHello (HelloRequest) returns (HelloResponse) {} 
}

message HelloRequest {
    string name = 1;
}

message HelloResponse {
    string message = 1;
}
```

github.com/keelerh/demo-radicle
Service

- A collection of actions the server can perform at the client’s request

```protobuf
github.com/keelerh/demo-radicle

service Greeter {
  rpc SayHello (HelloRequest) returns (HelloResponse) {};

message HelloRequest {
  string name = 1;
}

message HelloResponse {
  string message = 1;
}
```
Service

- A collection of actions the server can perform at the client’s request
Service

- A collection of actions the server can perform at the client’s request

```go
type GreeterService struct {}

func (s *GreeterService) SayHello(ctx context.Context, req *pb.HelloRequest) (*pb.HelloResponse, error) {
    return &pb.HelloResponse{
        Message: fmt.Sprintf("Hello %s", req.Name),
    }, nil
}
```

github.com/keelerh/demo-radicle
Service

- A collection of actions the server can perform at the client’s request

type GreeterService struct {}

func (s *GreeterService) SayHello(ctx context.Context, req *pb.HelloRequest) (*pb.HelloResponse, error) {
    return &pb.HelloResponse{
        Message: fmt.Sprintf("Hello %s", req.Name),
    }, nil
}
Service

- A collection of actions the server can perform at the client’s request

type GreeterService struct {}

func (s *GreeterService) SayHello(
ctx context.Context, req *pb.HelloRequest)
(*pb.HelloResponse, error) {
    return &pb.HelloResponse{
        Message: fmt.Sprintf("Hello %s", req.Name),
    }, nil
}
Service

- A collection of actions the server can perform at the client’s request

```go
package main

import (  
  "context"  
  "fmt"  
  "github.com/golang/protobuf/ptb_UNSPECIFIED"
)

// GreeterService is the server implementation of the gRPC HelloService interface.
func (s *GreeterService) SayHello(ctx context.Context, req *pb.HelloRequest) (*pb.HelloResponse, error) {  
  return &pb.HelloResponse{  
    Message: fmt.Sprintf("Hello %s", req.Name),  
  }, nil
}
```

github.com/keelerh/demo-radicle
Service

- A collection of actions the server can perform at the client’s request

```go
package demo_radicle

type GreeterService struct {}

func (s *GreeterService) SayHello(ctx context.Context, req *pb.HelloRequest) (*pb.HelloResponse, error) {
    return &pb.HelloResponse{
        Message: fmt.Sprintf("Hello %s", req.Name),
    }, nil
}
```

github.com/keelerh/demo-radicle
gRPC + Protobufs

- **gRPC**: a high performance, open-source remote procedure call (RPC) framework
- **Protobufs**: the Interface Definition Language (IDL) and underlying message interchange format for gRPC
Server

- A computer program or device that processes requests and delivers data over the network.

```go
def main() {
    lis, err := net.Listen("tcp", fmt.Sprintf(":%d", 8081))
    if err != nil {
        log.Fatal("failed to listen: %v", err)
    }

    grpcServer := grpc.NewServer()
    svc := helloworld.GreeterService{}
    pb.RegisterGreeterServer(grpcServer, &svc)

    if err := grpcServer.Serve(lis); err != nil {
        log.Fatal("failed to serve: %s", err)
    }
}
```

github.com/keelerh/demo-radicle
Server

- A computer program or device that processes requests and delivers data over the network

```
func main() {
    lis, err := net.Listen("tcp", fmt.Sprintf(":%d", 8081))
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }

    grpcServer := grpc.NewServer()
    svc := helloworld.GreeterService{}
    pb.RegisterGreeterServer(grpcServer, &svc)

    if err := grpcServer.Serve(lis); err != nil {
        log.Fatalf("failed to serve: %s", err)
    }
}
```

github.com/keelerh/demo-radicle
Server

- A computer program or device that processes requests and delivers data over the network

```go
func main() {
    lis, err := net.Listen("tcp", fmt.Sprintf(":%d", 8081))
    if err != nil {
        log.Fatal("failed to listen: %v", err)
    }

    grpcServer := grpc.NewServer()
    svc := helloworld.GreeterService{}
    pb.RegisterGreeterServer(grpcServer, &svc)

    if err := grpcServer.Serve(lis); err != nil {
        log.Fatal("failed to serve: %s", err)
    }
}
```

github.com/keelerh/demo-radicle
Server

- A computer program or device that processes requests and delivers data over the network

```
func main() {
    lis, err := net.Listen("tcp", fmt.Sprintf(":%d", 8081))
    if err != nil {
        log.Fatal("failed to listen: %v", err)
    }

    grpcServer := grpc.NewServer()
    svc := helloworld.GreeterService{}
    pb.RegisterGreeterServer(grpcServer, &svc)

    if err := grpcServer.Serve(lis); err != nil {
        log.Fatal("failed to serve: %s", err)
    }
}
```

github.com/keelerh/demo-radicle
Server

- A computer program or device that processes requests and delivers data over the network

```
func main() {
    lis, err := net.Listen("tcp", fmt.Sprintf("%s", 8081))
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }

    grpcServer := grpc.NewServer()
    svc := helloworld.GreeterService{
        pb.RegisterGreeterServer(grpcServer, &svc)
        if err := grpcServer.Serve(lis); err != nil {
            log.Fatalf("failed to serve: %s", err)
        }
    }
}
```

github.com/keelerh/demo-radicle
Server

- A computer program or device that processes requests and delivers data over the network

```go
func main() {
    lis, err := net.Listen("tcp", fmt.Sprintf(":%d", 8081))
    if err != nil {
        log.Fatal("failed to listen: %v", err)
    }
    grpcServer := grpc.NewServer()
    svc := helloworld.GreeterService{}
    pb.RegisterGreeterServer(grpcServer, &svc)
    if err := grpcServer.Serve(lis); err != nil {
        log.Fatal("failed to serve: %s", err)
    }
}

github.com/keelerh/demo-radicle
Greeter service

✅ Naive implementation

🚫 Production ready
Production service

- Metrics
Production service

Metrics

type Service struct {}

func (s *Service) SayHello(ctx context.Context, req *pb.HelloRequest) (*pb.HelloResponse, error) {
    helloCount.WithLabelValues(req.Name).Add(1)
    return &pb.HelloResponse{
        Message: fmt.Sprintf("Hello %s", req.Name)
    }, nil
}
Production service

- Metrics
- Logging
type GreeterService struct{}

func (s *GreeterService) SayHello(ctx context.Context, req *pb.HelloRequest) (*pb.HelloResponse, error) {
    helloCount.WithLabelValues(req.Name).Add(1)
    log.Printf("grpc: SayHello: %s", req.Name)
    return &pb.HelloResponse{
        Message: fmt.Sprintf("Hello %s", req.Name),
    }, nil
}

Production service

- Metrics
- Logging

github.com/keelerh/demo-radicle
Production service

- Metrics
- Logging
- Tracing
Production service

- Metrics
- Logging
- Tracing
- Rate limiting
Production service

- Metrics
- Logging
- Tracing
- Rate limiting
- ...

![Diagram showing layers of service with Metrics, Logging, Tracing, and Rate limiting]

@KeeleyErhardt
Building production services is hard.
What should I get out of this talk?

- Building production services is hard
- Existing solutions
- Intro to radicle
  - Overview
  - Demo
  - Benefits
- Key takeaways
Existing solutions

- Go Kit
Existing solutions

- Go Kit
Existing solutions

- Go Kit
Existing solutions

- Go Kit
Existing solutions

- Go Kit
- Go Micro
What should I get out of this talk?

- Building production services is hard
- Existing solutions

**Intro to radicle**
- Overview
- Demo
- Benefits

**Key takeaways**
Our solution

- radicle
Our solution

- radicle
  - No new concepts
Our solution

- radicle
  - No new concepts
  - Flexible
Our solution

- radicle
  - No new concepts
  - Flexible
  - Tested
Production services

- Service
- Metrics
- Logging
- Tracing
- Rate limiting
- ...
Production services
Production services
Production services
Production services

Service
- Metrics
- Logging
- Tracing
- Rate limiting

Service
- Metrics
- Logging
- Tracing
- Rate limiting
- Authentication
Production services
Common functionality is duplicated across multiple services.
Common functionality is duplicated across multiple services.
Production services

Service
- Metrics
- Logging
- Tracing
- Rate limiting

Service
- Metrics
- Logging
- Tracing
- Rate limiting

Service
- Metrics
- Logging
- Authentication

Server
Production services
Production services
Production services
Production services
Production services

Base server

Logging  Metrics  Tracing  Rate limiting  Authentication
Production services
Demo

github.com/keelerh/demo-radicle
Benefits of centralized and unified common functionality:

- Guarantees basic operability
Benefits of centralized and unified common functionality:

- Guarantees basic operability
- Standardizes services
Benefits of centralized and unified common functionality:

- Guarantees basic operability
- Standardizes services
- Promotes reusable service infrastructure
Benefits of centralized and unified common functionality:

- Guarantees basic operability
- Standardizes services
- Promotes reusable service infrastructure
- Reduces development time
What should I get out of this talk?

- Building production services is hard
- Existing solutions
- Intro to radicle
  - Overview
  - Demo
  - Benefits
- Key takeaways
1. Production services require a great deal of functionality outside of the service-level logic.
2. Don’t duplicate common functionality across multiple services
3. radicle is one option for simplifying the development of debuggable servers in Go
What should I get out of this talk?

- Building production services is hard
- Existing solutions
- Intro to radicle
  - Overview
  - Demo
  - Benefits
- Key takeaways
Liked what you heard?

say hi on 👤 improbable-eng.slack.com

...or come join us!
improbable.io/careers       improbable.hk/careers

github.com/improbable-eng