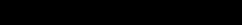
Building a Debuggable Server in Go

Keeley Erhardt
keeley@improbable.io

@KeeleyErhardt

SREcon EMEA 2018, Dusseldorf



Talk overview

What should I get out of this talk?

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

Talk overview

What should I get out of this talk?

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

github.com/keelerh/demo-radicle

Service

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

```
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

```
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

```
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

```
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
}
```

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
}
```

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
}
```

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
}
```

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

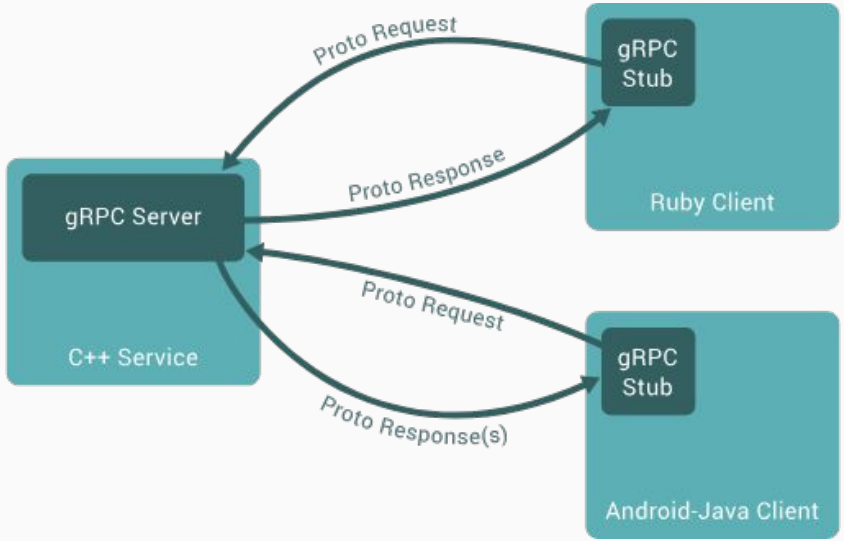


Image Source: http://www.grpc.io

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.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

```
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.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.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.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

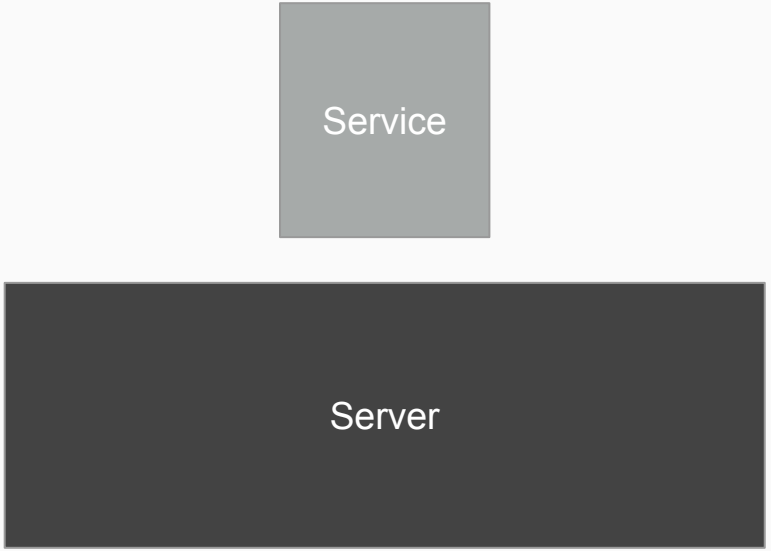
- A computer program or device that processes requests and delivers data over the network

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

Greeter service

- ✓ ~~Naive implementation~~
- ✗ Production ready



Production service

- Metrics

Production service

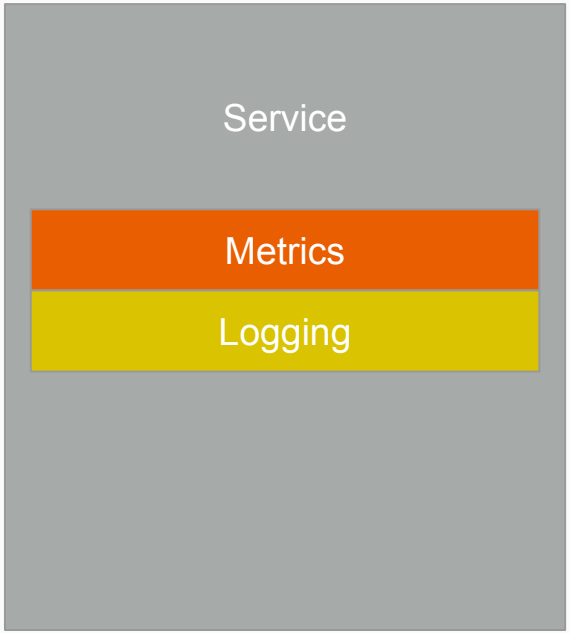
- Metrics

```
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
}
```

github.com/keelerh/demo-radicle

Production service

- Metrics
- Logging



Production service

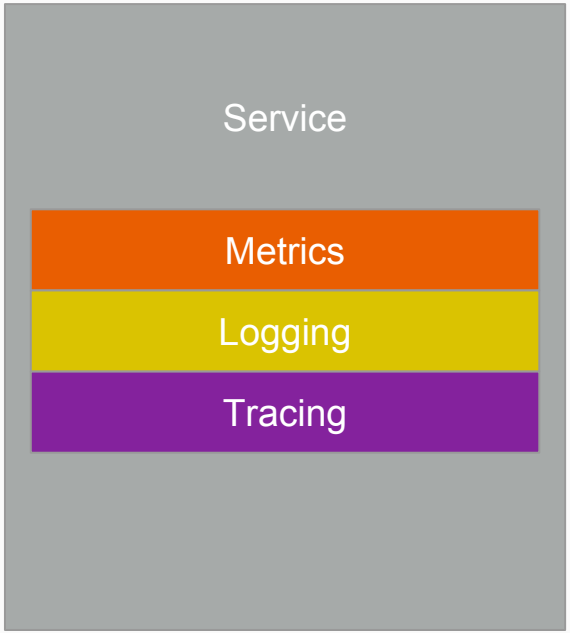
- Metrics
- Logging

```
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
}
```

github.com/keelerh/demo-radicle

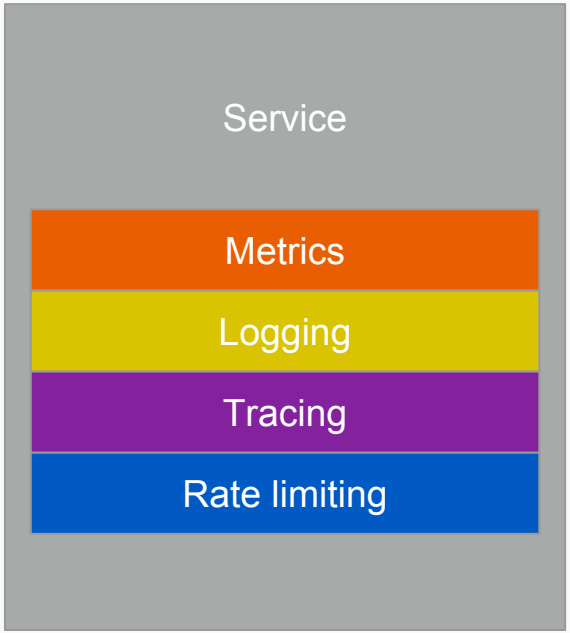
Production service

- Metrics
- Logging
- Tracing



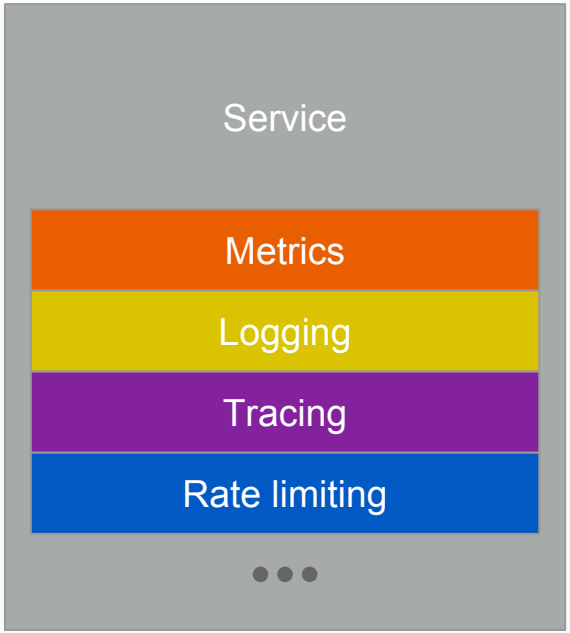
Production service

- Metrics
- Logging
- Tracing
- Rate limiting



Production service

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



Building production services is hard.

Talk overview

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

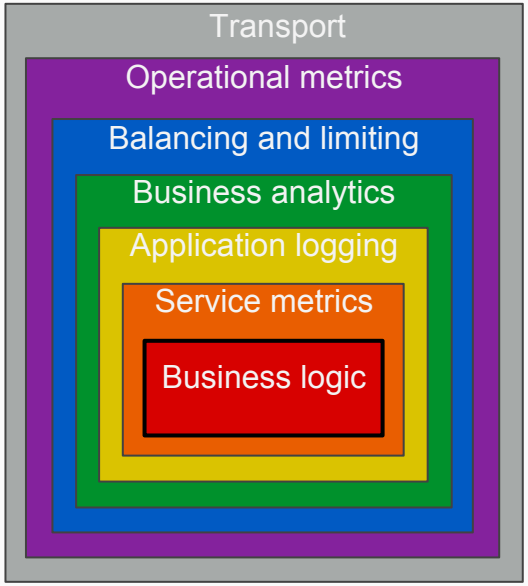


Image Source: https://gokit.io

Existing solutions

- Go Kit

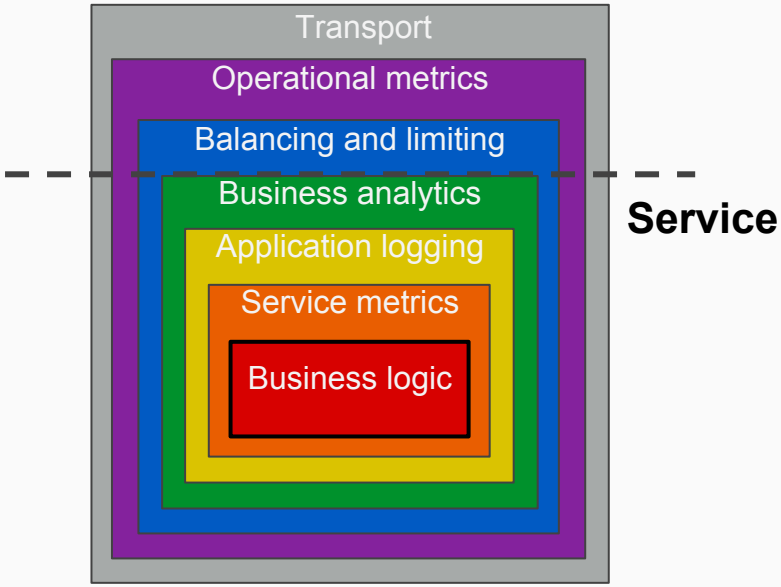


Image Source: https://gokit.io

Existing solutions

- Go Kit

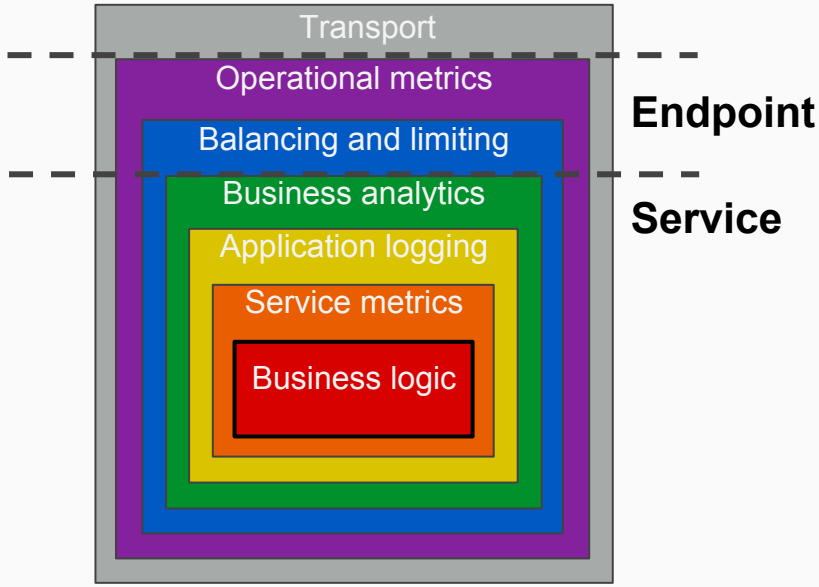


Image Source: https://gokit.io

Existing solutions

- Go Kit

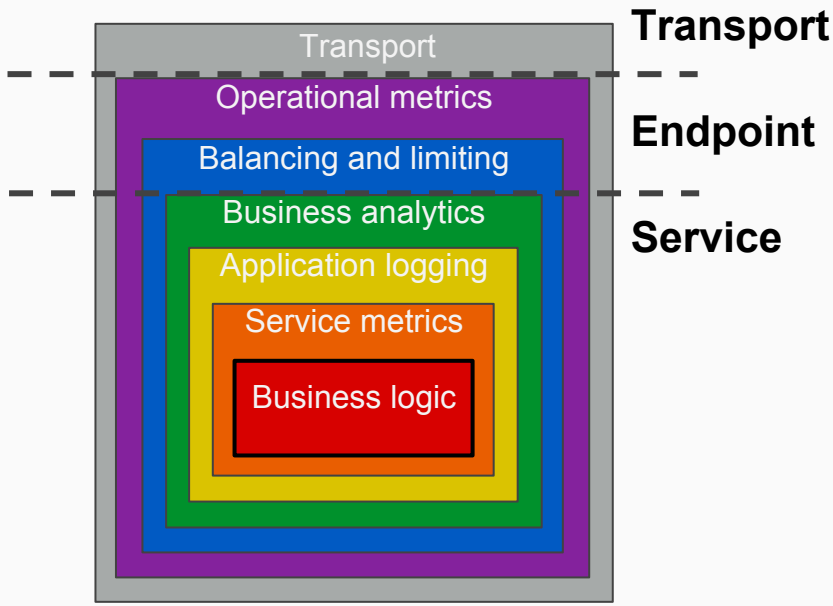


Image Source: https://gokit.io

Existing solutions

- Go Kit
- Go Micro

Image Source: https://micro.mu

Talk overview

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

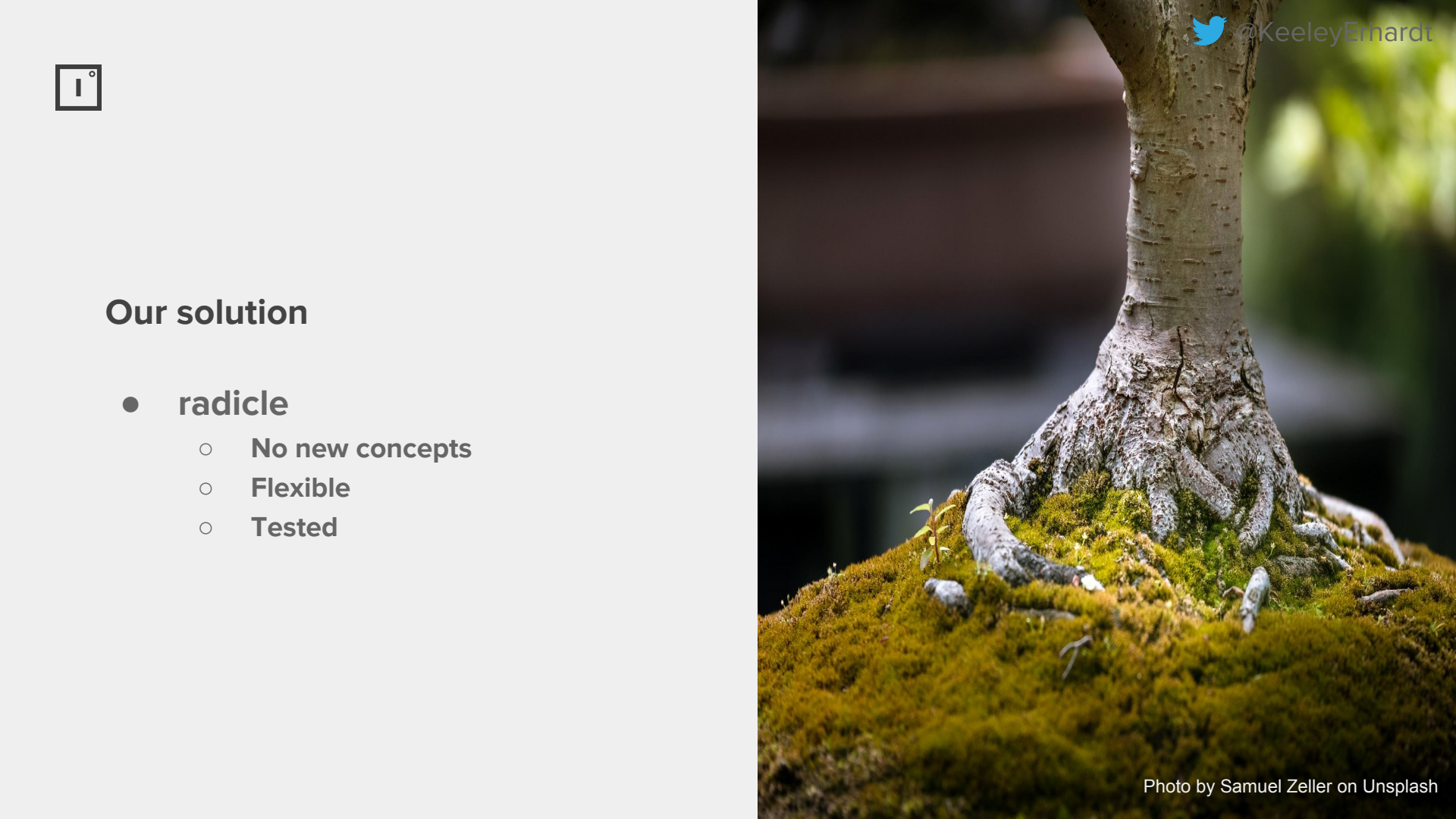
- radicle

Our solution

- radicle
  - No new concepts

Our solution

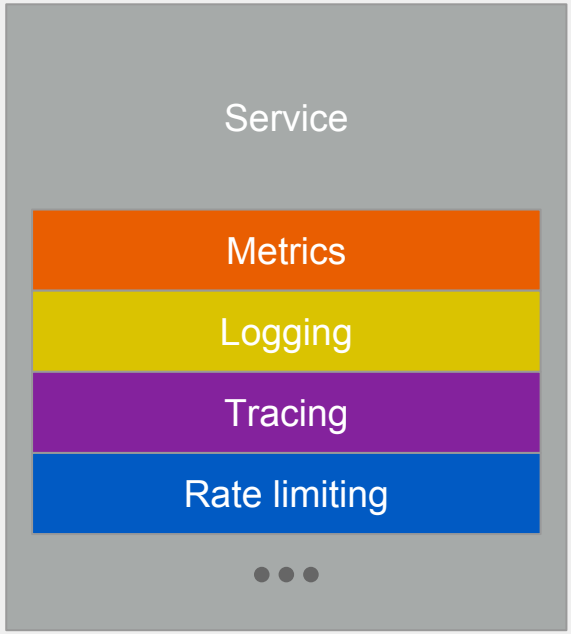
- radicle
  - No new concepts
  - Flexible



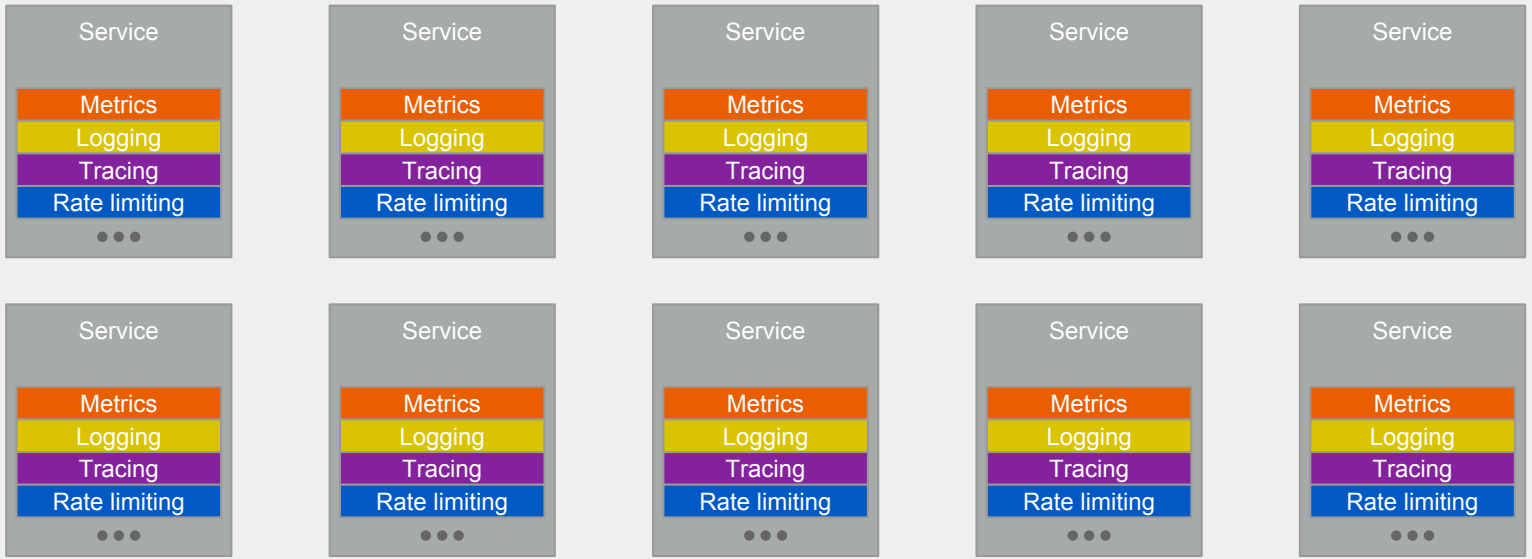
Our solution

- radicle
  - No new concepts
  - Flexible
  - Tested

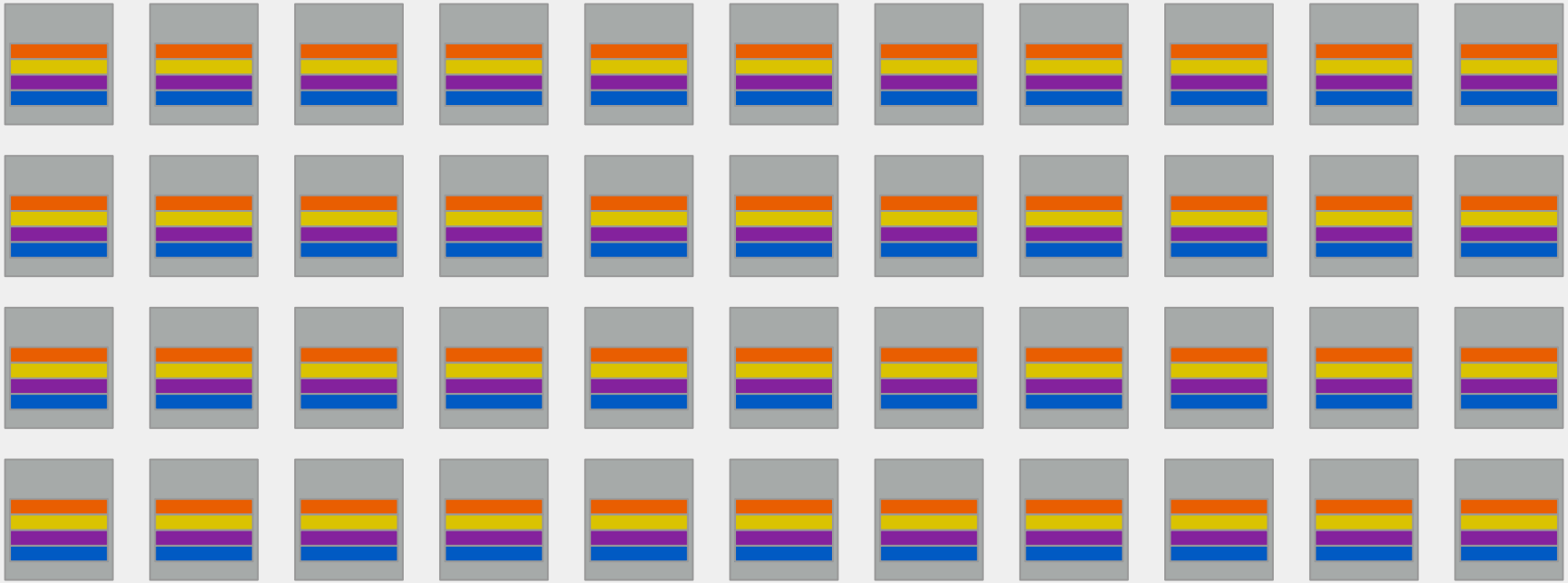
Production services



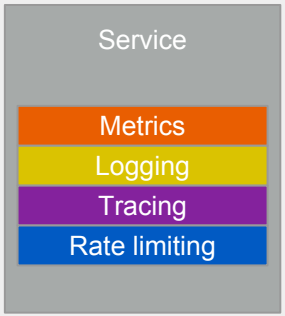
Production services



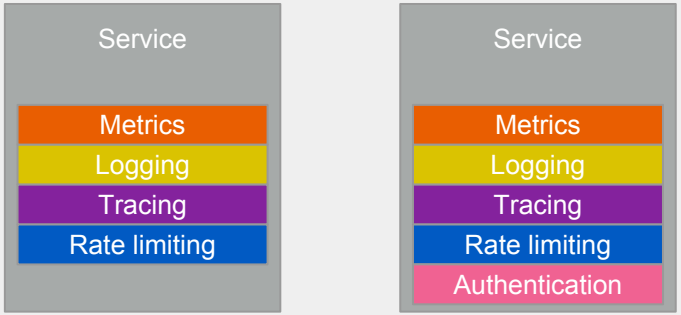
Production services



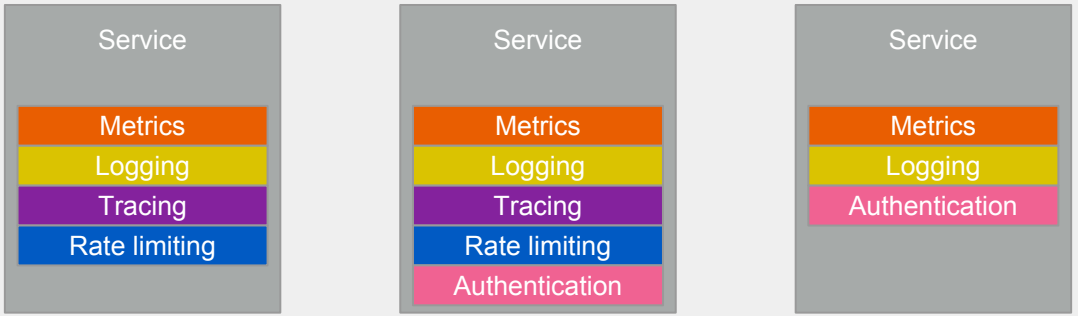
Production services



Production services



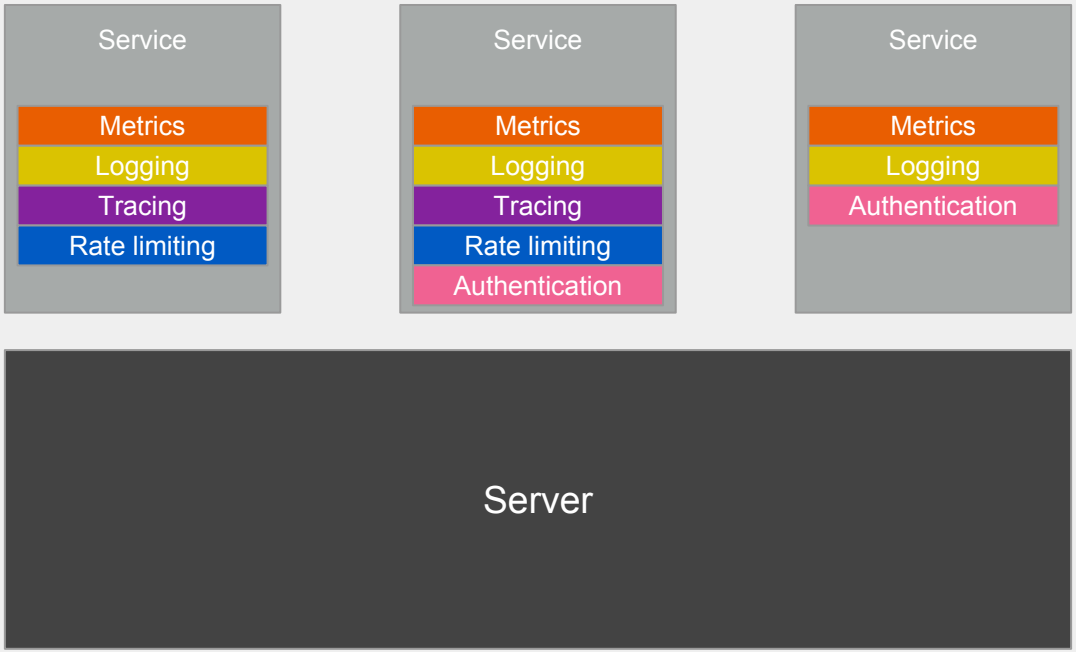
Production services



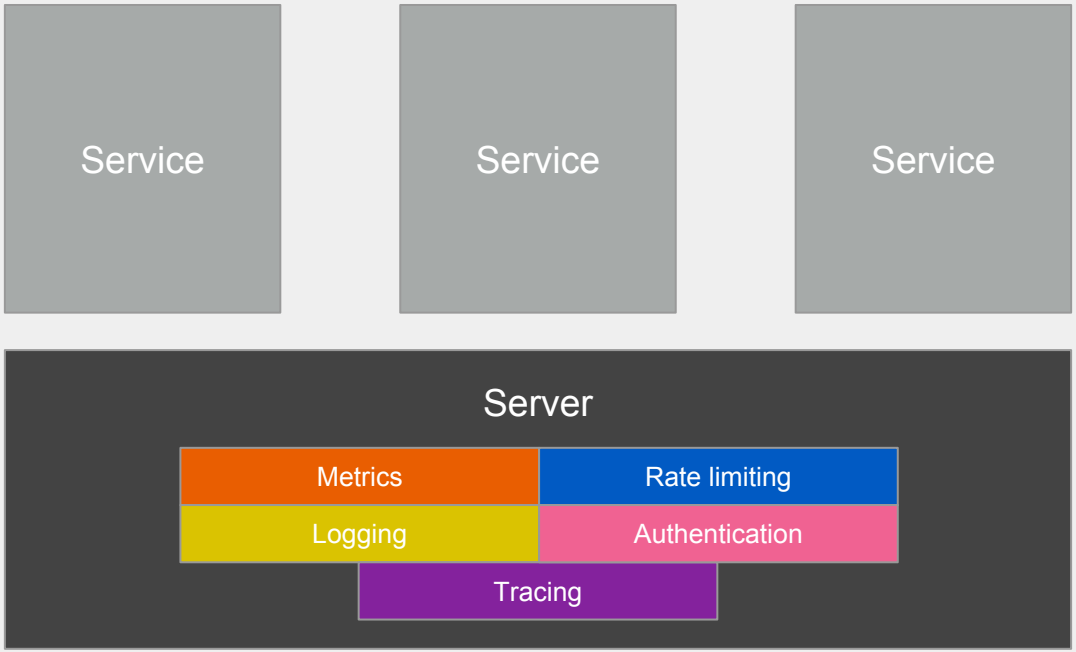
Common functionality is duplicated across multiple services.

~~Common functionality is duplicated across multiple services.~~

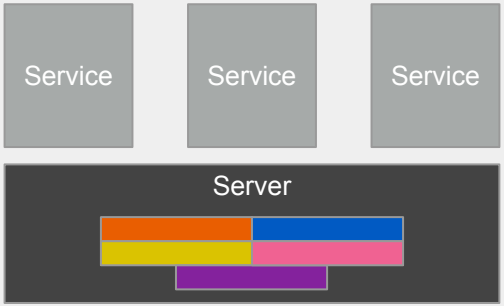
Production services



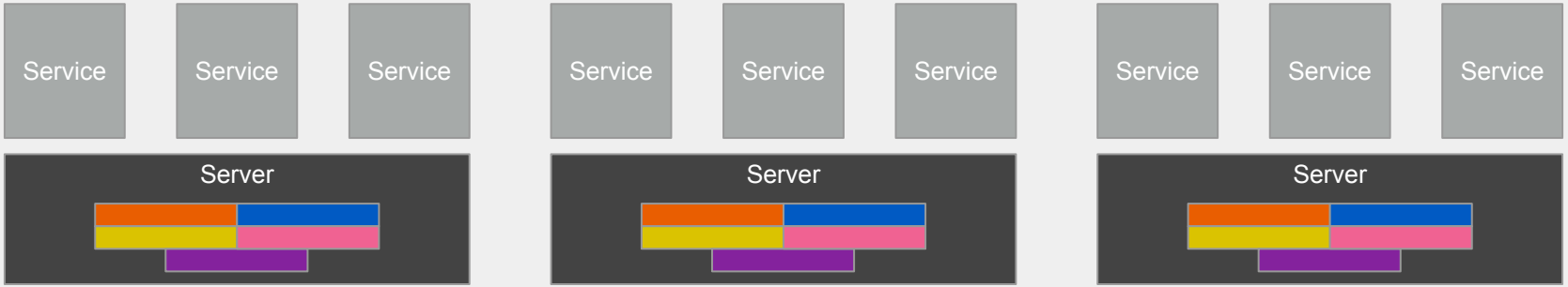
Production services



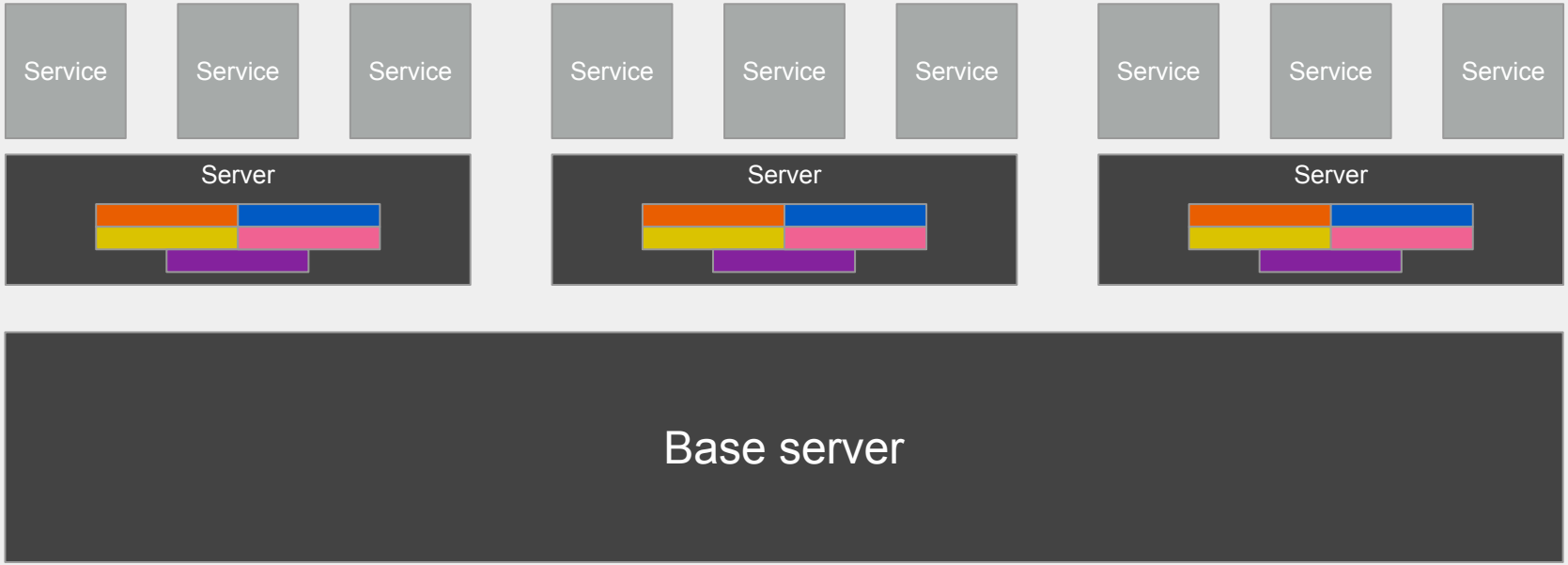
Production services



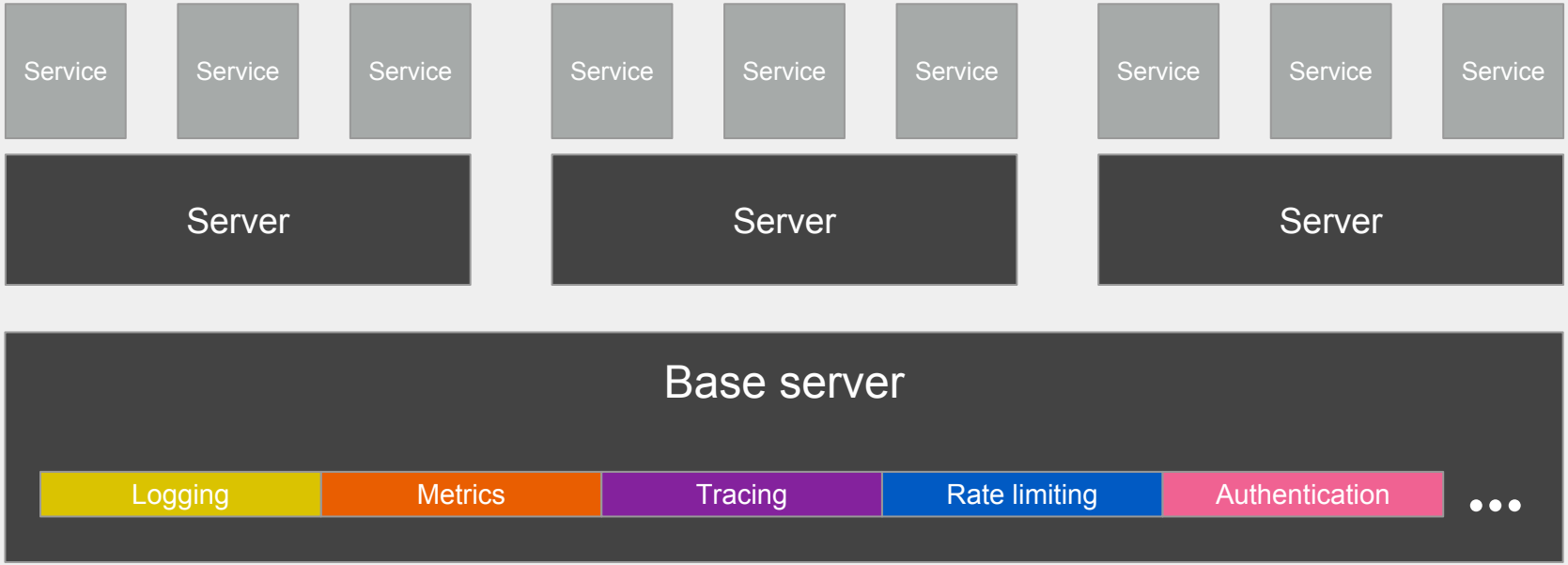
Production services



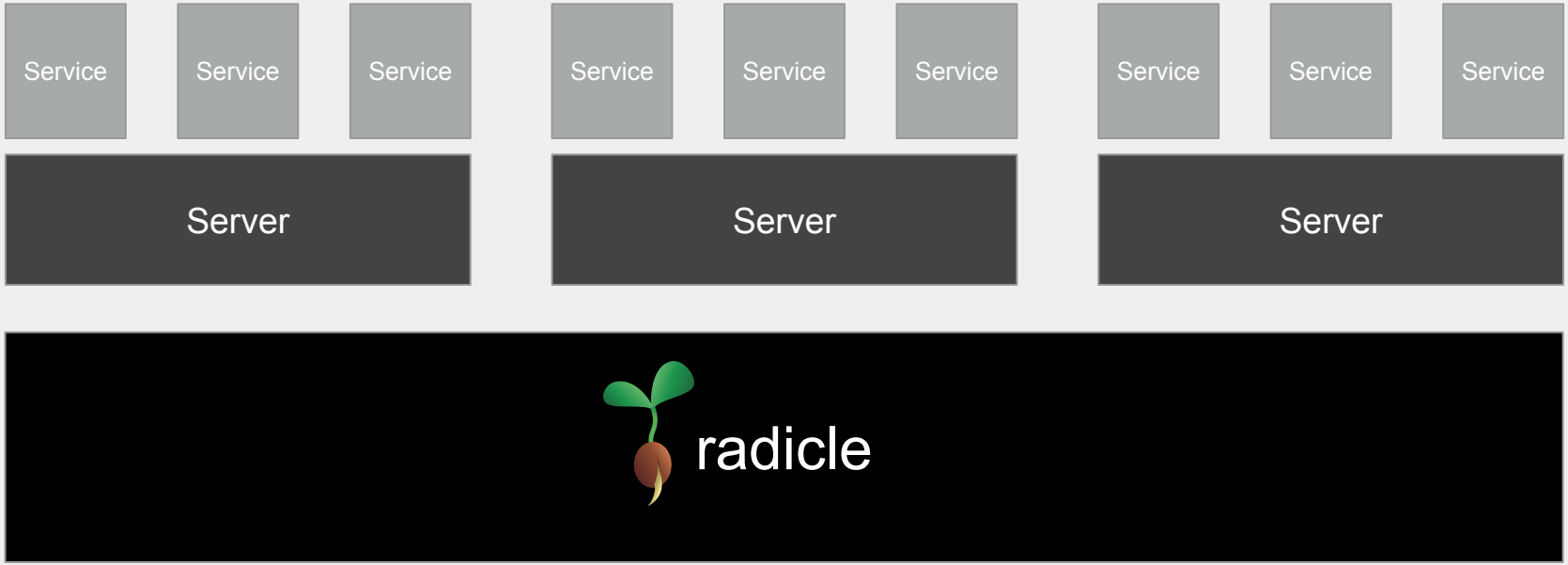
Production services



Production services



Production services



Demo

github.com/keelerh/demo-radicle

Benefits of centralized and unified common functionality:

- Guarantees basic operability

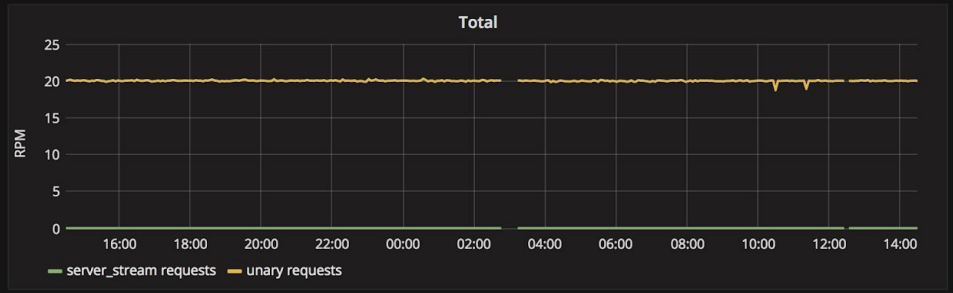
@KeeleyErhardt

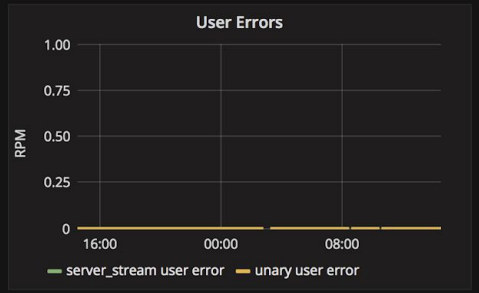
Generic gRPC Server

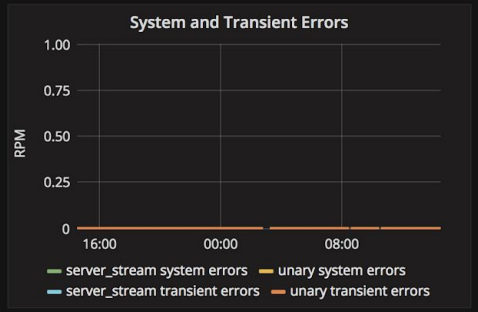
Zoom Out | Last 24 hours UTC

cluster | job | env testing | service All | deployment-probe Dashboards | Kibana Logs

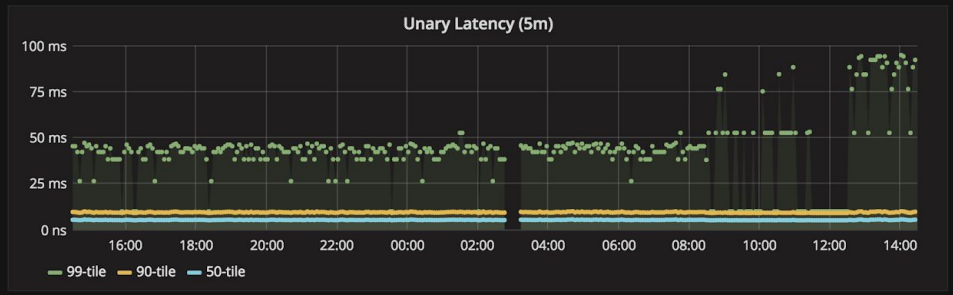
Requests (All)

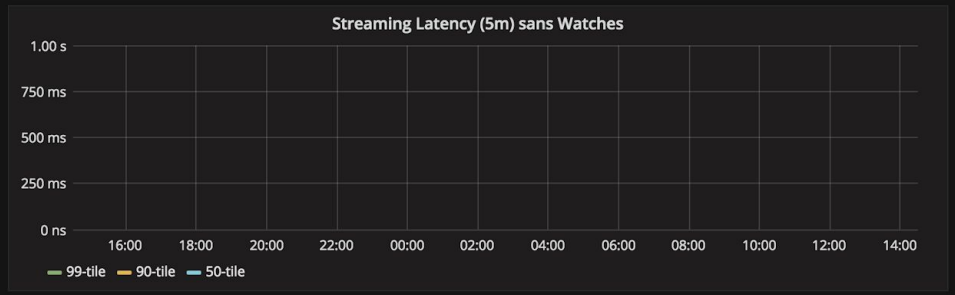






Latencies (All)





Drill Tables

Requests (RPM) Per Method — Last 5 minutes

Metric	Avg	M

Errors (RPM) Per Method Handler — Last 5 minutes

Metric	Avg	Max

Errors (RPM) Per Code Handler — Last 5 minutes

Metric	Avg	Max

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

Talk overview

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

Talk overview

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`