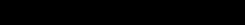
radicle

Building debuggable, production-ready servers in Go

Keeley Erhardt
@KeeleyErhardt

25 JUN 2018, LinuxCon + ContainerCon + CloudOpen China (LC3), Beijing



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

```
// $ protoc -I protos/ \
//     -I${GOPATH}/src \
//     --go_out=plugins=grpc:protos \
//     protos/helloworld.proto
//
// Code generated by protoc-gen-go. DO NOT EDIT.
// source: helloworld.proto
type GreeterServer interface {
    SayHello(context.Context, *HelloRequest)
(*HelloResponse, error)
}
```

github.com/keelerh/demo-radicle

Service

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

```
// $ protoc -I protos/ \
//     -I${GOPATH}/src \
//     --go_out=plugins=grpc:protos \
//     protos/helloworld.proto
//
// Code generated by protoc-gen-go. DO NOT EDIT.
// source: helloworld.proto
type GreeterServer interface {
    SayHello(context.Context, *HelloRequest)
(*HelloResponse, error)
}
```

github.com/keelerh/demo-radicle

Service

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

```
// $ protoc -I protos/ \
//     -I${GOPATH}/src \
//     --go_out=plugins=grpc:protos \
//     protos/helloworld.proto
//
// Code generated by protoc-gen-go. DO NOT EDIT.
// source: helloworld.proto
type GreeterServer interface {
    SayHello(context.Context, *HelloRequest)
(*HelloResponse, error)
}
```

github.com/keelerh/demo-radicle

Service

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

```
// $ protoc -I protos/ \
//     -I${GOPATH}/src \
//     --go_out=plugins=grpc:protos \
//     protos/helloworld.proto
//
// Code generated by protoc-gen-go. DO NOT EDIT.
// source: helloworld.proto
type GreeterServer interface {
    SayHello(context.Context, *HelloRequest)
(*HelloResponse, error)
}
```

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

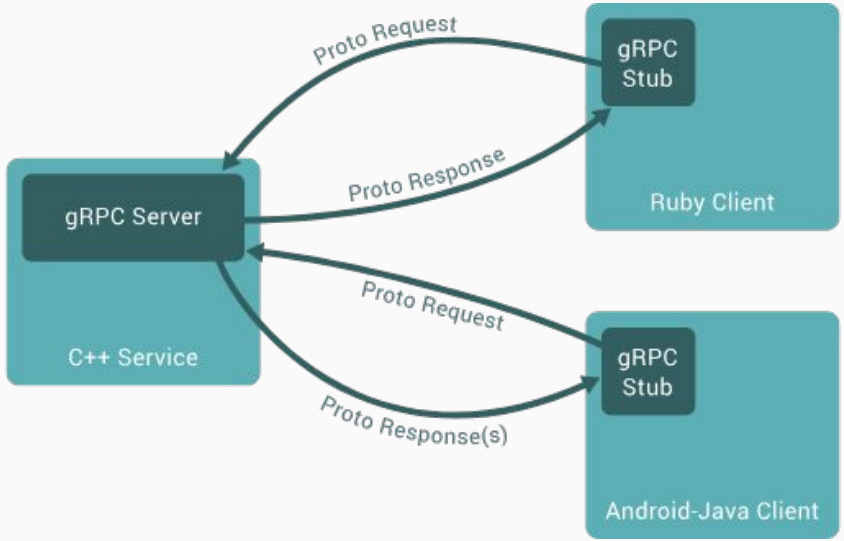


Image Source: http://www.grpc.io

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

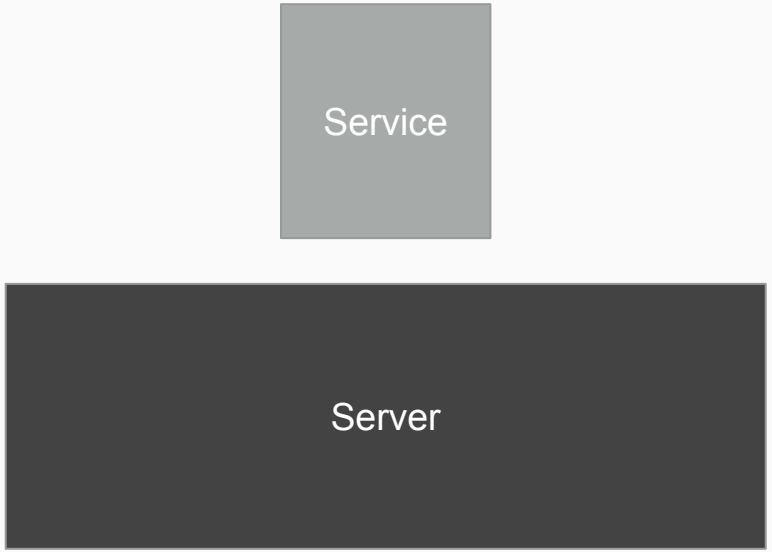
- A computer program or device that processes requests and delivers data over the network

```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

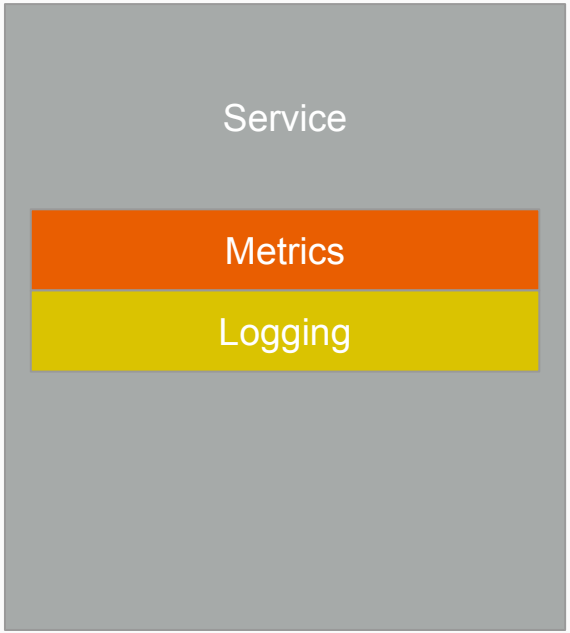
- Metrics

```
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

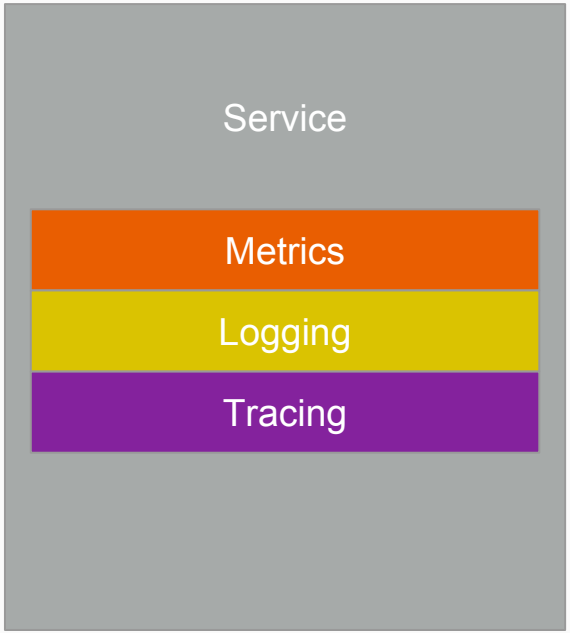
- Metrics
- Logging

```
type GreeterService struct{}

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

github.com/keelerh/demo-allium

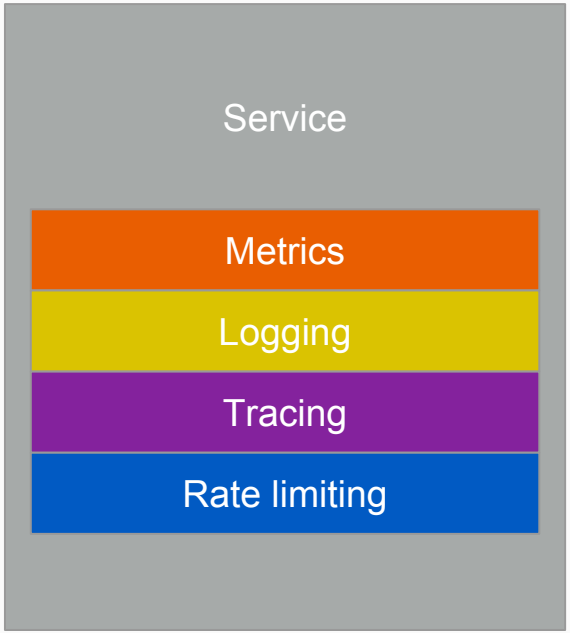
Production service

- Metrics
- Logging
- Tracing



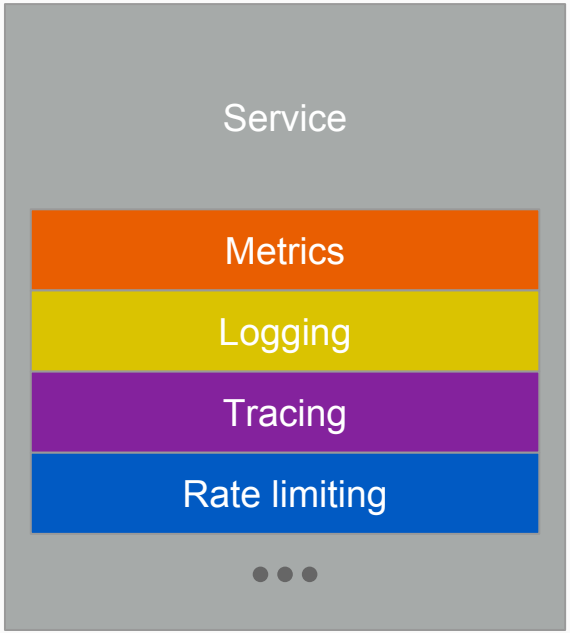
Production service

- 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
  - Example
  - Functionality
- Key takeaways

Existing solutions

- Go Kit

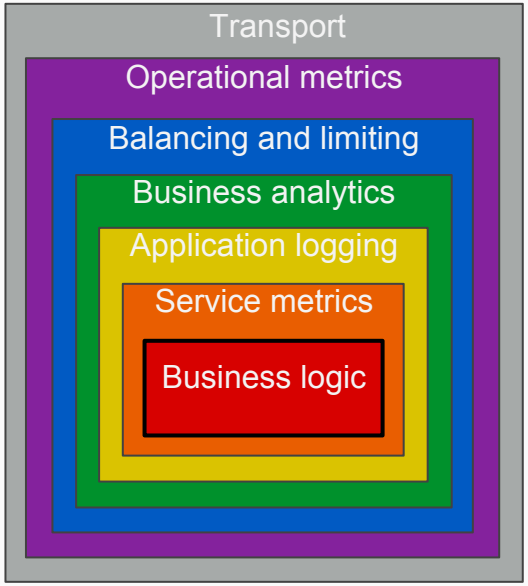


Image Source: https://gokit.io

Existing solutions

- Go Kit

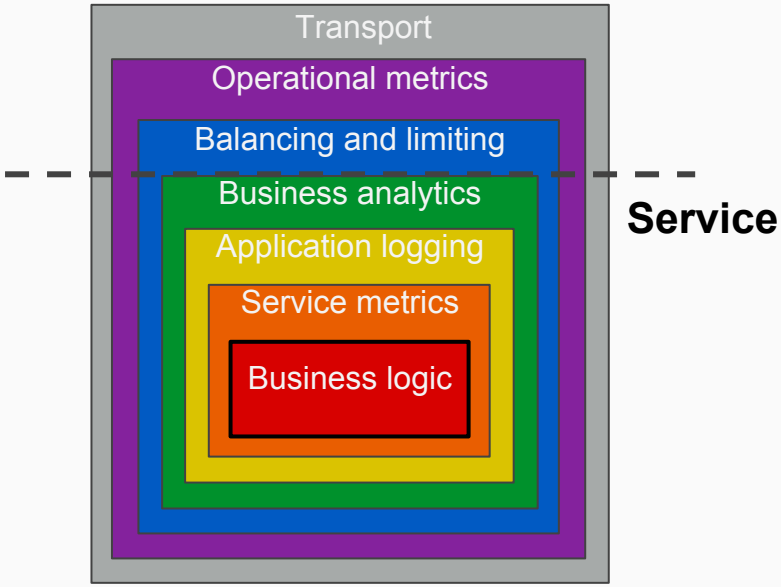


Image Source: https://gokit.io

Existing solutions

- Go Kit

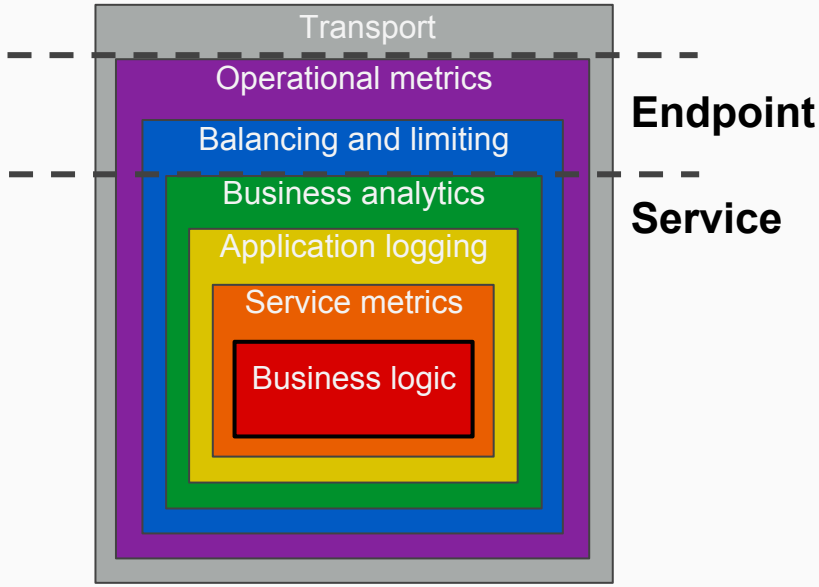


Image Source: https://gokit.io

Existing solutions

- Go Kit

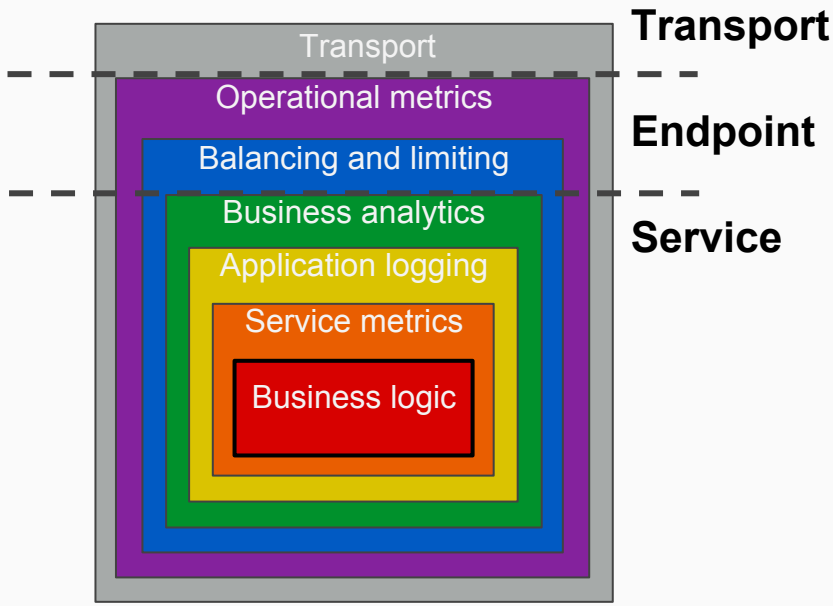


Image Source: https://gokit.io

Existing solutions

- Go Kit
- Go Micro

Image Source: https://micro.mu

radicle: debuggable, production-ready Go servers

radicle (noun) the part of a plant embryo
that develops into the primary root.

radicle

- No new concepts

Photo by Samuel Zeller on Unsplash

radicle

- No new concepts
- Easy to use

Photo by Samuel Zeller on Unsplash

radicle

- No new concepts
- Easy to use
- Flexible

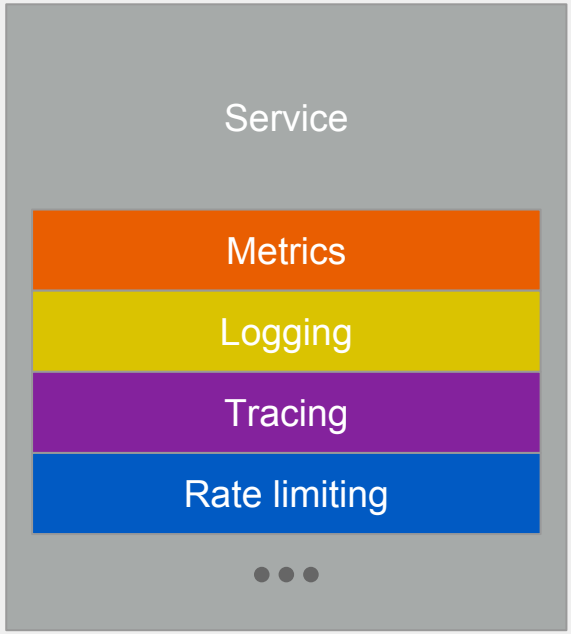
Photo by Samuel Zeller on Unsplash

radicle

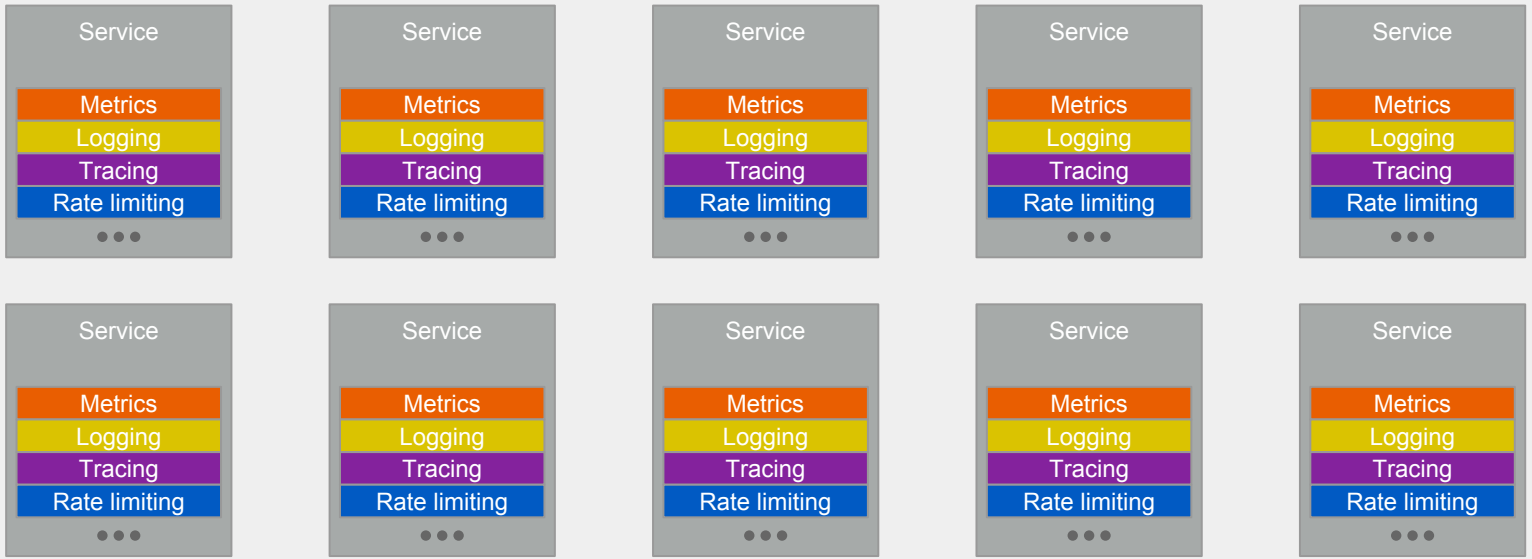
- No new concepts
- Easy to use
- Flexible
- Tested

Photo by Samuel Zeller on Unsplash

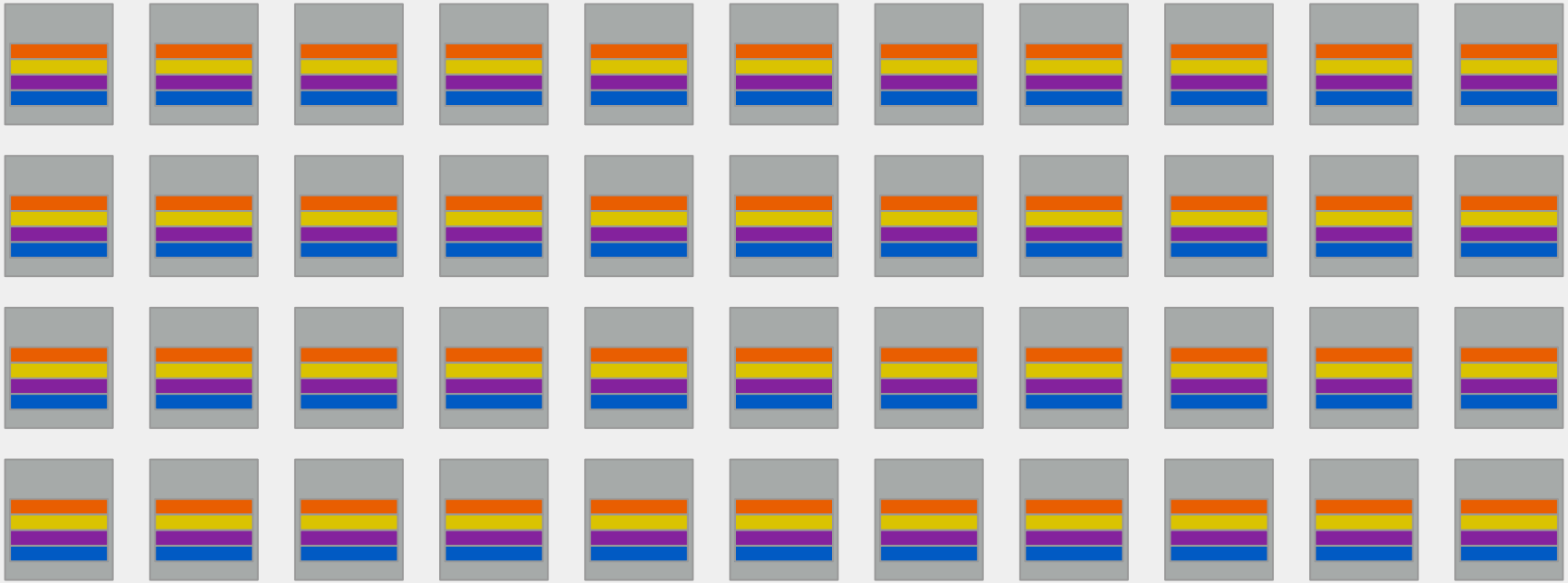
Production services



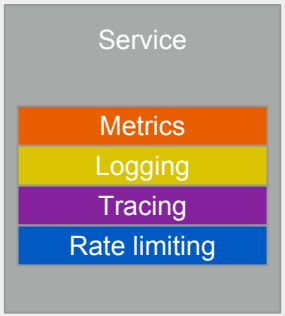
Production services



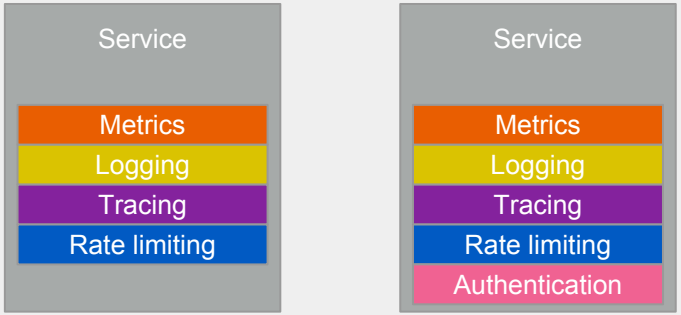
Production services



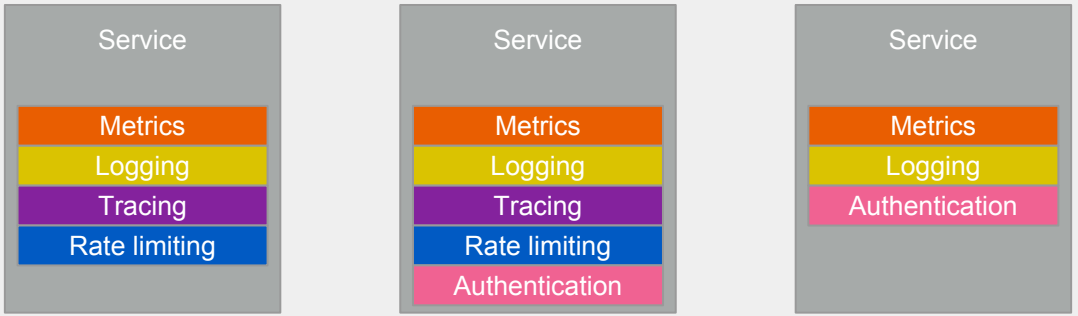
Production services



Production services



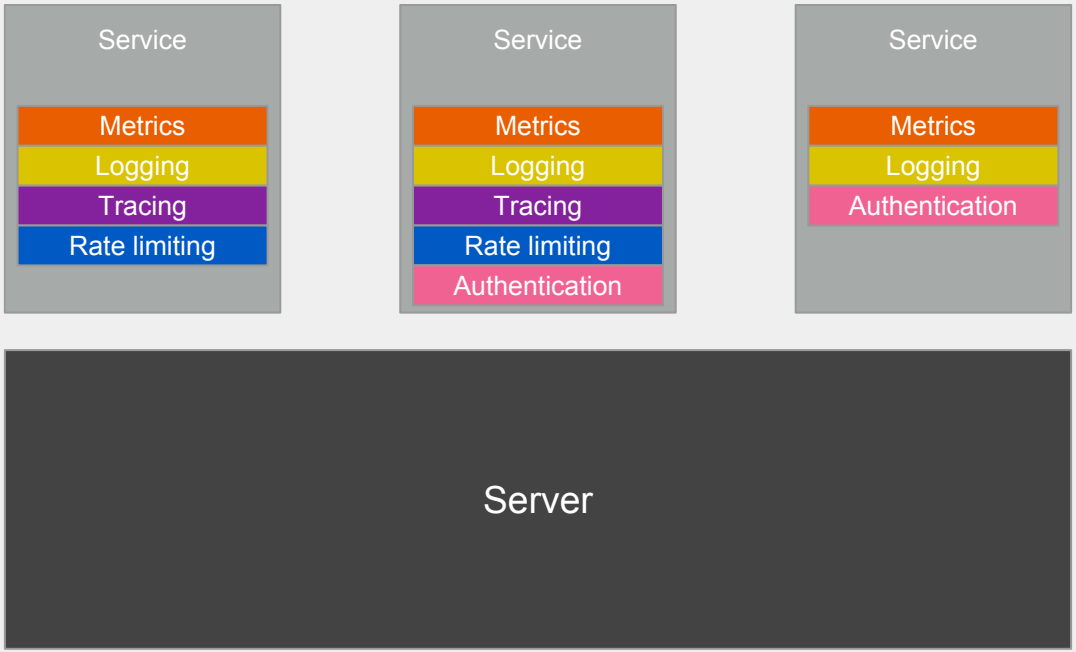
Production services



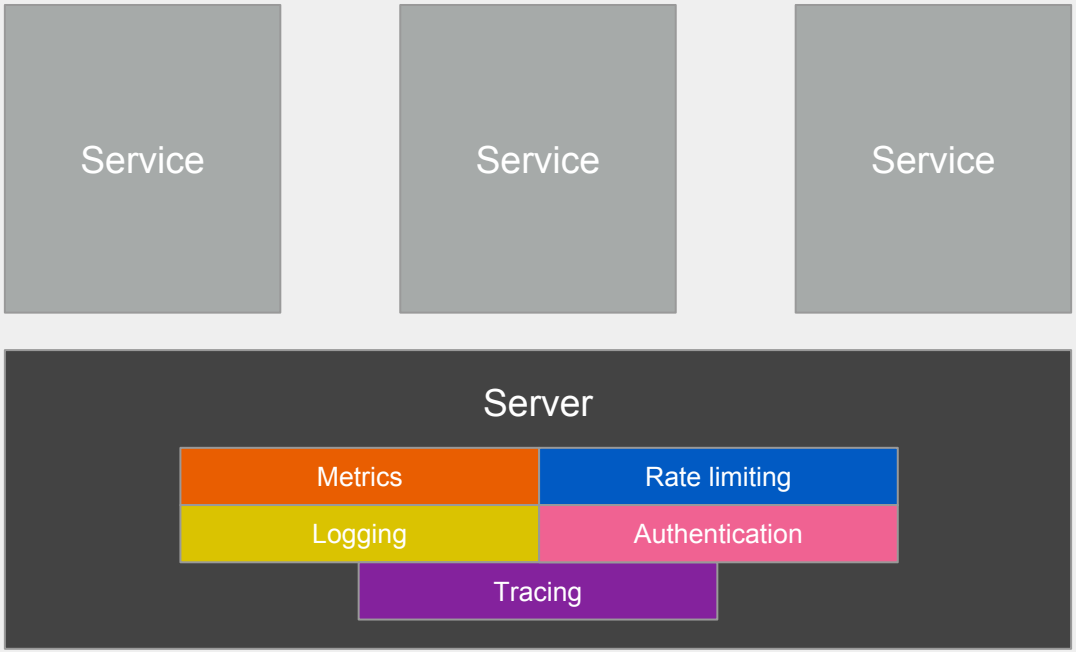
Common functionality is duplicated across multiple services.

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

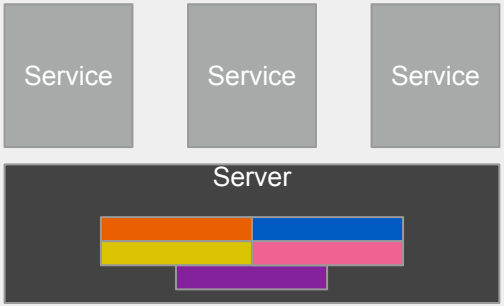
Production services



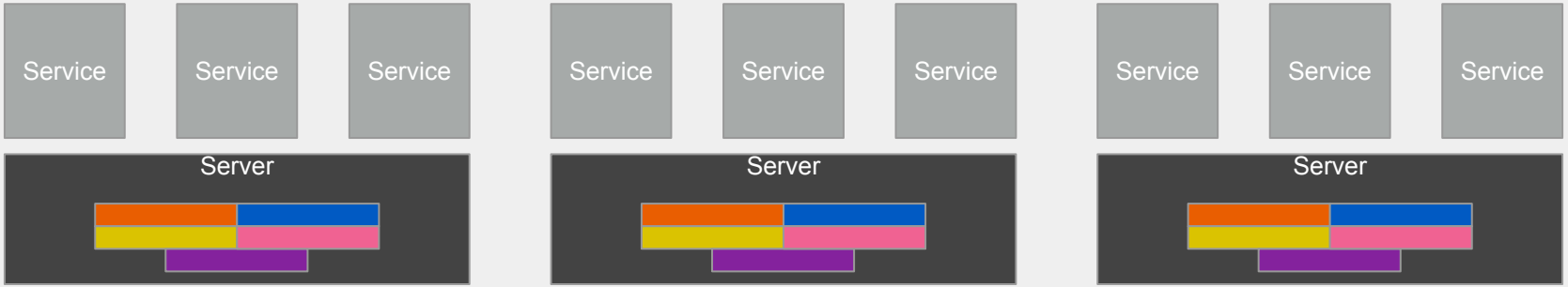
Production services



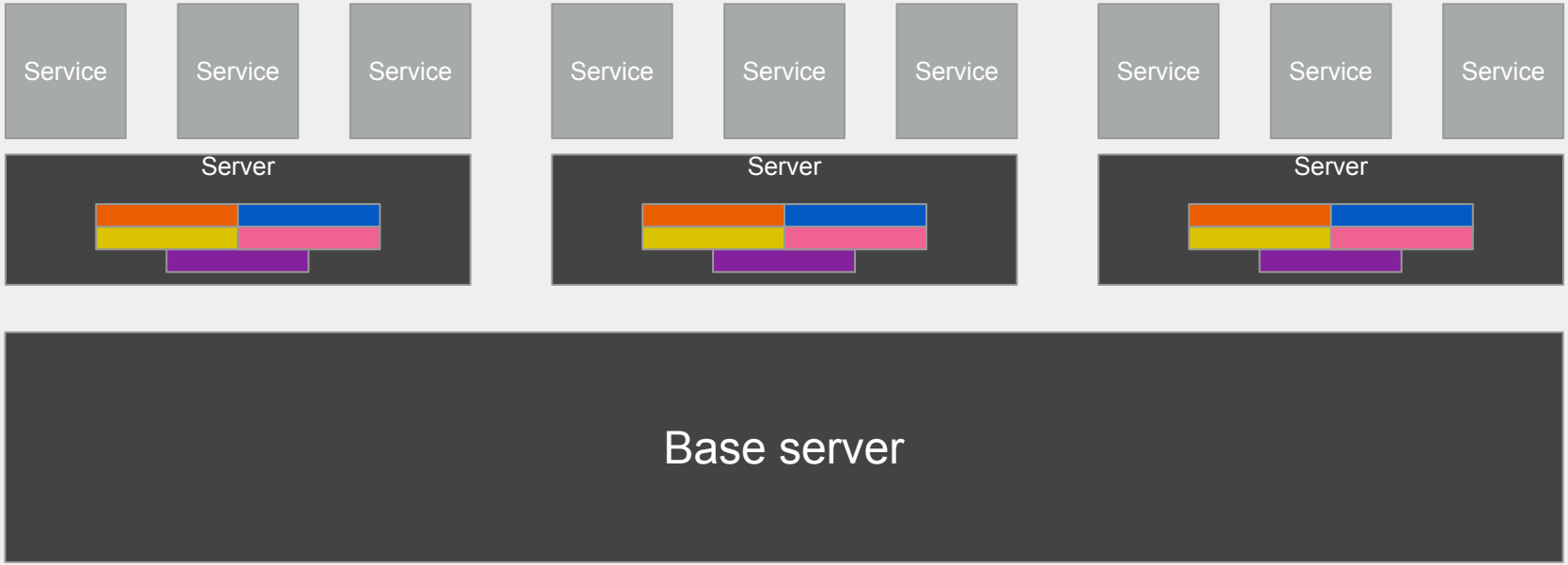
Production services



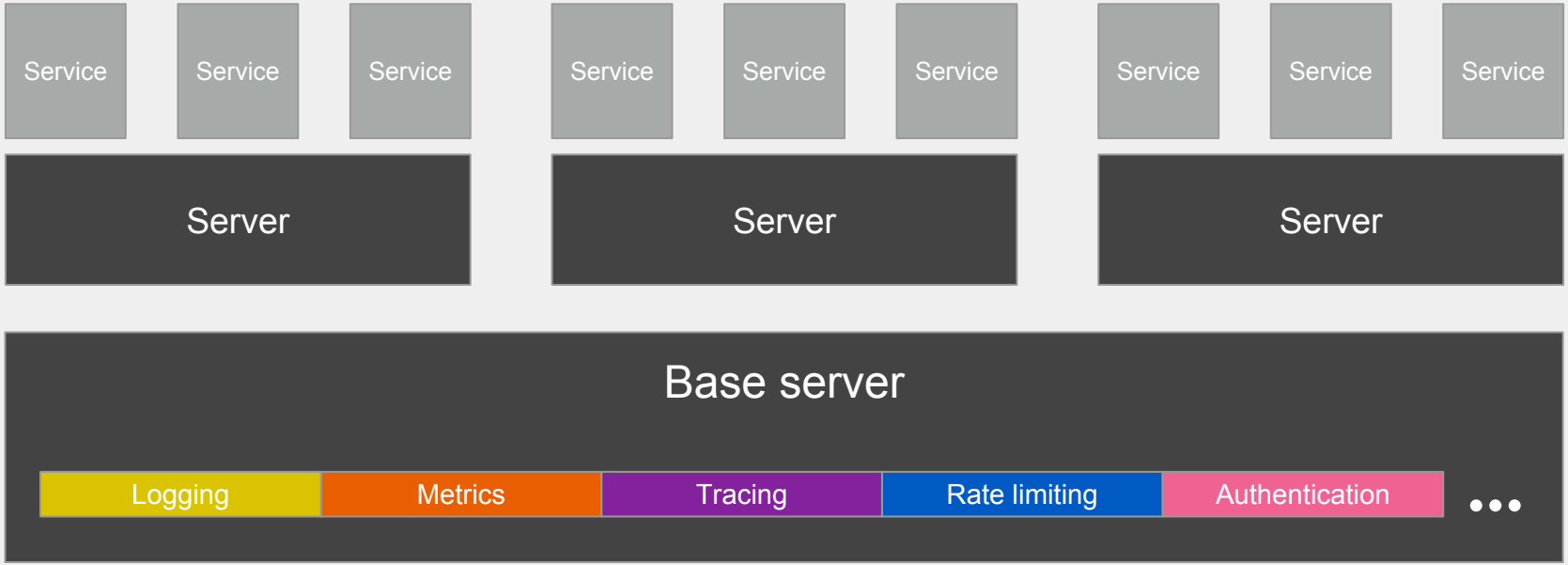
Production services



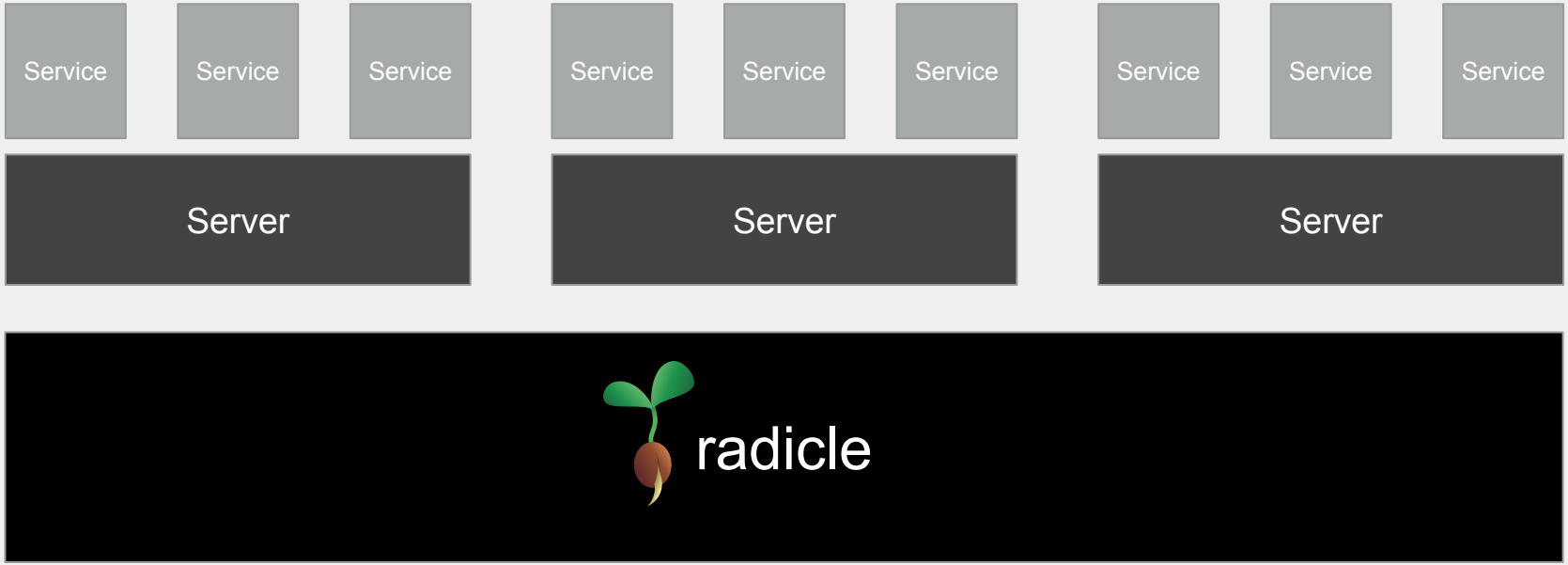
Production services



Production services



Production services



Benefits of centralized and unified common functionality:

- Guarantees basic operability

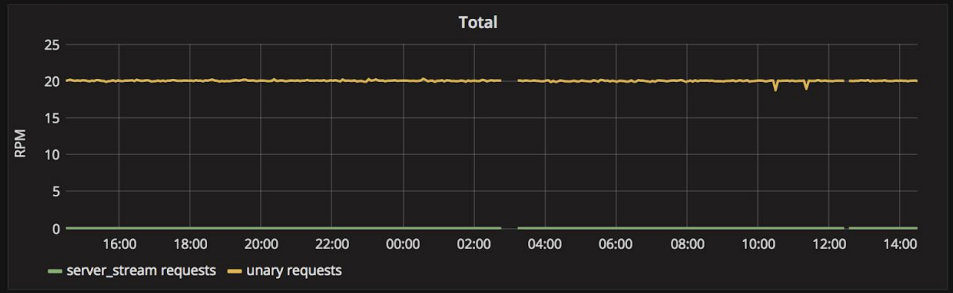
@KeeleyErhardt

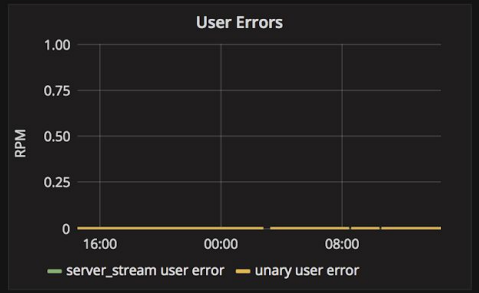
Generic gRPC Server

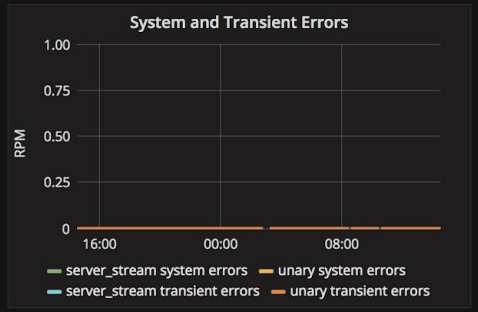
Zoom Out | Last 24 hours UTC

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

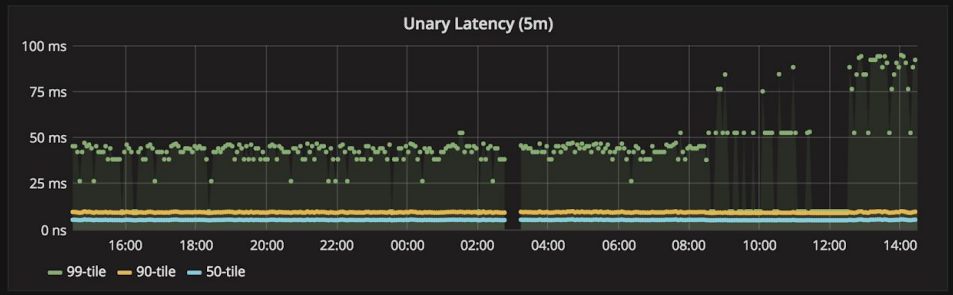
Requests (All)

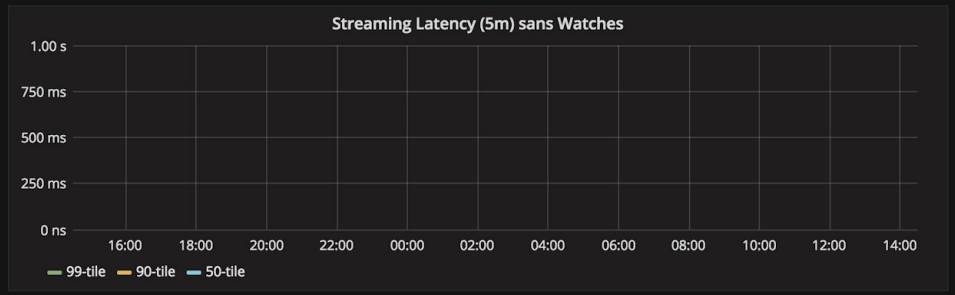






Latencies (All)





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:

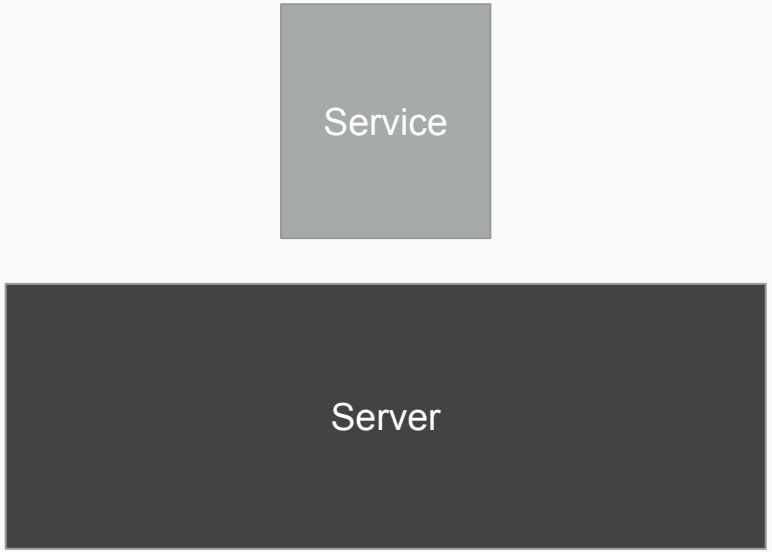
- Guarantees basic operability
- Standardizes services
- Promotes reusable service infrastructure
- Reduces development time

Benefits of centralized and unified common functionality:

- Guarantees basic operability
- Standardizes services
- Promotes reusable service infrastructure
- Reduces development time
- Enables diverse development

Greeter service

- ~~Naive implementation~~
- Production ready



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

github.com/keelerh/demo-radicle

```
service Greeter {
    rpc SayHello (HelloRequest) returns (HelloResponse) {
        option (google.api.http) = {
            post: "/greeter/hello"
            body: "*"
        };
    }
}
```

github.com/keelerh/demo-radicle

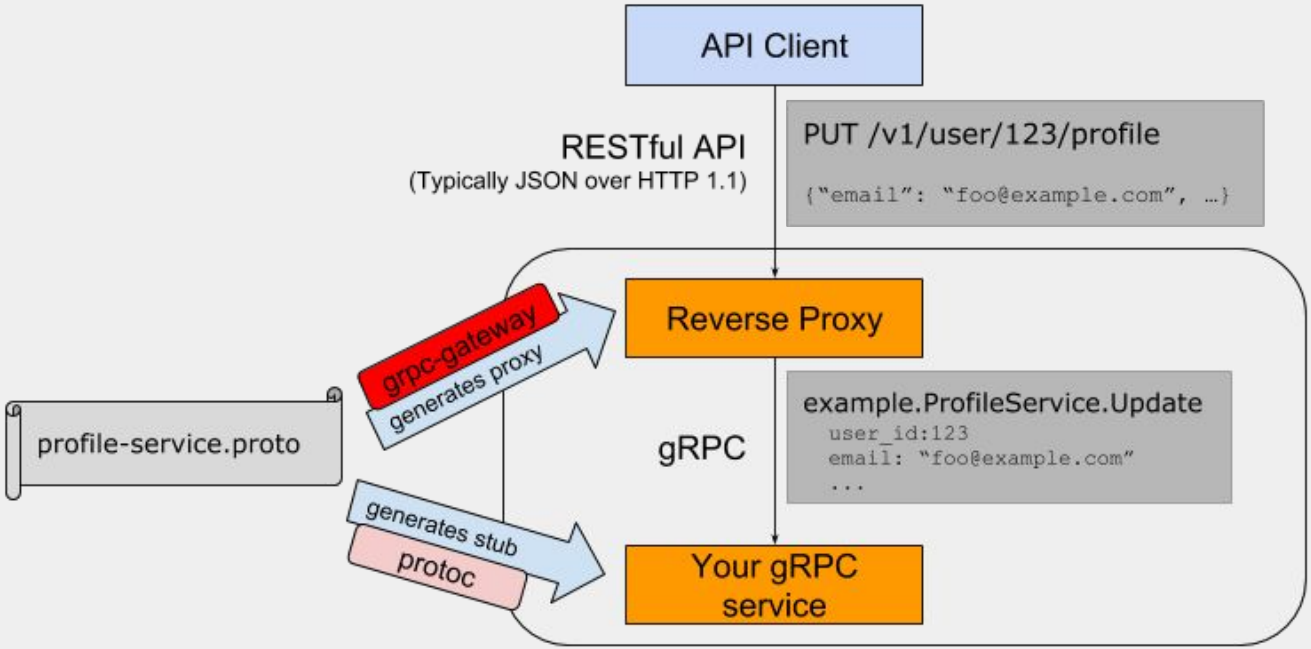


Image Source:
https://github.com/grpc-ecosystem/grpc-gateway

```go
func runHTTPServer() error {
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()

    mux := runtime.NewServeMux()
    if err := pb.RegisterGreeterHandlerFromEndpoint(
        ctx,
        mux,
        "localhost:8081",
        []grpc.DialOption{grpc.WithInsecure()},
    ); err != nil {
        return err
    }
    return http.ListenAndServe(":8080", mux)
}
```

github.com/keelerh/demo-radicle

```go
func runGRPCServer(lis net.Listener) {
    srv := grpc.NewServer()
    svc := &helloworld.GreeterService{}
    pb.RegisterGreeterServer(srv, svc)

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

github.com/keelerh/demo-radicle

```go
func main() {
    srv := radicle.NewServer(
        "greeter",
        radicle.Grpc(8081),
        radicle.Http(8080),
    )

    if err := srv.ListenAndServe(); err != nil {
        log.Fatalf("failed to start server: %v", err)
    }
}
```

github.com/keelerh/demo-radicle

```go
func runGRPCServer(lis net.Listener) {
    srv := grpc.NewServer()
    svc := &helloworld.GreeterService{}
    pb.RegisterGreeterServer(srv, svc)
    if err := srv.Serve(lis); err != nil {
        log.Fatalf("failed to serve: %v", err)
    }
}

func runHTTPServer() error {
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()
    mux := runtime.NewServeMux()
    if err := pb.RegisterGreeterHandlerFromEndpoint(
        ctx,
        mux,
        "localhost:8081",
        []grpc.DialOption{grpc.WithInsecure()},
    ); err != nil {
        return err
    }
    return http.ListenAndServe(":8080", mux)
}

func main() {
    grpcListener, err := net.Listen("tcp", ":8081")
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }
    go runGRPCServer(grpcListener)
    if err := runHTTPServer(); err != nil {
        log.Fatalf("failed to start HTTP server: %v", err)
    }
}
```



```go
func main() {
    srv := radicle.NewServer(
        "greeter",
        radicle.Grpc(8081),
        radicle.Http(8080),
    )

    if err := srv.ListenAndServe(); err != nil {
        log.Fatalf("failed to start server: %v", err)
    }
}
```

github.com/keelerh/demo-radicle

```go
func main() {
    srv := radicle.NewServer(
        radicle.Grpc(8081),
        radicle.Http(8080),
    )

    if err := srv.ListenAndServe(); err != nil {
        log.Fatalf("failed to start server: %v", err)
    }
}
```

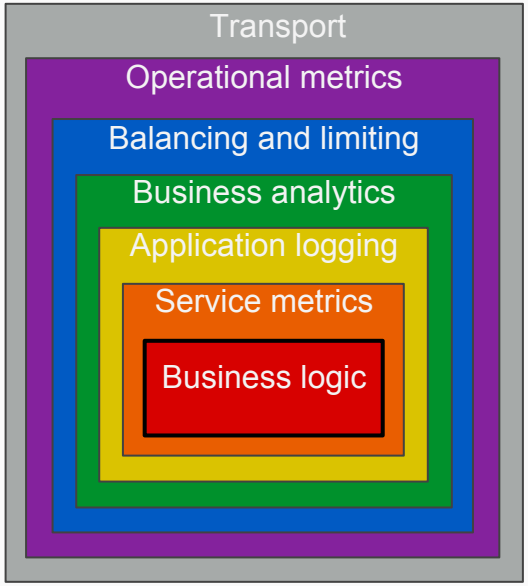
github.com/keelerh/demo-radicle

```go
func main() {
    srv := radicle.NewServer(
        radicle.Grpc(8081),
        radicle.Http(8080),
    )

    if err := srv.ListenAndServe(); err != nil {
        log.Fatalf("failed to start server: %v", err)
    }
}
```

github.com/keelerh/demo-radicle

“Onion Model”

- Enforces strict *separation of concerns* using the middleware (or decorator) pattern



Go + Microservices = Go Kit - Peter Bourgon, Cloud Native Con Europe 2017

gRPC interceptors

- Equivalent to HTTP middlewares
- Server interceptors intercept the execution of a RPC on the server

```
// UnaryHandler defines the handler invoked by
// UnaryServerInterceptor to complete the normal
// execution of a unary RPC.
type UnaryHandler func(ctx context.Context, req interface{})
(interface{}, error)

// UnaryServerInterceptor provides a hook to intercept the
// execution of a unary RPC on the server. It is the
// responsibility of the interceptor to invoke handler to
// complete the RPC.
type UnaryServerInterceptor func(
    ctx context.Context,
    req interface{},
    info *UnaryServerInfo,
    handler UnaryHandler) (resp interface{}, err error)
```

```
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

github.com/keelerh/demo-radicle

```
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

github.com/keelerh/demo-radicle

```
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

github.com/keelerh/demo-radicle

```
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

github.com/keelerh/demo-radicle

```
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

github.com/keelerh/demo-radicle

```go
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

github.com/keelerh/demo-radicle

```
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

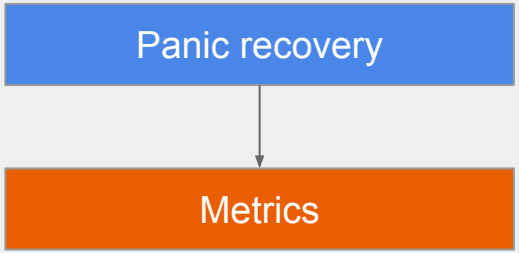
github.com/keelerh/demo-radicle

```
func UnaryServerInterceptor() grpc.UnaryServerInterceptor {
    return func(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (_ interface{}, err error) {
        defer func() {
            if r := recover(); r != nil {
                err = status.Errorf(codes.Internal, "%s", r)
            }
        }()
        return handler(ctx, req)
    }
}
```

github.com/keelerh/demo-radicle

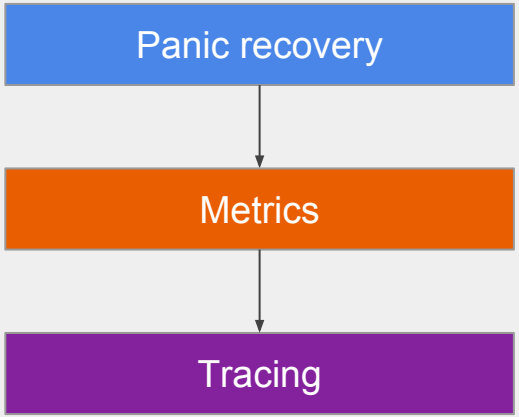
Panic recovery

```
recovery.UnaryServerInterceptor()
```




```
recovery.UnaryServerInterceptor()
```

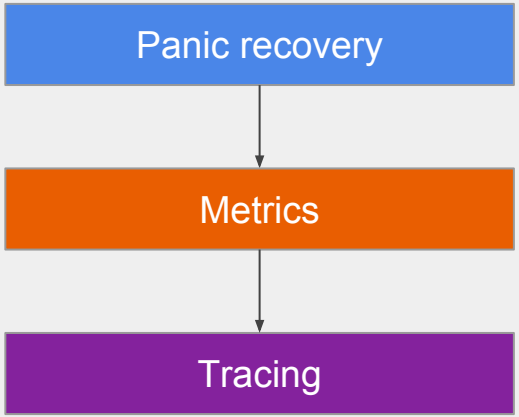
```
grpc_prometheus.UnaryServerInterceptor()
```




```
recovery.UnaryServerInterceptor()
```

```
grpc_prometheus.UnaryServerInterceptor()
```

```
grpc_opentracing.UnaryServerInterceptor()
```




```
grpc_middleware.WithUnaryServerChain(
    recovery.UnaryServerInterceptor(),
    grpc_prometheus.UnaryServerInterceptor(),
    grpc_opentracing.UnaryServerInterceptor(),
)
```

```go
func main() {
    interceptors := grpc_middleware.WithUnaryServerChain(
        recovery.UnaryServerInterceptor(),
        grpc_prometheus.UnaryServerInterceptor(),
        grpc_opentracing.UnaryServerInterceptor(),
    )
    srv := radicle.NewServer(
        radicle.Grpc(8081),
        radicle.GrpcServerOption(interceptors),
    )

    if err := srv.ListenAndServe(); err != nil {
        log.Fatalf("failed to start server: %v", err)
    }
}
```

github.com/keelerh/demo-radicle

```go
func main() {
    interceptors := grpc_middleware.WithUnaryServerChain(
        recovery.UnaryServerInterceptor(),
        grpc_prometheus.UnaryServerInterceptor(),
        grpc_opentracing.UnaryServerInterceptor(),
    )
    srv := radicle.NewServer(
        radicle.Grpc(8081),
        radicle.GrpcServerOption(interceptors),
    )

    if err := srv.ListenAndServe(); err != nil {
        log.Fatalf("failed to start server: %v", err)
    }
}
```

github.com/keelerh/demo-radicle

```go
func main() {
    interceptors := grpc_middleware.WithUnaryServerChain(
        recovery.UnaryServerInterceptor(),
        grpc_prometheus.UnaryServerInterceptor(),
        grpc_opentracing.UnaryServerInterceptor(),
    )
    srv := radicle.NewServer(
        "greeter",
        radicle.Grpc(8081),
        radicle.GrpcServerOption(interceptors),
    )

    if err := srv.ListenAndServe(); err != nil {
        log.Fatalf("failed to start server: %v", err)
    }
}
```

github.com/keelerh/demo-radicle

radicle functionalities

- Server configuration for multiple transports
- Pluggable middlewares (e.g. for metrics, logging, tracing)
- Debug endpoints

Photo by John Carlisle on Unsplash

HOME

STATUS

BUILD INFO

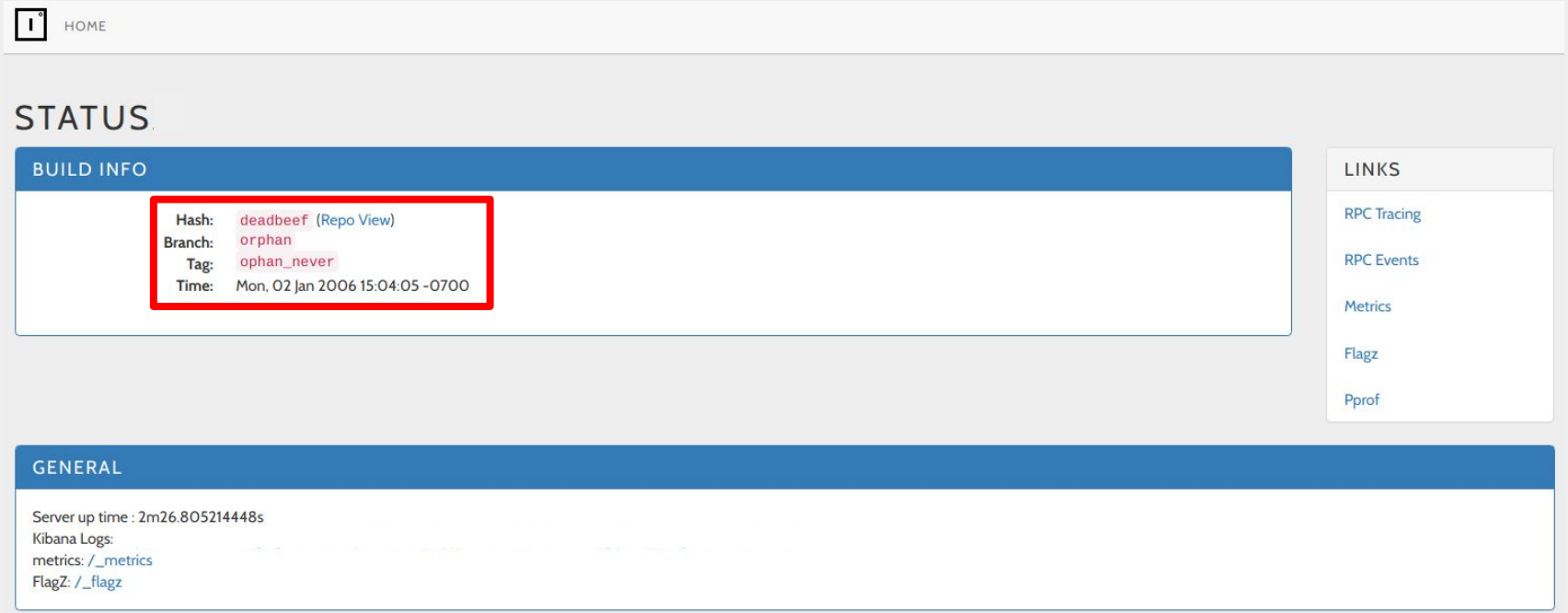
Hash:	`deadbeef` (Repo View)
Branch:	`orphan`
Tag:	`ophan_never`
Time:	Mon, 02 Jan 2006 15:04:05 -0700

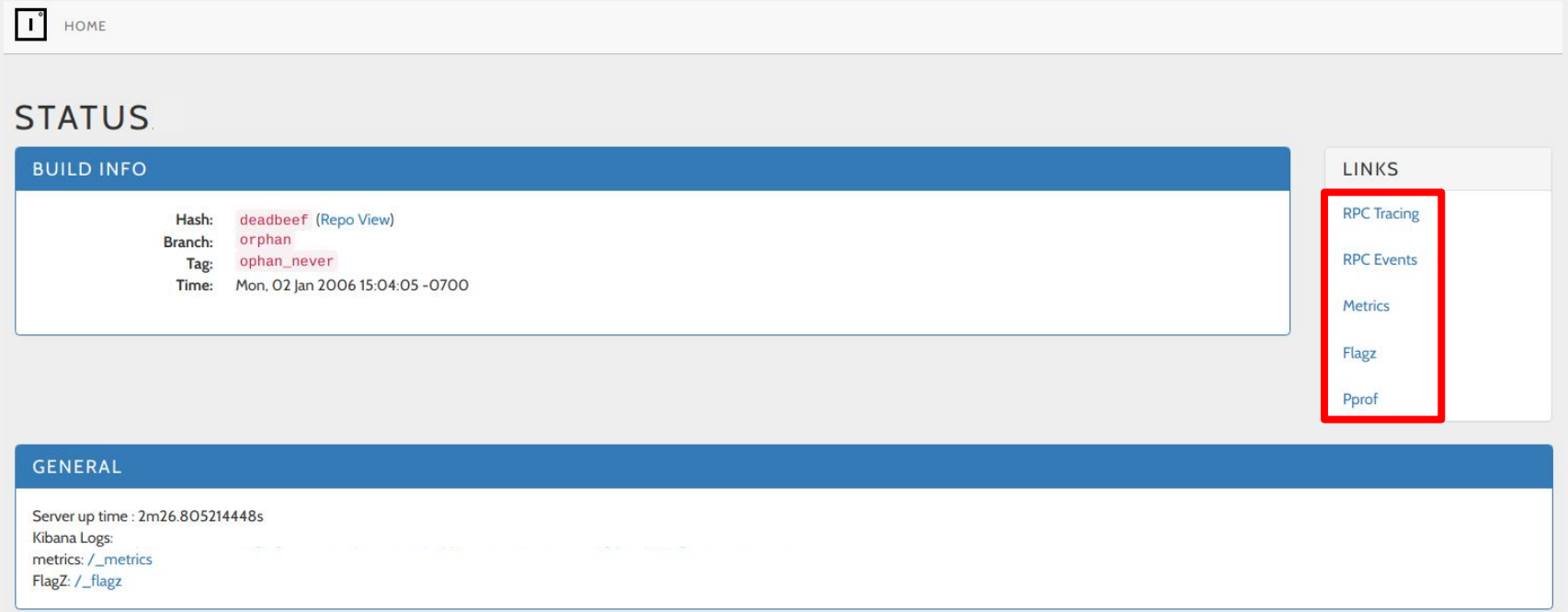
LINKS

- RPC Tracing
- RPC Events
- Metrics
- Flagz
- Pprof

GENERAL

Server up time : 2m26.805214448s
Kibana Logs:
metrics: /_metrics
FlagZ: /_flagz


HOME
STATUS
BUILD INFO
Hash: deadbeef (Repo View)
Branch: orphan
Tag: ophan_never
Time: Mon, 02 Jan 2006 15:04:05 -0700
LINKS
RPC Tracing
RPC Events
Metrics
Flagz
Pprof
GENERAL
Server up time : 2m26.805214448s
Kibana Logs:
metrics: /_metrics
FlagZ: /_flagz


HOME
STATUS
BUILD INFO
Hash: deadbeef (Repo View)
Branch: orphan
Tag: ophan_never
Time: Mon, 02 Jan 2006 15:04:05 -0700
LINKS
RPC Tracing
RPC Events
Metrics
Flagz
Pprof
GENERAL
Server up time : 2m26.805214448s
Kibana Logs:
metrics: /_metrics
FlagZ: /_flagz

@KeeleyErhardt

/debug/requests

grpc.Recv.ServerStatus	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
grpc.Sent.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
grpc.Sent.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
grpc.Sent.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
grpc.Sent.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./_status:	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.GET./debug/requests	[2 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.HEAD.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.HEAD./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.POST.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
http.Recv.POST./	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]

Family: grpc.Recv.ServerStatus

[Normal/Summary] [Normal/Expanded] [Traced/Summary] [Traced/Expanded]

Completed Requests

When	Elapsed (s)	
2018/06/18 22:43:09.250406	0.001178	/base.ServerStatus/HealthCheck
2018/06/18 22:43:06.250059	0.000469	/base.ServerStatus/HealthCheck
2018/06/18 22:43:03.249862	0.000607	/base.ServerStatus/HealthCheck
2018/06/18 22:43:00.250063	0.000498	/base.ServerStatus/HealthCheck
2018/06/18 22:42:57.295232	0.000620	/base.ServerStatus/HealthCheck
2018/06/18 22:42:54.249918	0.000469	/base.ServerStatus/HealthCheck
2018/06/18 22:42:51.249968	0.000484	/base.ServerStatus/HealthCheck
2018/06/18 22:42:48.249874	0.000556	/base.ServerStatus/HealthCheck
2018/06/18 22:42:45.250263	0.000539	/base.ServerStatus/HealthCheck
2018/06/18 22:42:42.250193	0.000703	/base.ServerStatus/HealthCheck

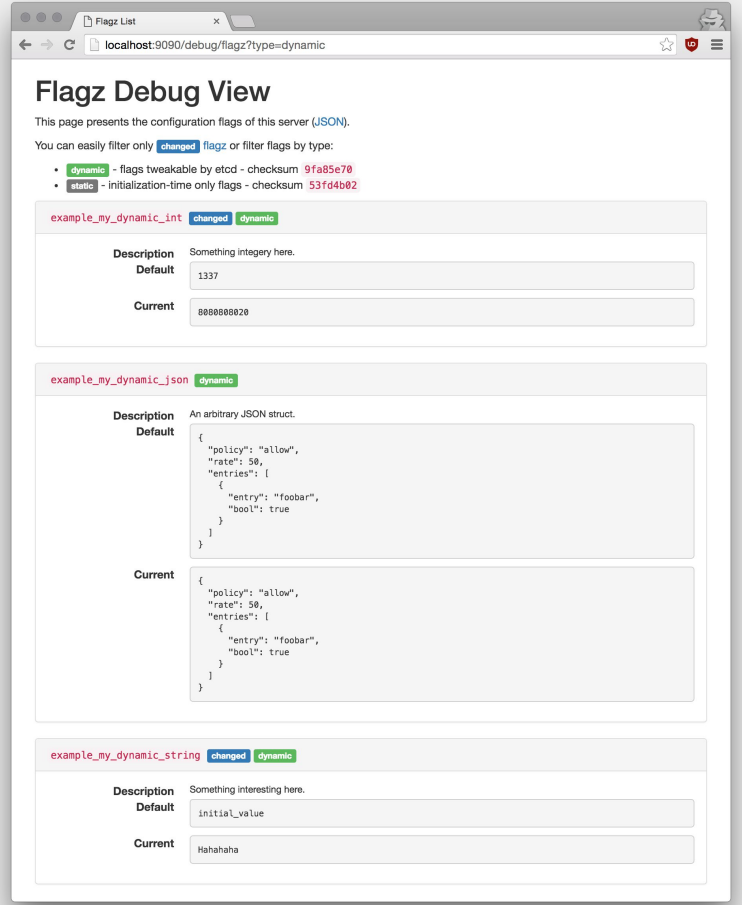


Image Source: https://github.com/mwitkow/go-flagz

```
goroutine profile: total 18
1 @ 0x40f412 0x4411c7 0x6af162 0x45ec61
#       0x4411c6        os/signal.signal_recv+0x156     /usr/local/go/src/runtime/sigqueue.go:116
#       0x6af161        os/signal.loop+0x21             /usr/local/go/src/os/signal/signal_unix.go:22

1 @ 0x42dd6a 0x42865b 0x427db9 0x556f88 0x556ff4 0x558831 0x5694b0 0x724b6c 0x7251cc 0x5f8e14 0x5f8f78 0x665c5b 0x666484 0x77946f 0x76f436 0x45ec61
#       0x427db8        net.runtime_pollWait+0x58                                       /usr/local/go/src/runtime/netpoll.go:160
#       0x556f87        net.(*pollDesc).wait+0x37                                       /usr/local/go/src/net/fd_poll_runtime.go:73
#       0x556ff3        net.(*pollDesc).waitRead+0x33                                   /usr/local/go/src/net/fd_poll_runtime.go:78
#       0x558830        net.(*netFD).Read+0x1a0                                         /usr/local/go/src/net/fd_unix.go:243
#       0x5694af        net.(*conn).Read+0x6f                                           /usr/local/go/src/net/net.go:173
#       0x724b6b        bufio.(*Reader).fill+0x10b                                      /usr/local/go/src/bufio/bufio.go:97
#       0x7251cb        bufio.(*Reader).Read+0x1bb                                      /usr/local/go/src/bufio/bufio.go:209
#       0x5f8e13        io.ReadAtLeast+0xa3                                             /usr/local/go/src/io/io.go:307
#       0x5f8f77        io.ReadFull+0x57                                                /usr/local/go/src/io/io.go:325
#       0x665c5a        golang.org/x/net/http2.readFrameHeader+0x7a
#       0x666483        golang.org/x/net/http2.(*Framer).ReadFrame+0xa3
#       0x77946e        google.golang.org/grpc/transport.(*framer).readFrame+0x2e
#       0x76f435        google.golang.org/grpc/transport.(*http2Client).reader+0xb5

1 @ 0x42dd6a 0x42865b 0x427db9 0x556f88 0x556ff4 0x558831 0x5694b0 0x724b6c 0x7251cc 0x5f8e14 0x5f8f78 0x665c5b 0x666484 0x77946f 0x772093 0x538830 0x53868a 0x538270 0x45ec61
#       0x427db8        net.runtime_pollWait+0x58                                       /usr/local/go/src/runtime/netpoll.go:160
#       0x556f87        net.(*pollDesc).wait+0x37                                       /usr/local/go/src/net/fd_poll_runtime.go:73
#       0x556ff3        net.(*pollDesc).waitRead+0x33                                   /usr/local/go/src/net/fd_poll_runtime.go:78
#       0x558830        net.(*netFD).Read+0x1a0                                         /usr/local/go/src/net/fd_unix.go:243
#       0x5694af        net.(*conn).Read+0x6f                                           /usr/local/go/src/net/net.go:173
#       0x724b6b        bufio.(*Reader).fill+0x10b                                      /usr/local/go/src/bufio/bufio.go:97
#       0x7251cb        bufio.(*Reader).Read+0x1bb                                      /usr/local/go/src/bufio/bufio.go:209
#       0x5f8e13        io.ReadAtLeast+0xa3                                             /usr/local/go/src/io/io.go:307
#       0x5f8f77        io.ReadFull+0x57                                                /usr/local/go/src/io/io.go:325
#       0x665c5a        golang.org/x/net/http2.readFrameHeader+0x7a
#       0x666483        golang.org/x/net/http2.(*Framer).ReadFrame+0xa3
#       0x77946e        google.golang.org/grpc/transport.(*framer).readFrame+0x2e
#       0x772092        google.golang.org/grpc/transport.(*http2Server).HandleStreams+0x202
#       0x53882f        google.golang.org/grpc.(*Server).serveStreams+0x15f
#       0x538689        google.golang.org/grpc.(*Server).serveNewHTTP2Transport+0x3d9
#       0x53826f        google.golang.org/grpc.(*Server).handleRawConn+0x46f
```

Key takeaways

What should I get out of this talk?

- Production services must be stable, reliable, and debuggable
- Don’t duplicate common functionality across multiple services
- radicle provides a configurable, ***debuggable, production-ready*** server

Liked what you heard?

say hi on `improbable-eng.slack.com`

...or come join us!

`improbable.hk/careers` `improbable.io/careers`

`github.com/improbable-eng`