A story about debugging of one Go application

Speaker

Yury Kulazhenkov

Golang, Python developer,

Plugins team, Dell EMC

Container Storage Interface (CSI) in 5 minutes

1. Containers
2. Kubernetes
3. Storage array

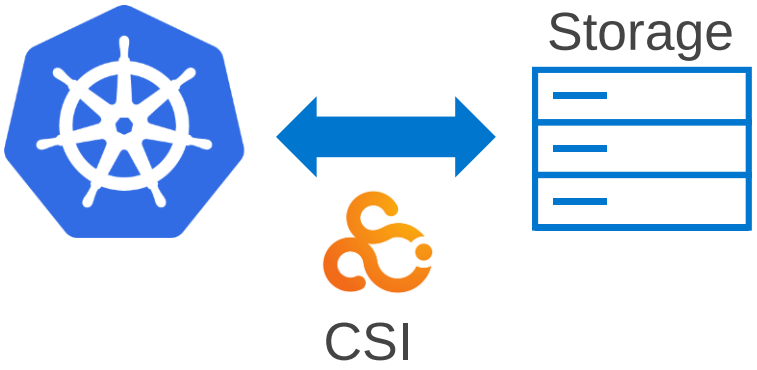
Storage array

Storage array is a integrated hardware-software solution for organizing reliable storage of information resources and providing guaranteed access to them.

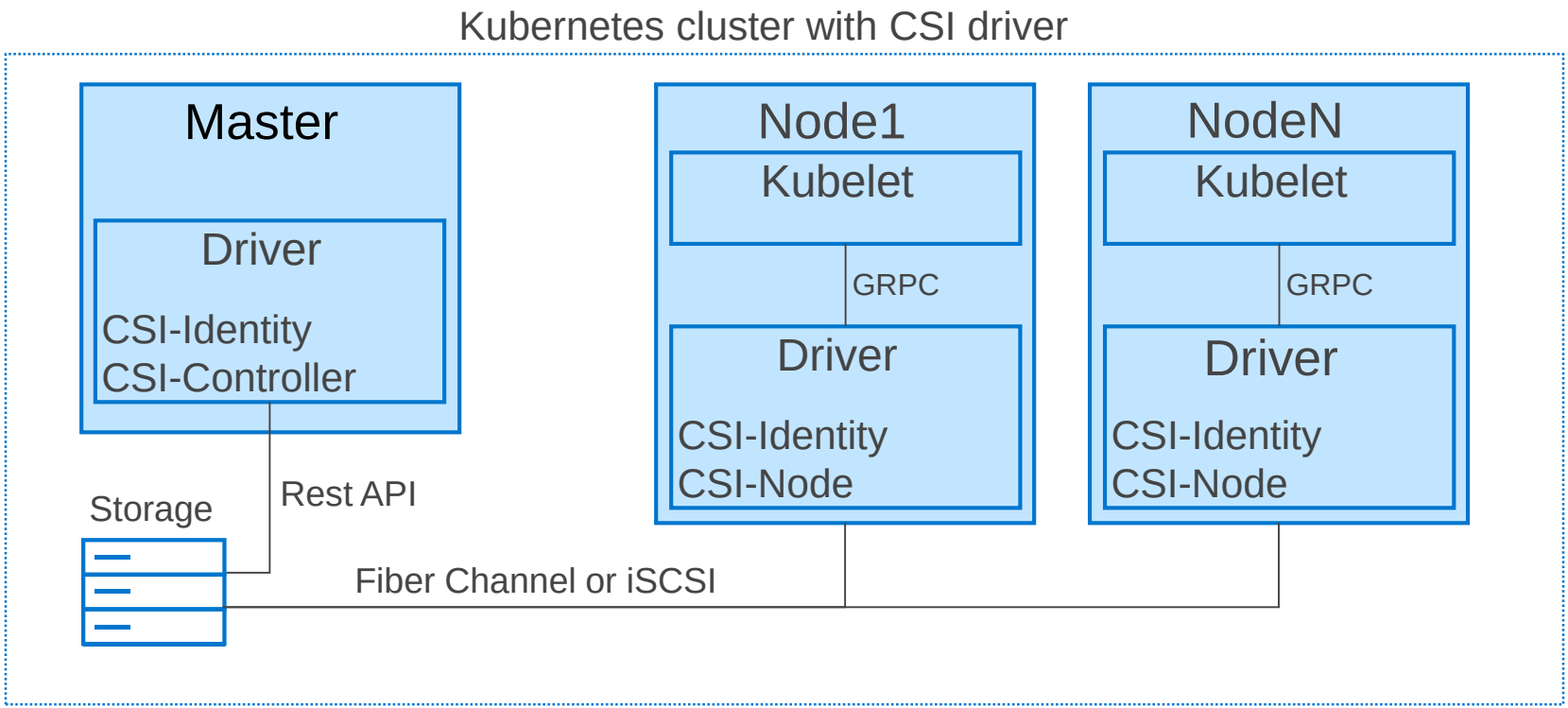
Protocols:

- iSCSI
- Fiber Channel

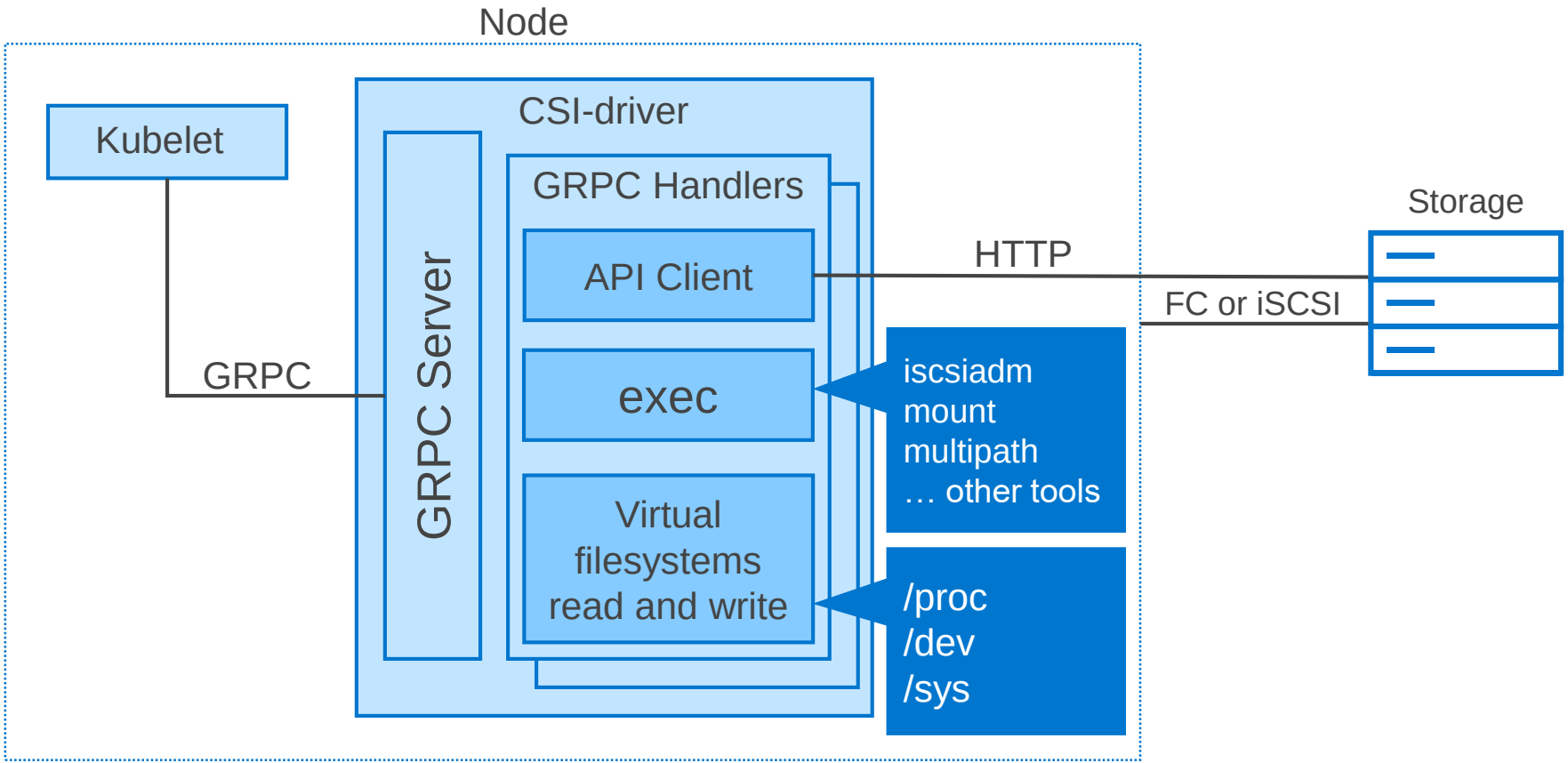
Container Storage Interface (CSI)



CSI driver - Kubernetes



CSI driver – Application



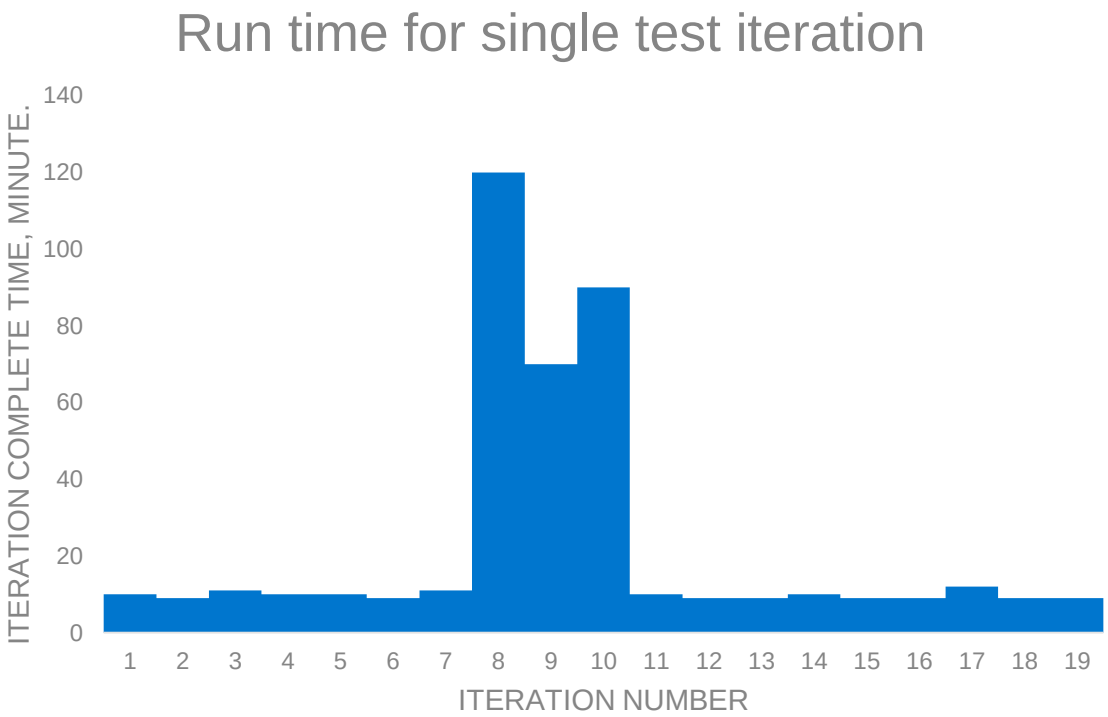
CSI driver – properties

1. Most functions rely on the external state and have side effects
2. Driver code is highly infrastructure dependent
3. Many parallel calls to the same code parts

CSI driver – Testing

1. Unit tests
2. Integration tests
3. End-to-end tests

CSI driver – End-to-end testing



Investigation begins

1. Storage array
2. Network
3. Kubernetes cluster
4. CSI driver

Problem: CSI driver is slowing down

Questions:

1. Which part of the application works slow?
2. Why?

Tracing – integration

```
import (
    ...
    pb "github.com/ydotk/xnettrace-example/foo"
    "google.golang.org/grpc"
    ...
)

type server struct {
    pb.UnimplementedFooServiceServer
}

func main() {
    lis, err := net.Listen("tcp", port)
    s := grpc.NewServer())
    pb.RegisterFooServiceServer(s, &server{})
    err = s.Serve(lis)
    ...
}
```

https://godoc.org/golang.org/x/net/trace

Tracing – integration, cont.

```go
// GRPC handler
func (s *server) GetFoo(ctx context.Context, in *pb.Foo) (*pb.Bar, error) {
    f1(ctx)
    f2(ctx)
    return &pb.Bar{Name: "Bar"}, nil
}

// any function 1
func f1(ctx context.Context) {
    ...
}

// any function 2
func f2(ctx context.Context) {
    ...
}
```

Tracing – integration, cont.

```
type tracingInterceptor struct {
    ...
}

func (ti *tracingInterceptor) handler(
       ctx context.Context,
       req interface{},
       info *grpc.UnaryServerInfo,
       handler grpc.UnaryHandler) (interface{}, error) {

    tr := trace.New(info.FullMethod, ti.generateReqID())
    defer tr.Finish()
    ctx = trace.NewContext(ctx, tr)
    return handler(ctx, req)
}

func (ti *tracingInterceptor) generateReqID() string {
    ...
}
```

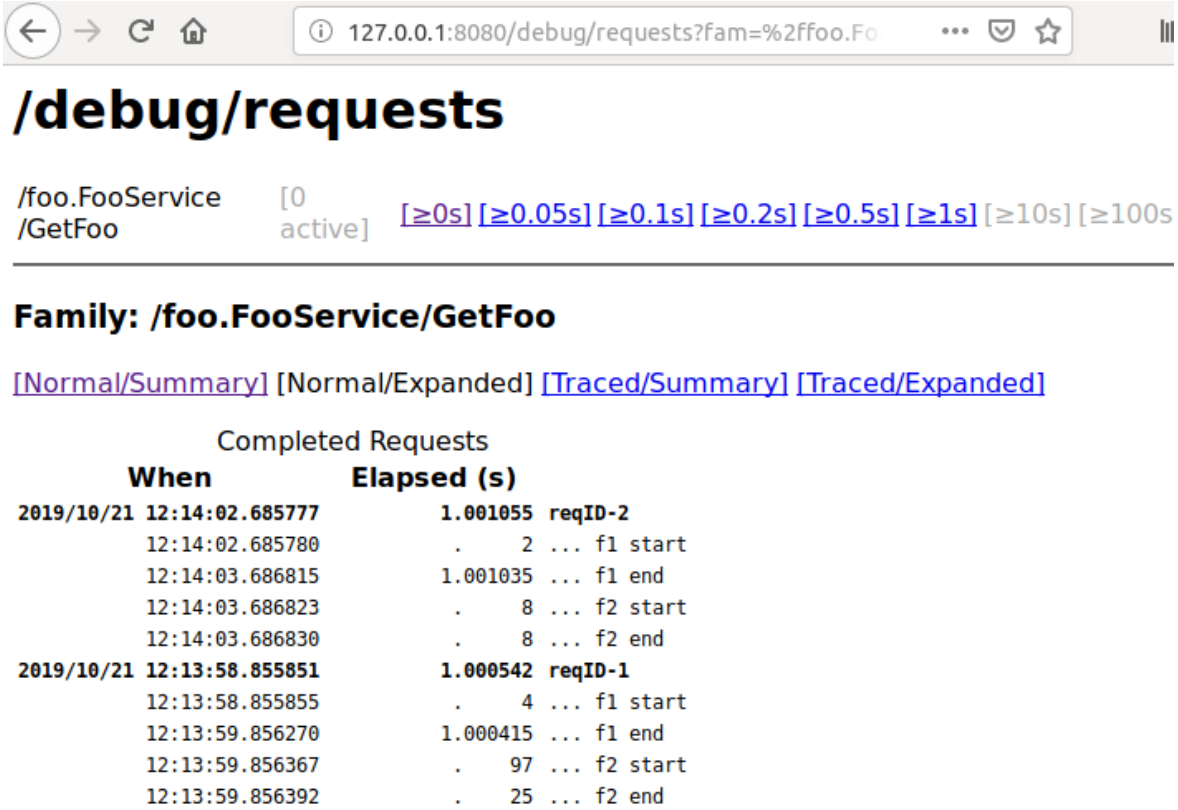
Tracing – integration, cont.

```
func main() {
    ...
    // register interceptor
    s := grpc.NewServer(
        grpc.UnaryInterceptor(interceptor.handler))
    }
    // start http server in separate goroutine
    go func() {
        http.ListenAndServe("127.0.0.1:8080", nil)
    }()
    ...
}
```

```
// add tracing support to the function
func f1(ctx context.Context) {
    if tr, ok := trace.FromContext(ctx); ok {
        tr.LazyPrintf("f1 start")
        defer tr.LazyPrintf("f1 end")
    }
    time.Sleep(time.Second)
    ...
}
```

Full example
https://github.com/ydotk/xnettrace-example

Tracing – analysis

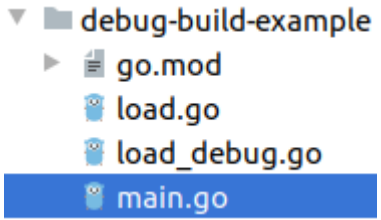


Metrics – integration

```
import "github.com/grpc-ecosystem/go-grpc-prometheus"
...
    s := grpc.NewServer(
        grpc.StreamInterceptor(grpc_prometheus.StreamServerInterceptor),
        grpc.UnaryInterceptor(grpc_prometheus.UnaryServerInterceptor),
    )
    grpc_prometheus.Register(s)
    http.Handle("/metrics", promhttp.Handler())
...
```

https://github.com/grpc-ecosystem/go-grpc-prometheus

Debug build – main.go

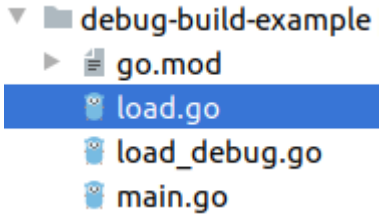



```go
package main

import (
    log "github.com/sirupsen/logrus"
)

func main() {
    loadApp()
}
```

Debug build – load.go



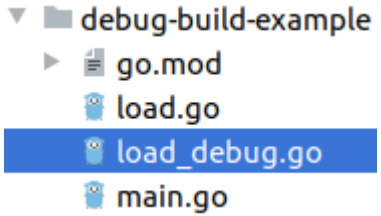

```go
// +build !debug

package main

import (
    log "github.com/sirupsen/logrus"
)

func loadApp() {
    fmt.Println("start release build")
}
```

Debug build – load_debug.go




```go
// +build debug

package main

import (
    log "github.com/sirupsen/logrus"
)

func loadApp() {
    fmt.Println("start debug build")
}
```

Debug build – run

Normal build

$ go run .

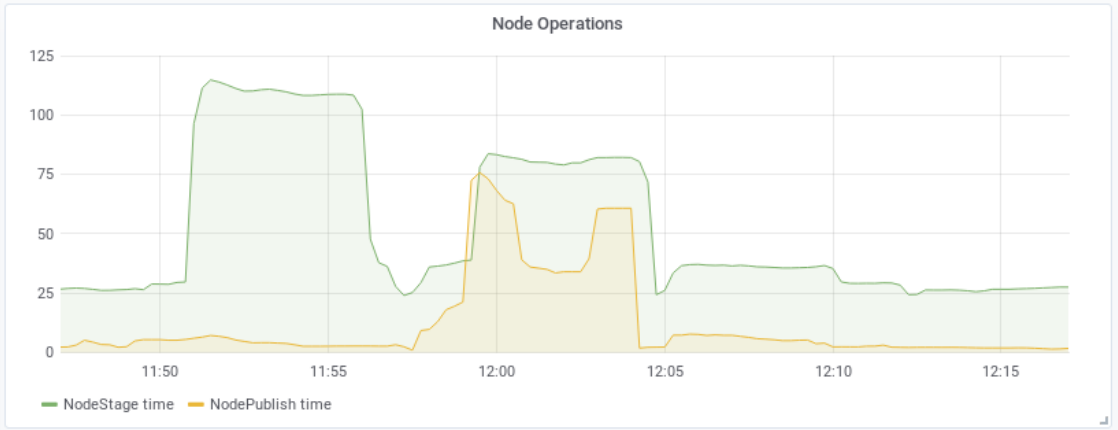
```
start release build
```

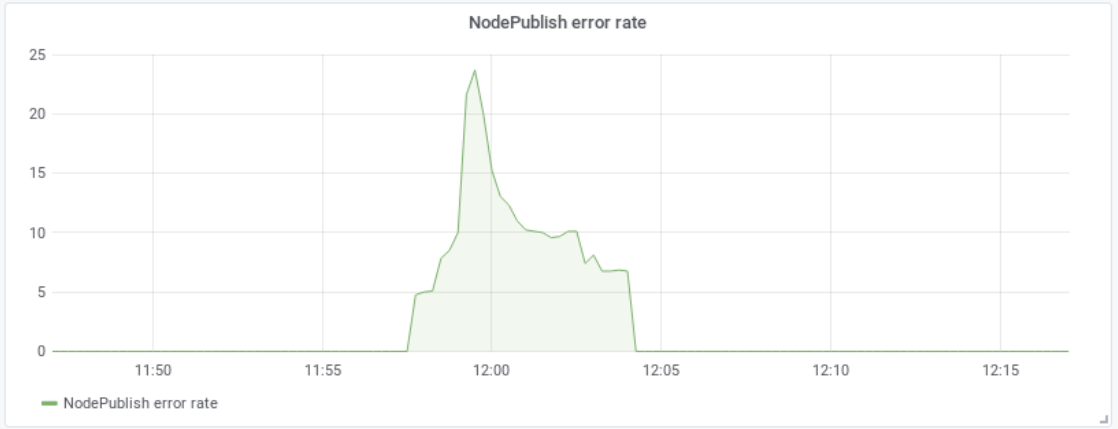
Debug build

$ go run -tags "debug" .

```
start debug build
```

Metrics – analysis





Logging – analysis

ERRO[2019-05-10 15:50:08] mount error: exit status 1

Logging – our rules for debug log

1. Log messages should contain all necessary information for its correct interpretation.

```
f := logrus.Fields{"RequestID": reqID, "Portal": portal, "Target": target}
logrus.WithFields(f).Debug("check iSCSI connection")
```

2. Add module and function name to all debug log messages

```
logrus.SetReportCaller(true)
```

3. Always write down stdout, stderr and exit code when you call external tools with os/exec

4. Always write down request and response data when you interact with REST API

Change logging level in runtime – example

```
var debug bool

func main() {
    sigCh := make(chan os.Signal, 1)
    signal.Notify(sigCh, syscall.SIGUSR1)
    go enableDebug(sigCh)
    log.Infof("start loop, PID: %d",
              os.Getpid())
    for {
        time.Sleep(time.Second)
        log.Debug("debug message")
    }
}
```

```
func enableDebug (ch chan os.Signal) {
    for {
        <- ch
        if debug {
            log.Info("disable debug")
            log.SetLevel(log.InfoLevel)
            debug = false
            continue
        }
        log.Info("enable debug")
        log.SetLevel(log.DebugLevel)
        debug = true
    }
}
```

Change logging level in runtime – run

$ go run .
 INFO[0000] Info message

INFO[0000] start loop, PID: 104823
INFO[0004] enable debug
DEBU[0005] debug message
DEBU[0006] debug message
DEBU[0007] debug message
DEBU[0008] debug message
INFO[0008] disable debug

turn on debug
$ kill -s USR1 104823

turn off debug
$ kill -s USR1 104823

Logging – EFK

RequestID	ID	WWN	msg
			t found
csi.requestid-396	ca61117e-69b0-475d-93ed-8b0f7074a8a0	68ccf098008607f3f9d0591485d68710	get scan mutex
csi.requestid-396	ca61117e-69b0-475d-93ed-8b0f7074a8a0	68ccf098008607f3f9d0591485d68710	run SCSI rescan
csi.requestid-396	ca61117e-69b0-475d-93ed-8b0f7074a8a0	68ccf098008607f3f9d0591485d68710	trying resolve scsi address for device
csi.requestid-396	ca61117e-69b0-475d-93ed-8b0f7074a8a0	68ccf098008607f3f9d0591485d68710	rescan scsi
csi.requestid-396	ca61117e-69b0-475d-93ed-8b0f7074a8a0	68ccf098008607f3f9d0591485d68710	waiting for device in /dev/disk/...

Problem Found


```
{
  "RequestID": "csi.request-id-1078",
  "command": "mount foo",
  "exitCode": 1,
  "level": "error",
  "mountPoint": "foo",
  "msg": "mount error",
  "stderr": "mount: foo: No such file or directory\n",
  "stdout": "",
  "time": "2019-06-16T10:52:35+03:00"
}
```

Results

1. Significantly reduced the time required to analyze problems.
2. We have all the required data to detect bottlenecks in our application.
3. We have access to historical data and can compare performance metrics between different versions of our application.
4. Team communicate more efficiently. No need to send log files via email/chat.

Integrated Solutions

https://www.jaegertracing.io

https://zipkin.io/pages/community.html

Remote debugger

Include debugger in application image

```
$ kubectl exec -ti <pod name> -c <container> -- \
  dlv --listen=:40000 --headless=true --api-version=2 attach 1
```

```
$ kubectl port-forward <pod name> 40000:40000
```

Delve: https://github.com/go-delve/delve

Remote debugger, cont.

Use debugger from separate privileged Kubernetes pod

$ squashctl --debugger dlv --namespace kube-csi --pod csi-node --container driver

Squash: https://squash.solo.io

Questions?