# Structured Logging in Go

Ashim Ghosh

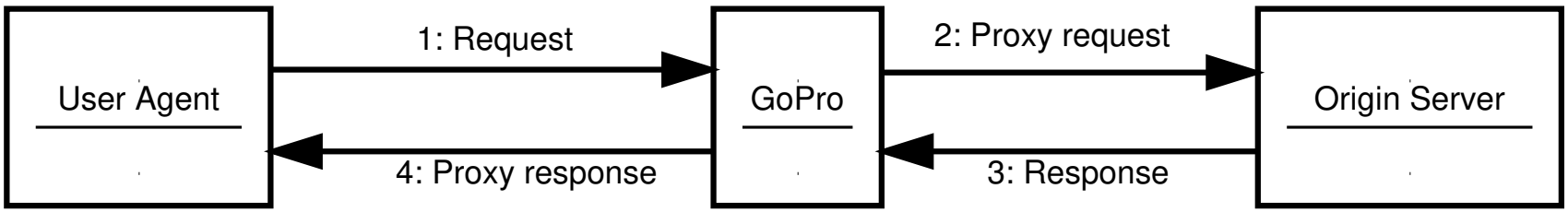
February 10, 2018

# About me

- Develop performant web services using C at PubMatic
- Amature Go developer
- Currently, developing my first, production-grade web service using Go

# Overview

# Scenario: Go Proxy (GoPro)




- Accept requests from *user agent*
- Proxy request to the *origin server*
- Proxy response from the origin server to the user agent

# Overview

# Prints

- Approach: Add plain print statements
- Documentation: [Online.] golang.org/pkg/fmt/
- Repository: Built into Go
- Demo: Branch print

# Prints

- Approach: Add plain print statements
- Documentation: [Online.] golang.org/pkg/fmt/
- Repository: Built into Go
- Demo: Branch print

- Merits: Simplicity
- Demerits: Level of abstraction Ease of parsing Log design Avoid missing log details Production readiness

# Overview

# Default Logger

- Approach: Use log package; simple, sensible, basic
- Fatal log-level considered bad?
- Documentation: [Online.] golang.org/pkg/log/
- Repository: Built into Go
- Demo: Branch pkg-log

# Default Logger

- Approach: Use log package; simple, sensible, basic
- Fatal log-level considered bad?
- Documentation: [Online.] golang.org/pkg/log/
- Repository: Built into Go
- Demo: Branch pkg-log

- Merits: Simplicity Production readiness
- Demerits: Level of abstraction (effectively only two levels) Ease of parsing Log design
- Avoid missing log details (support for adding some details to each log)

# Overview

# Levelled Logging

- Purpose: Ameliorate problem of level of abstraction
- Approach: Define levels of severity; add logs at appropriate level (after due consideration)
- Common Functions:
  - `{Debug, Info, Warn, Error, Fatal, Panic} X {, f, ln}`
  - `func SetLevel(int)`
- Too many log levels considered bad?
- glog package
- Documentation: [Online.] godoc.org/github.com/golang/glog
- Repository: [Online.] github.com/golang/glog

# Levelled Logging

- Purpose: Ameliorate problem of level of abstraction
- Approach: Define levels of severity; add logs at appropriate level (after due consideration)
- Common Functions:
  - `{Debug, Info, Warn, Error, Fatal, Panic} X {, f, ln}`
  - `func SetLevel(int)`
- Too many log levels considered bad?
- glog package
- Documentation: [Online.] godoc.org/github.com/golang/glog
- Repository: [Online.] github.com/golang/glog

- Merits: Level of abstraction | Production readiness
- Demerits: Ease of parsing | Log design | Avoid missing log details | Simplicity

# Overview

# Structured Logging

**structure** *(noun)*: a complex entity constructed of many parts.

- Thinking of logs as a structure; designing this structure and it's parts
- Machine readable structure.

# Example: Unstructured Log Line

## Sample log

```
req = &{Method:GET URL:/ Proto:HTTP/1.1 ProtoMajor:1 ProtoMinor:1
Header:map[User-Agent:[curl/7.47.0] Accept:[*/*]
X-Purl:[https://www.youtube.com/]] Body:{} GetBody:⟨nil⟩ ContentLength:0
TransferEncoding:[] Close:false Host:localhost:8080 Form:map[] PostForm:map[]
MultipartForm:⟨nil⟩ Trailer:map[] RemoteAddr:127.0.0.1:46312 RequestURI:/
TLS:⟨nil⟩ Cancel:⟨nil⟩ Response:⟨nil⟩ ctx:0xc42005e340}
```

## Considerations-

- Not machine readable?
- Not human readable?
- Who consumes logs?
- Too much detail? Yet, missing fields?
- Destroying the structure of a structure

# Design Structured Log Line: Universal fields

- Some field make sense for all applications

## Structured Log Line

```
{
  "level": "debug",
  "pkg": "main",
  "reqid": 1518232390,
  "time": "2018-02-10T08:43:10+05:30",
  "msg": "Received request from User Agent"
}
```

# Design Structured Log Line: Global Fields

- Application-specific fields
- Should be added to each log line

## Structured Log Line

```
{
  [...]

  "peer": "ua",
  "method": "GET",
  "url": "https://www.youtube.com/",
}
```

# Design Structured Log Line: Analysis-specific Fields

- Field required to answer questions specific to an analysis
- Example Analysis: Does the origin server upgrade to HTTP/2 when we make a HTTPS request?

## Structured Log Line

```
{
  [...]

  "httpver": "HTTP/1.1",
  "scheme": "https"
}
```

# Design Structured Log Line: Analysis-specific Fields

- Field required to answer questions specific to an analysis
- Example Analysis: Does the origin server upgrade to HTTP/2 when we make a HTTPS request?
- Challenge: How to ensure that we include sufficient fields to handle unforeseen, future fields?

## Structured Log Line

```
{
  [...]

  "httpver": "HTTP/1.1",
  "scheme": "https"
}
```

# Example Structured versus Unstructured Log Line

## Unstructured Log Line

```
req = &{Method:GET URL:/ Proto:HTTP/1.1 ProtoMajor:1 ProtoMinor:1
Header:map[User-Agent:[curl/7.47.0] Accept:[*/*]
X-Purl:[https://www.youtube.com/]] Body:{} GetBody:⟨nil⟩ ContentLength:0
TransferEncoding:[] Close:false Host:localhost:8080 Form:map[] PostForm:map[]
MultipartForm:⟨nil⟩ Trailer:map[] RemoteAddr:127.0.0.1:46312 RequestURI:/
TLS:⟨nil⟩ Cancel:⟨nil⟩ Response:⟨nil⟩ ctx:0xc42005e340}
```

## Structured Log Line

```
{"httpver":"HTTP/1.1", "level":"debug", "method":"GET", "msg":"Received
request from User Agent", "peer":"ua", "pkg":"main", "reqid":1518232390,
"scheme":"https", "time":"2018-02-10T08:43:10+05:30",
"url":"https://www.youtube.com/"}
```

# Overview

# Logrus: Overview

- Approach: Design "parts/fields" of each log; use Logrus package
- Documentation: [Online.] godoc.org/github.com/sirupsen/logrus
- Repository (and tutorial): [Online.] github.com/sirupsen/logrus
- Demo: Branch logrus

# Logrus: Code and Demo

Questions that logs can answer-

- Are there any errors? {simple}

# Logrus: Code and Demo

Questions that logs can answer-

- Are there any errors? {simple}
- What HTTP status codes are returned by the origin server? {doable}

# Logrus: Code and Demo

Questions that logs can answer-

- Are there any errors? {simple}
- What HTTP status codes are returned by the origin server? {doable}
- Does origin server upgrade to HTTP/2 for HTTPS? {analysis specific}

# Logrus: Code and Demo

Questions that logs can answer-

- Are there any errors? {simple}
- What HTTP status codes are returned by the origin server? {doable}
- Does origin server upgrade to HTTP/2 for HTTPS? {analysis specific}
- Can I replay what happened in a request? {deep analysis}

# Logrus: Code and Demo

Questions that logs can answer-

- Are there any errors? {simple}
- What HTTP status codes are returned by the origin server? {doable}
- Does origin server upgrade to HTTP/2 for HTTPS? {analysis specific}
- Can I replay what happened in a request? {deep analysis}
- Can I manage Time Series Data? {deep analysis}

# Logrus: Code and Demo

Questions that logs can answer-

- Are there any errors? {simple}
- What HTTP status codes are returned by the origin server? {doable}
- Does origin server upgrade to HTTP/2 for HTTPS? {analysis specific}
- Can I replay what happened in a request? {deep analysis}
- Can I manage Time Series Data? {deep analysis}
- Can I find out the most frequent requests from user agent? {business analysis}.

# Logrus: Merits and Demerits

- Merits: Level of abstraction | Ease of parsing | Good log design | Avoid missing log details | Production readiness
- Demerits: Simplicity? | Ugly?

# Conclusion

- Design logs as if they were composed of several fields.
- The logrus package provides structured, levelled and pluggable logging in Go.
- This presentation and GoPro code is available online at github.com/sarkutz/talk-go-logging


Thank You!