Logging in Go

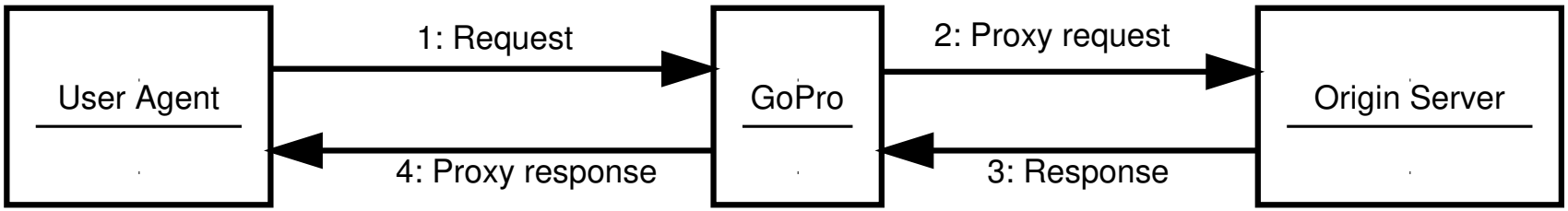
An Introduction to Structured Logging

Ashim Ghosh

February 2, 2018

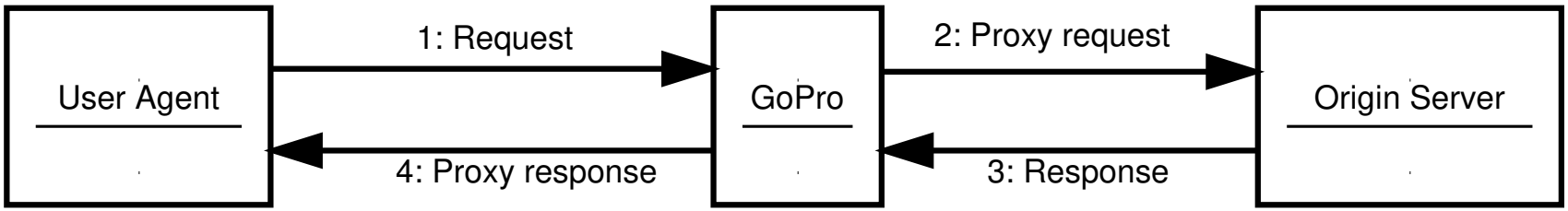
Overview

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

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

Why this scenario?

- A good model for the Ad Server
- Easy — won't get in the way

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 Easy 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) Easy 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, Error} X {, f, ln}`
  - `func SetLevel(int)`
- glog package
- Documentation: [Online.] godoc.org/github.com/golang/glog
- Repository: [Online.] github.com/golang/glog

- Merits: Level of abstraction | Production readiness
- Demerits: Easy 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
- An example

Logrus package: 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

Code and Demo

Questions that logs can answer-

- Are there any errors? {simple}

Code and Demo

Questions that logs can answer-

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

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? {feature specific}

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? {feature specific}
- Can I replay what happened in a request? {deep analysis}

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? {feature specific}
- Can I replay what happened in a request? {deep analysis}
- Can I manage stats and other Time Series Data? {deep analysis}

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? {feature specific}
- Can I replay what happened in a request? {deep analysis}
- Can I manage stats and other Time Series Data? {deep analysis}
- Can I find out the most frequent requests from user agent? {business queries}

Logrus package: Merits and Demerits

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

Overview

Ad Server

This slide is intentionally left blank

Ad Server

This slide is intentionally left blank . . .

for you and me to fill

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!