A go-logr de-entanglr

Francesco Romani - fromani@redhat.com

Problem statement

To debug, we need high verbosity logs.

High verbosity logs require storage and CPU time.

High verbosity logs are disabled by default in production.

Verbosity is increased only after an issue is found.

What if…

What if we have a way to always get high verbosity logs only for the selected few we care most about?

What if verbosity is bumped automatically when we hit these flows? (per-request/flow verbosity)

What if we can minimize the high verbosity cost?

Elevator pitch

Pluggable package to

split high-value per-flow logs

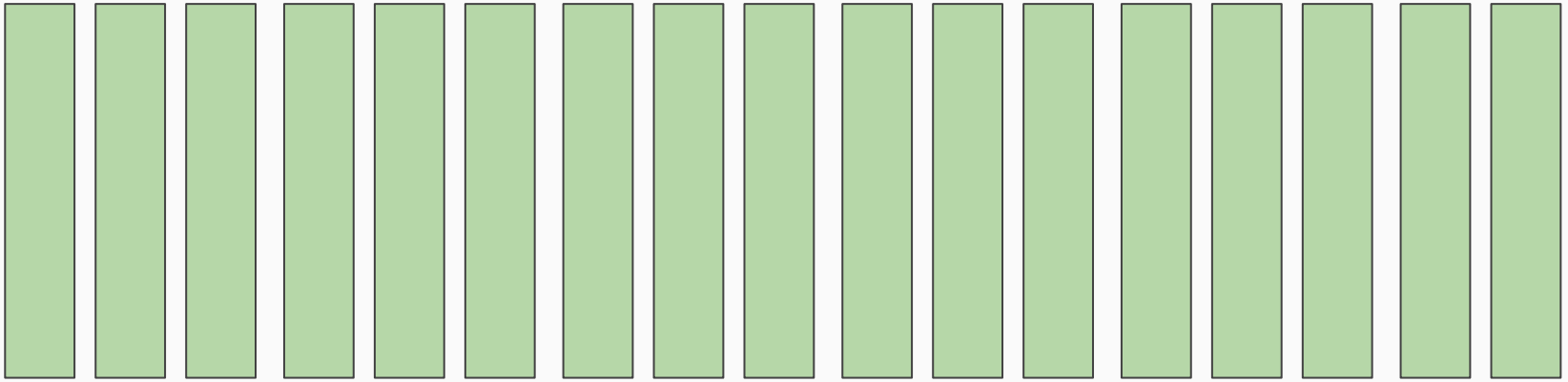
from the general log stream

What’s a log?

A log is a stream of interleaved unframed streams

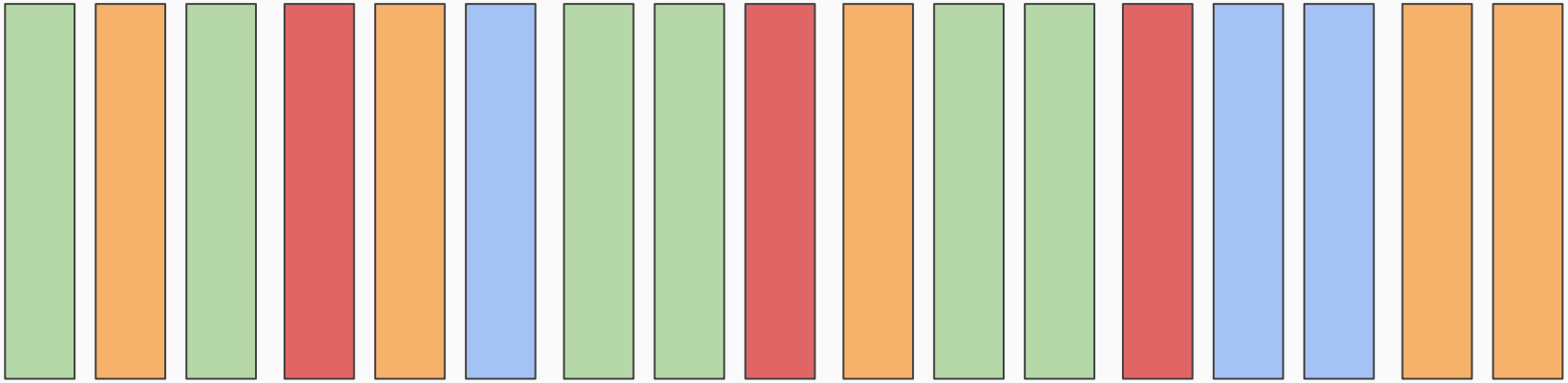
A stream

A stream is a continuous unbounded sequence of items



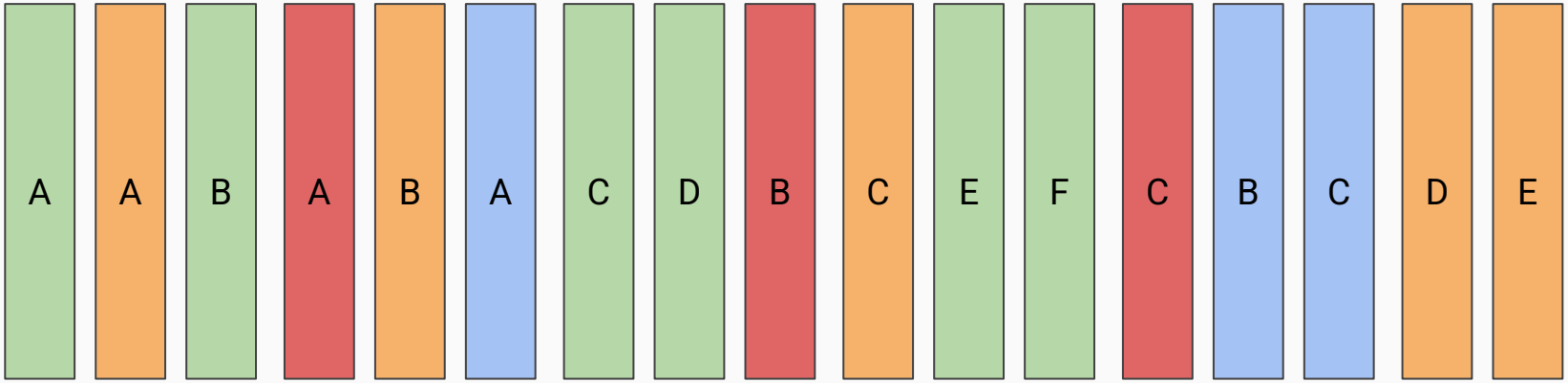
Interleaved streams

All components of a process produce log concurrently



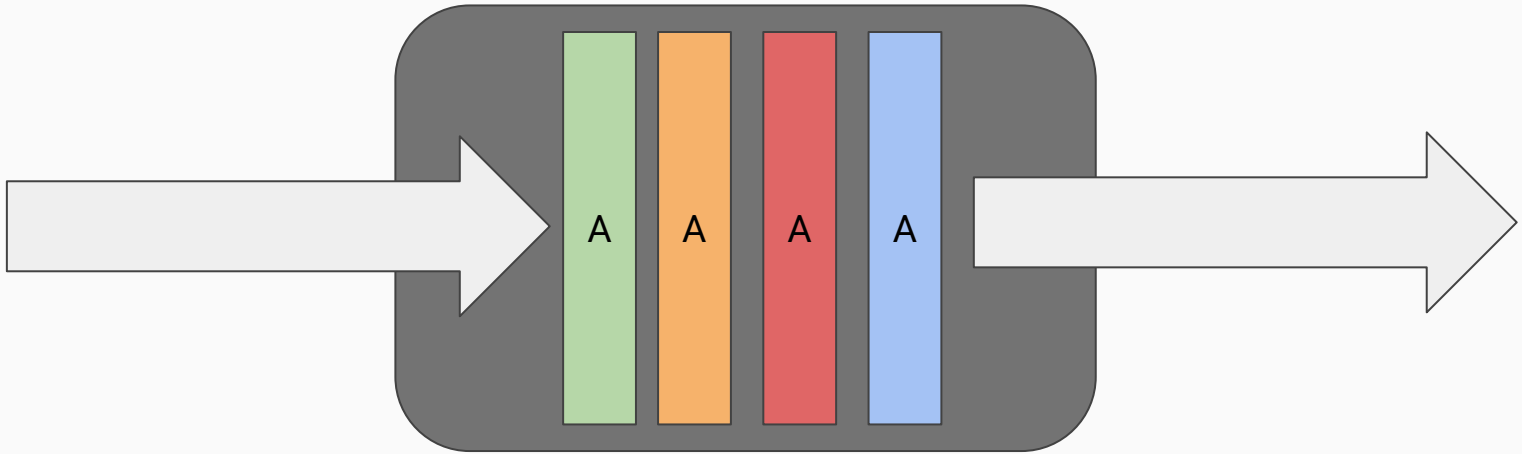
Unframed streams

Servers process requests. Which logs belong to which request?

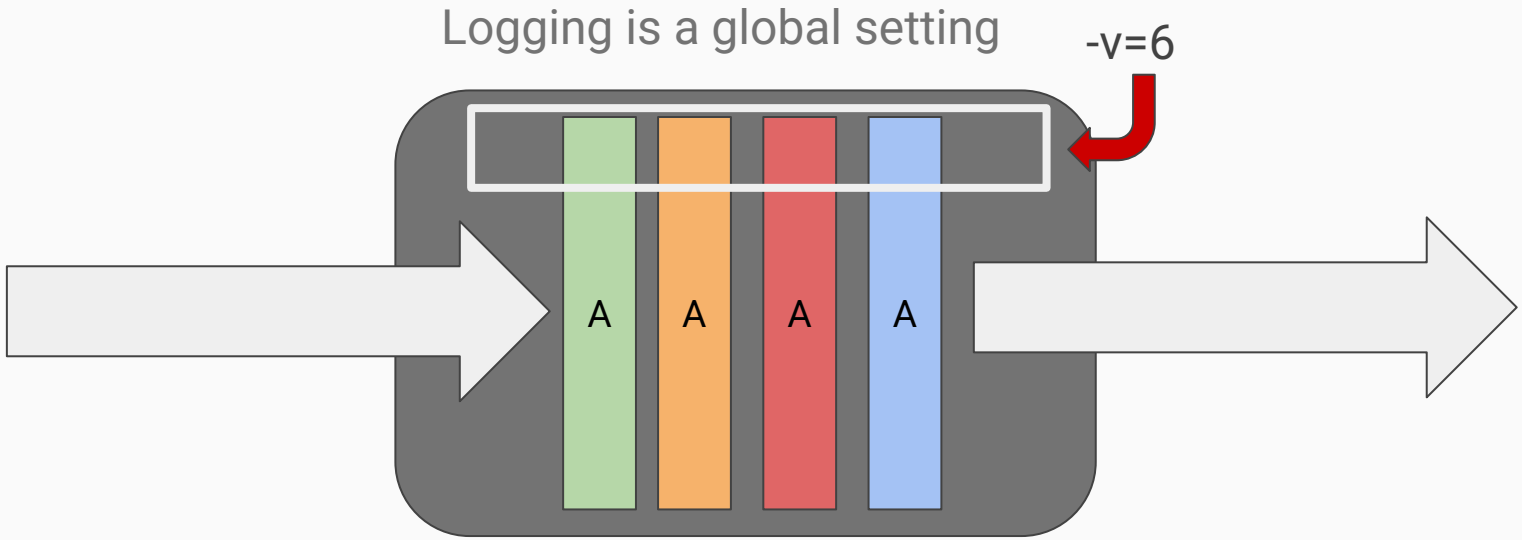


The request flow

A request travels across modules in a process

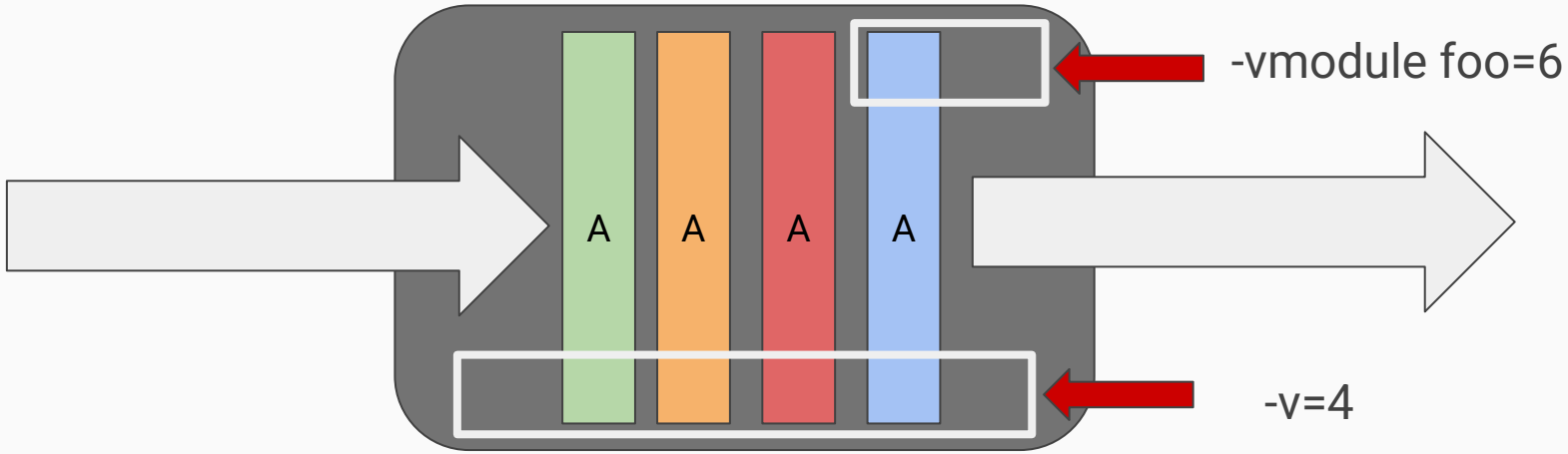


Logging is per process



Logging is per module

Logging is not only a global setting



Logging is per module: problem solved?

Problems with per-module verbosity:

- Seldom used :)
- Wrong focus: module vs request
- Need to know which modules I care about (can and will change over time)
- Complex configuration
- What if the module is used outside the request flow? (global cache…)

TL;DR: yeah we can maybe make it work somehow. Can we do better?

What about telemetry?

Telemetry is great, but solves a different problem

- Needs extra components in the cluster
- Needs instrumentation (code changes)
- Exports counters - we want journals

We already have and we will always have logs - let’s make them better

The verbosity dilemma

How do we set verbosity?

- Low verbosity: low storage usage, low information: “please retry with -v=4”
- High verbosity: high storage usage, high information, high noise (information I don’t care about - grep to the rescue, but still)

High verbosity is not recommended for production settings: creates unnecessary load, users don’t want this.

What do we need then?

The ideal solution would look like this:

- Focus on request flow
- High verbosity for everything needed to serve the request
- Low verbosity for everything else
- Modules should be verbose only for the bits related to the request
- Link together all the logs pertaining to the served request
- Retain the high-value logs longer than the low-value logs

Building blocks: structured logging

“Structured logs use key-value pairs so they can be parsed, filtered, searched, and analyzed quickly and reliably.”

Golang >= 1.21

Kubernetes (klog) >= 1.18 (or so)

use a key=value pair to identify the request boundaries in the logs (framing)

Building blocks: contextual logging

“Contextual logging is based on the go-logr API. The key idea is that libraries are passed a logger instance by their caller and use that for logging instead of accessing a global logger. The binary decides the logging implementation, not the libraries. The go-logr API is designed around structured logging and supports attaching additional information to a logger.” (ref)

Inject per-request loggers

Inject per-request loggers with different verbosity levels

Key idea: self-assessed verbosity level

How does a logger know it is serving a particular request, hence needs a higher verbosity without user intervention?

- We are using structured logging: key=value pairs for important information
- Inspect key-value pairs
- If pre-defined key-value pairs (+ logger name?) is found, bump verbosity automatically

What do we ~~need~~ have?

Leveraging the building blocks:

- **Focus on request flow - OK!**
- **High verbosity for everything needed to serve the request - OK!**
- **Low verbosity for everything else - OK!**
- **Modules should be verbose only for the bits related to the request -OK!**
- **Link together all the logs pertaining to the served request - OK!**
- Retain the high-value logs longer than the low-value logs - PENDING

Retain the high-value logs longer

A log is

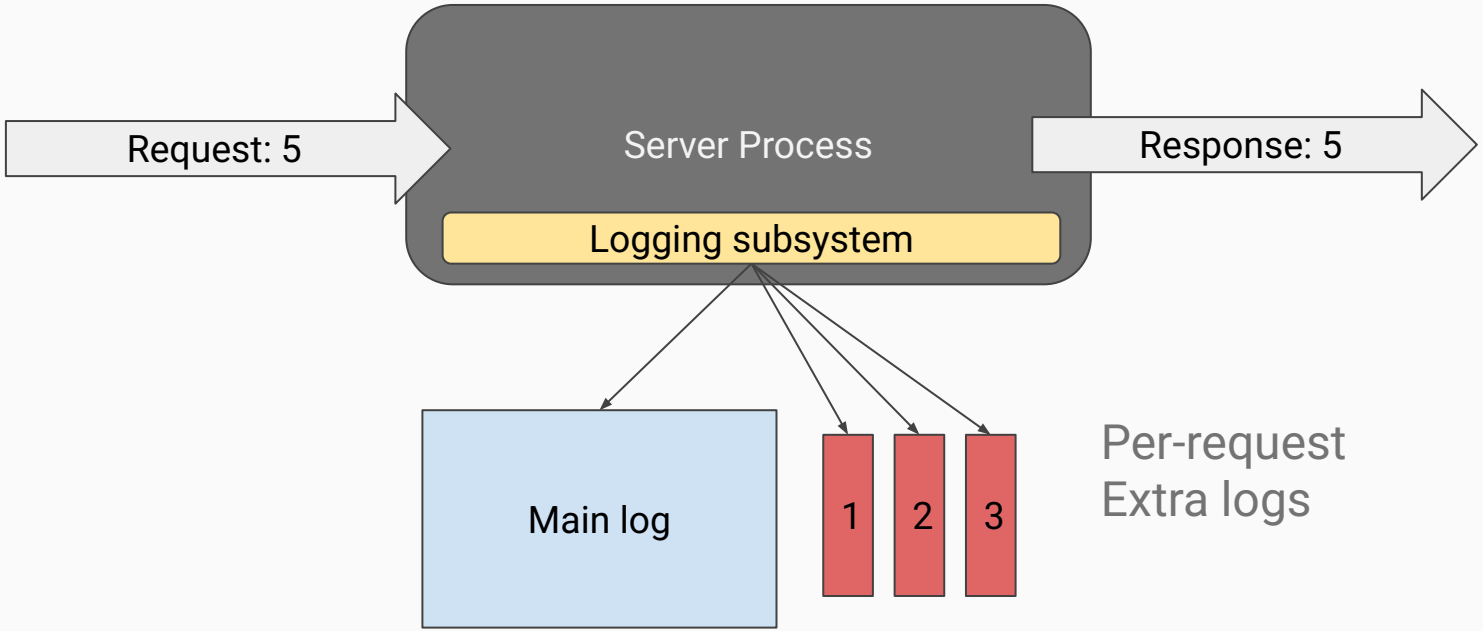
a stream of	<- PENDING: we are still putting everything in a single sink
interleaved	<- SOLVED! use key=value to deinterleave
unframed	<- SOLVED! Use key=value to mark request boundaries
streams	<- SOLVED! Bump verbosity using per-request loggers

Separate low-value from high-value

Key Idea: since we can now separate high-value logs, why don’t we send them to a different stream?

- No duplication: the main stream is low-verbosity anyway, just put the extra data elsewhere
- We can have different retention (rotation) policies per-stream: keep the high value streams longer

Separate low-value from high-value



Putting all together

https://github.com/k8stopologyawareschedwg/logtracr

Caveats:

- Alpha grade or less
- Design still in flux: how to inject matching funcs? How to handle rotation?
- Requires quite a lot of preparation in the hosting process **which should be done anyway to adhere to modern logging practices**

Do we **need** all this complexity?

NO, we don't.

We can “”””just””” **rearchitect** the application to

- Take multiple loggers
- Use multiple loggers in the relevant code flows

```
mainLogger.V(6).Info(“main message”, “answer”, 42)

specificLogger.Info(“main message”, “answer”, 42)
```

Do we **want** to rearchitect like that?

- Take multiple loggers
- Use multiple loggers in the relevant code flows

```
mainLogger.V(6).Info(“main message”, “answer”, 42)

specificLogger.Info(“main message”, “answer”, 42)
```

Thanks!

Questions?

fromani@redhat.com (or @gmail.com)

@fromani on (k8s) slack