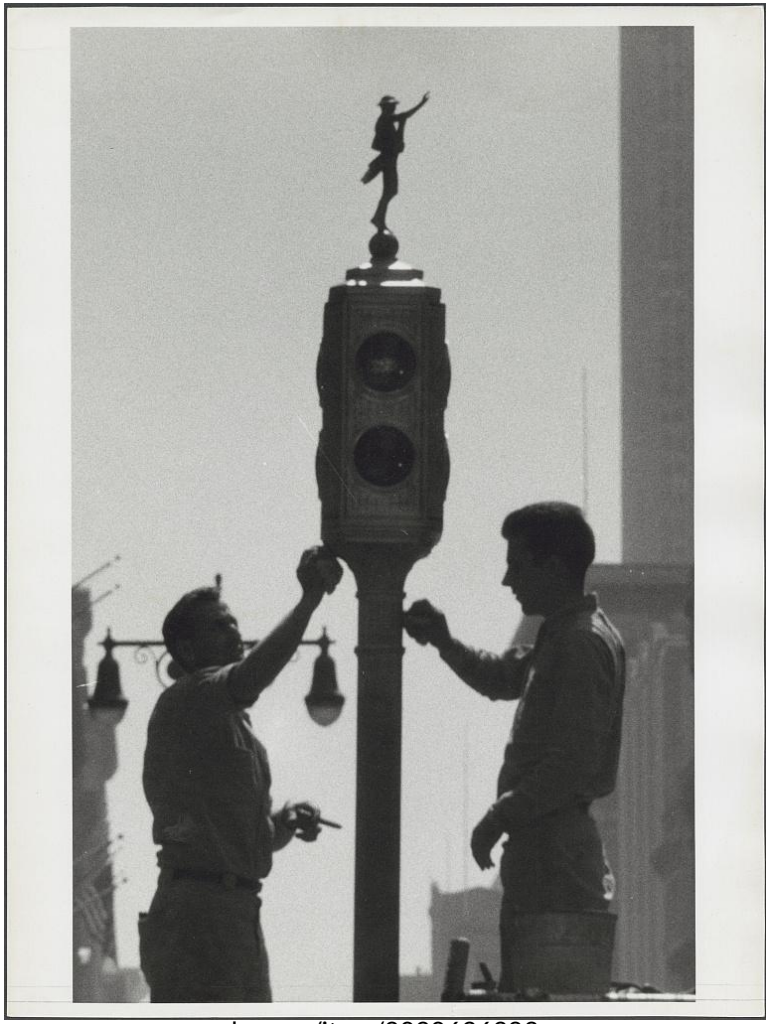
OpenResty, My Bestie

Tracing NGINX With Lua

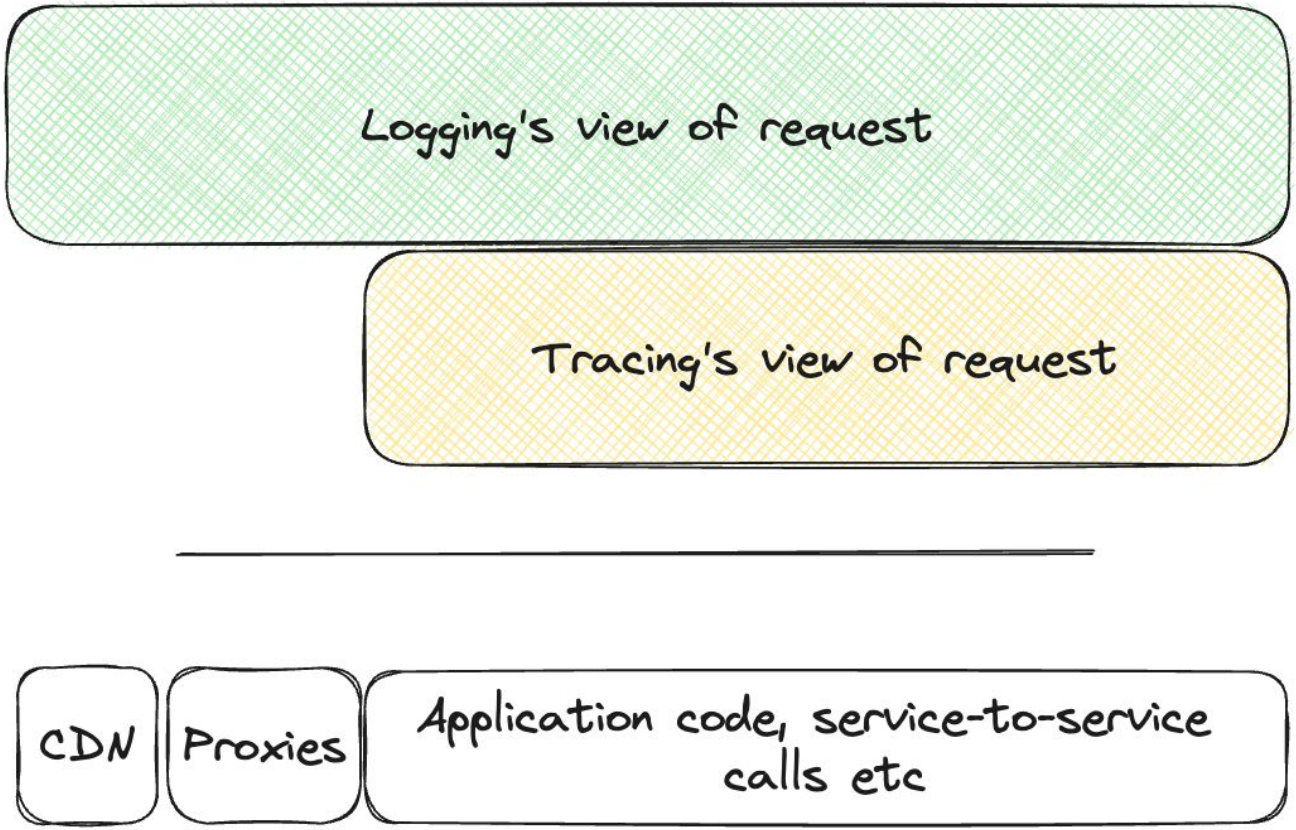
Monitorama 2023

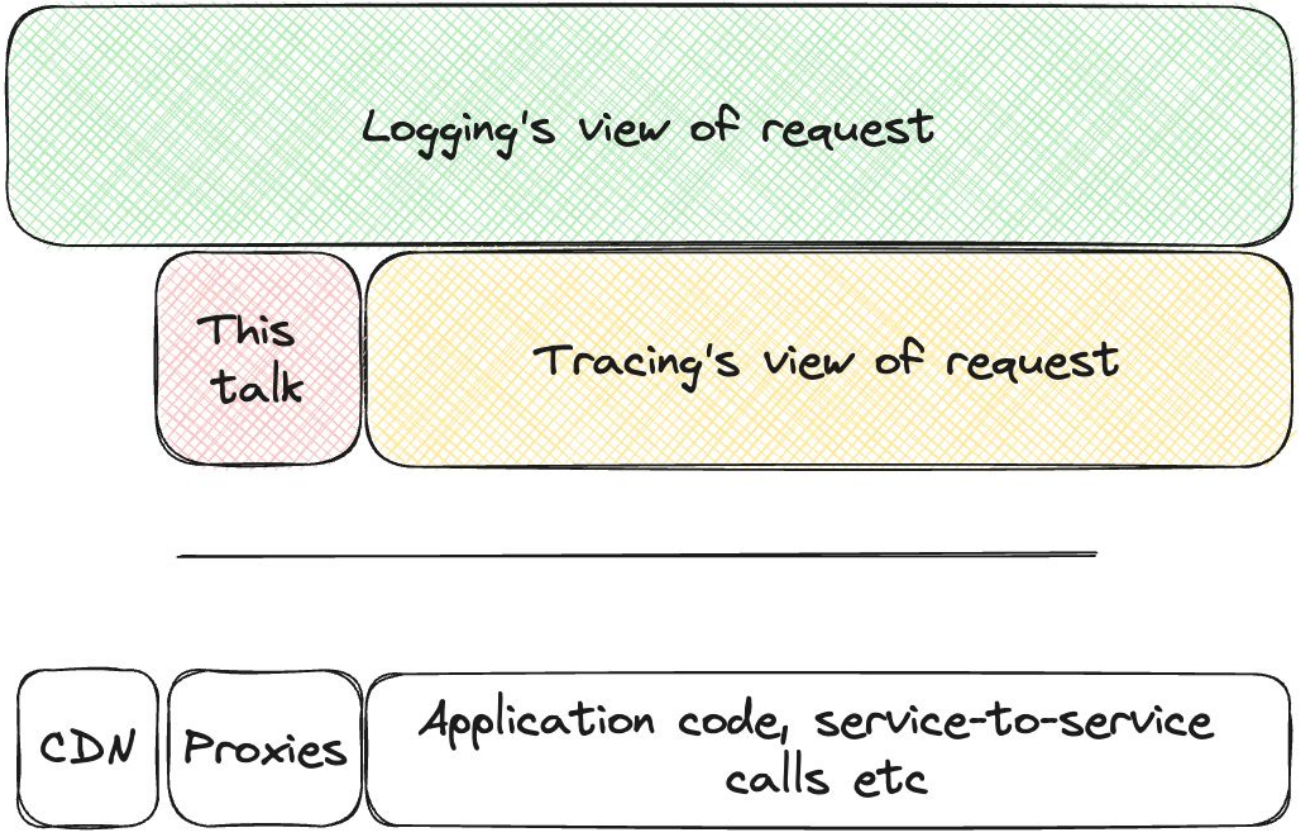
Sam Handler, Shopify



loc.gov/item/2020636383

- Sam Handler
- @plantfansam
- Staff Engineer in Observability at Shopify


Logging's view of request
Tracing's view of request
CDN
Proxies
Application code, service-to-service calls etc

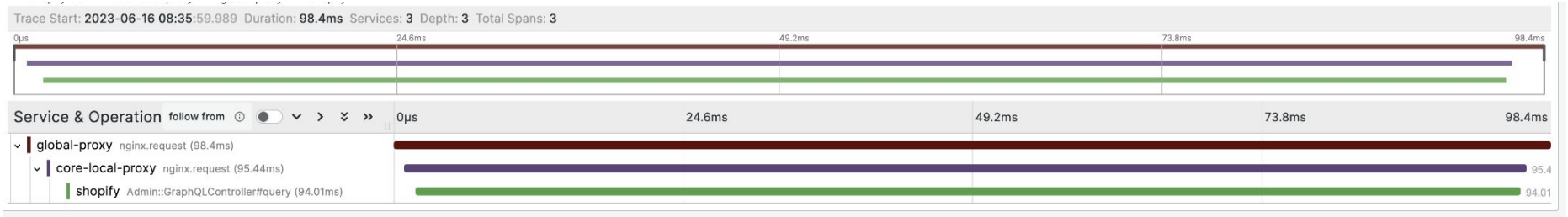

Logging's view of request
This talk
Tracing's view of request
CDN
Proxies
Application code, service-to-service calls etc

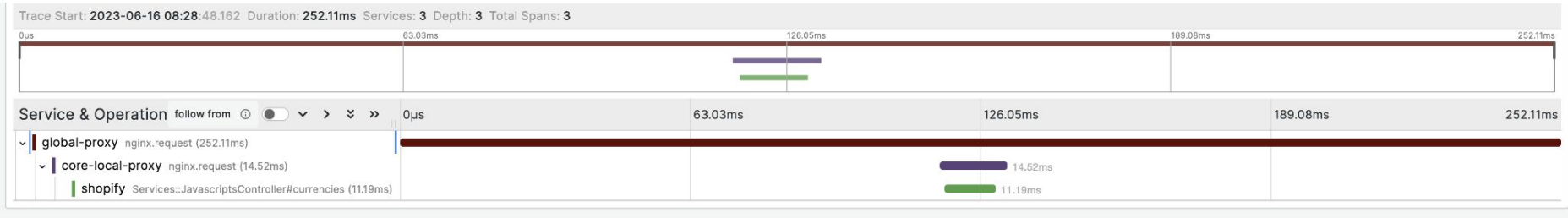
Agenda

👉 What tracing a proxy looks like

2 - The initial strategy

3 - Productionizing + demo


Trace Start: 2023-06-16 08:35:59.989 Duration: 98.4ms Services: 3 Depth: 3 Total Spans: 3
0µs
24.6ms
49.2ms
73.8ms
98.4ms
Service & Operation
follow from
global-proxy nginx.request (98.4ms)
core-local-proxy nginx.request (95.44ms)
shopify Admin::GraphQLController#query (94.01ms)
95.4
94.01


Trace Start: 2023-06-16 08:28:48.162 Duration: 252.11ms Services: 3 Depth: 3 Total Spans: 3
0µs
63.03ms
126.05ms
189.08ms
252.11ms
Service & Operation
follow from
global-proxy nginx.request (252.11ms)
core-local-proxy nginx.request (14.52ms)
shopify Services::JavascriptsController#currencies (11.19ms)
14.52ms
11.19ms


Trace Start: 2023-06-16 08:28:48.162 Duration: 252.11ms Services: 3 Depth: 3 Total Spans: 3
0µs
63.03ms
126.05ms
189.08ms
252.11ms
Service & Operation
follow from
global-proxy nginx.request (252.11ms)
nginx.request
Service: global-proxy
Duration: 252.11ms
Start Time: 0µs (08:28:48.162)
Child Count: 1
Logs for this span
Attributes: http.flavor = 1.1 | http.method = GET | http.scheme = https | http.status_code = 200 | http.target = /services/javascripts/currencies.js | http.user_agent = Mozilla/5.0 (Linux; Androi...
Resource: cloud.provider = gcp | cloud.region = asia-south1 | deployment.environment = production | host.name = nginx-ingress-controller-64f6b69b8-2vmq6 | k8s.cluster.name = routing-as...
SpanID: aafcc54ef144eb66
core-local-proxy nginx.request (14.52ms)
14.52ms | core-local-proxy::nginx.request
nginx.request
Service: core-local-proxy
Duration: 14.52ms
Start Time: 117.16ms (08:28:48.279)
Child Count: 1
Logs for this span
Attributes: http.flavor = 1.1 | http.method = GET | http.scheme = https | http.status_code = 200 | http.target = /services/javascripts/currencies.js | http.user_agent = Mozilla/5.0 (Linux; Androi...
Resource: cloud.provider = gcp | cloud.region = us-central1 | deployment.environment = production | host.name = nginx-ingress-controller-5c5848c899-89z8w | k8s.cluster.name = core-us-...
SpanID: 8f1137b259cd663e
shopify Services::JavascriptsController#currencies (11.19ms)
11.19ms

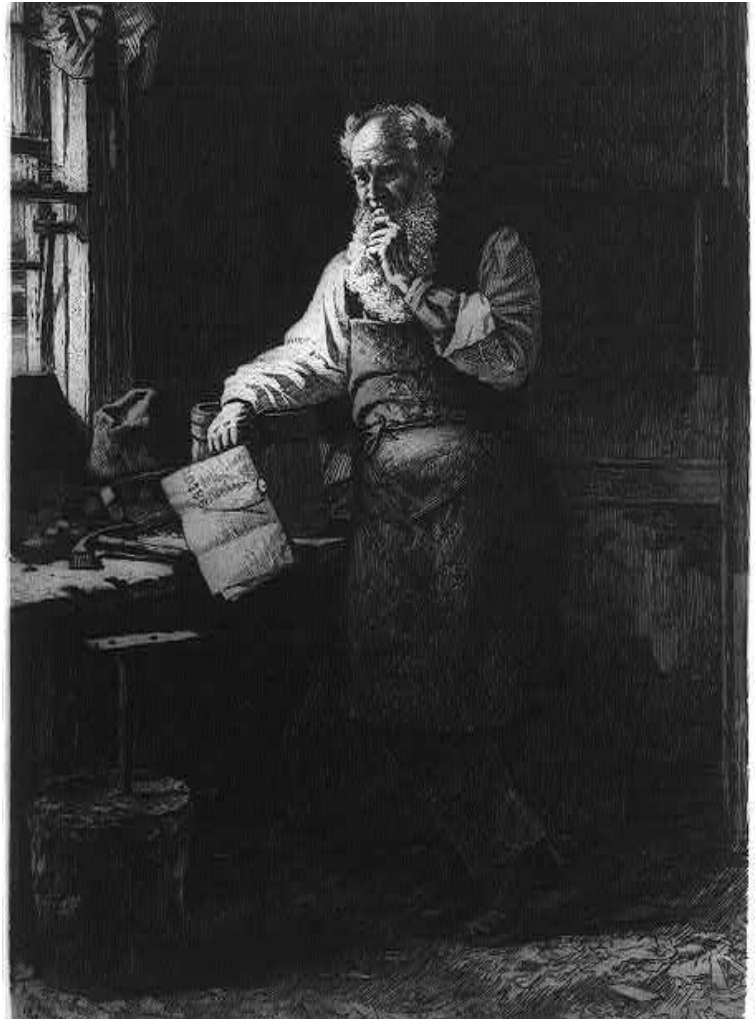
Agenda

1 - What tracing a proxy looks like

👉🏻 The initial strategy

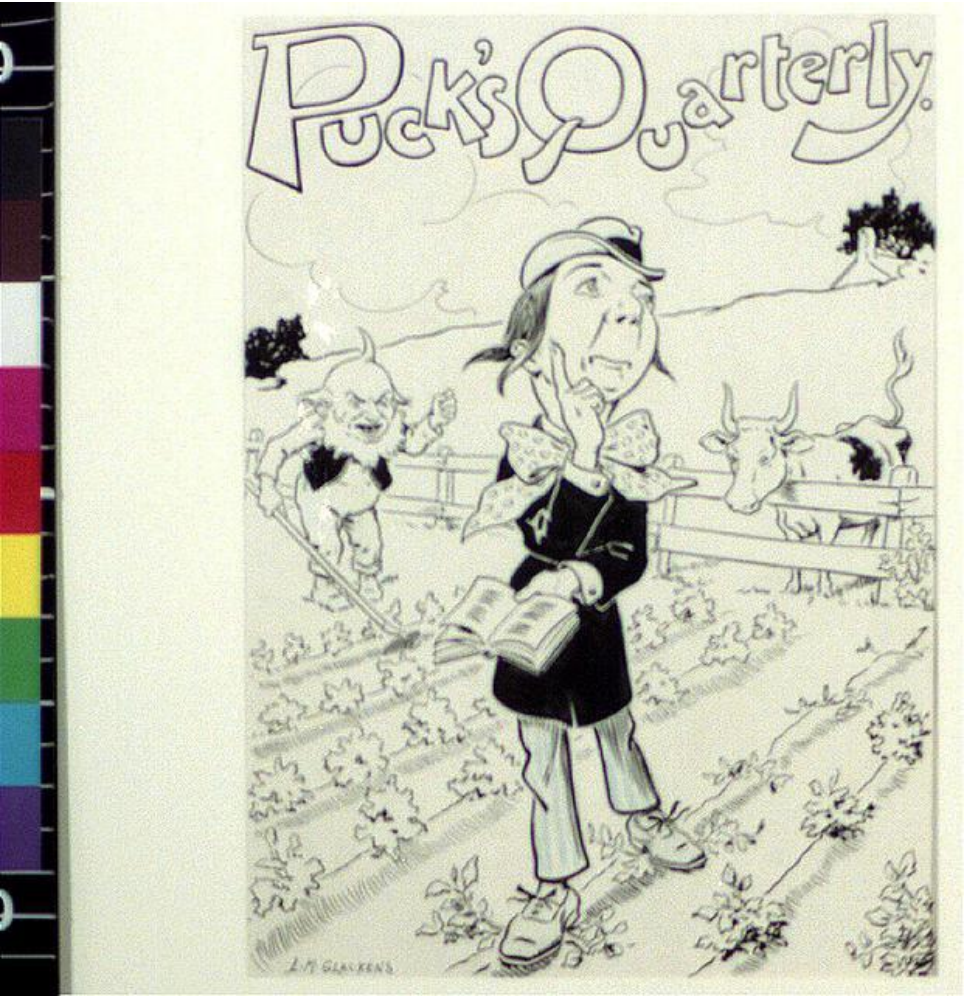
3 - Productionizing

OpenTelemetry or something else?



https://www.loc.gov/resource/cph.3a01622/

Shopify ❤️ OpenTelemetry



https://www.loc.gov/resource/acd.2a08815/

Use opentelemetry-cpp or use OpenResty directives?

opentelemetry-cpp NGINX module here
opentelemetry- lua lib here

```
http {
    server {
        listen 8080;
        location / {
            init_worker_by_lua_block {
                -- In Lua,set variables, do math ... whatever!
                ngx.log(ngx.INFO, "hi everyone")
            }
            rewrite_worker_by_lua_block {
                ngx.log(ngx.INFO, "hi from rewrite")
            }
            log_by_lua_block {
                ngx.log(ngx.INFO, "this runs after response to client")
            }
        }
    }
}
```

OpenResty directives docs are here

OpenResty + kubernetes/ingress-nginx

README is here

```lua
local _M = {}

function _M.init_worker()
  -- ingress-nginx calls during worker initialization
end

function _M.rewrite()
  -- ingress-nginx calls during rewrite phase
end

function _M.header_filter()
  -- ingress-nginx calls during header filter phase
end

function _M.body_filter()
  -- ingress-nginx calls during body filter phase
end

function _M.log()
  -- ingress-nginx calls during log phase
end
```

```
local trace_context_propagator = require(
    "plugins.opentelemetry.shopify_propagator").new()
local new_context = require("opentelemetry.context").new

function _M.rewrite()
  local tracer = _M.tracer() -- pretend it's implemented 🙂
  local inbound_context = trace_context_propagator:extract(
      new_context(), ngx.req)
  local request_span_ctx = tracer:start(inbound_context, "nginx.request")
  trace_context_propagator:inject(request_span_ctx, ngx.req)
  ngx.ctx["opentelemetry"] = { request_span_c = request_span_ctx }
end

function _M.header_filter()
  ngx.ctx.opentelemetry.request_span_c.sp:finish()
end
```

Agenda

1 - What tracing a proxy looks like

2 - The initial strategy

👉🏻 Productionizing

Shopify served 75,980,000 requests per minute
on Black Friday / Cyber Monday peak, 2022

twitter.com/ShopifyEng/status/1597983926126977024

Productionizing

01

Span export

Let's not DDOS ourselves

02

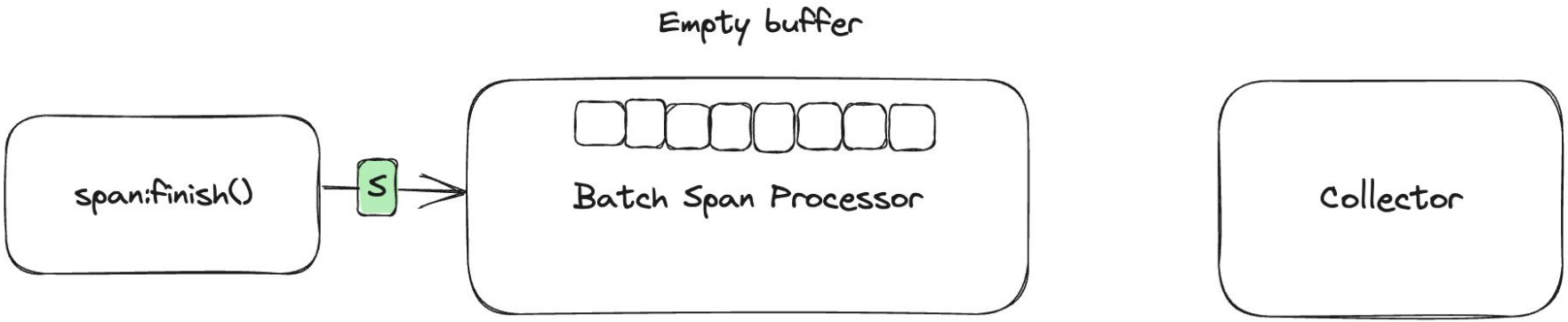
Process model

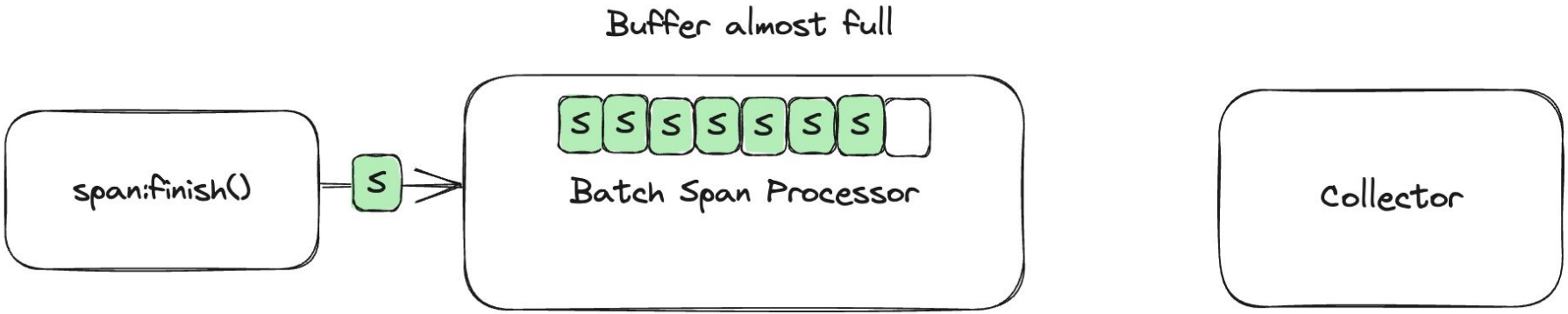
Memory in NGINX/OpenResty

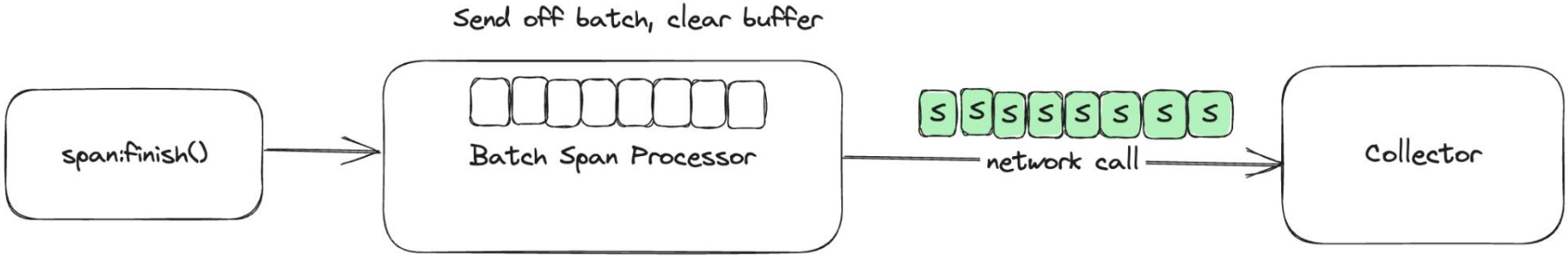
03

Resiliency/perf

Don't break the site


Empty buffer
span:finish()
S
Batch Span Processor
Collector


Buffer almost full
span:finish()
S
S S S S S S S
Batch Span Processor
Collector


Send off batch, clear buffer
span:finish()
Batch Span Processor
S S S S S S S S
network call
Collector

Productionizing

01

Span export

Let's not DDOS ourselves

02

Process model

Memory in NGINX/OpenResty

03

Resiliency/perf

Don't break the site

NGINX and Concurrency

nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale

Each NGINX worker runs
single threaded

nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/#process-model

OpenResty memory management

- “To globally share data among all the requests handled by the same Nginx worker process, encapsulate the shared data into a Lua module, use the Lua require builtin to import the module, and then manipulate the shared data in Lua…”

github.com/openresty/lua-nginx-module

OpenResty memory management

- “It is **usually recommended to share read-only data this way**. You can also share changeable data among all the concurrent requests of each Nginx worker process as long **as there is no nonblocking I/O operations…in the middle of your calculations**. As long as you do not give the control back to the Nginx event loop and ngx_lua's light thread scheduler (even implicitly), there can never be any race conditions in between. For this reason, **always be very careful when you want to share changeable data on the worker level**. Buggy optimizations can easily lead to hard-to-debug race conditions under load.”

  (emphasis mine)

github.com/openresty/lua-nginx-module

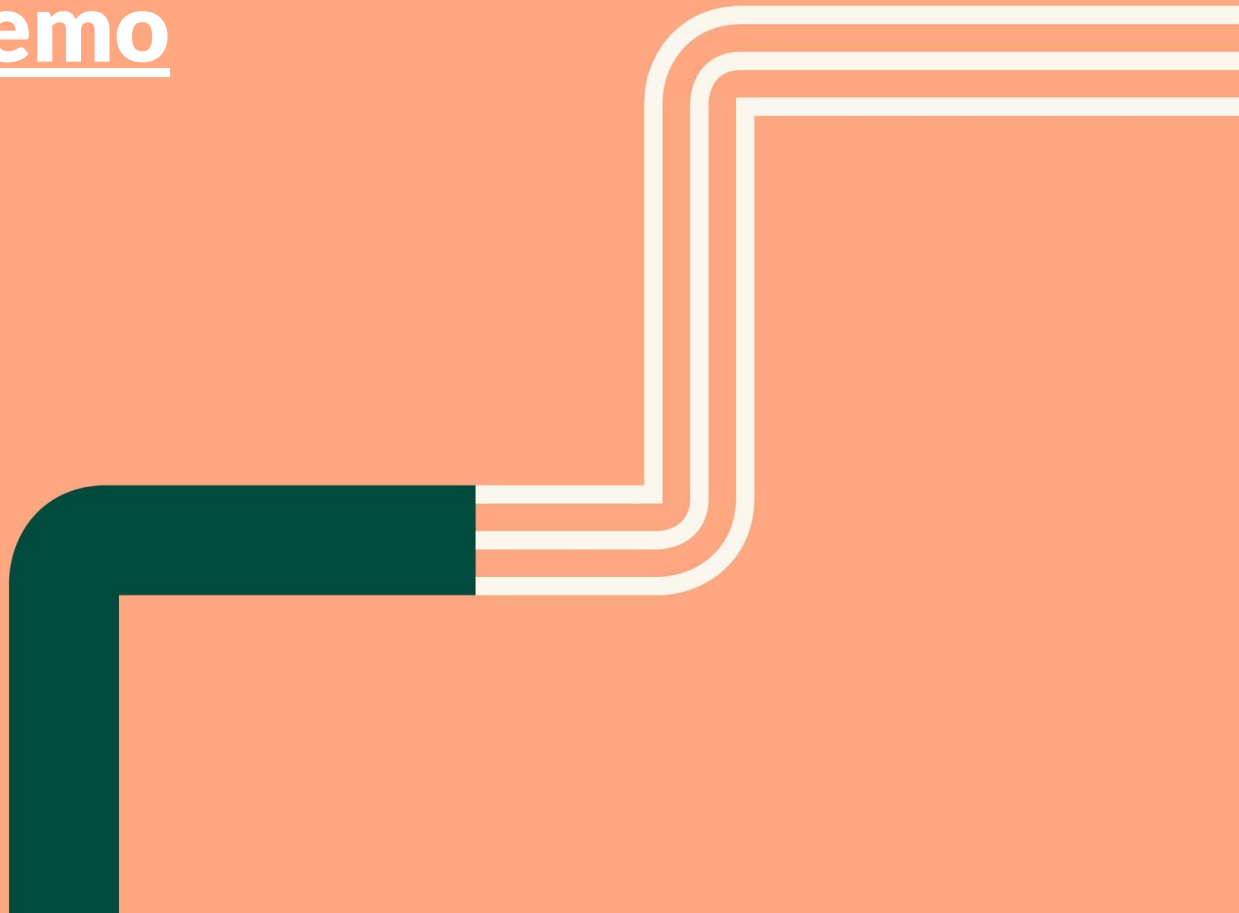
Process model recap

- NGINX has a master process with N child worker processes
- NGINX child worker processes run single-threaded
- Each request in an NGINX worker process shares the same Lua interpreter
- You can share changeable data across concurrent requests fielded by the same NGINX worker
- …as long as you don’t mess it up

The strategy

- On booting each NGINX worker, instantiate batch span processor and store on Lua module instance
- All requests fielded by worker X push to same batch span processor
- Don’t mess it up

A live demo



Productionizing

01

Span export

Let's not DDOS ourselves

02

Process model

Memory in NGINX/OpenResty

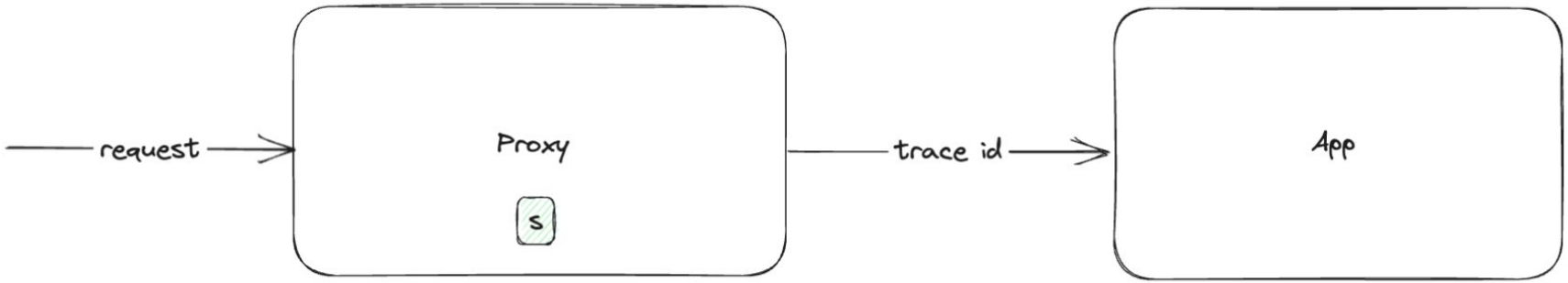
03

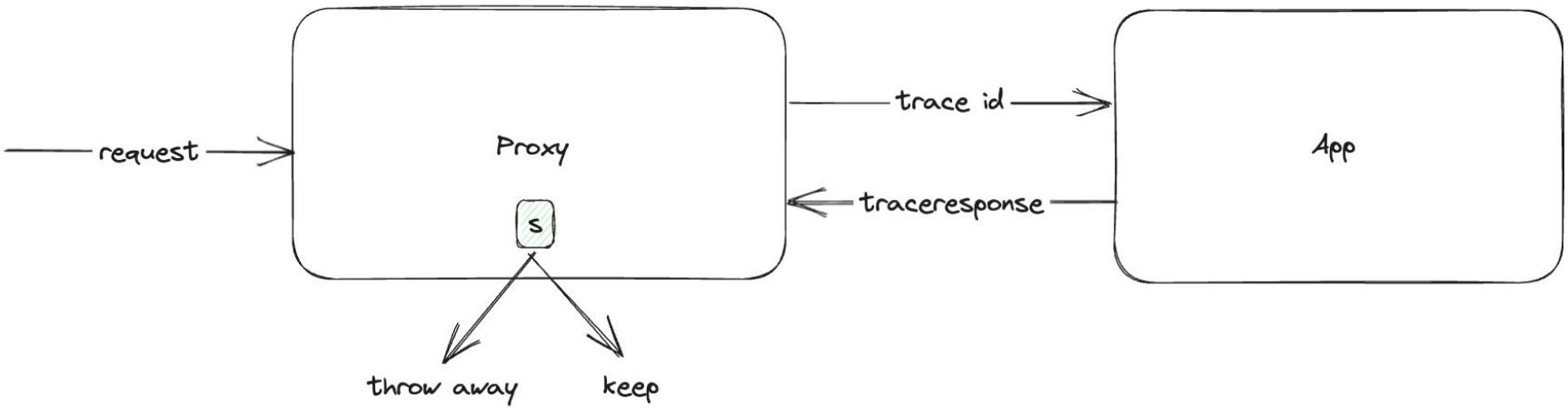
Resiliency/perf

Don't break the site

Resiliency and performance

- If span buffer is full, throw span away
- Add circuit breaker + backoff for calls to collector (pr, pr)
- Sped up trace and span ID generation by tweaking random number strategy (pr)
- Move almost everything to the log phase
- Manage trace volume with deferred sampling (w3c spec)


request
Proxy
S
trace id
App


request
Proxy
S
throw away
keep
trace id
traceresponse
App

What I skipped

- LuaJIT doesn’t love w3c-standard trace ids
- Converting configmap entries to Lua tables in NGINX config 😬
- I messed up the histograms and thought this plugin was really slow

Thank you!