

OpenResty XRay 簡介

對您的線上應用進行全天候、無死角的分析 and 診斷



試用 OpenResty XRay 產品

<https://openresty.com/tw/xray/>

info@openresty.com

軟體世界的挑戰

業務快速迭代

團隊成員水平不一

測試不充分

業務複雜度不斷提高

CPU 資源佔用過高

記憶體資源佔用過多（含記憶體洩漏）

硬碟 IO 資源不足

存在長延時響應

很難線下復現的異常和錯誤（含程序崩潰）

K8s/Docker 容器时代 的挑战

很多容器，很多应用，很多发行版，很多技术栈

容器最小化，缺乏最基本的调试工具

容器权限集合最小化

出问题时自动丢弃和重启容器，软件 Bug 容易被掩盖

容器虚拟化、微服务——软件复杂度进一步提高

傳統方法的缺點

侵入式需要修改應用

響應需求慢

需要大資料儲存和分析

僅限表面指標

只有現象，沒有原因

缺少深度全技術棧分析和診斷

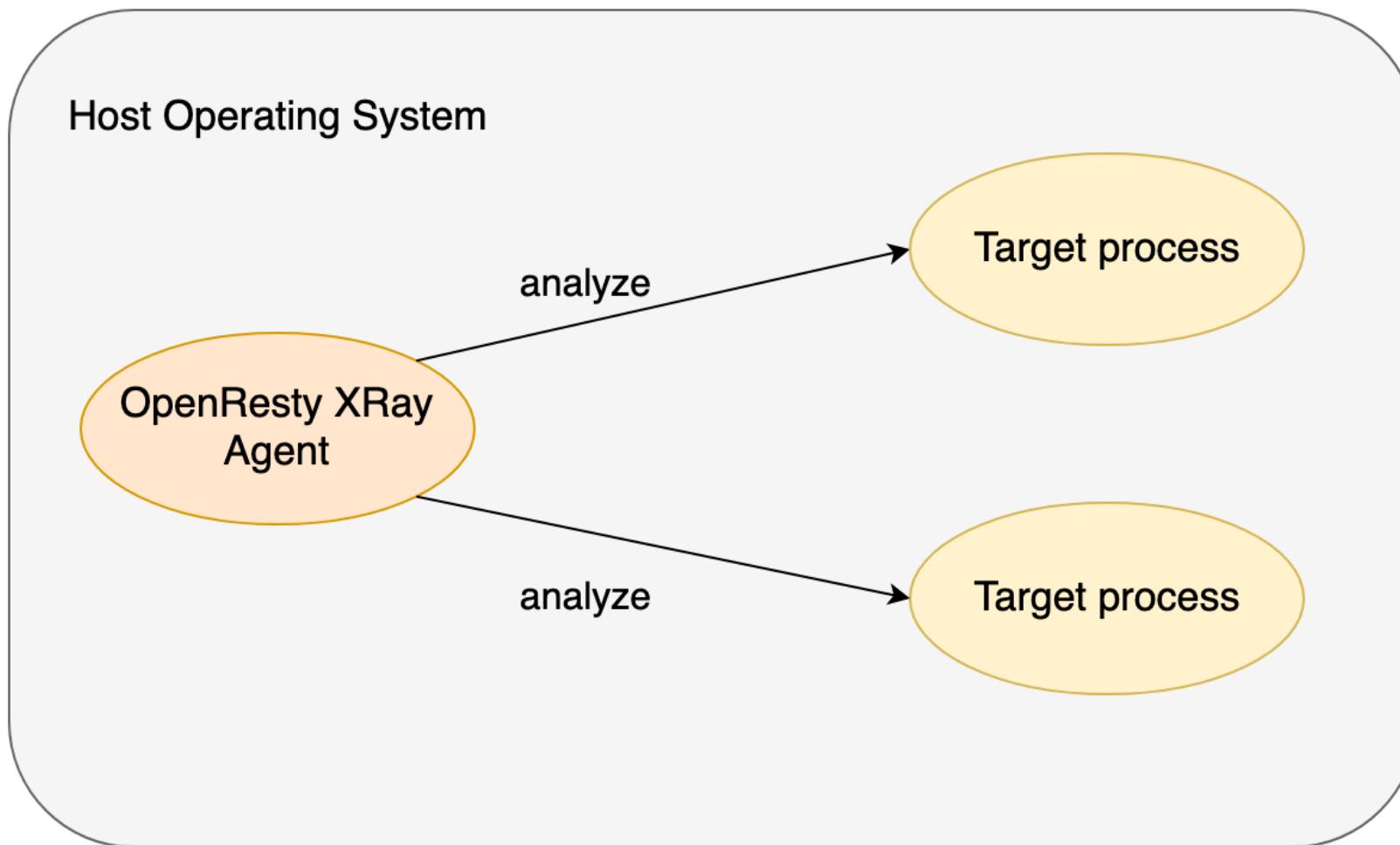
資料採集和處理流程複雜，開銷大，易出錯

OpenResty XRay

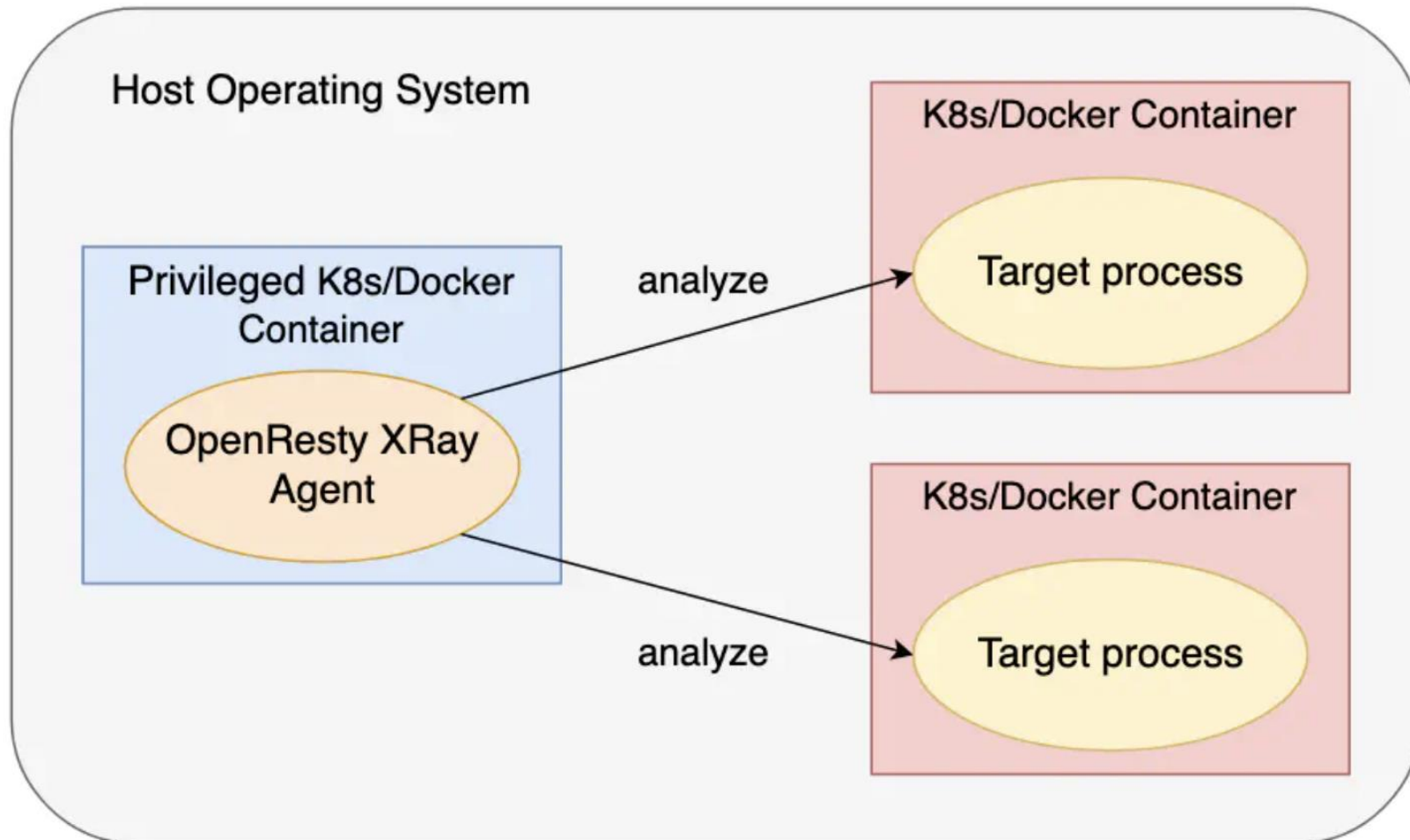


- OpenResty XRay 是一款動態追蹤產品
 - 可以實時分析各種雲和伺服器應用程式
 - 將執行中的程序和容器視為只讀資料庫，並提取必要的資訊來解決效能問題、異常、錯誤和安全漏洞
 - 擁有知識庫、推理引擎和數百個高階分析器
 - 可以在不改變或影響目標應用程式的情況下診斷和縮小深層問題的根源
-

OpenResty XRay 直接分析非容器应用程序

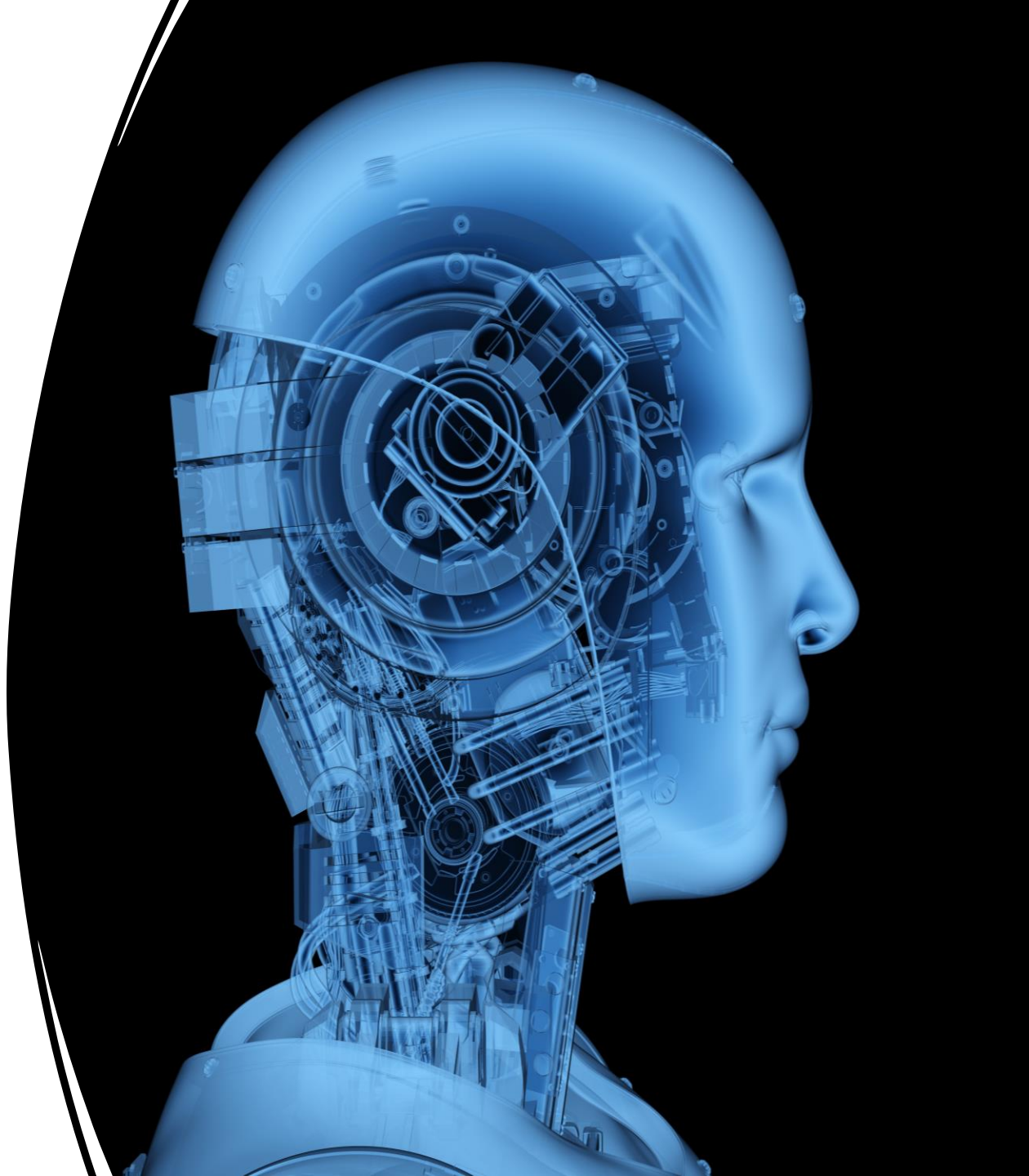


OpenResty XRay 穿透容器分析應用



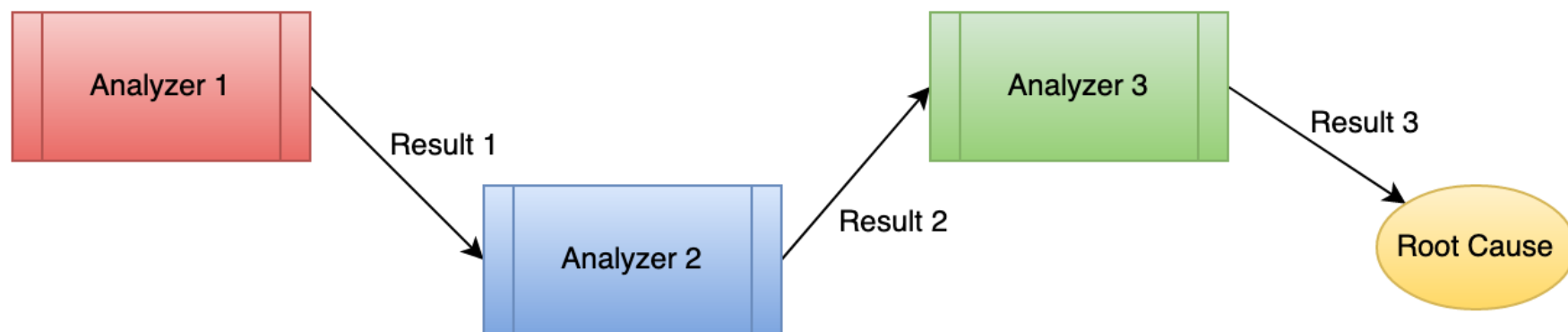
100% 非侵入式

- 無需修改您的應用
- 無需加裝新的外掛、模組或補丁到您的應用
- 無需注入任何程式碼到您的應用
- 無需重啟您的應用程序
- 無需在您的應用中使用特殊的啟動或編譯選項
- 無需重建您現有的應用容器或應用軟體包



OpenResty XRay 全自動取樣 無人值守的使用模式

- 定時取樣
- 事件驅動取樣 (CPU 變化, 記憶體變化, IO 變化, 異常錯誤)
- 鏈式推理驅動



極低的效能損耗

- 未取樣時的效能開銷嚴格為 0
- 取樣時的效能開銷通常不易覺察



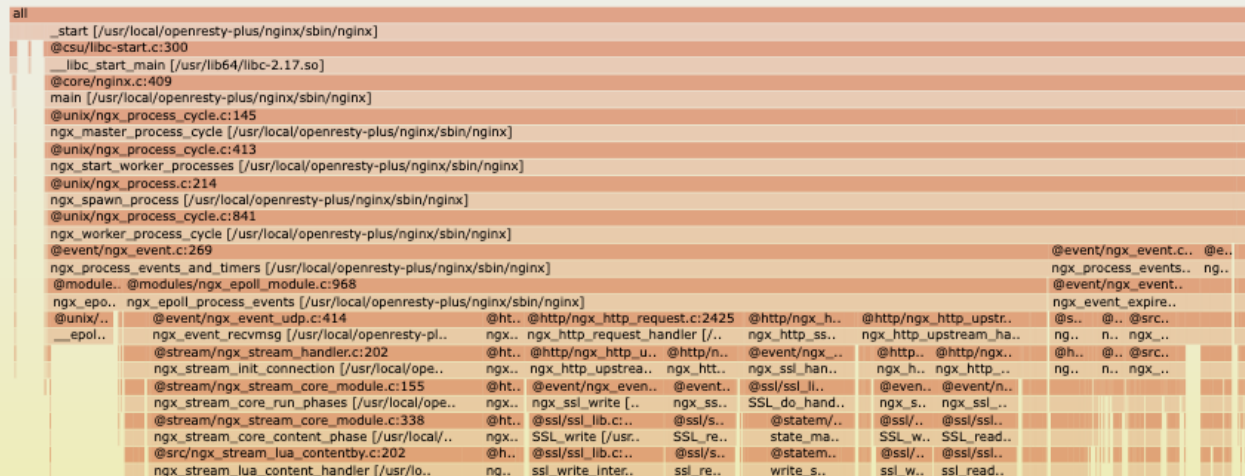
OpenResty XRay CPU 效能分析

- 高 CPU 使用率會降低系統穩定性和服務質量，甚至導致服務不可用
- CPU 時間在不同場景下，是如何分佈在不同程式碼路徑上的（火焰圖，火焰圖自動直譯器）
- 覆蓋不同軟體層面的程式碼路徑：業務程式語言層面（Lua/Python/PHP/Perl/Go/等等），系統程式語言層面（C/C++/Rust），作業系統核心層面（網路協議棧/程序排程器/記憶體管理/系統呼叫）
- 常見的 CPU 瓶頸舉例：重複計算（缺少快取）、SSL 握手相關、垃圾回收（GC）開銷，動態記憶體分配開銷、序列化與反序列化、意外的密集系統呼叫、死迴圈、病態正則表示式匹配、實現低效的（第三方）軟體庫、自旋鎖競爭

C-Land CPU Flame Graph for LuaJIT

Search

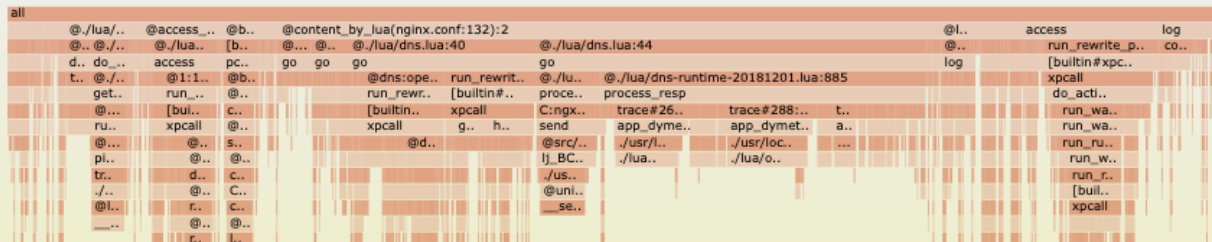
whole application



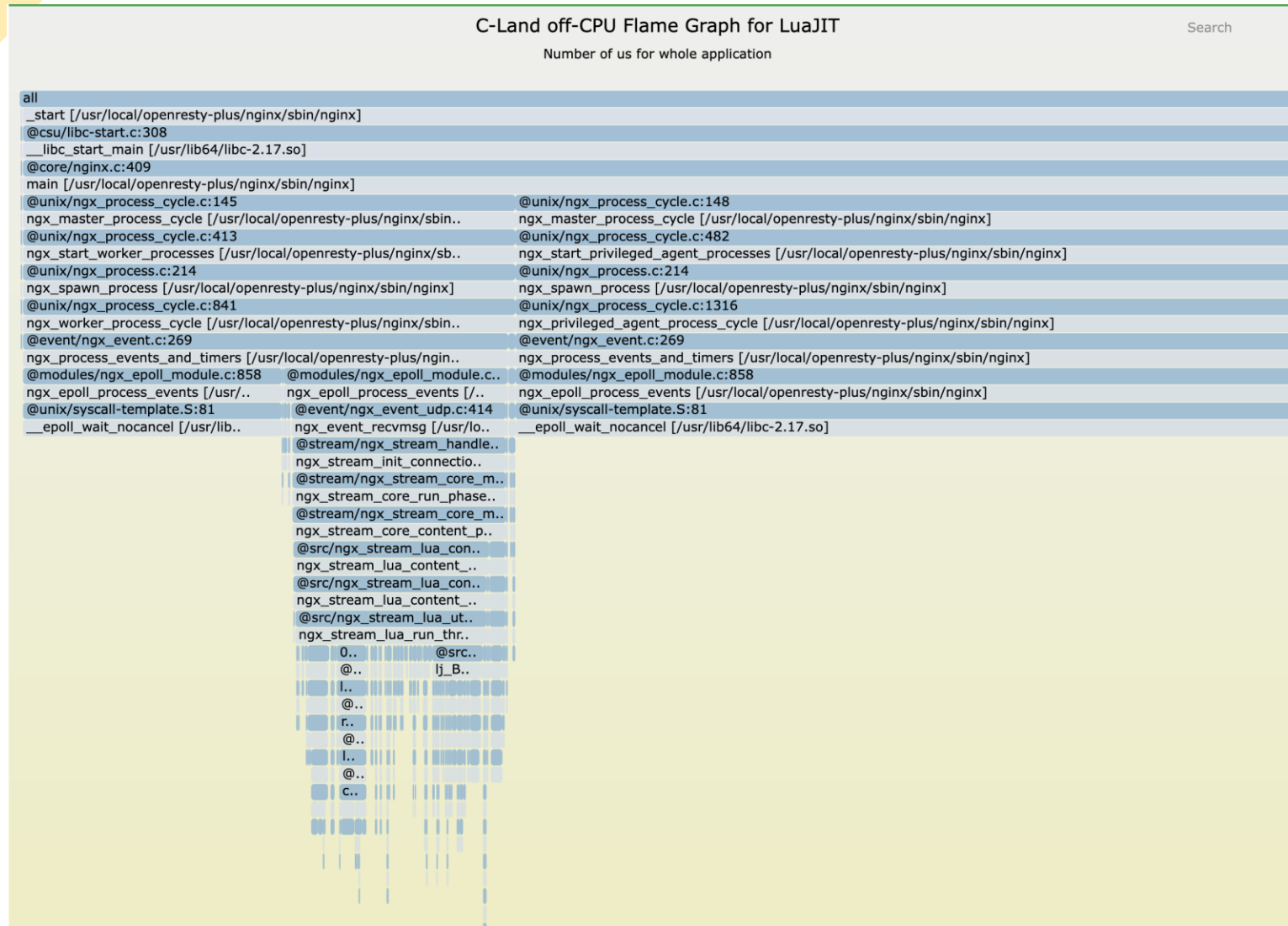
Lua-Land CPU Flame Graph

Search

whole application



OpenResty XRay CPU 阻塞 (off-CPU) 分析

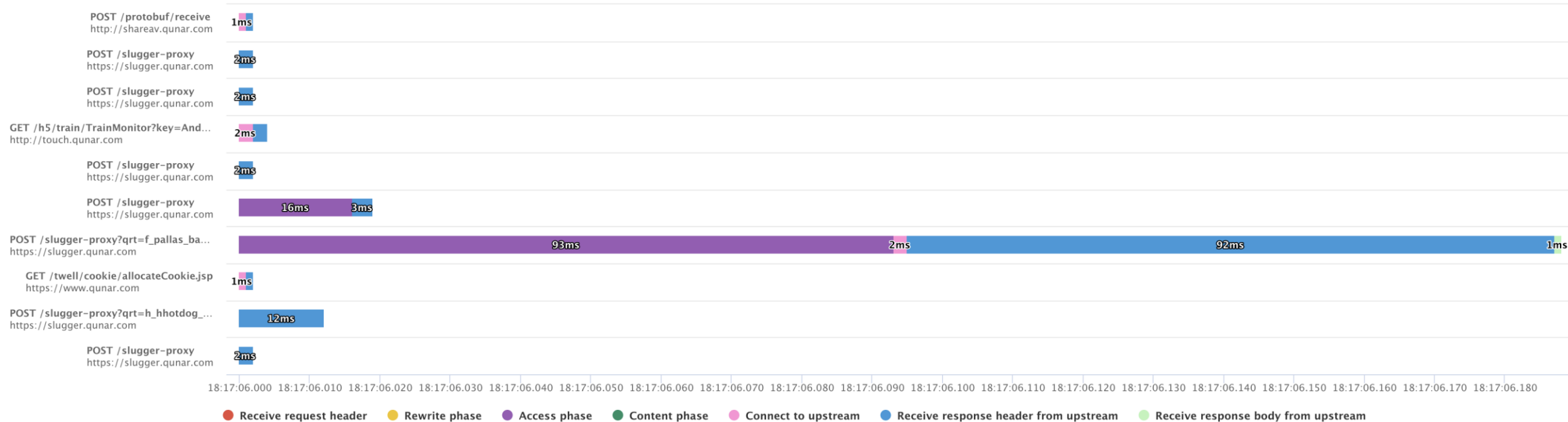


OpenResty XRay 請求延時分析

- 分解延時到不同應用操作和處理階段
- 精準抓包，只抓實際有問題的連線上的網路包（問題包括延時較高、超時、連線錯誤、上層應用報錯等等）
- 非同步非阻塞 IO 的延時統計（如 Lua 協程 yield 的時間在 Lua 程式碼路徑上的分佈）

Nginx Request Latency

In 0 seconds, total 10 requests, matched 10 requests

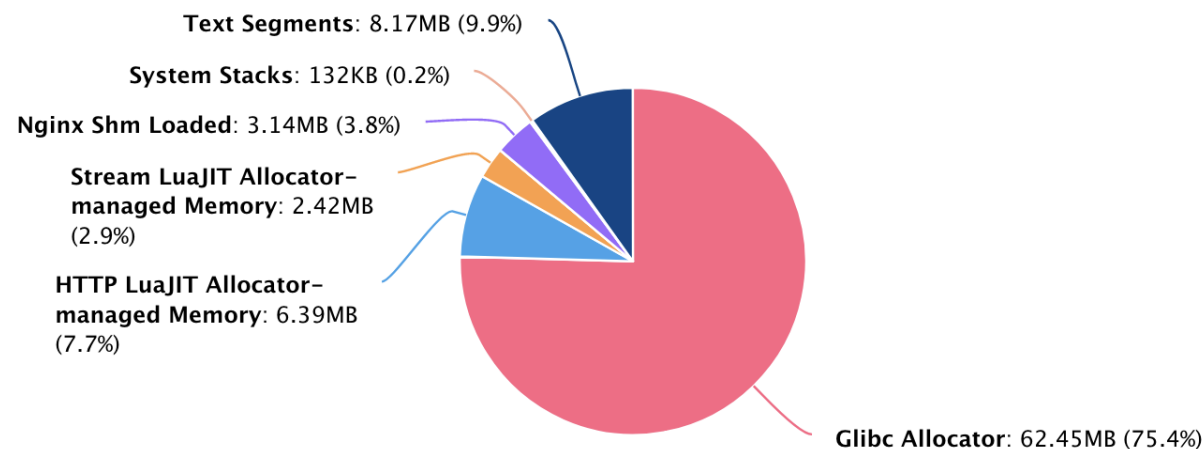


OpenResty XRay 記憶體使用分析

- Glibc/Jemalloc 等 C 記憶體分配器的記憶體使用（含 Glibc 記憶體碎片）
- 記憶體如何定量分佈在所有的 GC 物件上（比如 Lua 物件、Python 物件、PHP 物件等等），按 GC 物件之間的引用關係。
- 記憶體洩漏，記憶體碎片，還是延遲釋放？

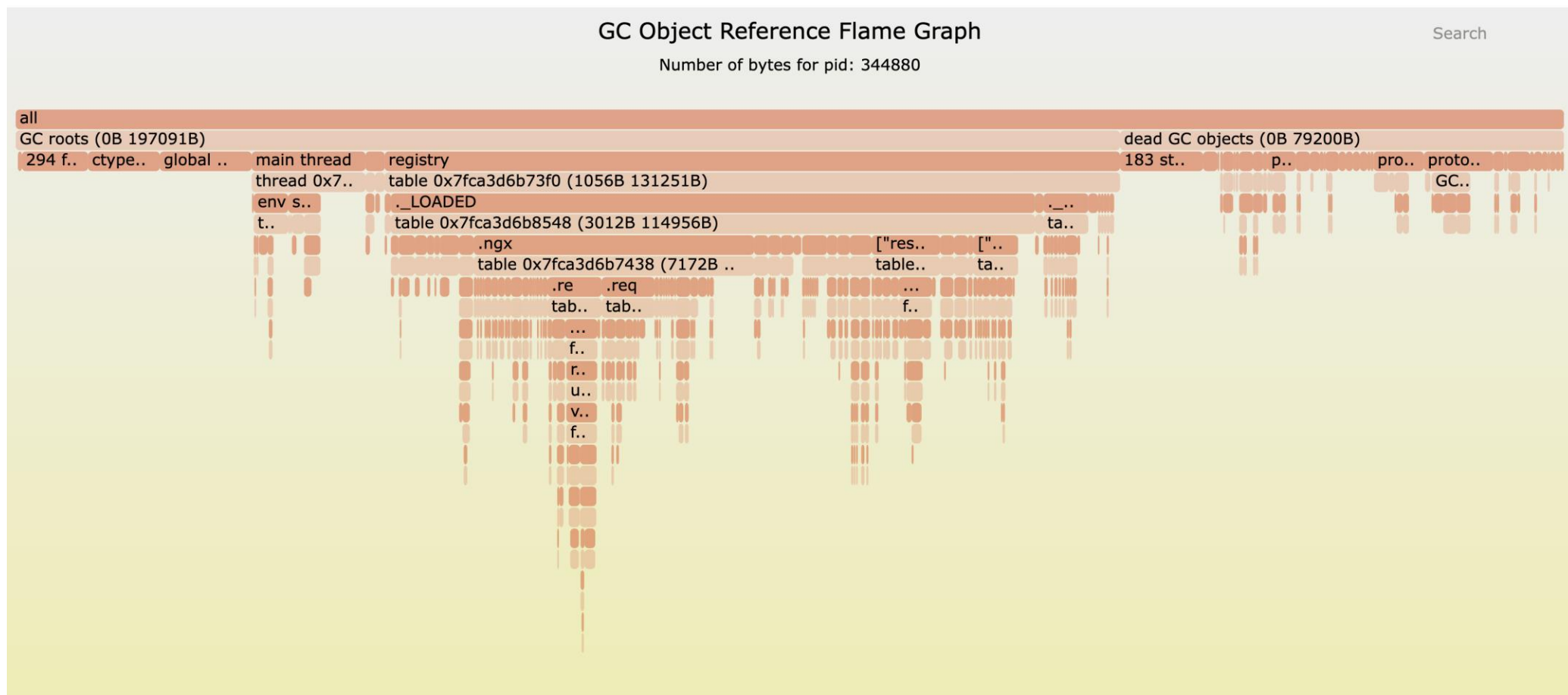
Application-Level Memory Usage Breakdown

02/04/2023 20:45, Total: 82.78MB



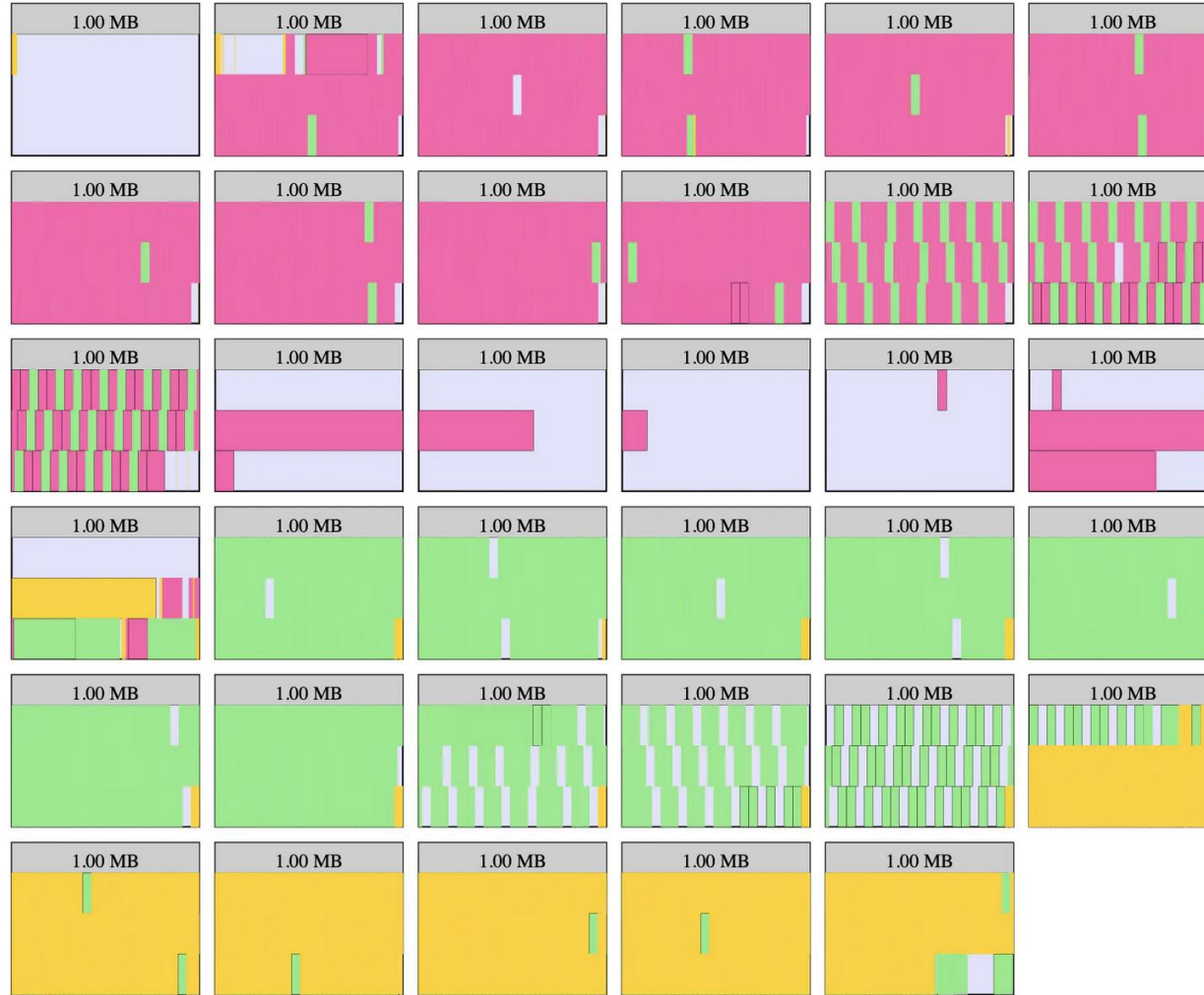
GC 物件引用關係火焰圖

定量顯示記憶體是如何在所有的物件引用路徑上分佈的



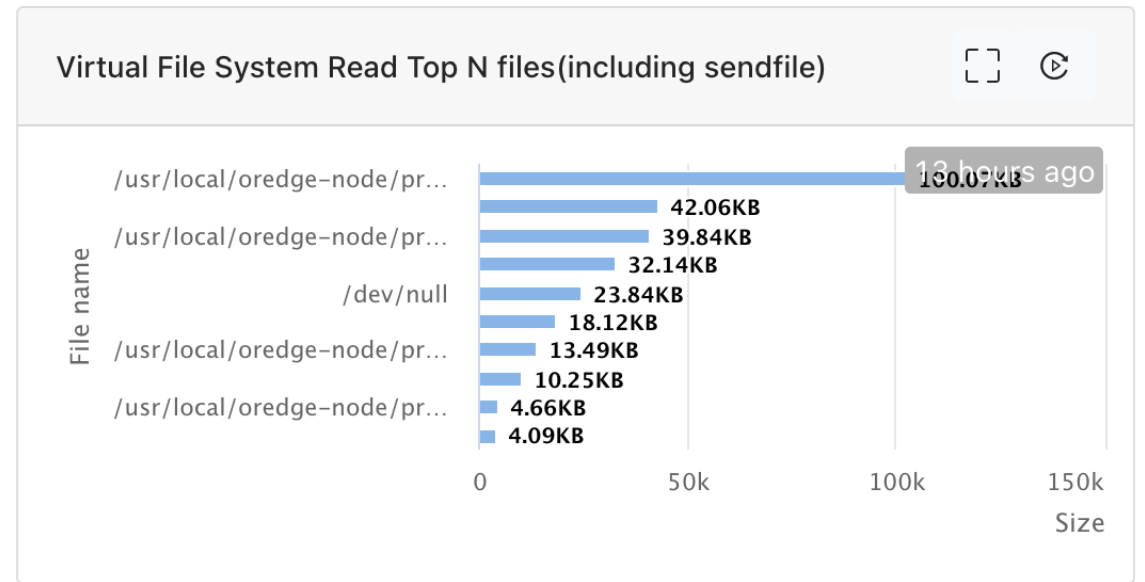
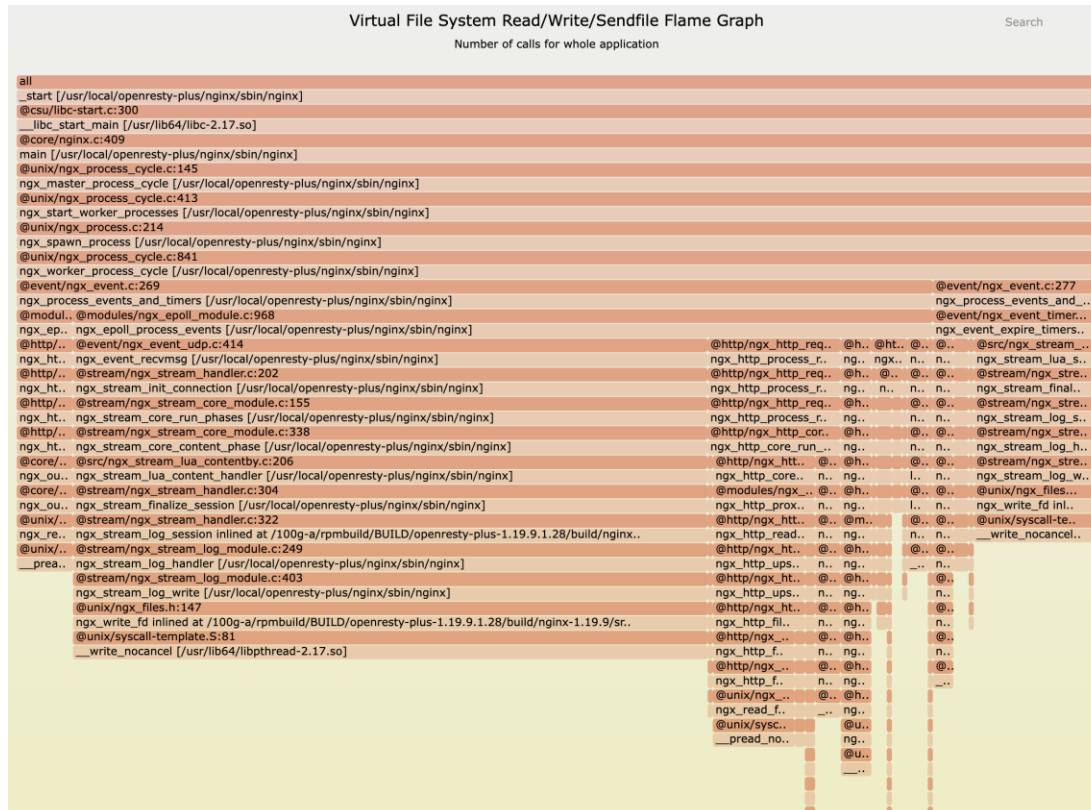
Nginx memory pool allocatons via the mmap syscall

For total 35.00 MB in 35 memory mappings with 49,464 chunks



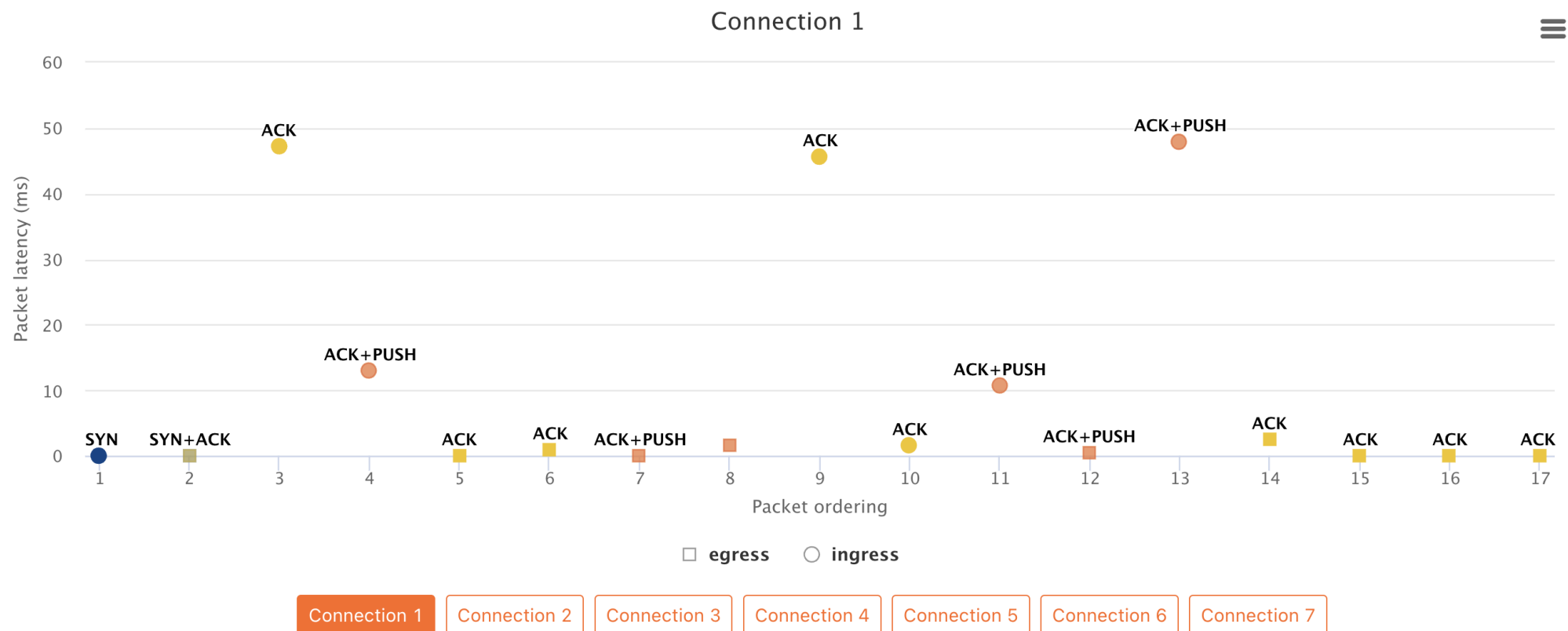
free other libc alloc
old cycle pool new cycle pool

OpenResty XRay 檔案 IO 效能分析



線上智慧網路抓包

只抓真正有異常的網路連線上的包



OpenResty XRay 自動分析診斷報告

Daily report Weekly report 2023-01-12

← Previous day

OpenResty XRay Analysis Report for Demo



See the full report →

Agent(s): 1

Online time: 23 hours

Watch time: 1 day

2023-01-04 16:00:00 ~ 2023-01-05 15:40:59

OpenResty ⓘ

CPU 25

off-CPU 1

Errors & Exceptions 4

Memory 16

CPU Usage: min: 0%, avg: 0.78%, max: 102%

Command line: nginx: master process /usr/local/openresty/nginx/sbin/nginx

Exe path: /usr/local/openresty/nginx/sbin/nginx

ⓘ CPU 25

▶ Lua code execution takes up to **100.00%** of the CPU time of the target processes. [More ↓](#)

Suggestions: To see how the CPU time is distributed across all the hot Lua code paths, we can analyze the *Lua-land CPU ...* [More ↓](#)

▶ **NEW** [10.1%] #1 of the hottest Lua code paths for CPU time: `lj_err_argt` ← `lj_lib_checkstr[10]` ← `lj_fff_fallback` ← `[builtin#strin` [More ↓](#)

▶ **NEW** [11.9%] #2 of the hottest Lua code paths for CPU time: `gc_onestep` ← `lj_gc_step` ← `lj_gc_step_jit` [More ↓](#)

▶ **NEW** [45.5%] #3 of the hottest Lua code paths for CPU time: `lj_BC_FUNCC` ← `print` ← `C:ngx_http_lua_ngx_print` [More ↓](#)

▶ The `string.gmatch()` Lua function calls takes up to **89.80%** of the CPU time of the target processes. [More ↓](#)

Suggestions: Try optimizing the CPU time usage on the code paths highlighted.

▶ **NEW** [29.2%] #1 of the hottest C-land code paths for CPU time: `__GI__writev` ← `ngx_writev` ← `ngx_linux_sendfile_chain` ← `ng` [More ↓](#)

閱讀自動分析和診斷報告的
部落格文章

自動記憶體問題診斷報告

i Memory 15

- NEW Glibc allocator takes up to **72.68 MB** in a target process. [More ↓](#)
- NEW Glibc arena takes up to **72.68 MB** in a target process. [More ↓](#)
- NEW In-use total memory in glibc arena takes up to **67.06 MB** in a target process. [More ↓](#)
- NEW Reserved free memory in glibc arena takes up to **5.67 MB** in a target process. [More ↓](#)
- NEW Free memory reserved by the LuaJIT allocator takes up to **12.19 MB** in a target process. [More ↓](#)
- NEW In-use total memory by the LuaJIT allocator takes up to **4.26 MB** in a target process. [More ↓](#)
- NEW Lua GC size of all types takes up to **1.45 MB** in a target process. [More ↓](#)
- NEW [**10.7%**] #1 of the hottest reference paths for LuaJIT GC object: `table 0x7f950c0a0828 (584B 524.40KB)` ← `[light userdata]` [More ↓](#)
- NEW [**12.4%**] #2 of the hottest reference paths for LuaJIT GC object: `trace 0x7f950a93a8e8 (2.36KB 161.13KB)` ← `next side` ← [More ↓](#)

自動延時分析與診斷

Latency 5

- ▼ ↑ 22.2% [100%] #1 of the hottest Lua code paths for Newly Created CoSocket: C:ngx_http_lua_socket_tcp_connect ← connect ← C:ngx_http_lua_socket_tcp_connect ← check_peer ← spawn_checker ← check_peers ← pcall ← [builtin#pcall]
See [Job 4418885005](#) for more details.
[Collapse ↑](#)
- ▶ [100%] #2 of the hottest Lua code paths for Newly Created CoSocket: C:ngx_http_lua_socket_tcp_connect ← connect ← C:ngx_http_lua_socket_tcp_connect ← _request ← request_admin ← pcall ← [builtin#pcall] [More ↓](#)
- ▶ ↑ 105.67 ms [106.97 ms] #1 of the hottest Lua code paths for Request Yield Latency: lua_yield ← lj_BC_FUNCC ← ngx_sleep ← C:ngx_http_lua_ngx_sleep ← limit_req_rate ← helper_1 ← xpcall ← [builtin#xpcall] ← run_rewrite_phase ← access ← access_by_lua(nginx.conf:586) [More ↓](#)
- ▶ ↑ 4.00 ms [16.00 ms] #2 of the hottest Lua code paths for Request Yield Latency: lua_yield ← ngx_stream_lua_socket_tcp_receive ← lj_BC_FUNCC ← receive ← C:ngx_stream_lua_socket_tcp_receive ← go ← content_by_lua(nginx.conf:140) [More ↓](#)
- ▶ [1.58 ms] #2 of the hottest Lua code paths for Request Yield Latency: lua_yield ← lj_BC_FUNCC ← ngx_sleep ← C:ngx_stream_lua_ngx_sleep ← limit_req ← process_req ← go ← content_by_lua(nginx.conf:132) [More ↓](#)

off-CPU 自動診斷報告

i off-CPU 8

- ▶ NEW Lua code execution takes up to **99.99%** of the off-CPU time of the target processes. [More ↓](#)

Suggestions: Try optimizing the off-CPU time usage on the code paths highlighted.

- ▶ NEW [21.5%] #1 of the hottest Lua code paths for off-CPU time: `__read_nocancel` ← `_IO_file_underflow@@GLIBC_2.2.5[2]` ← `_IO_default_xsgetn[2]` ← `lj_BC_FUNCC` ← `read` ← `[builtin#` [More ↓](#)
- ▶ NEW [31.7%] #2 of the hottest Lua code paths for off-CPU time: `__read_nocancel` ← `fread[2]` ← `lj_BC_FUNCC` ← `read` ← `[builtin#io.method.read]` ← `_getAccessLog` ← `_getIfNumber` [More ↓](#)
- ▶ NEW [32.6%] #3 of the hottest Lua code paths for off-CPU time: `__read_nocancel` ← `lj_BC_FUNCC` ← `read` ← `[builtin#io.method.read]` ← `_getProxyIgnoreHeaderInfo` ← `_getIfNumber` [More ↓](#)

異常錯誤自動診斷報告

i Errors & Exceptions 2

- ▶ NEW [100%] #1 of the hottest Lua code paths throwing out Lua exceptions: 71) ← no field package.preload['test'] ← no file '/' [More ↓](#)
- ▶ NEW [100%] #2 of the hottest Lua code paths throwing out Lua exceptions: 166) ← no field package.preload['resty.http'] ← r [More ↓](#)

安全問題自動分析

自動檢查和報告未開啟 TLS 加密的連線

動態掃描未開啟證書來源校驗的 TLS 連線

檢查使用了非安全的 SSL 協議版本

掃描遠端 shell 命令執行的事件與程式碼上下文



Core Dump 程序遺骸分析 (程序崩潰)

All

Core Dump Analysis

Application Type *

OpenResty

Analyzer *

openresty-core-dump-analysis (OpenResty Core Dump Analysis)

Core File *

/tmp/core.3859164

Executable File Path *

/usr/local/openresty/nginx/sbin/nginx

► Advanced Settings

Core File Meta Data

File Name: /tmp/core.3859164

Executable File Path: /usr/local/openresty/nginx/sbin/nginx

Size: 14.72MB

Analyze

從 Core Dump 檔案提取深層資訊

Analysis 7173425 [↗](#)

(gdb) lbt

```
C:ngx_md5_body
trace#1:access.lua:4
check_token
@/usr/local/openresty/lualib/access.lua:3
auth
@/usr/local/openresty/lualib/access.lua:21
@access_by_lua(nginx.conf:51):2
```

(gdb) full_lbt

```
C:ngx_md5_body
trace#1:access.lua:4
check_token
@/usr/local/openresty/lualib/access.lua:3
auth
@/usr/local/openresty/lualib/access.lua:21
headers = "access"
token = "hello"
@access_by_lua(nginx.conf:51):2
```

Compiler Output

Analyzer Output

Graphs

(gdb) ngx_process_info

```
parent: 3859163
process: worker 0
```

(gdb) cur_http_req

```
current phase: access
schema: http, req_size: 52, resp_size: 0GET / HTTP/1.1
Host: localhost:80
TOKEN: hello
```

(gdb) ubt

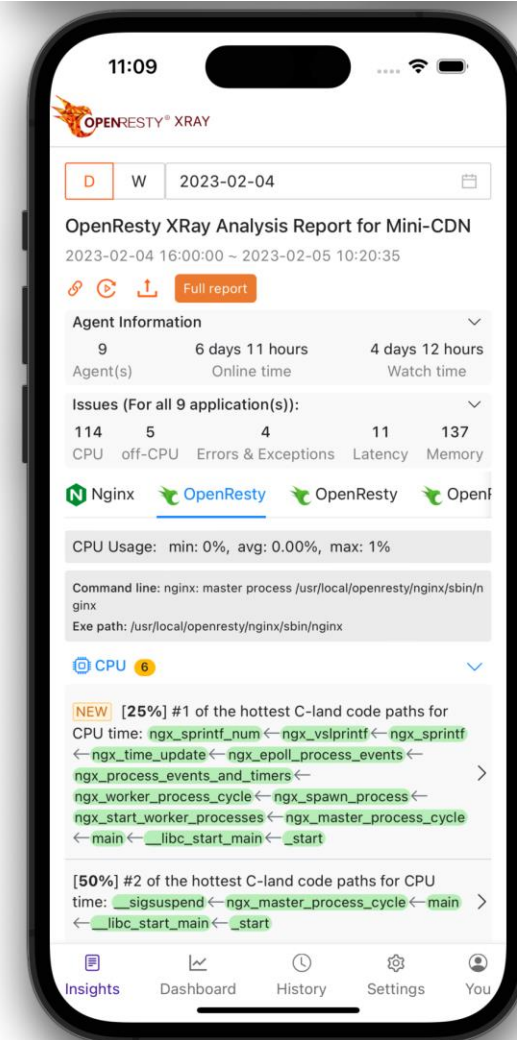
```
0x42e958 ngx_md5_body [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/src/core/ngx_md5.c:199]
0x42f1be ngx_md5_final [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/src/core/ngx_md5.c:91]
0x4ea997 ngx_http_lua_ffi_md5 [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/./ngx_lua-0.10.21/s
7f763447ffd3: []
0x4f86a1 ngx_http_lua_run_thread [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/./ngx_lua-0.10.2
```

OpenResty XRay

移動端 App

任何時候從任何地方察看您的線上應用

- Android (Google Play)
- iOS (蘋果商店)

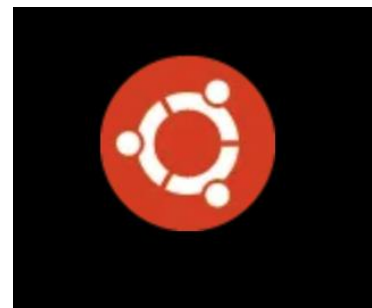


OpenResty XRay 並非只針對 OpenResty 應用

- Nginx, LuaJIT, OpenResty
- 初步支援：
Python, PHP, Perl, Redis, PostgreSQL
- 即將釋出：
Go, Ruby, NodeJS, Java



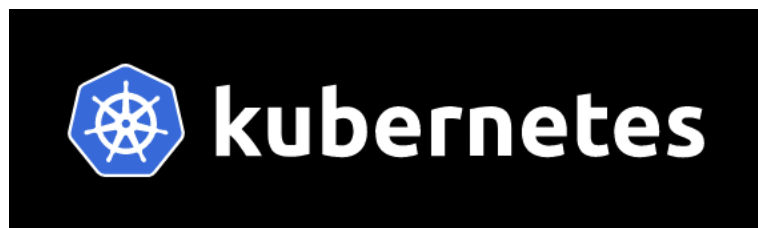
支援絕大多數主流 Linux 發行版和容器佈署方式



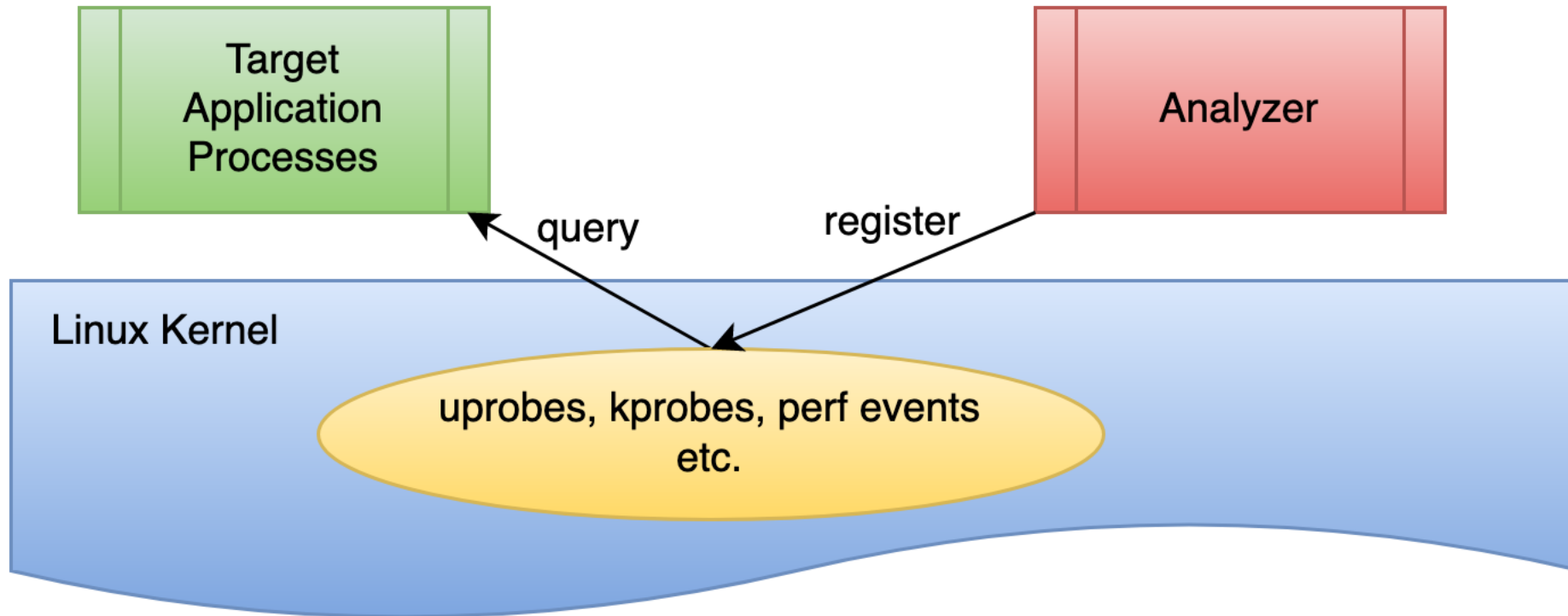
Ubuntu



Debian



OpenResty XRay 基於先進的動態追蹤技術



動態追蹤的優點

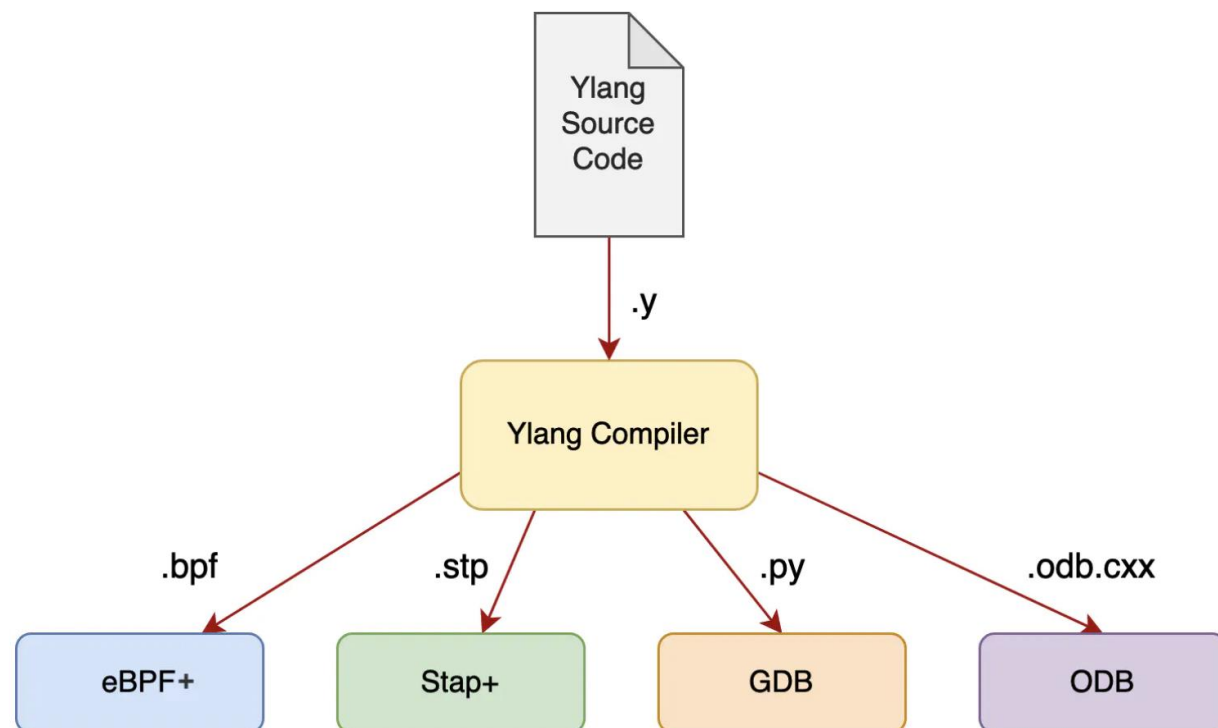
- 非侵入式，無需修改應用
- 熱插拔，通常無需應用配合（很多開源動態追蹤工具有時還是要求應用配合）
- 開銷通常較低，可以在資料來源進行聚合彙總
- 事發後的線上實時除錯能力
- 全技術棧分析無死角
- 按需取樣
- 不採樣時嚴格 0 損耗

OpenResty XRay 的全新一代動態追蹤技術

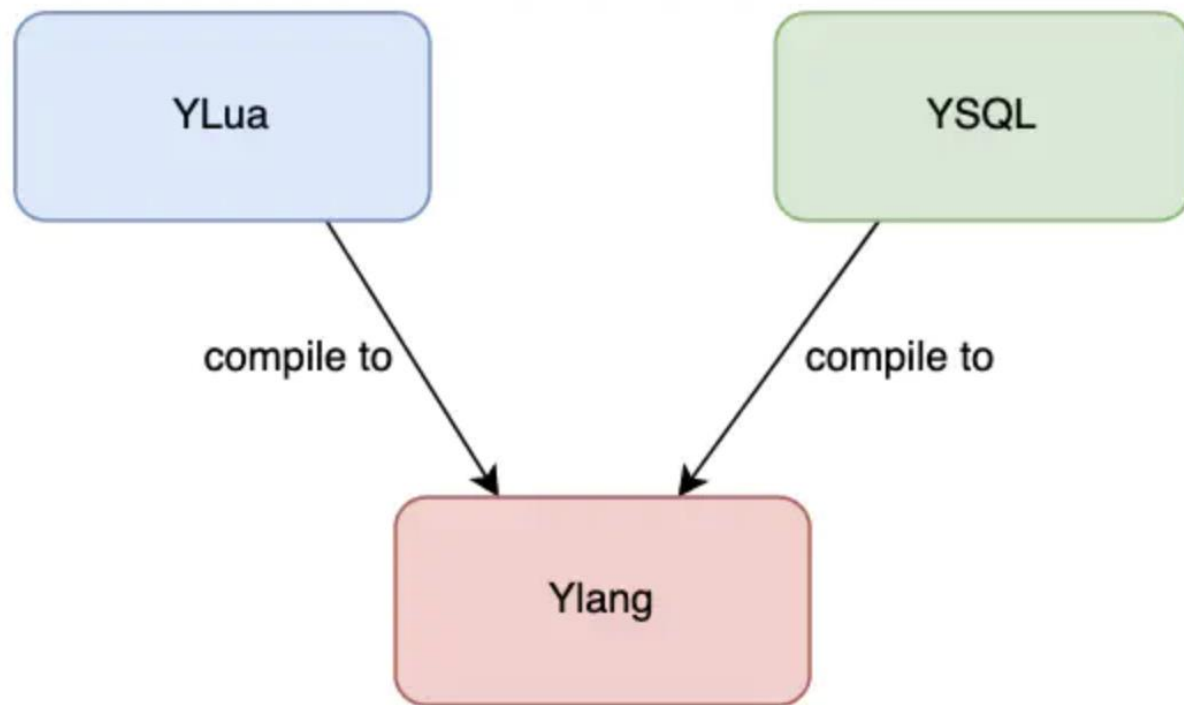
- Y 語言 (Ylang) 編譯器 (支援 GNU C 和標準 C 語言的絕大部分語法)
- Ylua 語言
- YSQL 語言
- Stap+ 顯著改進了 SystemTap
- eBPF+ 顯著改進了 eBPF (同時 LLVM+ 也大大改進了開源 LLVM)
- ODB 是超輕量版的 GDB
- Ylang 編譯器也能生成高度最佳化的 GDB 的 Python 擴充套件程式碼
- 針對線上生產環境的嚴格效能損耗控制

“一次編寫，到處執行”

從 Y 語言程式碼到各種不同執行時的程式碼



Y 語言之上更抽象的語言



在 OpenResty XRay 介面上編寫和除錯 Ylang/YLua/YSQL 等語言的分析工具

The screenshot displays the OpenResty XRay web interface. At the top, the header includes the OpenResty XRay logo, a version indicator (Version: 844), a dropdown menu for the current instance (prd.openresty.com (3...)), and user information (yichun@openresty.com, English). The main content area is titled "All analyzers / Edit analyzer" and features a sidebar on the left with navigation options: Overview, CPU, Memory, Disk, Network, Latency, Analyzers (highlighted), Load Gen, History, Events, Advanced, Custom Apps, Settings, and Upload.

The central editor area shows the configuration for an analyzer named "process-exit" with the description "C-land CPU Flame Graph". The language is set to YLang, and there are "Run" and "Save" buttons. The code editor contains the following YLang script:

```
1 // c-cpu: C-land CPU flame graph sampling tool.
2 // Copyright (C) OpenResty Inc.
3 // All rights reserved.
4
5 _target sig_atomic_t ngx_quit;
6 _target sig_atomic_t ngx_debug_quit;
7 _target ngx_uint_t ngx_exiting;
8 _target sig_atomic_t ngx_reconfigure;
9 _target sig_atomic_t ngx_reopen;
10
11 _probe _process.begin
12 {
13     printf("ngx_quit %ld\n", ngx_quit);
14     printf("ngx_exiting %ld\n", ngx_exiting);
15     printf("ngx_reconfigure %ld\n", ngx_reconfigure);
16     _exit();
17 }
18
19
```

Below the code editor are "Run" and "Clear the editor" buttons. A link for "Analysis history for this analyzer" is located at the bottom left of the editor area.

On the right side, there are several configuration panels:

- Try this analyzer against:** A section for selecting the target, with options for "By Applications", "By Processes", "By Executables", and "Whole System".
- Application:** A dropdown menu showing "openresty (PGID 1471, master proc)" and an "Update" button.
- Target Processes:** A dropdown menu showing "PID 2782 (worker process) CPU: 1%" and an "Update" button.
- Advanced Settings:** A section for advanced configuration options.
- YLang Settings:** A section for YLang-specific settings, including a note about dependent debug data and kernel debug info.

The "Kernel debug info" section has three radio buttons: "Not required", "Required", and "Good to have". The "Unwind data" section has a radio button set to "NO NEED" and a note: "Need unwind data when you need the backtrace."

OpenResty XRay 數百種標準分析器

▼ OpenResty XRay Standard Analyzers

- c-alloc-fgraph
- c-count-alloc-free
- c-memory
- c-memory-leak-fgraph
- c-off-cpu
- c-on-cpu
- collect-luajit-ffnames
- cpu-hogs
- epoll-loop-blocking-distr
- epoll-sched-latency-distr
- epoll-wait-ret-distr
- epoll-wait-timers
- epoll-wait-timers-fgraph
- file-system-fgraph
- func-latency-distr
- glibc-chunks
- jemalloc-bins

- kernel-on-cpu
- lj-add-timer-lua-fgraph
- lj-alloc-stats
- lj-c-memory-leak-fgraph
- lj-c-off-cpu
- lj-c-on-cpu
- lj-config
- lj-dump-loaded-mods
- lj-err-mem
- lj-excep-lua-fgraph
- lj-free-stats
- lj-gc-step-calls
- lj-lua-exception
- lj-gco-ref
- lj-gco-stat
- lj-lua-err-msg
- lj-lua-new-timer-errors
- lj-lua-newcdata
- lj-lua-newfunc
- lj-lua-newgco

- ngx-add-timer-event-fgraph
- lj-trace-stats
- ngx-add-timer-event-timer-distr
- lj-vm-states
- mmap-leaks
- musl-libc-chunks
- ngx-access-log-buffer-size
- ngx-config
- ngx-config-servers
- ngx-cpu-hottest-hosts
- ngx-cpu-hottest-uris
- ngx-downstream-keepalive-stats
- ngx-dump-req
- ngx-dump-timers
- ngx-epoll-wait-timers
- ngx-err-log-lvl-distr

除錯符號

- OpenResty XRay 有一箇中央包資料庫，索引了幾百 TB 的公開包的除錯符號，仍在快速增長
- 目標機器無需安裝或儲存除錯符號，只要除錯符號曾被 OpenResty XRay 中央包資料庫索引
- 無法找到除錯符號或編譯時未生成除錯符號的程式，OpenResty XRay 將有能力自動重建除錯符號（已有可工作的機器學習演算法的原型實現）



為眾多企業客戶所信任



Qunar.Com

zoom

瞭解更多

OpenResty XRay 常見問答

訪問 OpenResty XRay 官方部落格

訪問 OpenResty XRay 官方主頁

試用 OpenResty XRay 產品