Introduction to OpenResty XRay

Deep monitoring, analyses and diagnoses for your online applications

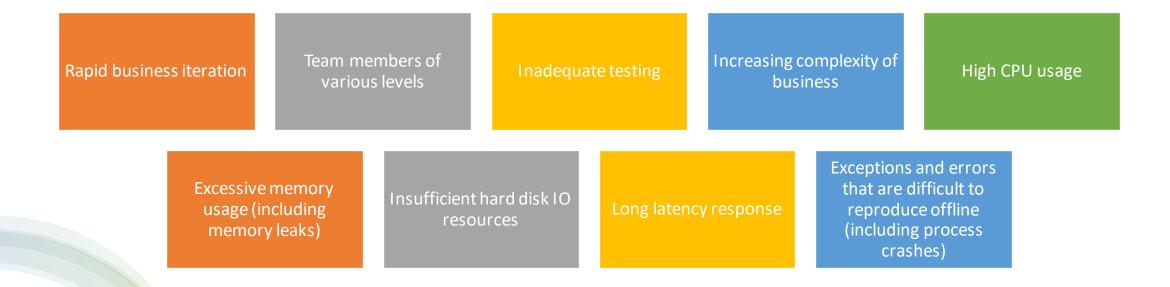


Try OpenResty XRay for free

https://openresty.com/en/xray/

info@openresty.com

Challenges in the World of Software



Challenges in the Age of K8s/Docker Containers Lots of containers, lots of applications, lots of distributions, lots of technology stacks

Minimized containers lack the most basic debugging tools

Minimized set of container permissions

Automatically discard and restart containers when something goes wrong. Software bugs are easily swept under the carpet

Virtualized containers, Microservices - further increase the software complexity

Disadvantages of Traditional Methods

Invasive - need to modify applications

Slow response

Need the storage and analysis of big data

Superficial indicators only

Observations without causes

Lack of in-depth full technology stack analysis and diagnosis

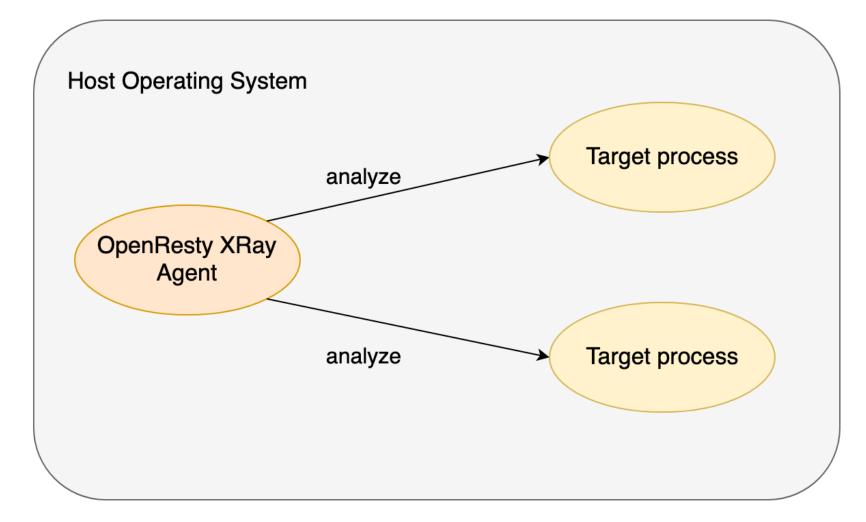
Complex data collection and processing process, high overhead and errorprone



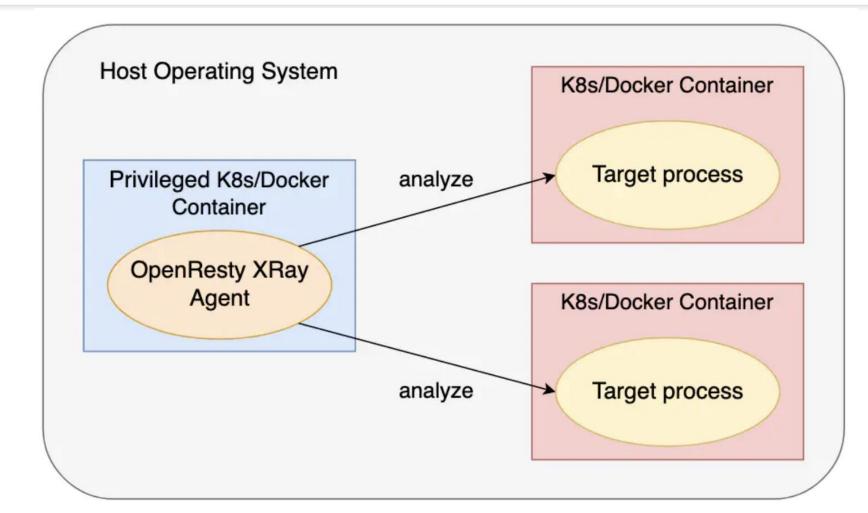
OpenResty XRay

- OpenResty XRay is a dynamic tracing product
- Enables real-time analysis of various cloud and server applications
- Treat running processes and containers as read-only databases and extract the necessary information to resolve performance issues, exceptions, errors, and security vulnerabilities
- With a knowledge base, inference engine, and hundreds of advanced analyzers
- Can diagnose and narrow down the root cause of deep problems without changing or affecting the target application

OpenResty XRay Analyzes Non-container Application Processes Directly

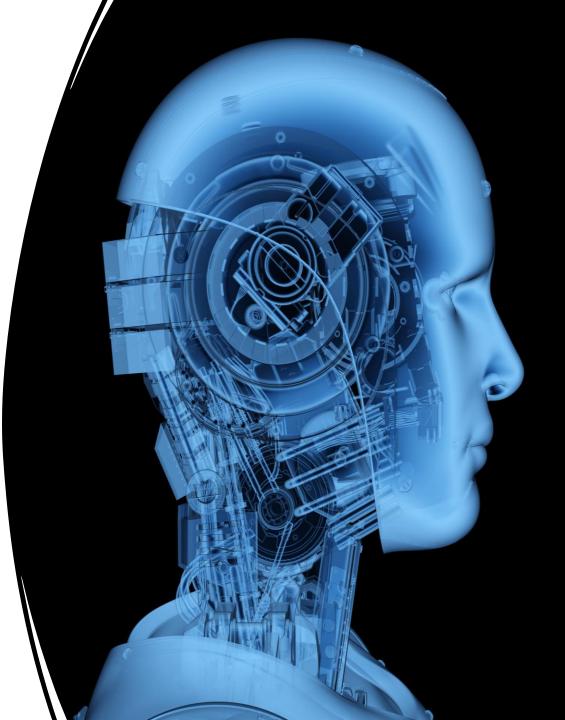


OpenResty XRay Penetrates Containers and Analyzes Applications



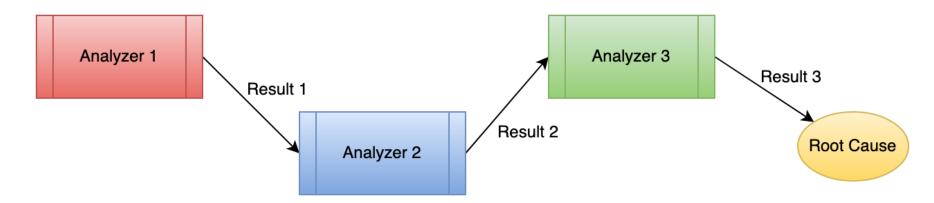
100% Non-Invasive

- No need to modify your application
- No need to add new plug-ins, modules, or patches to your application
- No need to inject any code into your applications
- No need to restart your application processes
- No need to use special startup or compilation options in your application
- No need to rebuild your existing application containers or application packages



OpenResty XRay Fully Automatic Sampling, Unattended Usage Mode

- Time sampling
- Event-driven sampling (CPU changes, memory changes, IO changes, exceptions and errors)
- Reasoning chains driven



Extremely Low Performance Overhead

- Performance overhead is strictly 0 when not sampling
- Performance overhead is often not noticeable when sampling



OpenResty XRay CPU Performance Analysis

- High CPU usage can reduce the system stability and quality of service, and even make services unavailable
- How CPU time is distributed over different code paths in different scenarios (flame graphs, automatic flame graph interpreter)
- Cover code paths of different software levels: business programming language level (Lua/Python/PHP/Perl/Go/etc.), system programming language level (C/C++/Rust), OS kernel level (network protocol stack/process scheduler/memory management/system calls)
- Examples of common CPU bottlenecks: duplicate computations (lack of cache), SSL handshake related, garbage collection (GC) overhead, dynamic memory allocation overhead, serialization and deserialization, unexpectedly frequent system calls, infinite loops, wrong regular expression matching, (third-party) software libraries with inefficient implementation, spinlock contention

	Sea	rch									
whole application											
all											
	_start [/usr/local/openresty-plus/nginx/sbin/nginx]										
	start.c:300										
	rt_main [/usr/lib64/libc-2.17.so]										
@core/ng											
	r/local/openresty-plus/nginx/sbin/nginx]										
	x_process_cycle.c:145	n al aud									
	er_process_cycle [/usr/local/openresty-plus/nginx/sbin/ x_process_cycle.c:413	nginxj									
	ngx_start_worker_processes [/usr/local/openresty-plus/nginx/sbin/nginx] @unix/ngx_process.c:214										
	(@unix/ngx_process.ct:214 ngx_spawn_process/(usr/local/openresty-plus/nginx/sbin/nginx]										
	<pre>x_process_cycle.c:841</pre>										
	er process cycle [/usr/local/openresty-plus/nginx/sbin/	nainx1									
	gx_event.c:269						@event/ngx_event.	c @e			
ngx_proce	ess_events_and_timers [/usr/local/openresty-plus/nginx	/sbin/ng	inx]				ngx_process_events	s ng			
	@modules/ngx_epoll_module.c:968						@event/ngx_event.				
ngx_epo.	ngx_epoll_process_events [/usr/local/openresty-plus/	nginx/sl	bin/nginx]				ngx_event_expire				
@unix/			@http/ngx_http_requ		@http/ngx_h	@http/ngx_http_upstr	@s @ @src				
epol	ngx_event_recvmsg [/usr/local/openresty-pl		ngx_http_request_ha		ngx_http_ss	ngx_http_upstream_ha	ng n ngx				
	@stream/ngx_stream_handler.c:202		@http/ngx_http_u		@event/ngx	@http @http/ngx	@h @ @src				
	ngx_stream_init_connection [/usr/local/ope		ngx_http_upstrea		ngx_ssl_han	ngx_h ngx_http	ng n ngx				
	@stream/ngx_stream_core_module.c:155 @ht. @event/ngx_even @event @ssl/ssl_li @even @event/n										
	ngx_stream_core_run_phases [/usr/local/ope ngx ngx_ssl_write [ngx_ss SSL_do_hand ngx_s ngx_ssl										
	@stream/ngx_stream_core_module.c:338 @ht @ssl/ssl_llb.c: @ssl/s @statem/ @ssl/ssl										
	ngx_stream_core_content_phase [/usr/local/ ngx SSL_write [/usr SSL_re state_ma SSL_w SSL_read										
	@src/ngx_stream_lua_contentby.c:202	@h	@ssl/ssl_lib.c:	@ssl/s	@statem	@ssl/ @ssl/ssl					
	ngx_stream_lua_content_handler [/usr/lo	ng	ssl_write_inter	ssl_re	write_s	ssl_w ssl_read					

Lua-Land CPU Flame Graph

Search

whole application

Search

II		@ livel	@	@L			hus hus farshess a								 	1
		@./lua/ @@./	@access @./lua	(@D		@	_by_lua(nginx.c @./lua/dns.lua		2	@./lua/d	an lunidd			@l	run_rewrite_p	log co
								1:40			15.108:44					co
		d do	access	pc	go	90	go			go	O fine (des see		0.05	log	[builtin#xpc	
		t @./	@1:1	@b			@dns:ope					ime-20181201.lua:	885		xpcall	
_		get	run	@		_	run_rewr	[builtin:	¥	proce	process_resp				do_acti	
		@	[bui	c			[builtin	xpcall		C:ngx	trace#26	trace#288:	t		run_wa	1.111
_		ru	xpcall	@			xpcall	g., ł	h	send	app_dyme	app_dymet	a		 run_wa	
		@	@	s			@d			@src/	./usr/l	./usr/loc			run_ru	111
		pi	@	@						lj_BC	./lua	./lua/o			run_w	
01		tr	d	c		1.1				./us					run_r.	
		./	@	C		1.0				@uni					[buil	
Ш.		@1	r.,	c		11				se					xpcall	
н.			@	@												
ΓĮ.	ш		n.,	1												

OpenResty XRay CPU Blocking (off-CPU) Analysis

C-Land off-CPU Flame Graph for LuaJIT

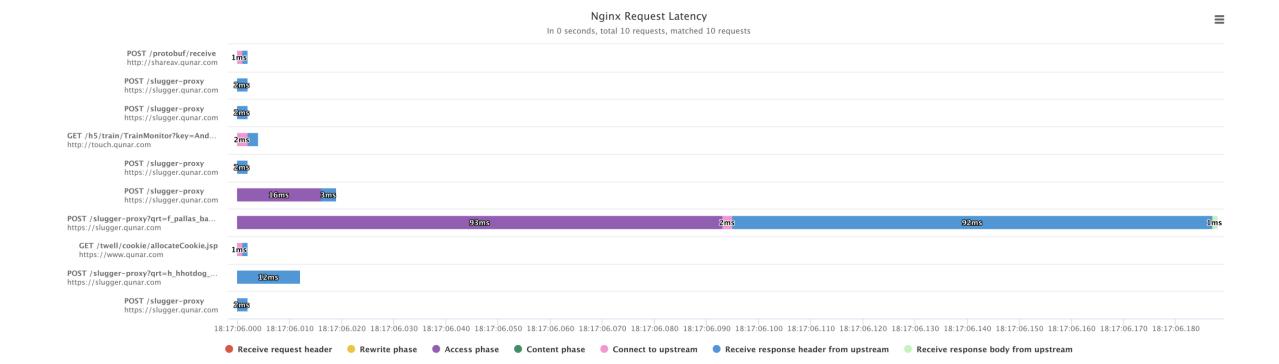
Number of us for whole application

Search

all _start [/usr/local/openresty-plus/ngin	vy/chin/nginy]						
@csu/libc-start.c:308							
libc_start_main [/usr/lib64/libc-2.1	7 col						
@core/nginx.c:409	.7.50]						
	/chip/painy]						
main [/usr/local/openresty-plus/ngin>	k/sbin/nginxj						
@unix/ngx_process_cycle.c:145	lanonroch, plus (nginy (shin	@unix/ngx_process_cycle.c:148 ngx_master_process_cycle [/usr/local/openresty-plus/nginx/sbin/nginx]					
ngx_master_process_cycle [/usr/loca	l/openresty-plus/nginx/sbin						
@unix/ngx_process_cycle.c:413	all (an annach i cline (a sinu (ab	@unix/ngx_process_cycle.c:482					
ngx_start_worker_processes [/usr/log	cal/openresty-plus/nginx/sb	ngx_start_privileged_agent_processes [/usr/local/openresty-plus/nginx/sbin/nginx]					
@unix/ngx_process.c:214	eactur plus (painy (ship (painy))	@unix/ngx_process.c:214					
ngx_spawn_process [/usr/local/openr	resty-plus/nginx/sbin/nginx]	ngx_spawn_process [/usr/local/openresty-plus/nginx/sbin/nginx]					
@unix/ngx_process_cycle.c:841		@unix/ngx_process_cycle.c:1316					
ngx_worker_process_cycle [/usr/local	i/openresty-plus/nginx/sbin	ngx_privileged_agent_process_cycle [/usr/local/openresty-plus/nginx/sbin/nginx]					
@event/ngx_event.c:269		@event/ngx_event.c:269					
ngx_process_events_and_timers [/us		ngx_process_events_and_timers [/usr/local/openresty-plus/nginx/sbin/nginx]					
@modules/ngx_epoll_module.c:858	@modules/ngx_epoll_module.c	@modules/ngx_epoll_module.c:858					
ngx_epoll_process_events [/usr/	ngx_epoll_process_events [/	ngx_epoll_process_events [/usr/local/openresty-plus/nginx/sbin/nginx]					
@unix/syscall-template.S:81	@event/ngx_event_udp.c:414	@unix/syscall-template.S:81					
epoll_wait_nocancel [/usr/lib	ngx_event_recvmsg [/usr/lo	epoll_wait_nocancel [/usr/lib64/libc-2.17.so]					
	@stream/ngx_stream_handle						
	ngx_stream_init_connectio						
	@stream/ngx_stream_core_m						
	ngx_stream_core_run_phase						
	<pre>@stream/ngx_stream_core_m ngx_stream_core_content_p</pre>						
	@src/ngx_stream_lua_con						
	ngx_stream_lua_content						
	@src/ngx_stream_lua_con.						
	ngx_stream_lua_content						
	@src/ngx_stream_lua_ut						
	ngx_stream_lua_run_thr						
	0 @src						
	@ lj_B						
	@						
	@						
	@						
	C						

OpenResty XRay Analysis of Request Latency

- Breakdown the latency to different operation and processing phases of applications
- Precise packet capture, only capture network packets on problematic connections (including high latency, timeouts, connection errors, when upper layer applications report errors, etc.)
- Latency statistics of asynchronous non-blocking IO (e.g. the distribution of Lua concurrent yield time over Lua code paths)

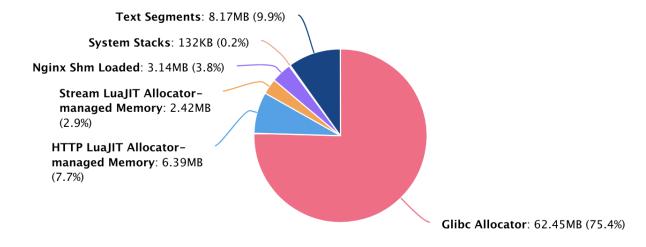


OpenResty XRay Memory Usage Analysis

- Memory usage of C memory allocators such as Glibc/Jemalloc (including Glibc memory fragmentation)
- How memory is distributed quantitatively over all GC objects (e.g. Lua objects, Python objects, PHP objects, etc.), by reference relationships between GC objects.
- Memory leak, memory fragmentation, or delayed release?

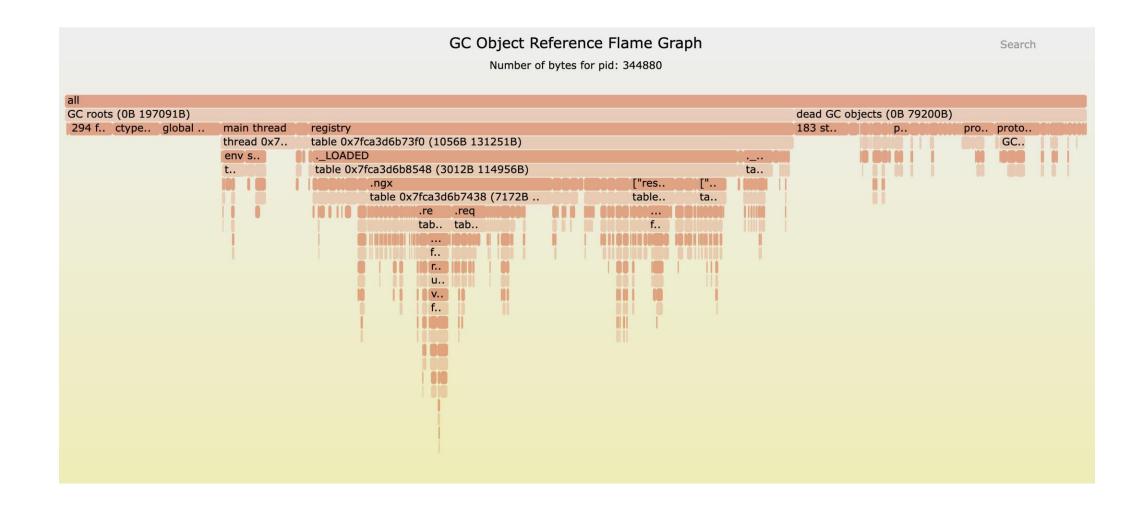
Application-Level Memory Usage Breakdown

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



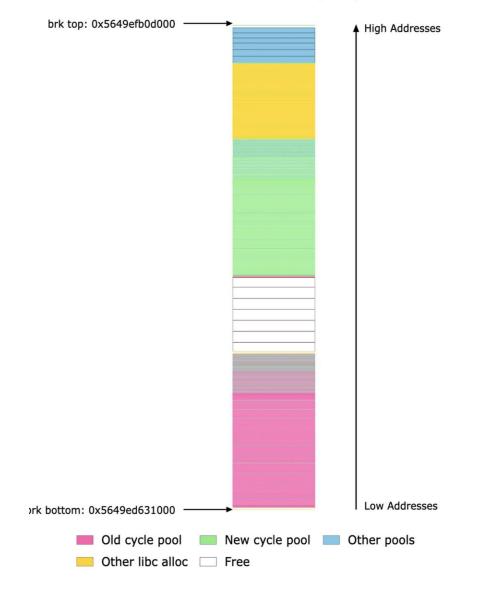
GC Object Reference Relationship Flame Graph

Quantitative visualization of how memory is distributed over all object reference paths



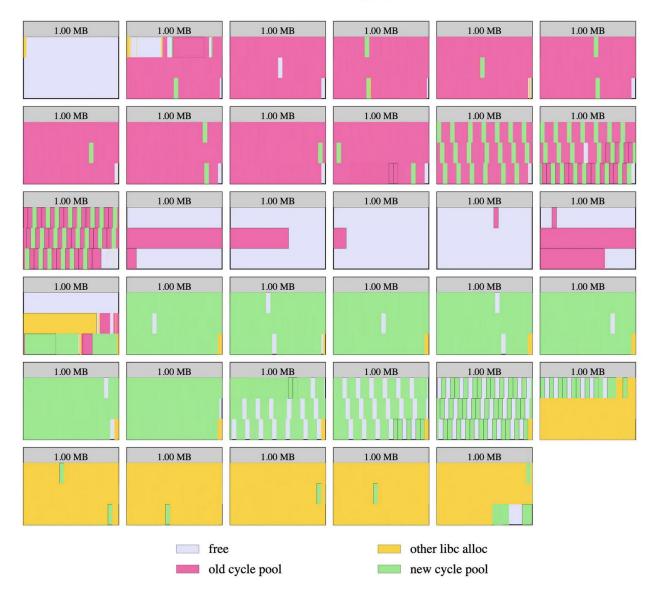
Nginx memory pool allocatons via the brk syscall

When the old and new nginx cycles coexist



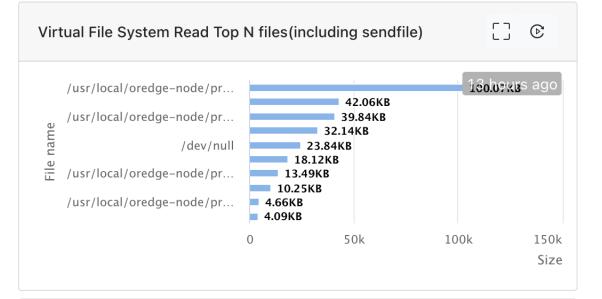
Nginx memory pool allocatons via the mmap syscall

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



OpenResty XRay File IO Performance Analysis

Virtual File System Read/Write/Sendfile File Number of calls for whole application	ame Graph	Search
	<pre>@http/ngx_http_reg. @h. @ht.@ ngx_http_process_r. ng. ngx. n. @http/ngx_http_reg. @h. @. @ ngx_http_process_r. ngn. r. @http/ngx_http_reg. @h @ ngx_http_core_nu ngn. n. @http/ngx_http_core_nu ngn. n. @http/ngx_http_core_nu ngn @ngx_http_core_n. ng @ngx_http_core. n. ng @ngx_http_prox. n. ng @ngx_http_prox. n. ng @ngx_http_red. n. ng @http/ngx_htt. @m. @m @m.</pre>	
	@u 	

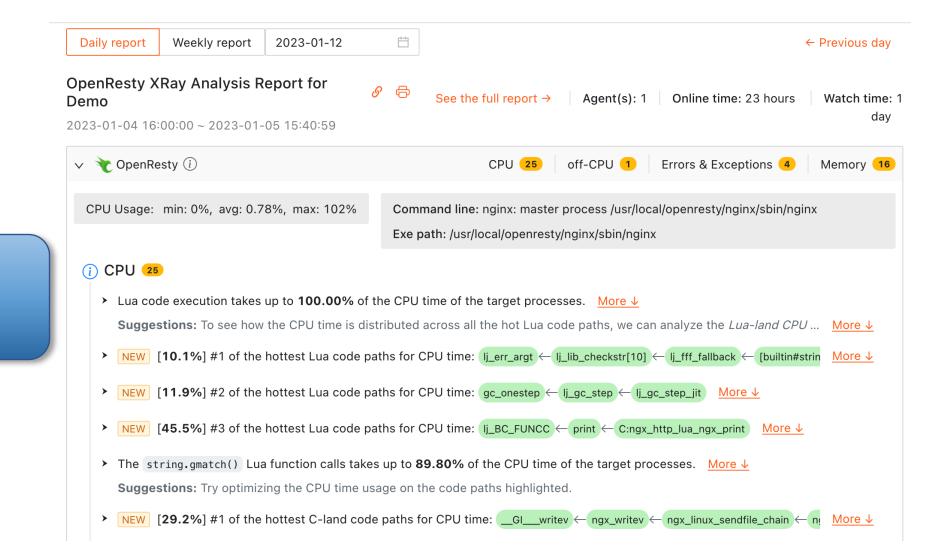


Online intelligent network packet capture

Only catch packets on abnormal network connections



OpenResty XRay Automated Analysis Diagnostic Reports



Read blog posts about automated analysis and diagnostic reports

Automatic Memory Problem Diagnostic Reports

i Memory 15

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

Automatic Latency Analysis and Diagnosis

) La	atency 5
¥	↑ 22.2% [100%] #1 of the hottest Lua code paths for Newly Created CoSocket: C:ngx_http_lua_socket_tcp_connect ← connect ←
	$C:ngx_http_lua_socket_tcp_connect \leftarrow check_peer \leftarrow spawn_checker \leftarrow check_peers \leftarrow pcall \leftarrow [builtin#pcall]$
	See Job 4418885005 for more details.
	<u>Collapse ↑</u>
>	$[100\%] #2 of the hottest Lua code paths for Newly Created CoSocket: C:ngx_http_lua_socket_tcp_connect \leftarrow c:ngx_http_lua_socket_tcp_connec$
>	↑ 105.67 ms [106.97 ms] #1 of the hottest Lua code paths for Request Yield Latency: $ ua_y = ua_y $
	$limit_req_rate \leftarrow helper_1 \leftarrow xpcall \leftarrow [builtin#xpcall] \leftarrow run_rewrite_phase \leftarrow access _ by_lua(nginx.conf:586) \qquad \underline{More \downarrow}$
>	↑ 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
	$\leftarrow C:ngx_stream_lua_socket_tcp_receive \leftarrow go \leftarrow content_by_lua(nginx.conf:140) \qquad \underline{More \downarrow}$
>	$[1.58 ms] #2 of the hottest Lua code paths for Request Yield Latency: lua_yield \leftarrow lj_BC_FUNCC \leftarrow ngx_sleep \leftarrow C:ngx_stream_lua_ngx_sleep \leftarrow limit_req \leftarrow$
	process_req ← go ← content_by_lua(nginx.conf:132) - More ↓

off-CPU Automatic Diagnostic Report

(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] ← Ij_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 ← Ij_BC_FUNCC ← read ← [builtin#io.method.read] ← _getProxylgnoreHeaderInfo ← _getIfNumber More ↓

Automatic Diagnostic Reports on Errors and Exceptions

(i) Errors & Exceptions 2

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

- no field package.preload['resty.http'] ← r More ↓

Automatic Analysis of Security Issues

Automatic checking and reporting of connections without TLS encryption

Dynamically scanning of TLS connections without certificate source verification

Check the usage of a non-secure version of the SSL protocol

Scan remote shell command execution events and code contexts



Core Dump Process Remains Analysis (Process Crashes)

All	Core Dump Analy	sis	
Applicati	on Type *	OpenResty ~	
Analyzer *		openresty-core-dump-analysis (OpenResty Core Dump Analysis)	
Core File	*	/tmp/core.3859164	
Executab	le File Path *	/usr/local/openresty/nginx/sbin/nginx	
► Advano	ced Settings		
Core F	ile Meta Data		
File Nar	me: /tmp/core.38	59164	
Executa	a <mark>ble File Path:</mark> /us	r/local/openresty/nginx/sbin/nginx	
Size : 14	.72MB		

Analyze

Extract Deep Information from Core Dump Files

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

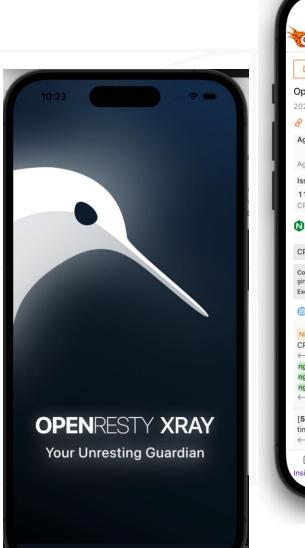
0x42e958 ngx_md5_body [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/src/core/ngx_md5.c:199] 0x42f1be ngx_md5_final [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/src/core/ngx_md5.c:91] 0x4ea997 ngx_http_lua_ffi_md5 [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/../ngx_lua-0.10.21/s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

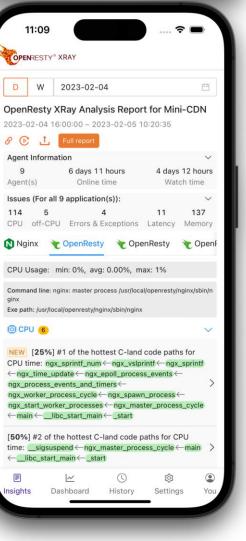
0x4f86a1 ngx_http_lua_run_thread [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/../ngx_lua-0.10.2

OpenResty XRay Mobile Apps

Watch your applications from any where, any time

- Android (Google Play)
- iOS (Apple Store)





OpenResty XRay is Not just for OpenResty Applications

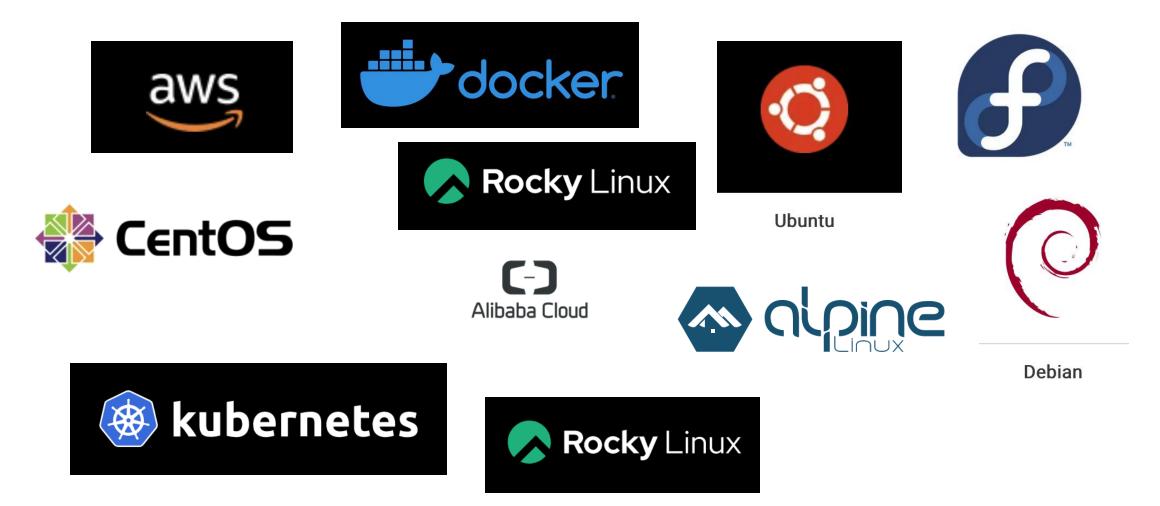
- Nginx, LuaJIT, OpenResty, Python, PHP, Go, Erlang, Perl, Envoy, Ruby, Redis, Rust, Kong
- Preliminary Support: PostgreSQL
- Coming soon: NodeJS, Java

sirror_mod.mirror_object peration == "MIRROR_X": irror_mod.use_x = True mirror_mod.use_y = False irror_mod.use_z = False _operation == "MIRROR_Y" lrror_mod.use_x = False lrror_mod.use_y = True lrror_mod.use_z = False _operation == "MIRROR_Z" rror_mod.use_x = False lrror_mod.use_y = False Irror_mod.use_z = True election at the end -add ob.select= 1 er ob.select=1 ntext.scene.objects.active "Selected" + str(modifie irror ob.select = 0 bpy.context.selected_object ata.objects[one.name].se Mint("please select exactle OPERATOR CLASSES ----vpes.Operator): X mirror to the select ject.mirror_mirror_x"

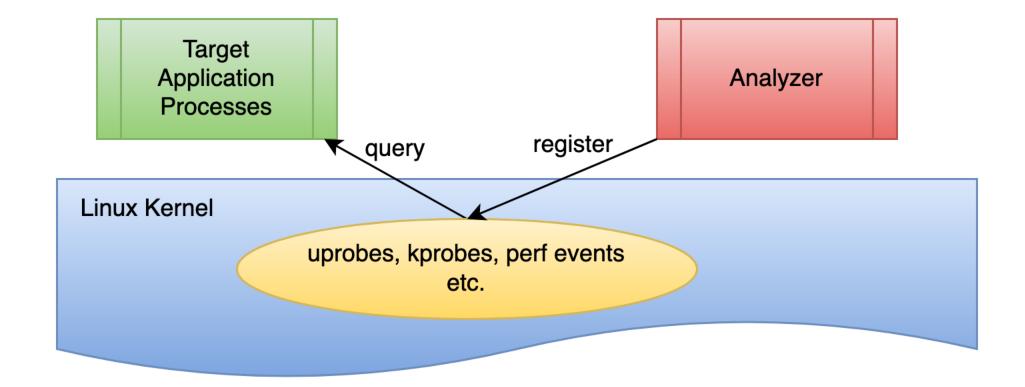
> oncext): next active object is not

ror X"

Supports Most Mainstream Linux Distributions and Container Deployment



OpenResty XRay is based On the Advanced Dynamic Tracing Technology



Advantages of Dynamic Tracing

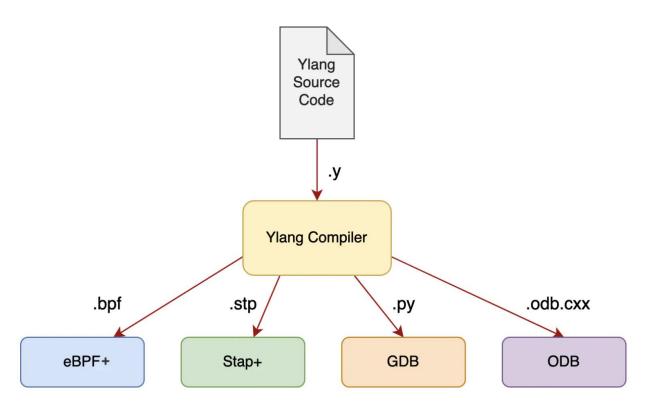
- Non-invasive, no need to modify application modifications
- Hot-plugging, usually do not need the cooperation of applications (many open-source dynamic tracing tools still require the cooperation of applications)
- Overhead is normally low and aggregation can be done at the data source
- Online real-time debug capability in a postmortem manner
- Full technology stack analysis from all angles
- On-demand sampling
- Strictly 0 loss when not sampling

OpenResty XRay New generation of Dynamic Tracing Technology

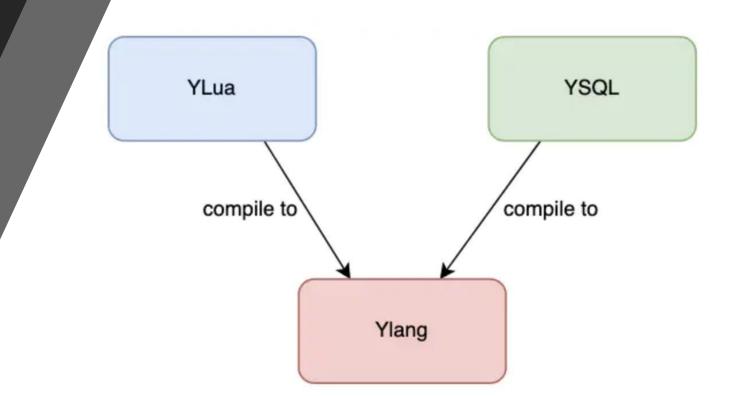
- Y-language (Ylang) compiler (supports most of the syntax of GNU C and standard C)
- Ylua Language
- YSQL Language
- Stap+ has significantly improved SystemTap
- eBPF+ significantly improves eBPF (while LLVM+ also significantly improves open source LLVM)
- ODB is an ultra-lightweight version of GDB
- The Ylang compiler can also generate highly optimized Python extension codes of GDB
- Stringent performance loss control aimed at online production environments

"Write once, run everywhere"

From Y-language code to various runtime codes



More Abstract Languages Which Are Based on Y Language



Write and Debug Analysis Tools Written in Languages like Ylang/YLua/YSQL On the OpenResty Xray Console's Web UI

OPENRES	TY® XRAY 📃 🔍 Version: 844) en unit prd.openresty.com (3 👻 📀	📋 🚺 🤤 🚍 yichun@openresty.com 👻 English 🕶						
Overview	All analyzers / Edit analyzer	Clone this analyzer + Add a new analyzer						
CPU	Analyzer name Analyzer description process-exit C-land CPU Flame Graph	 Try this analyzer against: 						
Memory Disk Network	YSQL YLua YLang C Learn YLang C Run B Save Vim mode OFF C	TargetByByByWholeApplicationsProcessesExecutablesSystem						
Latency	<pre>1 // c-cpu: C-land CPU flame graph sampling tool. 2 // Copyright (C) OpenResty Inc. 3 // All rights reserved.</pre>	Application openresty (PGID 1471, master proc ◆ ① Update						
Load Gen History	<pre>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;</pre>	Target Processes PID 2782 (worker process) CPU: 19 <						
Events Advanced	10 11 _probe _process.begin 12 {	▼ YLang Settings						
Custom Apps	<pre>13 printf("ngx_quit %ld\n", ngx_quit); 14 printf("ngx_exiting %ld\n", ngx_exiting); 15 printf("ngx_reconfigure %ld\n", ngx_reconfigure);</pre>	Dependent debug data to compile the analyzer We have a huge package archive database for public software and we will not install any debuginfo packages on the target machine. Kernel debug info:						
Settings Upload	16 _exit(); 17 } 18							
	Clear the editor	Not required Required Good to have Unwind data: NO NEED No Need unwind data when you need the backtrace.						

OpenResty XRay Hundreds of Standard Analyzers

 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-timerdistr 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-keepalivestats ngx-dump-req ngx-dump-timers ngx-epoll-wait-timers ngx-err-log-lvl-distr

Debug symbols

- OpenResty XRay has a central package database indexing hundred TB of debug symbols for public packages. This database is still growing rapidly
- The target machine does not need to install or store debug symbols, as long as they have been indexed by the OpenResty XRay Central Package Database
- For applications in which debug symbols cannot be found or were not generated at compile time, OpenResty XRay will be able to automatically rebuild debug symbols (prototype of working machine learning algorithms already exist)



Trusted By Lots of Enterprise Customers



Learn More

OpenResty XRay Frequently Asked Questions

Visit OpenResty Xray official blog

Visit OpenResty XRay official website

Try OpenResty XRay for free