bpftrace Dynamic kernel tracing

CONCEPTS

Needs root (or CAP_BPF/ CAP_PERFMON) and a kernel with BPF. Maps you fill with @ are printed automatically when the program ends.

A program is a list of probe / filter/ { action } clauses. Docs for current releases write begin and end in lowercase and use args.field; older scripts use BEGIN/END and args→field.

Probes in a one-liner are cheap to attach, but a hot probe (every syscall, every packet) adds real overhead. Filter by pid or comm first.

RUNNING

`bpftrace -e '<program>'`
Run an inline program

`bpftrace <file.bt>`
Run a script file

`bpftrace -l '<pattern>'`
List probes, wildcards allowed

`bpftrace -lv '<probe>'`
List a probe with its arguments

`bpftrace -p <pid> -e '<program>'`
Attach to or filter by a PID

`bpftrace -c '<cmd>' -e '<program>'`
Run a command and trace it

`bpftrace -f json -e '<program>'`
Output as JSON lines

PROBES

`kprobe:<func>`
Kernel function entry

`kretprobe:<func>`
Kernel function return (retval)

`tracepoint:<cat>:<name>`
Static kernel tracepoint

`uprobe:<path>:<func>`
User function entry

`begin { ... } / end { ... }`
Run once at start / at exit

PROBES (cont.)

`uretprobe:<path>:<func>`
User function return

`usdt:<path>:<prov>:<name>`
User static tracepoint

`profile:hz:<rate>`
Sample on every CPU at a rate

`interval:s:<n>`
Fire once per interval, on one CPU

`fentry:<func>`
Typed kernel entry, args.<name> (BTF)

BUILT-INS

`pid / tid`
Process / thread ID

`comm`
Process name

`arg0 ... arg5`
Raw function arguments

`retval`
Return value in return probes

`nsecs`
Timestamp in nanoseconds

`args.<field>`
Tracepoint or fentry argument

MAPS AND AGGREGATION

`@ = count()`
Count events

`@[<key>] = count()`
Count per key (comm, pid, probe)

`@[<key>] = sum(<n>)`
Sum a value per key

`@[<key>] = avg(<n>)`
Average per key

`@ = hist(<n>)`
Power-of-2 histogram

`@ = lhist(<n>, <min>, <max>, <step>)`
Linear histogram

`@start[tid] = nsecs`
Remember a timestamp per thread

`delete(@<map>, <key>)`
Remove one key

MAPS AND AGGREGATION (cont.)

`clear(@<map>)`
Remove all entries

`print(@<map>, <n>)`
Print the top n entries

`has_key(@<map>, <key>)`
Does the key exist?

Maps without a clear() or delete() are dumped at exit, including temporaries such as @start. Clear them in an end block.

OUTPUT AND FUNCTIONS

`printf("<fmt>\n", <args>)`
Formatted output

`str(<ptr>)`
Read a C string

`exit()`
Stop the program

`kstack(<n>)`
Kernel stack, top n frames

LANGUAGE

`probe /<predicate>/ { ... }`
Run the action only if true

`$x = <value>;`
Scratch variable, local to the probe

`(struct <name> *)<ptr>`
Cast to a kernel struct

SCRIPTS AND CONFIG

`#!/usr/bin/env bpftrace`
Shebang for a runnable .bt file

RECIPES

Syscalls by process

```
t:raw_syscalls:sys_enter {
  @[comm] = count();
}
# Ctrl+C prints the counts
```

Files opened, live

```
t:syscalls:sys_enter_openat {
  printf("%s %s\n",
    comm, str(args.filename));
}
```