PyCon Canada 2017

Using Python to make sense of system traces

EfficiOS

jeremie.galarneau@efficios.com
jgalar

Presenter

Jérémie Galarneau

EfficiOS Inc.

- Vice President
- http://www.efficios.com

Maintainer of

- LTTng-tools
- Babeltrace

Outline

- Quick introduction to tracing
- What is LTTng
- Using Python to work with traces
- Q&A

Isn’t tracing just another name for logging?

- Tracers are not completely unlike loggers
- Both save information used to understand the state of an application
  - Trying to cater to developers, admins, and end-users
- In both cases, a careful balance must be achieved between verbosity, performance impact, and usefulness
  - Logging levels
  - Enabling only certain events

Different goals, different tradeoffs

- Tracers focus on low-level events
  - syscalls, scheduling, filesystem events, etc. (1000+ events / second)
  - More events to capture means we must lower the cost / event
  - Binary format
  - Different tracers use different strategies to minimize cost

Different goals, different tradeoffs

- Traces are harder to work with than text log files
  - File size
  - Exploration is difficult
  - Must know the application / kernel to make sense of what was captured
  - Purpose-built tools are needed

LTTng: Linux Trace Toolkit Next Generation

Open source tracing framework for Linux first released in 2005

- Regroups a number of projects
  - LTTng-UST
  - LTTng-modules
  - LTTng-tools
  - LTTng-analyses

LTTng

Safe to use in production environments

- Reliability
  - Out of process trace configuration and collection facilities
- Low intrusiveness
  - Lock-free ring buffer
  - Wait-free RCU techniques and data structures
  - Non-blocking when buffers are full
- Dynamic configuration = no need to restart applications

LTTng

Unify many information sources

- Kernel
- C/C++ applications
- Python standard `logging` module
- Java (log4j and `java.util.logging`)
- Others
    - `/proc/lttng-logger`

User space tracing

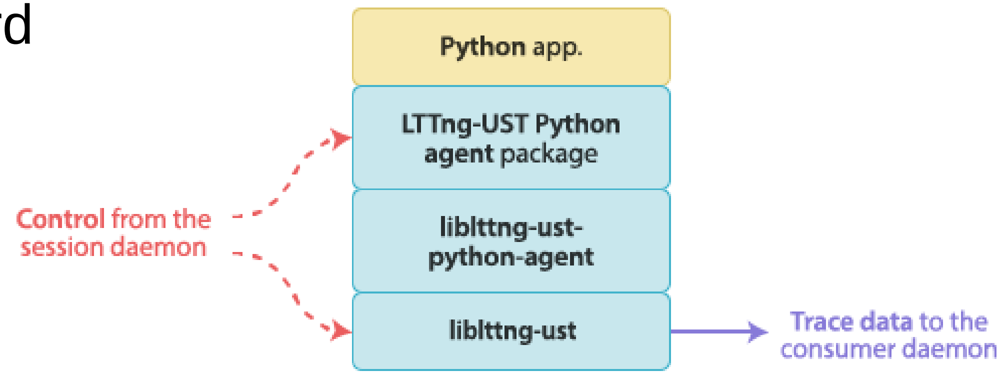
- Useable as a high-performance logging system
  - Can be blocking or non-blocking
- Emitting events from userspace applications provides *anchor points* in kernel traces
- Allows us to figure out *why* an application is performing a certain action

Instrumenting a Python application

LTTng-UST Python agent

- Logging handler for standard **`logging`** package
- Supports Python 2 and 3

```
import lttngust
```


Python app.
LTTng-UST Python agent package
liblttng-ust-python-agent
liblttng-ust
Control from the session daemon
Trace data to the consumer daemon

Better views

- User space developers are building custom tools!
  - Know exactly what they are looking for
  - Modelling their application
  - Tracking internal resources (worker threads, memory pools, connections, users, etc.)

- Originally text-based tools
  - Piping hundreds of GBs of text traces through `grep, sed, perl, awk...`
  - Lots of one-off scripts being passed around
  - Unmanageable, hard to maintain, etc.
  - Break when Babeltrace’s text output changes (new event fields)

Babeltrace Python Bindings

- Introduced Python bindings to read traces (2013)
  - Provide users with an easy way to “hack something together”
    - Debugging
    - Testing
  - Reasonably efficient under most scenarios
  - Scripts are maintained as internal tools

- Could we do the same for kernel space?

LTTng analyses

- Development started in early 2014
- Collection of utils
- Models some kernel subsystems to track their current state
  - Latency statistics and distributions (IO, Scheduling, IRQ)
  - System call statistics
  - IRQ handler duration
  - Top resource users

https://github.com/lttng/lttng-analyses

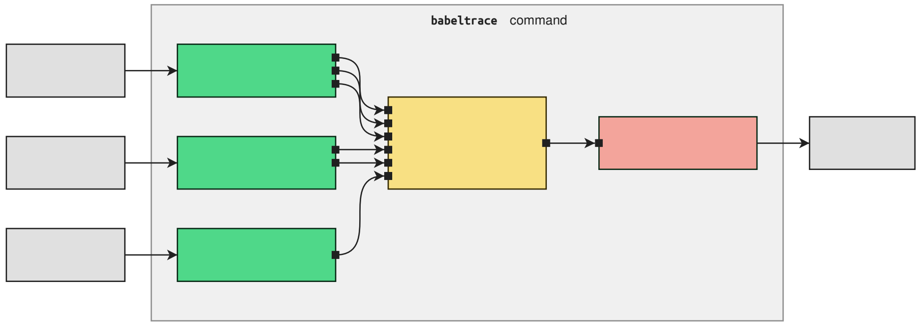
Talk about investigating using those scripts: https://www.youtube.com/watch?v=-lUfzp-2_bY

Building our own tools with Python

- Python has proven to be a great language to iterate on new tools
- Focus on finding ways to filter through the noise and find clues in traces
- Performance can be a concern if you try to work on too many events
  - Carefully choosing what you trace pays both at runtime and during the analysis

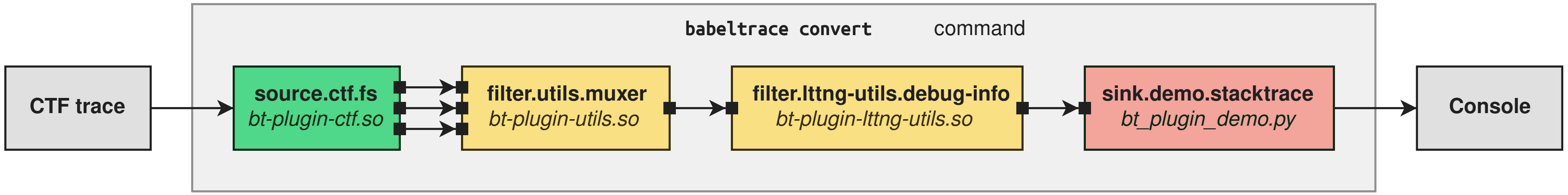
Babeltrace 2.0

- Provides generic components that can be combined to form a trace processing graph
  - CTF file system source
  - CTF file system sink
  - LTTng-live source
  - dmesg source
  - Muxer
  - Trimmer
  - Debugging information injecter



- Components can be written in C, C++, and Python

“Callstack” view



Questions ?

Babeltrace

- lttng.org
- lttng-dev@lists.lttng.org
- @lttng_project

- diamon.org/babeltrace
- github.com/efficios/babeltrace
- lttng-dev@lists.lttng.org