FOSDEM '19

A follow-up on LTTng container awareness

EfficiOS

mjeanson@efficios.com

Who am I?

- Michael Jeanson
- Software developer @ EfficiOS
- Debian Developer

EfficiOS

Plan

- What's LTTng and why use it
- What does it mean for us to "support" containers
- Progress since last year's talk
- An overview of current features
- What's coming next

EfficiOS

What's LTTng?

- 2 tracers
  - Kernel : lttng-modules
  - Userspace : lttng-ust
- A trace format : CTF
- A common cli tool / library : lttng-tools
- A cli trace reader : babeltrace
- Multiple graphical trace readers

EfficiOS

Why use LTTng?

- Combined kernel and user-space tracing solution
- Low overhead and stable, can be used on production systems
- Can be enabled / disabled and reconfigured at run-time
- Flexible storage of traces
  - Local disks
  - Network streaming
  - In memory ringbuffers

EfficiOS

What's a container?

- From a kernel perspective, there is no single concept of a container
- Multiple run-times with their own tooling
- But all based on kernel features like namespaces, cgroups and other isolation and security systems

EfficiOS

Container support?

- Supporting containers can be divided in 2 main tasks :
  - The traces we produce must contain adequate information to model and diagnose systems composed of containers
  - Our tooling and deployment strategies have to be adjusted to fit containerized systems

EfficiOS

Trace content

- What we have (queued for 2.12, ~April 2019)
  - Kernel tracer
    - Namespaces state dump to get system state at trace startup
    - Namespace contexts to classify and filter events
    - Syscall events to track namespace changes
  - Userspace tracer
    - Namespace contexts to correlate with kernel traces
- What's still missing
  - Container runtimes metadata and state change events

Trace content : Contexts

- What's an LTTng context?
  - An additional metadata field added to an event record along it's name and payload
  - For example : Process ID, thread ID, process name, hostname, perf counters, etc
  - Usefull for readability when manually processing traces
  - Used for filtering of events during tracing

EfficiOS

Trace content : Contexts

- Add a context for each namespace type for both kernel and user-space tracers
  - pid, user, cgroup, ipc, mnt, net, uts
- On the kernel side this is used to filter or classify events per container
- On the user-space side this is used to correlate events with kernel traces

EfficiOS

Kernel Tracer NS Contexts

- Syscalls and other kernel events with namespace contexts
  - tid: The unique process id on the host
  - vtid: The process id specific to this namespace
  - pid_ns: A unique identifier for this process pid namespace
- With this information, we can group processes into containers and do host to container process id mapping

[15:54:15.216386600] (+0.000006785) ns-contexts syscall_entry_gettimeofday: { cpu_id = 1 }, { procname = "redis-server", pid = 11734, vpid = 1, tid = 11734, vtid = 1, ppid = 11714, cgroup_ns = 4026531835, ipc_ns = 4026532571, net_ns = 4026532574, pid_ns = 4026532572, user_ns = 4026531837, uts_ns = 4026532570 }, { }

EfficiOS

Userspace Tracer NS Contexts

- Userspace events with contexts will allow correlation with kernel events in the analyses
  - vtid: Same field in the kernel events, allows to match with system wide process ids
  - pid_ns: Same field in the kernel events, allows per container filtering

[22:51:19.896554347] (+1.000484100) master-cheetah ust_tests_hello:tptest: { cpu_id = 1 }, { procname = "hello", vpid = 27486, vtid = 27486, pid_ns = 4026532298, user_ns = 4026532294 }, { intfield = 1, intfield2 = 0x1 }

EfficiOS

Contexts : Filtering Example

- Filter all syscalls from a docker container

```
# Get the pid of the docker container init process
$ pid=$(docker inspect --format '{{.State.Pid}}' my-container)

# Get the pid namespace id from this pid
$ pid_ns=$(lsns -n -t pid -o NS -p ${pid})

# Create a session and add the required contexts
$ lttng create my-container
$ lttng add-context -k -t procname -t pid -t vpid -t tid -t vtid -t
pid_ns

# Enable all the syscalls, filter by pid namespace for my-container
$ lttng enable-event -k --syscall -a --filter="\$ctx.pid_ns == ${pid_ns}"
```

EfficiOS

Simpler and Faster filtering

- LTTng has a “tracker” feature, the only one currently implemented is for process IDs
- We plan to add a namespace tracker
  - Instead of using a filter like this :

```
$ lttng enable-event -k --syscall -a \
    --filter=”\$ctx.pid_ns == ${pid_ns}”
```

  - You would add a tracking rule :

```
$ lttng enable-event -k --syscall -a
$ lttng track -k --pid_ns=”${pid_ns}”
```

EfficiOS

Trace content : State dump

- What's an LTTng state dump?
  - A series of event records emitted when the tracing session starts or when manually triggered
  - Initial state of file descriptors, net devices, processes, cpu topology, etc
  - Used by viewers to build an initial system state

Trace content : State dump

- Add an event record for each namespace type per process
  - pid, user, cgroup, ipc, mnt, net, uts
  - Include hierarchical information for the nested namespace types (pid and user)
- With this information we can build a list of running containers

EfficiOS

Kernel Tracer NS State dump

- Process state dump events for namespaces
  - The process “tid” is the primary key, it’s unique in the kernel across containers
  - Pid namespace can be nested, one event per level with “ns_level” to track the hierarchy

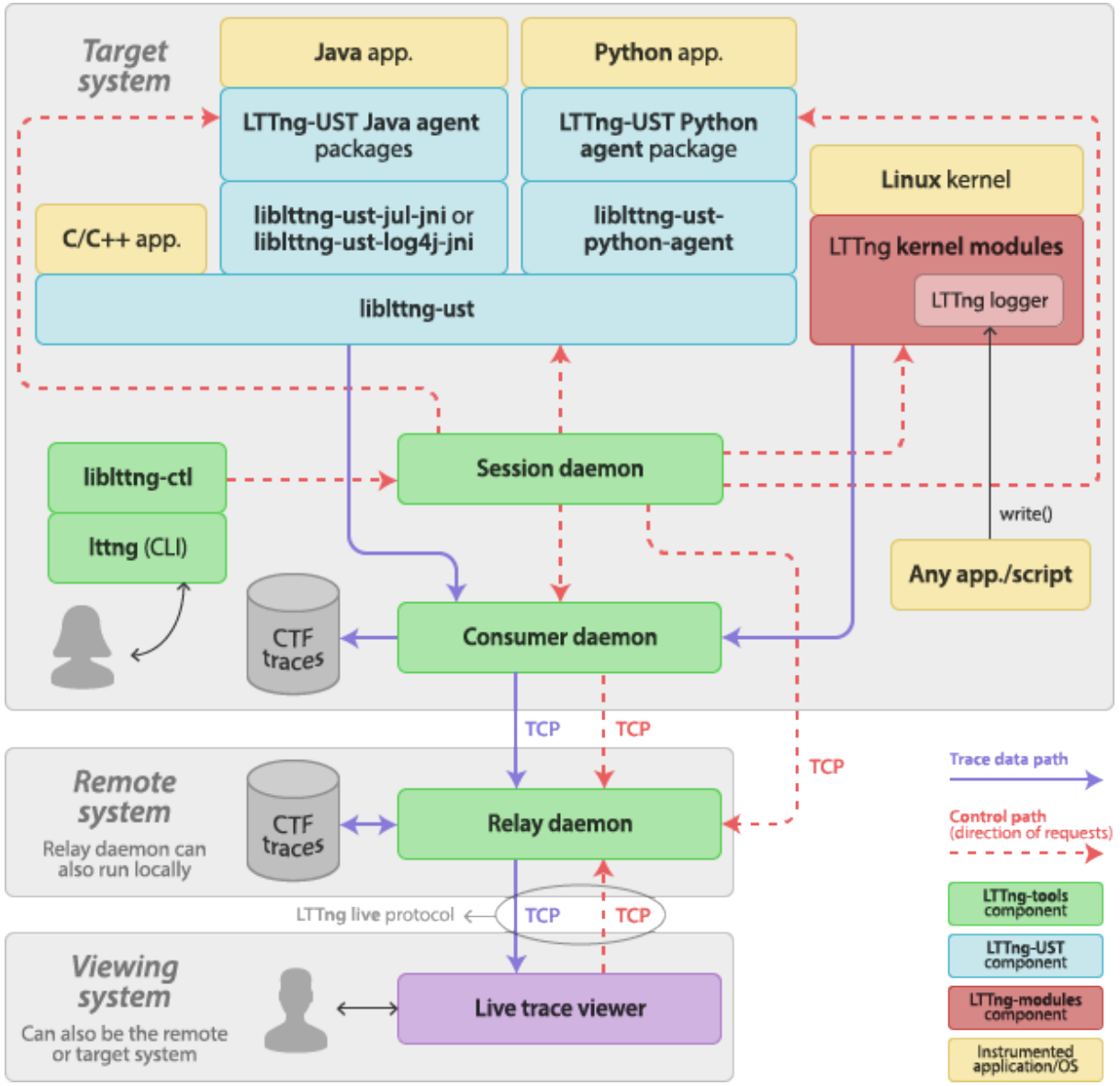
[15:54:05.937411441] (+0.000000501) ns-contexts lttng_statedump_process_state: { cpu_id = 1 }, { tid = 1527, pid = 1527, ppid = 1353, name = "systemd", type = 0, mode = 5, submode = 0, status = 5, cpu = 1 }
[15:54:05.937411834] (+0.000000393) ns-contexts lttng_statedump_process_pid_ns: { cpu_id = 1 }, { tid = 1527, vtid = 1, vpid = 1, vppid = 0, ns_level = 1, ns_inum = 4026532424 }
[15:54:05.937412212] (+0.000000378) ns-contexts lttng_statedump_process_pid_ns: { cpu_id = 1 }, { tid = 1527, vtid = 1527, vpid = 1527, vppid = 1353, ns_level = 0, ns_inum = 4026531836 }

EfficiOS

Tooling and deployment

- The current tooling is not “aware” of containers
- LTTng is comprised of many components that expect a “monolitic” system
- Security and authorization rely on unix users and groups with filesytem permissions and credential passing on unix sockets

Tooling and deployment



EfficiOS

Currently supported deployment

- Kernel tracer
  - Must be deployed on the host, kernel modules and control tools version must match
- Userspace tracer
  - Can be deployed on the host and in the containers, but can only trace processes in the same Namespaces
  - Version of the tracer can be different in each container
- Resulting traces from the host and multiple containers can then be post processed together

EfficiOS

Tooling : What's next?

- Decoupled container "aware" tooling
  - Cross container authorization for tracing control
  - ID mapping in the tracing configuration
  - Spec'ed and versioned protocol between consumerd and sessiond
  - Light-weight userspace tracer inside containers
  - Rethink default policies on tracing permissions and add configurability

EfficiOS

What do you need?

- We are interested in your needs and use cases

EfficiOS

Questions

LTTng Project

https://{git | www}.lttng.org

lttng-dev@lists.lttng.org

@lttng_project

#lttng on irc.oftc.net

EfficiOS