LTTng: The road to container awareness
Who am I?

- Michael Jeanson
- Software developer @ EfficiOS
- Debian Developer
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
Why LTTng?

- Low overhead
- Combined kernel and userspace tracing
- Can be enabled / disabled at runtime
- Flexible storage usage
  - Network streaming
  - In memory ringbuffers
What’s a Linux container?

- There is no canonical concept of a container in the kernel
- Multiple implementations like Docker, rkt, LXD and many others
- All based on kernel namespaces, cgroups and other isolation and security systems

EfficiOS
Current status

• The kernel tracer can be used on the host
  - No way to filter events per-container

• The userspace tracer can be used on the host and in the containers
  - One lttng instance on the host
  - One lttng instance per container
  - No context information to correlate traces between the host and the containers
What can we fix now?

- Add namespaces support to the kernel tracer
  - Add a context for each namespace type
  - Add namespace information to the statedump
- Build simple userspace statedump providers for container runtimes to do container-to-namespace mapping
- Add per-container views to current kernel analyses
Let’s write some patches

• Experimental tracer branches with a minimum viability implementation of the ns contexts and statedump
• Experimental lttng-tools branch with per context filtering
• A simple shellscript based container runtime statedump for Docker and LXD
• Experimental lttnganalyses branch with a per-container cputop and memtop analyses
Kernel Tracer

- Add a context for each namespace type
  - pid, user, cgroup, ipc, mnt, net, uts
- Add namespaces to the process statedump
  - Include hierarchical information for the nested namespace types (pid and user)
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 }, { }
Kernel Tracer NS Statedump

- Process statedump 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 order

[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
Userspace Tracer NS Contexts

• Add a context for each namespace type
  - pid, user, cgroup, ipc, mnt, net, uts
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 }
Filtering Example

- Filter all syscalls from a docker container

```bash
# 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}"```
Container Runtimes Statedump

- We need to map the kernel ids to human readable names
- Use small userspace helpers to dump this information
- One implementation per container runtime

[15:54:10.128336926] (+0.000000670) ns-contexts

ust_container_statedump:lttng_statedump_container: { cpu_id = 0 }, { container_name = "ample-adder", container_type = "lxd", pid_ns = 4026532354 }

[15:54:09.700828135] (+0.000000450) ns-contexts

ust_container_statedump:lttng_statedump_container: { cpu_id = 1 }, { container_name = "goofy_haibt", container_type = "docker", pid_ns = 4026532295 }

EfficiOS
**Analyses**

- Combine all this information to run kernel level analysis per container

### Per-TID Usage

<table>
<thead>
<tr>
<th>Process</th>
<th>Migrations</th>
<th>Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab (23205)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>lttng-consumerd (11932)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>apache2 (23033)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>apache2 (23007)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>gcc (23297)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>gcc (23294)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>lxd (23364)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>gcc (23300)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>lxd (23363)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>lxd (23362)</td>
<td>0</td>
<td>[20]</td>
</tr>
</tbody>
</table>

### Per-CPU Usage

<table>
<thead>
<tr>
<th>CPU</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>81.80 %</td>
</tr>
<tr>
<td>1</td>
<td>77.57 %</td>
</tr>
</tbody>
</table>

### Per-Container Usage

<table>
<thead>
<tr>
<th>Container</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>[HOST] (4026531836)</td>
<td>host</td>
</tr>
<tr>
<td>goofy_haibit (4026532295)</td>
<td>docker</td>
</tr>
<tr>
<td>ample-adder (4026532356)</td>
<td>lxd</td>
</tr>
<tr>
<td>coherent-macaque (4026532424)</td>
<td>lxd</td>
</tr>
<tr>
<td>master-cheetah (4026532491)</td>
<td>lxd</td>
</tr>
<tr>
<td>thirsty_meninsky (4026532572)</td>
<td>docker</td>
</tr>
<tr>
<td>some-redis (4026532637)</td>
<td>docker</td>
</tr>
</tbody>
</table>
Analyses


Per-Container Memory Allocations

<table>
<thead>
<tr>
<th>Container</th>
<th>Type</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>some-redis</td>
<td>docker</td>
<td>0</td>
</tr>
<tr>
<td>thirsty_meninsky</td>
<td>docker</td>
<td>0</td>
</tr>
<tr>
<td>master-cheetah</td>
<td>lxd</td>
<td>0</td>
</tr>
<tr>
<td>coherent-macaque</td>
<td>lxd</td>
<td>0</td>
</tr>
<tr>
<td>ample-adder</td>
<td>lxd</td>
<td>676</td>
</tr>
<tr>
<td>goofy_haibt</td>
<td>docker</td>
<td>0</td>
</tr>
<tr>
<td>[HOST]</td>
<td>host</td>
<td>232768</td>
</tr>
</tbody>
</table>

Per-Container Memory Deallocations

<table>
<thead>
<tr>
<th>Container</th>
<th>Type</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>some-redis</td>
<td>docker</td>
<td>0</td>
</tr>
<tr>
<td>thirsty_meninsky</td>
<td>docker</td>
<td>1</td>
</tr>
<tr>
<td>master-cheetah</td>
<td>lxd</td>
<td>0</td>
</tr>
<tr>
<td>coherent-macaque</td>
<td>lxd</td>
<td>0</td>
</tr>
<tr>
<td>ample-adder</td>
<td>lxd</td>
<td>85</td>
</tr>
<tr>
<td>goofy_haibt</td>
<td>docker</td>
<td>0</td>
</tr>
<tr>
<td>[HOST]</td>
<td>host</td>
<td>231060</td>
</tr>
</tbody>
</table>

Total memory usage:
- 233444 pages allocated
- 231146 pages freed
Scenario

- Run a kernel and userspace tracer on the host
  - Output to a remote relayd
  - Enable namespace contexts on syscalls
  - Enable statedump with namespace information
  - Periodically run container runtime statedump
- Run a userspace only tracer in the containers
  - Output to a remote relayd
  - Enable namespace contexts on application specific ust tracepoints
  - This in-container tracer doesn't need to be the same version as the host tracer, it can be deployed according to a different release cycle
- Run offline analysis that can correlate information from both sources
What’s next?

• Merge a finalized version of the patches upstream
• Add cgroups contexts and statedump
Longer Term Improvements

• Instrument container runtimes
  – Add events to track container lifetime
    • Kernel ids for namespace are ephemeral
  – Viewers and analyses could track containers across restart

• Add container support to control tools
  – For example, “lttng filter --container container-name”
What do you need?

- We are interested in your needs and use cases
Questions

LTTng Project

🌐 https://{git | www}.lttng.org
✉️ lttng-dev@lists.lttng.org
🐦 @lttng_project
👥 #lttng on irc.oftc.net