Put Some SRE in Your Shipped Software

Hard-won lessons from the world of SRE
The nature of the problem:

**Software Sucks.**

Once you’ve run software at scale, you have a deep understanding of how it is all tied together with **loose string** and **hope**.

We spend massive effort to operationalize our stacks.
The burning question:

Why Ship Software?
The trends are there. You can choose to see them or not.
Rules.

1. Crash landings should be both fast and controlled.
2. Post-mortems are fundamental.
3. Use circuit breakers.
4. Behavior is complex. Understand it.
5. Have a failure budget.
6. Instrumentation & observability have no equals.
Build upon the right layers

Crash Analysis

If you don’t know why it failed, you don’t know anything at all.
When in doubt or even curious

Expose Telemetry

Ideally, any question you would ask of a production system can be done so nondisruptively.
libcircmetrics

- C library; BSD license, fast, thread-safe, largely lockless.
  - Text metrics (version numbers, statuses, etc.)
  - Numeric gauges, counters (w/ CPU fanout)
  - Histograms (log-linear quantized) (9ns recording)
  - Simultaneous hierarchical (graphite-style) and tagged annotation support
  - JSON output
```c
stats_recorder_t *rec = stats_recorder_alloc();
stats_ns_t *ns = stats_register_ns(rec, NULL, "db");
stats_handle_t *h;

ns = stats_register_ns(rec, ns, "raw");
stats_ns_add_tag(nomns, "db-type", "raw");
stats_ns_add_tag(nomns, "db-impl", "nom");

h = stats.put_calls = stats_register(nomns, "put.calls", STATS_TYPE_COUNTER);
stats_handle_tagged_name(h, "calls");
stats_handle_add_tag(h, "operation", "put");
stats_handle_units(h, STATS_UNITS_REQUESTS);

h = stats.get_calls = stats_register(nomns, "get.calls", STATS_TYPE_COUNTER);
stats_handle_tagged_name(h, "calls");
stats_handle_add_tag(h, "operation", "get");
stats_handle_units(h, STATS_UNITS_REQUESTS);

h = stats.writes = stats_register(nomns, "put.tuples", STATS_TYPE_COUNTER);
stats_handle_tagged_name(h, "count");
stats_handle_add_tag(h, "operation", "put");
stats_handle_units(h, STATS_UNITS_TUPLES);

h = stats.write_latency = stats_register(nomns, "put.latency", STATS_TYPE_HISTOGRAM);
stats_handle_tagged_name(h, "latency");
stats_handle_add_tag(h, "operation", "put");
stats_handle_units(h, STATS_UNITS_SECONDS);

h = stats.write_batch = stats_register(nomns, "put.batchsize", STATS_TYPE_HISTOGRAM);
stats_handle_tagged_name(h, "batchsize");
stats_handle_add_tag(h, "operation", "put");
stats_handle_units(h, STATS_UNITS_TUPLES);
```
```c
uint64_t start = mtev_gethrtime();

int rv = database_put(ctx, write_objects, write_count);

uint64_t end = mtev_gethrtime();

/* maintain a histogram of write operation latency */
stats_set_hist_intscale(stats.write_latency, end - start, -9, 1);
/* maintain a histogram of batch sizes */
stats_set_hist_intscale(stats.write_batch, write_count, 0, 1);
/* total tuple count */
stats_add64(stats.writes, write_count);
if(rv != 0) {
    stats_add64(stats.errors, 1);
}
/* total total calls */
stats_add64(stats.put_calls, 1);
```
Known unknowns

Events & Distributed Tracing

A clearer story of what just happened.
Some stats...

- We only retain traces for a short period of time (up to about 3 days)
- We don’t trace with all detail on due to overhead
  - Full debugging on in a trace can produce up to 4Gb of trace data for a single user request
  - We do this sometimes via manual triggering as a debugging action
- Typically, between 10 and 2000 traces per request
- We use this as a debugging observability tool
During failure reconstruction, logs hold truth

Logging for humans

Computers talking to computers have better ways than logs. Logs are for computers talking to humans.
Real unknown unknowns are solved by:

**Dynamic Tracing**

- eBPF / bpftrace
- DTrace
IO Latency... single node... 2014
Your m8g, only need to be accessible

**Internalized MVP**

No additional apparatus. No additional deployment constraints
Shipping software means more operators

Codify Operational Assessment & Procedures

More operators, less average knowledge. Ensure procedures are repeatable.
Tools -> Solutions
Every effort to bring SRE techniques to software engineering makes SRE more accessible and useful in Cloud/SaaS engineering.
Thank You.

Theo Schlossnagle
CEO, Circonus

@postwait