# System and Application Logging at scale

# About me

Balázs Scheidler (bazsi@balabit.hu)

Original author for syslog-ng (since 1998)

Contributor and author of various other open source projects (Netfilter, iptables, Zorp)

# About logging

Initially, it seems quite simple

„Text with a timestamp”

Gets more and more complicated as we scale in complexity & volume

# Trivialisms

Timestamps

- High precision
- Timezones
- Year (rfc3164 has no year!)

Log levels

Log rotation

# Then: the app scales

The Stack has logs too!

Multiple application instances

Multiple tiers (DB, app, load balancers, ...)

Multiple hosts (in each of the tiers)

Network equipments

Timezones (when going global)

Volume

Analytics (performance and format wise)

# Then: the enterprise scales

Multiple applications

Each with a different stack

Requirements to collect, analise logs centrally (operations & security)

Different audiences (developers, operators, security)

Merging & Managing all this information is already quite **difficult**

And we want this out of the box, right?

:)

# What's stopping us?

Timestamps:

- original BSD syslog,
- its proprietary extensions,
- RFC3339 (aka ISO8601),
- UNIX timestamp
- With or without high precision
- With or without timezone information

# … and

Formats & containers:

- RFC3164: original BSD syslog,
- RFC5424: new, standardized syslog (not really caught up)
- Apache „combined” log format
- CSV, TSV, WELF, name-equals-value
- JSON
- stdout/stderr of applications (not originally meant to be used as logs)
- No standard character encoding! Utf8, latin & memory garbage in the same file

# Structured logging will solve these, right?

# Structured Logs

It has its own set of problems:

- No standardized field naming (MESSAGE, @message, msg)
- No standardized value types or representation, two fields with the same name, different representation (timestamp)

What we have is a **myriad** of **application specific dialects** of textual data, **some** of which is **easier to parse** (e.g. JSON) than others (e.g. text)

We should „just” fix all application to use consistent logging out there.

Never gonna happen.

**You** need to **fix the logs you really care** about with the tool of your choice.

# Which Tool?

syslog-ng (of course :)

# Why syslog-ng?

One of the earliest contenders (having been started in 1998)

Clear & flexible configuration syntax and great documentation

Evolving incrementally, morphing new requirements into working code

Performance

Community and mindshare

Deployed literally in millions of devices

syslog-ng is **not just about syslog** for a long time now

# Message model

Consider the syslog-ng message as a JSON document

- Generic name-value pairs that hold strings (or typed values)
- Timestamps
- Tags

# Message

```
$(format-json --scope rfc5424 --key *.*)

{
  "custom": {
    "name": "value"     # custom.name=”value”
  },
  "_unix": {
    "uid": "1000",      # .unix.uid
    "pid": "5887",      # .unix.pid
    "gid": "1000"       # .unix.gid
  },
  "PROGRAM": "bazsi",
  "PRIORITY": "notice",
  "MESSAGE": "almafa",
  "HOST": "bzorp",
  "FACILITY": "kern",
  "DATE": "Jul  4 10:51:05"
}
```

# Field names

Name-value pairs are named like JavaScript dot notation. These are valid syslog-ng field names:

- http.method
- postfix.rcpt_to
- .nodejs.winston.message

Transformation

- syslog-ng doesn't insist on its own naming, but rather it makes it possible to transform between schemas

Reserved fields

- Fields that start with a dot or underscore and those without a dot are reserved for use by syslog-ng.

# Processing model

Inputs are not files, they are streams of log entries

Streams are dispatched over a pipeline of processing elements

- Filter, parser, rewrite

Each message can be processed in multiple alternative paths, for example:

- Anonimize on one path
- Work on unanonimized data on another

Optional flow control

# JSON support

JSON is a first class citizen

Plugs nicely into the existing message & processing model

json-parser()

- Hierarchical documents → dot notation

$(format-json)

- Dot notation → hierarchical documents

# JSON on /dev/log

```
source s_local {
    channel {
        source { system(); };
        parser { json-parser(prefix(„.cee” marker(„@cee:”)); };
    };
};
```

# JSON from node.js

```
block source nodejs(localip(0.0.0.0) port(9003)) {
    channel {
        source {
            network(transport(tcp)
                    localip(`localip`) port(`port`)
                    flags(no-parse));
        };
        parser {
            json-parser(extract-prefix("[1]")
                        prefix(".nodejs.winston."));
        };
        rewrite {
            set("${.nodejs.winston.message}" value("MESSAGE"));
            set("" value(".nodejs.winston.message"));
        };
    };
};
```

# JSON to ElasticSearch

```
template(„$(format-json --scope rfc5424,nv-pairs
                        --pair @timestamp=\"${R_ISODATE}\"
                        --pair @message=\"${MSG}\"
                        --exclude DATE --exclude MESSAGE”)
```

# Transforming on Output

Structured messages come in many flavours

- syslog-ng, journald, ElasticSearch, rsyslog, RFC5424 all use name-value pairs
- „standard” formats like Apache combined also have fields

We need to represent any input format we get (and be happy!)

- syslog-ng is schemaless

But specialized systems do need a specific schema

- Kibana will not display messages unless it is in a specific dialect of JSON (e.g. @message contains the message)

„value-pairs” come to the rescue

# „Value Pairs” comes to the rescue!

# Value Pairs

A set of common functionalities built into many syslog-ng features

It allows a lightweight & simple transformation to happen before delivering the message:

- Query a set of name-value (fields) pairs using wildcards
- Add new fields, based on information already in the message
- Rename fields individually
- Rename fields based on wildcards

Exactly as needed by the target system

The message is not changed, only the output is rendered as specified

# Value Pairs in Action

```
$(format-json --scope rfc5424
              --pair @timestamp=\"${R_ISODATE}\"
              --pair @message=\"${MSG}\"
              --key .cee.*
              --exclude DATE --exclude MESSAGE)


# pull RFC5424 fields by default
# plus all values prefixed with .cee.
# add @timestamp, @message fields
# and remove MESSAGE & DATE
```

# Value Pairs in Action

```
$(format-json --scope rfc5424,nv-pairs
              --pair @timestamp=\"${R_ISODATE}\"
              --pair @message=\"${MSG}\"
              --key .cee.*
              --replace-prefix .cee.=devlog.
              --add-prefix foobar.
              --shift 3
              --exclude DATE --exclude MESSAGE)

# .cee.name → devlog.name → foobar.devlog.name → bar.devlog.name
```

# But what about field content?

# Template functions

The desired information is present

- but in a „slightly” different representation

Template functions take values from the message and transform it to something else

$(format-json) is a template function!

# Template Functions II

String functions (substr, strip, lowercase, uppercase, ...)

Numerical functions (addition, substraction, multiplication, ...)

Crypto related (md5, sha1, sha256, ..., uuid)

GeoIP, JSON

# What if I have to add my own?

# Hacking syslog-ng

syslog-ng is by all means a traditional UNIX project:

- Written in C
- Bison, flex
- Used to be monolithic, now it is plugin based

Being C, it is difficult to modify for some, especially the highly optimized parts

# Why not use a Scripting Language?

# Embedded Languages

We have experimental bindings for these languages:

- Lua (the first and most advanced)
- Python (my favourite)
- Perl (someone else's favourite :)

# Current State

It is possible to write a destination driver in any of those languages

No other functionality (template functions, parser, rewrite is possible for now)

Performance is not stellar, but is enough for most cases (Lua is best with about 50k/sec)!

# Potential

Use a highly optimized, well performing C core for the „boring stuff”: read, parse, deliver messages

Use a scripting language to prototype new functions fast

Use a scripting language to adapt it to your own environment

If performance is not enough, you can always push to C, without changing the configuration!

# Demo

# One more abstraction...

# Combining Primitives

syslog-ng has a bunch of specialized log processing functions

- parsers, rewrite rules, template functions, ...

In a lot of cases, a number of these primitives are needed for a specific task

# system() source

```
source s_local { system(); };

# This is what system() expands to:
channel {
    source {
        channel {
            source { unix-dgram("/dev/log" so_rcvbuf(8192)); };
            rewrite { set("${.unix.pid}" value("PID")
                          condition("${.unix.pid}" != "")); };
            };
            file("/dev/kmsg" program-override("kernel") flags(kernel) format(linux-kmsg));
        };
        parser {
            json-parser(prefix('.cee.') marker('@cee:'));
        };
    };
```

# Isn't `system()` nicer?

# Reusable Blocks

Making these functions reusable has a lot of benefits:

- Your configuration becomes **a lot** simpler
- Implementation details are abstracted away (system, ES)
- Complexity is hidden, **intent** becomes **obvious**
- You can use the same solutions multiple times in different projects
- Others can use it too

# syslog-ng Configuration Library (SCL)

# What else?

There's a lot of stuff going on that couldn't fit into this presentation:

- junctions and channels
- patterndb
- RSS destination
- Monitor source, Trigger source
- Graphite, Riemann, ...

Check out the incubator!

# Where do we want to go next

Make support for embedded languages deeper

- Easy prototyping, different plugin support
- Easy customization without touching C code

Application specific SCLs (e.g. Kibana destination)

Docker images

Salt/Chef/Puppet support

# Syslog-ng as a Project

Infrastructure:

- syslog-ng.org
- Github: github.com/balabit/syslog-ng
- syslog-ng Incubator: github.com/balabit/syslog-ng-incubator
- Mailing list: syslog-ng@lists.balabit.hu
- IRC: #syslog-ng channel on freenode

Open to Contributors

- SCL snippets
- Patterns
- Documentation
- Code
- Just open a pull request!

# Conclusions

Don't expect out-of-the-box experience (or only in very limited cases)

syslog-ng comes to the rescue, with its flexibility

Scripting is available, if C is not your language of choice

Open to contributors

# Questions?