# rsyslog meets Docker alles easy, oder?

Rainer Gerhards

# Agenda

- How we got started
- Some Use Cases
- Syslog Appliance Containers
- Introducing Meta-Config Layer
- Proper Structure – the “Container Stack”
- Some more Relations on rsyslog Development

# Containers? Umm, yes...

- of course, we knew about containers - but we did not care ;-)
- Had a very long TODO list with non-container things, but...
  - we have seen some users post container defs to help troubleshoot issues
  - occasionally, containers were briefly mentioned in the in issue trackers
  - and of course we knew they went into more and more widespread use

# Real Interest

- Curiosity wins…
  - isn’t it time to at least know better what’s required?
  - can we do something with them?
  - can we help our users to be more productive?
  - and a bit of “I want to know how it works in practice” (kind of pet project TBH…)
- Still we (thought we) had no real use case for ourselves, but the desire (and approval) to get going

# First Steps

- Focus: Docker (and **pure** Docker!)
- Learn to build and use containers
- First surprises:
  - Ah! Thus folks are interested in omstdout…
  - Oh, that’s why they would like to have ctl-c enabled in non-debug mode
  - Oops, the environment can be pretty unreliable (think: queue files and shutdown timeout)…

# Understanding Use Cases

- syslog appliance
  - ready to run system, setup for logging
  - user base not necessarily much interested in logging, just “get it going”
- high-end syslog server
  - main interest fresh and current functionality
  - for the experts
- rsyslog project itself
  - surprise, surprise: CI and build system
  - reproducible builds setup in easier way then VMs

# Background: rsyslog CI

- Local environment
  - Primarily manual test for single functionality
- Travis
  - Full test suite
  - Different environments (but limited to Travis images)
  - Static analyzer (clang, Coverity [daily])
  - Originally limited to Travis Images
- BuildBot
  - More environments, including Solaris
  - “Final Approval” Step
  - “costly” (time-wise)

# CI: the static analyzer problem

- We use clang static analyzer in CI
  - great for QA
  - but caused considerable pain
    - vastly different versions in CI and on WS OS
    - even worse: no way to disable false positives
    - very hard to reproduce CI failures - pain to fix
- initial “solution”: try to get everyone on the same page
- did not work out, not even in CI ...

# the static analyzer problem - II

- Next Try
  - single CI system, Buildbot only
  - exposing CI run static analysis result
  - works ok, but now requires full CI just to get analyzer result
- real solution: container with latest stable version of analyzer
  - usable everywhere, including on dev. desks
  - consistent updates, important re:false positives
  - easy to get full-fledged analysis report locally
  - thus also helps to save CI resources

# Rsyslog Logging Containers by Others

- some very well crafted containers
- often targeted to specific use cases (e.g. LogSene, Papertrail, ...)
- many provide just a base config
- some to be used with care
  - no signals (not PID 1)
  - no volumes (e.g *var/log* in container fs layer)
  - questionable queue sizes

# Our syslog apppliance

- goals
  - current rsyslog versions
  - easy to use for common use cases
  - usable without being rsyslog-literate
  - low footprint
- Obstacles
  - Restrictions in rsyslog config format
  - Availability of recent rsyslog packages (especially during development)

# Which base image?

- Alpine
  - Pretty recent packages, but not all features
  - Slim footprint
- Ubuntu
  - Relatively “fat”
  - Rsyslog daily packages available
- CentOS & Debian
  - Enterprise preference
  - Relatively new Packages, at least for CentOS

# Going Forward

- We liked Alpine pretty much
  - Small footprint is great (dev turnaround!)
  - Overall current software stack
  - Looks like often used in similar context
  - Admit some skepticism regarding musl clib
- Ubuntu in parallel
  - Easy to get current packages
  - Needed for CI purposes anyway
- Doing two in parallel showed to be a lot of duplicate work → initial concentration on Alpine

# Getting the Slim Image

- Initial plan was to build current rsyslog in image itself
- we should better have understood layers…
- While there seem to be ways to do cleanup, there is only one good solution: create Alpine packages
- Thankfully, Alpine packaging is simple
- So we ended up with building our own Alpine packages, now also published for everyone

# Meta-Config Layer

- rsyslog configuration can be complex
- some well-defined use cases
  - Forwarding to a logging service provider
  - Forwarding to a central host
  - Writing to Elasticsearch
  - Log File Storage
- Can be tackled by same method
  - Best practices defaults
  - Just some app-specific data
- Complexity can be totally removed for those cases!

```
# general container app settings:
export TZ=UTC
#export CONTAINER_SILENT=on  # do not emit startup message
export ENABLE_STATISTICS=on

# Do we write log files?
export ENABLE_LOGFILES=on # yes, we do (comment out to disable)
# Where do we write to?
# path for host-specific files is: /logs/hosts/HOSTNAME
export LOGFILES_STORE="/logs/hosts/%hostname:::secpath-replace%/messages.log"
# If you have an account with Logsene, enter your access
# information below:
export LOGSENE_TOKEN=4711
export LOGSENE_URL=logsene-receiver.eu.sematext.com

#export USE_VALGRIND=on
#export RSYSLOG_DEBUG="debug nostdout"
#export RSYSLOG_DEBUGLOG="/logs/rsyslog-internal-debug.log"
```

# Looks nice (hopefully ;-)).
# But how does it work?

- Usually transforming this into a usable rsyslog config requires heavy use of sed and friends…
  - But we want a read-only container
  - But we want a clean config process
- Thankfully we are the rsyslog developers ;-)
  - Add feature to conditionally do includes
  - Add feature to use environment vars in config
  - … and whatever else is useful …

# So we now have config like this

```
input(type="imptcp" port="514")
input(type="imudp" port="514")
input(type="imrelp" port="1601")
include(file="/etc/rsyslog.conf.d/log_to_logsene.conf"
        config.enabled=`echo $ENABLE_LOGSENE`)
include(file="/etc/rsyslog.conf.d/log_to_files.conf"
        config.enabled=`echo $ENABLE_LOGFILES`)
# we emit our own messages to docker console:
syslog.* :omstdout:

include(file="/config/droprules.conf" mode="optional")
# this permits the user to easily drop unwanted messages

action(name="main_utf8fix" type="mmutf8fix" replacementChar="?")

include(text=`echo $CNF_CALL_LOG_TO_LOGFILES`)
include(text=`echo $CNF_CALL_LOG_TO_LOGSENE`)
```

# We need a small helper script

```
export LOGSENE_TOKEN=4711
export LOGSENE_URL=logsene-receiver.eu.sematext.com
```

```
export CNF_CALL_LOG_TO_LOGSENE=
export ENABLE_LOGSENE=off
if [ "$LOGSENE_TOKEN" != "" ]; then
	if [ -z $LOGSENE_URL ]; then
		echo "Error: LOGSENE_TOKEN_URL unset"
		exit 1
	fi
	export ENABLE_LOGSENE=on
	export CNF_CALL_LOG_TO_LOGSENE="call log_to_logsene"
fi # end LOGSENE
```

```
include(file="/etc/rsyslog.conf.d/log_to_files.conf"
		config.enabled=`echo $ENABLE_LOGFILES`)
include(text=`echo $CNF_CALL_LOG_TO_LOGFILES`)
```

# Do we need root?

- Not really – just for binding to privileged ports
- Why not bind to > 1024 and let docker do the mapping?
- That way we can run the whole container unprivileged
- Of course, we need to be careful with permissions
- Current line of thought – not yet done
- Feedback from the audience???

# Goal almost reached…

- Easy to use thanks to
  - Meta-config
  - Command interface (help, edit-config, show-config, …)
- Small footprint due to Alpine & Pkgs
- Secure
  - Can be run read-only (can even be enforced)
  - Non-root?
- Dev-Friendly
  - Integrated debugging support

# Really?

- Well, kind of
- We still miss important use cases
  - Different distros
  - Client-side rsyslog
    (the infamous CentOS 7 imuxsock issue, …)
  - “bare essentials” container for rsyslog-literates
    - Base distribution
    - Current rsyslog
    - Nothing else

# Target: a container stack

- We want to build a set of containers that base on each other
  - Distro base image plus current components
  - Meta config level
- Same can also be applied to development containers
- We still face combinatorical explosion, so need to limit to important cases
- Automatted build system seems inevitable in the longer term (dockerhub sufficient…???)

# What happens if you break clear structure…

- rsyslog-doc showed what happens if you are lazy
  - rsyslog-doc is **general** doc project
  - rsyslog-website is a rsyslog-doc user, with **special format requirements**
- We use a container in doc-generation CI
- … as well as when generating the website doc

# What happens if you break clear structure… II

- Originally I wanted to get away with a single container, but
  - caused confusion
  - … and lots of discussion & lost work
- We now mirror the two different targets via two containers
  - rsyslog/rsyslog_doc_gen is used for doc project
  - rsyslog/rsyslog_doc_gen_website for site
  - they base on each other
- Lesson learned: do not breach layers!

# Open Issues: Shutdown

- Rsyslog uses in-memory queues to handle large message bursts
- On shutdown, queues can be persisted to disk – this takes (potentially a lot of) time
- Does not play well with Docker 10 sec exit timeout
- We currently address this by using smaller queues
- Question: can we enforce a longer shutdown? And shall we?

# New Challenges for the config Optimizer

- New features for meta-config like to see updates in config optimizer
  - constant expression folding
  - completely empty if conditions (special case of constant folding)
  - completely empty if then/else blocks (and control statements in general)
- Will be addressed
  - during regular development
  - Side-by-side with new meta config capabilities

# Open Issues: Permissions

- Approach: recommend that user assigns uid in config explicitly
- We want to run under the current user context
- This is yet another reason why we consider to have listening ports bind to non-privileged ports
- We would prefer to have an option to enforce this, but this seems not to be possible

# BTW: more uses inside rsyslog development

- Improve Travis Experience
  - Speed up tests (dependency install!)
  - Improve reliability (3rd party repo rebuilds!)
  - Different Distro Tests
  - Locally reproducible
  - Open issue: full testbench
- Troubleshooting
  - Different Distros (quick check)
  - “problem description environment”
- Packaging
  - Right now for Alpine
  - Other Distros in the future as well (or OBS!)

# Future Work

- still lots to do
- imfile rewrite, permit to monitor “external” files (on host machine)
- imuxsock automatic monitoring of log sockets
- full review of docker (and others!) logging infrastructure
- Building family of containers as described
- **please provide feedback – what DO you want?**

# Discussion

What would you like to see?
What would you recommend?
What are we missing or misunderstanding?

# Summary

- I have described our motivation,
- use cases we see and plan to support,
- how we could integrate containers into our own workflow,
- our strategy towards production containers
  - meta config
  - client vs. server containers
  - “Container Stack”
- And finally discussed some yet-unsolved problems

# Questions?

- rgerhards@adiscon.com

- https://hub.docker.com/u/rsyslog/
- https://github.com/rsyslog/rsyslog-docker