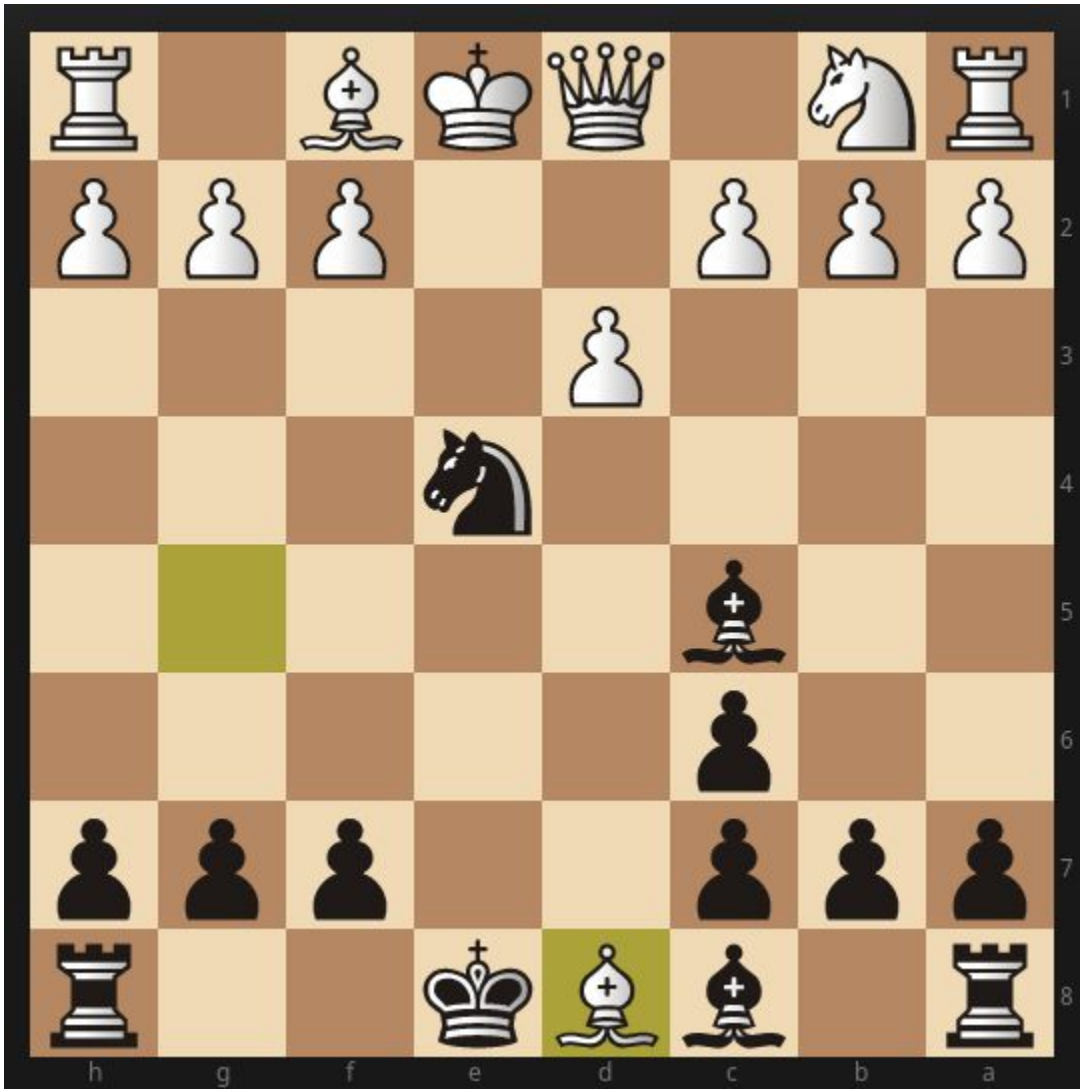


Mate in two as black.

Mate in two as black.

Lukáš Zapletal

Performance Co-Pilot and Ruby

Lessons learned integrating an app without native PCP bindings

Me and the context

Me

Lukáš Zapletal
Software Engineer, Red Hat
Red Hat Satellite 6 team

Core The Foreman team member
Fedora community member

Owner of bare-metal provisioning, logging, monitoring and SELinux

Interest in solving performance issues

The context

Foreman: software for server management:

- provisioning (bare-metal)
- VM or cloud management
- hardware discovery
- semi-automated provisioning
- configuration mg. bootstrap and inventory
- configuration mgmt. facts and reports
- remote execution
- content management (yum, puppet)
- more features via plugins

Ruby on Rails application, 10 years of existence, performance bugs from time to time.

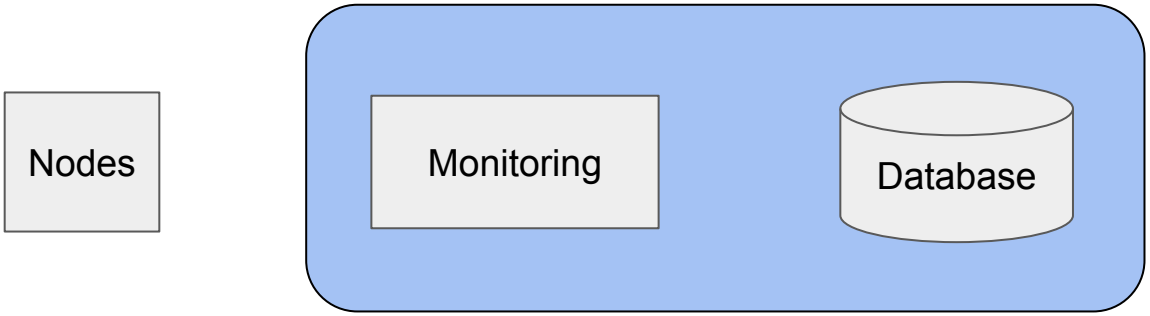
www.theforeman.org

Performance Co-Pilot

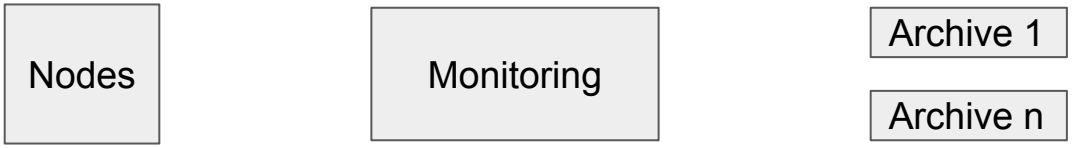
PCP is a **different** monitoring software.

Typical monitoring software

Typical monitoring software



PCP is different



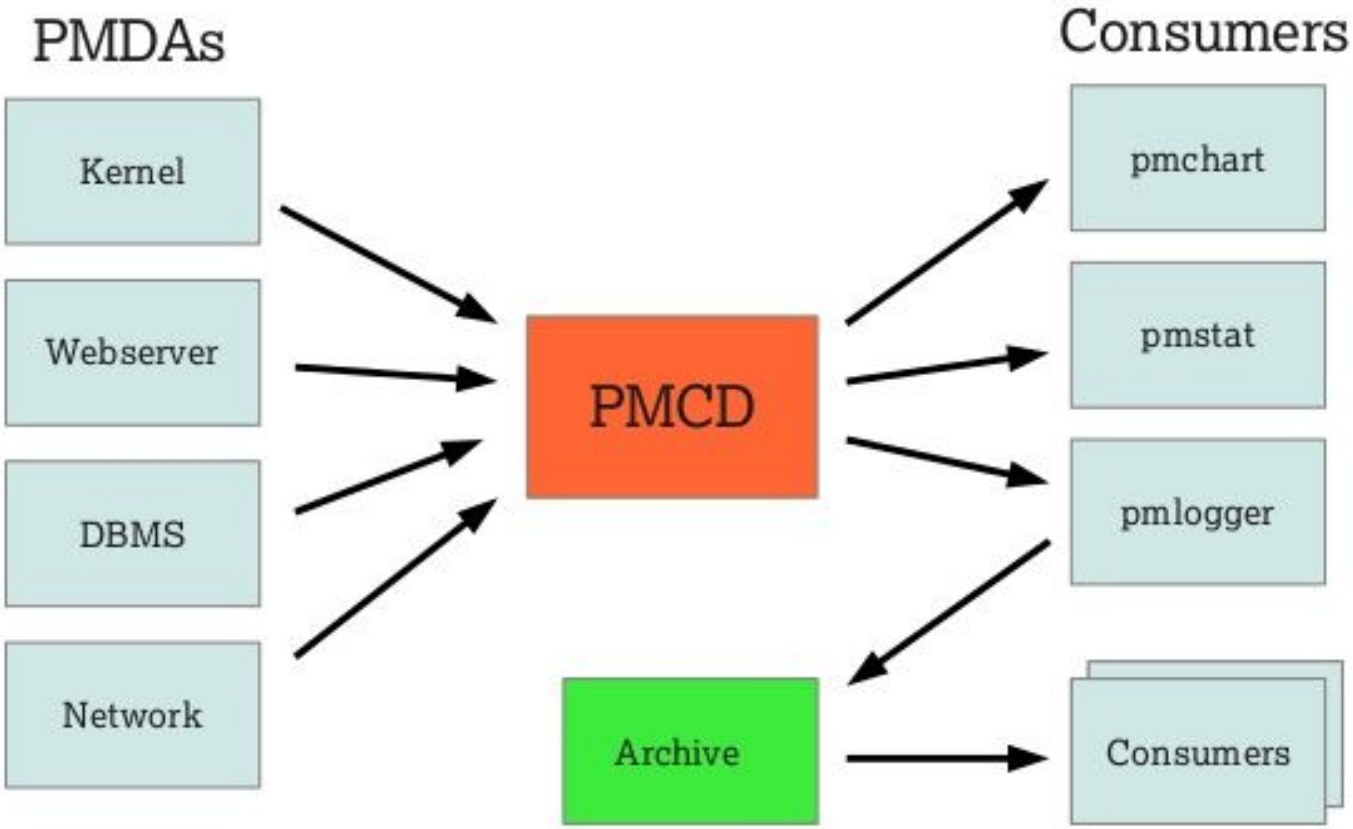
Open source framework...

... toolkit for monitoring and analyzing live and historical system performance.

Key features of PCP

- lightweight (PCP RPM about 4 MB)
- distributed (local and remote monitoring)
- included (all major Linux distributions, part of base RHEL/CentOS, BSD)
- no external database needed (daily archive files)
- metrics with type, unit and semantic (bytes/sec or kB/sec)
- live and/or historical data
- high-resolution (1 sec, in use by Netflix)
- hotproc monitoring (details from hot/picked processes)
- export to 3rd parties (Graphite, InfluxDB, ElasticSearch, Zabbix, Nagios)
- extensive command-line toolkit for analysis
- graphical tools and 3rd party dashboards available (QT GUI, TUI, web)
- many agents available (100+ packages in Fedora named pcp-pmda*)
- easily extensible with good documentation and man pages
- stable with 20+ years of existence (open-sourced Dec 1999 by SGI)

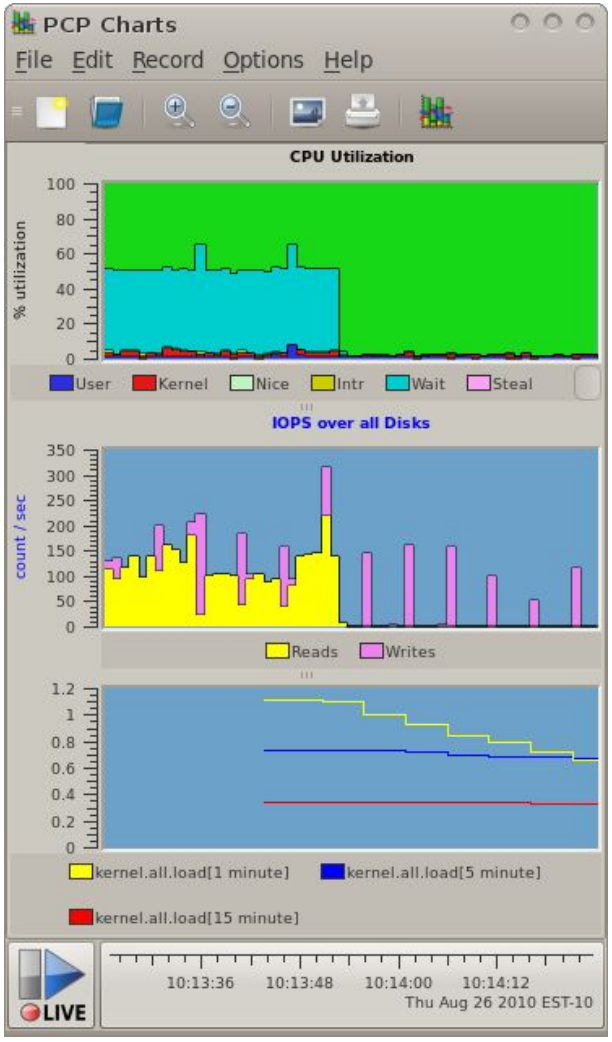
Performance Co-Pilot - Architecture



PMDA Python example

```
import time
import cpmapi as c_api
from pcp.pmda import PMDA, pmdaMetric
from pcp.pmapi import pmUnits, pmContext as PCP

class TrivialPMDA(PMDA):
    def trivial_fetch_callback(self, cluster, item, inst):
        if (cluster == 0 and item == 0):
            return [int(time.time()), 1]
        return [c_api.PM_ERR_PMID, 0]

    def __init__(self, name, domain):
        PMDA.__init__(self, name, domain)
        self.connect_pmcd()
        self.add_metric('trivial.time',
                pmdaMetric(self.pmid(0, 0),
                c_api.PM_TYPE_U32, c_api.PM_INDOM_NULL, c_api.PM_SEM_COUNTER,
                pmUnits(0, 1, 0, 0, c_api.PM_TIME_SEC, 0)),
                'time in seconds since 1 Jan 1970',
                'The time in seconds since the epoch (1st of January, 1970).')
        self.set_fetch_callback(self.trivial_fetch_callback)
        self.set_user(PCP.pmGetConfig('PCP_USER'))

if __name__ == '__main__':
    TrivialPMDA('trivial', 250).run()
```

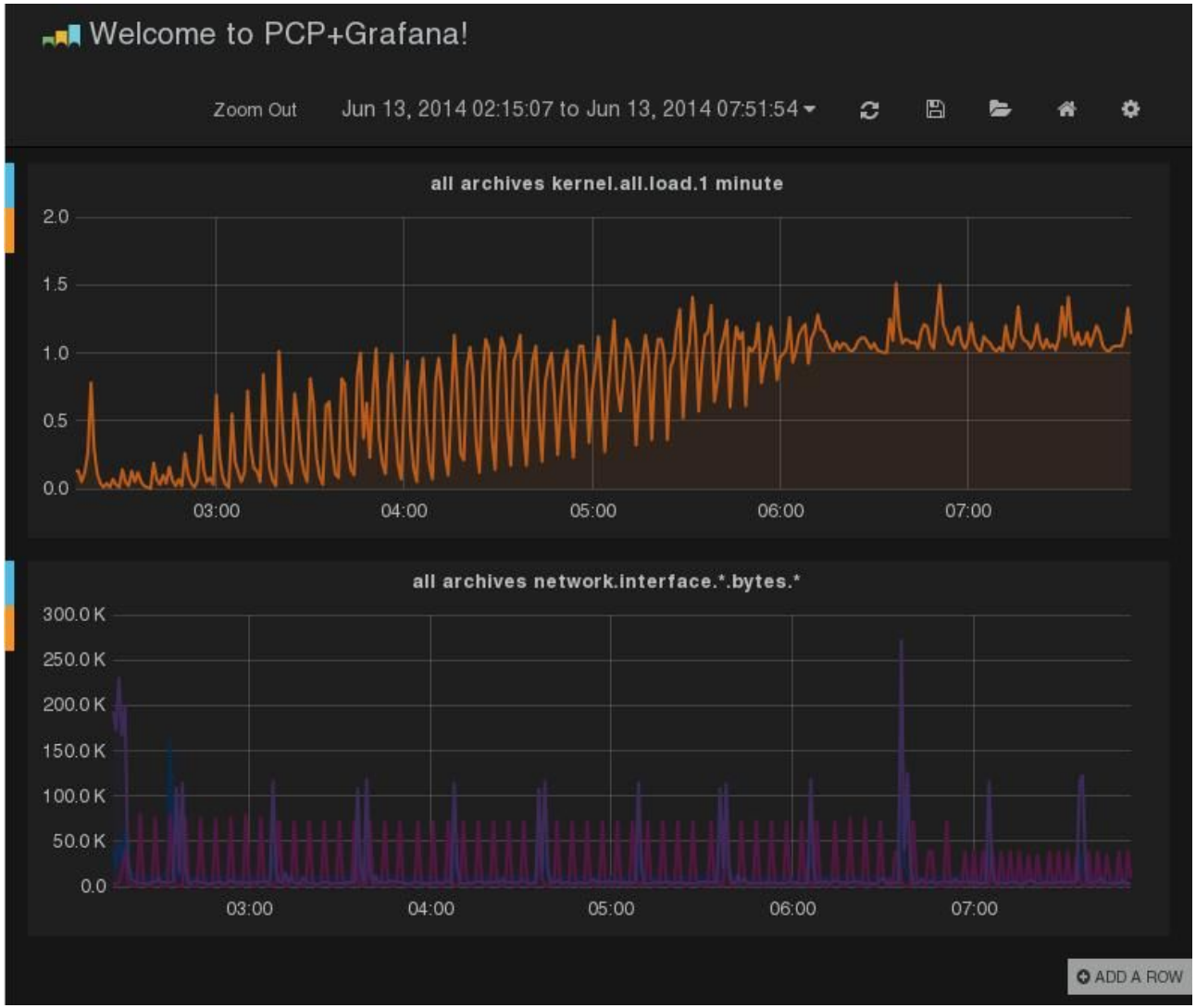
PCP - GUI


PCP Charts
File Edit Record Options Help
CPU Utilization
% utilization
100
80
60
40
20
0
User Kernel Nice Intr Wait Steal
IOPS over all Disks
count / sec
350
300
250
200
150
100
50
0
Reads Writes
1.2
1
0.8
0.6
0.4
0.2
0
kernel.all.load[1 minute] kernel.all.load[5 minute]
kernel.all.load[15 minute]
LIVE
10:13:36 10:13:48 10:14:00 10:14:12
Thu Aug 26 2010 EST-10

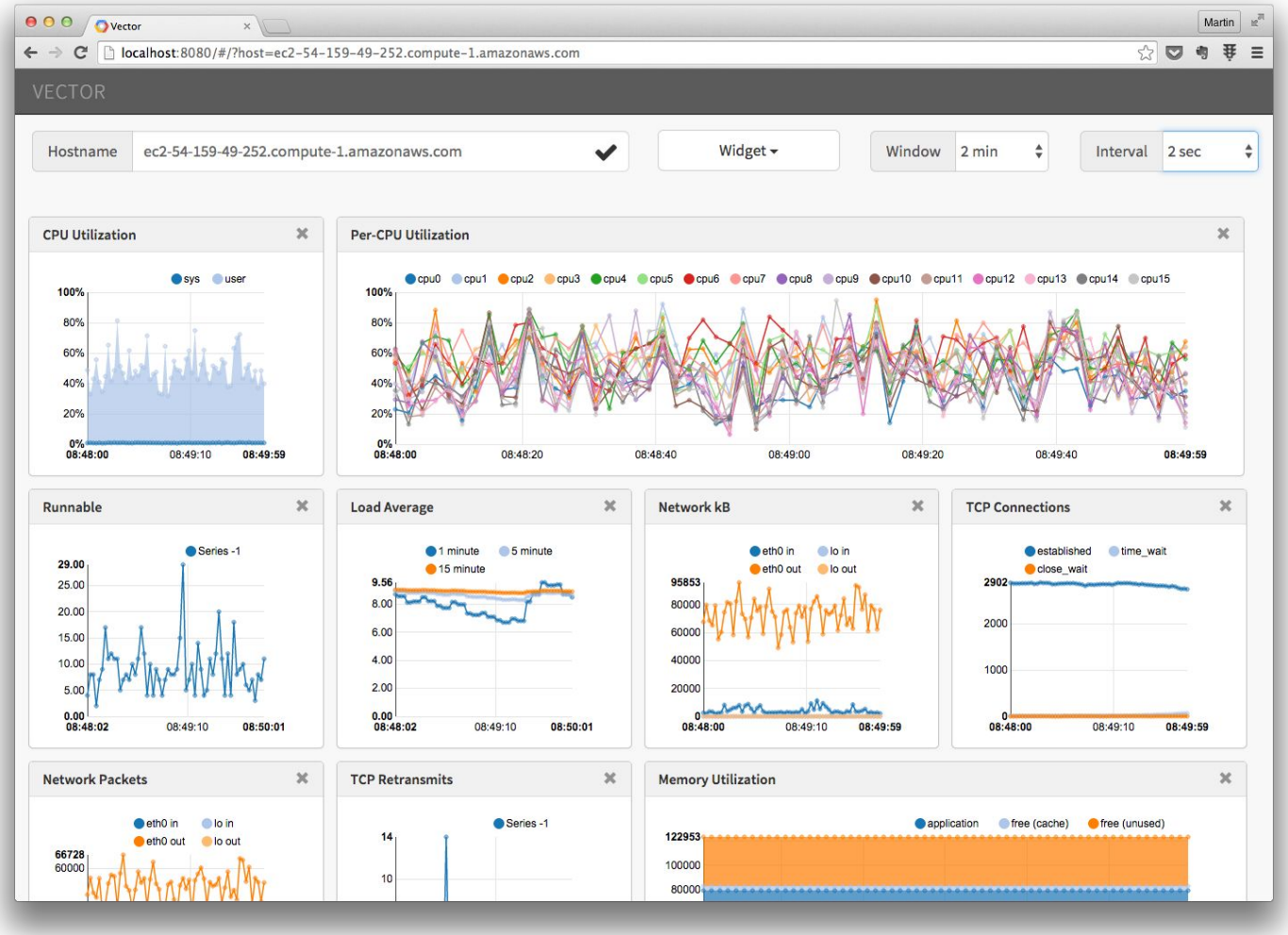
Dashboards shipping with PCP

```
yum -y install pcp-webapi \
  pcp-webapp-grafana \
  pcp-webapp-vector

systemctl start pmwebd

systemctl enable pmwebd

firewall-cmd --add-port=44323/tcp

firewall-cmd --permanent --add-port=44323/tcp
```

PCP - Grafana (historical data)



PCP - Vector (live data)



Demo time

You said Ruby?

Monitoring a Ruby app

Either Ruby or any language/runtime without native library/bindings.

APIs available in PCP:

- Writing an agent
  (PMAPI, libpmda, C/C++, Python, Perl, Java)
- Instrument app
  (MMV PMDA, MMV API, libmmv, C/C++, Python, Perl, Java, Go)
- Tracing points
  (Trace PMDA/API, libtrace, C/C++, shell: pmtrace)

Trace API way

Created a rubygem called pcptrace: https://rubygems.org/gems/pcptrace

```
yum -y install pcp-pmda-trace @development-tools pcp-devel
cd /var/lib/pcp/pmdas/trace
./Install
gem install pcptrace
```

Trace API way

```
#!/usr/bin/env ruby
require "pcptrace"

# reached a point in source code
PCPTrace::point("a_point")

# observation of an arbitrary value
PCPTrace::obs("an_observation", 130.513)

# a counter - increasing or decreasing
PCPTrace::counter("a_counter", 1)

# time spent in a transaction (or block)
PCPTrace::begin("a_transaction")
# ...
PCPTrace::end("a_transaction")
```

Trace API way

Yay, another open-source software announced:

https://lukas.zapletalovi.com/2018/03/tracing-ruby-apps-with-pcp.html

Unfortunately PCP devs told me:

- Aggregating is not flexible (fixed rolling window)
- Not a good fit for multi-process environment
- Trace API is very slow
- They are going to deprecate and remove it

Monitoring a Ruby app

APIs available in PCP:

- Writing an agent
  (PMAPI, libpmda, C/C++, Python, Perl, Java)
- Instrument app
  (MMV PMDA, MMV API, libmmv, C/C++, Python, Perl, Java, Go)
- ~~Tracing points~~
  (Trace PMDA/API, libtrace, C/C++, shell: pmtrace)

MMV API way

No wrapper needed, client code can write to shared memory directly (e.g. Golang library called Speed).

An abandoned work-in-progress Ruby client library exists, but Ruby don't support mapped memory (mmap gem is available tho).

Biggest concern is still multi-process environment.

Solution: statsd protocol and separate agent/daemon.

What is statsd

A de-facto standard - a text-based UDP protocol.

https://github.com/etsy/statsd

Three basic types: counter, duration, gauge.

Some extended types or extensions (set, string, labels).

```
api.session_created:114|ms

cpu.temp:42.3|g

db.query.success:2|c
db.query.fail:1|c
```

MMV API way

New project was born: pcp-mmvstatsd: https://github.com/lzap/pcp-mmvstatsd

- a daemon which
- listens to UDP packets
- uses golang Speed instrumenting library
- takes advantage of HDR histogram aggregation
- sends the data via MMV API to PCP
- works great
- how Foreman integrates with PCP today

MMV API way

Yay, yet another open-source software announced:

https://theforeman.org/2018/07/monitoring-and-telemetry-of-foreman-118.html

Unfortunately PCP devs told me:

- They don't like this design
- MMV was meant for instrumenting and not agents
- I should really use PMAPI and write a PMDA
- Does not map labels properly and creates hundreds of metrics
- Temporary solution

Monitoring a Ruby app

APIs available in PCP:

- Writing an agent
  (PMAPI, libpmda, C/C++, Python, Perl, Java)
- ~~Instrument app~~
  (MMV PMDA, MMV API, libmmv, C/C++, Python, Perl, Java, Go)
- ~~Tracing points~~
  (Trace PMDA/API, libtrace, C/C++, shell: pmtrace)

PMAPI way

I am mentoring an ongoing diploma work at Palacký University in Olomouc to write a proper PMDA:

- PMDA statsd
- multi-threaded high-performance design
- pluggable parser support
- pluggable aggregation
- HDR histogram
- written in C/C++
- label mapping
- configurable
- PMDA for other unsupported languages
- replacement for deprecated Trace API

1

Evolution is still better than revolution as you learn a lot along the way.

Q&A

This talk already available as
http://bit.ly/pcp-and-ruby