iVoyeur
Rediscovering collectd

DAVE JOSEPHSEN

Dave Josephsen is the sometime book-authoring developer evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

I recently saw Shadaj Laddad's talk at this year's OSCON, entitled "The Wonders of Programming" [1]. If you haven't had the pleasure, Shadaj is a 14-year-old programmer who (among many other things) wrote a bioinformatics Scala library. In the talk, he describes how he was encouraged to program computers from the age of six, and he gives helpful tips to parents and other children who are interested in pursuing computer programming.

It probably doesn't surprise any of us that Shadaj credits Lego Mindstorms as having an early impact on his understanding of systems programming. Mindstorms are, of course, actually programmable (in myriad languages), but I think many engineers come back to Legos in general when asked about toys that awakened in them a love of science, technology, engineering, mathematics, and, more generally, building things and solving problems.

In my own childhood, if Legos ever became boring, they didn't remain so for long. Again and again, as my interests changed, Legos always found a way to become relevant again. I would rediscover them when I needed a ramp to jump hot-wheels, or when we were one blaster short, and wanted to reenact *Star Wars Episode 4* from memory (an almost daily occurrence among my third-grade friends). Later, I would rediscover them when I needed just the right-sized wedge to keep my Commodore tape drive functioning or a box to house an 8088 project to prevent it from grounding out. Even last week, I rediscovered them when I was looking for a clever way to keep track of the myriad groupings of household keys [2].

Maybe it's silly, but I've often wondered over the years of using and implementing little UNIX tools that do one thing well, or more recently, Web-based micro-services architecture, what percentage of systems engineering tools and practices we owe to Legos. To be sure, modular, single-purpose primitives that can be combined to form more complex entities is just one of many design methodologies, and it's an obvious one that no doubt predates the actual creation of Lego by several thousand years.

How many times have we reinvented the monitoring system? How many times have we redefined what a monitoring system even is? I couldn't tell you, being not disposed to anthropology myself. I can tell you, however, that every time someone finds a problem for which the commonly adopted monitoring systems aren't well adapted, that person usually winds up implementing a set of primitives that meets the need. When we build new monitoring infrastructure, it seems we inevitably rediscover the building blocks, and whenever this happens, silly or not, I have to admit it feels exactly the same as rediscovering my Legos.

I've been working a lot with collectd lately, a project that, were monitoring systems Legos, would probably be a valued and coveted block. At my day job, we've just finished implementing some service-side, turnkey support for it, to remove dependencies and make it easy for our customers who happen to use collectd to ship their measurements to us out of the box, so I've been playing with collectd a great deal over the past few weeks.

Having not used it for a few years, I'd forgotten what a nifty tool it is, and having rediscovered this particular Lego block has been a lot of fun for me. Looking through my GitHub repo of articles, I'm surprised to find that I've never actually written about collectd here, which is an oversight I'd like to correct now.

(Re)Introducing Collectd

Collectd is a modular metrics collection daemon written in C. Collectd loops through a list of user-specified plugins, executing each to gather performance metrics from the OS, or locally running user-space processes. Once gathered, collectd outputs these metrics on a set interval using one or more output plugins to targets like log files, aggregation daemons like StatsD, and metrics-processing systems like Librato. Collectd is a great way to begin collecting data; it offers a ton of useful metrics for a very small operational investment.

Collectd is a great fit for you if:

- You want a flexible standalone collection agent to collect performance metrics from your systems using the standalone agent pattern.
- You're running Virtual Machine instances and want to grab per-instance CPU/Disk/Memory metrics.
- You want a simple way to collect metrics from running server processes like MySQL, Apache, Redis, Nginx, or MongoDB.
- You're looking for a well-documented, widely used and trusted open-source collection agent that is available on most Linux distributions.

Installing Collectd

Collectd gets installed on every system you want to monitor, and it's pretty simple to install. It runs as a standalone daemon process and is configured by way of a classical UNIX conf file in /etc/collectd. You can obtain and build collectd from source, but packages exist for all major distros, and most small ones. For example, on a Debian-based system, you'd enter

```
apt-get install collectd
```

How Does It Work?

Collectd generally follows the standalone agent pattern. It runs on every host you want to monitor, and it either reports directly to an upstream metrics aggregator or writes metrics to the local file system.

Starting the Daemon

Debian-based distros start collectd automatically when you install it, but if collectd isn't already running, you should be able to start the daemon using the appropriate init method for your OS, or directly by executing collectdmon. If collectd won't run, or if it appears to be constantly restarting, you can run it manually with an -f switch, which will prevent it from forking into the background.

Collectd also includes the -t switch, which tests the validity of the configuration file and is helpful for troubleshooting startup problems.

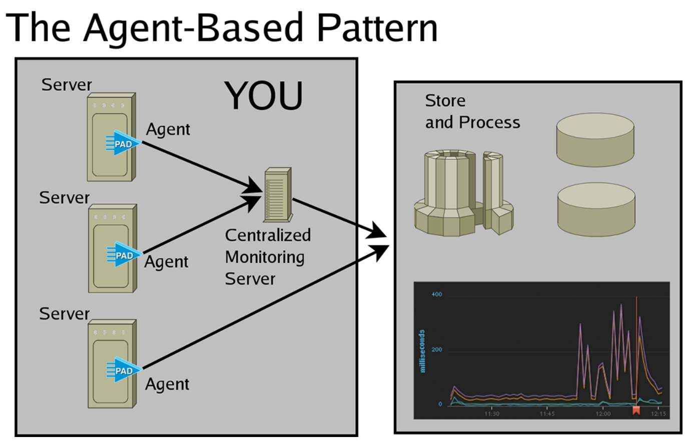

Figure 1: Collectd works as a standalone collection agent, indicated by the PAD icons in this figure.

Plugins

Collectd's behavior is dictated by two types of plugins. Input plugins gather performance data from the OS or applications running on the system. The CPU input plugin, for example, interacts with the OS to measure the same CPU-related metrics returned by the UNIX top command, like the percentage of time the CPU spends executing user-space processes or waiting on I/O.

The Nginx plugin, by comparison, queries a running Nginx server to gather metrics like the current number of requests and connection information. Users are encouraged to write their own plugins to pull data from specific resources and contribute them back to the project so others can benefit from them.

Output plugins are then used to send the gathered metrics data to other services for storage or analysis. The write_http plugin is one example of an output plugin, sending metrics data to a remote Web server in the prescribed JSON format. Other output plugins support graphing systems like RRDtool, the AMQP message transport, or even humble CSV files.

Many plugins exist for collectd. The default collectd installation on the current Ubuntu LTS (Trusty) comes preconfigured with 100 plugins, 14 of which are automatically enabled. To give you a feel for the sorts of metrics that are collected out of the box, here are all of the input plugins that were enabled on my test Trusty box by default:

- battery: for systems with internal batteries like laptops
- cpu: CPU stats (%wait, %user, etc.)
- df: file-system capacity (e.g., inodes free)
- disk: disk performance (I/O per second)
- entropy: measures the effectiveness of the PRNG
- interface: network interface (I/O per second)
- irq: times per second the OS has handled an interrupt
- load: 1, 5, and 15-minute load average
- memory: RAM usage
- processes: number of processes grouped by state (running, sleeping, stopped, etc.)
- swap: swap capacity and usage
- users: number of users currently logged in

Plugin Configuration and Dependencies

For each plugin that collectd loads, there is a LoadPlugin line in the collectd.conf file. Some plugins require only this line, although most require some additional configuration to do things like specify formats or locate files or directories in the file system.

A few plugins depend on other plugins to operate. A notable example is the JMX plugin, which requires the Java plugin to function. Settings and dependency information for each plugin are fully documented at the collectd wiki.

Mind the Polling Interval

Collectd's polling interval is controlled by the Interval attribute in the collectd.conf file. Because many upstream visualization tools make assumptions based on this interval, you should think carefully about your desired resolution, set it once and avoid changing it, and take steps to ensure that this setting remains the same on every host.

Modifying collectd's polling interval will affect the resolution of your metrics in upstream visualization systems. Some systems handle this better than others. RRDtool, for example, is heavily dependent on a preconfigured polling interval, so changing this setting could render your existing RRDs inoperable. Again, set it carefully, and then leave it alone.

Rollups with Collectd's Network Plugin

With collectd's network plugin, it's possible to specify one or more collection servers, to which all hosts emit their metrics. This can simplify per-host configuration and minimize network access control permissions, providing a means to aggregate and proxy a site-wide metrics stream by configuring the server to write to an upstream service like Librato.

A friend once told me that usually engineering was about building things, but that sometimes it was also about destroying things, to see what can be made from the parts. I think I might add that sometimes our job is to play with different kinds of parts, because playing with parts teaches us about how things could be built.

Being a building block, collectd isn't going to replace a monolithic monitoring system like Sensu or Nagios, and I'm certainly not advocating that you destroy a functional monitoring system only to rebuild it on collectd, but because most (if not all) monitoring systems can be made to accept data from collectd, it's worth playing with, whether you're already running a monolithic system or just trying to figure out what pieces fit with what. Even if you've played with it before, you might learn something new. I did.

Take it easy.

References

[1] "The Wonders of Programming": http://www.oscon.com/oscon2014/public/schedule/detail/35956.

[2] A Lego keyholder: https://www.youtube.com/watch?v=5TdYhkVutkQ.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Do you have a USENIX Representative on your university or college campus?

If not, USENIX is interested in having one!

The USENIX Campus Rep Program is a network of representatives at campuses around the world who provide Association information to students, and encourage student involvement in USENIX. This is a volunteer program, for which USENIX is always looking for academics to participate. The program is designed for faculty who directly interact with students. We fund one representative from a campus at a time. In return for service as a campus representative, we offer a complimentary membership and other benefits.

A campus rep's responsibilities include:

- Maintaining a library (online and in print) of USENIX publications at your university for student use
- Distributing calls for papers and upcoming event brochures, and re-distributing informational emails from USENIX
- Encouraging students to apply for travel grants to conferences
- Providing students who wish to join USENIX with information and applications
- Helping students to submit research papers to relevant USENIX conferences
- Providing USENIX with feedback and suggestions on how the organization can better serve students

In return for being our "eyes and ears" on campus, the Campus Representative receives access to the members-only areas of the USENIX Web site, free conference registration once a year (after one full year of service as a Campus Representative), and electronic conference proceedings for downloading onto your campus server so that all students, staff, and faculty have access.

To qualify as a campus representative, you must:

- Be full-time faculty or staff at a four year accredited university
- Have been a dues-paying member of USENIX for at least one full year in the past

For more information about our Student Programs, contact
Julie Miller, Marketing Communications Manager, julie@usenix.org

www.usenix.org/students