Container Performance Analysis at DockerCon 2017

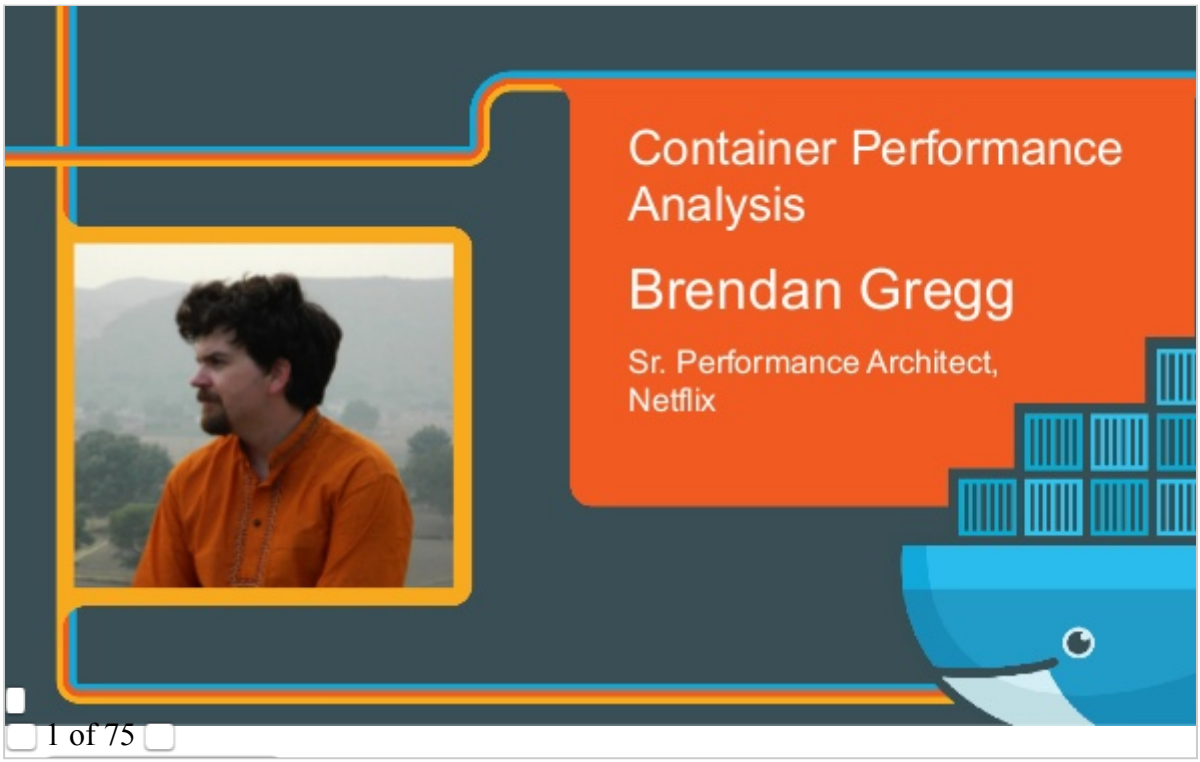
15 May 2017

At DockerCon 2017 I gave a talk on Linux container performance analysis, where I showed how to identify three types of performance bottlenecks in a container environment:

1. In the host vs container, using system metrics.
2. In application code in containers, using CPU flame graphs.
3. Deeper in the kernel, using tracing tools.

The talk video is on youtube (42 mins):

And the slides are on slideshare:



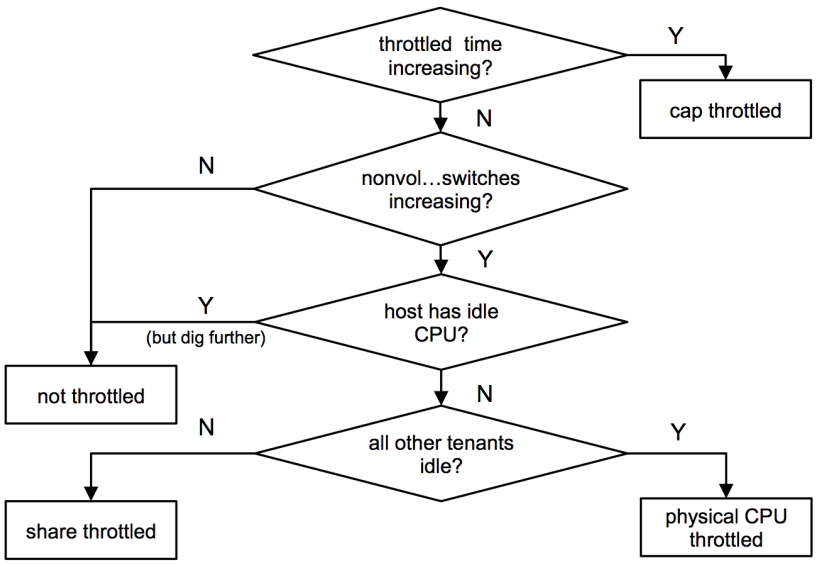
This talk was a tour of container performance analysis on Linux. I included a quick summary of the necessary background, cgroups and namespaces, as well as analysis methodologies, before digging into the actual tools and metrics. An overall takeaway is to know what is possible, not necessarily learning each tool in detail, as you can look them up later when necessary.

I included many performance analysis tools, including basics including top, htop, mpstat, pidstat, free, iostat, sar, perf, and flame graphs; container-aware tools and metrics including systemd-cgtop, docker stats, /proc, /sys/fs/cgroup, nsenter, Netflix Vector, and Intel snap; and advanced tracing-based tools including iosnoop, zfsslower, btrfsdist, funccount, runqlat, and stackcount.

Reverse Diagnosis

I'm a fan of performance analysis methodologies, and I discussed how my USE method can be applied to container resource controls. But some controls, like CPU shares and disk I/O weights, get tricky to analyze. How do you know if a container is currently throttled by its share value, vs the system?

To make sense of this, I came up with a reverse diagnosis approach: starting with a list of all possible outcomes, and then working backwards to see what metrics are required to identify one of the outcomes. I summarized it for CPU analysis with this flow chart:



The first step refers to /sys/fs/cgroup/.../cpu.stat -> throttled_time, which indicates when a cgroup (container) is throttled by its hard cap (eg, capped at 2 CPUs). Since that's a straightforward metric, we check it first to take that outcome off the operating table, and continue.

See the talk for more details, where I also included a few scenarios beforehand to see if the audience could identify the bottleneck. Try it yourself: it's hard (then try it with the above flow chart!). This may become easier over time as more metrics are added to diagnose states, and time in states, so also check for updates to cgroup metrics in the kernel.

Netflix Titus

The environment I've been analyzing is Netflix Titus, which I summarized at the start of the talk. It was covered in a post published just before my talk: The Evolution of Container Usage at Netflix.

DockerCon was fun, and a big event: 6,000 attendees. My talk won a "top speaker" award, which also meant I delivered it a second time for those who didn't catch the first one. Thanks to the Docker staff for putting on a great conference, and for everyone for attending my talk.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).