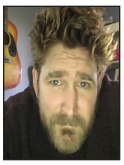
iVoyeur
Bridge Building

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

For the last year I've worn the title "Developer Evangelist," which aside from the anxiety you'd expect to accompany any large, sudden, and ill-considered career transformation, has been a fantastic experience.

I'd estimate that about a third of my time is spent in a sort of forensic exploration of my own engineering team. I comb through GitHub commit logs, chatroom scrollback, and sometimes interview my coworkers in search of narratives that I think might become good conference talks, or blog posts. Yes. They actually pay me to do this.

Once I have a story I think is interesting, I work up a proposal that describes it, and then I go about trying to find a conference that would be a good fit for the story. A talk about how we arrived at our data storage schema might interest the attendees of LISA, while a story about a microservices re-architecture might be more interesting to the audience at Velocity. I submitted over 40 CFPs in 2014, and wound up giving 12 conference talks last year—a staggering number, at least to me—and every single time I was surprised by what the conference organizers accepted and rejected.

Being an interloper in their midst—someone invited from the outside to tell a story—accentuates the otherness between us and awakens in me the anthropological observer. I see micro-community everywhere these days, a side effect of my almost weekly personal interaction with these regional, fleeting, ad hoc flocks of people who compute, but in some ever slight but critically important way differ from how I compute.

Entire conferences of Software Architects, or people who Ruby. Pythonists focused on data science, or PHP mobile Web developers, or software engineers fighting for diversity, all the same, and yet drastically different. Like Bedouin brought suddenly together by the hundreds or thousands for a few short days. The punctuated equilibrium of nerd community evolution. Each event feels so rare, and yet there are so many nerd microcommunities that their gatherings occur by the thousands each year. Even among each of them, there are smaller micro-communities—groups that cluster around this or that tool, methodology, or model of thought.

When Turing wrote about universal machines, I wonder whether he had any notion of the heterogeneity of human social interactions they would foster. Speaking of universal machines, consider the JVM, for example. Not only does the JVM itself have its own conference [1], and therefore its own community, but nearly 20 different programming languages run on top of the JVM [2], almost all of which have their own conference and community.

There's certainly some overlap, but it's difficult to imagine one person who is both an avid participant in the Rhino AND Jacl communities (much less a JVM uber-linguist who participates in them all). So we can think of the JVM as a sun at the center of a solar system of communities. Many of which, like JRuby, simultaneously orbit other stars.

Here is yet another way that the JVM is a fascinating piece of software: it creates tribalism by encouraging dissent and competition with respect to things like language semantics, while at the same time giving everyone a common ground to stand on by abstracting away the uncontroversial: memory management, parallelism, and garbage collection.

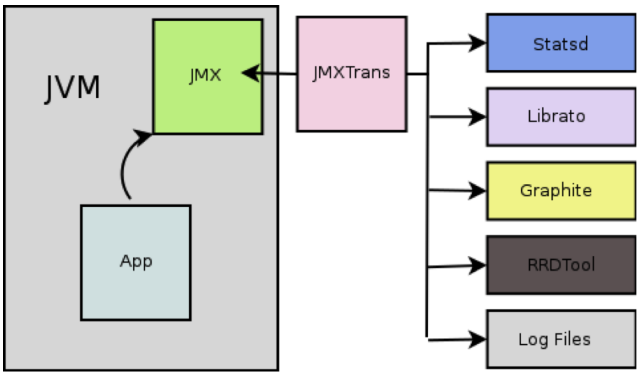


Figure 1: Jmxtrans follows the centralized polling pattern.

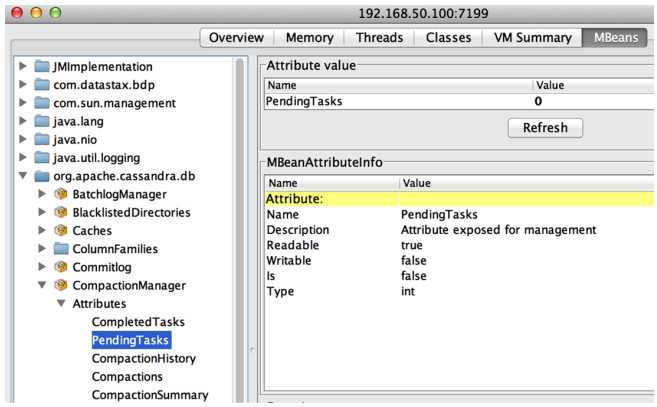


Figure 2: Exploring Cassandra's JMX metrics with JConsole

Given the (supposed) topic of this column, monitoring might have occurred to you as another common ground the JVM languages share. To be sure, JMX provides a single means of extracting monitoring and metrics data from any language or application that runs on the JVM. But monitoring is also a community of its own, with its own fair share of dissent and competition. This month, I thought we'd look at jmxtrans, a small tool that bridges these worlds, by extracting JVM metrics and shipping them to the monitoring tool of your choice.

In its own words, jmxtrans is: "the missing connector between speaking to a JVM via JMX on one end and whatever logging/monitoring/graphing package that you can dream up on the other." If you aren't familiar, Java Management Extensions (JMX) is the formally provided channel for exporting metrics from applications running inside the JVM to other processes. Jmxtrans acts as a glue-layer between an application that exposes data via JMX and various monitoring and metrics tools that import and make use of that data.

As you can see in Figure 1, jmxtrans is a specialized centralized poller. It periodically polls a running JMX process, grabbing the metrics you're interested in and emitting them (via a series of output writers) to your monitoring system. Because it polls JMX, it obviously requires that JMX be enabled on the JVM that's running your application.

You can enable JMX by specifying several options to the JVM when you start it. The simplest (and also most insecure) configuration consists of these four options:

- Djava.rmi.server.hostname= <IP Address>
- Dcom.sun.management.jmxremote.rmi.port= <PORT>
- Dcom.sun.management.jmxremote.ssl=false
- Dcom.sun.management.jmxremote.authenticate=false

A minimally useful configuration should use authentication and SSL, both of which depend on your environment. See the official JMX documentation for secure configuration information [3].

Once enabled, the JMX service will be available on the IP and port that you specify. Many different tools can interact with a running JMX service, including jmxtrans, jvisualvm, and JConsole.

Because the specific metrics available depend on what your application exports, you'll need to use a tool like JConsole to explore the JMX service. I'm using it in Figure 2 to explore some of the metrics exported by an Apache Cassandra server with JMX enabled. JConsole is included with the JDK, so you shouldn't need to download or install anything to get it running. Connect to a running JMX instance with:

```
jconsole <IP>:<PORT>
```

The names in the main window will correspond to the metric names I'll refer to later when I configure jmxtrans. The hierarchical list in the left windowpane corresponds to jmxtrans's obj configuration attribute, while the list on the right corresponds to attr names.

Even if your application doesn't export any metrics at all, JMX will still emit helpful memory-management, CPU, and thread-usage metrics from the JVM, as you can see in Figure 3. These metrics are otherwise difficult to obtain from a running JVM.

Once you have JMX enabled for the application you want to monitor, and you've decided on the metrics you're interested in graphing, you're ready to install jmxtrans.

How Do I Install It?

Jmxtrans is hosted on GitHub. You can download prepackaged versions for Red Hat or Debian systems, or a source code zip file from the downloads page [4]. The Red Hat and Debian packages both ship with an init script, which makes it easy to start or stop the jmxtrans service on those systems, and a config file in /etc that you can use to modify the behavior of the jmxtrans daemon itself (including its polling interval).

The jmxtrans service reads from configuration files placed in /var/lib/jmxtrans. Here's an example of a JSON config for

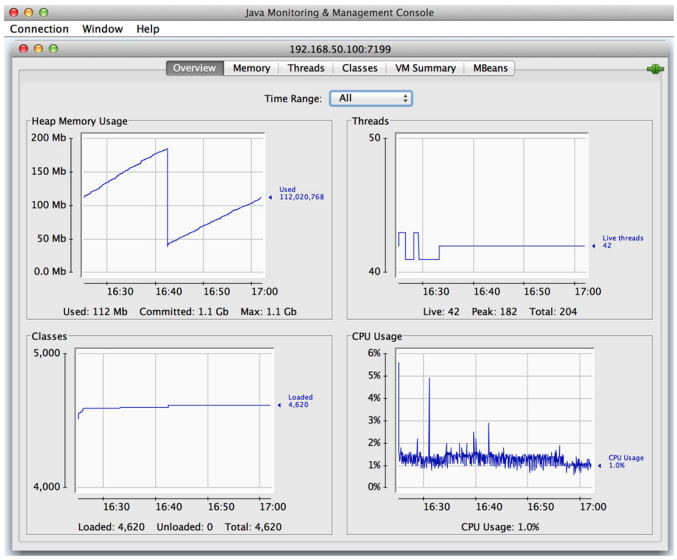


Figure 3: JMX automatically exports CPU, thread, and memory for every application that runs on the JVM.

jmxtrans that'll capture a couple of Cassandra metrics that refer to compactions on a single Cassandra node, and send them to Librato.

```
{
  "servers" : [ {
    "host" : "192.168.50.100",
    "port" : "7199",
    "queries" : [ {
      "obj" : "org.apache.cassandra.
db:type=CompactionManager,*",
      "attr" : [ "PendingTasks", "TotalBytesCompacted"],
      "outputWriters" : [ {
        "@class" : "com.googlecode.jmxtrans.model.output.
LibratoWriter",
        "settings" : {
           "username" : "dave@librato.com",
           "token" :
"cd4b234567545cb2453243qcbt546dbd43d5371"
        }
      } ]
    } ]
  } ]
}
```

Jmxtrans uses *output writers* to emit metric data to a specific monitoring system. In the example above, we're using the Librato output writer, but many are available to push metrics to systems like StatsD, Graphite, RRDtool, and flat-files. Since the outputWriters attribute is a JSON array, you can specify more than one output writer for a given series of metrics, and jmxtrans will emit to all of them simultaneously. You should start to see data in your interface of choice in a few seconds (Figure 4).

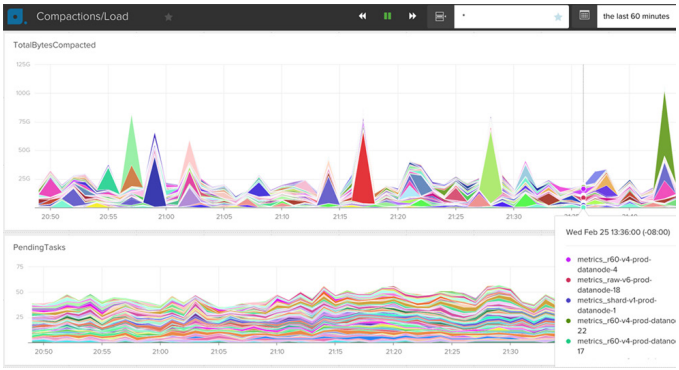


Figure 4: Data should appear in your metrics interface within 30 seconds.

Jmxtrans will also emit stack-traces and error logs into a log file in /var/log/jmxtrans. Check this file if you don't begin to see metrics appear in a few seconds.

Caveat Emptor

At the time of this writing, the jmxtrans project was in the process of moving their build process from Ant to Maven, and, as a result, the Librato output writer was not included in the packaged versions of jmxtrans. If you're installing jmxtrans from one of the pre-packaged distributions, and the Librato writer is not available, you can work around it by cloning this [5] jmxtrans repository, and building and installing it manually with:

```
maven clean install -Pdpkg
```

This notion I've been riffing on, that compelling software inevitably begets community, implies another way of looking at glue-code like jmxtrans in general. These tools, the ones that everyone uses but none of which will ever have anything like a conference of its own, form the substrate that joins communities together. As someone who has spent many years working on glue code, that's a comforting thought.

Take it easy.

References

[1] JVM Language Summit: http://openjdk.java.net/projects/mlvm/jvmlangsummit/.

[2] List of JVM Languages: http://en.wikipedia.org/wiki/List_of_JVM_languages.

[3] Secure installation of jmxtrans: http://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html.

[4] Jmxtrans download page: https://github.com/jmxtrans/jmxtrans/downloads.

[5] Jmxtrans developer fork: https://github.com/praste/jmxtrans/tree/dpkg.