Vertex Labs

Remote Logging

Getting started with the Vertex Labs Remote Logging framework for Java

User guide

Table of Contents

Logging detail view

Double clicking on the detail entries will copy that text onto the clipboard.

Quick filter

You can do compound queries in the quick filter. For example

+one +two –three

Will display any events containing the words one and two but not three in any of their fields. If you need to filter for the plus and minus characters themselves, you can escape them using ‘\’. For example

+Timer\-2 +record 2

will find any events containing ‘Timer-2’ and ‘record 2’.

Highlighting

You can provide custom background colour highlighting based on phrases in the log events. This feature is fairly simplistic at the moment, and is configured via the frontend.properties.xml by adding something along these lines:

```
<highlighters>
      <phraseHighlighter colourHex='0xff0000' phrase='red'/>
      <phraseHighlighter colourHex='0x00ff00' phrase='green'/>
      <phraseHighlighter colourHex='0x0000ff' phrase='blue'/>
</highlighters>
```

This configuration will make anything with the word ‘red’ in it appear with a red background, and blue for ‘blue’ and green for ‘green’. There is no regular expression support yet, and the string contains check is case sensitive.

Charting

Parsers

The parsers section in the xml file allows you to specify regular expressions that will match log event messages and extract important values.

Chunkers

In order to provide per second information, the charting features use the concept of Chunkers. These are responsible for taking in a timestamp and 'chunking' it into a defined interval. A chunker with an interval of 1000ms (1 second) will throw away the milliseconds elements of the timestamp and result in the information being processed against the second – this allows us to sum/mean/total all of the values received within that interval to provide aggregated results on the charts. The most common chunker interval is 1000ms which provides 1 second granularity. You can create as many chunkers as you want; maybe 1 second is overkill for your application and you can live with 30 second updates, or maybe you might want to go down to a lower granularity for lower latency applications. One word of warning – the lower the granularity, the more load you'll be putting on the LogViewer, so be careful with lower values.

Parser

The parser element lives inside the Chunker and represents a configurable label generator. It expects a formatting instruction, for example :

```
<format value="{host}/{source}/{label}" />
```

This instruction says you'd like the captured results to look like

myServer/myApplication-instance1/ordersProcessed

This format is useful if you want to differentiate between different hosts and different source applications. Maybe you want to sum the number of orders processed across all hosts and instances; you would use a formatting instruction of `<format value="{label}" />` so it will not include the other information in the label.

Patterns

The other sub element of the parser contains the most important aspect, the pattern definitions. These explicitly define the logging messages you'd like to capture. There are two styles of message you can capture:

- Lines with variables
- Lines without variables you'd just like to count

The first example is the more complex; if we wanted to chart orders processed again, and our application happened to log this at information level :

"Order from user 'shopper' processed successfully in 43ms with 3 items in their basket"

We could use this pattern to capture it:

<pattern value="Order from user '{}' processed successfully in {orderTime}ms with {basketSize} items in their basket"/>

We could then use the keywords *orderTime* and *basketSize* in a charting matcher definition to plot these values. Notice that the first empty pair of braces is required as it is a variable aspect of the line, but it isn't named as currently only numeric values can be captured.

The second type of message capture is suited to slightly less informational logging lines, for example:

"Order successfully dispatched to payments system"

This provides no variable fields to distinguish it, but you still might want to chart the number of times this happens per second. To do this you can use the simple pattern approach :

```
<pattern value=" Order successfully dispatched to payments system "
name="ordersDispatched"/>
```

You would then use the keyword *ordersDispatched* in your charting matcher, remembering to use the *Count* aggregator.

Troubleshooting

This is probably the most troublesome part of the logging viewer – getting the exact string syntax with {keyword} sections in exactly the right place is a very precise science. Behind the scenes the braces are replaced with (.*?) regular expressions, and *some* other reserved regular expression characters are escaped, but there maybe certain lines that will be tricky to parse unless you escape other aspects yourself using '\'. You can also set *debug="true"* on an individual pattern; this will dump out the messages being processed and also the regex being used to match them. This can often point out where your pattern isn't quite matching the logging line and help point you in the right direction to fix the pattern.

If you don't want any replacement of regex keywords you can use *cleanup="false"* to ensure that you regex strings get passed in without changes.

SocketTextReader

This is a hub feature that allows you to stream any text source into the hub. This is useful in situations where you want to distribute information that comes from a source can't support the standard java logging interfaces. To demonstrate this feature we'll use an example that can be really useful for monitoring linux or unix systems; streaming vmstat information from your server machines into the logging stream.

To enable the socket text reader, you need to provide the following settings to your hub VM before it starts up.

- -DsocketTextReader.active=true
- -DsocketTextReader.port=<port to bind to, defaults to 58780>
- -DsocketTextReader.level=<integer value for the logging level at which to log the incoming text>
- -DsocketTextReader.startString=<string that will be prepended to the start of the incoming lines, defaults to "socketTextReader : ">
- -DsocketTextReader.endString=<string that will be appended to the end of the incoming lines, default is blank>

The log events generated by the SocketTextReader have the following properties:

- They are timestamped with the hub local time
- The source address is set to the host IP of the client socket – this means you can't easily differentiate between two sources streaming from the same machine based on the source
- The source host is set to the host name of the client socket
- The message field is populated with text lines coming in from the socket (using BufferedReader's expectations of what an end of line looks like)

Launching the hub with those settings will bind it to the additional listening port. You can quickly test it using telnet; it should echo whatever you type into the logging stream so you can see it using the LogViewer.

Going back to our example, I'm going to choose –DsocketTextReader.startString=vmstat: to give us some context on the logging lines (and make it easy to chart it in a few minutes.) Now run this command from your linux/unix machine:

```
vmstat 1 | netcat hub port
```

where hub is the host or IP of your hub machine, and the port value matches whatever port you chose. This should result in vmstat information streaming into the logging hub. From here you can add some charting features using this parser information:

```
<pattern
value="vmstat:\s+{runQueue}\s+{blocking}\s+{virtualMemory}\s+{freeMemory}\s+{bufferMemory}\s+{
cacheMemory}\s+{swappedIn}\s+{swappedOut}\s+{blocksIn}\s+{blocksOut}\s+{interupts}\s+{contextS
witches}\s+{userTime}\s+{kernelTime}\s+{idleTime}\s+{waitingTime}" cleanup="false"/>
```

NOTE – DIFFERENT LINUX/UNIX DISTRIBUTIONS WILL FEATURE SLIGHTLY DIFFERENT COLUMNS, SO TAKE THIS AS AN EXAMPLE RATHER THAN A DEFINITIVE SOLUTION!

You could then construct a charting page along these lines :

```
<page title="vmstat" rows="4" columns="4">
  <chart title="CPU - user time" showLegend="false">
        <matcher value="*/userTime/Mean"/>
  </chart>

  <chart title="CPU - kernel time" showLegend="false">
        <matcher value="*/kernelTime/Mean"/>
  </chart>

  <chart title="CPU - idle time" showLegend="false">
        <matcher value="*/idleTime/Mean"/>
  </chart>

  <chart title="CPU - waiting time" showLegend="false">
        <matcher value="*/waitingTime/Mean"/>
  </chart>

  <chart title="Procs - Run queue size" showLegend="false">
        <matcher value="*/runQueue/Mean"/>
  </chart>

  <chart title="Procs - Blocking processes count" showLegend="false">
        <matcher value="*/blocking/Mean"/>
  </chart>

  <chart title="System - interrupts" showLegend="false">
        <matcher value="*/interrupts/Mean"/>
  </chart>

  <chart title="System - context switches" showLegend="false">
        <matcher value="*/contextSwitches/Mean"/>
  </chart>

  <chart title="Memory - virtual memory used" showLegend="false">
        <matcher value="*/virtualMemory/Mean"/>
  </chart>

  <chart title="Memory - free memory" showLegend="false">
        <matcher value="*/freeMemory/Mean"/>
  </chart>

  <chart title="Memory - buffer memory" showLegend="false">
        <matcher value="*/bufferMemory/Mean"/>
  </chart>

  <chart title="Memory - cache memory" showLegend="false">
        <matcher value="*/cacheMemory/Mean"/>
  </chart>

  <chart title="Swap - memory swapped in from disk" showLegend="false">
        <matcher value="*/swappedIn/Mean"/>
  </chart>

  <chart title="Swap - memory swapped out to disk" showLegend="false">
        <matcher value="*/swappedOut/Mean"/>
  </chart>

  <chart title="IO - blocks received" showLegend="false">
        <matcher value="*/blocksIn/Mean"/>
  </chart>

  <chart title="IO - blocks sent" showLegend="false">
        <matcher value="*/blocksOut/Mean"/>
  </chart>
</page>
```

Java web start

Its quite nice to be able to host your logging viewer configuration – with all the hub details and parsers etc – on a webserver to allow you whole team access to the latest configuration. This is quite easy to do with java web start, assuming you have access to a web server. Here's a few steps to point you in the right direction:

1) Create a folder on your webserver to host the web start files
2) Copy the following files into that folder
   a) vertexlabs-all-1.0.xxx.signed.jar
   b) frontend.properties.xml
   c) parsers.xml
   d) swingFrontEnd.properties
3) Create a new file called logViewer.jnlp with the following contents

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+"
   codebase="http://localhost/"
   href="logViewer2.jnlp">
   <information>
      <title>Remote Logging Viewer</title>
      <vendor>Vertex Labs</vendor>
   </information>
   <resources>
      <!-- Application Resources -->
      <j2se version="1.5+"
          href="http://java.sun.com/products/autodl/j2se"/>
      <jar href="vertexlabs-all-1.0.xxx.signed.jar" main="true" />
   </resources>
       <security>
    <all-permissions/>
  </security>
   <application-desc   name="Remote Logging Viewer"   main-class="com.jamesshaw.logging.frontend.SwingFrontEnd">
        <argument>http://localhost/swingFrontEnd.properties</argument>
        <argument>http://localhost/frontend.properties.xml</argument>
        <argument>http://localhost/parsers.xml</argument>
    </application-desc>
</jnlp>
```

4) Replace the *vertexlabs-all-1.0.xxx.signed.jar* with the actual name of the jar you are using
5) Replace *http://localhost/* with the actual URL path people will use to access the files on the web server

You should be able to type the full url to the .jnlp file into your browser to download the file. From there double clicking on the .jnlp file should launch the application using the settings you've provided in the files hosted on the web server. The application requires access to the hubs you have configured in the frontend.properties.xml, which means that it cannot operate in the standard sandbox – so you will have to accept the security certificate check at the start. (The jar is signed with a temporary self-generated certificate at the moment.)