**Setup LIVE MONITOR**

**Execution (SERVER):**

Start Script: **runZeeKMonitor.sh**

ExampleLocation:**/home/workspace/aman/ZeeKMonitor/bin**

Stop Script is killing following process

ChartProcessor

DataProcessor

dataCollector.sh

Start Script will start following in background and noHUP mode.

**DataProcessor.java**

**ChartProcessor.java**

**dataCollector.sh**

Both of them required **chart.properties** as argument to be executed.

*Setup a new chart:*

**Step 1**: **create a chart in chart.properties** . Sample shown below.

chart.count=2 Total number of charts to be processed, increase this by 1 to create new chart

chart.0.name=Response Time Chart Title of first chart; first chart is chart 0

chart.0.columns=TIME,RT Column names of this chart

chart.0.maxdata=500 Number of rows to be preserved to process chart

chart.0.receiveport=7703 This port number will be exclusively used to receive this chart’s data

chart.0.rawdata=/home/workspace/aman/ZeeKMonitor/out/RT.out

This is the raw file created from data received via chart.0.receiveport

chart.0.png=/usr/share/apache-tomcat-6.0.37/webapps/Monitors/charts/RT.png

This is the processed PNG file which is finally displayed @webpage.

chart.0.refresh=30

This is the rate at which chart is recreated from dump.

#

chart.1.name=BTPS

chart.1.columns=TIME,BTPS

chart.1.maxdata=500

chart.1.receiveport=7803

chart.1.rawdata=/home/workspace/aman/ZeeKMonitor/out/BTPS.out

chart.1.png=/usr/share/apache-tomcat-6.0.37/webapps/Monitors/charts/BTPS.png

chart.1.refresh=30

**The green fore-color above is not part of chart.properties.**

**Example Location of chart.properties (/home/workspace/aman/ZeeKMonitor)**

**Step 2:**  **Setup a Client to send the data**

This could be a simple script. What does server’s port need is COMMA separated values using UDP.

For example ***chart.0*** above just needs

“12:23:43,4564\n” (without quotes and a must carriage return at the end)

There are two ways to write your own script in any language to send this data

**1. Time from Log itself**

\* Sleep for XX seconds.

\* cut and grep the required value to be transmitted in such a way that time is also captured

\* loop all

**Example Snippet (perl):**

`tail -100 /tmp/XXXXX.log | grep YYYY | cut -d" " -f3,15 | grep "^..:..:.." | sed 's/ Avg:/,/g' > a.txt`;

$newData = `tail -1 a.txt`;

*Example Explanation*

The above example, RT in mentioned format along with “\n” at the end 

Check “3,15” field extracted which is TIME and Avg:XXXX from the logs.

The script replaces **”Avg:”** with **”,”** (comma)

**2. Auto-Generated Time from script**

\*Sleep XX seconds.

\* Calculate time in HH:mm:ss, store it in a variable ***data***

\* cut/grep required value to be sent to server and append extracted value to the data above with “comma”

**Example (perl):**

`tail -100 /tmp/XXXXX.log | grep YYYY | cut -d" " -f15 | cut -d: -f2 > ${tempFile}`;

$newData = `tail -1 ${tempFile}`;

($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(); #perl way of getting local time

$newDataToSend = sprintf("%02d:%02d:%02d,%s",$hour,$min,$sec,$newData); #append all carefully

***#IMPORTANT: the terminal \n is also sent!***

Don’t worry about the programming language here, the ***$newDataToSend*** is the data being sent to the server which has self generated time followed by comma and followed by the data to be sent.

***Sample Client script is also supplied***

***Summary:***

The client script running at client is sending data to server. The process

DataProcessor is catching all the data. ChartProcessor is processing all raw data into PNG files.

There is another script dataCollector.sh which is running in server and is basically tailing last XX lines from dump data into another file for ChartProcessor to process (managed by variable chart.X.maxdata)

A sample perl function has been created to send UDP data to server

sub sendData{

my $sock = new IO::Socket::INET(PeerAddr => ***$zeekMonHost***,

PeerPort => ***$port***,

Proto => 'udp', Timeout => 1) or die('Error opening socket.');

print $sock @\_[0];

}

The bold-italic variables needs to be set and the subroutine can be used as

sendData($newDataToSend);

Purpose of Step#02 is to keep running always and keep always sending data.

**Step 3**: Upgrade LIVE-MONITOR chartdisplay.properties to include new Chart

The tomcat where the **Monitors.war** is deployed ...change following config file

**…./webapps/Monitors/WEB-INF/classes/config/chartdisplay.properties**

**Contents:**

HTML.TITLE=Live Monitoring

TOTAL.GROUPS=1

GROUP.1.NAME=ABC

GROUP.1.ROWS=2

GROUP.1.L1.IMG=charts/RT.png

GROUP.1.R1.IMG=charts/BTPS.png

GROUP.1.L2.IMG=charts/PassRate.png

GROUP.1.R2.IMG=charts/TPS.png

The above properties will produce following web page

*\*\*\*\*\*\*\* ← Chart is removed to preserve IPR*

With normal deployments the URL for this page would be

[http://XX.XX.XX.XX:8080/Monitors/LiveMonitors?group=](http://10.134.84.61:8080/Monitors/LiveMonitors?group=R3)ABC

**Points:**

\* On addition of new charts, restart chart server. ***On a restart, all the old data is lost***

\* Monitors.war is used is the doc, the build and deployment step can change location and name of the web archive based on final process and naming.

\* Bugs could stop chart processing, probably bug would be bad data collected in dump. To troubleshoot, check the “out” location of dump file to see what data was wrong. Modify client (and server scripts) accordingly.

\*