# RTView

## Technical Note

SL Corporation 18 January 2016
TN-712

# How to Bring TIBCO Monitoring Metrics and Alerts into Splunk Dashboards Using RTView

**RTView Enterprise Monitor is a mature and robust platform that collects, analyzes, and archives monitoring data from a broad range of middleware products, from TIBCO and other vendors, as well as open source solutions. This article describes in detail how any of the current or historical metrics collected by RTView, as well as the alerts it generates, can be imported and made visible in custom dashboards created using Splunk. This permits Splunk users to quickly get visibility into a broad range of middleware monitoring information, without having to develop and maintain the collection and analytics functions already contained within RTView.**

## RTView and Splunk – Background

RTView has become the de facto standard for monitoring complex applications built around middleware components such as the TIBCO suite of messaging, business process, and analytics products. Many TIBCO customers use RTView component monitoring products purchased through TIBCO, or the complete and comprehensive RTView Enterprise Monitor suite obtained directly from SL Corporation.

Despite strong competition from open source, much of IT has embraced Splunk for its powerful log analysis features, as well as its ability to ingest, store, visualize, monitor, and analyze virtually any type of data. Splunk provides a simple UI for building dashboards, and many API/SDK options which have spawned a variety of user-supported apps and add-ons (available for download at https://splunkbase.splunk.com/).

However, you will find little support for TIBCO-centric monitoring data in splunkbase. This is partly due to the fact that it's simply not that easy to do it well. In order to monitor TIBCO apps, you need to use the proprietary "Hawk" protocol and EMS Admin API, and perform additional processing on the collected metrics. SL's RTView dataserver natively supports Hawk and makes it painless to collect metrics for BusinessWorks, BusinessEvents, EMS, and other TIBCO solutions. In this technical note, we describe a simple and cost-effective way to get this important data into Splunk.

## RTView Dataserver Basics

The RTView dataserver is the collection component for the RTView EM suite. Architecturally, any number of dataservers may be distributed in local or remote datacenters to collect host, network, or application metrics. A central set of servers handle alerting, configuration management, and display, but these servers "refer" to the dataservers as necessary to reference data rather than aggregating the data locally. Hence, the architecture scales as needed to support very large populations of

monitored resources. But, for purposes of this article, we'll look at deploying only a single dataserver and exposing the data it collects to Splunk.

Our test dataserver is configured by specifying the appropriate "solution packages" and corresponding properties. A solution package is a bundled collection of cache definitions, cache source files (templates for data acquisition), alert definitions, and user interface displays. As an example, the following properties configure the dataserver to collect host data via hawk.

*rtvapm_package=hawkmon*
*rtvapm_package=hostbase*
*collector.sl.rtview.hawk.hawkconsole sl_qa ems sl_qa tcp://10.16.200.118:7222 admin -*
*collector.sl.rtview.hawk.agentGroup WIN_AGENTS SLHOST16(sl_qa) SLHOST15(sl_qa)*

The hostbase package contains the basic caches, alerts, and displays needed for generic host metrics, regardless of the protocol used to collect these metrics. The hawkmon add-on adds support for collection of host metrics via hawk. Hawk messages can be carried by either of two different transports. The "ems" transport is TIBCO's Enterprise Message Server. (Alternatively, the Rendezvous messaging middleware could be specified by using "rv".) The example hawkconsole property defines a connection to an EMS server topic, and the agentGroup specifies that we collect for two specific hosts whose hawk microagents are also configured to use ems.

When a dataserver is started with this configuration, we can access the collected host metrics via the following REST query to the dataserver.

*http://<hostname>:8068/hostbase_rtvquery/cache/HostStats/current*

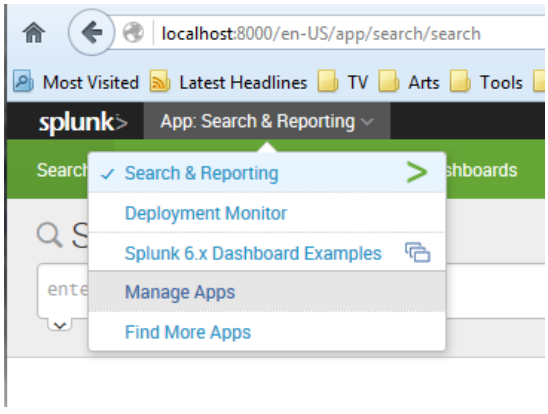
By default, the response format will be XML, as shown below, where a row of metadata describes the data columns followed by a row of data for each host (some data omitted for brevity). For use with splunk, we'll tack "?fmt=json" to this query to get our results in an easier-to-parse json format:

```
<?xml version="1.0"?>
<dataset>
 <metadata>
  <column name="time_stamp" type="date"/>
  <column name="domain" type="string"/>
  <column name="hostname" type="string"/>
  <column name="OS_Name" type="string"/>
  <column name="OS_Version" type="string"/>
  <column name="Uptime" type="long"/>
   <column name="numCPUs" type="int"/>
  <column name="MemTotal" type="double"/>
  <column name="MemUsed" type="double"/>
  <column name="MemFree" type="double"/>
  <column name="MemUsedPerCent" type="double"/>
  <column name="userPerCentCpu" type="double"/>
  <column name="systemPerCentCpu" type="double"/>
  <column name="idlePerCentCpu" type="double"/>
  <column name="usedPerCentCpu" type="double"/>
 </metadata>
 <data>
```

```
<row>
 <time_stamp>1450987153921</time_stamp>
 <domain>myHawkDomain</domain>
 <hostname>SLHOST15(sl_qa)</hostname>
 <OS_Name>Win32</OS_Name>
 <OS_Version>6.1</OS_Version>
 <Uptime>7525657</Uptime>
 <numCPUs>2</numCPUs>
 <MemTotal>8192.0</MemTotal>
 <MemUsed>3733.26171875</MemUsed>
 <MemFree>4458.73828125</MemFree>
 <MemUsedPerCent>45.57</MemUsedPerCent>
 <userPerCentCpu>5.069760901455278</userPerCentCpu>
 <systemPerCentCpu>-1.0</systemPerCentCpu>
 <idlePerCentCpu>94.93023909854472</idlePerCentCpu>
 <usedPerCentCpu>5.0697609014552825</usedPerCentCpu>
</row>
<row>
 <time_stamp>1450987137189</time_stamp>
 <domain>myHawkDomain</domain>
 <hostname>SLHOST16(sl_qa)</hostname>
 <OS_Name>Win32</OS_Name>
 <OS_Version>6.1</OS_Version>
 <Uptime>6914605</Uptime>
 <numCPUs>4</numCPUs>
 <MemTotal>8192.0</MemTotal>
 <MemUsed>7304.38671875</MemUsed>
 <MemFree>887.61328125</MemFree>
 <MemUsedPerCent>89.16</MemUsedPerCent>
 <userPerCentCpu>9.086940514895092</userPerCentCpu>
 <systemPerCentCpu>-1.0</systemPerCentCpu>
 <idlePerCentCpu>90.91305948510491</idlePerCentCpu>
 <usedPerCentCpu>9.08694051489509</usedPerCentCpu>
</row>
</data>
</dataset>
```


## Configuring Splunk to Get RTView Data

In order to query the RTView dataserver from splunk, download the REST add-on from splunkbase which can be found at https://splunkbase.splunk.com/app/1546/. Install the add-on by selecting "Manage Apps" from the "App" menu in the upper left corner of your browser window. In the Apps window, click the "Install app from file" button, use the "Browse..." button to set the path to the downloaded file, and click "Upload".



After the REST interface is installed, you'll need to update a python script to handle the specific json format returned by queries to the RTView dataserver. Edit *"<SplunkHome>/etc/apps/rest_ta/bin/responsehandlers.py*" and add the following code:

```
class slRtvEventHandler:

   def __init__(self,**args):
      pass

   def __call__(self, response_object,raw_response_output,response_type,req_args,endpoint):
      if response_type == "json":
         parsedJson = json.loads(raw_response_output)
         for rtvDataRow in parsedJson["data"]:
            rtvDataRow["Timestamp"] =
datetime.datetime.fromtimestamp(rtvDataRow["time_stamp"]/1000).strftime('%d-%b-%Y %H:%M:%S')
            print_xml_stream(json.dumps(rtvDataRow))
      else:
         print_xml_stream(raw_response_output)
```

We'll use this class when configuring connections to REST-ful endpoints in the next sections. The slRtvEventHandler converts the json returned by queries to the RTView dataserver into python objects, then extracts the "data" section containing rows of tabular data and writes each row to Splunk as separate events. (If this is not done, Splunk treats the entire query result as an "event" object and it will be difficult to pull it apart to display in Splunk views.)

Before pushing each event to Splunk, you can optionally enrich the data in various ways. Here, we reformat the integer timestamp into a date/time string. Note that Splunk can also perform simple transformations like this example, but it may sometimes be advantageous to persist these changes in the stored data in order to optimize searches.

## Ingest Host Metrics Collected via Hawk

Given a working dataserver and Splunk with the REST add-on, we can now configure a connection to pull data into Splunk. Click the "Settings" menu item in the splunk browser interface and select "Data inputs", then click on the "add new" action for the REST type. Configure the connection fields with the following values:

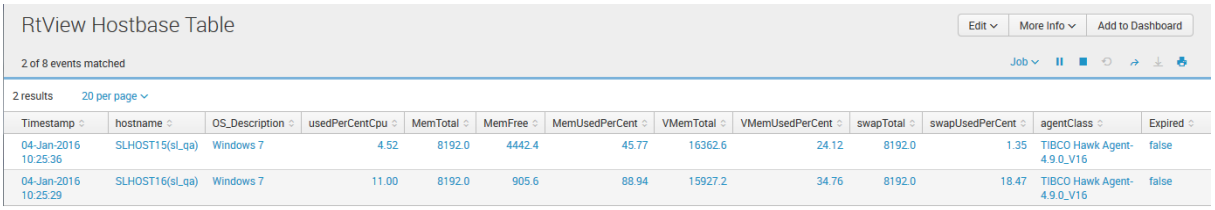
*Endpoint URL: http://<dataserver url:port>/hostbase_rtvquery/cache/HostStats/current*
*HTTP Method: GET*
*URL Arguments: fmt=json*
*Response Type: json*
*Response Handler: slRtvEventHandler*
*Request Timeout: 10*
*Backoff Time: 60*
*Polling Interval: 30*

*Set sourcetype: Manual*
*Source type: rtv_hostbase*

Save these settings, and then go to the "App: Search & Reporting" screen and click the "Data Summary" button. On the "Data Summary" pop-up, the "Sourcetypes(*)" tab should now show that data for a new source type "rtv_hostbase" is available. Click on this source type and examine the events. The search string for this display is simply "sourcetype=rtv_hostbase". We'll want to create tabular reports to visualize this data, so set the search time to a "1-minute window" in "Real time" and paste the following into the search window:

*sourcetype=hostbase | dedup hostname sortby +_time | eval usedPerCentCpu=round(usedPerCentCpu,2),swapUsedPerCent=round(swapUsedPerCent,2),MemUsedPerCent=round(MemUsedPerCent,2),MemFree=round(MemFree,1),VMemTotal=round(VMemTotal,1) | sort hostname | table Timestamp,hostname,OS_Description,usedPerCentCpu,MemTotal,MemFree,MemUsedPerCent,VMemTotal,VMemUsedPerCent,swapTotal,swapUsedPerCent,agentClass*

The dedup clause de-duplicates the data (as indexed by hostname) so that no matter how many samples RTView may return each minute, Splunk will display the latest sample for each host. Now save this search as a report and open the report to see the following:


RtView Hostbase Table

Edit | More Info | Add to Dashboard

2 of 8 events matched

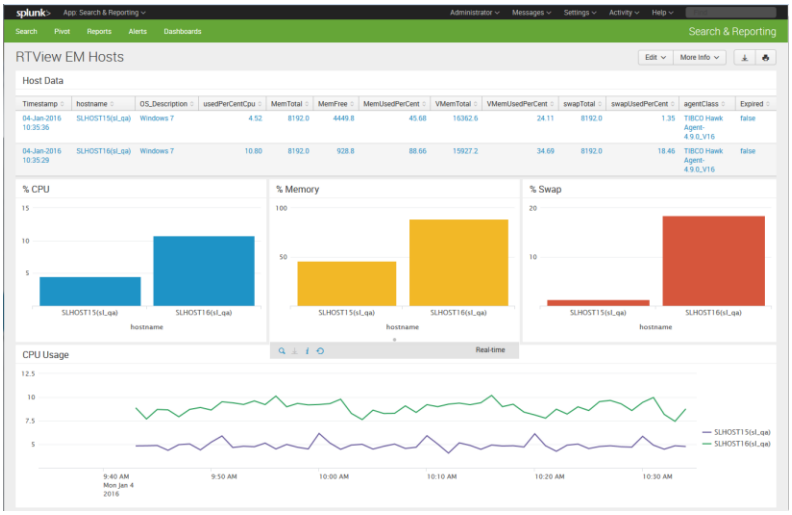
Job

2 results 20 per page

| Timestamp | hostname | OS_Description | usedPerCentCpu | MemTotal | MemFree | MemUsedPerCent | VMemTotal | VMemUsedPerCent | swapTotal | swapUsedPerCent | agentClass | Expired |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 04-Jan-2016 10:25:36 | SLHOST15(sl_qa) | Windows 7 | 4.52 | 8192.0 | 4442.4 | 45.77 | 16362.6 | 24.12 | 8192.0 | 1.35 | TIBCO Hawk Agent-4.9.0_V16 | false |
| 04-Jan-2016 10:25:29 | SLHOST16(sl_qa) | Windows 7 | 11.00 | 8192.0 | 905.6 | 88.94 | 15927.2 | 34.76 | 8192.0 | 18.47 | TIBCO Hawk Agent-4.9.0_V16 | false |

The Note the “Expired” column added to the tabular host data by RTView. This boolean indicates that RTView was unable to collect new data for the monitored resource during the last collection period. Normally, you would add Splunk alerts to catch the cases where a key metric (e.g., “usedPerCentCpu”) was above a threshold. Adding an alert on the “Expired” status will let you know when the resource is not available. If the resource does not recover within a configurable “rowExpirationTimeForDelete”, the expired data will be dropped from the RTview cache, and will then disappear from the Splunk displays.

This report can be used as-is, or included in dashboards like the following:




The following search was used to create the trend chart:

*sourcetype=hostbase | timechart span=1m avg(usedPerCentCpu) by hostname*

The timechart command averages the data collected for each host to one minute resolution. This resampling makes it easy to compare CPU utilization across multiple hosts in a tabular view, as shown below.

| _time | SLHOST15(sl_qa) | SLHOST16(sl_qa) |
|---|---|---|
| 2016-01-04 09:49:00 | 4.406645 | 8.911433 |
| 2016-01-04 09:50:00 | 5.225768 | 8.638420 |
| 2016-01-04 09:51:00 | 5.888798 | 9.515960 |
| 2016-01-04 09:52:00 | 4.666743 | 9.402363 |
| 2016-01-04 09:53:00 | 4.795472 | 9.229947 |
| 2016-01-04 09:54:00 | 4.744746 | 9.606964 |
| 2016-01-04 09:55:00 | 5.134764 | 9.216946 |
| 2016-01-04 09:56:00 | 4.510736 | 10.120487 |
| 2016-01-04 09:57:00 | 4.991758 | 8.982936 |
| 2016-01-04 09:58:00 | 4.692744 | 9.340452 |
| 2016-01-04 09:59:00 | 4.509506 | 9.181441 |

The preceding dashboard displayed data for two test hosts. In a more realistic setting with perhaps hundreds of hosts, you'll want to use Splunk's filtering or aggregation commands to limit the data displayed. For example, you could sort on "usedPerCentCpu", then use the "head" command to limit the panels to the top ten resources with the highest CPU utilization. Or, use the "top" command on "agentClass" to see the distribution of Hawk versions deployed in your enterprise.

## Add Metrics for TIBCO EMS and Business Works

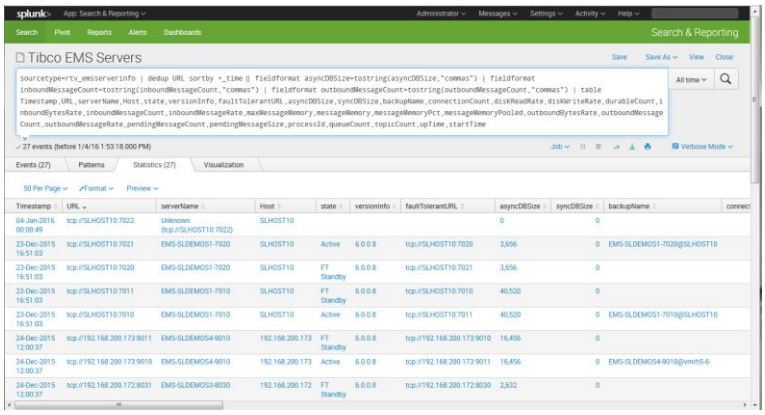
As additional examples of ingesting TIBCO data into Splunk, we'll look at data from BusinessWorks and the EMS messaging bus in the following sections. It's simple to add the collection of EMS and BW data, by specifying additional RTView packages to the dataserver configuration and restarting the dataserver:

*rtvapm_package=emsmon*
*rtvapm_package=bwmon*

Once the EMS and BW metrics are being collected by the dataserver, simply configure another REST endpoint (see above, "Ingest Host Metrics Collected via Hawk") for EMS data; use an appropriate URL and source type.

*Endpoint URL: http://<dataserver IP:port>/emsmon_rtvquery/cache/EmsServerInfo/current*
*Source type: rtv_emsserverinfo*

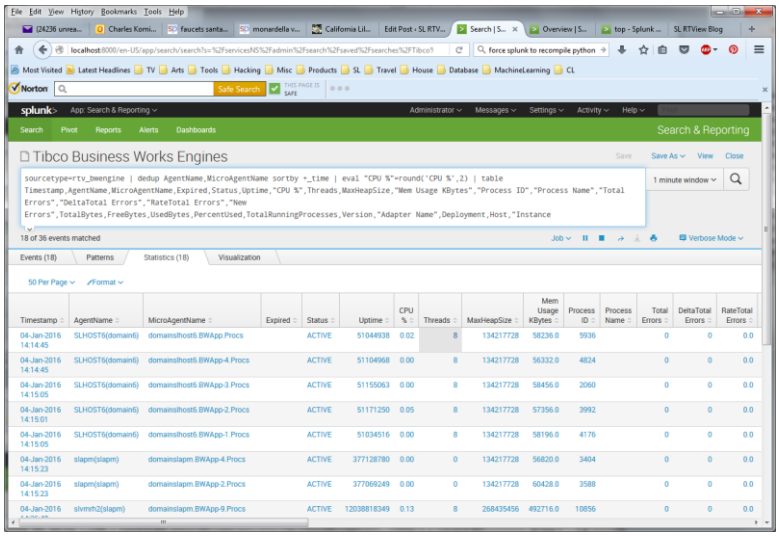
Save the endpoint, then switch to the Apps:Search page, click "Data Summary" and verify that data is being collected for the new source type. Click on the new source type to see the new data and edit the search string as shown in the following image.




For Business Works, configure a REST endpoint for BW data as in the previous section.

*Endpoint URL: http://<dataserver IP:port>/bwmon_rtvquery/cache/BwEngines/current*
*Source type: rtv_bwengine*

Save the endpoint, then switch to the Apps:Search page, click "Data Summary" and verify that data is being collected for the new source type. Click on the new source type to see the new data and edit the search string as shown in the following image.



In the sections above, we have shown how to display metrics from hosts, EMS servers, and BW engines. It is important to note that the RTView dataserver also provides extensive alert configuration and management functionality. The state and detail of all alerts generated in an RTView dataserver may be imported into Splunk, using the same mechanism described above.

## Conclusion

As demonstrated, it is quite easy to pull data from RTView into Splunk, where you can easily manipulate it for display in Splunk dashboards. (Note that the focus in this article is on TIBCO data, but this same technique applies for all data monitored by RTView, be it Oracle, IBM, Open Source, or other.) The benefits of this approach include robust and easily configured RTView collection that exists today without the need to create and deploy your own specialized collectors, the ability to create views of this data in Splunk that are easily maintained to meet your needs over time.

The solution we've discussed here uses a single component of the RTView EM architecture - the RTView dataserver. EM is a mature platform that easily scales to meet the needs of the smallest to the largest of the fortune 500 companies. Hence, as new monitoring requirements crystallize, you can be confident that a growth path exists to further complement and expand your monitoring and analysis capabilities.