Reprinted from the

Proceedings of the Linux Symposium

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium,*
Thin Lines Mountaineering

C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium,*
Thin Lines Mountaineering

Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*

Systems Monitoring Shootout

Finding your way in the Maze of Monitoring tools

Kris Buytaert
Inuits
Kris.Buytaert@inuits.be

Tom De Cooman
Inuits
Tom.DeCooman@inuits.be

Frederic Descamps
Inuits
Frederic.Descamps@inuits.be

Bart Verwilst
Inuits
Bart.Verwilst@inuits.be

Abstract

The open source market is getting overcrowded with different Network Monitoring solutions, and not without reason: monitoring your infrastructure is becoming more important each day. You have to know what's going on for your boss, your customers, and for yourself.

Nagios started the evolution, but today OpenNMS, Zabix, Zenoss, GroundWorks, Hyperic, and different others are showing up in the market.

Do you want light-weight, or feature-full? How far do you want to go with your monitoring, just on an OS level, or do you want to dig into your applications, do you want to know how many query per seconds your MySQL database is serving, or do you want to know about the internal state of your JBoss, or be alerted if the OOM killer will start working soon?

This paper will provide guidance on the different alternatives, based on our experiences in the field. We will be looking both at alerting and trending and how easy or difficult it is to deploy such an environment.

1 Some Definitions

The monitoring business has its own set of terms, which we will gladly explain.

How do you want your monitoring system being "served?" A light version? Full-blown and feature-full? The most important question is: how far do you want to go with your monitoring? Just on the OS level, or do you want to dig into your applications, do you want to know how many query per seconds your MySQL database is serving, do you want to know about the internal state of your JBoss, or be triggered if the OOM killer will start working soon? . . . As you see, there are several ways of monitoring depending on the level of detail.

In our monitoring tool, we add hosts. This host can be any device we would like to monitor. Next we need to define what parameter on the host we would like to check, how we are going to get the data, and at which point we'd consider the values not within normal limits anymore. The result is called a *check*. There are several ways to 'get' the required data. Most monitoring tools can use SNMP as a way to gather the required data. Either the tool itself performs an SNMP-get, or it receives data via an SNMP-trap. A lot of tools can also work with external scripts that can be 'plugged in.' Most of the time you can use a script in whichever language you like (bash, perl, C, java, etc.), as long as it sticks to certain rules set by the monitoring tool. These rules make sure the tool can understand the data returned. Checks that are performed by the monitoring tool itself are called active checks. The monitoring tool 'polls' another device to get some data out of it. Checks performed and submitted by external applications are called passive checks. Passive checks are useful for monitoring hosts and services on that host that are, e.g., not directly reachable for the monitoring server or where no direct check is possible or available. An SNMP Trap can be implemented in a monitoring solution via a passive check.

Alerting is the way to send a warning signal. Usually this is an automatic signal warning of a danger. In monitoring services, alerts can be sent via different methods when available: an email is sent to one or many users with the warning message and the result of the check having a problem. A message is sent on the mobile of one or many users with the description of the problem. The monitoring interface highlights the problem in a user-friendly way, usually with colors depending on the severity level of the problem. We obviously also want Instant Messaging to be used. All the alerts are related to services that the monitoring system must check. The level and the signal of the alerts can be setup independently by services. All good monitoring systems are able to send different notifications depending on the time frame. So for example during the night only the on-call person will be notified via SMS of the problem.

The data the monitoring tool is gathering is also stored for historical reference. This can be used to generate reports, so one can view the status detail for a specific host or service over a certain period. General availability can be checked and reviewed. Custom availability reports can be generated depending on the monitoring tools capabilities. E.g., the uptime of a certain host or service. How many times it went down, for how long, also providing a time related percentage of downtime or uptime. We then can report a summary of all events and status linked to the services we are monitoring.

Next to reports themselves, some tools also provide a way to chart the gathered data. Although monitoring and trending are actually two different businesses, some tools are combining them into a single interface. The capabilities of the monitoring tool might be enough to suite your historical data trending needs, still there are other specific tools available which are pretty good in performing only the trending task. These standalone tools mostly provide more options, or might provide an easier way to accomplish certain tasks than the monitoring tools.

Trending and reporting can definitely help in perfecting/ fine-tuning the monitoring task. Using trends we can adjust already implemented monitoring rules to situations noticed in the trends/reports. Checks can be adjusted to certain situations. We can disable a check during a certain time period, or we could loosen the limits a check is set to trigger on during this time interval.... In other words, it can help to make a check more accurate. So in the end to establish a more accurate monitoring system, which also gives us the ability to notice upcoming issues and react on them, making the business of system administration more proactive.

Trending is reading of the variation in the measurement of data over several intervals. In your environment you are interested in how many queries your databases parses per second, and how this evolves over time. You are also interested in figuring out how many traffic passes over a certain switch port and if you have enough bandwidth capacity left. Trending therefore is not so much related to the uptime of certain services but to the behavior of the service itself.

2 What are we looking for?

We started out asking ourselves what we want in a network and infrastructure monitoring. Some interesting topics came up. We look at an NMS from different viewpoints, from a user viewpoint, from a sysadmin viewpoint, and also from a developer viewpoint.

Note that we are experienced systems people, but often new to the tools, and that's probably the way you want it. We want to know how fast you can get up to speed with a tool while not having to spent hours and days tuning the platform, or even read manuals.

Monitoring tools are typically set up by System Administrators or Infrastructure Architects and then handed over to either Junior staff or Operations people. That means that once it is up and running, everybody should be able to use it. A good an clean GUI is a requirement.

However, as you are going to add over 100 servers at once to a monitoring application, you do not want to use a web-interface for that the process. You want to be able to write a simple for-loop for a variety of servers to create config-files, database-scripts, adjust them, and add them. In a later stage, you want some administrators to add hosts to the system through some GUI, but with a template-based system so they can reuse what you prepared.

When introducing a monitoring tool in a large environment, that also means that you are rather reluctant to install a daemon on all these hosts—unless you can fully automate that installation.

SNMP is a great tool to manage/monitor just about anything in your network. You can call this an advantage because SNMP is a package available in every OS, does not have many dependencies prior to installing, it is easy to configure for simple setups, and most of all: quick result! (The ability to add SNMP-scripts of course is a big plus.) However, unlike the name tells us, SNMP is everything but simple and requires in-depth thinking of how to provision our SNMP configuration at all the hosts, so configuring SNMP might be as difficult as adding a client specific to your monitoring, too.

We want to be able to automate as much as possible, so we prefer a config-file-based approach. It doesn't matter whether it has its own syntax, as long as it is still human-readable. A web interface of course is also a must; however, it should be supportive, not enforcing. You should be able to create a configuration that the webinterface doesn't overrule. We have to realize that performing an action on 3 hosts via GUI is feasible, but on 300, it isn't. So we would rather settle for less GUI features that we never use.

We all have to please managers and customers, so we want a tool that is capable of creating very clear and complete reports and graphs. Ideally graphs that can be mapped against each other to pinpoint concurrent events, etc.

It goes without saying that a monitoring tool should be stable. As the tool is monitoring events in our infrastructure, it needs to be up and running all the time in order to be able to escalate issues. It also needs to be able to scale; infrastructures grow at a terrific speed these days, so the monitoring tool you are starting out with today must be capable of either delegating groups of machines or manage them itself.

When looking at the resources used in your infrastructure, you are interested in memory usage, IO performance, etc. So you want your monitoring tool to be almost invisible. We mentioned before that ideally no plugins have to be installed, but if they have to be installed, we want them to be lightweight. A JVM requiring a big chunk of your precious RAM just to see if a process is still running might be overkill. A monitoring tool should blend in with the infrastructure, not add more requirements to it.

A good Monitoring Platform has capabilities of both telling us how a machine performed in the past, how much time a certain service was available and how long it wasn't. We want to see trends in usage—e.g., whether disk and memory usage have been changing over time.

Ideally you also get a separate status page, no technical data, but a clean status page along with persons to contact in case of problems.

A good monitoring tool has plenty of alternative notification methods. SMS and mail are primary methods, but Instant Messaging solutions are more and more a standard requirement, too. And you want to be able to configure these notifications, select different methods based on the time, how critical an incident is, and so on. Also you don't want to be spammed by the tool, you want relevant escalation when appropriate, but you don't want a message flooding your inbox that you will filter away anyhow.

A great framework is a good start, but without an active community it is probably as bad as a commercial tool that also requires you to buy a lot of extra features. You need a plugin model that allows you to add checks and functionalities in different languages: PHP, bash, perl, C, etc. So having a clear and powerful API is a must. That way your community will be able to write new checks for services that you either don't know yet or haven't heard about yet.

3 Nagios

Nagios is considered by many to be the godfather of host and service monitoring on Linux. It is without doubt the best known monitoring tool out there, and is available by default in all major Linux distributions. Nagios version 3.0 was released in March 2008, but will only be picked up by most distributions in the next 6-12 months. In the meantime, version 2.11 is the most common in today's distributions. Nagios was created by Ethan Galstad and is licensed under the GPL v2. Many others contributed to Nagios since its conception through plugins, add-ons, bugfixes, suggestions, testing, bug reporting, ideas, and feature requests. Where as Nagios was incepted in 1999, Ethan only recently started Nagios Enterprises to commercially support the Open Source project.

One of the greatest features of Nagios is the great scriptability of its configuration. Nagios is fully configured with text files, which can easily be generated on the fly to perfectly fit into your network. Everything from hosts, services, contacts to groups and alert escalation plans can be configured in this way. Configuring Nagios comes with a pretty steep learning curve, though, for first-time users. At frequent intervals the Nagios monitoring deamon will run checks on hosts and services you specify via a mechanism of external plugins that then return status information.

Problems can be reported to the administrators by means of SMS, email, instant messages, or a variety of other ways. Escalation is also supported, so that when the first contact doesn't acknowledge the problem within a predefined time frame, another person or group can be notified in order to get the problem resolved. Nagios works by running predefined checks on a configurable interval through a plethora of external plugins that return status information to the Nagios service itself. Since Nagios is solely an alerting tool, and lacks a fancy GUI with graphs, trending, and other monitoring features, it is usually accompanied by a separate tool that handles those features, such as Cacti. The Nagios/Cacti combo is the most popular one. Several independent projects also try to improve and extend the pretty rudimentary and boring looks of the Nagios web interface by implementing a new web-based frontend on top of the Nagios data and configuration, thus merging the reporting and monitoring/trending features into one single rich web interface.

Nagios by itself is pretty lightweight, with a C-based backend and a cgi-based frontend (web GUI), and requires hardly any dependencies to get it in a working state, and can be used on a very modest (virtual) machine.

4 GroundWorks Open Source

A couple of months ago we decided to testrun the GroundWorks Open Source 1.6 RPM. We didn't get far as we immediately had to with our efforts, because GroundWorks decided that their RPM should modify our httpd initscripts and point to its own httpd installation in `/usr/local/groundworks`, hence breaking other tools we had on the platform.

So this was our second encounter with the GroundWorks platform. GroundWorks expects you to untar a tarball, then run its installation script which comes to ask you if it can install a bunch of RPMs. Or you can just try to install these RPMs manually, which fails as the RPM expects you to have set your JAVA home. I've never seen an RPM that failed in the preinstall because of lacking environment settings, and this obviously shouldn't happen as people expect to be able to install RPMs from a repository during boot time, obviously lacking those environment variables.

So we opted for the installer. It requires SELinux to be disabled, complains about lack of memory (it expects one GB and I only have 256MB). It also tells me I need 40GB of disk space rather than telling me how much disk space I really need, it explains on failing me with 200MB short on my rootfs. The installer failed multiple times on me, first telling me I need a different java-1.5.06 java version than the java-1.5.06 version I had installed.

The installer knows that mysql isn't started, but doesn't try starting it itself—it tells me to start it before continuing, which I do in another console, and then it aborts because `mysql` isn't started. Error handling seems to be a feature for the next installer version as the installer today fails on you without errors. Although the underlying RPM installation process gave clean and clear errors, the GroundWorks Monitor 5.2 failed to capture them, not even in its logfile.

So we were pretty disappointed with the install process. Sadly GroundWorks still didn't fix the Apache problem mentioned before, either.

As we are setting up different test platforms in a remote isolated lab, we often tunnel port 80 of the monitoring tool to another port on our own machine. GroundWorks was the first tool that complained about having tunneled its port 80 to my local port 8888, and insisted on refreshing itself to `localhost:80`—hence a totally different environment, although a problem I could solve. In this case, you are often forced to tunnel your monitoring tool and map it on different ports, and a web application should be unaware of that.

The next problem was the documentation. The installer process points you to a Bookshelf in the application, a Bookshelf you can't access unless you are logged on to the system, which doesn't work, as you never got any information about a default username or password combination.

After 3 days, the forum came back with the obvious `admin:admin` answer that didn't work for me at first. It worked now.

The auto discovery seemed to work only partly, it found some hosts, but it seems it doesn't update the Nagios config files, so I couldn't figure out how to get them in the Nagios overview.

For a Nagios-based tool, you would expect a good and easy installation procedure and great documentation. You'd expect to build around a great tool and make it better.

5 Zenoss

Bill Karpovich, Mark Hinkle, and Erik Dahl are starting to become regular names in the Open Management industry; they bring us Zenoss. Zenoss gives you a single, integrated solution for monitoring your entire infrastructure—network, servers, and applications. They claim to support inventory, configuration, availability, performance, and events of your services. Zenoss comes in a Free community edition and a different commercially supported version. Its Free version includes Availability Monitoring, Performance Monitoring, Event Management, and core reporting functionality.

Zenoss likes to compare itself to both proprietary and open source tools, claiming that unlike the others it is both easy to install and configure, and it's affordable. It is more open, brings no vendor lock-in and has better community collaboration.

The Zenoss architecture breaks down into three parts. A user part with the WebConsole/Portal, a Data Layer (where all the data lives), and the Process Layer that collects the data via standard protocols. Zenoss is one integrated package, not some different packages glued together into a bigger whole. You can configure templates and map instances to those templates.

In the data layer, Zenoss uses three places to store its data, its CMDB (Configuration Management DataBase) is an object model stored in Zope (ZODB). It is obvious that for historical data they use RRDTool, and the events are being stored in a MySQL database. A nice mix-and-match to store everything they need.

The actual work is done by a series of daemons and control services that provide node discovery, configuration modeling, availability monitoring, performance monitoring, event collection, and automated responses. Each of these services can run as a single local instance or be distributed, hence providing a scalable solution. Zendisc stands for the device discovery; Zenmodeller is used to get configuration details and map resources to the resource model; whereas ZenWinmodeler uses WMI to discover windows-based services. Different plugins such as ZenPin, ZenStatus, ZenPerfSNMP, and ZenProcess are used to check services. ZenSyslog, ZenEventlog, and ZenTrap are used to collect events, and as the names already show, they respectively collect syslog events, WMI events, and SNMP traps. And let's not forget ZenActions, which is responsible for notifications and automated action scripts.

Zenoss is heavily based on Zope and Python for its web framework, which classifies is as a rather lightweight platform.

Zenoss has a prebuild VMware Applicance; RHEL, Fedora, and CentOS packages; and source tarballs available for SuSE, Debian/Ubuntu, Gentoo, OSX, FreeBSD, and Solaris.

We downloaded the zenoss-2.1.3-0.el5 RPM from `SF.net`.

Upon initial startup, it populates the database and starts building parts of itself. It almost performs a clean install in `/opt`, although different files in its directory don't belong to the package ;(

From there it is on to the webbrowser to use the web-GUI to configure everything. Its autodiscovery functionality uses SNMP. So any host with SNMP will be autodetected. Others will remain unknown. Autodiscovery seems to work, but only partially; it detects all of the devices that use SNMP, but fails to recognize details as configured in our `snmpd` config (process lists, diskspace usage, etc.). We flooded our root partition on one of the machines on purpose; `snmpd` was configured to start alerting at 10% left. Zenoss failed to recognize that. Shouldn't we expect this to be working? In the Main Views, Zenoss has a nice Ajax host relation diagram, it realizes hosts are connected to different networks and maps them out—really nice feature.

6 OpenNMS

'OpenNMS: professional software, amateur marketing.' That's their slogan, and it is a good reflection of what they stand for. Their site is mostly technical documentation, no fluff on how good they are how many features they have, just plain and correct facts.

OpenNMS is one of the older Open NMS platforms around. Back when they started out, Nagios was the lean and mean monitoring tool and OpenNMS was the Enterprise Grade platform that would take on HP OpenView, IBM Tivoli, and other proprietary monitoring tools.

In 2001 the choice was easy, you either had Nagios or OpenNMS, if you had SNMP and weren't afraid of deploying a J2EE appserver, you went for OpenNMS; otherwise, Nagios. Today things have changed :)

An OpenNMS instance can watch a large number of nodes from a single server, with a minimal amount of configuration and reconfiguration work needed. In an OpenNMS platform, you can define flexible rules to specify how often and when certain devices are polled, to whom different alerts are sent, and so on.

The basic element that OpenNMS monitors—an interface—is uniquely identified by an IP address. Services are mapped to those interfaces, and a number of interfaces on the same devices can be mapped together to a *node*, in OpenNMS terminology. OpenNMS was one of the first tools around to support autodiscovery. OpenNMS first polls a device; it tries to connect to an IP address as defined in the range, then uses SNMP to collect data.

Events are the core of a Network Monitoring system. OpenNMS has a daemon running called `eventd`. It makes the distinction between two types of events: those that are generated by OpenNMS itself, and those that are generated via SNMP traps. Events trigger actions such as logging a message, triggering an automatic action via an external script, or triggering the notification system.

Notifications can be sent to users or groups as defined in OpenNMS, One can configure delays, escalations, and email addresses to send the alerts to.

Our CentOS testbed was fairly pleased with some good installation documentation on how to install OpenNMS using yum. OpenNMS has its own repository, `yum.opennms.org`.

7 Zabbix

Zabbix is a network management platform created by Alexei Vladishev. It is designed to monitor and track the status of various network services, servers, and other network hardware. Zabbix has a mission: "To create a superior monitoring solution available and affordable for all." Zabbix was released for the first time in 2001, and the Zabbix company was founded in 2005 in Riva, Latvia.

Zabbix has three main parts: the daemon, the agent, and the web interface. The daemon collects all data from the agents and populates the database. The independent web interface then parses that data from the database and provides the users with an overview of what's happening. It uses MySQL, PostgreSQL, SQLite, or Oracle to store data. Its web-based frontend is written in PHP. Zabbix offers several monitoring options. Simple checks can verify the availability and responsiveness of standard services such as SMTP or HTTP without installing any software on the monitored host. A Zabbix agent can also be installed on UNIX and Windows hosts to monitor statistics such as CPU load, network utilization, disk space, etc. As an alternative to installing an agent on hosts, Zabbix also includes support for monitoring via SNMP.

As Zabbix uses a database to store its data, this also involves some extra configuration steps. Installing via the system's package manager will be straightforward, as dependencies will be resolved, too. The Zabbix package contains the MySQL (or other) tables to be imported in the database. As you install Zabbix on a server, you'll probably want the Server and the Webserver installed. Note that you don't really need to place them on the same machine, but of course, you can. The default user is `admin`, with no password assigned.

On the Zabbix-server side, all configuration is done using the web interface. For initial setup, connect as user `admin`. The most important part is *Configration* in the top-most menu.

In the configuration part, a lot of things can be configured or customized—from adding hosts to customizing the look and feel of the web interface itself.

We'll start with some general Zabbix-info. Simple example: we have a host running an FTP server; when the FTP service goes down, we'd like to be alerted. As with probably all kinds of monitoring applications, you first need to add a host. (More on templates later, let's just assume we defined a host with a name and an IP address). When this is done, an *item* for this host can be created. An item has all the data to define how a check is to be performed on the host. (Important ones: a name for the item, a check type: info about what data we want and how to get it, a check interval.) The result is that a *key* is stored for a certain host (e.g., FTP-key being 0 or 1, off or on). A simple check is used by Zabbix to monitor agentless hosts. They include ICMP, HTTP, and others. The next thing is to make sure the system notices when the FTP service goes down and does send us a notification. In Zabbix terms, this means we need to define a *trigger* and an *action*. The trigger is quite simple in this case: when the FTP is down, trigger to

YES. The trigger uses an *expression*, in which a key is evaluated. Now that we have a key and we have configured a trigger on it, we want the system to send us an email when the trigger's state changes. In Zabbix, this is done by adding an *Action*. An action has an *Event Source* (in this case, it is a trigger), a *condition*, and an *operation*. In the condition, we make the link with the correct trigger; in the operation, we simply say to send an email to a certain user. A Zabbix-template is made of several items and triggers. All templates are actually just some kind of special host-definitions, which are put into the templates-group. It does not contain any actions. So when linking a host with a template, a lot of items and triggers will be added for that host, but you will still need to define actions yourself. Most of the default templates in Zabbix are using items based on keys grabbed from the Zabbix agent. So if you want to get up and running as soon as possible, just install the Zabbix-agent on the host to be monitored. One way to add hosts is by using *Discovery*. Zabbix can *scan* a network, or a part of it, and add hosts it finds. The scan can be defined. E.g., Zabbix can check a network range for hosts that are running the Zabbix agent, or you can set other requirements like the presence of a ssh-server, etc. Based on the output of this scan, Zabbix can use an *Action* to do something with the host it has found. E.g., a discovered Linux host that is running the Zabbix-agent can directly be put in the Linux-group and the Linux Template can be assigned right away.

As mentioned earlier, Zabbix will use an action to act on a trigger. Zabbix can use different methods to send out alerts, such as email, Jabber messages, or text messages to a mobile phone. One can also define the days and hours on which a person can receive alerts.

Zabbix uses items that have a key containing data; this data can easily used to produce graphs. And Zabbix seems to be built keeping this in mind. By default, the data of items can be seen in a graph. Just go to the top-most *Monitoring* tab and select *Latest data*. Graphs viewed here are called *Simple Graphs*. Data is stored during one year by default, but this can be changed when creating an item. Zabbix can also monitor Applications. For example, an application MySQL Server may contain all items which are related to the MySQL server: availability of MySQL, disk space, processor load, transactions per second, number of slow queries, etc. Apart from the default graphs, you can also create custom graphs. With the default ones, there is always just one item present in the graph. When you create a graph yourself, you can group multiple items in one graph.

Zabbix also provides an *Overview* page. Several of these pages can be created; they are called *Screens* in Zabbix terminology. A screen is a combination of several types of information—e.g., Simple graphs, custom created graphs, host info, trigger info, or event info. One can define what is being displayed in a screen. It could be set up in a way as to create a perfect status-page for the monitoring system.

8 Hyperic-HQ

Hyperic claims to be Open Source Web Infrastructure Monitoring and Management Software; it aims at automating your operations. Hyperic HQ is GPL, but they also offer a Silver Support package that includes low-cost support.

Hyperic has Auto-Discovery, it understands a lot of technologies over 9 different operating systems, and it is within their goals to manage everything centrally and quickly, allowing their customers to focus on serious issues. HQ collects both real-time and historical metrics from production environments including hardware, network, and application layers of your infrastructure with what they claim is no need for invasive instrumentation. Hyperic does performance tracking, alerting upon performance problems or inventory changes and even diagnoses errors to issue corrective actions remotely. They claim to be able to correlate events, config changes, or even security events in your environment.

The list of tools and platforms that Hyperic HQ knows about is growing every day. Not only does Hyperic manage these products, it does so by talking to the native APIs that these products provide. Unlike different other tools we've ran into, Hyperic goes IN the application you are monitoring.

Looking at Hyperic HQ from an architectural point of view, they isolate different layers. They start out with a platform which is a machine / operating system combination or a network or storage devices. Hyperic HQ likes to look at components such as the CPU, the Network interfaces, or the Filesystems. One step further is the server. The server is the actual piece of software installed on the machine—it could be a web server, a database, or a messaging server. Next up is the service. An example might be the vhost that is configured within a web server. The bigger picture for HypericHQ is the Application, which is usually a combination of different components that need to be working, the combination of an Apache virtual host, filesystem, and a MySQL database.

Hyperic is a typical Agent Server setup, The Hyperic agent is a "lightweight Java-based client" that consumes between 10 and 70MB of memory, depending on the number of plugins enabled. Its kernel is capable of processing commands from its agent subsystem like measuring, controlling, autodiscovery, and event tracking, and it will be acting as a listener to delegate requests to the appropriate system. On top of that kernel a plugin layer is available that allows different subsystems to interact with a particular product. The Agent also acts as a local cache for data when it can't reach the HQ server.

For the HQ-server, at least 1GB of RAM is advised—but for deployments with more than 25 agents, however, 4GB is recommended. Hyperic HQ is running its own Jboss Application server which might be an unwanted overhead for some people. Hyperic HQ also ships with its own PostgreSQL database. Hyperic HQ server is in charge of processing all incoming monitoring data, detecting alert conditions and triggering sending out messages, managing the inventory, scheduling auto-discovery scans, and last but not least, processing all the commands initiated by the end user. One of the features not often seen in a monitoring tool is the availability to cluster the framework, meaning that one can spread the load of different aspects over more nodes of the monitoring framework.

Hyperic HQ has an active community and even its own HyperForge where people can find all kind of different plugins.

Back in the early days they only had a couple of tarballs, but those were fine. However, when you want to automate the agent installation, a package such as an RPM would be better. Seems like recently indeed Hyperic figured out that an RPM would be interesting. . .

The RPM does almost everything for you. Manually starting the HQ Server, however, should be done using the Hyperic user. The problem with this RPM, however, is that it unpacks a tarball. From here it is a similar action with your clients. After configuring your clients (you can copy a prepared `agent.properties` file around), you'll see a list of autodetected services in your Dashboard. And Hyperic proposes you to add this to its inventory. Now, Hyperic isn't flawless—it detects a lot of services, only to fail, finding different versions of that same service. For example, it finds the local HQ JBoss that it requires for its own workings, but fails to find the JBoss 4.0.3 installed on another server that it is supposed to monitor. However, modifying the parameters in the `agent.properties` file should solve that.

Most other Monitoring systems just detect your MySQL being up or down. Hyperic tells you how fast your indexes and tables are growing, and how many QPS you have on a specific table. The information is in there; however, we have to admit that sometimes it takes a while to find it. :)

Apart from looking at your database, Hyperic also gives you a GUI to optimize and check your tables. So Hyperic goes beyond monitoring, alerting, and trending into fully managing your infrastructure.

Also, when working with different JBoss versions, Hyperic knows about JMX, giving you more fine-grained information.

The big disadvantage of Hyperic is that it requires an awful lot of resources. Hyperic might call a 70MB-eating JVM lightweight, but we call it fat and overweight (especially when compared to other alternatives). When you are trying to measure performance of an ill-behaving server, adding the Hyperic client to that server will, for sure, have an impact on your system. On the other hand, however, you can really drill into an application with Hyperic, you can look deep inside your databases and application servers. And that with almost no configuration efforts.

9 Conclusion

We covered a lot in just 3.5 weeks; getting a full-blown monitoring tools shootout done is a lot of work. We tried our best to test as much as possible and to get a good idea of what worked where and how things worked out of the box. We spent equal time on most of the solutions.

With such little time we had little data for trending analysis, but we haven't shut down our boxen, they'll be gathering more data, and we'll update our findings as we go.

Nagios is still one of the most-deployed tools around, lots of people prefer it because they are used to it and it does what they expect. But do we want more? Do we want easier installations, trending, and so on?

Looking at the installation effort, GroundWorks is obviously the poorest performer in the class. Other tools have prepackaged builds in mainstream Linux distributions that simply work. Where as GroundWorks makes the process more difficult and fails to deliver clear documentation.

On the Auto discovery part, both Agent-based tools score fairly good. Compared to Zabbix, Zenoss doesn't even come close in discovering services, or at least isn't that intuitive.

Zabbix also positively surprised us regarding trending and quick graph correlation, combined with a good set of default templates that get you up to speed in no time.

Part of being a good tool is the capability of not having to look at a manual, of not having to redefine a set of terms so your users understand what you mean. GroundWorks and Zenoss fail there.

We've seen GroundWorks trying to add its own features to Nagios. Because they wanted to build on the shoulders of giants, not reinventing the wheel. The question, however, is: did they make it better? We're not convinced.

On the other hand, we see the OpenNMS and Hyperic people with different monitoring approaches (agent-based vs. agent-less). The OpenNMS folks have written a plugin for Hyperic-HQ that interacts with OpenNMS. This way you really get the best of both worlds. You get the good features of OpenNMS network device integration within one view of Hyperic's application overview.

Hyperic gives you a lot more than a regular monitoring tool, as it goes deep into the applications itself. On the negative side, you might call HypericHQ bloated.

Mark Hinkle, however, mentioned an interesting point in his blog:

> "Nagios that has been around longer than any of the monitoring solutions mentioned here they have a large base of plugins and tests used to check status. Hyperic, Groundwork, OpenNMS, and Zenoss all support Nagios plugins as it is the most utilitarian approach to expanding their products rather than create new standards that might prevent users from using previous customizations and gives flexibility to try new solutions. This adherence to standards enforced (or at least motivated) by users rather than vendors is a bit of a novelty."

As mentioned, these are our initial findings. We plan to continue our evaluation and keep you posted on our site.