Abstract: By extending software typically used by network administrators, DBAs can implement a customized, web-based systems management console to monitor DB2 performance and health, track server resources, and even watch over user applications. An established suite of open source monitoring tools provides a framework for publishing a limitless variety of key status indicators to the web.
Key points

• The case for web-based monitoring
  • We’ll see how no other approach to monitoring offers the flexibility required in today’s complex, distributed 24x7 environments.

• Setting up a monitoring framework
  • Although many web-based monitoring packages exist, we’ll focus on Nagios, which has a proven track record in managing heterogeneous services.

• Key DB2 performance indicators
  • Due to time constraints, this will not get much coverage, other than a few tips I have found to be useful. Fortunately, there will be quite a few other presentations covering this particular topic in detail.

• Storing and graphing historical performance
  • RRDTool is an impressive yet lightweight tool for producing the pictures that are worth thousands of words. We’ll touch on its architecture and features.

• Best practices for monitoring and notification
  • Nagios offers a sophisticated architecture for managing alerts that, when properly exploited, can virtually eliminate chatter to your administrators.
Why web-based monitoring?

- Accessibility
- Extensibility
- Scalability
- Security
- Affordability

I have worked with web-based monitoring tools since 2000, and I prefer them over any other type of monitoring product. I will spend the next few slides detailing the benefits I have realized by adopting an extensible, open-source monitoring tool for managing DB2 and other services.
Accessibility (just give me a URL)

- Across firewalls
- Over a VPN connection (or even dialup)
- From older computers and browsers
- From nearly any wireless handset (WAP, WML)
- Without specialized client-side software
- From users outside of the IT department

Not all web tools are alike. I have worked with some monitoring packages that do run in a browser, but only so they can run a huge Java applet that effectively makes them unusable from remote locations, even over a VPN. This slide is referring to web monitoring tools that produce clean, lightweight HTML, degrading gracefully onto handheld devices, without taking forever to load over slow connections.

Although it sounds a bit far-fetched, I have worked with companies where top-level executives were avid consumers of the IT web dashboard. In those situations, simple HTML-based monitoring products win out over software that must be installed on each workstation. There are few downsides to exposing the IT dashboard to employees across the enterprise, and many advantages. For one thing, IT typically gets fewer calls from users asking “is everything OK on the system?”
Extending web-based monitors

- Disparate web consoles can be stitched together
- Scrape the text off the pages for other uses
- Most web-based monitors support custom scripts
- Also monitor the web servers, app, net, hw, etc.
- You may even notice surprising correlations

Web-based monitors and management consoles are so popular that you may already have a variety of them running in your shop. With very little effort, you can build a simple web page that front-ends them to create a kind of portal site to your IT assets. If those web pages require too much authority to view, you can scrape a subset of the information off those pages with Perl or PHP and publish that instead.

One of the big benefits of decent web-based monitoring tools is the ability to consolidate status information from all of your systems so you can start spotting correlations between them. For example, if you notice that a memory leak on a web server typically precedes a lock pileup in DB2, you can respond to that by taking more aggressive action sooner on that web server before it drags the database down with it. These types of relationships are much easier to notice if you have a single web dashboard that monitors all of your services.
But what if we need to add more…

• Adding capacity to a web app server isn’t too scary
• More HTTP users ≠ more DB2 connections
• Individual users are not triggering service checks
• Implement at the department level, if needed

There are a lot of monitoring products out there that do not make use of a centralized monitoring server. In those cases, each user who is monitoring the system opens up his own connections and polls them himself. This poses a few problems when more users want to monitor production, especially if the monitoring package itself charges on a per-seat basis. When all monitoring is performed by a single monitoring server that is front-ended by a web server, there is no risk of overburdening the production servers with monitoring requests. More users just means that the monitoring web server gets more requests, and the production servers being monitored see no additional impact.
Who should see what?

- Websites can be secured with logins and SSL
- Web servers enable precise and granular policies
- Every single access is already being logged
- Good monitoring apps allow user groups and roles
- Monitoring users never connect directly to DB2

You will inevitably find that web-based monitors will contain some information or functionality that is not acceptable to publish to everyone. At the very least, you would not want the full contents of your web monitoring console to be publicly accessible to your competitors over the Internet. No matter what the monitoring tools provide in terms of security, you also have the ability of controlling access to specific areas of the website through the web server config files, such as httpd.conf in Apache. Decent web-based monitoring tools will also offer the ability to define security levels and employ password-based authentications within the monitoring software.
So what will this get you?

Real-world example
One company learned they should reboot their J2EE servers in sequence instead of all at once.

Web-based monitoring revealed how long it took for idle DB2 connections to be released and closed.

By staggering app server restarts, DB2 wasn’t overwhelmed with connection requests.

Insights like this one, along with many others, become quite clear once the relevant systems are monitored and graphed. The app servers in this case were Jakarta Tomcat servers that used JDBC connection pooling. The Tomcat instances were grabbing the maximum number of allowable DB2 connections immediately during startup, and gradually releasing them after they sat idle for a while.
Another real-world revelation

• I.T. started graphing app server load, UOW elapsed time, and DB2 lockwait
• This revealed a correlation between app server load and average DB2 lockwait time
• When the app server was overburdened, its UOWs took longer, so DB2 locks were held longer
• By noticing the slowdowns earlier, they could fix the app server problem before DB2 deadlocks piled up

This type of analysis is difficult to perform if you don’t have an extensible monitoring package. If you’re only monitoring the database server, you run the risk of missing the problem while it is still small enough to remedy without disruption. In this case, they were able to quiesce the overburdened app server and provision more app servers onto their load-balancing switch to handle the spike in the workload.
Enterprise monitoring on the cheap

- GNU/Linux distribution
- + Apache HTTP server
- + Nagios web-based monitoring framework
- + Custom plugins from NagiosExchange.org
- + RRDTool logging and graphing utility
- + Splunk Server log and message file analyzer
- + (insert your favorite scripting language here)

Grand total: USD$0 (CAD$0, EUR€0)

It’s true…all of the above products are available for use in commercial, production environments for no cost whatsoever. However, there are licensing issues that may come into play should you decide to start redistributing any open-source software with some other programs.

I wanted to mention the Splunk Server even though there isn’t enough time to go into it in the slides. The version I’m referring to is limited to processing 500MB of log files each day. Versions with more logging capacity are available for purchase by the vendor.
What's the catch then?

• “[It's] only free if your time is worth nothing”
• Experiencing what many call “dependency hell”
• Complexity grows as you add more packages
• Who do you call for support?
• What if a package stops being free?

Fortunately, open source monitoring has become mainstream, so you definitely won’t be alone.

I’ve found that IT departments have much more freedom with their time than with their software budget. This means they typically have a better chance of implementing and customizing free software than they have of getting budgetary approval to buy a vendor package that may perfectly meet their needs right out of the box. Is this right? Maybe not, but it is all too often the reality. However, IT folks are usually pretty good at looking things up and asking their colleagues for help, so they stand a good chance of successfully implementing even the most onerous open source packages.

Sadly, there have been cases where a developer or vendor has started charging money for an open source package that previously cost nothing. I used to be a big proponent of Big Brother, a popular web-based monitor that held up well over the years, but Quest bought the product and started charging licensing to all but the non-profit users. It’s still a good product, and arguably worth the price, but it’s no longer free.
Introducing **Nagios**

- Created by Ethan Galstad (Minnesota)
  - Initially released as NetSaint in 1999
- Hosted as a SourceForge.net project since 2001
- Free software as per GNU Public License V2
- Monitoring engine runs plugins (service checks)
  - Remote servers can run their own plugins
- CGI programs provide a web-based mgmt console

http://www.nagios.org

Nagios is frequently cited as one of the top tools for systems administration, and is being adopted by large companies with sophisticated IT environments. Because it has such a large user base, there is a wealth of community-developed plugins, as well as several comprehensive articles and tutorials.
Sample Nagios dashboard

Followed by host-level or service-level details

Most Nagios status pages refresh every 90 sec

These summary boxes appear throughout many of the status pages in Nagios and make it easy to determine at a glance if there are any problems in your environment.

Nagios can determine if a given host is unreachable based on which hosts are defined as parents to other hosts. In this example, a host has gone down and the two hosts beneath it have been marked as unreachable.

Pending service checks are ones that have not provided any results, either because Nagios was recently started and the scheduled checks have not run yet, or because those service checks are not scheduled to run at all.
Active monitoring

Nagios schedules and initiates the service checks defined within its configuration files.

Depending on your network security policy, active monitoring may not be feasible for all services, since the server being monitored must be able to accept an inbound TCP/IP connection from the Nagios server.
Passive monitoring

Servers are responsible for performing their own service checks and reporting the results to the Nagios server.

Passive monitoring offers more flexibility, since more servers are typically allowed to make outbound TCP/IP connections than accept inbound requests.
Service checks are plugins

- Plugin performs a check and return a status code
  - Returning additional descriptive text is optional
- Nagios ships with no plugins; go out and get some
- Lots of custom plugins have already been built
- When rolling your own, try to adhere to standards
- Keep plugin’s resource impact low (use sh or Perl)

Between the Nagios plugin development documentation and the GNU and POSIX coding guidelines, there is a lot of good advice about writing clean, portable scripts to perform service checks.
Nagios installation

- See if your distro offers a pre-built Nagios package
  - SuSE, Ubuntu, Debian, Fedora, among others
  - Otherwise, you will be compiling from source
- Check to see if your network admins already use it
  - If not, you will want to convince them to help out
- Config files are a whole project unto themselves
  - This can be much more difficult than installing DB2

After building Nagios from source, I was pleased to see how much easier the installation process can be when the Linux distribution offers Nagios as an installable package. My tests were performed on Ubuntu 5.10 and did everything short of customizing the Nagios configuration files.
Words of advice

“Don't expect to be able to compile Nagios and start it up right off the bat. It’s not that easy. In fact, it's pretty difficult. If you don't want to spend time learning how things work and getting things running smoothly, don't bother using this software. Instead, pay someone to monitor your network [or] install Nagios for you.

“…Nagios is difficult enough to configure when you've got a good grasp of what's going on, and nearly impossible if you don't. Do yourself a favor and read before blindly attempting to install and run Nagios. If you're the type who doesn't want to take the time to read the documentation, you'll probably find that others won't find the time to help you out when you have problems.”

“Advice for beginners” - Nagios documentation

Although the tone could be a bit less curt, the message is basically fair. The folks on the Nagios mailing lists are generally quite helpful, once it’s determined that the person requesting assistance has a reasonable level of understanding about Nagios.
But it can also be easy

1. Fresh install of Ubuntu Linux 5.1 Breezy Badger
2. Apply recommended Ubuntu updates (Synaptic)
3. Install pre-built Apache from Ubuntu packages
   • Test by browsing http://localhost
4. Install Nagios 1.3 from Ubuntu packages
   • nagios-text is simpler than nagios-mysql
5. http://localhost/nagios should show a login box
6. Built-in service check is already pinging gateway

This is how I installed Nagios for my tests and experiments. I liked the fact that Ubuntu Linux (http://www.ubuntu.com) costs nothing and provides a pre-built version of Nagios that greatly simplified the setup tasks. The only downside is that Nagios has released a 2.0 version that is not yet included in the Ubuntu packages, but that will hopefully be remedied soon.

The Nagios packages include two versions of the CGIs – one built to use text files, and the other writing to a MySQL database. Given the number of things you are already tackling to bring up Nagios, you may decide to use the simpler nagios-text package, which does not require MySQL to be installed and configured as well.
Nagios configuration files

- `nagios.cfg` – main file, read by Nagios and CGIs
- `cgi.cfg` – controls behavior of Nagios web console
- Object config files control all Nagios concepts
  - Services, hosts, contacts, escalation policy, etc.
- Many third party tools ease the pain of config files
- Use an iterative approach to reduce frustration

Nagios ships with sample versions of each config file that can often be used with little modification. However, the more you stray from the default settings, such as install paths, the more you’ll need to tweak those sample files to get Nagios working. To help with the complexity of the config files, the Nagios daemon can be run with a `–v` flag that will validate the config file specified by the user.
Additional Nagios resources

• Online documentation at http://nagios.org/docs
• Developer/user community at NagiosExchange.org
• Plugin development project at nagiosplug.sf.net
• Groundwork Monitor Architect (Nagios config GUI)
  • Available for free at http://itgroundwork.com
  • The Nagios Book, full preview at NagiosBook.org

“Nagios Ain’t Gonna Insist On Sainthood!”

The Nagios Book, written by Chris Burgess, is not yet published, and may not be available online once it is. Either way, the book offers a good deal of assistance with demystifying several aspects of Nagios, especially the configuration files.
Key DB2 performance indicators

- Bufferpool hit ratio
- Page-level I/O stats
- Prefetch efficiency
- Piped vs. overflowed sorts
- Statements per transaction
- Average lock wait time
- O/S level CPU load average
- Database files closed
- Stolen agents
- Package cache overflows
- Secondary log files open
- Longest running UOW
- Rows read per statement
- O/S level iowait percentage

…and many others as well.

Learn which ones are problems in your shop and watch them.

Don’t forget to monitor your application response time, too.

This section is not going to cover as much detail as you will find in other presentations on DB2 performance tuning. There are several excellent presenters who cover the exact formulas for revealing important DB2 statistics. The purpose here is to briefly point out some tips that will help get those statistics published by monitoring scripts to a central console.
How do I get all those numbers?

- GET SNAPSHOT command (SYSMON or higher)
  - Classic method for gathering monitoring data
  - Formatted text output requires parsing
- db2pd utility (SYSADM only) *inspired by onstat*
  - Efficiently accesses internal DB2 registers
  - Also requires parsing (wouldn’t XML be great?)
- Snapshot UDFs (SYSMON or higher)
  - As flexible as SELECTing from any SQL table
  - Requires a connection to one of the databases

As you can see, each approach has their pros and cons. You will most likely end up using a combination of these methods to obtain the health and performance data you’ll be publishing.
Snapshot UDF considerations

- They can be run on any database in the instance
  - So run snapshot UDFs on a dummy database
  - Live database won’t see or count the SQL then
- Push the calculations right into the SQL
  - But use DECIMAL() to keep the precision
- Try to avoid front-ending the UDFs with views
  - Views add overhead to each DB2 instance

Some statistics are affected by the number of statements executed, so it helps to segregate monitoring statements from those of the application workload.

The database name is a parameter in each snapshot UDF, so you can specify the production database name no matter which database is actually executing the UDF.

I’ve seen Nagios plugins for DB2 that require custom views to be built, and I am not a fan of the practice. The UDF call does add a bit of clutter to the SELECT statement, but it is manageable in my opinion without building a set of views to hide it.
Easy DB2 monitoring

- DB2's Health Monitor already runs many checks
  - So health monitor UDFs make for easy targets
- You may have other scripts laying around already
  - Use wrapper scripts to re-use that logic
- This one is nice and quick:
  - `db2 get snapshot for db on <dbname> | grep currently`
- If you don’t know where to start, mine the old logs
  - `db2diag.log and <instname>.nfy`

Just by looking at `db2diag.log` and the instance notification file, you can get a good feel for the types of problems your DB2 server has been encountering up until now. By focusing on monitoring those problems first, you are likely to have the greatest impact on reducing downtime.
One last DB2 tip…

- Did you know DB2 UDB includes a web monitor?
- DB2 Web Tools was added in V8.1
  - Deployed as J2EE .war file
  - Runs in many popular servlet containers
- Web Health Center displays health indicators
- Web Command Center provides SQL access
  - HTTPS protocol supported
- Help your J2EE guys deploy it into Tomcat / JRun

The existence of this tool regularly takes people by surprise. You may want to investigate the possibility of using it instead of Control Center on remote machines, especially when DB2’s service port is blocked to the outside world.
Storing and graphing historical performance

- You could dump everything into DB2 Express-C
  - But you’d still need programs to render graphs
- How long should you retain the raw data?
- Does it matter if some results never quite arrived?
- Counters just keep growing, but you want deltas
- How are other shops handling this?

Some monitoring tools build up a specialized data warehouse of performance information to be mined at a later date, but that may be a bit too elaborate for smaller shops. Fortunately, there are tools that offer much of those features with less complexity.
Don’t use MRTG, use RRDTool

• 1995: MRTG was a nice router traffic graphing tool
  • Clever folks frequently abused it for other uses
• Tobias Oetiker adapted his MRTG into RRDTool
• Fixed data storage size with rich graphing ability
  • Not just for routers anymore
• Industry standard for graphing time-series data

http://rrdtool.org

People are regularly sending impressive samples of their RRDTool graphs to Tobi for display on the RRDTool website. The gallery section serves as an inspiration of what is possible with the tool.
How a round-robin database works

- RRD database file is pre-allocated with space
  - The file never grows beyond the specified size
- Newest data points have the best resolution
- Older data points pruned to make room for new
- Resulting gaps are averaged over time – “lossy”

One very nice thing about the RRD architecture is that since an RRD file is pre-allocated and never grows, it can continue to function on a server that has run out of disk space.
Defining data points

- RRDTool knows about counters vs. gauges
- It also handles counters that reset or wrap around
- You can store actual values or derive from them
- Holt-Winters forecasting algorithm is available
  - Data smoothing, also flags improbable values
  - Enables confidence bands / standard deviation

Because of this degree of flexibility, it seems like DB2, with its mix of snapshot counters and gauges, and RRDTool were meant for each other.
Turning numbers into graphs

- RRDTool can graph actual data or a calculation
- Reverse Polish Notation for entering formulas
- Graphing engine extremely flexible with options
- Graph definition is not part of the RRD file
- Graphs can be added or redefined at any time

This graph came from a production DB2 server that ran a high-volume forms processing website. Most of those connections were managed in pools by application servers, so significant growth in submitted transactions was handled by pool managers that smoothly increased or reduced their connection count. The value of this graph is its ability to quickly convey the relative usage levels of the database throughout the day.
More eye candy

- RRDTool can store and graph multiple data points
- Calculations can involve more than one variable
- Labels and legend are automatically maintained

So what story is this graph trying to tell us?

This graph was useful in helping us spot any runaway processes that gobbled up disk space, as well as giving us an idea of how long it took to complete the nightly DB2 backup.
Getting RRDTool up and running

- RRDTool is often built directly from source
- zlib, libpng, libart_lgpl, and freetype required
- Other than that, it’s a standard “configure + make”
- Populate test databases quickly via `rrdtool update`
- It’s worth the effort to publish graphs onto the web
  - Integrate with your web-based monitoring tools

No matter what you are trying to run RRDTool on, the odds are that someone else has already gotten it running there and may have even posted a HOWTO or other article describing how they did it. Integrating Nagios with RRDTool is also very popular, and is documented rather well.
Alerts and notifications

Host A belongs to a Nagios host group, which alerts a contact group
Service B can be defined to alert a different contact group

Nagios will attempt to alert all members of both contact groups, but only after testing all applicable alert filters.

All of this is defined in the Nagios object configuration files.

The layered approach that Nagios offers to defining notification policies makes it easy to significantly change the notification rules for short-term reasons, and then revert back to the normal policies later on.
When a service repeatedly goes into and out of an alert state, it is called “flapping”, and Nagios allows each service to have its own description of what type of behavior should be treated as flapping.

Host and service-specific filters are good ways to avoid alerting administrators for problems on non-production services.
Effective notification policies

- Describe your enterprise hierarchy in the config
  - For example, httpd depends on DB2
  - Helps to avoid alerting downstream victims
- Avoid hardcoding specific people in notify configs
  - Use dba@foo.com instead of jsmith@foo.com
  - You can easily manage the aliases elsewhere
- Exploit filters to reduce unnecessary alerts

It’s a lot easier to win buy-in from entire team if the monitoring system isn’t bothering them all night long. Any effort in fine-tuning the notification rules to send alerts only when absolutely necessary will factor greatly in the success of the implementation. However, the users will likely revolt if alerts are not sent out soon enough, so a balance based on a solid understanding of the system is necessary to keep both sides happy.
Summary

- Flexible web-based monitors are worth the effort
- Work toward an enterprise-wide dashboard
- Best of breed open source tools are easy to spot
- Others have already done much of the heavy lifting
- DB2 fits into this pattern better than ever

I hope these slides have helped you see the possibilities presented by just a few of the more well-known open source tools out there. Although many of the open-source monitoring resources for databases tend to favor MySQL or Oracle, they can often be adapted or mimicked for use in a DB2 environment. Aggressive, proactive monitoring is the key to understanding what “normal” looks like on any given system, which makes it a lot easier to spot abnormal behavior that might otherwise go unnoticed.
Fred R. Sobotka
FRS Consulting, Inc.
fred@frsconsulting.com