A recipe using an Open Source monitoring tool for performance monitoring of a SaaS application.

Sergiy Fakas, TOA Technologies

Nagios is a popular open-source tool for fault-monitoring. Because it does not have performance data reporting abilities in basic installation, it has not been used for performance monitoring. This paper is a detailed case study of instrumenting a commercial SaaS solution with Nagios for both fault and performance monitoring using:

- Nagiosgraph to store and graph performance data;
- custom scripts and NRPE to securely access data requiring root privileges;
- scripts simulating end-user activity for SaaS application availability and performance monitoring;
- monitoring management as part of ITIL-compliant procedures.

Background

TOA Technologies provides a SaaS mobile workforce management solution. The application is multitiered, consists of many components, connects to client’s CRM and ERP systems using different protocols, and is:

1. Mission critical role for client’s business 6-7 days a week, 365 days per year, 18 hours per day on average.
2. A complex SaaS application with patented non-deterministic internal algorithms, and layered transparent interfaces.
3. A client base doubling in both volume and count every year on average, each with customized configurations.
4. “New kid in the block” effect – SaaS solution’s performance and reliability must exceed on-premise legacy solutions to be successful in market. Industry-leading SLA (Service Level Agreement) must be achieved.

Requirements analysis

Even well-designed and implemented applications which use reliable operating systems (OS) and redundant hardware need careful proactive monitoring. Implementation of a reliable, flexible and effective fault monitoring system is a key successful factor for all SaaS solutions. Successful support of complex SaaS services needs performance monitoring, and accurate and comprehensive data for Capacity Management. A complex application and a large domain of input data renders impossible relevant performance modeling and capacity planning using traditional statistical methods. After defining initial requirements both for fault and performance monitoring, we found that these requirements match in most key aspects, e.g the monitoring system must be:

1. Flexible, with the ability to create and use application specific methods of parameter datum acquisition.
2. Agile, and provide information in a comprehensive but user-friendly manner without undue costly customization.
3. Scalable, to support a fast growing user base, count of hosts, and complexity of services.
4. Minimal footprint on the resources of monitored hosts, both in terms of MIPS and bandwidth required.
5. Secure architecture to support an ISO-27001, TIA-942, and PCI-DSS compliant isolated datacenter topology.
6. Simple to deploy and cost-effective to support at the highest levels of availability in the industry (99.995%).
7. Open source, in order to be able to perform source code audit at the highest world standards for compliance.

All performance metrics are a subset of fault monitoring indicators; no waste was accepted in the design of the system, and all outputs have a defined procedure for all data produced. This dataset includes metrics from every
service abstraction level – from hardware and OS performance metrics up to application and client-specific processes performance indicators. E.g.

- Hardware metrics: CPU usage, I/O rate, HDD free space, network connection latency.
- OS metrics: Apache throughput, Apache sessions count, MySQL set of performance indicators.
- Application basic metrics: count of application server sessions, length of incoming and outgoing queue for connection proxy agents, average amount of transactions per time interval.
- Client-specific metrics: performance of periodical batch data processing tasks, amount of requests to specific service components, latency of external 3rd party and client's services interfaces.

Any violation of predefined thresholds requires an appropriate procedural reaction from a Service Desk engineer to prevent service performance degradation or service outage. Service Desk engineers need access to dynamic metrics as well as historical data to be able to make purposeful and effective support decisions. The monitoring group needs historical data for fault monitoring audits and to establish reasonable threshold values.

**Opportunity**

If the requirements listed above could be met with Nagios, we could deploy only a single tool for fault and performance monitoring, with all the inherent cost and process reduction opportunities. We were successful.

**Nagios as performance data collection and display tool**

Nagios\(^1\) is a well known and widely used open-source fault monitoring tool. As the de-facto industry standard, Nagios provides a user-friendly web interface, flexible and scalable architecture and the ability to create and use any service-specific checks. In addition to the considerable number of checks available from the basic installation of Nagios, it is easy to create further needed specific plugins using available tools including (but not limited to): shell scripts, PHP or Perl scripts, and binary command line executables. Interface to such plugins is documented\(^2\) and must consist of predefined return values and at least one line of text to STDOUT. This text can include text information reflecting the state of indicators accompanied with pre-formatted performance data separated by the “pipe” (‘|’) character. Below is a sample of Nagios’s “out-of-the-box” plugin check_load output which checks average CPU load for last 1, 5 and 15 minutes:

```
OK - load average: 0.46, 0.68, 0.64|load1=0.456;1.000;2.000;0; load5=0.681;1.000;2.000;0;
load15=0.645;1.000;2.000;0;
```

Part of the string preceded by '|' is the performance data. The first figure after the metric identifier (load1, load5 and load15) is the actual value of the CPU average load, followed by “WARNING” and “CRITICAL” threshold values. This output is easy to parse and import using very simple, and therefore fast code.

All data which Nagios receives from plugins are stored as text in log files and can be used for creating a performance database, as will be shown later in this paper. However, these logs need additional processing which can consume considerable time and resources, making use of this data for operational needs almost impossible. That is why Nagios performance graphing add-ons have been developed by the Nagios community. Such add-ons provide instant performance monitoring using “on the fly” plugin output parsing, and store data in the widely used RRD format. The most popular add-ons are Nagiosgraph\(^3\), NagiosGrapher\(^4\) and PNP4Nagios\(^5\).

---

\(^1\) [http://www.nagios.org/](http://www.nagios.org/)

\(^2\) [http://nagios.sourceforge.net/docs/3_0/pluginapi.html](http://nagios.sourceforge.net/docs/3_0/pluginapi.html)

\(^3\) [http://sourceforge.net/projects/nagiosgraph/](http://sourceforge.net/projects/nagiosgraph/)

\(^4\) [https://www.nagiosforge.org//gf/project/nagiosgrapher/](https://www.nagiosforge.org//gf/project/nagiosgrapher/)

\(^5\) [http://sourceforge.net/projects/pnp4nagios/](http://sourceforge.net/projects/pnp4nagios/)
We implemented Nagiosgraph, and all samples in this paper use graphs built by this add-on. Links to Nagiosgraph's produced performance graphs are incorporated into the standard host's "Service details" page, as well as into certain service details pages. It is possible to examine performance data at the host level using a page to show graphs for all custom-configured performance metrics. This page contains all performance data graphs aligned by a time datum line to compare these metrics for the given period of time.

E.g. Fig. 1 demonstrates a sample\textsuperscript{6} performance graph page (the top fragment) generated for all performance indicators being collected on the back-end host. The "CPU load" metric correlates with processes running better than with the number of application server sessions, an example of tuning data to anticipated analysis use.

\textsuperscript{6} This and all screenshots in paper have been altered in order to hide real information about client and real server names but shows actual performance data.
For every particular indicator it is possible to review predefined performance graphs for last day, last week, last month and last year to examine performance historical trend at a glance. Fig. 2 is the example of such a page and shows historical data for incoming streamed data request queue size, one of the key performance indicators for the application's B2B interface. These graphs demonstrate weekly load patterns and trends of increasing incoming transaction volume during the given month.
Since NagiosGraph utilizes RRD files as a performance data storage it is obvious that we can easily export data to any statistical tool, in order to perform more precise trend analysis. Thus Nagios with NagiosGraph provides the same level of functionality as most of the widely used dedicated performance monitoring tools. Combination of fault-monitoring and performance monitoring in a single tool saves the effort of supporting multiple monitoring systems, prevents excess load on the network and the monitored hosts, and provides an agile interface to historical data for support engineers (and subsequent management and customer uses).
Nagiograph also allows usage of Nagios logs to import data into a performance DB, so long as it does not alter performance log content and uses dedicated RRD files. This aspect and advantages of log processing in this manner will be shown later in this paper.

**Mitigating security risks**

Mitigating security risks is a top priority in building applications with complex topologies for SaaS solutions. According to many surveys regarding user’s perception of SaaS, security-related topics are always the biggest concern. Since the monitoring host needs to have access to every server in the datacenter, it becomes one of the weakest points amongst all servers present there. Nagios keeps all configuration, including access credentials, in plain text files which is convenient for the support of monitoring systems, but extremely dangerous from a security point of view. Compromising such a monitoring host can give an intruder access to every host. In these circumstances, agentless monitoring via remote SSH terminal access, a usual solution for the UNIX world, creates significant security risks. Restricting monitoring account permissions can’t help, since even restricted accounts with remote access privileges can be dangerous. Once logged in, an intruder can create excess workload or apply specific exploits to gain root privileges. Such exploits are known and even such secure and stable OS such as FreeBSD can be affected.

Thus, despite the obvious extra effort needed for support, monitoring agents mitigate security risks by tapering possible intruder’s access down to an ability to call only a limited set of harmless commands. It is also reasonable to employ the SSL protocol and configure monitoring agents to be listening to non-standard ports. Nagios is accompanied with Nagios Remote Plugin Executor (NRPE) which provides all these needed features, and uses simple and manageable configuration files with a minimal footprint at the monitored host.

The same tapering techniques can be used to monitor MySQL, which involves similar security risks. Besides the standard useful set of performance metrics available to unprivileged accounts, some values require “system” privileges to access the data. “Binary log position”, which is needed for replication monitoring, is one example of such a parameter. Security risks of providing system-level privileges to accounts used for monitoring are higher than with remote access to hosts: there is no need for any exploits in this case – the system privileged MySQL account is intended to have the ability to drop tables or databases, for example.

We instead encapsulated privileged queries into stored procedures, thereby avoiding the need to provide system privileges to the monitoring account. The idea of this well-known approach is that all clauses of stored procedures are executed in system privileged context while un-privileged accounts are able to call only a predefined set of such stored procedures. In order to unify deployment of monitoring MySQL, all stored procedures used for monitoring purposes have been allocated into a dedicated DB published to security DBs.

**Customizing Nagios and service for comprehensive monitoring**

It is natural that the “out-of-the-box” set of Nagios plugins are limited to common OS checks and metrics. It is also obvious that this set of ready-made checks is not sufficient for monitoring of a commercial application. There is a set of metrics for different components of service which needs careful performance monitoring and capacity evaluation. These metrics represent the state of the internal application’s data structures and interfaces that must be created at the development stage. These interfaces must be uniform for all application components in order to decrease the number of different plugins needed to collect application’s performance indicators.

We implemented internal standards for application state and performance data interfaces. There are two (2) standard interfaces for state and performance information for all our applications:

1. SOAP function.
2. HTML page with a standard table consisting of 2 columns: “Parameter” and “Value”.

Despite the fact that SOAP functions better than HTML, we need to use the 2nd interface because not all our applications utilize XML, but all applications use HTTP. So, adding XML support to such components would increase complexity and decrease reliability. Having two (2) standard interfaces for every application used in our service, the Monitoring Team is able to save effort and develop fewer plugins for monitoring our application.

“Incoming requests queue size”, illustrated with Fig. 2 is a good example of this approach. This is the performance indicator for the one of our B2B proxy agents which interfaces with client’s CRM system. It processes incoming streaming data and feeds it into our main application in real-time. Any connection disruption or performance degradation affects client’s CRM and also makes data represented to end users unreliable. That is why this proxy application needs careful monitoring for state of connection, processing performance and error rate.

Taking these requirements into account, the development team defined key performance indicators collected into an HTML table. During User Acceptance Test, the Monitoring team collects sample performance data using standard plugin for HTML interface. On the basis of this data, we extrapolate it to evaluate capacity needed for this B2B interface and thresholds for fault monitoring. After this interface is in production, we use reasonable thresholds for monitoring and employ standard tested plugins for that purpose.

Another good example of a custom developed plugin is the plugin used for monitoring of end-user interface availability and responsiveness. This is a common problem in complex systems: the engine is running but transport is broken. We check service availability and measure response time via same interface that service’s end users utilize. We developed our availability check for the XHTML application interface. XHTML interfaces have been chosen because most of our end users employ it, and it minimizes traffic, one of the largest real environmental impacts for a SaaS service. Our web interface is build using Ajax, which makes pages “heavy”.

This plugin, developed using Perl (as with most of our custom plugins) opens an XHTML application interface (used for mobile devices), logs into our service, gets the end user service initial page and then logs out of the application. Time spent for every step is provided in performance data from the plugin, as well as total time. Nagios runs this probe every two (2) minutes around the clock. Having execution time for every step, we are able to define which component causes service performance degradation or outage. Thresholds are set for totals, and this parameter is contracted in our Service Level Agreement as our “service response time”. Our company provides “Service level” reports to our clients on a monthly basis.

This report caused the problem which demonstrated limits of Nagiosgraph with storing performance data. To minimize file size and resources required for trending, RRD file contains the averages of performance data but not exact values. Customers, however care about exact service responsiveness at the certain time. We therefore had to use another approach to reporting performance data. Since Nagios stores the output of the plugin from every check attempt as well as other useful information [time of check, check state (OK, WARNING or CRITICAL), some debug information etc.], in order to be able to create service level reports and perform deep analysis Monitoring Team developed a tool which uploads data from this log into a MySQL database.

These 2 examples of custom built plugins demonstrates Nagios flexibility in building well-tailored monitoring, and different approaches of performance reporting.

**Monitoring management as part of ITIL processes.**

Due to SaaS requirements, characteristics of our service and market strategy, we need to provide monitoring customized for every client of a rapidly growing user base. A single tool for performance and fault monitoring supports cost control efforts and provides increased ability to use a holistic approach for developing and deploying monitoring in a volatile environment. There is no silver bullet; even the most powerful and flexible tool can’t provide a comprehensive and reliable solution without skilled personnel and mature, well-defined processes. To ensure high level of service, ITIL Monitoring management processes are mandatory for our firm.

The Monitoring group is the part of Level 3 support team inside the Service Assurance department. This enables effective coordination and interaction of Monitoring management with the Service Level, Capacity, Release, Change and Configuration management processes.

Monitoring requirements should be established at the earliest stages of the new release life-cycle as well as capacity requirements and implementation procedures. Using standard technical requirements, architecture and libraries, the implementation department provides standard interfaces for monitoring on 80/20 basis. The remaining 20% of each solution is new and requires additional specification of interfaces and preliminary capacity evaluation.
During User Acceptance Test, the Monitoring group in cooperation with the Service Desk performs testing of standard and custom monitoring for the new release. At the same time, the Capacity Planner is able to receive preliminary (test) performance information, which allows valid capacity estimation for production.

Before each new release, the Monitoring group in cooperation with Customer Support and the Service Desk develops a customized Monitoring Plan. The Monitoring plan is the kind of standard document used in our company which describes all checks deployed to production with each release. For every check it describes:
1. Name of check.
2. Method used for monitoring.
3. Poll interval.
4. Time period.
5. Critical and warning thresholds.
6. Is this check intended for performance monitoring.

This document provides the ability to store monitoring configuration in human readable format and develop Knowledge DB articles for Service Desk using client’s support documents. We also use standard Monitoring Plan templates to deploy new hosts or service instances into production. Standard documents and monitoring deployment processes incorporated into Release and Change management ensure timely, reliable monitoring for our service. The Monitoring configuration audit is performed by Configuration manager.

The monitoring system deployed with solution described in this paper provides and inexpensive source of quality data for Capacity management and Service Level management. This information being fed into ITIL compliant processes enables proactive support decisions and consistently high availability and performance of our service.

**Conclusion**

De-facto industry standard monitoring tool Nagios, accompanied with graphing solution and in-house developed plugins and tools provides backbone solution for high flexible and efficient performance and fault monitoring system for complex, fast growing SaaS product. Integration of this system with matured ITIL compliant process saves up lot of efforts for support and planning of our service.