About GroundWork Monitor

WAS THIS PAGE HELPFUL? Leave Feedback

Overview

This page presents the *GroundWork Monitor Theory of Operation* including in-depth technology and procedural overviews.

1.0 Technology Overview

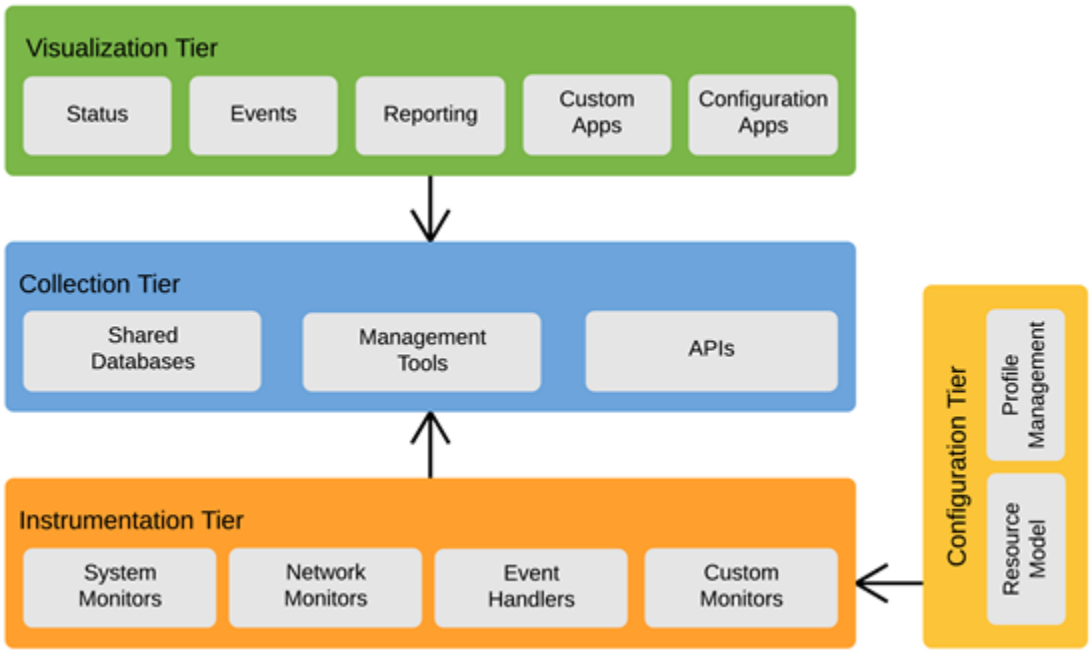
GroundWork Monitor combines modular open source technologies with a unified information management subsystem to provide a low cost and consistently managed monitoring solution that is also extremely flexible. This is achieved through the use of a loosely-coupled component-based architecture, whereby a core set of data-management services provide collection and control functions while independent and self-contained instrumentation and visualization components provide the input and output services that are needed to manage a specific network. By using this approach, GroundWork Monitor is able to provide a stable and consistent monitoring platform, while also allowing for an almost unlimited number of monitoring and reporting tools to be implemented as discrete components.

1.1 The Logical Information Model

From a high-level perspective, GroundWork Monitor uses an abstract information model comprised of four distinct logical tiers, as illustrated in the diagram below. First, components in the *Configuration* tier define the network resources that need to be monitored, and also describe how monitoring data should be processed after it has been gathered. Once the resources have been defined, task-specific instrumentation components (*Instrumentation* tier) are used to collect sampling data through a variety of different monitoring interfaces. After that, integration tools in the *Collection* tier gather up the data for processing and storage. Finally, web applications in the *Visualization* tier at the top of the model retrieve and display the information to the users.

This loosely-coupled component-based approach provides a managed yet extensible framework that ensures data flows through the system in a consistent and predictable way, while also allowing GroundWork Monitor users to deploy the monitoring and reporting tools that are needed for each specific network environment. As such, customers can mold GroundWork Monitor to suit the specific needs of their own environment, while still preserving a consistent information management model.

Figure: Logical Information Model



1.2 The Software Architecture

Although the information model is useful for thinking about the way information flows through the system, it's also important to recognize that it is a highly abstracted representation of the overall architecture, and does not always fully mirror the system-level mechanics.

In particular, GroundWork Monitor makes heavy use of several independent open source tools and technologies, most of which are self-contained entities in their own right. However, GroundWork Monitor does not force these component tools into a fixed framework, but instead allows the constituent tools and technologies to operate in their native modes, and then relies on integration technologies to tie the otherwise independent components into the broader monitoring platform. This approach preserves the power and capabilities of the individual components, while also ensuring that the overall information flow delivers the necessary level of functionality.

This concept is illustrated in the figure, which shows the major components of GroundWork Monitor along with many of the commonly used component tools. As can be seen from that example, a fully-configured GroundWork Monitor system can have multiple discrete components performing their own task-specific functions, while the information management toolkit provides the integration technologies that are needed to bridge the individual components into a unified and cohesive monitoring platform.

As stated earlier, the logical information model can be used as a reference guide to help understand how information flows through the system, and this remains true even when multiple components are involved. Regardless of the number of components in use, resources are still described by the Configuration subsystem, while sampling data is still retrieved by components in the Instrumentation tier, then stored and processed by the Collection tier, and eventually read for display by one or more tools in the Visualization tier.

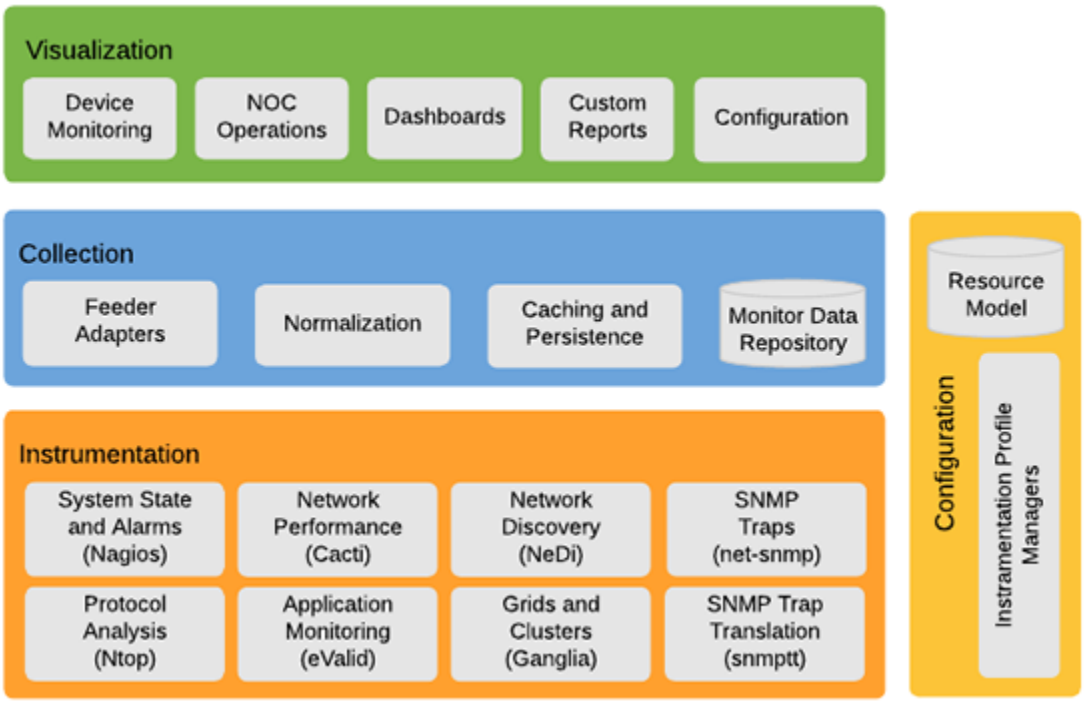
Using the information model to examine the Software Architecture diagram, we can see that the Configuration tier governs the behavior of the *Instrumentation* and *Collection* tiers. In particular, this tier is responsible for cataloging the network resources that need to be monitored, which is subsequently reflected in all of the other tiers. However, this tier is also responsible for describing the monitoring components that are used by the *Instrumentation* tier to gather the actual readings, and it is also responsible for controlling how the resulting data is eventually processed and stored in the *Collection* tier.

Next, the *Collection* tier incorporates the shared databases and integration tools that are needed to tie the independent components into a cohesive system. Parts of this subsystem provide embedded database tables that store global resource definitions (such as host and service assets), as well as embedded database tables for the collected data readings, but this tier also includes some standalone databases that are specific to some of the individual components (such as the round-robin database files that store long-term performance readings for historical graphing). Apart from the database components, this tier also contains tools that perform the necessary integration tasks (such as pulling data from external log files, or pushing configuration changes out to the discrete components), as well as development tools that provide controlled access to the shared databases and system management services.

Finally, the top of the stack contains the web-based applications that constitute the *Visualization* tier. The *JBoss Portal* is an open source and standards-based environment for hosting and serving a portal's Web interface and publishing and managing its content. The *JBoss Portal* unites GroundWork Monitor web applications into a common presentation framework. GroundWork Monitor includes a variety of web applications ranging from single-purpose tools that display the contents of a specific database, to complex applications that pull data from multiple sources simultaneously, and can also integrate wholly separate web applications from remote hosts if needed.

Taken as a whole, the loosely-coupled component-based architecture provides a tremendous amount of flexibility. On the one hand, the structured information model provides a consistent platform that facilitates extensibility without sacrificing manageability, while the use of modular component technologies within each tier also allows administrators to customize the monitoring platform to reflect the requirements of their specific network. This concept is explored further throughout the remainder of this paper, which discusses the tiers of the information model and their constituent component technologies in greater detail.

Figure: The Software Architecture



1.3 The Configuration Tier

Although the configuration tasks are not a formal part of the formal data-gathering and processing cycle, they still play a crucial part of the overall information flow.

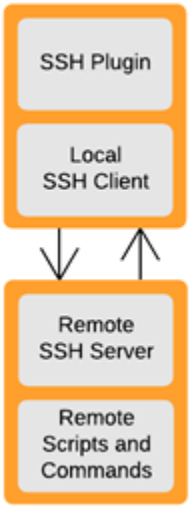
For one thing, the *Configuration* tier is where network resources are itemized, which in turn affects almost all of the other components across the system. This includes the low-level monitoring data that is initially gathered for each of the itemized network resources, but also includes the components in the *Collection* tier that have to map secondary data sources to known resources, and even includes lightweight components such as the lists of known hosts and services that are used by some of the reporting tools. In this regard, the configuration tools in this tier affect everything else that the system does, and are absolutely crucial to the successful operation of GroundWork Monitor, even if these components are not directly used in the ongoing data-processing cycle.

Beyond the basic configuration tasks, the *Configuration* tier also affects how the other two lower tiers function - it controls how the resource-specific data is gathered by the *Instrumentation* tier, and it also tells the data-processing tools what to do with the data after it has been collected (such as whether or not the performance readings should be recorded into the long-term performance databases).

For example, the main component in this tier is a web-based configuration front-end to Nagios that is based on the open source *Monarch* configuration tool that is separately published by *GroundWork*. Specifically, the configuration tool stores the *Nagios* configuration settings into an application-specific embedded database table, and then uses integration scripts to synchronize the configuration tables with the GroundWork Monitor asset database. Once the configuration has been finalized, back-end scripts generate the actual *Nagios* configuration files, and also update any other application-specific configuration repositories that the system knows about.

This specific arrangement has the primary benefit of relieving administrators from having to configure *Nagios* control files by hand, but more importantly it also becomes possible to ensure that all of the other components have the exact same view of the network, meaning that administrators are further relieved from having to reconfigure all the other components after every change to Nagios. This approach also makes it much simpler to extend *Nagios* with new features and add-ons, which in turn contribute to the power of *Nagios* itself.

1.4 The Instrumentation Tier



Although GroundWork Monitor supports many different tools for gathering monitoring data, it uses the open source *Nagios* monitoring toolkit as its primary instrumentation mechanism. This is largely due to the fact that *Nagios* has an open and lightweight textual plug-in interface that can work with almost any command-line tool, and this simple and open interface has resulted in a huge library of *Nagios* plug-ins being developed that can collectively monitor thousands of different sensors and variables, with very little effort required by the system administrator.

Just for starters, *Nagios* includes a variety of Linux scripts and utilities that can be used to fetch operating system details from the local server, including things like disk and memory usage, CPU utilization levels, and other types of system information, with all of this information being obtained by simply executing a local command and parsing the response text. But since *Nagios* uses a lightweight command-based interface for these tools, it is also possible to pull the same kind of data from other systems simply by calling on a redirection mechanism of some kind.

For example, *Nagios* provides a generic SSH wrapper that allows the "local" Linux utilities and scripts to be executed on remote systems, simply by creating the appropriate SSH user and session keys and then copying the desired tools to the target systems. In this setup, *Nagios* calls the SSH wrapper tool with parameters that identify the target host and the remote command to be executed, while the remote output is passed back to *Nagios* for processing. More importantly, the SSH wrapper is not specifically limited to the bundled *Nagios* commands, and in fact can be used with any command as long as the target program has the ability to return an appropriately formatted response. As such, *Nagios* is able to use the SSH wrapper to query almost any kind of networked platform, as long as the target command can return a compatible textual response.

Nagios also supports remote command execution through a Nagios-specific client-server protocol called NRPE which is similar to SSH but more direct. In this model, remote systems run an NRPE listener, while the Nagios server uses an NRPE client to connect with remote hosts and issue local commands, with the command output subsequently being returned to the Nagios server. As with the SSH interface, the NRPE listener allows any remote command to be executed as long as the target program is able to return a properly formatted response.

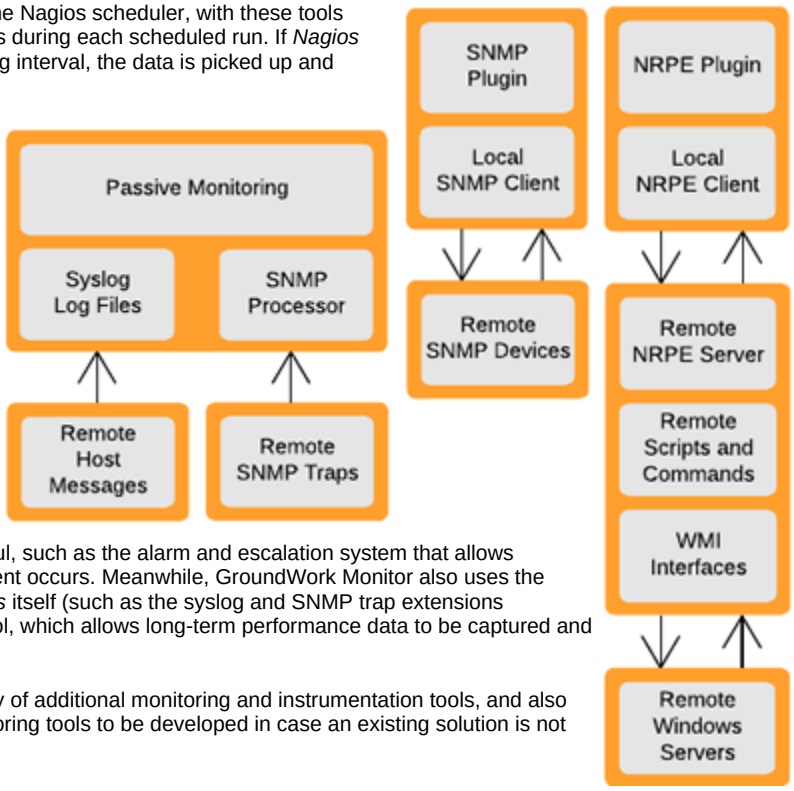
GroundWork also provides a version of NRPE for Windows which includes a collection of VBScript utilities for gathering Windows-specific data through WMI calls. Since WMI also works over the network, the GroundWork Monitor port of NRPE for Windows also has the inherent ability to function as a WMI proxy for an entire network of Windows devices, whereby the GroundWork Monitor server simply identifies the target Windows host in a parameter to the VBScript commands.

Nagios also provides a variety of tools for performing network-level tests, such as ICMP "ping" tests that simply check for host availability and responsiveness, as well as TCP/IP probes that can see if a specific port is responding to connection requests within an acceptable threshold. Nagios also provides the ability to actively test network applications by utilizing the network protocols themselves, such as logging into a remote mail server, or querying for a well-known DNS domain name, with all of these tests ensuring that the target service is actually functioning correctly and not just listening for incoming connections.

Naturally, *Nagios* also supports the use of SNMP, which allows it to be used for everything from network device probes to detailed system queries, with the only restriction being the number of SNMP MIBs available. As with the other plug-ins, *Nagios* calls a local SNMP client and provides it with the target host and SNMP MIB to query, while the client returns the collected data to Nagios in the necessary format.

Apart from the probe-based plug-ins, Nagios also provides a "passive" monitoring interface that allows

autonomous tools to gather information independently from the Nagios scheduler, with these tools recording important output into log files that *Nagios* processes during each scheduled run. If *Nagios* finds any new event messages in these files at the next polling interval, the data is picked up and processed the same as if it were a regular monitoring event.



Commercial versions of GroundWork Monitor include plug-ins that capture and filter syslog messages and SNMP traps and then pass the resulting messages to Nagios over the passive interface, both of which allow *Nagios* to be informed of asynchronous, out-of-band events that might otherwise be missed by the poll-based tools. On a more general level, Nagios also provides a network protocol called NSCA that allows remote systems to push local command output to Nagios' passive interface across the network, which effectively allows multiple independent monitoring systems to feed monitoring data to a single master server for storage purposes.

Cumulatively, the number and breadth of available plug-ins make Nagios an exceptionally powerful monitoring system that is capable of detecting almost any kind of critical event. However, GroundWork Monitor also makes use of other *Nagios* component technologies which are equally as powerful, such as the alarm and escalation system that allows administrative personnel to be notified whenever a critical event occurs. Meanwhile, GroundWork Monitor also uses the open *Nagios* interfaces to add additional value back to *Nagios* itself (such as the syslog and SNMP trap extensions described above), but also by integrating *Nagios* with RRDtool, which allows long-term performance data to be captured and graphed by *Nagios* directly.

As stated earlier, GroundWork Monitor also supports a variety of additional monitoring and instrumentation tools, and also provides programming interfaces that allow whole new monitoring tools to be developed in case an existing solution is not feasible for a particular monitoring task.

1.5 The Collection Tier

In simple terms, the *Collection* tier of GroundWork Monitor provides the glue that binds the Instrumentation and *Visualization* tiers together. More specifically, this tier contains the databases that are of interest to the system as a whole, and also provides the integration tools that bridge the independent components into the overall system. Most of the components in this tier are provided by an open source package known as *GroundWork Foundation*, although some of the independent software packages also provide additional functionality that has been rolled into this tier.

For data storage, *GroundWork Foundation* uses a collection of embedded database tables that provide globally-relevant data (such as object tables that store state information for all of the monitored hosts and services), as well as event tables that store records of all significant monitoring events (such as warning messages that have been generated by *Nagios*). By storing this data in a common location, it can be accessed and used by multiple tools under a common access-control framework.

Separately, GroundWork Monitor also provides a collection of control scripts and programming libraries that provide integration services to the components that need it. The control scripts provide active integration services that move information around as needed, such as moving event records from Nagios to the appropriate database table. Meanwhile, the Perl, PHP, Java and SOAP programming libraries included with *GroundWork Foundation* provide passive interfaces to applications when they need to interact with the system directly.

Apart from the *Foundation* package, the *Collection* tier also contains some other globally-useful technologies that are sourced from other packages. The most obvious examples of this can be found in the Nagios control services that are crucial to the operation of GroundWork Monitor as a whole. For example, *Nagios* provides tools that allow the administrator to immediately probe a specific device for its current condition, and GroundWork Monitor incorporates this tool into the *Collection* tier rather than replicate the functionality in separate tools.

Similarly, GroundWork Monitor uses the RRD toolkit to store and graph long-term performance data, and as such the RRD database and graphing service fall within the scope of the *Collection* tier, even though the technology is sourced from the RRD component instead of the *Foundation* package.

1.6 The Visualization Tier

The top tier in the GroundWork Monitor architecture is the *Visualization* tier, which contains the web applications that are used to display data managed by the *Collection* tier. There are a dozen or so web applications included in GroundWork Monitor by default, a primary component in this tier is the *JBoss Portal* framework and as mentioned previously the *JBoss Portal* unites GroundWork Monitor web applications into a common presentation framework. For detailed information regarding *JBoss* see http://www.jboss.org/.

GroundWork Monitor includes several applications that are integrated into the basic package. For example, GroundWork Monitor includes a status viewer application that provides an overview of a specific host or service by incorporating data from multiple sources, including summary information from the *Foundation* database, graphs from the RRD subsystem, and log messages from events that are stored in *Foundation*. In addition, the status viewer also includes controls that allow the user to manipulate and probe the *Nagios* view of the target resource by way of the *Nagios* control interface.

Meanwhile, GroundWork Monitor includes a console application that provides a simple table view of the *Foundation* event database, which allows network operators to become aware of problems as soon as they have been detected.

GroundWork Monitor also provides several tools for generating long-term analytical reports. First, *Nagios* has its own ability to produce simple graphs of per-host or per-service availability data, all of which are available through the Nagios web interface. Meanwhile, GroundWork Monitor also include an "Reports" application which presents significant historical events across multiple hosts and services simultaneously, providing a snapshot view of the complete network. Separately, commercial versions of GroundWork Monitor also include the open source BIRT reporting tool which allows for the creation of hand-tailored reports.

2.0 Procedural Overview

As was discussed in Chapter 1, GroundWork Monitor combines modular open source monitoring and presentation tools with a unified information management subsystem, resulting in a highly extensible yet consistently managed monitoring platform.

From a high-level architectural perspective, these components fall into one of four tiers - *Configuration*, *Instrumentation*, *Collection*, and *Visualization* - with information flowing through the system as it is specified, gathered, processed, and ultimately displayed. However, this is a highly abstracted view of the overall monitoring process, and the actual data-processing mechanics can often be somewhat more elaborate than the logical model suggests, especially as additional software components and functions are enabled.

This is largely due to the fact that GroundWork Monitor makes heavy use of open source component technologies that frequently have their own information processing models, while GroundWork Monitor relies on integration tools to bring those components into the information model. As such, the software in use tends to be much more important towards day-to-day operations, while the logical information model is most useful as a reference tool for understanding how information flows through the system.

Chapter 1 described the logical information model that is used by GroundWork Monitor and also discussed the major system components as they fit within the context of that architecture. In this chapter, we will examine the information flow from the perspective of the individual software components themselves.

2.1 Resource Configuration

Although the configuration tasks may not seem to be a formal part of the ongoing data-collection and processing cycle, they play a crucial part of the overall information flow in GroundWork Monitor. At a bare minimum, resources must be defined before they can be monitored or displayed to the end-user, and as such the configuration process determines everything else that will happen in the system later. However, the configuration process also affects a variety of subordinate components (such as performance graphing and event collation) which are less obvious but just as critical to the long-term satisfactory use of the system.

Architecturally, the basic unit of data in GroundWork Monitor is the measurement reading that is taken at a specific time for a specific resource on a specific host. Before this data can be collected, however, the system administrator must first define the target host, and must also define the services on that host which need to be monitored. Administrators must also specify the data that they want to collect - do they only want the basic availability information, or do they also want to capture numeric performance readings historical analysis? - and they must also assign general options such as scheduling intervals and notification contacts. Once a resource and all of its necessary options have been defined, a command scheduler will periodically execute the appropriate monitoring tool for that resource, thereby producing a sample reading for the resource in question. From there, the measurement data is subsequently used to perform additional tasks throughout GroundWork Monitor , according to the options that have been defined.

The process of defining a resource and its options is managed through the *Configuration* tier, and specifically the open source *GroundWork Monarch* configuration tool for *Nagios*. This application presents data-entry screens and forms to the user via a web interface, while reading and storing the configuration settings into an embedded database on the back-end. Once the administrator decides to commit the configuration, additional processing is invoked which generates the configuration files and databases that are needed by the individual software components. In this manner, all of the different software components throughout GroundWork Monitor are guaranteed to always have a consistent view of the hosts and services that are being monitored, without requiring the administrator to configure each component separately.

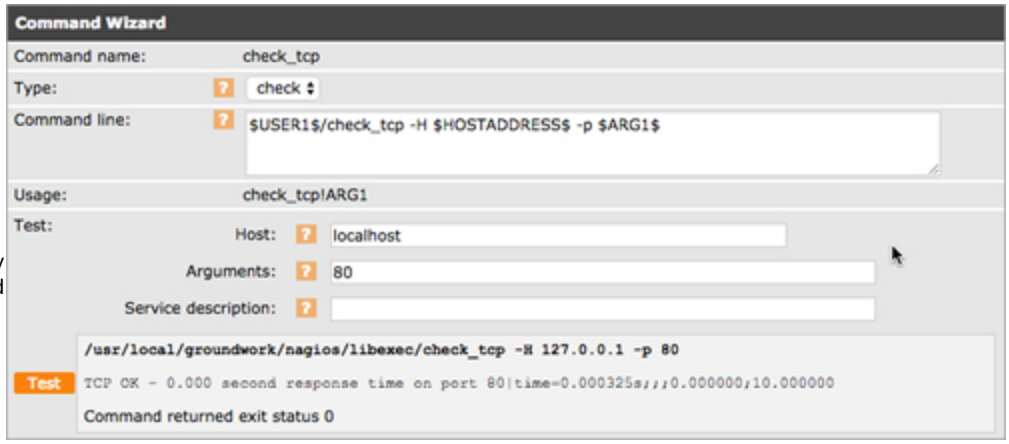
2.2 Command and Service Definitions

Resource definitions in *Nagios* use a nested chain of templates and profiles, each of which define a particular aspect of a common resource type. A complete resource definition is constructed by linking the templates and profiles to a specific instance of a specific resource on a specific target device.

The most atomic unit in this model is the command definition, which essentially provides a plug-in definition that tells *Nagios* how to monitor a particular resource type. The command definition specifies the program or script that will be called, as well as any options that are needed for the plug-in to function. *Nagios*-specific variables (also known as "macros") that will be expanded at run-time can also be used in the command definition, which allows resource-specific parameter values to be filled in as they are needed.

For example, GroundWork Monitor provides a command definition called "check_tcp" shown in the diagram to the right, that allows *Nagios* to test if a particular TCP port on a specific host is listening for incoming connection requests. The full command definition specifies a path to the program, a "-H" option and "$HOSTADDRESS$" variable that specifies the target host,

and a "-p" option and "$ARG1$" variable that specifies the target port number. The use of variables in the command definition means that this single command can be used to construct multiple probes, simply by filling in the variable data with target-specific data.

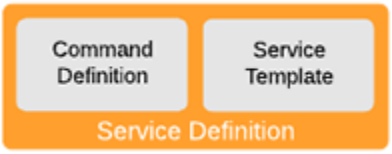


It's important to understand that the command definition is not processed by *Nagios* directly, but instead is only used to define a plug-in that can be called upon later. Once a command definition has been created, it is ultimately mapped to one or more resources, with the resulting resource-specific command being used to monitor the specific resource. Put more simply, the command definition merely provides a building block for defining the actual resource monitors later. GroundWork Monitor includes several dozen predefined command definitions for this purpose, ranging from simple tools like the one shown above, all the way up to advanced tools that connect with application services on remote hosts to collect application-specific readings.

Apart from the command definition, resource monitors in Nagios also requires that certain kinds of governance data also be defined for each monitoring job. Whereas the command definitions describe the basic command-line to be used, the governance data describes control elements, such as the desired polling frequency, the persons to contact if a critical event is detected.

These values can be entered manually when a resource is defined, but they are usually defined in a service template, which is a reusable object similar to the command definition. By default, GroundWork Monitor includes a single "generic-service" template that uses the most common settings, but additional templates can be defined if some resources need different control settings (such as needing to have different administrative contacts).

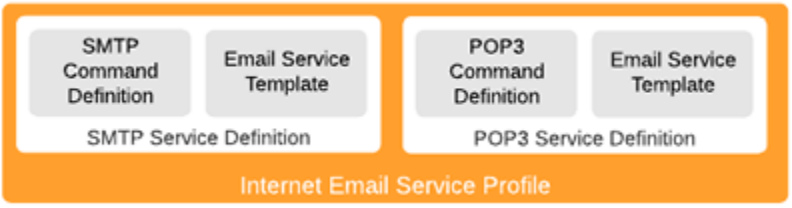
The combination of a command definition and its associated governance data produces a service definition, which cumulatively defines the monitoring behavior for a specific type of resource. Service definitions can inherit all of the attributes from the command definition and service template, or administrators can override some of the attributes, either by filling in variable values from the underlying command definition, or by replacing the governance data from the service template with service-specific values.



For example, the "check_tcp" command definition described earlier has a command-line variable for specifying the target port number. By explicitly defining this variable value as "80" (for HTTP) or "25" (for SMTP), it's possible for the single command definition to be used as a cornerstone for multiple service definitions. Conversely, a generic "check_tcp" service definition could also be created that simply left these variable values undefined, thereby allowing the port number variable to be populated when the specific resource itself is defined.

GroundWork Monitor includes several dozen predefined service definitions, and additional service definitions can be created as they are needed, either by defining new command definitions or service templates, or simply by changing the values that already exist.

GroundWork Monitor also allows multiple service definitions to be linked together as part of a service profile, which can be helpful when a common set of service definitions need to be defined for multiple hosts. For example, if you have a few dozen email systems that all have SMTP and POP3 servers, you can create an "Internet Email" service profile that references the appropriate service definitions, and then apply the service profile to those hosts without needing to explicitly apply each individual service definition to each host. Service profiles can be defined as they are needed, and GroundWork Monitor also includes several dozen predefined profiles in the box.



2.3 Host Definitions

Host definitions follow many of the same principles as command and service definition, albeit with some significant differences. In particular, a host definition does not require the multiple layers of abstraction found in command and service definitions, but instead only needs to have an IP address or hostname, along with some host-specific control data. In the latter case, host templates are typically used in the same way as service templates, and satisfy the same kind of purposes (IE, some hosts need different administrative contacts).

At the next level of abstraction, host profiles are the combination of a host template and one or more service profiles. For example, you can have a generic "Windows Server" host profile that combines a generic host template with service profiles for system-level SNMP and WMI lookups,

along with WMI-based Exchange queries and the Internet email service profile discussed earlier. From there it would be a simple matter of applying the host template to all of the Windows-based Exchange servers on your network.

Nagios also allows individual host definitions to be grouped into logical hierarchies through the use of host groups. Host groups can have some basic attribute definitions (such as a specific contact group or escalation tree), but they are most often used for filtering and sorting purposes by the presentation tools. By default, GroundWork Monitor provides a "Linux Servers" host group that only contains the local system, but host groups can be created to represent any kind of logical grouping that may be desired. For example, you can have host groups that refer to geographic locations, or organizational divisions, or any other kind of arrangement that may be desired. GroundWork Monitor also allows you to predefine a host group within a host profile, so that the organizational hierarchy is imposed on all of the affected host definitions.

Taken as a whole, the judicious use of templates, profiles and host groups allows an administrator to pre-configure almost any resource imaginable. For example, a web server at an off-site data-center might have a host template that notifies a third-party service provider about hardware-level errors, while a "ping" service profile notifies the hosting provider of network reachability problems, while application specific service profiles notify internal help-desk staff of application issues, with all of these settings being predefined in various templates and/or profiles. As stated earlier however, GroundWork Monitor also allows hosts and their resources to be manually defined if needed, overriding the profile and template settings.

Furthermore, the use of templates and profiles within the configuration application (and specifically its underlying relational database) allows for some fairly robust inheritance capabilities. For example, if the contact group for a specific service definition needs to be changed, then the administrator only needs to change the associated service template, and the new contact information will be automatically inherited by all of the associated service definitions the next time the configuration data is saved.

2.4 The Data Collection Cycle

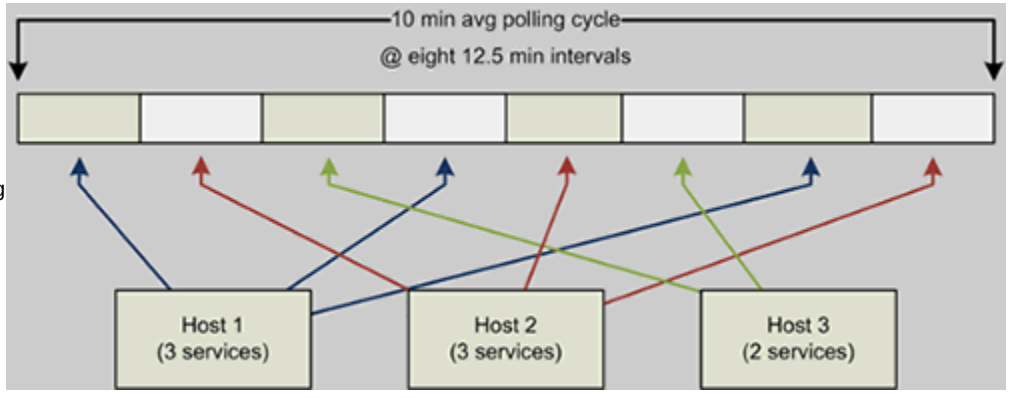
In GroundWork Monitor, almost all of the regular data-collection duties are carried out by *Nagios*. Although GroundWork Monitor supports the use of independent and autonomous monitoring tools, and also provides its own development libraries and APIs for custom monitoring solutions, Nagios is almost always the preferred method of collecting monitor data due to the fact that its simple and lightweight textual interface is remarkably easy to work with. As a result, Nagios has a large library of service definitions that are already included in GroundWork Monitor, while many others are available for download from third-party Internet sites. For most needs, it is very likely that a suitable *Nagios* plug-in already exists.

Nagios provides three crucial functions to the data-collection phase of the overall information flow. First, it provides a command scheduler that intelligently coordinates plug-in processing across all of the defined resources. The second critical function is the execution of the raw plug-in commands, which is the where the monitoring data is actually gathered. Finally, *Nagios* examines the plug-in response data, and takes additional action based on the response that was received. These three functions are completed in linear fashion, with the cycle continuously repeating for as long as a resource definition exists.

2.5 The Nagios Command Scheduler

Whenever the *Nagios* service is restarted, it takes a survey of all the defined hosts and their associated resources, and then populates a rotating schedule of commands to execute. The formula that is used to create the command schedule can be somewhat arcane, involving multiple algorithms and variables, but the basic principle behind the algorithm is that Nagios tries to evenly distribute the rate at which it will execute commands. This helps to ensure that the local server does not get overloaded when a new polling period begins, and it also helps to prevent target servers from being overloaded with complex queries. Without these precautions in place, Nagios could easily become the source of problems that it subsequently believes to have detected.

One of the major parts of this formula involves distributing all of the service definitions across the available time window in an even fashion. Essentially, this is achieved by dividing the average polling interval by the number of resource definitions in the command queue, which yields an evenly spaced set of execution intervals.



In addition, the *Nagios* command scheduler also interleaves queries on a per-host basis so that the target systems do not get hit with all their queries simultaneously. This is achieved by dividing the number of resource definitions by the number of active host definitions, with the resulting value being used as a "skip" value. *Nagios* begins the scheduling process by adding the first command to the first opening in the schedule window, then skips ahead in the command queue by the determined "skip value" and adds the next command to the current slot in the schedule. This process is continually repeated until all of the commands have been added to all of the slots in the command schedule.

Taken together, these techniques allow the *Nagios* scheduler to issue commands at a reasonably low rate, while also spreading the query volume across all of the servers in a reasonably even manner. This concept is illustrated in the diagram below, which shows how eight discrete monitoring tasks for three different hosts would theoretically be distributed across a 10-minute polling window. In that example, the 10-minute window is broken into eight 1.25-minute intervals, with a skip value of "3". The first command from the first host is placed into the first available slot, then the scheduler moves three places and adds the second command from the first host to that slot, and so forth, with this process repeating until all of the commands have been added to the scheduler.

Although the above explanation covers the most important factors in determining how commands are scheduled, it's important to recognize that there are many other variables and options that are taken into consideration when the *Nagios* command queue is being filled, and the two mechanisms described above are by no means the only criteria (for a detailed description of the full algorithm and all of its variables, please refer to Nagios documentation.

Furthermore, sometimes an external event can also affect how a command is processed, regardless of how the command was originally scheduled. For example, every host definition has a special service definition that is used to determine whether or not the host is online. If this test indicates that the target host has become unreachable, then other service checks for that host may be suspended, resulting in the resource-specific commands being skipped at their next scheduled interval.

It is also possible to have more resource definitions than the Nagios server can monitor in the allowed polling interval, which may result in scheduled commands being deferred or skipped. Generally speaking, the maximum number of commands that can be executed within the available polling window will vary according to CPU and memory availability on the local and remote hosts, network bandwidth and latency, the complexity of the command, the length of time required to fully process each command, and other such factors. As a rule, Nagios will always attempt to execute as many monitoring jobs as it can, but if you consume all of your available computing resources with too many probes, the remaining jobs will be rescheduled, potentially resulting in incomplete monitoring runs.

2.6 The Nagios Command Processor

The *Nagios* command interface is the central point for the collection of monitoring data. It is the facility where monitoring commands are executed and response data is read and captured. This is perhaps the most important subsystem in the entire GroundWork Monitor collection, given that it is where all of the data collected by Nagios is initially obtained.

The *Nagios* command interface is remarkably simple, especially considering its importance. Essentially, *Nagios* does nothing more than expand whatever variables have been defined for the service definition at hand, submits the full command line to the operating system for processing, and then reads the response output.

As was discussed earlier, resource monitoring in Nagios uses command definitions that are mapped to service definitions, which are in turn mapped to a specific resource on a specific host, with variable values in the underlying command-line being filled in by the service definition or the resource-specific definition, as needed. As this relates to the *Nagios* command interface, the command-line that is used by *Nagios* is sourced from the final resource definition, since that is the last point where variable substitution can occur.

For general status measurement, *Nagios* uses four different response codes: "OK" responses indicate that the monitored resource is operating within its normal bounds, while a "WARNING" response means that the resource is operating outside normal bounds but is not in a crisis condition (yet), while "CRITICAL" means that the resource is either not operating or has reached a crisis condition, and "UNKNOWN" means that some other condition has occurred that the command does not know how to interpret. The status responses are primarily provided as POSIX return codes through STDERR, with "0" meaning "OK", "1" meaning "WARNING", "2" meaning "CRITICAL", and "3" meaning "UNKNOWN".

However, *Nagios* also expects textual response data to appear in STDOUT, with this data typically providing the textual equivalent of the status response codes along with any error text that may be needed for alerting and notification purposes, as well as any measurement data that may be needed for performance monitoring.

Historically, *Nagios* has used a relatively free-form textual response, although most plug-ins use a format similar to "<SERVICE> <STATUS> - <Textual message>", where "<SERVICE>" is the service name, "<STATUS>" is a textual representation of the return codes described above, and "<Textual message>" is the free-form response message.

With the addition of performance monitoring, however, the response format was specifically amended to include structured data. In particular, plug-ins that return performance data are now required to append a vertical pipe symbol ("|") to the end of the free-form text message, with detailed performance data following the pipe character. The performance data is required to take the exact form of "<label>=<value>;<warning-threshold>;<critical-threshold>;<minimum-value>;<maximum-value>." This syntax provides all the information that is needed for a command to determine the current performance of a resource, relative to that specific resource. In those cases where a particular field is not appropriate, the data is omitted, but the semi-colon separators must still be provided. For more details on the response format rules, refer to Nagios plug-in developer's guidelines.

2.7 The Nagios Passive Interface

Most service definitions use command-based probes to actively query a specific resource, but in some cases it can be preferable or even necessary to allow external probes and tools to bring the monitoring data to *Nagios* instead. This is especially useful with services that can generate spurious messages at somewhat random intervals, since those types of events cannot be easily monitored through *Nagios'* poll-based scheduler.

For example, GroundWork Monitor provides tools that can pickup SNMP traps and SYSLOG messages independently of *Nagios*, and then use the passive interface to feed the event message into Nagios for additional processing (such as generating notification messages or event alarms). Meanwhile, *Nagios* also provides a network protocol called NSCA which allows remote systems to submit monitor results directly to Nagios, which in turn helps to alleviate load on the local server.

In these cases, external system processes queue up incoming messages for *Nagios* to process later, while the *Nagios* command scheduler also queues up "passive" checks that are executed in the same manner as regular commands. Whenever the passive check for a specific resource rolls around, *Nagios* reads the associated input, and then treats the discovered data as if it had been obtained from a "normal" command, with the data being essentially indistinguishable from that point forward.

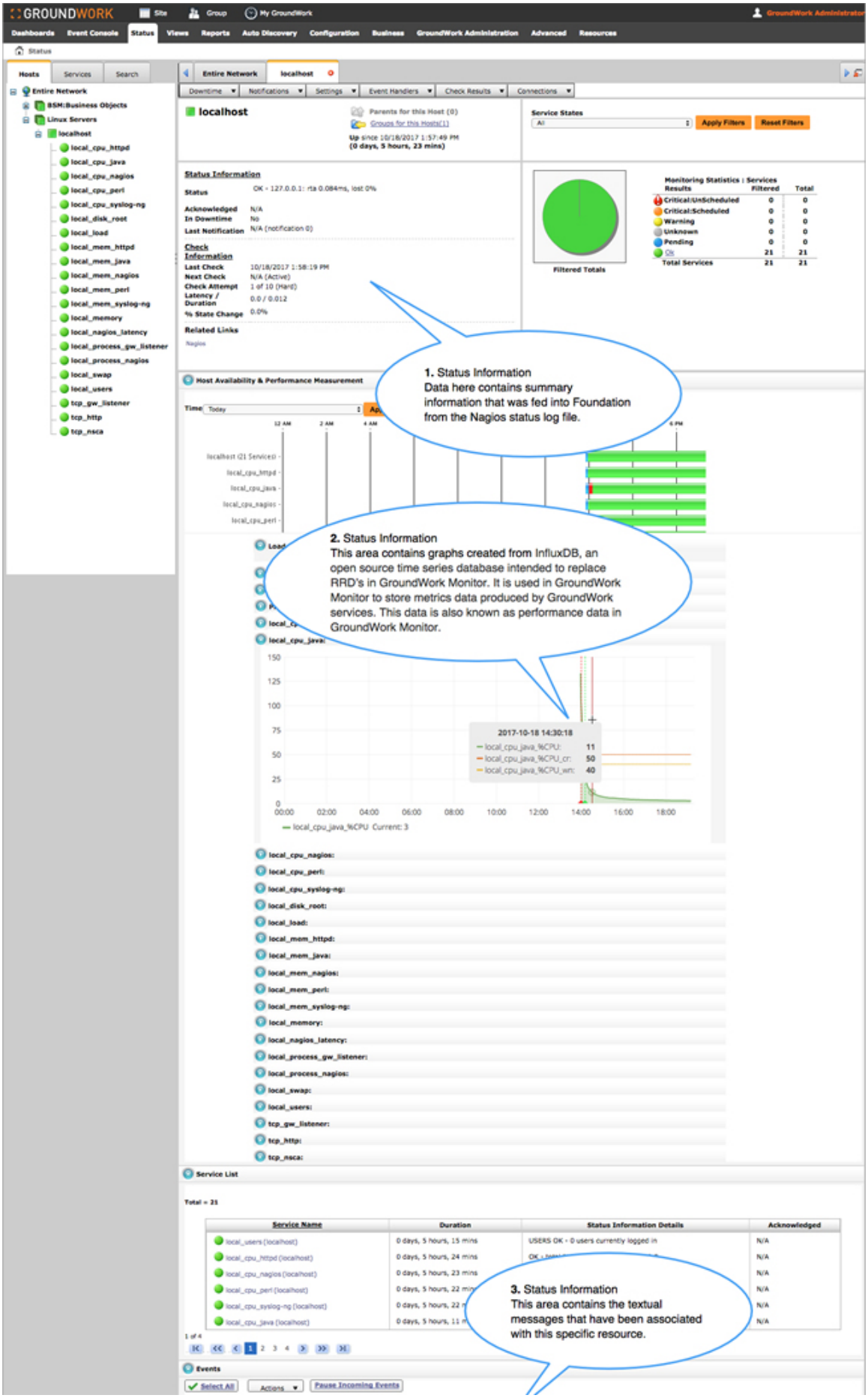
2.8 Information Processing

Although *Nagios* collects response data as each command completes, it does not begin to process the command output right away, but instead places the response data into a global event queue for processing at the next available command interval (which may happen immediately, or may happen later if *Nagios* is busy on other tasks). Once an available command window rolls around, *Nagios* starts a "reaper" event that gathers up all of the output data that has been queued, and then parses through each of the recorded results.

As part of this process, *Nagios* first records details about the commands that were executed for each resource, along with the output that was returned from each of the commands. This logfile is subsequently read by the *GroundWork* integration scripts, which in turn updates the appropriate *Foundation* database tables; the status codes are stored in a basic state-tracking table that is used to monitor basic availability, while the free-form response text is stored in a log table that provides a textual view of historical events.

Separately, *Nagios* also records the collected performance data into another log file so that it can be used to build long-term performance databases, if they have been requested. Every five minutes, the Nagios event processor executes a Perl script supplied with GroundWork Monitor that creates and appends the resource-specific round-robin databases with the collected performance data,

which in turn allows the



historical sampling data to be graphed by presentation tools.

Cumulatively, these repositories can provide a substantial amount of information about a particular resource. For example, the *Status* application in GroundWork Monitor uses all of the collected data to display summary information about a specific resource, including its current status, the major event messages which have been received, and the long-term performance readings. This concept is illustrated below and shows a typical view of the *Status* application, and also indicates the database sources that are used to build the web page. Data in block number **(1)** contains summary information that was fed into *Foundation* from the *Nagios* status log file, while data in block **(2)** contains a graph that is created from the round-robin performance databases (which are themselves sourced from the Nagios performance logs), and data in block **(3)** contains the textual messages that have been associated with this specific resource.

2.9 State Tracking

Apart from recording output messages into the various log files, the *Nagios* "reaper" process is also responsible for examining the response status, taking any action that may be required (such as generating an alarm), and then resubmitting the monitoring tasks to the command scheduler. Essentially, this portion of the output process is where *Nagios* determines if a resource is up or down, and if some kind of additional processing needs to occur.

The principle piece of information in this process is the return code from the plug-in that was used to query the resource in question. Resource definitions that return an "OK" status are simply noted, and their associated commands are resubmitted to the scheduler for execution at the next available interval. However, if a service returns a WARNING or CRITICAL status result, then Nagios will queue a request to probe the host itself to make sure that the target device itself is up and reachable.

If the host check for a failing service also returns a non-OK result, then Nagios assumes that the host is down, and takes whatever action has been specified for the target host as a whole. However, if the host check succeeds, then the specific resource is assumed to be in a failing state, and *Nagios* takes whatever action has been specified for that specific resource.

Services that return a non-OK status are determined to be in a "soft" error state until a specified number of attempts have been made, at which point the service will be flagged as being in a "hard" error state. Services that are flagged in either of these states can use event handlers to attempt some kind of automated recovery (event handlers are usually defined in the service and host templates, but can also be assigned globally if desired). When the service reaches a hard error state, notification messages will be issued to the appropriate contact.

2.10 The GroundWork Foundation Database

Although GroundWork Monitor does not replicate the entire Nagios status log locally, GroundWork Monitor do copy some of the event data into an embedded database so that administrators have quick and easy access to the major events that have been recorded.

This data is primarily used by the GroundWork Monitor *Event Console* application, shown in the diagram below, but it is also incorporated into some other presentation tools as well (most notably the *Status* viewer application).

The number of records that are created and stored in the *GroundWork Foundation* database is determined by filters that the administrator defines. By default, the database only stores major events (such as major state changes), but it can capture as much information as the administrator desires. Likewise, critical information from the SYSLOG and SNMP Trap tools that are bundled into commercial versions of GroundWork Monitor can also be extracted and stored into the console table. In fact, the *Foundation* database can be extended to receive and store event data from practically any event source.

Event records in the *Event Console* application can be "cleared" when they have been acknowledged and dealt with by an operator, so that old records do not clutter the application view. However, the underlying events will remain in the database until they are manually purged, thereby allowing for long-term reporting and analysis to function properly.

Another critical aspect of this service comes from the way that the integration scripts can automatically eliminate duplicate records, which can help to reduce database overhead. For example, if two events occur that are considered identical, the database will simply update the "Last Inserted" field of the first entry with the timestamp from the last entry, and then increment the "Message Count" field to show that multiple instances of the same response have been detected. This algorithm can be changed if administrators require a different level of granularity, or if they need to specify different criteria for identifying identical messages.

2.11 Performance Data

In GroundWork Monitor, long-term storage of performance data is handled by the

	Received By GW	Message Count	Host	Service	Status	Message	Application Type	Severity	Last Detected	First Detected	Acknowledgedby	Comments
	10/18/2017 2:10:18 PM	1	localhost	local_cpu_java	OK	OK - total %CPU for process java : 26.2	NAGIOS	OK	10/18/2017 2:10:18 PM	10/18/2017 2:10:18 PM		
	10/18/2017 2:10:04 PM	1	bsm-host	bsm-service-01	OK	OK - 0 problem(s) of 6 member(s).	BSM	LOW	10/18/2017 2:10:04 PM	10/18/2017 2:10:04 PM		
	10/18/2017 2:07:39 PM	1	localhost	tcp_nsca	OK	TCP OK - 0.001 second response time on	NAGIOS	OK	10/18/2017 2:07:21 PM	10/18/2017 2:07:21 PM		
	10/18/2017 2:07:09 PM	1	localhost	tcp_http	OK	HTTP OK: HTTP/1.1 200 OK - 1296 bytes i....	NAGIOS	OK	10/18/2017 2:06:52 PM	10/18/2017 2:06:52 PM		
	10/18/2017 2:06:39 PM	1	localhost	tcp_gw_listener	OK	TCP OK - 0.001 second response time on	NAGIOS	OK	10/18/2017 2:06:23 PM	10/18/2017 2:06:23 PM		
	10/18/2017 2:06:09 PM	1	localhost	local_users	OK	USERS OK - 1 users currently logged in	NAGIOS	OK	10/18/2017 2:05:55 PM	10/18/2017 2:05:55 PM		
	10/18/2017 2:05:39 PM	1	localhost	local_swap	OK	SWAP OK - 100% free (4911 MB out of 491....	NAGIOS	OK	10/18/2017 2:05:26 PM	10/18/2017 2:05:26 PM		
	10/18/2017 2:05:09 PM	1	localhost	local_process_...	OK	NAGIOS OK: 6 processes, status log upda....	NAGIOS	OK	10/18/2017 2:04:58 PM	10/18/2017 2:04:58 PM		
	10/18/2017 2:04:39 PM	1	localhost	local_process_...	OK	PROCS OK: 1 process with args 'groundwo....	NAGIOS	OK	10/18/2017 2:04:29 PM	10/18/2017 2:04:29 PM		
	10/18/2017 2:04:09 PM	1	localhost	local_nagios_l...	OK	OK: Nagios latency: Min=0.000, Max=0.00....	NAGIOS	OK	10/18/2017 2:04:01 PM	10/18/2017 2:04:01 PM		
	10/18/2017 2:03:38 PM	1	localhost	local_memory	OK	Memory OK - 21.0% (3462656 kB) used	NAGIOS	OK	10/18/2017 2:03:32 PM	10/18/2017 2:03:32 PM		
	10/18/2017 2:03:08 PM	1	localhost	local_mem_syst...	OK	OK - total %MEM for process syslog-ng :....	NAGIOS	OK	10/18/2017 2:03:03 PM	10/18/2017 2:03:03 PM		
	10/18/2017 2:02:53 PM	1	localhost	local_mem_perl	OK	OK - total %MEM for process perl : 3.6	NAGIOS	OK	10/18/2017 2:02:35 PM	10/18/2017 2:02:35 PM		
	10/18/2017 2:02:23 PM	1	localhost	local_mem_nagi...	OK	OK - total %MEM for process nagios : 0.0	NAGIOS	OK	10/18/2017 2:02:06 PM	10/18/2017 2:02:06 PM		
	10/18/2017 2:01:53 PM	1	localhost	local_mem_java	OK	OK - total %MEM for process java : 10.5	NAGIOS	OK	10/18/2017 2:01:38 PM	10/18/2017 2:01:38 PM		
	10/18/2017 2:01:23 PM	1	localhost	local_mem_httpd	OK	OK - total %MEM for process httpd : 0.0	NAGIOS	OK	10/18/2017 2:01:09 PM	10/18/2017 2:01:09 PM		
	10/18/2017 2:00:53 PM	1	localhost	local_load	OK	OK - load average: 2.03, 1.76, 0.98	NAGIOS	OK	10/18/2017 2:00:41 PM	10/18/2017 2:00:41 PM		
	10/18/2017 2:00:23 PM	1	localhost	local_disk_root	OK	DISK OK - free space: / 35747 MB (87% i....	NAGIOS	OK	10/18/2017 2:00:12 PM	10/18/2017 2:00:12 PM		
	10/18/2017 2:00:18 PM	1	localhost	local_cpu_java	CRITICAL	CRITICAL - total %CPU for process java	NAGIOS	CRITICAL	10/18/2017 2:00:18 PM	10/18/2017 2:00:18 PM		
	10/18/2017 2:00:04 PM	1	bsm-host		UP	UP - BSM host running	BSM	LOW	10/18/2017 2:00:04 PM	10/18/2017 2:00:04 PM		

open source RRD toolkit, which provides "round-robin" databases and graphing functions that cumulatively allow for time-based views of historical readings. Essentially, the RRD toolkit uses fixed-length database files with a first-in, first-out rotation policy, and also uses a moving-average algorithm that consolidates and removes older data as new samples get added. As a result, short-term readings are always available in their raw, high-precision form, while older readings have a lower resolution as a result of the long-term averaging algorithm. This model allows a single database file to provide snapshot views at multiple time intervals, ranging from one minute to one year, while the underlying database files always remain at a fixed size.

The RRD databases are seeded by a Perl script provided with GroundWork Monitor that is executed by *Nagios*' job scheduler every few minutes. Whenever performance data for a monitored resource is discovered in the log file, the Perl script appends the data to the appropriate RRD database, or creates the database if it does not exist. However, it is also important to note that the Perl script provided with GroundWork Monitor does not explicitly require *Nagios* performance readings, and can instead infer performance data from other information if needed. For example, it can examine the time required to establish a connection with the remote target, and create a graph based on that data, instead of having to rely on the Nagios-specific performance readings.

In order for the Perl script to know which databases should be created and how the data should be captured and organized, GroundWork Monitor provides a separate *Performance* configuration option within the *Configuration* application solely for this purpose. This application allows the administrator to specify the types of services that need to have performance databases, how to map plug-in output to the RRD database files, how to parse the performance data, and how to update the RRD databases with new readings.

By default, the performance configuration system contains several pre-defined settings for frequently used service definitions, such as CPU and memory utilization, network interface traffic, system load, and other common metrics. If additional performance graphs for other resources are desired, they must be defined in the performance configuration tool before the RRD databases can be created, appended or graphed.

Another noteworthy aspect of this area is that RRD databases can store multiple readings in separate columns within a single file, if needed. This allows for complex graphs that refer to multiple types of readings (such as a network interface graph that shows the amount of traffic sent and received in a single view), but with the caveat that certain changes to the columns may require a new database to be created.