Nagios – Writing a Custom Worker

The Industry Standard in IT Infrastructure Monitoring

Purpose

This document describes the high-level steps for writing a custom Nagios Core 4 worker. Nagios Core workers are a new concept to Nagios Core 4 and there may be a situation when you would want to write a worker to preform a specialized task.

Target Audience

This document is intended for use by Nagios Core and Nagios XI Administrators who want to extend the functionality of Nagios Core by writing custom workers.

Overview

Nagios Core workers provide a service to Nagios Core to perform certain tasks. These tasks include running checks and event handlers as well as handling external commands. Nagios Core is bundled with a generic core worker that will perform any of the previously mentioned tasks.

There may be times when you want a special-purpose Core worker to perform specialized tasks or perform a task more efficiently than the generic core worker. Examples of special-purpose Core workers might include a worker that performs SNMP checks or one that runs Perl-based checks by starting a persistent Perl interpreter as part of the worker, similar to the functionality previously provided by embedded Perl in Nagios Core.

Worker Startup

When a worker starts up it must tell Nagios Core that it is available to perform work. It does so by registering with Nagios Core through the query handler interface. The query handler interface uses a Unix domain socket. The default location for this socket is `/usr/local/nagios/var/rw/nagios.qh`, but the location can be changed in the main configuration file, `nagios.cfg`. The registration takes the form

```
@wproc register name=<name>;pid=<pid>[;max_jobs=<maxjobs>][;plugin=<plugin>]
```

where

- `<name>` is any string up to 128 characters that uniquely identifies the worker process. If you start multiple workers with the same executable, then each must have a unique name. One way of handling this is to include the PID in the worker name.
- `<pid>` is the worker process PID.
- `<maxjobs>` is an optional parameter indicating the maximum number of jobs the worker can process
- `<plugin>` is an optional parameter containing either just the executable name or the full path to the executable of the plugin that this worker will handle. Multiple `plugin=<plugin>` parameters may be passed.

Core to Worker Communication

When Nagios Core has a task to perform, it checks to see which workers are able to perform the task based on the number of jobs a worker is currently handling and its maximum number of jobs, as well as the plugins it has indicated it can run. When the worker is chosen, Nagios Core sends it a command in the form:

```
job_id=<id>\0type=<type>\0command=<command>\0timeout=<timeout>\0\1\0\0
```

where

- `<id>` is a ID provided by Nagios Core to identify the job and which will be sent back in the response. The value of `<id>` should be of no interest to the worker.

- `<type>` is provided by Nagios Core to identify the job type and which will be sent back in the response. The value of `<type>` should be of no interest to the worker.
- `<command>` is the command that the worker is to execute
- `<timeout>` is the timeout for executing `<command>`.

When the worker has completed its job, it will send the response back in the form:

```
job_id=<id>\0type=<type>\0start=<start>\0stop=<stop>\0runtime=<runtime>\0outstd=<stdout>\0out
err=<stderr>\0exited_ok=<exitflag>\0wait_status=<waitstatus>\0\1\0\0
```

where

- `<id>` is the job ID passed in the job request
- `<type>` is the job type passed in the job request
- `<start>` is the time the job was started in the form `<sec>.<usec>` where `<sec>` is the number of seconds since the beginning of the Unix epoch and `<usec>` is microseconds
- `<stop>` is the time the job was completed in the same form as `<start>`
- `<runtime>` is the amount of time the job ran in the format `<sec>.<usec>`
- `<stdout>` is the output of the command sent to standard out
- `<stderr>` is the output of the command sent to stderr
- `<exitflag>` is either 0 or 1, 1 indicating the job exited OK, 0 indicating that it did not exit OK.
- `<waitstatus>` is the integer status set by the `wait*()` system call used to execute the command

In the case of an error, the following information should be returned

- `error_msg` - An error message generated by the worker process
- `error_code` - The error code generated by the worker process

For more information about job responses, see the Doxygen generated documentation as described in Additional Information.

Worker Development

Workers can be developed in any programming language. If a worker is developed in C, the libnagios library should be used because it provides a number of components that are designed to work with workers and, in fact, are used with the standard worker provided with Nagios Core.

Additional Information

Doxygen generated documentation is available with the Nagios Core distribution. To generate the documentation, run 'make dox' from the root directory of the Nagos Core distribution. The HTML formatted documentation will be created in the Documentation subdirectory.

For an overview of Unix domain sockets, visit http://beej.us/guide/bgnet/output/html/multipage/theory.html.

A simple sample worker can be found in workers/ping under the root directory of the Nagios Core distribution. This worker registers itself as being able to handle check_ping requests.

The code for the standard worker is also very useful for understanding how workers work. Most of the applicable code is found in base/workers.c