NMIS 9 Administration Notes

Overview of the Major Components of NMIS9

The NMIS9 daemon bin/nmisd

In NMIS9 almost all work is controlled, scheduled and executed by the nmis daemon and its worker child processes.

The nmis daemon is controllable using the typical `service` interface with the service name being "nmis9d"; e.g. `sudo service nmis9d restart` .
The daemon should be running by the end of the initial NMIS9 installation.

The primary CLI took bin/nmis-cli

The nmis-cli tool is your primary tool to interact NMIS on the command line; e.g. for querying the status of the nmis daemons, for scheduling new operations and for scheduling outages.
Besides these administrative duties the cli tool is currently the only entity that can create saved reports (which is scheduled using a minimal NMIS 9 cron job).

The Node administration CLI tool admin/node_admin.pl

Like with NMIS8, in NMIS9 nodes can be administered using the GUI or with the node_admin cli tool. NMIS9's version has a few extra features over NMIS8's but otherwise doesn't differ excessively.
The node admin tool is described in more detail on the Node Administration Tools page.

Because of its reliance on a database NMIS9 is more strict about identifying objects, which means that nodes for example are identified exclusively by UUIDs. Node names are of course still present, but as informal properties only. The relationship between these is queried most easily by the node_admin tool using the `act=list_uuid` operation.

The GUI

The adminstrative capabilities of the NMIS9 GUI are almost identical to how NMIS8 worked; the only major exception being that "Edit and Update Node" cannot display any logs of the Node Update operation as that's scheduled asynchronously. The NMIS9 GUI plays a slightly more passive and and limited role, i.e. only schedules certain operations for the nmis daemon to pick up - different from NMIS8 where some of these were executed directly by GUI components.

The Database

NMIS9 makes extensive use of MongoDB behind the scenes; most of the time that should be invisible to you past the initial installation stage, where you will have to interact with `setup_mongodb.pl` to prime the environment.

NMIS9 is much more powerful than NMIS8 when it comes to clustering; amongst other things that also means that each NMIS9 installation has to be uniquely identified by what we call its `cluster_id` configuration setting (which is automatically generated for you during the initial installation).

Interacting with the daemon directly

The NMIS9 daemon accepts just a small number of command line arguments, which are shown when you run it with `-h` or `--help` :

```
./bin/nmisd -?
Usage: nmisd [option=value...] [act=command]

 act=version: print version of this daemon and exit
 act=stop: shut down running daemon and workers
 act=abort: terminate all workers and kill running daemon

if no act argument is present: daemon starts

option foreground=1: stay in the foreground, don't daemonize
option max_workers=N: overrides the configuration
option debug=0/1: print extra debug information

option confdir=path: path to configuration files
```

The most commonly used ones would be `act=stop` and `act=abort`:

- With `stop` you're instructing a running nmis daemon and all its workers to terminate gracefully, i.e. when any operations that were in progress are completed.
- With `abort` a running nmis daemon and its workers are stopped immediately and without regard to operations that are in progress.

In both of these cases no new nmis daemon is started.

Job Scheduling in NMIS9

In NMIS9 the nmis daemon controls the scheduling of all work based on various heuristics and manages a queue of these jobs; the nmis daemon's worker processes then pick and process jobs from the queue. Normally all job scheduling is automatic but it is possible to manually schedule activities using the nmis cli.

All enqueued jobs have a target execution time and a priority value.

The nmis daemon normally does not schedule another instance of a particular job, if that job is already active or overdue for processing. In such a case you'll see a log message warning about this issue.

If two or more already scheduled jobs should interfere with each other (e.g. a manually scheduled job for the same operation on one node where another job with the same parameters is already active), then the nmis daemon either discards the new job or postpones the new job for a short period to let the active job finish: the configuration item `postpone_clashing_schedule` sets the number of seconds to postpone. In both cases a log message will warn you about the unexpected clash.

Priorities

Each job instance is given a priority value (between 0 and 1 inclusive, 1 meaning highest priority), and the queue processing takes these into account. Jobs ready for processing are selected first by highest priority, then by scheduled job execution time (i.e. with equal priority the most overdue job is picked first).

The normal priorities are configured in `Config.nmis` in the `priority_schedule` section, with these defaults:

```
        "priority_schedule" => {
                "priority_escalations" => 0.9,
                "priority_collect" => 0.85,
                "priority_update" => 0.8,
                "priority_plugins" => 0.85, # post-update and post-collect plugins
                "priority_services" => 0.75,
                "priority_thresholds" => 0.7,
                "priority_metrics" => 0.7,
                "priority_configbackup" => 0.3,
                "priority_purge" => 0.3,
                "priority_dbcleanup" => 0.3,
                "priority_selftest" => 0.2,
                "priority_permission_test" => 0.1,
        },
```

If you schedule a job manually then you can give it a priority value of your choice; if you don't then nmis-cli defaults to `job.priority=1` (i.e. highest).

Periodically Scheduled Jobs

The nmis daemon automatically schedules various activities periodically, based on global configuration settings. This overview is part of nmis-cli's schedule listing output:

```
Operation                    Frequency
Escalations                  1m30s
Metrics Computation          2m
Configuration Backup         1d
Old File Purging             1h
Database Cleanup             1d
Selftest                     15m
File Permission Test         2h
```

The configuration items controlling these activities' scheduling frequencies are grouped in the `schedule` section of `Config.nmis`, with these defaults:

```
        'schedule' => {
                # empty, 0 or negative to disable automatic scheduling
                'schedule_configbackup' => 86400,
                'schedule_purge' => 3600,
                'schedule_dbcleanup' => 86400,
                'schedule_selftest' => 15*60,
                'schedule_permission_test' => 2*3600,
                'schedule_escalations' => 90,
                'schedule_metrics' => 120,
                'schedule_thresholds' => 120, # ignored if global_threshold is false or threshold_poll_node is
true
        },
```

If you want to manually schedule one of these with nmis-cli, use the suffix after `schedule_` as the job type, e.g. `permission_test` for the extended selftest.

Node Activity Scheduling

The node-centric actions (e.g. collect, update) are scheduled based on the node's last activity timestamps and its polling policy, which works the same as in NMIS8. Service checks are scheduled based on the service's period definition, again mostly unchanged from NMIS8.

When the Updates and Collects last occurred can be found using:

Fault-recovery

If a job remains stuck as active job for too long then the nmis daemon will abort it and reschedule a suitable new job. Such stuck jobs can appear in the queue if you terminate the nmis daemon with `act=abort` or `service nmis9d stop`, because these actions immediately kill the relevant processes and don't take active operations into account.

When and whether NMIS should attempt to recover from stuck jobs is configurable, in `Config.nmis` under `overtime_schedule`, with these defaults:

```
"overtime_schedule" => {
                # empty, 0 or negative to not abort stuck overtime jobs
                "abort_collect_after" => 900, # seconds
                "abort_update_after" => 7200,
                "abort_services_after" => 900,
                "abort_configbackup_after" => 900, # seconds
                'abort_purge_after' => 600,
                'abort_dbcleanup_after' => 600,
                'abort_selftest_after' => 120,
                'abort_permission_test_after' => 240,
                'abort_escalations_after' => 300,
                'abort_metrics_after' => 300,
                'abort_thresholds_after' => 300,
        },
```

NMIS also warns about unexpected queue states, e.g. if there are too many overdue queued jobs or if there are excessively many queued jobs altogether.

Parameters to prevent the queue getting too big

When the server has limited resources and cannot process the jobs in time, there is a risk of the jobs getting stacked in the queue. One of the symptoms we can observe in the logs:

```
Performance warning: N overdue queued jobs!
```

There are two configuration parameters that can help and can be set in Config.nmis:

- There was no default **abort_plugins_after** option in the configuration. This value can be added in Config.nmis:

  ```
  'overtime_schedule' => {
      'abort_plugins_after' => 7200, # Seconds
     ...
  }
  ```

- The schedule keeps adding these jobs into the queue. The workers can discard these jobs changing the configuration options postpone_clashing_schedule to 0.
  `'postpone_clashing_schedule' => 0,`

After theses two changes, nmis9d daemon needs to be restarted.

Interacting with the daemon using nmis-cli

Just like all other NMIS9 command line tools nmis-cli shows an overview of its arguments and capabilities when you run it with `-h` or `--help` (or without any arguments whatsoever):

```
./bin/nmis-cli
Usage: nmis-cli [option=value...] <act=command>

 act=fixperms
 act=config-backup
 act=noderefresh
 act=daemon-status (or act=status)

 act=schedule [at=time] <job.type=activity> [job.priority=0..1] [job.X=....]
  act=schedule-help for more detailed help
 act=list-schedules [verbose=t/f] [only=active|queued] [job.X=...]
 act=delete-schedule id=<schedule_id|ALL> [job.X=...]
 act=abort id=<schedule_id>

 act=purge [simulate=t/f] [info=t/f]
 act=dbcleanup [simulate=t/f] [info=t/f]

 act=run-reports period=<day|week|month> type=<all|times|health|top10|outage|response|avail|port>

 act=list-outages [filter=X...]
 act=create-outage [outage.A=B... outage.X.Y=Z...]
 act=update-outage id=<outid> [outage.A=B... outage.X.Y=Z...]
 act={delete-outage|show-outage} id=<outid>
 act=check-outages [node=X|uuid=Y] [time=T]
  act=outage-help for more detailed help
```

Process Status

To find out what processes are running and doing what, use `act=status` or `act=process-status` ; it'll provide you with an overview like the following example:

```
./bin/nmis-cli act=status
PID             Daemon Role
24084           nmisd scheduler
24103           nmisd fping
24109           nmisd worker services nodeOne
24111           nmisd worker <idle>
24113           nmisd worker collect nodeSeven
24115           nmisd worker <idle>
```

Normally you should have one "nmisd scheduler" process, one "nmisd fping" worker and a few workers. The default configuration (see config item `nmisd_max_workers`) is to start up and maintain 10 workers. In the example above two of these are idle and two are currently processing particular jobs. Please take note of the process id or PID; both are relevant for logging (e.g. finding particulars in the log file as well as adjusting the logging verbosity).

Queue Status

It is often useful to find out what activities are currently being performed and what and how much work is enqueued for future processing.
nmis-cli shows this information when run with the argument `act=list-schedules`, like this:

```
./bin/nmis-cli act=list-schedules
Active Jobs:
Id                      When                    Status
What                Parameters
5d3a483ec6c2b15e1411a7df  Fri Jul 26 10:24:30 2019  In Progress since Fri Jul 26 10:24:30 2019
collect             <skipped, too long>
5d3a483ec6c2b15e1411a7e1  Fri Jul 26 10:24:30 2019  In Progress since Fri Jul 26 10:24:30 2019
collect             <skipped, too long>
...

Queued Jobs:
Id                      When                    Priority    What                Parameters
No queued jobs at this time.
```

In this example, two jobs are in progress, and no jobs are queued, awaiting processing. Because jobs may have substantial amounts of job parameters, the display omits these parameters unless you add the option `verbose=1` to the nmis-cli invocation. With verbose, you'll see a result like this:

```
./bin/nmis-cli act=list-schedules verbose=1
Active Jobs:
Id                      When                    Status
What                Parameters
5d3a48fc0a6b3126df1a1a55  Fri Jul 26 10:27:40 2019  In Progress since Fri Jul 26 10:27:40 2019 (Worker 2511)
collect             {'force'=1,'uuid'='286d04c7-149c-4b47-9697-75cf927f3ade','wantsnmp'=1,'wantwmi'=1}
...
```

The important aspects of this verbose display are the 'uuid', which uniquely identifies the node in question for this particular collect operation, and the job 'Id' which is visible in the logs and can be used to abort a job if problems arise.

How to delete Queued Jobs or abort Active Jobs

You can remove queued jobs individually or wholesale using the `act=delete-schedule` option of nmis-cli; either pass in the job's Id, (e.g. `id=5d3a48fc0a6b3126df1a1a55`) or use the argument `id=ALL` with optional further job property filters (e.g. `job.type=services job.uuid=<somenodeuuid>`) to delete just the matching jobs.

A similar operation is possible for aborting active jobs, but please be aware of possible negative consequences: if you abort an active job with `act=abort`, then the worker process handling that job is forcibly terminated immediately which may result in data corruption.

Manual Scheduling of Jobs

The nmis cli can be used to create new job schedules manually, and the expected arguments for queue management are shown when you run nmis-cli with `act=schedule-help` (or `act=schedule` without any further parameters):

```
./bin/nmis-cli act=schedule-help
...
Supported Arguments for Schedule Creation:

at: optional time argument for the job to commence, default is now.

job.type: job type, required, one of: collect update services
  thresholds escalations metrics configbackup purge dbcleanup
  selftest permission_test or plugins

job.priority: optional number between 0 (lowest) and 1 (highest) job priority.
 default is 1 for manually scheduled jobs

For collect/update/services:
job.node: node name
job.uuid: node uuid
job.group: group name
  All three are optional and can be repeated. If none are given,
  all active nodes are chosen.

For collect:
job.wantsnmp, job.wantwmi: optional, default is 1.

For plugins:
job.phase: required, one of update or collect
job.uuid: required, one or more node uuids to operate on

job.force: optional, if set to 1 certain job types ignore scheduling policies
 and bypass any cached data.
job.verbosity: optional, verbosity level for just this one job.
 must be one of 1..9, debug, info, warn, error or fatal.
job.output: optional,  if given as /path/name_prefix or name_prefix
 then all log output for this job is saved in a separate file.
 path is relative to log directory, and actual file is
 name_prefix-<timestamp>.log.
job.tag: somerandomvalue
 Optional, used for post-operation plugin grouping.
```

For example, if you wanted to schedule a forced `update` operation for one particular node to be performed five minutes from now, you'd use the following invocation:

```
./bin/nmis-cli act=schedule job.type=update at="now + 5 minutes" job.node=testnode job.force=1
Job 5d3a5e2d3feeed1f19c46e55 created for node testnode (6204cd3d-3cc1-4a3a-b91e-e269eb5042a4) and type update.

# or with job.priority, job.verbosity and job.output
bin/nmis-cli act=schedule job.type=update job.priority=1 job.node=testnode job.verbosity=9 job.output=/tmp
/localhost.log job.force=1
Job 5e7d67dec6c2b14bd3679101 created for node testnode (3d994eb5-dcba-46de-bb90-914b5dde822f) and type update.
```

If successful nmis-cli will report the queue Id and the expanded parameters of your new job.

Administrative and Other CLI Operations

- If you edit or transfer NMIS files across machines then some file permissions may change for the worse, and the NMIS9 selftest may alert you about invalid file permissions.
  The fastest way to resolve this is to use the nmis cli with the `act=fixperms` argument.
- The `config-backup` argument instructs nmis-cli to produce a backup of your configuration files right now;
  normally configuration backups are performed automatically and daily.

Logging and Verbosity

Standard Log Files

- `logs/fping.log`: the fping worker process (managed by the nmis daemon) logs all its operations to this log file.
- `logs/auth.log`: contains all authentication-related logging that the NMIS9 GUI produces, in the same format that NMIS8 used.
- `logs/event.log`: contains all nmis node events in a machine-consumable format, identical to NMIS8.
- `logs/nmis.log`: all log data that isn't directed elsewhere goes into this log file.

Please note that in NMIS9 all logs are written to in buffered form: information may arrive on disk a few seconds delayed, but at much less performance cost that NMIS8 incurred.

Log files are now also kept open permanently, until the nmis daemon is instructed to reopen them (by sending a `SIGHUP` signal the the nmis daemon process).

The format of the log files `fping.log` and `nmis.log` has changed:

```
[Thu Jul 25 10:38:09 2019] [info] nmisd[1325] Found 7 nodes due for services operation
```

Now all log messages are prefixed by time tag, severity level and the process name/role and process identifier of the process in question. In the example above the supervisor component of the nmis daemon has logged this informational announcement.

What gets logged?

NMIS9 is able to log a bit more detail than NMIS8, but much more controllable in terms of what to include when.
There are 13 verbosity levels (in increasing order of noisiness): `fatal`, `error` , `warn` , `info` , `debug` (or `debug1`), `debug2`, `debug3` and so on to `debug9` .
All messages with severities `debug1` to `debug9` are logged with the tag "`[debug]`".

When you set a particular verbosity level then all messages of higher verbosity are suppressed; e.g. at level `info` messages of severity `fatal` , `error`, `warn` and `info` are logged but messages belonging to severities `debug1` to `debug9` are suppressed.

1. By default the configuration property `log_level` controls all logging. The default value for this is `info`.
2. If you start the nmis daemon with a `debug=<level>` command line argument, then that will be used for this daemon and its workers.
3. For node-admin and nmis-cli invocations the same `debug=<level>` command line argument is available.
4. A manually scheduled daemon job can have custom `verbosity` and `output` propertes, which control verbosity and target log file for the processing of this job only.
5. All NMIS daemon instances can be instructed to change their verbosity levels on the fly while the processes remain running, by sending particular UNIX signals to those processes.

Per-job verbosity and custom log file

If a job schedule includes the property `job.verbosity=<level>` , then the job will be processed with that verbosity level. At the end of processing the previous verbosity level is restored.

The related but independent property `job.output=<prefixtext>` instructs the NMIS daemon to divert all logging for this one job to a different log file. The log data is saved in the normal `logs` directory, and the file is named `<prefixtext>-<highprecision-timestamp>.log` , e.g. `logs/quicktest-1564031667.44838.log`. When processing completes log output reverts back to the standard log file.

Adjusting process verbosity levels on the fly

All NMIS daemon processes listen for two particular UNIX signals:

- When a daemon process instance receives the `SIGUSR1` signal, it increments its verbosity by one level, e.g. from `warn` to `info`, or from `debug2` to `debug3` .
- When a daemon receives the `SIGUSR2` signal, it decrements its verbosity by one level.

In both cases a message is logged at the new verbosity level, e.g.:

```
[Thu Jul 25 12:05:06 2019] [debug] nmisd[1325] received SIGUSR1, incremented verbosity level to debug, debug to 2
[Thu Jul 25 12:05:34 2019] [info] nmisd[1325] received SIGUSR2, decremented verbosity level to info, debug to 0
```

How to determine which process to signal?

- use `nmis-cli act=status` to see the list of active daemon processes and use `kill` with the correct process id,
- or use a smarter `kill`-replacement like `pkill` and select by full daemon command line,
  e.g. `pkill -ef -USR2 "nmisd fping"`