MONIT

UNIX™ Systems Management

Introduction

monit is a utility for managing and monitoring, processes, files, directories and devices on a Unix system. Monit conducts automatic maintenance and repair and can execute meaningful causal actions in error situations. E.g. monit can start a process if it does not run, restart a process if it does not respond and stop a process if it uses to much resources. You may use monit to monitor files, directories and devices for changes, such as timestamp changes, checksum changes or size changes.

Monit is controlled via an easy to configure control file based on a free-format, token-oriented syntax. Monit logs to syslog or to its own log file and notifies you about error conditions via customisable alert messages. Monit can perform various TCP/IP network checks, protocol checks and can utilize SSL for such checks. Monit provides a HTTP(S) interface and you can use a browser to access the monit server.

MONIT

Installation instructions

1. Download the `monit-x.y(.z).tar.gz` package.
   (x.y.z denotes version numbers, for instance; monit-4.1.tar.gz)

2. **Installing monit**:

```
$ tar zxvf monit-x.y.z.tar.gz
$ cd monit-x.y.z
$ ./configure (use ./configure --help to view available options)
$ make && make install
```

3. By default `monit` is installed in `/usr/local/bin/` and the `monit.1` man-file in `/usr/local/man/man1/` you can change this default location by using the --prefix option to ./configure.

What to monitor?

- You may use monit to monitor dæmon processes or similar programs running on localhost. Monit is particular useful for monitoring dæmon processes, such as those started at system boot time from `/etc/init.d/`. For instance mail servers, print servers, database servers, application servers, http servers and sshd.
- You may also use monit to monitor files, directories and devices on localhost. Monit can monitor these items for changes, such as timestamp changes, checksum changes or size changes. This is also useful for security reasons; you can monitor the md5 checksum of files that should not change.
- You may even use monit to monitor remote hosts. First and foremost monit is an utility for monitoring and mending services on localhost, but if a service depends on a remote service, e.g. a database server or an application server, it can be useful to test the remote host as well.

How to monitor?

- monit is configured and controlled via a control file called `monitrc`. The default location for this file is `~/.monitrc`. If this file does not exist, monit will try *`/etc/monitrc`* and finally `./monitrc`. You may also specify the control file monit should use via the -c command-line switch. For instance,

  ```
  $ monit -c /usr/local/etc/monitrc
  ```

- Before monit is started the first time, you can test the control file for syntax errors:

  ```
  $ monit -t
  $ Control file syntax OK
  ```

  If there was an error, monit will print an error message to the console, including the line number in the control file from where the error was found. For instance (in this case, a wrongly formatted email address):

  ```
  $ monit: parse error 'hauktildeslash.com' at line 114
  ```

- The monit software package comes with a monitrc example file which you can look at and use as a starting point when you write your first control file. There is also a comprehensive example collection at: http://www.tildeslash.com/monit/examples.html

Starting and stopping

Once you have created a monit control file you can start monit from the console, like so:

```
$ monit
```

You can change some configuration directives via command-line switches, but for simplicity it is recommended that you put these in the control file.

If monit was configured to run in dæmon mode (strongly recommended) and all goes well, it will now detach from the terminal and run as a dæmon process. As a dæmon, monit runs in cycles; It monitors services, then goes to sleep for a configured period, then wakes up and start monitoring again in an endless loop until you choose to stop monit by this command:

```
$ monit quit
```

You may instead choose to start monit from `init` via `/etc/inittab`. If the monit dæmon should stop due to an error, the init program will re-spawn a new monit process. Init will also automatically start monit at machine boot time. To learn more on how to set-up monit to run from init, read the monit man file.

The monit control file

The control file consists of a series of service entries and global option statements in a free-format, token-oriented syntax. Comments begin with a # and extend through the end of the line. There are three kinds of tokens in the control file: grammar keywords, numbers and strings. On a semantic level, the control file consists of only two types of entries:

- **Global set entries**
  A global set entry starts with the keyword set and the item to configure.
- **One or more service entries**
  A service entry starts with the keyword check followed by the service type to monitor. A service entry can contain different if-tests to be performed on the service. Some if-tests are particular to a certain service type. For instance, a md5 checksum test may only be used in a check file entry.

A process entry requires a path to the pid file[1] for the program. A file, a directory or a device entry requires the path to the item to monitor. All other statements are optional.

—

[1] A pid file is a file containing a program's process identification number (pid)

MONIT

Global statements

The start of a monit control file may look like this:

```
(1) # Monit control file
(2)
(3) set daemon 120                          # Poll in 2-minute intervals
(4) set logfile syslog facility LOG_daemon # Default facility is LOG_USER
(5) set mailserver mail.foo.bar            # Default smtp server is localhost
(6) set alert sysadm@foo.bar               # Alert system admin on any event
(7) set httpd port 2812 address localhost
(8)     allow localhost
(9)     allow admin:monit
```

Line 1 is a comment. In line 3, monit is setup to start in dæmon mode and check each service in 2 minute cycles (value in seconds). In line 4, monit is told to log error and status messages via the syslog dæmon. Monit needs a SMTP server to send alert messages, and in line 5 we specify the SMTP server monit should use. Line 6 defines sysadm@foo.bar as recipient for any type of event. Line 7 specify that monit should start its own built-in mini-httpd server and bind the server to local host only. If you want the http server to be accessible from other machines, simply omit the address to bind to. Line 8 defines access control to the httpd server; Hosts allowed to connect to the http server should be listed with one or more `allow host` statements.

The allow statement in line 9 is special, it specify a *username:password* pair used for HTTP Basic Authentication - the Browser will pop-up an authentication box asking you to submit a username and a password before you can access the monit server. By reading the documentation you can also learn how you can configure the monit http server to use SSL.

Monitoring a process

```
(1) check process apache with pidfile "/usr/local/apache/logs/httpd.pid"
(2)       start = “/etc/init.d/httpd start"
(3)       stop = "/etc/init.d/httpd stop"
(4)       if failed port 80 and protocol http
(5)          and request "/cgi-bin/printenv" then restart
(6)       if cpu usage is greater than 60 percent for 2 cycles then alert
(7)       if cpu usage > 98% for 5 cycles then restart
(8)       if 2 restarts within 3 cycles then timeout
(9)   alert foo@bar.baz
```

Line 1 starts a process service entry. It contains the keyword `check` and a descriptive name for the process to monitor, in this case apache. Monit also require the process `pidfile` to be stated.

If you want monit to start, stop or restart the process you must submit a start and stop program. Line 2-3 defines those programs for this process. In line 4 we ask monit to open a connection to the apache server and request a document from the server. If this test should fail, monit will restart the apache process.

In line 6-7 the process characteristics are tested and if one of the tests are true an action is executed. In line 8 a timeout is set. The rationale for a timeout is that monit should not waste time trying to start a service if it has problems running stable. Finally in line 9 monit is requested to send alert messages to `foo@bar.baz`. Monit raise alerts if a service error occurs.

Monitoring files

The following example demonstrate how you can use monit to automatically restart apache if the apache configuration file `httpd.conf` was changed:

```
(1) check file httpd.conf with path /usr/local/apache/conf/httpd.conf
(2)       if changed checksum
(3)          then exec "/usr/local/apache/bin/apachectl graceful"
```

In line 1 monit is setup to monitor the apache configuration file. A check file entry requires a descriptive name for the entry and a full path to the file to monitor. In line 2 we test the md5 checksum for the configuration file and if the test fails, (because the content of the file was changed) then in line 3 apache is restarted via an exec statement.

Here is another example where we test the size of the apache log file and conduct a log rotate if the size grows above a specified size.

```
(1) check file access_log with path /var/log/access_log
(2)       if size > 100 Mb
(3)          then exec "/usr/sbin/logrotate -f rotate_apache_now"
```

Normally you will run `logrotate` at night from (e.g.) `crond` but if the log file suddenly should grow very large it may be necessary to do an “emergency” rotate. Another option is simply to ask monit to send an alert if the log file grows very large. In this case, simply replace the exec statement with alert.

Monitoring directories

You can use monit to monitor directories for changes. In this example we test the standard `sbin` directory. If content was added or removed from this directory we raise an alert. For the argument sake, we assume a fairly static system where the content of `sbin` should not change. Of course, monitoring `/sbin` like this, may not be appropriate for your system, <u>but</u> if you want to test a directory on your system for changes this is the test to use:

```
(1) check directory sbin with path /sbin
(2)       if changed timestamp then alert
(3)       alert foo@bar.baz
```

In line 1 monit is setup to monitor the `/sbin` directory. A directory service entry requires a descriptive name for the service and a full path to the directory to monitor. In line 2, the directory's timestamp is tested and if it was changed, monit will raise an alert to the address stated in line 3.

Here we turn the table and monitor a directory that *should* change. This directory is expected to change its content on an hourly basis. If no files were added (or removed) from the directory within an hour (from the time monit was started) then monit will send an alert to the address in line 4. As soon as the condition will recover, monit will send execute consistency checking script and send alert:

```
(1) check directory db_tmp with path /data/tmp
(2)       if timestamp > 1 hour the alert
(3)          else if recovered then exec "/check/my/db"
(4)       alert foo@bar.baz
```

Monitoring a device

Here is an example where monit is used to monitor a CDROM station. Admittedly, it's not a very practical example, but it is fun and involve moving parts.

```
(1) check device CDROM with path /dev/cdrom
(2)       start "/bin/mount /dev/cdrom"
(3)       stop "/bin/bash -c '/bin/umont /dev/cdrom; /usr/bin/eject;'"
```

In line 1, we ask monit to check the cdrom device `/dev/cdrom`. To automatically mount the cdrom if it is not mounted, we add a start method in line 2, informing monit how this device should be started. In line 3 we also add a stop method, specifying the commands for unmounting the device and also ejecting the cdrom from its disk tray. From monit's web interface you can now, for instance, push the start button to mount the cdrom and push the stop button to umount and eject the cdrom.

A more practical example, after all, is to use this feature to monitor a hard disk and raise alerts if the device is about to run out of available space or inodes, such as:

```
(1) check device disk1 with path /dev/hda1
(2)       start = "/bin/mount /dev/hda1"
(3)       stop = "/bin/umont /dev/hda1"
(4)       if space usage > 90% then alert
(5)       if space usage > 99% then stop
(6)       if inode usage > 90% then alert
(7)       if inode usage > 99% then stop
(8)       alert foo@bar.baz
```

Remote host monitoring (1)

You can use monit to monitor and test services running on a remote host.

```
 (1) check host rhn.redhat.com with address rhn.redhat.com
 (2)       if failed port 80 protocol http and request "/help/about.pxt"
 (3)          then alert with the mail-format {subject: RedHat is down again!}
 (4)       if failed port 443 type TCPSSL and protocol http
 (5)          with timeout 15 seconds then alert
 (6)       alert foo@bar.baz
 (7)
 (8) check host up2date with address 66.187.224.51 # This is ftp.redhat.com
 (9)       if failed port 21 protocol ftp
(10)          then exec "/usr/bin/snpp -m
(11)               'Monit: $MONIT_EVENT for $MONIT_SERVICE' rladams"
```

The first entry monitors the remote host `rhn.redhat.com` on two ports; the standard HTTP and HTTPS port numbers using the http protocol test. In line 4 we need to use a SSL socket type to test a secure https server. The name after the host keyword is a descriptive name. You may use any name here, in this example we simply use the same host name we will connect to.

The second entry monitors the remote ftp server `ftp.redhat.com` on the standard ftp port number, i.e. port 21. If this test should fail, instead of sending an SMTP alert, as in the first entry, we choose to execute an external program to handle the notification. (`snpp` is used to send a text message to a Pager Terminal). $MONIT_EVENT and $MONIT_SERVICE (plus a few more) are environment variables available to programs executed from within monit.

Remote host monitoring (2)

Monit also support a ping test you can use to check if a remote host is up:

```
(1) check host tildeslash.com with address tildeslash.com
(2)       if failed icmp type echo with timeout 15 seconds then alert
(3)       alert foo@bar.baz
```

In line 1 we define a remote host entry for the host `tildeslash.com`. In line 2 the ping test is defined; monit is asked to send *one* icmp ping package and wait for up to 15 seconds for a reply from the remote host. If the server did not reply within 15 seconds an alert is sent to the email address stated in line 3. The `timeout` part of the icmp statement is optional and if not used, timeout defaults to 5 seconds. If used, you can of course set the timeout to a higher or to a lower value than 15 seconds.

The icmp test may only be used in a `check host` entry and monit must run with super user privileges, that is, the root user must run monit. The reason is that the icmp test utilize a raw socket to send the icmp package and only the super user is allowed to create a raw socket.

Protocol test (1)

When testing a connection, either at localhost or at a remote host, monit can test the server at the protocol level. Monit support tests for many of the more popular protocols used on Internet today, such as FTP, HTTP and SMTP (See the documentation for the complete list.). The rationale for a protocol test is to test the server better, for instance some servers may allow you to open a connection, but the server may be overloaded and cannot reply. This is typical for (Java based) Application Servers, the server will accept a connection but the request processing machinery itself has gone into a dead-lock and cannot process the request. A protocol test will discover such problems.

Here are some connection-test examples where a protocol test is used:

```
(1)       if failed port 80 protocol http then ...
(2)       if failed port 25 protocol smtp then ...
```

The HTTP protocol test at line 1 will, if the connection succeed, request the default document from the server. In HTTP terms that is;

```
    GET / HTTP/1.1
    Host: foo.bar.baz
    Connection: close
```

If the server answers with an *error* code, such as; `HTTP/1.1 500 Internal server error` the protocol test fails. The SMTP test is similar, if the mail server answers with the `220` status code upon connection, all is well, otherwise the test fails. The other supported protocol tests are built on the same principle.

Protocol test (2)

Although monit support quite a few protocols it supports far from all. This is one of the reasons we also supply a general protocol test based on `send` and `expect` strings. This feature will allow you to test, if not all then most of the *text based* protocols out there. Here is a simple example for testing a HTTP server using send/expect strings instead of the built-in HTTP protocol test:

```
 (1) check host tildeslash.com with address tildeslash.com
 (2)       if failed port 80
 (3)          send "GET / HTTP/1.0\r\nHost: tildeslash.com\r\n\r\n"
 (4)          expect "HTTP/[0-9\.]{3} 200 .*\r\n"
 (5)       then ...
```

The test starting at line 2 opens a connection to the host `tildeslash.com`. If the connection succeeded, monit sends the string in line 3 and in line 4 we define the string we expect the server to return. If the server returns another string the test fails. Notice that you can use *regular expressions* in the `expect` string. You can, of course also, use as many send/expect strings as you want to. Here is another example for testing a SMTP server:

```
(1)       if failed port 25
(2)          expect "^220.*\r\n"
(3)          send "HELO localhost.localdomain\r\n"
(4)          expect "^250.*\r\n"
(5)          send "QUIT\r\n"
(6)          expect "^221.*\r\n" ...
```

Available actions

From the previous examples you will notice that several different if-tests were used. Monit provides a number of such tests, all on this common format:

IF <TEST> THEN ACTION

If the <TEST> should validate to true then the selected action is executed. Here are the available actions you can choose from:

- alert sends the user an alert message.
- restart restarts the service *and* sends an alert. Restart is conducted by first calling the service's registered stop method, wait for the service to stop and then call the service's start method.
- start starts the service *and* sends an alert.
- stop stops the service by calling the service's registered stop method *and* sends an alert. If monit stops a service it will not be checked by monit anymore nor restarted again later. To reactivate monitoring of the service again you must explicitly enable monitoring from monit's web interface or from the console using the *monitor* argument.
- exec executes an arbitrary program *and* sends an alert.
- unmonitor will disable monitoring of the service *and* send an alert. The service will not be checked by monit anymore nor restarted again later. To reactivate monitoring of the service again you must explicitly enable monitoring from monit's web interface or from the console using the *monitor* argument.

Security checks

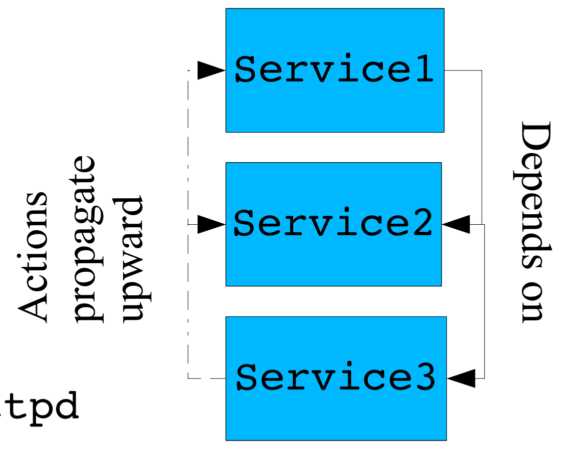
For security reasons you may want to monitor files for changes and get a notification if a file was tampered with (i.e. changed) and if it was recovered. The standard way to do this is to use the check file statement and utilize a checksum test. You can also test the file's uid[1], gid and permissions.

```
(1) check file httpd with path /usr/local/apache/bin/httpd
(2)       if failed checksum then alert
(3)       if failed uid root then alert
(4)       if failed gid root then alert
(5)       if failed permission 750 then alert
(6)       alert foo@bar.baz
```

Here we test the underlying apache binary used in the previous example. The checksum test in line 2 will raise an alert if the binary was changed, to the address specified in line 6. (You must always remember to specify an alert address in an entry if you want to receive an alert message, either globaly by 'set alert ...' or localy per service as shown above). This is all fine and dandy, but what if you are also using the same binary to start and stop a service from monit? If the binary was changed then you probably do not want monit to start the program using the changed binary if it was tampered with. The solution is to request monit, <u>not</u> to monitor the process anymore and instead, send you a checksum alert. To do this you will need to setup a depend link between the process entry and the check file entry. This is easy and in the next slide we demonstrate how to do this.

[1] The uid and gid refer to the owner and group id of the file, permissions are access restrictions for the file, in octal format. You may also use uid, gid and permission tests in a directory or in a device entry.

Dependencies



```
(1) check process apache
(2)    with pidfile "/usr/local/apache/logs/httpd.pid"
(3)    ...
(4)    depends on httpd
(5)
(6) check file httpd with path /usr/local/apache/bin/httpd
(7)       if failed checksum then unmonitor
(8)       alert foo@bar.baz
```

The first entry is the process entry for apache shown earlier (abbreviated for clarity). The fourth line sets up a dependency between this entry and the service entry *httpd* in line 6. A depend tree works as follows, if an action is conducted in a lower branch it will propagate upward in the tree and for every dependent entry execute the same action.

In this case, if the checksum should fail in line 7 then an unmonitor action is executed and the apache binary is not checked anymore. *But since the apache process entry depends on the httpd entry this entry will also execute the unmonitor action.* In short, if the checksum test for the httpd binary file should fail, both the check file httpd entry and the check process apache entry is set in unmonitoring mode.

A dependency tree is a general construct and can be used between al types of service entries and span many levels as shown in the illustration above and propagate any supported action.

Remote security check

It is possible to use monit to test the checksum for documents returned by a http server. This feature is particularly useful if you provide software packages on a remote host and want to test that a package is not changed, such as in this example where we test the monit package itself:

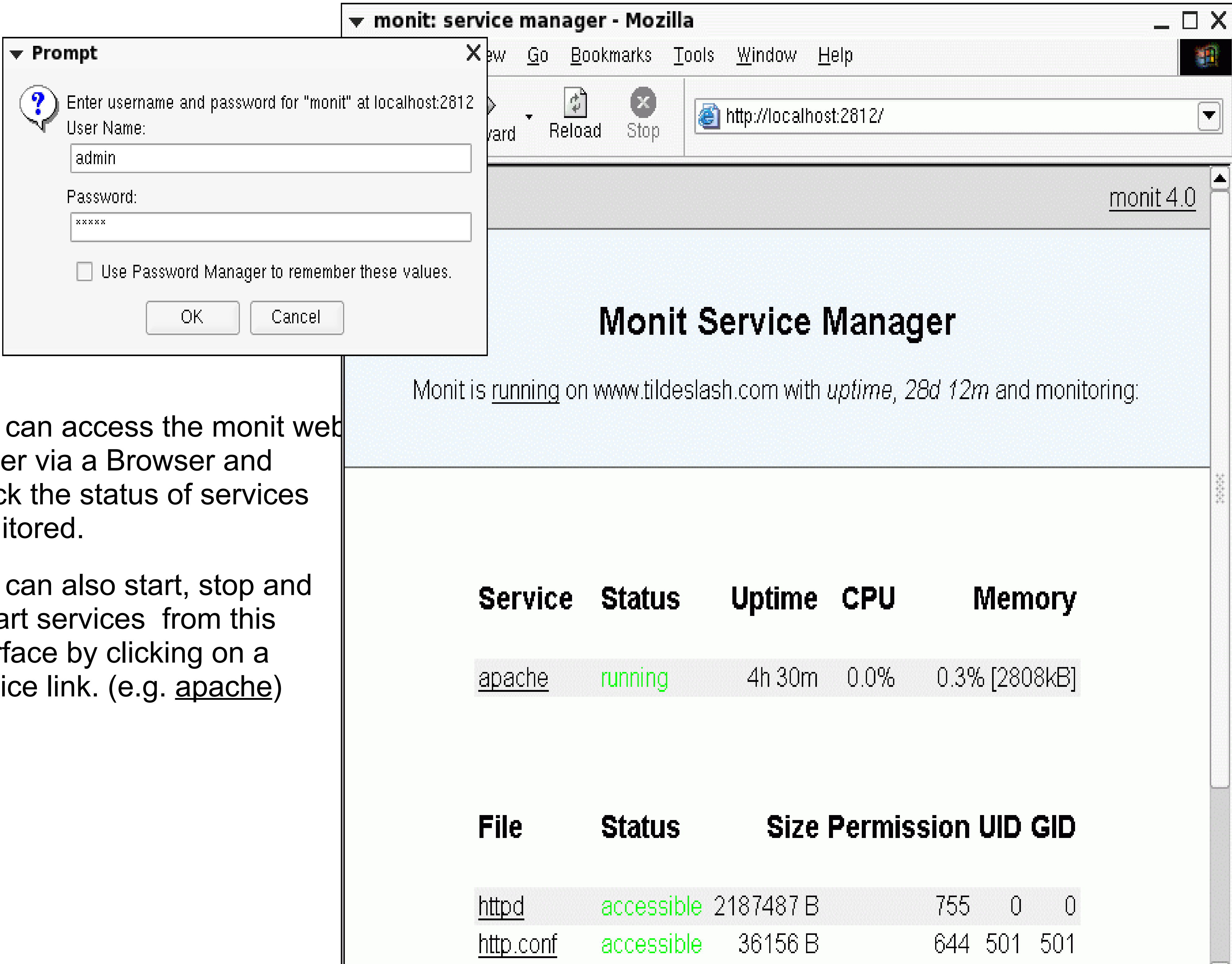
```
(1) check host tildeslash.com with address tildeslash.com
(2)       if failed port 80 protocol http
(3)          request "/monit/dist/monit-4.0.tar.gz"
(4)                  with checksum f9d26b8393736b5dfad837bb13780786
(5)                       then alert
(6)       alert foo@bar
```

Monit will download and compute a checksum for the document (in the above case, `"/monit/dist/monit-4.0.tar.gz"`) and compare the computed checksum with the expected checksum in line 4. If the sums does not match, then, in this case, an alert is sent to foo@bar informing that the HTTP protocol test failed.

Note that monit will **not** test the checksum if the server does not set the HTTP *Content-Length* header. A HTTP server should set this header when serving static documents (i.e. a file). A HTTP server will often use chunked transfer encoding instead when dynamic content is served (e.g. documents created by a CGI-script or a Servlet). Testing the checksum for dynamic content is not very useful. There are no limitation on the document size, but keep in mind that monit will use time to download the document over the network so it's probably smart not to ask monit to compute a checksum for documents larger than 1Mb or so, depending on you network connection of course.

MONIT

Web-Interface


monit: service manager - Mozilla
ew Go Bookmarks Tools Window Help
ard Reload Stop
http://localhost:2812/
Prompt
OK
Cancel
monit 4.0

Enter username and password for "monit" at localhost:2812
User Name: admin
Password: *****
☐ Use Password Manager to remember these values.

Monit Service Manager

Monit is running on www.tildeslash.com with *uptime, 28d 12m* and monitoring:

Service	Status	Uptime	CPU	Memory
apache	running	4h 30m	0.0%	0.3% [2808kB]

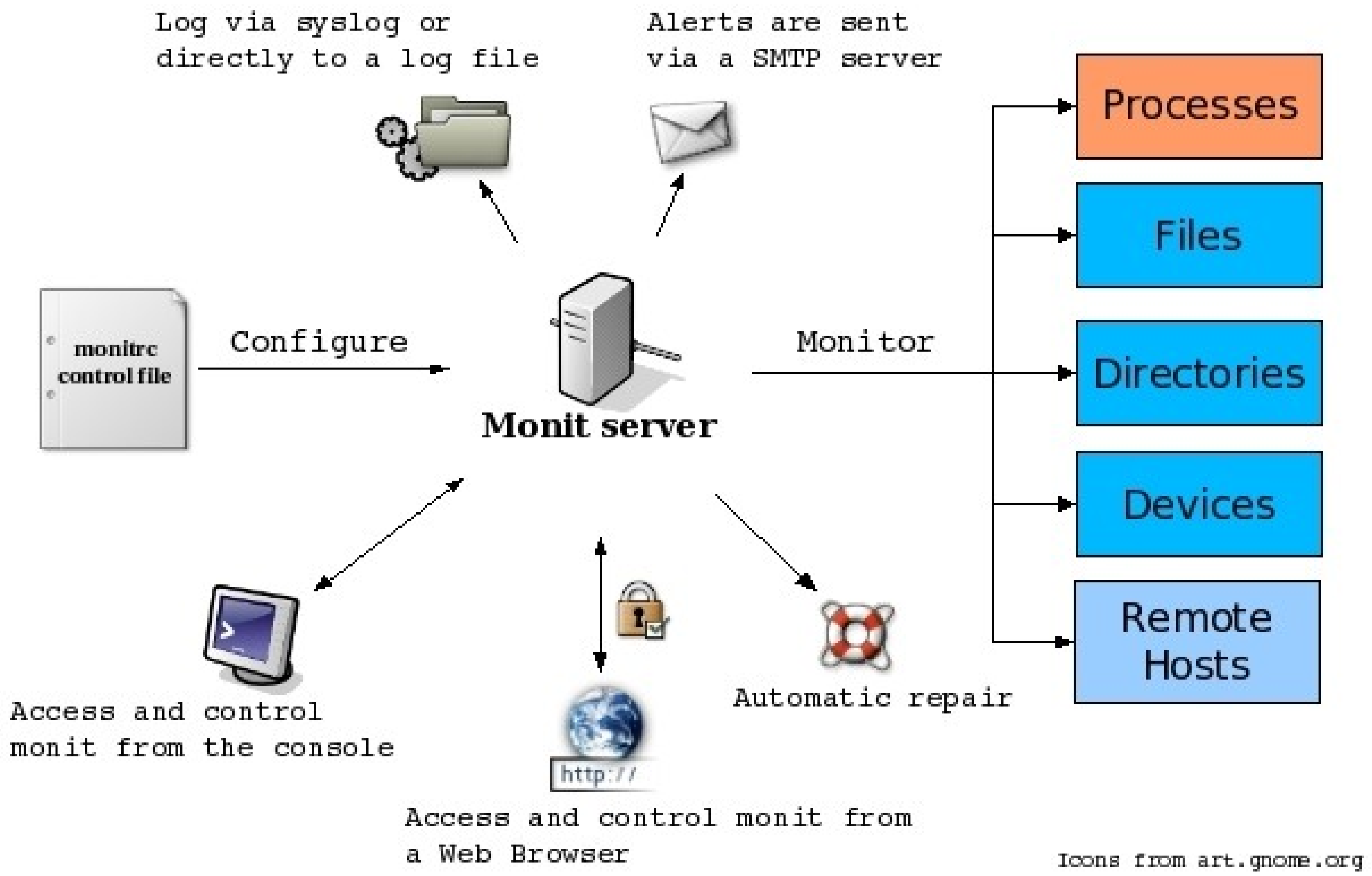
File	Status	Size	Permission	UID	GID
httpd	accessible	2187487 B	755	0	0
http.conf	accessible	36156 B	644	501	501

You can access the monit web server via a Browser and check the status of services monitored.

You can also start, stop and restart services from this interface by clicking on a service link. (e.g. apache)

MONIT

Overview


Log via syslog or directly to a log file
Alerts are sent via a SMTP server
monitrc control file
Configure
Monit server
Monitor
Processes
Files
Directories
Devices
Remote Hosts
Access and control monit from the console
http://
Access and control monit from a Web Browser
Automatic repair
Icons from art.gnome.org

MONIT

Support

Documentation

- Extensive documentation is provided with the software
- Documentation is also available online at: http://www.tildeslash.com/monit/

Mailing list

- If you have questions or comments about the software or documentation please subscribe to the monit mailing list at: http://mail.nongnu.org/mailman/listinfo/monit-general You may also search in the mailing list archive for answers.

Professional support and contract development

- The developers behind monit provides contract support and contract development. If you are looking for this kind of services please contact us at: monitgroup@tildeslash.com or visit http://www.tildeslash.com/monit/services.html. You may also call +47 97 14 12 55 or send a fax to + 47 22 71 29 99 (Opening hours: 10-18 CET). The project has developers from USA, Germany, Norway and the Czech Republic.