simplevisor Documentation

Release 1.2

Massimo Paladin

June 27, 2016

Contents

CHAPTER 1

Main Features

- ***Standalone services*** The supervisor is able to run default services like *httpd*, *mysql* which are invoked with a command of the type:

```
$ service httpd start
```

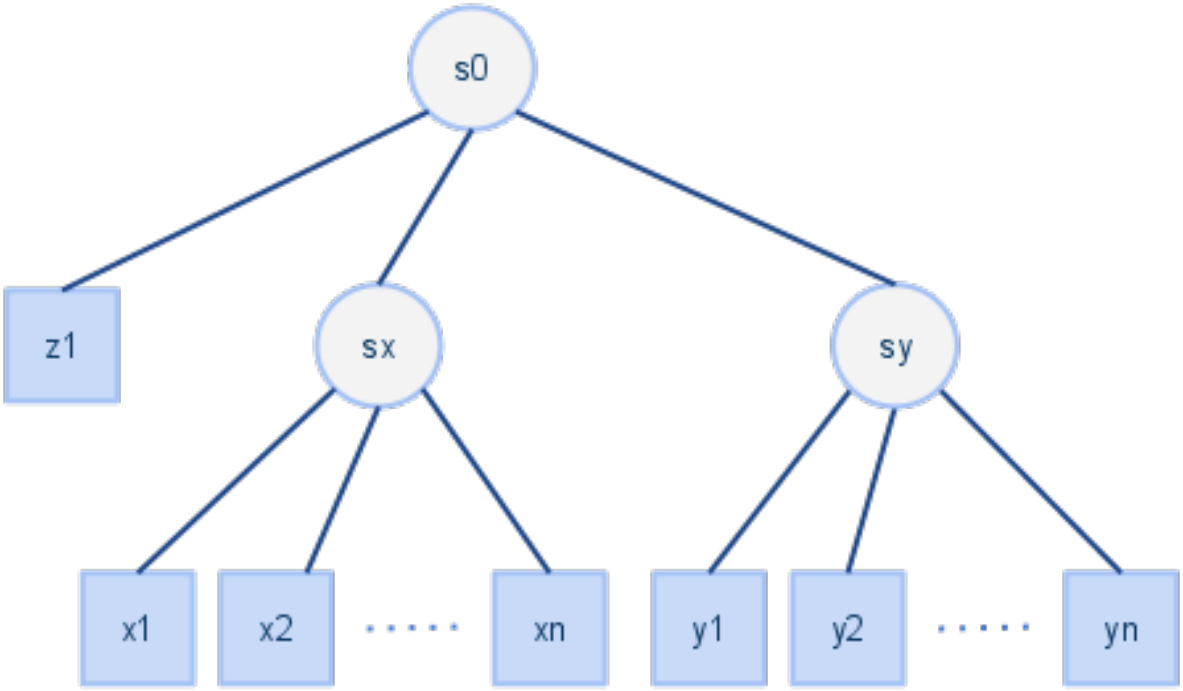
- ***Executable*** The supervisor is able to run any executable program, not only standard services:

```
/opt/whatever/instance/bin/service --option 1 --other "foo bar" --config /opt/whatever/insta
```

- ***Dead or "hang" state handling*** Something important is to be able to handle services in an apparently running state but which are hanging. This should be handled from the service startup script.
- ***OTP hierarchy of services*** Inspired by OTP supervisor it support hierarchies of services. Services should be grouped together acting as a single service to the parent.

  A supervision tree is composed by supervisors and workers:

  - workers are identified by xN, yN and zN.
  - supervisors are identified by sX.



- ***OTP strategies for handling services*** Inspired by the OTP platform there are different strategies for handling group of services and their behavior.
- ***Commands: start/stop/status/restart*** Different commands can be specified to handle a service:
  - start
  - stop
  - status
  - restart (defaults to stop+start)
- ***Ensure expected state*** Each service should have an expected state. Possible states are:
  - running: expect the service to be running fine
  - stopped: the service should be disabled
- ***Daemon mode*** Supervisor is able to run in daemon mode continuously checking and applying the given configuration.
- ***One shot mode*** Supervisor can run in "one shot mode", which means it goes through the services handling and then exit. In order to handle the strategies correctly the state of the services is stored in order to be read during the next execution.
- ***Root not required*** Supervisor does not require to be run as *root* user, it is able to run as any user of course limited by the user privileges.

CHAPTER 2

Installation

You can install simplevisor through different sources.

2.1 pip/easy_install way

You can automatically install it through *easy_install*:

```
easy_install simplevisor
```

or *pip*:

```
pip install simplevisor
```

2.2 tarball

You can install it through the tarball, download the latest one from http://pypi.python.org/pypi/simplevisor, unpack it, cd into the directory and install it:

```
version=X
wget http://pypi.python.org/packages/source/s/simplevisor/simplevisor-${version}.tar.gz
tar xvzf simplevisor-${version}.tar.gz
cd simplevisor-${version}
# Run the tests
python setup.py test
# Install it
python setup.py install
```

2.3 rpm

RPMs are available for *Fedora* main branches and *EPEL 5/6*, you can simply install it with *yum*:

```
yum install python-simplevisor
```

CHAPTER 3

Configuration

Simplevisor has one main configuration file. The default format of this configuration file is the Apache Style Config and the configuration parsing is handled by the Perl Config::General module.

Alternatively, you can use JSON configuration files (via *–conftype json*): this removes the need for Perl but the features listed below will not be supported.

Here are the main features supported in the Apache Style Config syntax:

comments comments are allowed through lines starting with #

blank lines blank lines are ignored

file inclusion file inclusion is supported to allow modularization of the configuration file; it is possible to include a file which is in the same folder or in its subtree with the following directive: <<include relative_file_path.conf>>

variable interpolation variable interpolation is supported in order to reduce verbosity and duplication in the main blocks of the configuration file.

simplevisor and *entry* sections allow variables declaration, variables are declared like any other fields with the only restriction that their name is prefixed with *var_*:

```
...
var_foo = bar
...
```

You can use variables in the value of a field, you can not use them inside keys and their scope is the subtree of declaration. They can be used surrounded by curly braces and prefixed by a dollar: *${var_name}*.

An usage example:

```
...
var_foo = bar
property_x = ${var_foo} the rest of the value
...
```

The options specified through the command line have the priority over the options declared in the configuration file.

You can find a configuration example in the *examples* folder, it is called:

```
simplevisor.conf.example
```

copy and edit the file:

```
# Simplevisor has one main configuration file. The format of the configuration
# file is the Apache Style Config, the configuration parsing is handled
# with Perl Config::General module for commodity.
#
# Following features are supported:
```

```
#
# - Apache Style Config syntax
# - comments are allowed through lines starting with #
# - blank lines are ignored
# - file inclusion is supported to allow modularization of the
#   configuration file. It is possible to include a file which is in the same
#   folder or in its subtree with the following directive:
#   <<include relative_file_path.conf>>
# - variable interpolation is supported in order to reduce verbosity and
#   duplication in the main blocks of the configuration file.
#   simplevisor and entry sections allow variables declaration, variables
#   are declared like any other fields with the only restriction that their
#   name is prefixed with var_:
#       ...
#       var_foo = bar
#       property_x = ${var_foo} the rest of the value
#       ...
#   You can use variables in the value of a field, you can not use them inside
#   keys and their scope is the subtree of declaration.

<simplevisor>
    # file used to store the status
    store = /var/cache/simplevisor/simplevisor.json

    # pid file, ignored if simplevisor-control is used
    #pidfile = /path/to/pid

    # interval (sleep time) between supervision cycles, from the end
    # of one cycle to the start of the next one, in seconds
    #interval = 120

    # configure the logging system, must be one of: stdout,syslog,file
    log = stdout

    # if logging system is file you need to specify a log file,
    # check that the logfile is writable by the specified user.
    #logfile = /var/log/simplevisor/simplevisor.log

    # the loglevel is warning by default,
    # the available log levels are: debug,info,warning,error,critical
    #loglevel = info
</simplevisor>

    <<include simplevisor.services.example>>
```

where *simplevisor.services.example* could look like:

```
<entry>
    type = supervisor
    name = svisor1
    window = 12
    adjustments = 3
    strategy = one_for_one
    <children>
            <entry>
                type = service
                name = httpd
                expected = stopped
                control = /sbin/service httpd
```

```
            </entry>
        <<include other_service.conf>>
    </children>
</entry>
```

and *other_service.conf* could look like:

```
<entry>
    type = service
    name = custom1
    start = /path/to/script --conf /path/to/conf --daemon
    # If you cannot provide a status or stop command you can specify a
    # pattern which will be used to look for the process in the process
    # table, however this is supported only on linux.
    # If not specified start command is used as pattern.
    pattern = /path/to/script --conf /path/to/conf --daemon
</entry>
```

CHAPTER 4

simplevisor command

simplevisor 1.2 - simple daemons supervisor

4.1 SYNOPSIS

simplevisor [–conf CONF] [–conftype CONFTYPE] [–daemon] [–interval INTERVAL] [-h] [–log LOG] [–logfile LOGFILE] [–loglevel LOGLEVEL] [–logname LOGNAME] [-p PIDFILE] [–store STORE] [–version] command [path]

4.2 DESCRIPTION

Simplevisor is a simple daemons supervisor, it is inspired by Erlang OTP and it can supervise hierarchies of services.

COMMANDS

If a path is given or only one service entry is given:

for a given X command run the service X command where service is the only entry provided or the entry identified by its path

If a path is given and the root entry is a supervisor:

restart_child tell a running simplevisor process to restart the child identified by the given path; it is different from the restart command as described above because, this way, we are sure that the running simplevisor will not attempt to check/start/stop the child while we restart it

If a path is not given and the root entry is a supervisor:

start start the simplevisor process which start the supervision. It can be used with –daemon if you want it as daemon

stop stop the simplevisor process and all its children, if running

status return the status of the simplevisor process

check return the comparison between the expected state and the actual state. 0 -> everything is fine 1 -> warning, not expected

single execute one cycle of supervision and exit. Useful to be run in a cron script

wake_up tell a running simplevisor process to wake up and supervise

stop_supervisor only stop the simplevisor process but not the children

stop_children only stop the children but not the simplevisor process

check_configuration only check the configuration file

pod generate pod format help to be used by pod2man to generate man page

rst generate rst format help to be used in the web doc

help same as -h/–help, print help page

4.3 OPTIONS

positional arguments:

command check, check_configuration, help, pod, restart, restart_child, rst, single, start, status, stop, stop_children, stop_supervisor, wake_up

path path to a service, subset of commands available: start, stop, status, check, restart

optional arguments:

–conf CONF configuration file

–conftype CONFTYPE configuration file type (default: apache)

–daemon daemonize, ONLY with start

–interval INTERVAL interval to wait between supervision cycles (default: 60)

-h, –help print the help page

–log LOG available: null, file, syslog, stdout (default: stdout)

–logfile LOGFILE log file, ONLY for file

–loglevel LOGLEVEL log level (default: warning)

–logname LOGNAME log name (default: simplevisor)

-p, –pidfile PIDFILE the pidfile

–store STORE file where to store the state, it is not mandatory, however recommended to store the simplevisor nodes status between restarts

–version print the program version

4.4 EXAMPLES

Create and edit the main configuration file:

```
## look for simplevisor.conf.example in the examples.
```

Run it:

```
simplevisor --conf /path/to/simplevisor.conf start
```

to run it in daemon mode:

```
simplevisor --conf /path/to/simplevisor.conf --daemon start
```

For other commands:

```
simplevisor --help
```

Given the example configuration, to start the httpd service:

```
simplevisor --conf /path/to/simplevisor.conf start svisor1/httpd
```

4.5 AUTHOR

Massimo Paladin <massimo.paladin@gmail.com> -

CHAPTER 5

simplevisor-control command

5.1 NAME

simplevisor-control - run simplevisor as a service

5.2 SYNOPSIS

simplevisor-control command [path]

5.3 DESCRIPTION

simplevisor-control command can be used to run simplevisor as a service.

5.4 OPTIONS

command one of: start, stop, restart, status, check

path look at simplevisor man page for path behavior

5.5 EXAMPLES

On linux you can look at the script shipped in the examples folder which is called simplevisor-new-instance, it creates folders and the configuration to run a simplevisor instance.

```
mkdir -p /var/lib/myinstance/bin
mkdir -p /var/lib/myinstance/data
mkdir -p /var/lib/myinstance/etc
```

Create a file /var/lib/myinstance/bin/service with content and make it executable:

```
#!/bin/sh
#
# init script that can be symlinked from /etc/init.d
#
```

```
# chkconfig: - 90 15
# description: my simplevisor instance

. "/var/lib/myinstance/etc/simplevisor.profile"
exec "/usr/bin/simplevisor-control" ${1+"$@"}
```

/var/lib/myinstance/etc/simplevisor.profile could look like:

```
# main
export SIMPLEVISOR_NAME=myinstance
# if you want to run it as another user:
#export SIMPLEVISOR_USER=games
export SIMPLEVISOR_CONF=/var/lib/myinstance/etc/simplevisor.conf
export SIMPLEVISOR_PIDFILE=/var/lib/myinstance/data/simplevisor.pid
export SIMPLEVISOR_LOCKFILE=/var/lib/myinstance/data/simplevisor.lock
```

Create */var/lib/myinstance/etc/simplevisor.conf* according to simplevisor documentation.

For Red Hat or Fedora you can symlink service script:

```
ln -s /var/lib/myinstance/bin/service /etc/init.d/myinstance
```

And use it as a normal service:

```
/sbin/service myinstance start|stop|status|restart|check
```

5.6 AUTHOR

Massimo Paladin <massimo.paladin@gmail.com>

CHAPTER 6

Supervisor Abstraction

This module implements a `Supervisor` class.

An example of supervisor declaration:

```
<entry>
    type = supervisor
    name = supervisor1
    window = 12
    adjustments = 3
    strategy = one_for_one
    expected = none
    <children>
        .... other supervisors or services
    </children>
</entry>
```

6.1 Parameters

name unique name of the supervisor.

window window of supervision cycles which should be considered when defining if a supervisor is in a failing state.

adjustments maximum number of cycles on which a child adjustment was needed in the given window of supervision cycle in order to consider it a failure.

strategy

- one_for_one: if a child process terminates, only that process is restarted.
- one_for_all: if a child process terminates, all other child processes are terminated and then all child processes, including the terminated one, are restarted.
- rest_for_one: If a child process terminates, the *rest* of the child processes i.e. the child processes after the terminated process in start order are terminated. Then the terminated child process and the rest of the child processes are restarted.

expected none|running|stopped

children children structure.

6.2 Required Parameters

```
- children section is required or supervisor is not useful
```

6.3 Default Parameters

```
- name = supervisor
- expected = none
- window = 12
- adjustments = 3
- strategy = one_for_one
```

CHAPTER 7

Service Abstraction

This class implements a `Service` abstraction. As service we mean a process which runs in daemon mode.

An example of service declaration:

```
<entry>
    type = service
    name = httpd
    expected = stopped
    control = /sbin/service httpd
</entry>
```

another example for a standalone script:

```
<entry>
    type = service
    name = custom1
    start = /path/to/script --conf /path/to/conf --daemon
    # If you cannot provide a status or stop command you can specify a
    # pattern which will be used to look for the process in the process
    # table, however this is supported only on Linux.
    # If not specified start command is used as pattern.
    pattern = /path/to/script --conf /path/to/conf --daemon
</entry>
```

If one of the parameters contains one or more spaces you should quote them in url-like style, invoked commands are `urllib.unquote()` before being launched like in this example:

```
...
start = /path/to/script --conf /path/to/conf --space hello%20world start
...
```

The stdout and the stderr of the commands executed is logged as debug level within the configured log system.

The commands declared should provide return codes according to the default LSB Unix return codes, for more info visit LSB Core Specification:

```
0        program is running or service is OK
1        program is dead and /var/run pid file exists
2        program is dead and /var/lock lock file exists
3        program is not running
4        program or service status is unknown
5-99     reserved for future LSB use
100-149  reserved for distribution use
150-199  reserved for application use
200-254  reserved
```

7.1 Parameters

control the control of the command to run. If specified it will be the prefix of *start/stop/status* commands.

daemon if the service command runs in foreground and you wish to daemonize it you can declare this option with value the pidfile path that should be used for the daemonization.

If control is specified this option is ignored.

Given the start command:

```
start = /path/to/script --conf /path/to/conf
```

and declaring:

```
daemon = /path/to/script_pidfile.pid
```

it is like specifying the following pair of commands/values:

```
start = /usr/bin/simplevisor-loop -c 1 --pidfile /path/to/script_pidfile.pid --daemon /path/to/s

stop = /usr/bin/simplevisor-loop --pidfile /path/to/script_pidfile.pid --quit

status = /usr/bin/simplevisor-loop --pidfile /path/to/script_pidfile.pid --status
```

Hence, if *daemon* is specified *stop* and *status* command are overwritten.

expected expected state of the service. Valid values are *running* and *stopped*.

name unique name of the *worker/service*.

path the path for executing the commands. Multiple values should be separated by colons.

pattern used to look for the service in the process table for stop and status commands if they are not specified and control is also not specified. Accepted values are valid python regular expressions: `re`.

restart specify a custom restart command.

If <control> is specified:

- if <restart> is not specified "<control> restart" is executed
- if <restart> = "stop+start" a "<control> stop" followed by a "<control> start" is executed
- else "<restart>" is executed

If <control> is not specified:

- if <restart> is not specified "<stop>" followed by "<start>" is executed
- else "<restart>" is executed

start specify a custom start command.

If <control> is specified:

- if <start> is not specified "<control> start" is executed
- else "<start>" is executed

If <control> is not specified:

- "<start>" is executed

status specify a custom status command.

If <control> is specified:

- "<control> status" is executed

If <control> is not specified:

- if <status> is specified "<status>" is executed
- else it will look for it in the process table either looking for the start command or the provided pattern.

Status commands are expected to exit with return code according to the following following:

- *0:* the service is running fine
- *3:* the service is stopped
- *other:* return code is interpreted as dirty/zombie/hang state

stop specify a custom stop command.

If <control> is specified:

- "<control> stop" is executed

If <control> is not specified:

- if <stop> is specified "<stop>" is executed
- else it will look for it in the process table either looking for the start command or the provided pattern and then kill it.

timeout the maximum timeout for any service command, set to 60 seconds by default.

7.2 Required Parameters

```
- name
- one of: start, control
```

7.3 Default Parameters

```
- expected = running
- timeout = 60
- all the others are default to None
```

CHAPTER 8

Getting Started

Simplevisor is a simple daemons supervisor, it is inspired by Erlang OTP and it can supervise hierarchies of services.

Dependencies:

```
argparse for python < 3.2
simplejson for python < 2.6
```

Install it:

```
easy_install simplevisor
# look at the installation page for details
```

create and edit the main configuration file:

```
simplevisor.conf.example available in the examples
# check the configuration page for details
```

run it with:

```
simplevisor --conf /path/to/simplevisor.conf start
# or as a daemon
simplevisor --conf /path/to/simplevisor.conf --daemon start
```

check the help page:

```
simplevisor help
```

if you want to run it as a service user simplevisor-control as init script.

Author: Massimo.Paladin@gmail.com

S

S