ncolony

Release [ncolony, version 17.9.0]

Sep 19, 2017

Contents

NColony is a process monitor. It starts processes, and restarts them when they die, or when asked to.

NColony is a Python package available from PyPI, and developed on GitHub. It is based on Twisted, and works on both Python 2 and Python 3.

NColony is guided by the following principles:

- Commitment to code quality: we use code reviews, unit testing, integration testing, and static checking to increase the quality of our code.
- Different domains in different processes: the main NColony daemon just starts, shuts down, and restarts processes. Other concerns, such as health checking and network control, are handled by other processes – optionally managed by NColony.
- Run processes directly as children, to avoid race-condition-prone `pid` file mechanisms for monitoring.

Since NColony should be reliable, seeing as how it is monitoring other processes, the hope is that by keeping the code small and high quality, we can make it as stable as possible.

NColony is developed under a *Code of Conduct*, inspired by the Contributor covenant.

CHAPTER 1

Introduction

Overview

NColony is a system to control and monitor a number of processes on UNIX-like systems. Its primary use-case is to run servers, and it is specifically optimized to container architectures like Docker.

Installing

The recommended way to install is using virtualenv and pip:

```
$ python -m virtualenv venv
$ venv/bin/python -m pip install ncolony
```

When running other python processes with ncolony, it is possible to either run them from the ncolony virtual environment or from a separate virtual environment.

For more options with pip installation, for example for network-less install, see the pip documentation.

NColony components

NColony is built on the Twisted framework. Most of its parts are implemented as twistd plugins, allowing the end-user to control features like logging, reactor selection and more.

twistd ncolony

> The process monitor is called as the "ncolony" twistd plugin. It starts processes, and continuously monitors both process state and configuration, and makes sure they are in sync.
>
> The monitor configuration is a directory with a file per process. It also monitors a messages directory with "ephemeral" configuration: mostly restart requests.

twistd ncolony-beatcheck

This plugin, intended to be run under the ncolony monitor, will look at other processes' configuration, check if they are supposed to beat hearts (periodically touch a file) and message ncolony with a restart request if the heart does not beat for too long.

`twistd ncolony-scheduler`

This plugin, intended to be run under the ncolony monitor, will periodically run a short-lived process. This allows the main ncolony plugin to assume all of its processes are long-lived, while still supporting short-lived processes. This is useful, e.g., for log-rotation or other periodic clean-up tasks.

`python -m ncolony ctl`

Control program – add, remove and restart processes.

`python -m ncolony reaper`

"PID 1". Designed to work with the ncolony monitor as a root process in a container environment. It is designed to run only one program, and then reap any children it adopts.

CHAPTER 2

Running NColony

In this section, we assume everything to be running in an environment where "pip install ncolony" has taken place. This will usually be a virtualenv, which should be activated.

Running ncolony monitor

Create two directories: `conf` and `messages`. For simplicity, we will assume that they live in the root directory, `/` – `/conf/` and `/messages/`.

```
twistd -n ncolony --messages /messages --config /conf
```

Of course, it is useless to run a monitor without any processes to monitor:

```
python -m ncolony ctl add my-cat-program --cmd cat
```

This will add the cat program as a monitored program. Because `cat` hangs forever reading from stdin, it will never die. It is important to note that it does not matter what order we run these in. In fact, if we now shut down (via CTRL-C) the ncolony monitor, and start it again, it will start the `cat` program again.

`twistd` Command-Line Options

A full set of twistd command-line options can be found in the twistd help (available via **`twistd --help`**).

`twistd ncolony` Command-Line Options

Option: –config DIR Directory for configuration

Option: –messages DIR Directory for messages

Option: –frequency SECONDS Frequency of checking for updates [default: 10]

Option: –pid DIR Directory of PID files

Option: -t SECONDS, –threshold SECONDS How long a process has to live before the death is considered instant, in seconds. [default: 1]

Option: -k SECONDS, –killtime SECONDS How long a process being killed has to get its affairs in order before it gets killed with an unmaskable signal. [default: 5]

Option: -m SECONDS, –minrestartdelay SECONDS The minimum time (in seconds) to wait before attempting to restart a process [default: 1]

Option: -M SECONDS, –maxrestartdelay SECONDS The maximum time (in seconds) to wait before attempting to restart a process [default: 3600]

`python -m ctl` Command-Line Options

The following must be given before the subcommand:

Option: –messages DIR directory of NColony monitor messages

Option: –config DIR directory of NColony monitor configuration

The following follow the subcommand:

restart-all Takes no arguments

restart, remove Only one positional argument – name of program

`python -m ctl add` Command-Line Options

Option: –cmd CMD Name of executable

Option: –arg ARGS Add an argument to the command

Option: –env NAME=VALUE Add an environment variable

Option: –uid UID Run as given user id (only useful if ncolony monitor is running as root)

Option: –gid GID Run as given group id (only useful if ncolony monitor is running as root)

Option: –extras EXTRAS a JSON-encoded dictionary with extra configuration parameters. Those are not used by the monitor itself, but are available to the running program (as the variable NCOLONY_CONFIG) and to other programs which scan the configuration directory.

For programmatic access, it is recommended to use the `ncolony.ctllib` module from a Python program instead of passing arguments to a `python -m ncolony ctl` subprocess.

Logging

The log of `ncolony` itself is configured by using the `twistd` log configuration. While `ncolony` will log processes' stdout/err, it is highly encouraged to set logs for these. Ideally, `ncolony` logs should also show either catastrophic errors in processes, such that even the log could not be opened, or messages that are sent before the log is set.

CHAPTER 3

Configuration

The ncolony configuration is stored in a directory, `conf`. There is no default – this directory needs to be passed explicitly to the ncolony server as well as to the control command.

The usual command to modify this configuration is **`python -m ncolony ctl add`** (although `remove` is also useful, of course). In order to modify a command, add should be called with the same name. NColony will automatically restart the command when its configuration changes.

Examples

Running Sentry:

```
$ python -m ncolony ctl add sentry --cmd /myvenv/sentry \
    --arg start --arg --config=/etc/sentry.conf
```

or

```
from ncolony import ctllib

ctllib.add(name='sentry', cmd='/myenv/sentry',
           args=['start', '--config=/etc/sentry.conf'])
```

Running a Twisted demo server:

```
$ python -m ncolony ctl add demo-server --cmd /myvenv/twistd \
    --arg --nodaemon --arg web
```

or

```
from ncolony import ctllib

ctllib.add(name='demo-server', cmd='/myenv/twistd',
           args=['--nodaemon', 'web'])
```

Note the `--nodaemon`: programs run by ncolony should not daemonize, so that ncolony can properly monitor them.

CHAPTER 4

Running processes

Nondaemonizing

Just like Supervisor and Daemontools, NColony assumes that the processes it runs do not daemonize. While many servers will daemonize by default, it is often possible to keep them in the foreground by passing the right command-line options or setting the right configuration file variables.

When configuring an ncolony command, first try running it from the command-line. If the prompt returns immediately (or, indeed, at all unless the program unexpectedly crashed) that means the program daemonizes itself.

Supervisor and Daemontools both have many resources about how to achieve that properly. There are also examples in The DJB Way. While both of these also come with work-around scripts that try to de-daemonize processes, both approaches are fundamentally broken. Not through any lack of effort by the authors, but because it is impossible to solve it correctly. NColony takes the purist tack that these things should be solved at the daemon level.

If working around those problems is important enough, it is possible to install fghack or pidproxy in order to semi-un-daemonize servers.

Environment

Processes will be run in an environment composed of:

- Environment variables explicitly requested by `add`
- `NCOLONY_NAME` (name of the process)
- `NCOLONY_CONFIG` (JSON-encoded configuration passed to ncolony itself)

Note that the environment that ncolony runs under will not be inherited, and no other variables are automatically set. In particular, if `USER` or `HOME` are needed, they should be passed explicitly by `add`.

CHAPTER 5

API

ncolony.service

Implement the 'ncolony' twistd plugin.

```
$ twistd ncolony --config <dir> --messages <dir>
```

Will run a service that brings up all processes described in files in the configuration directory (and shuts them down if the files ago away), and listens for restart messages on the messages directory.

class ncolony.service.**Options**
Options for ncolony service

postOptions()
Checks that required messages/config directories are present

class ncolony.service.**TransportDirectoryDict**(*output*)
Dict-like object that writes the 'pid' value to a directory

This dict-like object assumes all the values have a 'pid' attribute, and writes that attribute into a file named the same as the key in the given directory.

ncolony.service.**get**(*config*, *messages*, *freq*, *pidDir=None*, *reactor=None*)
Return a service which monitors processes based on directory contents

Construct and return a service that, when started, will run processes based on the contents of the 'config' directory, restarting them if file contents change and stopping them if the file is removed.

It also listens for restart and restart-all messages on the 'messages' directory.

Parameters

- **config** – string, location of configuration directory
- **messages** – string, location of messages directory
- **freq** – number, frequency to check for new messages and configuration updates
- **pidDir** – {twisted.python.filepath.FilePath} or None, location to keep pid files

- **reactor** – something implementing the interfaces {twisted.internet.interfaces.IReactorTime} and {twisted.internet.interfaces.IReactorProcess} and

Returns service, {twisted.application.interfaces.IService}

ncolony.service.**makeService**(*opt*)
Return a service based on parsed command-line options

Parameters **opt** – dict-like object. Relevant keys are config, messages, pid, frequency, threshold, killtime, minrestartdelay and maxrestartdelay

Returns service, {twisted.application.interfaces.IService}

ncolony.ctllib

This module can be used either as a library from Python or as a commandline using the wrapper `ctl` via

```
$ python -m ncolony ctl <arguments>
```

Description of the service's interface is in <figure out how to do a back-reference>.

The add/remove messages add/remove configuration files for processes. Since removal of a configuration is equivalent to killing the process, nothing else nees to be done to rid of needed processes.

All functions which are meant to be used as a library API

Add is the most complicated function, because it needs to be able to express every aspect of the process. It allows control of the name, command, arguments, environment variables and uid/gid.

Removal is pretty simple, since it only needs the name.

Restart also needs just the name.

Restart-all does not need even the name, since it restarts all processes.

class ncolony.ctllib.**Places**(*config*, *messages*)

config
Alias for field number 0

messages
Alias for field number 1

ncolony.ctllib.**add**(*places*, *name*, *cmd*, *args*, *env=None*, *uid=None*, *gid=None*, *extras=None*)
Add a process.

Parameters

- **places** – a Places instance
- **name** – string, the logical name of the process
- **cmd** – string, executable
- **args** – list of strings, command-line arguments
- **env** – dictionary mapping strings to strings (will be environment in subprocess)
- **uid** – integer, uid to run the new process as
- **gid** – integer, gid to run the new process as

Returns None

ncolony.ctllib.**call**(*results*)
Call results.func on the attributes of results

Params result dictionary-like object

Returns None

ncolony.ctllib.**main**(*argv*)
command-line entry point

–messages: messages directory

—config: configuration directory

subcommands:

add: name (positional)

–cmd (required) – executable

–arg – add an argument

–env – add an environment variable (VAR=value)

–uid – set uid

–gid – set gid

remove: name (positional)

restart: name (positional)

restart-all: no arguments

ncolony.ctllib.**remove**(*places*, *name*)
Remove a process

Params places a Places instance

Params name string, the logical name of the process

Returns None

ncolony.ctllib.**restart**(*places*, *name*)
Restart a process

Params places a Places instance

Params name string, the logical name of the process

Returns None

ncolony.ctllib.**restartAll**(*places*)
Restart all processes

Params places a Places instance

Returns None

ncolony.directory_monitor

Monitor directories for configuration and messages

ncolony.directory_monitor.**checker**(*location*, *receiver*)
Construct a function that checks a directory for process configuration

The function checks for additions or removals of JSON process configuration files and calls the appropriate receiver methods.

Parameters

- **location** – string, the directory to monitor
- **receiver** – IEventReceiver

Returns a function with no parameters

ncolony.directory_monitor.**messages**(*location*, *receiver*)
Construct a function that checks a directory for messages

The function checks for new messages and calls the appropriate method on the receiver. Sent messages are deleted.

Parameters

- **location** – string, the directory to monitor
- **receiver** – IEventReceiver

Returns a function with no parameters

ncolony.interfaces

Interface definitions

class ncolony.interfaces.**IMonitorEventReceiver**

add()
New file appeared

Params name string, file name

Params contents string, file contents

Returns None

remove()
File went away

Params name string, file name

Returns None

message()
Message sent

Params contents string, message contents

Returns None

ncolony.process_events

Convert events into process monitoring actions.

class ncolony.process_events.**Receiver**(*monitor*, *environ=None*)
A wrapper around ProcessMonitor that responds to events

Params monitor a ProcessMonitor

add(*name*, *contents*)
Add a process

Params name string, name of process

Params contents string, contents parsed as JSON for process params

Returns None

message(*contents*)
Respond to a restart or a restart-all message

Params contents string, contents of message parsed as JSON, and assumed to have a 'type' key, with value either 'restart' or 'restart-all'. If the value is 'restart', another key ('value') should exist with a logical process name.

remove(*name*)
Remove a process

Params name string, name of process

ncolony.schedulelib

Construct a Twisted service for process scheduling.

```
$ twistd -n ncolonysched --timeout 2 --grace 1 --frequency 10 --arg /bin/echo --arg␣
↪hello
```

class ncolony.schedulelib.**Options**
Options for scheduler service

opt_arg(*arg*)
Argument

class ncolony.schedulelib.**ProcessProtocol**(*deferred*)
Process protocol that manages short-lived processes

childConnectionLost(*reason*)
Ignore childConnectionLoss

childDataReceived(*fd*, *data*)
Log data from process

Params fd File descriptor data is coming from

Params data The bytes the process returned

makeConnection(*transport*)
Ignore makeConnection

processEnded(*reason*)
Report process end to deferred

Params reason a Failure

processExited(*reason*)
Ignore processExited

ncolony.schedulelib.**makeService**(*opts*)
Make scheduler service

Params opts dict-like object. keys: frequency, args, timeout, grace

ncolony.schedulelib.**runProcess**(*args*, *timeout*, *grace*, *reactor*)
Run a process, return a deferred that fires when it is done

Params args Process arguments

Params timeout Time before terminating process

Params grace Time before killing process after terminating it

Params reactor IReactorProcess and IReactorTime

Returns deferred that fires with success when the process ends, or fails if there was a problem spawning/terminating the process

Python Module Index

n

Index