TcpNetLock Documentation

Release 0.1.8

Horacio G. de Oro

Jul 20, 2018

Contents:

CHAPTER 1

TcpNetLock

1.1 Network lock based on TCP sockets

- Free software: GNU General Public License v3
- Documentation: https://tcpnetlock.readthedocs.io/
- GitHub: https://github.com/hgdeoro/tcpnetlock/
- Docker: https://hub.docker.com/r/hgdeoro/tcpnetlock/

1.2 Why?

While deploying applications to Kubernetes, I needed a way to make sure that some potential concurrent, distributed actions, are not executed concurrently. For example:

- **database migrations**: just one Pod in the Kubernetes cluster should be able to apply the database migrations
- for **batch jobs**, different workers could be working on the same resource, this can be avoided with this lock mechanism

Of course, Zookeeper is a MUCH BETTER solution, but that's too much for my use cases...

1.3 How it works

Assuming the server is running on localhost, let's get a lock using telnet:

```
$ telnet localhost 7654
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

To try to acquire a lock, send:

```
lock,name:django-migrations
```

Server responds with:

```
ok
```

From that point, and while the TCP connection is open, you have the lock.

If you try the same in a different terminal, you will get:

```
$ telnet localhost 7654
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
lock,name:django-migrations        <= you write
not-granted                        <= server response
Connection closed by foreign host. <= server closed the connection
```

Here the server responded with **not-granted** and closed the TCP connection. The lock was not granted to you.

But, in real-life scenarios, you would use the provided utility **tcpnetlock_do**:

```
$ tcpnetlock_do --lock-name django-migrations -- python manage.py migrate
```

To test it, you will need the server running. To get the server running with Docker, just run:

```
$ docker pull hgdeoro/tcpnetlock
$ docker run -ti --rm -p 7654:7654 hgdeoro/tcpnetlock
```

Alternatively, you can install the package in a virtualenv and launch the server:

```
$ virtualenv -p python3.6 venv
$ source venv/bin/activate
$ pip install tcpnetlock
$ tcpnetlock_server --info
INFO:root:Started server listening on localhost:7654
```

1.4 Features

- Runs on Python 3.6 / Python 3.5
- Do not require external libraries
- Ready to use Docker image (based on Alpine)
- Includes server and python client
- Includes utility to run Linux commands while holding the lock
- Simple protocol: you can get a lock even with *netcat*

1.5 Appendix: netcat

Since the protocol is just text over a TCP connection, you can get a lock just writing the right text overt the TCP connection and leaving that TCP connection open, and that's the default behaviour of netcat:

```
$ echo 'lock,name:LOCK_NAME' | nc localhost 7654
```

The first line uses netcat to open the TCP connection and tries to get the lock.

The biggest problem would be to READ the response to the server (will be one of 'ok' or 'not-granted') while send *nc* to the background. We can use a *fifo* for that:

```
$ echo 'lock,name:LOCK_NAME' | nc -v localhost 7654 | tee /tmp/.tcpnetlock &
$ result=$(head -n 1 /tmp/.tcpnetlock)
```

Even though this works, using one of the two existing python clients (*tnl_client* and *tnl_do*) would be much better.

1.6 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install TcpNetLock, run this command in your terminal:

```
$ pip install tcpnetlock
```

This is the preferred method to install TcpNetLock, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 Docker image

To run it using Docker, get the image:

```
$ docker image pull hgdeoro/tcpnetlock
```

And then launch a container:

```
$ docker run --rm -ti -p 7654:7654 hgdeoro/tcpnetlock
```

2.3 From sources

The sources for TcpNetLock can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/hgdeoro/tcpnetlock
```

Or download the tarball:

```
$ curl  -OL https://github.com/hgdeoro/tcpnetlock/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use TcpNetLock in a project:

```
import tcpnetlock
```

CHAPTER 4

tcpnetlock

4.1 tcpnetlock package

4.1.1 Subpackages

tcpnetlock.cli package

Submodules

tcpnetlock.cli.common module

class tcpnetlock.cli.common.**BaseMain**

add_app_arguments()

add_logging_arguments()

create_args(*args*)

create_parser()

main()

run(*args*)

setup_logging()

class tcpnetlock.cli.common.**PositiveInteger**(*allow_zero=True*)

4.1.2 Submodules

4.1.3 tcpnetlock.common module

exception tcpnetlock.common.**ClientDisconnected**
Bases: *tcpnetlock.common.TcpNetLockException*

class tcpnetlock.common.**Counter**
Counter, just that. We don't care about atomicity.

count

incr()

exception tcpnetlock.common.**InvalidClientIdError**
Bases: *tcpnetlock.common.TcpNetLockException*

Raised by the client if the provided client-id is not valid.

exception tcpnetlock.common.**InvalidLockNameError**
Bases: *tcpnetlock.common.TcpNetLockException*

Raised by the client if the provided LOCK name is not valid.

exception tcpnetlock.common.**TcpNetLockException**
Bases: exceptions.Exception

Base class for exceptions.

class tcpnetlock.common.**Utils**

static valid_lock_name(*lock_name*)
Returns True if the provided lock name is valid

static validate_client_id(*client_id*, *accept_none=True*)
Raises InvalidClientIdError if client-id is invalid. Pass if it's None

4.1.4 tcpnetlock.constants module

4.1.5 tcpnetlock.protocol module

4.1.6 Module contents

Top-level package for TcpNetLock.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/hgdeoro/tcpnetlock/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

TcpNetLock could always use more documentation, whether as part of the official TcpNetLock docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/hgdeoro/tcpnetlock/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *tcpnetlock* for local development.

1. Fork the *tcpnetlock* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/tcpnetlock.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv tcpnetlock
$ cd tcpnetlock/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 tcpnetlock tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/hgdeoro/tcpnetlock/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst).

Then, sousrce *.bashrc*:

```
source .bashrc
```

And you from there, it's easy. To run tox, coverage, and try to install the package in a new virtualenv:

```
$ tnl pre-release
```

If everything went ok, do the release (this will bump the version):

```
$ tnl release
```

and then, upload to pypi and GitHub:

```
$ tnl upload
```

CHAPTER 6

Credits

6.1 Development Lead

- Horacio G. de Oro <hgdeoro@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.8 (2018-07-20)

- Make period of cleanup configurable via environment variables
- FIX response for stats action
- Add tests for background thread
- Add client method to call the stats action

7.2 0.1.7 (2018-07-19)

- Log release of lock causde by client disconnecting
- Add .stats action
- Add background thread to cleanup old locks
- Remove global server state hold on Context class

7.3 0.1.6 (2018-07-06)

- FIX tag used in Docker image
- log server version when starting

7.4 0.1.5 (2018-06-26)

- FIX variable name used for building Docker image

- Change theme for docs

7.5 0.1.4 (2018-06-26)

- Implements retries for cli tnl_do

7.6 0.1.3 (2018-06-22)

- Server logs granted lock
- Add description to CLI
- Client use environment variables for host/port
- Add __str__ to Action to have better logs in server

7.7 0.1.2 (2018-06-21)

- Update client & server to handle errors in a better way
- Add tests
- Update docs

7.8 0.1.1 (2018-06-20)

- Add .bashrc (for developers)
- Fix setup.py

7.9 0.1.0 (2018-06-19)

- Docker start server with –info by default
- Adds cloudbuild.yaml to facilitate building in GCP
- Change in protocol to detect unintended uses
- Detect invalid requests and always send response to client
- BIG refactor of server and client classes
- Add lot of tests (current coverage: 99%)

7.10 0.0.8 (2018-06-18)

- Refactor messy code from server, client and cli

7.11 0.0.7 (2018-06-17)

- Code cleanup and refactor
- Add tests
- Implements run_with_lock script to make really easy to use from shell scripts

7.12 0.0.6 (2018-06-16)

- Create shell script to be sourced, to facilitate use of tcpnetlock from shell scripts

7.13 0.0.5 (2018-06-15)

- Update CONTRIBUTING (documents commands for the full release process)
- Disable upload to pypi from Travis-CI

7.14 0.0.4 (2018-06-15)

- Encapsulate Lock, adds client id and timestamp
- Implement sending of keepalive from client
- Remove use of 'click'
- Start server from cli with configurable parameters (listen address, port, etc)
- Use client id to identify who has the lock

7.15 0.0.3 (2018-06-15)

- Validate lock name in server
- FIX client to handle RESPONSE_ERR response
- Add unittests
- Refactor locks into server class
- Use threading for test server
- Make code compatible with Python 3.5

7.16 0.0.2 (2018-06-15)

- Implements RELEASE of locks
- FIX release of lock when client closes the connection
- Validates lock name
- Code refactoring

7.17 0.0.1 (2018-06-15)

- Add files from cookiecutter-pypackage
- Migrate test cases to pytest

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index