cynic Documentation

Release 1.0

Ruslan Spivak

Sep 27, 2017

Contents

CHAPTER 1

What is Cynic?

These days almost any application has several integration points like database, payment gateway, or some Web service that it consumes over HTTP.

All communication with the remote systems happens over the network and both networks and those systems often go wonky.

If we do not test the behavior of **our** system when the remote end operates out of spec and goes haywire the only place for testing becomes in production which is, as we all know, for some systems is less than acceptable.

Because the calls to the remote systems use network, the socket connection can have different failure scenarios, for example:

- The remote end resets the connection by sending a TCP RST packet
- The connection may be established, but the response is never sent back and the connection is not closed (If you don't use socket timeouts in your app you may be in trouble at some point).
- The remote end can send garbage data as the response
- The service can send HTML over HTTP instead of the expected JSON response
- The HTTP service can send one byte of the response data every 30 seconds
- The remote HTTP service sends only headers and no body
- The service can send megabytes of data instead of expected kilobytes
- Etc.

It would be good to be able to test the behavior of our application when some of those conditions happen.

Cynic tries to help with that testing. Basically it's a test harness (test double) that can be used to simulate crafty and devious remote systems right from your command line.

Cynic will try hard to cause injury to your system.

Its goal is to make your system under test cynical.

Read the formatted docs at http://cynic.readthedocs.org

WATCH SCREENCAST - https://vimeo.com/43375697

CHAPTER 2

Installation

```
$ [sudo] pip install cynic
```

Or the bleeding edge version from the git master branch:

```
$ [sudo] pip install git+https://github.com/rspivak/cynic.git#egg=cynic
```

CHAPTER 3

Quick intro

Start Cynic which in turn will start multiple services on different ports:

```
$ cynic
INFO     [2012-06-03 23:44:35,603] server: Starting 'HTTPHtmlResponse'   on port 2000
INFO     [2012-06-03 23:44:35,603] server: Starting 'HTTPJsonResponse'   on port 2001
INFO     [2012-06-03 23:44:35,604] server: Starting 'HTTPNoBodyResponse' on port 2002
INFO     [2012-06-03 23:44:35,604] server: Starting 'HTTPSlowResponse'   on port 2003
INFO     [2012-06-03 23:44:35,604] server: Starting 'RSTResponse'        on port 2020
INFO     [2012-06-03 23:44:35,604] server: Starting 'RandomDataResponse' on port 2021
INFO     [2012-06-03 23:44:35,604] server: Starting 'NoResponse'         on port 2022
INFO     [2012-06-03 23:44:35,604] server: Starting 'LogRecordHandler'   on port /tmp/
↪_cynic.sock
```

CHAPTER 4

Test different services

1. Connect to the service on port 2020 and get a TCP RST packet right away which causes 'Connection reset by peer' message on the command line

```
$ curl http://localhost:2020
curl: (56) Recv failure: Connection reset by peer
```

2. Connect to the service on port 2021 and get back 7 bytes of random data

```
$ telnet localhost 2021
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
#6
Connection closed by foreign host.
```

3. Connect to the service on port 2001 to get the HTTP JSON response

```
$ curl http://localhost:2001
{"message": "Hello, World!"}
```

So you got the basic idea. Point your application's integration points to the Cynic's services and see how your app reacts.

Let's check out Cynic's help options

```
$ cynic -h
Usage: cynic [options]

Test harness to make your system under test cynical

Options:
  -h, --help            show this help message and exit
  -c CONFIG_PATH, --config=CONFIG_PATH
                        Path to an INI configuration file. If no configuration
                        file is provided then default configuration is
                        applied. To see the default configuration use -d
                        option described below.
  -d, --dump            Dump default configuration to STDOUT
```

As you can see if we start **Cynic** without **-c** option the default configuration is used and **-d** is used to dump that configuration to the standard out.

Let's see what's included in the default configuration

Default configuration

```
$ cynic -d

############################################################
# HTTP protocol specific                                   #
############################################################

[handler:httphtml]
# sends simple 'hello world!' HTML page over HTTP as a response
# and terminates
class = cynic.handlers.httphtml.HTTPHtmlResponse
#args = ('/tmp/test.html', )
host = 0.0.0.0
port = 2000

[handler:httpjson]
# sends simple 'hello world!' JSON over HTTP as a response
# and terminates
class = cynic.handlers.httpjson.HTTPJsonResponse
#args = ('/tmp/test.json', )
host = 0.0.0.0
port = 2001

[handler:httpnone]
# sends headers, but not the response body and terminates
class = cynic.handlers.httpnone.HTTPNoBodyResponse
host = 0.0.0.0
port = 2002

[handler:httpslow]
# sends one byte of the response every 30 seconds.
# when the data to be sent is exhausted - terminates
class = cynic.handlers.httpslow.HTTPSlowResponse
#args = ('/tmp/test.json', 'application/json', 1)
host = 0.0.0.0
port = 2003


############################################################
# Any TCP socket protocol                                  #
############################################################

[handler:reset]
```

```
# accepts a connection, sends an RST packet right away
# and terminates
class = cynic.handlers.reset.RSTResponse
host = 0.0.0.0
port = 2020

[handler:random]
# accepts a connection, sends 7 bytes from the /dev/urandom device
# and terminates
class = cynic.handlers.rnd.RandomDataResponse
host = 0.0.0.0
port = 2021

[handler:noresp]
# accepts a connection, but doesn't send any response back.
# sleeps for 24 hours and exits
class = cynic.handlers.noresp.NoResponse
host = 0.0.0.0
port = 2022

############################################################
# System handlers used internally by the Cynic server      #
############################################################

[handler:unixlog]
# a logging server that accepts connections over Unix socket
# from multiple local processes to output passed log records
class = cynic.handlers.log.LogRecordHandler
host = /tmp/_cynic.sock
port = 0
family = unix
```

There are basically two types of handlers:

1. The ones that deal with any TCP socket protocol
2. HTTP specific handlers

Let's have a closer look at some of them.

CHAPTER 5

cynic.handlers.httpslow.HTTPSlowResponse

This handler sends one byte of the HTTP response every 30 seconds. The config part is as follows

```
[handler:httpslow]
# sends one byte of the response every 30 seconds.
# when the data to be sent is exhausted - terminates
class = cynic.handlers.httpslow.HTTPSlowResponse
#args = ('/tmp/test.json', 'application/json', 1)
host = 0.0.0.0
port = 2003
```

where

class - a fully qualified dotted Python name of the handler class

args - a tuple of Python values to pass as positional arguments to the handler's constructor.

> **First argument** specifies absolute path to a file to read the data from instead of using a default data sting 'Hello, world!'
>
> **Second argument** specifies the value of HTTP's Content-Type response header
>
> **Third argument** specifies time interval in seconds, default is 30, after which additional byte is sent to the client

host - an IP address to bind the service to (For Unix socket it's a file path)

port - port to listen on (not applicable for Unix sockets)

Even with this service alone you can be creative and come up with several test scenarios that will make the life of your system under test quite unbearable:

1. Specify file in the *args* that contains megabytes of data and see how your system handles such a large response
2. You can change file path and content type arguments to send HTML, JSON, XML, Plain text, etc
3. You can send HTML data but set Content-Type header value to *application/json*
4. You can change time interval to test your socket read timeout expiration or lack thereof.

5. You can have all above as separate services on different ports. Just add *[handler:httpslow1]*, *[handler:httpslow2]*, etc. sections to the INI file and tweak the *args*.

Extending Cynic with custom handlers

It's very easy to add your own handler to the Cynic.

1. To add a new TCP handler inherit from *cynic.handlers.base.BaseHandler* and implement the *handle* method which directly interacts with a TCP socket.
2. To add a new HTTP handler inherit from *cynic.handlers.base.BaseHTTPHandler* and implement your custom do_GET, do_POST, do_PUT, etc methods. For more information about the handler see BaseHTTPRequest-Handler
3. Add a section *[handler:my_new_name]* to the INI configuration file with corresponding configuration parameters.

XXX: Full example?

Acknowledgments

- Many ideas are taken from Release It!

License

CHAPTER 6

Indices and tables

- genindex
- modindex
- search