Yandex.Tank Documentation

Release 1.11.2

Yandex

Jun 08, 2019

Contents

Author Alexey Lavrenuke

Version 1.11.2

Date Jun 08, 2019

Homepage Yandex.Tank Homepage on Github

Download Launchpad PPA Pypi

Documentation PDF Documentation

License GNU LGPLv3

Issue tracker GitHub Issues

Contents:

CHAPTER 1

Getting started

Welcome to Yandex.Tank documentation. Yandex.Tank is an extensible load testing utility for unix systems. It is written in Python and uses different load generator modules in different languages.

1.1 Getting Help

Gitter.im

1.2 What are the Yandex.Tank components?

- `Core` - basic steps of test prepare, configuration, execution. Artifacts storing. Controls plugins/modules.
- `Load generators` - modules that uses and controls load generators (load generators NOT included).
- `Artifact uploaders` - modules that uploads artifacts to external storages and services.
- `Handy tools` - monitoring tools, console online screen, autostops and so on.

Note: Using `phantom` as a load generator for mild load tests (less then 1000rps) an average laptop with 64bit Ubuntu (10.04/. . . /13.10) would be sufficient. The tank could be easily used in virtual machine if queries aren't too heavy and load isn't too big. Otherwise it is recommended to request a physical server or a more capable virtual machine from your admin.

1.3 Running Yandex.Tank

1.Install tank to your system *Installation*

2.Tune your system *Routing and firewall*

3.And run the tutorial *Tutorials*

4.If you are skilled enough, feel free to use *Advanced usage*.

5.For developers *Modules*.

1.4 See also

Evgeniy Mamchits' phantom - Phantom scalable IO Engine

Alexey Lavrenuke's pandora - A load generator in Go language

Gregory Komissarov's firebat - test tool based on Phantom

BlazeMeter's Sense - service for storing and analysing performance test results

CHAPTER 2

Installation

2.1 Docker container

Install docker and use `direvius/yandex-tank` (or, if you need jmeter, try `direvius/yandex-tank:jmeter-latest`) container. Default entrypoint is `/usr/local/bin/yandex-tank` so you may just run it to start test:

```
docker run \
    -v $(pwd):/var/loadtest \
    -v $SSH_AUTH_SOCK:/ssh-agent -e SSH_AUTH_SOCK=/ssh-agent \
    --net host \
    -it direvius/yandex-tank
```

- `$(pwd):/var/loadtest` - current directory mounted to /var/loadtest in container to pass data for test (config file, monitoring config, ammo, etc)
- tank will use load.yaml from current directory as default config, append `-c custom-config-name.yaml` to run with other config
- you may pass other additional parameters for tank in run command, just append it after image name
- `$SSH_AUTH_SOCK:/ssh-agent` - ssh agent socket mounted in order to provide use telegraf plugin (monitoring). It uses your ssh keys to remotely login to monitored hosts

If you want to do something in the container before running tank, you will need to change entrypoint:

```
docker run \
    -v $(pwd):/var/loadtest \
    -v $SSH_AUTH_SOCK:/ssh-agent -e SSH_AUTH_SOCK=/ssh-agent \
    --net host \
    -it \
    --entrypoint /bin/bash \
    direvius/yandex-tank
```

Start test Within container with yandex-tank command:

```
yandex-tank -c config-name.yaml # default config is load.yaml
```

2.2 Installation from PyPi

These are the packages that are required to build different python libraries. Install them with *apt*:

```
sudo apt-get install python-pip build-essential python-dev libffi-dev gfortran libssl-
↪dev
```

Update your pip:

```
sudo -H pip install --upgrade pip
```

Update/install your setuptools:

```
sudo -H pip install --upgrade setuptools
```

Install latest Yandex.Tank from master branch:

```
sudo -H pip install https://api.github.com/repos/yandex/yandex-tank/tarball/master
```

You'll probably need Phantom load generator, so install it from our ppa:

```
sudo add-apt-repository ppa:yandex-load/main && sudo apt-get update
sudo apt-get install phantom phantom-ssl
```

2.3 Installation .deb packages

Note: **Deprecated**. Deb packages aren't renewed in PPA.

You should add proper repositories on Debian-based environment.

For instance, add following repos to `sources.list` :

```
deb http://ppa.launchpad.net/yandex-load/main/ubuntu trusty main
deb-src http://ppa.launchpad.net/yandex-load/main/ubuntu trusty main
```

or this way

```
sudo apt-get install python-software-properties
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:yandex-load/main
```

Then update package list and install `yandex-tank` package:

```
sudo apt-get update && sudo apt-get install yandex-tank
```

CHAPTER 3

Routing and firewall

3.1 Firewall

Before test execution, please, check service availability. If service is running on server with IP `x.x.x.x` and listening for TCP port `zz`, try to connect to it with `telnet` like this: `telnet x.x.x.x zz` If everything OK, you'll see:

```
$ telnet 203.0.113.1 80
Trying 203.0.113.1...
Connected to 203.0.113.1. Escape character is '^]'.
```

Otherwise if port is unreachable:

```
$ telnet 203.0.113.1 80 Trying 203.0.113.1...
telnet: Unable to connect to remote host: Connection timed out
```

Note: it's just an example, programs like `nc/nmap/wget/curl` could be used as well, but not ping!)

3.2 Routing

OK, the service is reachable, next thing you should know is how far Yandex.Tank is located from the service you'd like to test. Heavy load can make switch to be unresponsible or to reboot, or at least it may lead to network losses, so the test results would be distorted. Be careful. Path estimation could be done by execution of `tracepath` command or it analogs (`tracert/traceroute`) on Yandex.Tank machine:

```
$ tracepath 203.0.113.1
1:  tank.example.com (203.0.113.1)             0.084ms pmtu 1450
1:  target.load.example.com (203.0.113.1)           20.919ms reached
1:  target.example.com (203.0.113.1)             0.128ms reached
Resume: pmtu 1450 hops 1 back 64``
```

(continues on next page)

(continued from previous page)

```
Hops count = 1 means that tank and target are in closest location.

$ tracepath 24.24.24.24
1:  1.example.com (203.0.113.1)                0.084ms pmtu 1450
1:  2.example.com (203.0.113.1)          0.276ms
1:  3.example.com (203.0.113.1)          0.411ms
2:  4.example.com (203.0.113.1)               0.514ms
3:  5.example.com (203.0.113.1)              10.690ms
4:  6.example.com (203.0.113.1)                0.831ms asymm  3
5:  7.example.com (203.0.113.1)                0.512ms
6:  8.example.com (203.0.113.1)                0.525ms asymm  5
7:  no reply
```

In the second example you'd better find another closer located tank.

3.3 Tuning

To achieve top performance you should tune the source server system limits:

```
ulimit -n 30000

net.ipv4.tcp_max_tw_buckets = 65536
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 0
net.ipv4.tcp_max_syn_backlog = 131072
net.ipv4.tcp_syn_retries = 3
net.ipv4.tcp_synack_retries = 3
net.ipv4.tcp_retries1 = 3
net.ipv4.tcp_retries2 = 8
net.ipv4.tcp_rmem = 16384 174760 349520
net.ipv4.tcp_wmem = 16384 131072 262144
net.ipv4.tcp_mem = 262144 524288 1048576
net.ipv4.tcp_max_orphans = 65536
net.ipv4.tcp_fin_timeout = 10
net.ipv4.tcp_low_latency = 1
net.ipv4.tcp_syncookies = 0
net.netfilter.nf_conntrack_max = 1048576
```

Note: `tcp_tw_recycle` has been removed as of Linux 4.12.

This is because Linux now randomizes timestamps per connection and they do not monotonically increase. If you're using Linux 4.12 with machines using tcp_tw_recycle and TCP timestamps are turned on you will see dropped connections. You can of course disable it like so *echo 0 > /proc/sys/net/ipv4/tcp_timestamps* (temporarily, use sysctl.conf for permanent changes).

Details on 4.12 removing tcp_tw_recycle: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=4396e46187ca5070219b81773c4e65088dac50cc

CHAPTER 4

Tutorials

So, you've installed Yandex.Tank to a proper machine, it is close to target, access is permitted and server is tuned. How to make a test?

Note: This guide is for `phantom` load generator.

Create a file on a server with Yandex.Tank: **load.yaml**

```
phantom:
  address: 203.0.113.1:80 # [Target's address]:[target's port]
  uris:
    - /
  load_profile:
    load_type: rps # schedule load by defining requests per second
    schedule: line(1, 10, 10m) # starting from 1rps growing linearly to 10rps during␣
↪10 minutes
console:
  enabled: true # enable console output
telegraf:
  enabled: false # let's disable telegraf monitoring for the first time
```

And run: `$ yandex-tank -c load.yaml`

`phantom` have 3 primitives for describing load scheme:

1. `step (a,b,step,dur)` makes stepped load, where a,b are start/end load values, step - increment value, dur - step duration.

   Examples:

   - `step(25, 5, 5, 60)` - stepped load from 25 to 5 rps, with 5 rps steps, step duration 60s.
   - `step(5, 25, 5, 60)` - stepped load from 5 to 25 rps, with 5 rps steps, step duration 60s

2. `line (a,b,dur)` makes linear load, where `a,b` are start/end load, `dur` - the time for linear load increase from a to b.

   Examples:

   - `line(10, 1, 10m)` - linear load from 10 to 1 rps, duration - 10 minutes
   - `line(1, 10, 10m)` - linear load from 1 to 10 rps, duration - 10 minutes

3. `const (load,dur)` makes constant load. `load` - rps amount, `dur` - load duration.

   Examples:

   - `const(10,10m)` - constant load for 10 rps for 10 minutes.
   - `const(0, 10)` - 0 rps for 10 seconds, in fact 10s pause in a test.

Note:

You can set fractional load like this: `line(1.1, 2.5, 10)` - from 1.1rps to 2.5 for 10 seconds.

Note: `step` and `line` could be used with increasing and decreasing intensity:

You can specify complex load schemes using those primitives.

Example: `schedule: line(1, 10, 10m) const(10,10m)`

linear load from 1 to 10rps during 10 minutes, then 10 minutes of 10rps constant load.

Time duration could be defined in seconds, minutes (m) and hours (h). For example: `27h103m645`

For a test with constant load at 10rps for 10 minutes, `load.yaml` should have following lines:

```
phantom:
  address: 203.0.113.1:80 # [Target's address]:[target's port]
  uris:
    - /uri1
    - /uri2
  load_profile:
    load_type: rps # schedule load by defining requests per second
    schedule: const(10, 10m) # starting from 1rps growing linearly to 10rps during 10
↪minutes
console:
  enabled: true # enable console output
telegraf:
  enabled: false # let's disable telegraf monitoring for the first time
```

4.1 Preparing requests

There are several ways to set up requests:

- Access mode
- URI-style
- URI+POST
- request-style.

Note: Request-style is default ammo type.

Note: Regardless of the chosen format, resulted file with requests could be gzipped - tank supports archived ammo files.

To specify external ammo file use `ammofile` option.

Note: You can specify URL to ammofile, http(s). Small ammofiles (~<100MB) will be downloaded as is, to directory `/tmp/<hash>`, large files will be read from stream.

Note: If ammo type is uri-style or request-style, tank will try to guess it. Use `ammo_type` option to explicitly specify ammo format. Don't forget to change `ammo_type` option if you switch format of your ammo, otherwise you might get errors.

Example:

```
phantom:
  address: 203.0.113.1:80
  ammofile: https://yourhost.tld/path/to/ammofile.txt
```

4.1.1 URI-style, URIs in load.yaml

YAML-file configuration: Don't specify `ammo_type` explicitly for this type of ammo.

Update configuration file with HTTP headers and URIs:

```
phantom:
  address: 203.0.113.1:80
  load_profile:
    load_type: rps
    schedule: line(1, 10, 10m)
  header_http: "1.1"
  headers:
    - "[Host: www.target.example.com]"
    - "[Connection: close]"
  uris:
    - "/uri1"
    - "/buy"
    - "/sdfg?sdf=rwerf"
    - "/sdfbv/swdfvs/ssfsf"
console:
  enabled: true
telegraf:
  enabled: false
```

Parameter `uris` contains uri, which should be used for requests generation.

Note: Pay attention to the sample above, because whitespaces in multiline `uris` and `headers` options are important.

4.1.2 URI-style, URIs in file

YAML-file configuration: `ammo_type: uri`

Create a file with declared requests: `ammo.txt`

```
[Connection: close]
[Host: target.example.com]
[Cookie: None]
/?drg tag1
/
/buy tag2
[Cookie: test]
/buy/?rt=0&station_to=7&station_from=9
```

File consists of list of URIs and headers to be added to every request defined below. Every URI must begin from a new line, with leading `/`. Each line that begins from `[` is considered as a header. Headers could be (re)defined in the middle of URIs, as in sample above.

Example: Request `/buy/?rt=0&station_to=7&station_from=9` will be sent with `Cookie: test`, not `Cookie: None`.

Request may be marked by tag, you can specify it with whitespace following URI.

4.1.3 URI+POST-style

YAML-file configuration: `ammo_type: uripost`

Create a file with declared requests: `ammo.txt`

```
[Host: example.org]
[Connection: close]
[User-Agent: Tank]
5 /route/?rll=50.262025%2C53.276083~50.056015%2C53.495561&origin=1&simplify=1
class
10 /route/?rll=50.262025%2C53.276083~50.056015%2C53.495561&origin=1&simplify=1
hello!clas
7 /route/?rll=37.565147%2C55.695758~37.412796%2C55.691454&origin=1&simplify=1
uripost
```

File begins with optional lines [...], that contain headers which will be added to every request. After that section there is a list of URIs and POST bodies. Each URI line begins with a number which is the size of the following POST body.

4.1.4 Request-style

YAML-file configuration: `ammo_type: phantom`

Full requests listed in a separate file. For more complex requests, like POST, you'll have to create a special file. File format is:

```
[size_of_request] [tag]\n
[request_headers]
[body_of_request]\r\n
[size_of_request2] [tag2]\n
[request2_headers]
[body_of_request2]\r\n
```

where `size_of_request` – request size in bytes. 'rn' symbols after `body` are ignored and not sent anywhere, but it is required to include them in a file after each request. Pay attention to the sample above because 'r' symbols are strictly required.

Note: Parameter `ammo_type` is unnecessary, request-style is default ammo type.

sample GET requests (null body)

```
73 good
GET / HTTP/1.0
Host: xxx.tanks.example.com
User-Agent: xxx (shell 1)

77 bad
GET /abra HTTP/1.0
Host: xxx.tanks.example.com
User-Agent: xxx (shell 1)

78 unknown
GET /ab ra HTTP/1.0
Host: xxx.tanks.example.com
User-Agent: xxx (shell 1)
```

sample POST requests (binary data)

```
904
POST /upload/2 HTTP/1.0
Content-Length: 801
Host: xxxxxxxxx.dev.example.com
User-Agent: xxx (shell 1)

^.^........W.j^1^.^.^.²..^^.i.^B.P..-!(.l/Y..V^.      ...L?...S'NR.^^vm...3Gg@s...d'.\^
↪.5N.$NF^,.Z^.aTE^.
._.[..k#L^`\RE.J.<.!,.q5.F^iΔĬq..^6..P..H.`..i2
.".uuzs^^F2...Rh.&.U.^^..J.P@.A......x..ly^?.u.p{4..g...m.,..R^.^.^......].^^.^J...p.
↪ifTF0<.s.9V.o5<..%!6ļS.Æ......C^&.....^.^y...v]^YT.1.#K.ibc...^.26...   ..7.
b.$...j6.f...W.R7.^1.3....K`%.&^..d..{{      l0..^\..^X.g.^.r.(!.^^...4.1.$\ .%.8$(.n&
↪..^^q.,.Q..^.D^.].^.R9.kE.^.$^.I..<..B^..^.h^^C.^E.|....3o^.@..Z.^.s.$[v.
527
POST /upload/3 HTTP/1.0
Content-Length: 424
Host: xxxxxxxxx.dev.example.com
User-Agent: xxx (shell 1)

^.^........QMO.0^.++^zJw.^$^.^.^V.J....vM.8r&.T+...{@pk%~C.G../z^.7....l...-.^W"cR....
↪. .&^?u.U^^.^.....{^.^..8.^.^.I.EĂ.p...'^.3.Tq..@R8....RAiBU..1.Bd*".7+.
.Ol.j=^.3..n....wp..,Wg.y^.T..~^..
```

sample POST multipart:

```
533
POST /updateShopStatus? HTTP/1.0
User-Agent: xxx/1.2.3
Host: xxxxxxxxx.dev.example.com
Keep-Alive: 300
Content-Type: multipart/form-data; boundary=AGHTUNG
Content-Length:334
Connection: Close

--AGHTUNG
Content-Disposition: form-data; name="host"

load-test-shop-updatestatus.ru
--AGHTUNG
Content-Disposition: form-data; name="user_id"

1
--AGHTUNG
Content-Disposition: form-data; name="wsw-fields"

<wsw-fields><wsw-field name="moderate-code"><wsw-value>disable</wsw-value></wsw-field>
↪</wsw-fields>
--AGHTUNG--
```

sample ammo generators you may find on the *Ammo generators* page.

4.2 Run Test!

1. Request specs in load.yaml – run as `yandex-tank -c load.yaml`
2. Request specs in ammo.txt – run as `yandex-tank -c load.yaml ammo.txt`

Yandex.Tank detects requests format and generates ultimate requests versions.

`yandex-tank` here is an executable file name of Yandex.Tank.

If Yandex.Tank has been installed properly and configuration file is correct, the load will be given in next few seconds.

4.3 Results

During test execution you'll see HTTP and net errors, answer times distribution, progressbar and other interesting data. At the same time file `phout.txt` is being written, which could be analyzed later.

If you need more human-readable report, you can try Report plugin, You can found it here

If you need to upload results to an external storage, such as Graphite or InfluxDB, you can use one of existing artifacts uploading modules *Modules*

4.4 Tags

Requests could be grouped and marked by some tag.

Example:

```
73 good
GET / HTTP/1.0
Host: xxx.tanks.example.com
User-Agent: xxx (shell 1)

77 bad
GET /abra HTTP/1.0
Host: xxx.tanks.example.com
User-Agent: xxx (shell 1)

75 unknown
GET /ab HTTP/1.0
Host: xxx.tanks.example.com
User-Agent: xxx (shell 1)
```

`good`, `bad` and `unknown` here are the tags.

Note: RESTRICTION: utf-8 symbols only

4.5 SSL

To activate SSL add `phantom: {ssl: true}` to `load.yaml`. Now, our basic config looks like that:

```
phantom:
  address: 203.0.113.1:443
    load_profile:
      load_type: rps
      schedule: line(1, 10, 10m)
  ssl: true
```

Note: Do not forget to specify ssl port to *address*. Otherwise, you might get 'protocol errors'.

4.6 Autostop

Autostop is an ability to automatically halt test execution if some conditions are reached.

4.6.1 HTTP and Net codes conditions

There is an option to define specific codes (404,503,100) as well as code groups (3xx, 5xx, xx). Also you can define relative threshold (percent from the whole amount of answer per second) or absolute (amount of answers with specified code per second).

Examples:

`autostop: http(4xx,25%,10)` – stop test, if amount of 4xx http codes in every second of last 10s period exceeds 25% of answers (relative threshold).

`autostop: net(101,25,10)` – stop test, if amount of 101 net-codes in every second of last 10s period is more than 25 (absolute threshold).

`autostop: net(xx,25,10)` – stop test, if amount of non-zero net-codes in every second of last 10s period is more than 25 (absolute threshold).

4.6.2 Average time conditions

Example: `autostop: time(1500,15)` – stops test, if average answer time exceeds 1500ms.

So, if we want to stop test when all answers in 1 second period are 5xx plus some network and timing factors - add autostop line to load.yaml:

```
phantom:
  address: 203.0.113.1:80
  load_profile:
    load_type: rps
    schedule: line(1, 10, 10m)
autostop:
  autostop:
    - time(1s,10s)
    - http(5xx,100%,1s)
    - net(xx,1,30)
```

4.7 Logging

Looking into target's answers is quite useful in debugging. For doing that use parameter writelog, e.g. add `phantom: {writelog: all}` to `load.yaml` to log all messages.

Note: Writing answers on high load leads to intensive disk i/o usage and can affect test accuracy.**

Log format:

```
<metrics>
<body_request>
<body_answer>
```

Where metrics are:

`size_in size_out response_time(interval_real) interval_event net_code` (request size, answer size, response time, time to wait for response from the server, answer network code)

Example:

```
user@tank:~$ head answ_*.txt
553 572 8056 8043 0
GET /create-issue HTTP/1.1
Host: target.yandex.net
User-Agent: tank
Accept: */*
Connection: close


HTTP/1.1 200 OK
Content-Type: application/javascript;charset=UTF-8
```

For `load.yaml` like this:

```
phantom:
  address: 203.0.113.1:80
  load_profile:
    load_type: rps
    schedule: line(1, 10, 10m)
  writelog: all
autostop:
  autostop:
    - time(1,10)
    - http(5xx,100%,1s)
    - net(xx,1,30)
```

4.8 Results in phout

phout.txt - is a per-request log. It could be used for service behaviour analysis (Excel/gnuplot/etc) It has following fields: `time, tag, interval_real, connect_time, send_time, latency, receive_time, interval_event, size_out, size_in, net_code proto_code`

Phout example:

```
1326453006.582          1510    934     52      384     140     1249    37      478
↪  0       404
1326453006.582   others       1301    674     58      499     70      1116    37
↪478     0       404
1326453006.587   heavy       377     76      33      178     90      180     37
↪478     0       404
1326453006.587          294     47      27      146     74      147     37      478
↪  0       404
1326453006.588          345     75      29      166     75      169     37      478
↪  0       404
1326453006.590          276     72      28      119     57      121     53      476
↪  0       404
1326453006.593          255     62      27      131     35      134     37      478
↪  0       404
1326453006.594          304     50      30      147     77      149     37      478
↪  0       404
1326453006.596          317     53      33      158     73      161     37      478
↪  0       404
1326453006.598          257     58      32      106     61      110     37      478
↪  0       404
1326453006.602          315     59      27      160     69      161     37      478
↪  0       404
1326453006.603          256     59      33      107     57      110     53      476
↪  0       404
1326453006.605          241     53      26      130     32      131     37      478
↪  0       404
```

Note: contents of phout depends on phantom version installed on your Yandex.Tank system.

4.9 Graph and statistics

Use Report plugin OR use your favorite stats packet, R, for example.

4.10 Thread limit

`instances: N` in `load.yaml` limits number of simultanious connections (threads).

Example with 10 threads limit:

```
phantom:
  address: 203.0.113.1:80
  load_profile:
    load_type: rps
    schedule: line(1, 10, 10m)
  instances: 10
```

4.11 Dynamic thread limit

You can specify `load_type: instances` instead of 'rps' to schedule a number of active instances which generate as much rps as they manage to. Bear in mind that active instances number cannot be decreased and final number of them must be equal to `instances` parameter value.

Example:

```
phantom:
  address: 203.0.113.1:80
  load_profile:
    load_type: instances
    schedule: line(1,10,10m)
  instances: 10
  loop: 10000 # don't stop when the end of ammo is reached but loop it 10000 times
```

Note: When using `load_type: instances` you should specify how many loops of ammo you want to generate because tank can't find out from the schedule how many ammo do you need

4.12 Custom stateless protocol

In necessity of testing stateless HTTP-like protocol, Yandex.Tank's HTTP parser could be switched off, providing ability to generate load with any data, receiving any answer in return. To do that use tank_type parameter:

```
phantom:
  address: 203.0.113.1:80
  load_profile:
    load_type: rps
    schedule: line(1, 10, 10m)
  instances: 10
  tank_type: none
```

Note: Indispensable condition: Connection close must be initiated by remote side

4.13 Gatling

If server with Yandex.Tank have several IPs, they may be used to avoid outcome port shortage. Use `gatling_ip` parameter for that. load.yaml:

```
phantom:
   address: 203.0.113.1:80
   load_profile:
     load_type: rps
     schedule: line(1, 10, 10m)
   instances: 10
   gatling_ip: IP1 IP2
```

CHAPTER 5

Advanced usage

5.1 Command line options

Yandex.Tank has an obviously named executable `yandex-tank`. Here are available command line options:

-h, –help show command line options

-c CONFIG, –config=CONFIG Read options from yaml file. It is possible to set multiple configuration files by specifying the option serveral times.

Default: `./load.yaml`

-i, –ignore-lock Ignore lock files.

-f, –fail-lock Don't wait for lock file, quit if it's busy.

Default behaviour is to wait for lock file to become free

-l LOG, –log=LOG Main log file location.

Default: `./tank.log`

-m, –manual-start Tank will prepare for test and wait for Enter key to start the test.

-n, –no-rc Don't read `/etc/yandex-tank/*.ini` and `~/.yandex-tank`

-o OPTION, –option=OPTION Set an option from command line. Options set in cmd line override those have been set in configuration files. Multiple times for multiple options.

Format: `<section>.<option>=value`

Example: `yandex-tank -o "console.short_only=1" --option="phantom.force_stepping=1"`

-s SCHEDULED_START, –scheduled-start=SCHEDULED_START Run test on specified time, date format YYYY-MM-DD hh:mm:ss or hh:mm:ss

-q, –quiet Only print WARNINGs and ERRORs to console.

-v, –verbose Print ALL, including DEBUG, messages to console. Chatty mode.

Add an ammo file name as a nameless parameter, e.g.: `yandex-tank ammo.txt` or `yandex-tank ammo.gz`

5.2 Advanced configuration

Configuration files organized as yaml-files. Those are files partitioned into named sections that contain 'name: value' records.

Example:

```
phantom:
  address: example.com:80
  load_profile:
    load_type: rps
    schedule: const(100,60s)
autostop:
  autostop:
    - instances(80%,10)
    - time(1s,10s)
```

Note: A common rule: options with the same name override those set before them (in the same file or not).

5.2.1 Default configuration files

Note: `--no-rc` command line option disables this behavior

Yandex.Tank reads all `*.yaml` from `/etc/yandex-tank` directory, then a personal config file `~/.yandex-tank`. So you can easily put your favourite settings in `~/.yandex-tank`

Example: `tank.artifacts_base_dir`, `phantom.cache_dir`, `console.info_panel_width`

5.2.2 Multiline options

Use indent to show that a line is a continuation of a previous one:

```
autostop:
  autostop:
    - time(1,10)
    - http(404,1%,5s)
    - net(xx,1,30)
```

Note: Ask Yandex.Tank developers to add multiline capability for options where you need it!*

5.2.3 Time units

Default : milliseconds.

Example:

```
``30000 == 30s``
``time(30000,120)`` is an equivalent to ``time(30s,2m)``
```

Time units encoding is as following:

Abbreviation	Meaning
ms	millisecons
s	seconds
m	minutes
h	hours

Note: You can also mix them: `1h30m15s` or `2s15ms`.

5.3 Artifacts

As a result Yandex.Tank produces some files (logs, results, configs etc). Those files are placed with care to the **artifact directory**. An option for that is `artifacts_base_dir` in the `tank` section. It is recommended to set it to a convenient place, for example, `~/yandex-tank-artifacts`; it would be easier to manage the artifacts there.

5.4 Sources

Yandex.Tank sources are here.

5.5 load.yaml example

```
phantom:
  address: "ya.ru:80"
  instances: 1000
  load_profile:
    load_type: rps
    schedule: const(1,30) line(1,1000,2m) const(1000,5m)
  header_http: "1.1"
  uris:
    - "/"
    - "/test"
    - "/test2"
  headers:
    - "[Host: www.ya.ru]"
    - "[Connection: close]"
autostop:
  autostop:
    - http(5xx,10%,5s)
```

CHAPTER 6

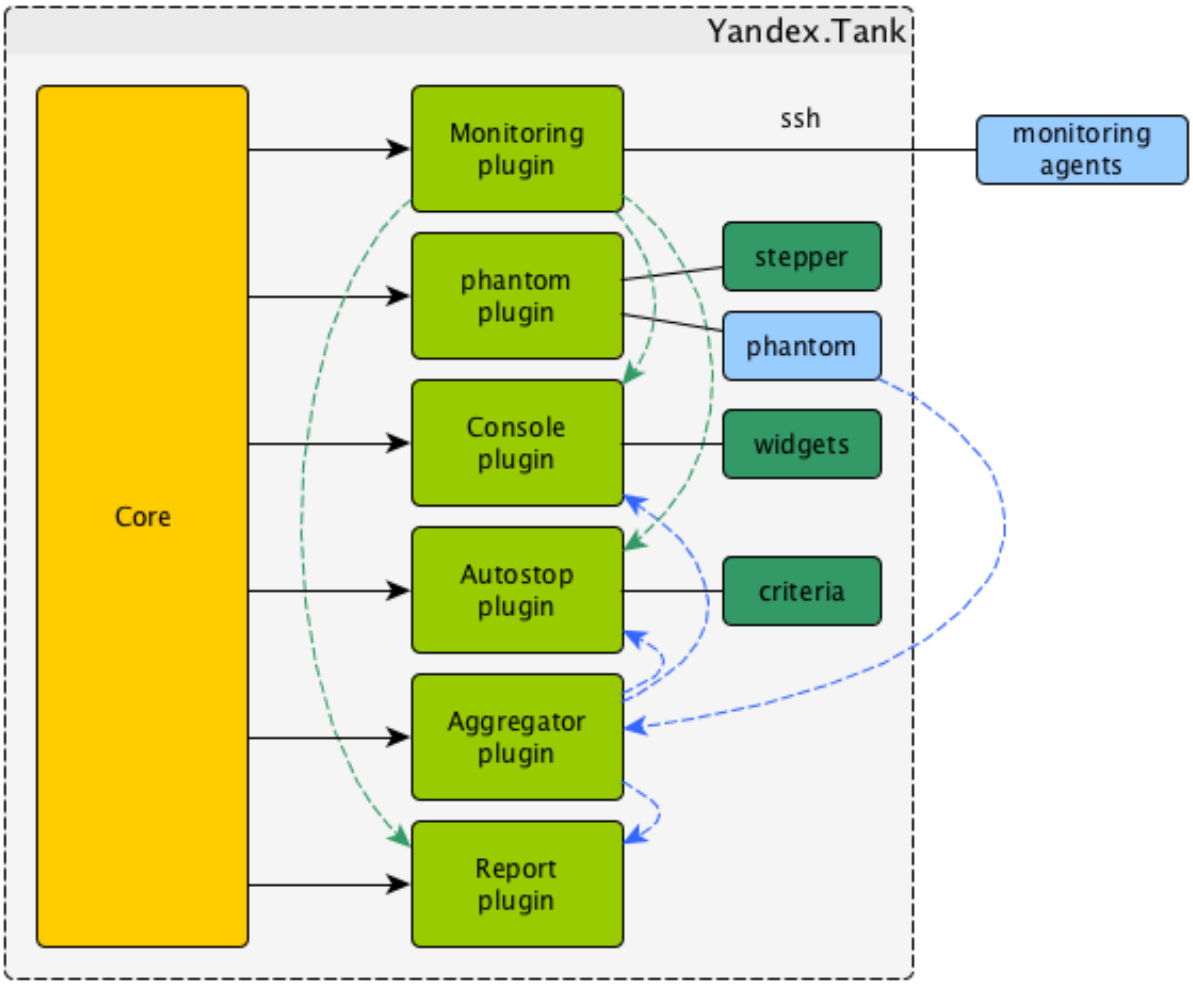
Modules

6.1 TankCore

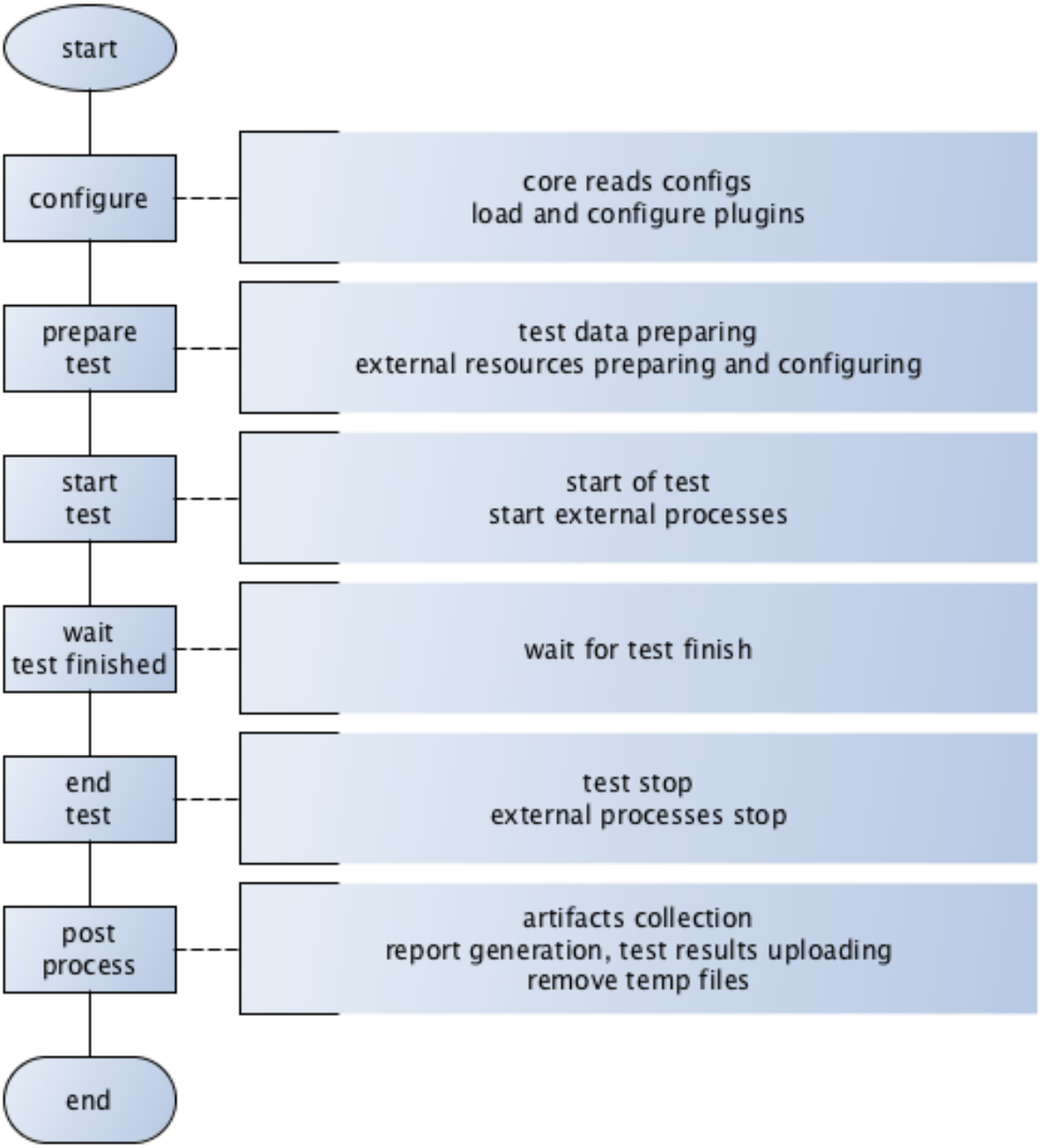
Core class. Represents basic steps of test execution. Simplifies plugin configuration, configs reading, artifacts storing. Represents parent class for modules/plugins.

yaml file section: **core**

6.1.1 Architecture



6.1.2 Test lifecycle



6.1.3 Options

Basic options:

lock_dir Directory for lockfile.

Default: `/var/lock/`.

plugin_<pluginname> Path to plugin. Empty path interpreted as disable of plugin.

artifacts_base_dir Base directory for artifacts storing. Temporary artifacts files are stored here.

Default: current directory.

artifacts_dir Directory where to keep artifacts after test.

Default: directory in `artifacts_base_dir` named in Date/Time format.

flush_config_to Dump configuration options after each tank step (yandex.tank steps. sorry, russian only) to that file

taskset_path Path to taskset command.

Default: taskset.

affinity Set a yandex-tank's (python process and load generator process) CPU affinity.

Default: empty.

Example: `0-3` enabling first 4 cores, '0,1,2,16,17,18' enabling 6 cores.

6.1.4 consoleworker

Consoleworker is a cmd-line interface for Yandex.Tank.

Worker class that runs and configures TankCore accepting cmdline parameters. Human-friendly unix-way interface for yandex-tank. Command-line options described above.

6.1.5 apiworker

apiworker is a python interface for Yandex.Tank.

Worker class for python. Runs and configures TankCore accepting `dict()`. Python-frinedly interface for yandex-tank.

Example:

```
from yandextank.api.apiworker import ApiWorker
import logging
import traceback
import sys

logger = logging.getLogger('')
logger.setLevel(logging.DEBUG)

#not mandatory options below:
options = dict()
options['config'] = '/path/to/config/load.ini'
options['manual_start'] = "1"
options['user_options'] = [
    'phantom.ammofile=/path/to/ammofile',
    'phantom.rps_schedule=const(1,2m)',
]
log_filename = '/path/to/log/tank.log'
#=====================================

apiworker = ApiWorker()
apiworker.init_logging(log_filename)
```

(continues on next page)

(continued from previous page)

```
try:
    apiworker.configure(options)
    apiworker.perform_test()
except Exception, ex:
    logger.error('Error trying to perform a test: %s', ex)
```

6.1.6 exit codes

```
{
    "0": "completed",
    "1": "interrupted_generic_interrupt",
    "2": "interrupted",
    "3": "interrupted_active_task_not_found ",
    "4": "interrupted_no_ammo_file",
    "5": "interrupted_address_not_specified",
    "6": "interrupted_cpu_or_disk_overload",
    "7": "interrupted_unknown_config_parameter",
    "8": "interrupted_stop_via_web",
    "9": "interrupted",
    "11": "interrupted_job_number_error",
    "12": "interrupted_phantom_error",
    "13": "interrupted_job_metainfo_error",
    "14": "interrupted_target_monitoring_error",
    "15": "interrupted_target_info_error",
    "21": "autostop_time",
    "22": "autostop_http",
    "23": "autostop_net",
    "24": "autostop_instances",
    "25": "autostop_total_time",
    "26": "autostop_total_http",
    "27": "autostop_total_net",
    "28": "autostop_negative_http",
    "29": "autostop_negative_net",
    "30": "autostop_http_trend",
    "31": "autostop_metric_higher",
    "32": "autostop_metric_lower"
}
```

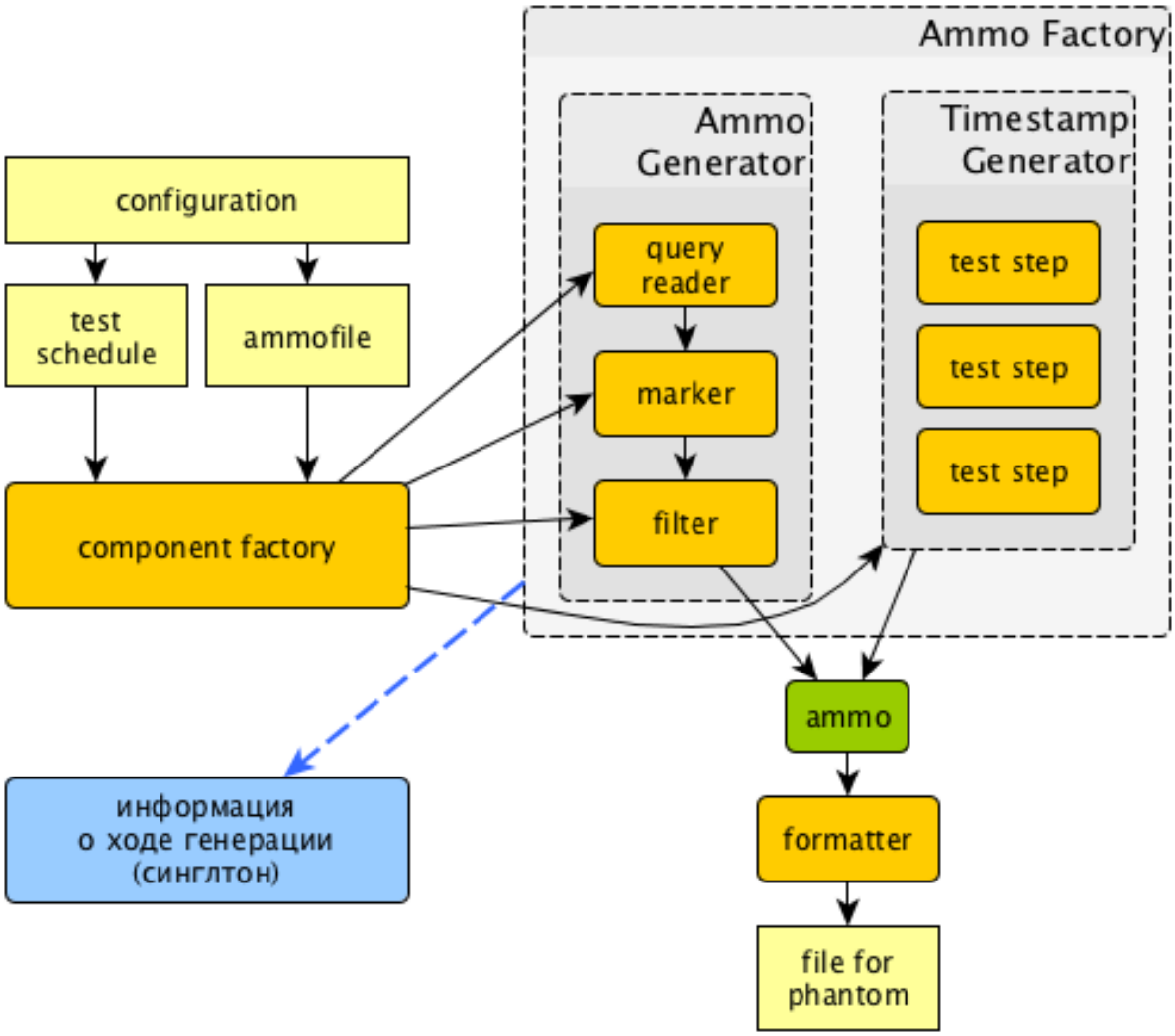
6.2 Load Generators

6.2.1 Phantom

Load generator module that uses phantom utility.

yaml file section: **phantom**

How it works



Options

Basic options

ammofile Ammo file path (ammo file is a file containing requests that are to be sent to a server. Could be gzipped).

load_profile Load profile behaviour. Specify load_type (`rps`, schedule load by defining requests per second or `instances` - schedule load defining concurrent active threads) and schedule.

```
phantom:
  address: [hostname]:port
  load_profile:
    load_type: rps #
    schedule: line(1, 10, 10m) # starting from 1rps growing linearly to␣
↪10rps during 10 minutes
```

instances Max number of instances (concurrent requests).

loop Number of times requests from ammo file are repeated in loop.

ammo_limit Limit request number.

autocases Enable marking requests automatically. `autocases: 2` means 2 uri path elements will be used. I.e `/hello/world/please/help` will produce case `_hello_world`

chosen_cases Use only selected cases.

There are 3 ways to constrain requests number:

- by load_type `rps` and `schedule`,
- by requests number with `ammo_limit`
- by loop number with `loop` option.

Tank stops if any constraint is reached. If stop reason is reached `ammo_limit` or `loop` it will be mentioned in log file. In test without load_type `rps` ammofile with requests used once by default.

Additional options

writelog Enable verbose request/response logging.

Default: 0.

Available options: 0 - disable, all - all messages, proto_warning - 4+5+network errors, proto_error - 5+network errors.

ssl Enable SSL.

Default: 0.

Available options: 1 - enable, 0 - disable.

timeout Response timeout.

Default: `11s`.

Note: Default multiplier is `seconds`. If you specify `10`, timeout will be 10 seconds. Currently we support here multipliers: 'd' for days, 'h' for hours, 'm' for minutes, 's' for seconds Examples: `0.1s` is 100 milliseconds. `1m` for 1 minute.

---

address Address of target.

Default: `127.0.0.1`.

Format: `[host]:port`, `[ipv4]:port`, `[ipv6]:port`. Tank checks each test if port is available.

gatling_ip Use multiple source addresses. List, divided by spaces.

tank_type Available options: `http` and `none` (raw TCP).

Default: `http`.

eta_file Path to ETA file.

connection_test Test TCP socket connection before starting the test.

Default: 1.

Available options: 1 - enable, 0 - disable.

URI-style options

uris URI list, multiline option.

headers HTTP headers list in the following form: *[Header: value]*.

header_http HTTP version.

Default: `1.0`

Available options: `1.0` and `1.1`.

Note: HTTP/2.0 is NOT supported by this load generator. Use Pandora or BFG.

---

stpd-file cache options

use_caching Enable cache.

Default: `1`.

cache_dir Cache files directory.

Default: base artifacts directory.

force_stepping Force stpd file generation.

Default: `0`.

Advanced options

phantom_path Phantom utility path.

Default: `phantom`.

phantom_modules_path Phantom modules path.

Default: `/usr/lib/phantom`.

config Use given (in this option) config file for phantom instead of generated.

phout_file Import this phout instead of launching phantom (import phantom results).

stpd_file Use this stpd-file instead of generated.

threads Phantom thread count.

Default: `<processor cores count>/2 + 1`.

buffered_seconds Amount of seconds to which delay aggregator, to be sure that everything were read from phout.

additional_libs List separated by whitespaces, will be added to phantom config file in section `module_setup`

method_prefix Object's type, that has a functionality to create test requests.

Default: `method_stream`.

source_log_prefix Prefix, added to class name that reads source data.

Default: empty.

method_options Additional options for method objects. It is used for Elliptics etc.

Default: empty.

affinity Set a phantom's CPU affinity.

Example: `0-3` enabling first 4 cores, '0,1,2,16,17,18' enabling 6 cores.

Default: empty.

TLS/SSL additional options

Note: `ssl: 1` is required

client_cipher_suites Cipher list, consists of one or more cipher strings separated by colons (see man ciphers).

Example: client_cipher_suites = RSA:!COMPLEMENTOFALL

Default: empty.

client_certificate Path to client certificate which is used in client's "Certificate message" in Client-authenticated TLS handshake.

Default: empty.

client_key Path to client's certificate's private key, used for client's "CertificateVerify message" generation in Client-authenticated TLS handshake.

Default: empty.

Phantom http-module tuning options

phantom_http_line First line length.

Default: `1K`.

phantom_http_field_num Headers amount.

Default: `128`.

phantom_http_field Header size.

Default: `8K`.

phantom_http_entity Answer `size`.

Default: `8M`.

Note: Please, keep in mind, especially if your service has large answers, that phantom doesn't read more than defined in `phantom_http_entity`.

Artifacts

phantom_*.conf Generated configuration files.

phout_*.log Raw results file.

phantom_stat_*.log Phantom statistics, aggregated by seconds.

answ_*.log Detailed request/response log.

phantom_*.log Internal phantom log.

Multi-tests

To make several simultaneous tests with phantom, add proper amount of sections to special section `multi` for `phantom`. All subtests are executed in parallel. Multi-test ends as soon as one subtest stops.

Example:

```
phantom:
  address: hostname:port
  load_profile:
    load_type: rps
    schedule: const(1,30s)
  uris:
    - /
  autocases: 1
  multi:
    - address: hostname1:port1
      load_profile:
        load_type: rps
        schedule: const(1,10s)
      uris:
        - /123
        - /321
      ssl: 1
      autocases: 1
    - address: hostname2:port2
      load_profile:
        load_type: rps
        schedule: const(1,10s)
      uris:
        - /123
        - /321
      ssl: 1
      autocases: 1
telegraf:
  enabled: false
```

Options that apply only for main section: `buffered_seconds`, `writelog`, `phantom_modules_path`, `phout_file`, `config`, `eta_file`, `phantom_path`

6.2.2 JMeter

JMeter module uses JMeter as a load generator. To enable it, disable phantom first (unless you really want to keep it active alongside at your own risk), enable JMeter plugin and then specify the parameters for JMeter:

```
phantom:
  enabled: false
jmeter:
  enabled: true
```

yaml file section: **jmeter**

Options

jmx Testplan for execution.

args Additional commandline arguments for JMeter.

jmeter_path Path to JMeter, allows to use alternative JMeter installation.

Default: `jmeter`

buffered_seconds Amount of seconds to which delay aggregator, to be sure that everything were read from jmeter's results file.

jmeter_ver Which jmeter version tank should expect. Currently it affects the way connection time is logged, but may be used for other version-specific settings.

Default: `3.0`

ext_log Available options: `none`, `errors`, `all`. Add one more simple data writer which logs all possible fields in jmeter xml format, this log is saved in test dir as `jmeter_ext_XXXX.jtl`.

Default: `none`

all other options in the section They will be passed as User Defined Variables to JMeter.

Timing calculation issues

Since version 2.13 jmeter could measure connection time, latency and full request time (aka <interval_real> in phantom), but do it in it's own uniq way: latency include connection time but not recieve time. For the sake of consistency we recalculate <latency> as <latency - connect_time> and calculate <recieve_time> as <interval_real - latency - connect_time>>, but it does not guranteed to work perfectly in all cases (i.e. some samplers may not support latency and connect_time and you may get something strange in case of timeouts).

For jmeter 2.12 and older connection time logging not avaliable, set `jmeter_ver` properly or you'll get an error for unknown field in Simlpe Data Writer listner added by tank.

Artifacts

<original jmx> Original testplan.

<modified jmx> Modified test plan with results output section.

<jmeter_*.jtl> JMeter's results.

<jmeter_*.log> JMeter's log.

6.2.3 BFG

(What is BFG) BFG is a generic gun that is able to use different kinds of cannons to shoot. To enable it, disable phantom first (unless you really want to keep it active alongside at your own risk), enable BFG plugin and then specify the parameters for BFG and for the gun of your choice.

There are three predefined guns: Log Gun, Http Gun and SQL gun. First two are mostly for demo, if you want to implement your own gun class, use them as an example.

But the main purpose of BFG is to support user-defined scenarios in python. Here is how you do it using 'ultimate' gun.

1. Define your scenario as a python class, for example, `LoadTest` (in a single-file module, for example, `test` in current working directory `./`), or a package:

```
import logging
log = logging.getLogger(__name__)


class LoadTest(object):
    def __init__(self, gun):

        # you'll be able to call gun's methods using this field:
        self.gun = gun

        # for example, you can get something from the 'ultimate' section of a config␣
↪file:
        my_var = self.gun.get_option("my_var", "hello")

    def case1(self, missile):
        # we use gun's measuring context to measure time.
        # The results will be aggregated automatically:
        with self.gun.measure("case1"):
            log.info("Shoot case 1: %s", missile)

        # there could be multiple steps in one scenario:
        with self.gun.measure("case1_step2") as sample:
            log.info("Shoot case 1, step 2: %s", missile)
            # and we can set the fields of measured object manually:
            sample["proto_code"] = 500

            # the list of available fields is below

    def case2(self, missile):
        with self.gun.measure("case2"):
            log.info("Shoot case 2: %s", missile)

    def setup(self, param):
        ''' this will be executed in each worker before the test starts '''
        log.info("Setting up LoadTest: %s", param)

    def teardown(self):
        ''' this will be executed in each worker after the end of the test '''
        log.info("Tearing down LoadTest")
        #It's mandatory to explicitly stop worker process in teardown
        os._exit(0)
        return 0
```

2. Define your options in a config file:

```
phantom:
  enabled: false
bfg:
  enabled: true
  instances: 10
  gun_config:
    class_name: LoadTest
    module_path: ./
    module_name: test
    init_param: Hello
  gun_type: ultimate
  load_profile:
    load_type: rps
    schedule: const(1, 30s)
```

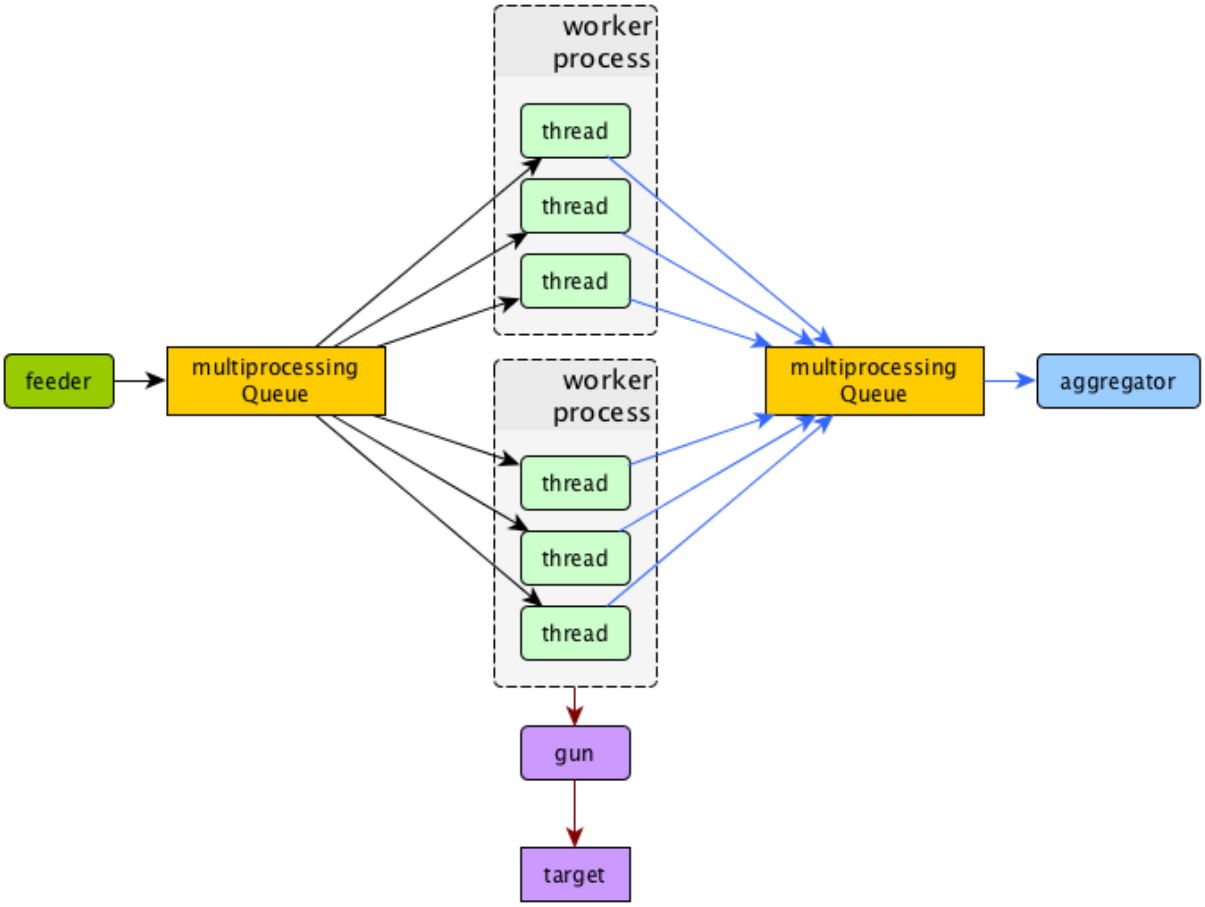
3. Create an ammo file: Ammo format: one line – one request, each line begins with case name separated by tab symbol ('t'). Case name defines the method of your test class that will be executed. The line itself will be passed to your method as 'missile' parameter. If there was no case name for an ammo, the 'default' case name will be used

```
case1<TAB>my-case1-ammo
case2<TAB>my-case2-ammo
my-default-case-ammo
```

Note: TIP: if each line is a JSON-encoded document, you can easily parse it inside your scenario code

4. Shoot em all!

How it works



BFG Worker Type

By default, BFG will create lots of processes (number is defined by `instances` option). Every process will execute requests in a single thread. These processes will consume a lot of memory. It's also possible to switch this behavior and use `gevent` to power up every worker process, allowing it to have multiple concurrent threads executing HTTP requests.

With green worker, it's recommended to set `instances` to number of CPU cores, and adjust the number of real threads by `green_threads_per_instance` option.

worker_type Set it to `green` to let every process have multiple concurrent green threads.

green_threads_per_instance Number of green threads every worker process will execute. Only affects `green` worker type.

BFG Options

yaml file section: **bfg**

gun_type What kind of gun should BFG use.

gun_config Gun configuration options

ammo_type What ammo parser should BFG use.

Default: `caseline`.

pip Install python modules with `pip install --user` before the test. If you need multiple modules use multiline options, i.e.:

```
pip=grequests
  msgpack
```

init_param An initialization parameter that will be passed to your `setup` method.

other common stepper options

Ultimate Gun Options

yaml gun_type section: **ultimate**

Specify `gun_config` with:

module_path Path to your module

module_name Python module name

class_name Class that contains load scenarios, default: LoadTest

The fields of measuring context object and their default values:

send_ts A timestamp when context was entered.

tag A marker passed to the context.

interval_real The time interval from enter to exit. If the user defines his own value, it will be preserved. Microseconds.

connect_time Microseconds. Default: 0

send_time Microseconds. Default: 0

latency Microseconds. Default: 0

receive_time Microseconds. Default: 0

interval_event Microseconds. Default: 0

size_out Bytes out. Integer. Default: 0

size_in Bytes in. Integer. Default: 0

net_code Network code. Integer. Default: 0

proto_code Protocol code (http, for example). Integer. Default: 200

SQL Gun Options

SQL gun is deprecated. Use ultimate gun.

6.2.4 Pandora

Pandora is a load generator written in Go. For now it supports only SPDY/3 and HTTP(S). Plugins for other protocols (HTTP/2, Websocket, XMPP) are on the way.

First of all you'll need to obtain a binary of pandora and place it somewhere on your machine. By default, Yandex.Tank will try to just run `pandora` (or you could specify a path to binary in `pandora_cmd`). Disable phantom first (unless you really want to keep it active alongside at your own risk), enable Pandora plugin and then specify the parameters.

```
# load.yaml

phantom:
  enabled: false
pandora:
  package: yandextank.plugins.Pandora
  enabled: true
  pandora_cmd: /usr/bin/pandora # Pandora executable path
  config_file: pandora_config.yml # Pandora config path
```

```
# pandora_config.yml

pools:
  - id: HTTP pool                    # pool name (for your choice)
    gun:
      type: http                     # gun type
      target: example.com:80         # gun target
    ammo:
      type: uri                      # ammo format
      file: ./ammo.uri               # ammo file path
    result:
      type: phout                    # report format (phout is compatible for Yandex.
↪Tank)
      destination: ./phout.log       # report file name

    rps:                             # RPS scheduler - controls throughput over test
      type: line                     # linear growth load
      from: 1                        # from 1 responses per second
      to: 5                          # to 5 responses per second
      duration: 2s                   # for 2 seconds

    startup:                         # startup scheduler - control the level of␣
↪parallelism
      type: once                     # start 5 instances
      times: 5                       #
```

You may specify pandora config contents in tank's config file via `config_content` option. This option has more priority over config_file option.

Create `ammo.uri` file, put your ammo inside and start the test.

```
# ammo.uri

/my/first/url
/my/second/url
```

Schedules

`Pandora` has two main RPS scheduler types: 1. `line` - makes linear load, where `from` and `to` are start and end, *duration* is a time for linear load increase from `from` to `to`.

2. `const` - makes constant load, where `ops` is a load value for `duration` time.

Custom_guns

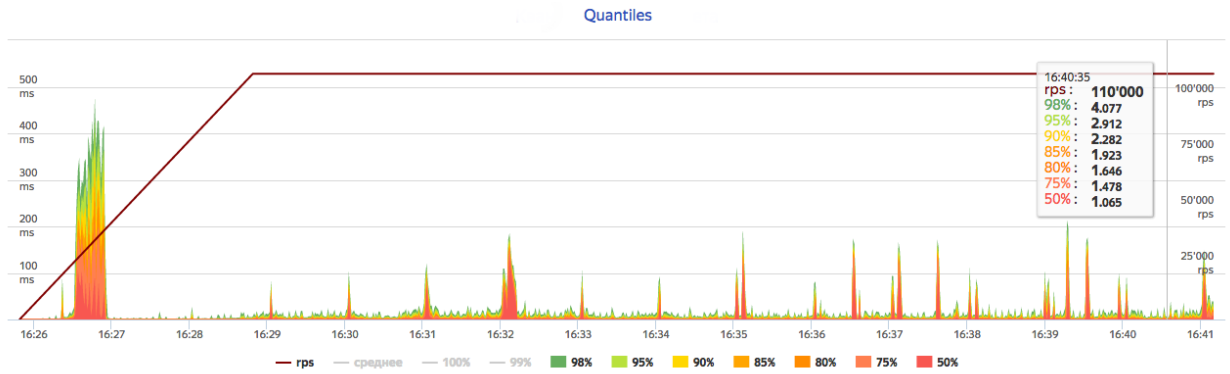
You can create you own Golang-based gun with *pandora*. Feel free to find examples at [pandora documentation](https://yandexpandora.readthedocs.io/en/develop/)

6.3 Artifact uploaders

Note: Graphite uploader and BlazeMeter Sense are not currently supported in the last Yandex.Tank version. If you want one of them, use 1.7 branch.

6.3.1 Yandex.Overload

Overload is a service for performance analytics made by Yandex. We will store your performance experiments results and show them in graphic and tabular form. Your data will be available at https://overload.yandex.net.



yaml file section: **overload**

Options

token_file Place your token obtained from Overload (click your profile photo) into a file and specify the path here

job_name (Optional) Name of a job to be displayed in Yandex.Overload

job_dsc (Optional) Description of a job to be displayed in Yandex.Overload

Example:

```
overload:
  token_file: token.txt
  job_name: test
  job_dsc: test description
```

6.3.2 InfluxDB

InfluxDB uploader.

yaml file section: **influx**

Options

address (Optional) InfluxDB address. (Default: 'localhost')

port (Optional) InfluxDB port. (Default: 8086)

database (Optional) InfluxDB database. (Default: 'mydb')

username (Optional) InfluxDB user name. (Default: 'root')

password (Optional) InfluxDB password. (Default: 'root')

labeled (Optional) Send per-label (ammo tags) stats to influxdb. (Default: false)

histograms (Optional) Send response time histograms to influxdb. (Default: false)

prefix_measurement (Optional) Add prefix to measurement name. (Default: '')

tank_tag (Optional) Tank tag. (Default: 'unknown')

custom_tags (Optional) Dict of custom tags, added to every sample row.

Example:

```
influx:
  enabled: true
  address: yourhost.tld
  database: yourbase
  tank_tag: 'mytank'
  prefix_measurement: 'your_test_prefix_'
  labeled: true
  histograms: true
```

6.4 Handy tools

6.4.1 Auto-stop

The Auto-stop module gets the data from the aggregator and passes them to the criteria-objects that decide if we should stop the test.

yaml file section: **autostop**

Options

autostop Criteria list in following format: `type(parameters)`

Basic criteria types

time Stop the test if average response time is higher then allowed.

Example: `time(1s500ms, 30s) time(50,15).`

Exit code - 21

http Stop the test if the count of responses in time period (specified) with HTTP codes fitting the mask is larger then the specified absolute or relative value.

Examples: `http(404,10,15) http(5xx, 10%, 1m).` Exit code - 22

net Like `http`, but for network codes. Use `xx` for all non-zero codes.

Exit code - 23

quantile Stop the test if the specified percentile is larger then specified level for as long as the time period specified.

Available percentile values: 25, 50, 75, 80, 90, 95, 98, 99, 100.

Example: `quantile (95,100ms,10s)`

instances Available when phantom module is included. Stop the test if instance count is larger then specified value.

Example: `instances(80%, 30) instances(50,1m).`

Exit code - 24

metric_lower and metric_higher Stop test if monitored metrics are lower/higher than specified for time period.

Example: metric_lower(127.0.0.1,Memory_free,500,10).

Exit code - 31 and 32

Note: metric names (except customs) are written with underline. For hostnames masks are allowed (i.e target-*.load.net)

steady_cumulative Stops the test if cumulative percentiles does not change for specified interval.

Example: `steady_cumulative(1m).`

Exit code - 33

limit Will stop test after specified period of time.

Example: `limit(1m).`

Basic criteria aren't aggregated, they are tested for each second in specified period. For example autostop=time(50,15) means "stop if average responce time for every second in 15s interval is higher than 50ms"

Advanced criteria types

total_time Like `time`, but accumulate for all time period (responses that fit may not be one-after-another, but only lay into specified time period).

Example: `total_time(300ms, 70%, 3s).`

Exit code - 25

total_http Like `http`, but accumulated. See `total_time`.

Example: `total_http(5xx,10%,10s) total_http(3xx,40%,10s).`

Exit code - 26

total_net Like `net`, but accumulated. See `total_time`.

Example: `total_net(79,10%,10s) total_net(11x,50%,15s)`

Exit code - 27

negative_http Inversed `total_http`. Stop if there are not enough responses that fit the specified mask. Use to be shure that server responds 200.

Example: `negative_http(2xx,10%,10s).`

Exit code - 28

negative_net Inversed `total_net`. Stop if there are not enough responses that fit the specified mask.

Example: `negative_net(0,10%,10s).`

Exit code - 29

http_trend Stop if trend for defined http codes is negative on defined period. Trend is a sum of an average coefficient for linear functions calculated for each pair points in last n seconds and standart deviation for it

Example: http_trend(2xx,10s).

Exit code - 30

riteria for specific tag

All criteria except `limit` could be used not for all test, but for a specially tagged uri.

Example: `time(1s, 5s, /latest/index/)` Stop test if average response time is higher than 1s ONLY from uri with tag `/latest/index/` for 5s.

It can be used for developing specific test success criteria for each uri.

Example:

```
autostop:
  autostop:
    - http(4xx, 20%, 15s, GET /weff?id=1)
    - http(4xx, 5%, 5s, POST /authorize)
```

Stop test if there're more than 5% of 4xx codes for uri with tag `POST /authorize` or if there're more than 20% of 4xx codes for uri with tag `GET /weff?id=1`.

6.4.2 Telegraf

Runs metrics collection through SSH connection. You can debug your SSH connection using `yandex-tank-check-ssh` tool. It is supplied with Yandex.Tank.

Credits to https://github.com/influxdata/telegraf for metric collection agent.

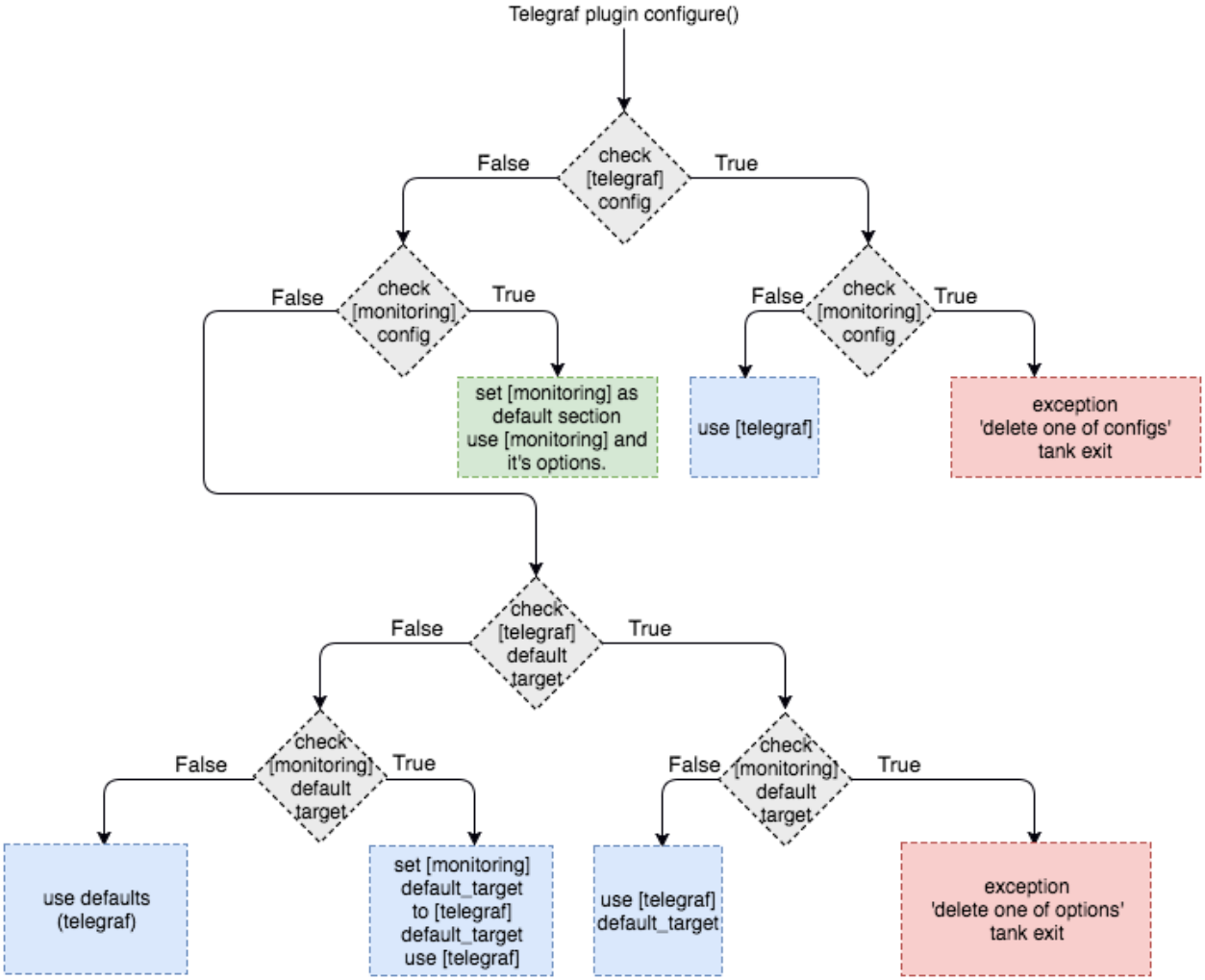
```
# load.yaml
# ...
telegraf:
  enabled: true
  package: yandextank.plugins.Telegraf
```

In https://github.com/yandex/yandex-tank/blob/master/yandextank/core/config/00-base.yaml it is already done. Please, don't use both `yandextank.plugins.Telegraf` and `yandextank.plugins.Monitoring` simultaneously.

Config file section: **telegraf**

You can use old monitoring config format, if you specify it in **monitoring** section. Telegraf plugin transparently supports it. You can use new monitoring config format, if you specify it in **telegraf** section.

Backward compatibility logic:



Telegraf plugin automatically uploads telegraf collector binary to target from tank if exists.

Options

config Path to monitoring config file.

Default: `auto` means collect default metrics from `default_target` host. If `none` is defined, monitoring won't be executed. Also it is possible to write plain multiline XML config.

default_target An address where from collect "default" metrics. When phantom module is used, address will be obtained from it.

ssh_timeout Ssh connection timeout.

Default: 5s

disguise_hostnames Disguise real host names.

Default: 0

Configuration

Net access and authentication

Telegraf requires ssh access to hosts for copy and executing agents/telegraf collector binaries on them. SSH session is established with user account specified by "username" parameter of Host element, otherwise current user account, so you need to copy your public keys (ssh-copy-id) and enable nonpassword authorization on hosts. If connection establishing failed for some reason in `ssh_timeout` seconds, corresponding message will be written to console and monitoring log and task will proceed further. Tip: write to `.ssh/config` next lines to eliminate `-A` option in `ssh`

```
StrictHostKeyChecking no
ForwardAgent yes
```

Configuration file format

Config is an XML file with structure: root element `Monitoring` includes elements `Host` which contains elements-metrics Example:

```
<Monitoring>
    <Host address="somehost.tld" interval="1" username="netort">
        <CPU fielddrop='["time_*", "usage_guest_nice"]'></CPU>
        <Kernel fielddrop='["active", "inactive", "total", "used_per*", "avail*"]'></
↪Kernel>
        <Net fielddrop='["icmp*", "ip*", "udplite*", "tcp*", "udp*", "drop*", "err*"]
↪' interfaces='["eth0","eth1","lo"]'></Net>
        <System fielddrop='["n_users", "n_cpus", "uptime*"]'></System>
        <Memory fielddrop='["active", "inactive", "total", "used_per*", "avail*"]'></
↪Memory>
        <Disk devices='["vda1","sda1","sda2","sda3"]'></Disk>
        <Netstat />
        <Custom diff="1" measure="call" label="test">curl -s -H 'Host: host.tld'
↪'http://localhost:6100/stat'  | python -c 'import sys, json; j = json.load(sys.
↪stdin); print "\n".join(`c["values"]["accept"]` for c in j["charts"] if c["name"]␣
↪== "localqueue_wait_time")'</Custom>
        <Source>/path/to/file</Source>
        <TelegrafRaw>
            [[inputs.ping]]
            urls = ["127.0.0.1"]
            count = 1
        </TelegrafRaw>
    </Host>
```

(continues on next page)

(continued from previous page)

```
    <Host address="localhost" telegraf="/usr/bin/telegraf">
        <CPU percpu="true"></CPU>
        <NetResponse address="localhost:80" protocol="tcp" timeout="1s"></NetResponse>
        <Net fielddrop='["icmp*", "ip*", "udplite*", "tcp*", "udp*", "drop*", "err*"]
↪' interfaces='["eth0","eth1","docker0","lo"]'></Net>
    </Host>
</Monitoring>
```

Element `Host`

Contains address and role of monitored server. Attributes:

address="<IP address or domain name> Server adddress. Mandatory. Special mask `[target]` could be used here, which means "get from the tank target address"

port="<SSH port>" Server's ssh port. Optional.

Default: 22

python="<python path>" The way to use alternative python version. Optional.

interval="<seconds>" Metrics collection interval. Optional.

Default: 1 second

comment="<short commentary>" Short notice about server's role in test. Optional.

Default: empty

username="<user name>" User account to connect with. Optional.

Default: current user account.

telegraf="/path/to/telegraf" Path to telegraf binary on remote host. Optional.

Default: */usr/bin/telegraf*

Example: `<Host address="localhost" comment="frontend" interval="5" username="tank"/>`

Metric elements

Metric elements in general are set by metrics group name.

There are plenty of config-wide configuration options (such as 'fielddrop', 'fieldpass' etc, you can read about them here: https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md

List of metrics group names and particular metrics in them:

- **CPU**
  - percpu - default: false
- System
- Memory
- **Disk**
  - devices - default: ",".join(["'vda%s","sda%s"' % (num, num) for num in range(6)]). Format sample: ["sda1", "docker0"]

- **Net**
  - interfaces - default: “,”.join([“‘eth%s”’ % (num) for num in range(6)]). Format sample: [“eth0”,”eth1”]
- Netstat
- Kernel
- KernelVmstat
- **NetResponse**
  - protocol - default: “tcp”. Protocol, must be “tcp” or “udp”
  - address - default: “:80”. Server address and port
  - timeout - default: “1s”. Set timeout
  - send - default: None. Optional string sent to the server
  - expect - default: None. Optional expected string in answer
- **Custom**
  - diff - default: 0
  - measure - default: call - metric value is a command or script execution output. Example: *<Custom measure=”call” diff=”1” label=”Base size”>du -s /var/lib/mysql/ | awk ‘{print $1}’</Custom>*
- **TelegrafRaw**
  - raw telegraf TOML format, transparently added to final collector config
- **Source**
  - additional source file in telegraf json format, can be used to add custom metrics that needs complex processing and do not fit into standart custom metrics (like log parsing with aggregation). Custom metrics do not include timestamps but source does. You can import async data with Source.

  Config Host section example: `<Source>/path/to/file</Source>`

  File format: `jsonline`. Each line is a json document.

  Example: `{"fields":{"metric_name_1":0,"metric_name_2":98.27694231863998, },"name":"custom_group-name","timestamp":1503990965}`

6.4.3 Console on-line screen

Shows usefull information in console while running the test

Config file section: **console**

Options

short_only Show only one-line summary instead of full-screen. Usefull for scripting.

Default: 0 (disabled)

info_panel_width relative right-panel width in percents,

Default: 33

disable_all_colors Switch off color scheme

Available options: 0/1

Default: 0

disable_colors Don't use specified colors in console. List with whitespaces. Example: `WHITE GREEN RED CYAN MAGENTA YELLOW`

6.4.4 Aggregator

The aggregator module is responsible for aggregation of data received from different kind of modules and transmitting that aggregated data to consumer modules.

6.4.5 ShellExec

The ShellExec module executes the shell-scripts (hooks) on different stages of test, for example, you could start/stop some services just before/after the test. Every hook must return 0 as an exit code or the test is terminated. Hook's stdout will be written to DEBUG, stderr will be WARNINGs.

Example:

```
# load.yaml
# ...
shellexec:
  start: /bin/ls -l
```

Note: Command quoting is not needed. That line doesn't work: `start="/bin/ls -l"`

Config file section: **shellexec**

Options

prepare The script to run on prepare stage.

start The script to run on start stage.

poll The script to run every second while the test is running.

end The script to run on end stage.

post_process The script to run on postprocess stage

6.4.6 Resource Check

Module checks free memory and disk space amount before and during test. Test stops if minimum values are reached.

yaml file section: **rcheck**

Options

interval How often to check resources.

Default interval: `10s`

disk_limit Minimum free disk space in MB.

Default: `2GB`

mem_limit Minimum free memory amount in MB.

Default: `512MB`

6.4.7 RC Assert

Module checks test's exit code with predefined acceptable codes. If exit code matches, it is overrides as 0. Otherwise it is replaced with code from option `fail_code`

yaml file section: **rcassert**

Options

pass list of acceptable codes, delimiter - whitespace.

Default: empty, no check is performed.

fail_code Exit code when check fails, integer number.

Default: 10

CHAPTER 7

Ammo generators

sample req-style ammo generator (python):

`usage: cat data | python make_ammo.py` For each line of 'data' file this script will generate phantom ammo. Line format: `GET||/url||case_tag||body(optional)`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys


def make_ammo(method, url, headers, case, body):
    """ makes phantom ammo """
    # http request w/o entity body template
    req_template = (
          "%s %s HTTP/1.1\r\n"
          "%s\r\n"
          "\r\n"
    )

    # http request with entity body template
    req_template_w_entity_body = (
          "%s %s HTTP/1.1\r\n"
          "%s\r\n"
          "Content-Length: %d\r\n"
          "\r\n"
          "%s\r\n"
    )

    if not body:
        req = req_template % (method, url, headers)
    else:
        req = req_template_w_entity_body % (method, url, headers, len(body), body)

    # phantom ammo template
```

(continues on next page)

(continued from previous page)

```
    ammo_template = (
        "%d %s\n"
        "%s"
    )

    return ammo_template % (len(req), case, req)


def main():
    for stdin_line in sys.stdin:
        try:
            method, url, case, body = stdin_line.split("||")
            body = body.strip()
        except ValueError:
            method, url, case = stdin_line.split("||")
            body = None

        method, url, case = method.strip(), url.strip(), case.strip()

        headers = "Host: hostname.com\r\n" + \
            "User-Agent: tank\r\n" + \
            "Accept: */*\r\n" + \
            "Connection: Close"

        sys.stdout.write(make_ammo(method, url, headers, case, body))


if __name__ == "__main__":
    main()
```

sample POST multipart form-data generator (python)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import requests

def print_request(request):
    req = "{method} {path_url} HTTP/1.1\r\n{headers}\r\n{body}".format(
        method = request.method,
        path_url = request.path_url,
        headers = ''.join('{0}: {1}\r\n'.format(k, v) for k, v in request.headers.
↪items()),
        body = request.body or "",
    )
    return "{req_size}\n{req}\r\n".format(req_size = len(req), req = req)

#POST multipart form data
def post_multipart(host, port, namespace, files, headers, payload):
    req = requests.Request(
        'POST',
        'https://{host}:{port}{namespace}'.format(
            host = host,
            port = port,
            namespace = namespace,
        ),
        headers = headers,
        data = payload,
```

(continues on next page)

(continued from previous page)

```
        files = files
    )
    prepared = req.prepare()
    return print_request(prepared)

if __name__ == "__main__":
    #usage sample below
    #target's hostname and port
    #this will be resolved to IP for TCP connection
    host = 'test.host.ya.ru'
    port = '8080'
    namespace = '/some/path'
    #below you should specify or able to operate with
    #virtual server name on your target
    headers = {
        'Host': 'ya.ru'
    }
    payload = {
        'langName': 'en',
        'apikey': '123'
    }
    files = {
        # name, path_to_file, content-type, additional headers
        'file': ('image.jpeg', open('./imagex.jpeg', 'rb'), 'image/jpeg ', {'Expires
↪': '0'})
    }

    print post_multipart(host, port, namespace, files, headers, payload)
```

CHAPTER 8

Config reference

8.1 Core

8.1.1 `core` (dict)

- (no description).

`affinity` **(string)** *- specify cpu core(s) to bind tank process to, http://linuxhowtos.org/manpages/1/taskset.htm. Default:* `""`

`api_jobno` **(string)** *- tankapi job id, also used as test's directory name - determined by tank.*

`artifacts_base_dir` **(string)** *- base directory to store tests' artifacts directories. Default:* `./logs`

`artifacts_dir` **(string)** *- directory inside base directory to store test's artifacts, defaults to api_jobno if null.*

`cmdline` **(string)** *- (no description).*

`exitcode` **(integer)** *- (no description).*

`flush_config_to` **(string)** *- path to store config.*

`ignore_lock` **(boolean)** *- if tank is locked (*.lock file(s) presented in lock_dir), shoot nevertheless. Default:* `False`

`lock_dir` **(string)** *- directory to store *.lock files. Default:* `/var/lock/`

`message` **(string)** *- (no description).*

`operator` **(string)** *- your username.*

`pid` **(integer)** *- (no description).*

`taskset_path` **(string)** *- (no description). Default:* `taskset`

`uuid` **(string)** *- (no description).*

allow_unknown False

8.1.2 `version` (string)

- *(no description).*

8.2 ShootExec

8.2.1 `cmd` (string)

- *command that produces test results and stats in Phantom format.* **Required.**

8.2.2 `output_path` (string)

- *path to test results.* **Required.**

8.2.3 `stats_path` (string)

- *path to tests stats. Default:* `None`

nullable True

8.3 Influx

8.3.1 `address` (string)

- *(no description). Default:* `localhost`

8.3.2 `chunk_size` (integer)

- *(no description). Default:* `500000`

8.3.3 `custom_tags` (dict)

- *(no description). Default:* `{}`

8.3.4 `database` (string)

- *(no description). Default:* `mydb`

8.3.5 `grafana_dashboard` (string)

- *(no description). Default:* `tank-dashboard`

8.3.6 `grafana_root` (string)

- *(no description). Default:* `http://localhost/`

8.3.7 `labeled` (boolean)

- *(no description). Default:* `False`

8.3.8 `password` (string)

- *(no description). Default:* `root`

8.3.9 `port` (integer)

- *(no description). Default:* `8086`

8.3.10 `prefix_measurement` (string)

- *(no description). Default:* `""`

8.3.11 `tank_tag` (string)

- *(no description). Default:* `unknown`

8.3.12 `username` (string)

- *(no description). Default:* `root`

8.4 Telegraf

8.4.1 `config_contents` (string)

- *used to repeat tests from Overload, not for manual editing.*

8.4.2 `config` (string)

- *Path to monitoring config file. Default:* `auto`

one of

`<path/to/file.xml>` path to telegraf configuration file

`auto` collect default metrics from default_target host

`none` disable monitoring

8.4.3 `default_target` (string)

- host to collect default metrics from (if "config: auto" specified). Default: `localhost`

8.4.4 `disguise_hostnames` (boolean)

- Disguise real host names - use this if you upload results to Overload and dont want others to see your hostnames. Default: `True`

8.4.5 `kill_old` (boolean)

- kill old hanging agents on target(s). Default: `False`

8.4.6 `ssh_timeout` (string)

- timeout of ssh connection to target(s). Default: `5s`

examples

`10s` 10 seconds

`2m` 2 minutes

8.5 Autostop

8.5.1 `autostop` (list of string)

- list of autostop constraints. Default: `[]`

[list_element] (string) *- autostop constraint.*

examples

`http(4xx,50%,5)` stop when rate of 4xx http codes is 50% or more during 5 seconds

examples

`[quantile(50,100,20), http(4xx,50%,5)]` stop when either quantile 50% or 4xx http codes exceeds specified levels

8.5.2 `report_file` (string)

- path to file to store autostop report. Default: `autostop_report.txt`

8.6 JMeter

8.6.1 `affinity` (string)

- Use to set CPU affinity. Default: `""`

nullable True

8.6.2 `args` (string)

- additional commandline arguments for JMeter. Default: `""`

8.6.3 `buffer_size` (integer)

- jmeter buffer size. Default: `None`

nullable True

8.6.4 `buffered_seconds` (integer)

- Aggregator delay - to be sure that everything were read from jmeter results file. Default: `3`

8.6.5 `exclude_markers` (list of string)

- (no description). Default: `[]`

[list_element] (string) *- (no description).*

empty False

8.6.6 `ext_log` (string)

- additional log, jmeter xml format. Saved in test dir as jmeter_ext_XXXX.jtl. Default: `none`

one of [`none`, `errors`, `all`]

8.6.7 `extended_log` (string)

- additional log, jmeter xml format. Saved in test dir as jmeter_ext_XXXX.jtl. Default: `none`

one of [`none`, `errors`, `all`]

8.6.8 `jmeter_path` (string)

- Path to JMeter. Default: `jmeter`

8.6.9 `jmeter_ver` (float)

- Which JMeter version tank should expect. Affects the way connection time is logged. Default: `3.0`

8.6.10 `jmx` (string)

- Testplan for execution.

8.6.11 `shutdown_timeout` (integer)

- timeout for automatic test shutdown. Default: `10`

8.6.12 `variables` (dict)

- variables for jmx testplan. Default: `{}`

8.7 Pandora

8.7.1 `affinity` (string)

- Use to set CPU affinity. Default: `""`

nullable True

8.7.2 `buffered_seconds` (integer)

- (no description). Default: `2`

8.7.3 `config_content` (dict)

- Pandora config contents. Default: `{}`

8.7.4 `config_file` (string)

- Pandora config file path. Default: `""`

8.7.5 `expvar` (boolean)

- Toggle expvar monitoring. Default: `True`

8.7.6 `pandora_cmd` (string)

- Pandora executable path. Default: `pandora`

8.8 Android

8.8.1 `volta_options` (dict)

- (no description).

8.9 ResourceCheck

8.9.1 `disk_limit` (integer)

- *(no description). Default:* `2048`

8.9.2 `interval` (string)

- *(no description). Default:* `10s`

8.9.3 `mem_limit` (integer)

- *(no description). Default:* `512`

8.10 Bfg

8.10.1 `address` (string)

- *Address of target. Format: [host]:port, [ipv4]:port, [ipv6]:port. Port is optional. Tank checks each test if port is available.*

examples `127.0.0.1:8080`

`www.w3c.org`

8.10.2 `ammo_limit` (integer)

- *Upper limit for the total number of requests. Default:* `-1`

8.10.3 `ammo_type` (string)

- *Ammo format. Default:* `caseline`

8.10.4 `ammofile` (string)

- *Path to ammo file. Default:* `""`

tutorial_link http://yandextank.readthedocs.io/en/latest/core_and_modules.html#bfg

8.10.5 `autocases` (integer or string)

- *Use to automatically tag requests. Requests might be grouped by tag for later analysis. Default:* `0`

one of

`<N>` use N first uri parts to tag request, slashes are replaced with underscores

`uniq` tag each request with unique uid

uri tag each request with its uri path, slashes are replaced with underscores

examples

2 /example/search/hello/help/us?param1=50 -> _example_search

3 /example/search/hello/help/us?param1=50 -> _example_search_hello

uniq /example/search/hello/help/us?param1=50 -> c98b0520bb6a451c8bc924ed1fd72553

uri /example/search/hello/help/us?param1=50 -> _example_search_hello_help_us

8.10.6 cache_dir (string)

- *stpd-file cache directory. If not specified, defaults to base artifacts directory. Default:* `None`

nullable True

8.10.7 cached_stpd (boolean)

- *Use cached stpd file. Default:* `False`

8.10.8 chosen_cases (string)

- *Use only selected cases. Default:* `""`

8.10.9 enum_ammo (boolean)

- *(no description). Default:* `False`

8.10.10 file_cache (integer)

- *(no description). Default:* `8192`

8.10.11 force_stepping (integer)

- *Ignore cached stpd files, force stepping. Default:* `0`

8.10.12 green_threads_per_instance (integer)

- *Number of green threads every worker process will execute. For "green" worker type only. Default:* `1000`

tutorial_link http://yandextank.readthedocs.io/en/latest/core_and_modules.html#bfg

8.10.13 `gun_config` (dict)

- Options for your load scripts.

`base_address` **(string)** *- base target address.*

`class_name` **(string)** *- class that contains load scripts. Default:* `LoadTest`

`init_param` **(string)** *- parameter that's passed to "setup" method. Default:* `""`

`module_name` **(string)** *- name of module that contains load scripts.*

`module_path` **(string)** *- directory of python module that contains load scripts. Default:* `""`

allow_unknown True

tutorial_link http://yandextank.readthedocs.io/en/latest/core_and_modules.html#bfg

8.10.14 `gun_type` (string)

- Type of gun BFG should use. **Required.**

tutorial_link http://yandextank.readthedocs.io/en/latest/core_and_modules.html#bfg-options

one of [`custom`, `http`, `scenario`, `ultimate`]

8.10.15 `header_http` (string)

- HTTP version. Default: `1.0`

one of

`1.0` http 1.0

`1.1` http 1.1

8.10.16 `headers` (list of string)

- HTTP headers. Default: `[]`

[list_element] **(string)** *- Format: "Header: Value".*

examples `accept: text/html`

8.10.17 `instances` (integer)

- number of processes (simultaneously working clients). Default: `1000`

8.10.18 `load_profile` (dict)

- Configure your load setting the number of RPS or instances (clients) as a function of time, or using a prearranged schedule. **Required.**

`load_type` **(string)** *- Choose control parameter.* **Required.**

one of

`instances` control the number of instances

rps control the rps rate

stpd_file use prearranged schedule file

schedule (string) - *load schedule or path to stpd file.* **Required.**

examples

const(200,90s) constant load of 200 instances/rps during 90s

line(100,200,10m) linear growth from 100 to 200 instances/rps during 10 minutes

test_dir/test_backend.stpd path to ready schedule file

tutorial_link http://yandextank.readthedocs.io/en/latest/tutorial.html#tutorials

8.10.19 loop (integer)

- Loop over ammo file for the given amount of times. Default: `-1`

8.10.20 pip (string)

- pip modules to install before the test. Use multiline to install multiple modules. Default: `""`

8.10.21 uris (list of string)

- URI list. Default: `[]`

[list_element] (string) - *URI path string.*

examples `["/example/search", "/example/search/hello", "/example/search/hello/help"]`

8.10.22 use_caching (boolean)

- Enable stpd-file caching. Default: `True`

8.10.23 worker_type (string)

- (no description). Default: `""`

tutorial_link http://yandextank.readthedocs.io/en/latest/core_and_modules.html#bfg-worker-type

8.11 RCAssert

8.11.1 fail_code (integer)

- (no description). Default: `10`

8.11.2 `pass` (string)

- (no description). Default: `""`

8.12 ShellExec

8.12.1 `catch_out` (boolean)

- show commands stdout. Default: `False`

8.12.2 `end` (string)

- shell command to execute after test end. Default: `""`

8.12.3 `poll` (string)

- shell command to execute every second while test is running. Default: `""`

8.12.4 `post_process` (string)

- shell command to execute on post process stage. Default: `""`

8.12.5 `prepare` (string)

- shell command to execute on prepare stage. Default: `""`

8.12.6 `start` (string)

- shell command to execute on start. Default: `""`

8.13 JsonReport

8.13.1 `monitoring_log` (string)

- file name for monitoring log. Default: `monitoring.log`

8.13.2 `test_data_log` (string)

- file name for test data log. Default: `test_data.log`

8.14 DataUploader

8.14.1 `api_address` (string)

- *api base address. Default:* `https://overload.yandex.net/`

8.14.2 `api_attempts` (integer)

- *number of retries in case of api fault. Default:* `60`

8.14.3 `api_timeout` (integer)

- *delay between retries in case of api fault. Default:* `10`

8.14.4 `chunk_size` (integer)

- *max amount of data to be sent in single requests. Default:* `500000`

8.14.5 `component` (string)

- *component of your software. Default:* `""`

8.14.6 `connection_timeout` (integer)

- *tcp connection timeout. Default:* `30`

8.14.7 `ignore_target_lock` (boolean)

- *start test even if target is locked. Default:* `False`

8.14.8 `job_dsc` (string)

- *job description. Default:* `""`

8.14.9 `job_name` (string)

- *job name. Default:* `none`

8.14.10 `jobno_file` (string)

- *file to save job number to. Default:* `jobno_file.txt`

8.14.11 `jobno` (string)

- number of an existing job. Use to upload data to an existing job. Requres upload token.

dependencies upload_token

8.14.12 `lock_targets` (list or string)

- targets to lock. Default: `auto`

one of

`auto` automatically identify target host

`list_of_targets` list of targets to lock

tutorial_link http://yandextank.readthedocs.io

8.14.13 `log_data_requests` (boolean)

- log POSTs of test data for debugging. Tank should be launched in debug mode (–debug). Default: `False`

8.14.14 `log_monitoring_requests` (boolean)

- log POSTs of monitoring data for debugging. Tank should be launched in debug mode (–debug). Default: `False`

8.14.15 `log_other_requests` (boolean)

- log other api requests for debugging. Tank should be launched in debug mode (–debug). Default: `False`

8.14.16 `log_status_requests` (boolean)

- log status api requests for debugging. Tank should be launched in debug mode (–debug). Default: `False`

8.14.17 `maintenance_attempts` (integer)

- number of retries in case of api maintanance downtime. Default: `10`

8.14.18 `maintenance_timeout` (integer)

- delay between retries in case of api maintanance downtime. Default: `60`

8.14.19 `meta` (dict)

- additional meta information.

8.14.20 `network_attempts` (integer)

- number of retries in case of network fault. Default: `60`

8.14.21 network_timeout (integer)

- delay between retries in case of network fault. Default: `10`

8.14.22 notify (list of string)

- users to notify. Default: `[]`

8.14.23 operator (string)

- user who started the test. Default: `None`

nullable True

8.14.24 send_status_period (integer)

- delay between status notifications. Default: `10`

8.14.25 strict_lock (boolean)

- set true to abort the test if the the target's lock check is failed. Default: `False`

8.14.26 target_lock_duration (string)

- how long should the target be locked. In most cases this should be long enough for the test to run. Target will be unlocked automatically right after the test is finished. Default: `30m`

8.14.27 task (string)

- task title. Default: `""`

8.14.28 threads_timeout (integer)

- (no description). Default: `60`

8.14.29 token_file (string)

- API token.

8.14.30 upload_token (string)

- Job's token. Use to upload data to an existing job. Requres jobno.

dependencies jobno

8.14.31 `ver` (string)

- version of the software tested. Default: `""`

8.14.32 `writer_endpoint` (string)

- writer api endpoint. Default: `""`

8.15 Phantom

8.15.1 `additional_libs` (list of string)

- Libs for Phantom, to be added to phantom config file in section "module_setup". Default: `[]`

8.15.2 `address` (string)

- Address of target. Format: [host]:port, [ipv4]:port, [ipv6]:port. Port is optional. Tank checks each test if port is available. **Required.**

empty False

examples `127.0.0.1:8080`

`www.w3c.org`

8.15.3 `affinity` (string)

- Use to set CPU affinity. Default: `""`

examples

`0,1,2,16,17,18` enable 6 specified cores

`0-3` enable first 4 cores

8.15.4 `ammo_limit` (integer)

- Sets the upper limit for the total number of requests. Default: `-1`

8.15.5 `ammo_type` (string)

- Ammo format. Don't forget to change ammo_type option if you switch the format of your ammo, otherwise you might get errors. Default: `phantom`

tutorial_link http://yandextank.readthedocs.io/en/latest/tutorial.html#preparing-requests

one of

`access` Use access.log from your web server as a source of requests

`phantom` Use Request-style file. Most versatile, HTTP as is. See tutorial for details

uri Use URIs listed in file with headers. Simple but allows for GET requests only. See tutorial for details

uripost Use URI-POST file. Allows POST requests with bodies. See tutorial for details

8.15.6 ammofile (string)

- Path to ammo file. Ammo file contains requests to be sent to a server. Can be gzipped. Default: `""`

tutorial_link http://yandextank.readthedocs.io/en/latest/tutorial.html#preparing-requests

8.15.7 autocases (integer or string)

- Use to automatically tag requests. Requests might be grouped by tag for later analysis. Default: `0`

one of

<N> use N first uri parts to tag request, slashes are replaced with underscores

uniq tag each request with unique uid

uri tag each request with its uri path, slashes are replaced with underscores

examples

2 /example/search/hello/help/us?param1=50 -> _example_search

3 /example/search/hello/help/us?param1=50 -> _example_search_hello

uniq /example/search/hello/help/us?param1=50 -> c98b0520bb6a451c8bc924ed1fd72553

uri /example/search/hello/help/us?param1=50 -> _example_search_hello_help_us

8.15.8 buffered_seconds (integer)

- Aggregator latency. Default: `2`

8.15.9 cache_dir (string)

- stpd-file cache directory. Default: `None`

nullable True

8.15.10 chosen_cases (string)

- Use only selected cases. Default: `""`

8.15.11 client_certificate (string)

- Path to client SSL certificate. Default: `""`

8.15.12 client_cipher_suites (string)

- Cipher list, consists of one or more cipher strings separated by colons (see man ciphers). Default: `""`

8.15.13 `client_key` (string)

- Path to client's certificate's private key. Default: `""`

8.15.14 `config` (string)

- Use ready phantom config instead of generated. Default: `""`

8.15.15 `connection_test` (boolean)

- Test TCP socket connection before starting the test. Default: `True`

8.15.16 `enum_ammo` (boolean)

- (no description). Default: `False`

8.15.17 `file_cache` (integer)

- (no description). Default: `8192`

8.15.18 `force_stepping` (integer)

- Ignore cached stpd files, force stepping. Default: `0`

8.15.19 `gatling_ip` (string)

- (no description). Default: `""`

8.15.20 `header_http` (string)

- HTTP version. Default: `1.0`

one of

`1.0` http 1.0

`1.1` http 1.1

8.15.21 `headers` (list of string)

- HTTP headers. Default: `[]`

[list_element] (string) *- Format: "Header: Value".*

examples `accept: text/html`

8.15.22 `instances` (integer)

- Max number of concurrent clients. Default: `1000`

8.15.23 `load_profile` (dict)

- Configure your load setting the number of RPS or instances (clients) as a function of time,or using a prearranged schedule. **Required.**

`load_type` (string) *- Choose control parameter.* **Required.**

one of

`instances` control the number of instances

`rps` control the rps rate

`stpd_file` use prearranged schedule file

`schedule` (string) *- load schedule or path to stpd file.* **Required.**

examples

`const(200,90s)` constant load of 200 instances/rps during 90s

`line(100,200,10m)` linear growth from 100 to 200 instances/rps during 10 minutes

`test_dir/test_backend.stpd` path to ready schedule file

tutorial_link http://yandextank.readthedocs.io/en/latest/tutorial.html#tutorials

8.15.24 `loop` (integer)

- Loop over ammo file for the given amount of times. Default: `-1`

8.15.25 `method_options` (string)

- Additional options for method objects. It is used for Elliptics etc. Default: `""`

8.15.26 `method_prefix` (string)

- Object's type, that has a functionality to create test requests. Default: `method_stream`

8.15.27 `multi` (list of dict)

- List of configs for multi-test. All of the options from main config supported. All of them not required and inherited from main config if not specified. Default: `[]`

8.15.28 `name` (string)

- Name of a part in multi config. **Required.**

8.15.29 `phantom_http_entity` (string)

- Limits the amount of bytes Phantom reads from response. Default: `8M`

8.15.30 `phantom_http_field_num` (integer)

- Max number of headers. Default: `128`

8.15.31 `phantom_http_field` (string)

- Header size. Default: `8K`

8.15.32 `phantom_http_line` (string)

- First line length. Default: `1K`

8.15.33 `phantom_modules_path` (string)

- Phantom modules path. Default: `/usr/lib/phantom`

8.15.34 `phantom_path` (string)

- Path to Phantom binary. Default: `phantom`

8.15.35 `phout_file` (string)

- deprecated. Default: `""`

8.15.36 `port` (string)

- Explicit target port, overwrites port defined with address. Default: `""`

regex d{0,5}

8.15.37 `source_log_prefix` (string)

- Prefix added to class name that reads source data. Default: `""`

8.15.38 `ssl` (boolean)

- Enable ssl. Default: `False`

8.15.39 `tank_type` (string)

- Choose between http and pure tcp guns. Default: `http`

one of

`http` HTTP gun

`none` TCP gun

8.15.40 `threads` (integer)

- Phantom thread count. When not specified, defaults to <processor cores count> / 2 + 1. Default: `None`

nullable True

8.15.41 `timeout` (string)

- Response timeout. Default: `11s`

8.15.42 `uris` (list of string)

- URI list. Default: `[]`

[list_element] (string) *- URI path string.*

examples `["/example/search", "/example/search/hello", "/example/search/hello/help"]`

8.15.43 `use_caching` (boolean)

- Enable stpd-file caching for similar tests. Set false to reload ammo file and generate new stpd. Default: `True`

8.15.44 `writelog` (string)

- Enable verbose request/response logging. Default: `0`

one of

`0` disable

`all` all messages

`proto_error` 5xx+network errors

`proto_warning` 4xx+5xx+network errors

8.16 Console

8.16.1 `cases_max_spark` (integer)

- length of sparkline for each case, 0 to disable. Default: `120`

8.16.2 `cases_sort_by` (string)

- field for cases data sort. Default: `count`

one of [`count`, `net_err`, `http_err`]

8.16.3 `disable_all_colors` (boolean)

- disable colors in full output. Default: `False`

8.16.4 `disable_colors` (string)

- (no description). Default: `""`

8.16.5 `info_panel_width` (integer)

- width of right panel. Default: `33`

8.16.6 `max_case_len` (integer)

- max lenght of case name, longer names will be cut in console output. Default: `32`

8.16.7 `short_only` (boolean)

- do not draw full console screen, write short info for each second. Default: `False`

8.16.8 `sizes_max_spark` (integer)

- max length of sparkline for request/response sizes, 0 to disable. Default: `120`

8.16.9 `times_max_spark` (integer)

- max length of sparkline for fractions of request time, 0 to disable. Default: `120`

CHAPTER 9

Indices and tables

- genindex
- modindex
- search