bombard Documentation

Release 1.18.1

Andrey Sorokin

Nov 13, 2019

Contents

CHAPTER 1

Introduction

Bombard is a tool for stress test and benchmarking your HTTP server. Especially it's good to simulate a heavy load and initial burst of simultaneous HTTP requests with complex logic.

It is designed to be extremely simple yet powerful tool to load test functional behavior.

Thanks to optional Python inlines you can fast and easy describe complex logic for the tests.

Test report shows you how many requests per second your server is capable of serving and with what latency.

CHAPTER 2

Requests description

Requests can be just URL or contain JSON described like this

```
getToken:
    url: "{host}auth"  # use custom {host} variable to stay DRY
    method: POST
    body:  # below is JSON object for request body
        email: name@example.com
        password: admin
    extract:  # get token for next requests
        token:
```

In first request you can get security token as in example above.

And use it in next requests like that:

```
postsList:
   url: "{host}posts"
   headers:
       Authorization: "Bearer {token}"  # we get {token} in 1st request
   script: |
       for post in resp[:3]:  # for 1st three posts from response
           # schedule getPost request (from ammo section)
           # and provide it with id we got from the response
           reload(ammo.getPost, id=post['id'])
```

Included examples. To list examples

```
bombard --examples
```

CHAPTER 3

Command line

From command line you can change number of threads, loop count, supply vars, customize report and so on.

Also you can bootstrap your own `bombard.yaml` file from any example you like:

```
bombard --init --example simple
```

CHAPTER 4

Report

Example of report for the command:

```
bombard --example simple --repeat 2 --threshold 100
```

```
11 Apr 21:35:19     1 0.2 ms (thread 0) postsList >>> GET jsonplaceholder.typicode.com/posts
11 Apr 21:35:19     1 0.1 sec (thread 0) postsList <<< 200 (26.9 kb) GET jsonplaceholder.typicode.com/posts
11 Apr 21:35:19     2 0.1 ms (thread 0) getPost >>> GET jsonplaceholder.typicode.com/posts/1
11 Apr 21:35:19     3 1.0 ms (thread 2) getPost >>> GET jsonplaceholder.typicode.com/posts/1
11 Apr 21:35:19     4 1.2 ms (thread 4) getPost >>> GET jsonplaceholder.typicode.com/posts/2
11 Apr 21:35:19     5 1.5 ms (thread 3) getPost >>> GET jsonplaceholder.typicode.com/posts/2
11 Apr 21:35:19     6 2.0 ms (thread 5) getPost >>> GET jsonplaceholder.typicode.com/posts/3
11 Apr 21:35:19     7 2.4 ms (thread 6) getPost >>> GET jsonplaceholder.typicode.com/posts/3
11 Apr 21:35:19     4 73.7 ms (thread 4) getPost <<< 200 (0.3 kb) GET jsonplaceholder.typicode.com/posts/2
11 Apr 21:35:19     3 74.8 ms (thread 2) getPost <<< 200 (0.3 kb) GET jsonplaceholder.typicode.com/posts/1
11 Apr 21:35:19     7 75.0 ms (thread 6) getPost <<< 200 (0.3 kb) GET jsonplaceholder.typicode.com/posts/3
11 Apr 21:35:19     2 76.8 ms (thread 0) getPost <<< 200 (0.3 kb) GET jsonplaceholder.typicode.com/posts/1
11 Apr 21:35:19     6 76.0 ms (thread 5) getPost <<< 200 (0.3 kb) GET jsonplaceholder.typicode.com/posts/3
11 Apr 21:35:19     5 0.2 sec (thread 3) getPost <<< 200 (0.3 kb) GET jsonplaceholder.typicode.com/posts/2
11 Apr 21:35:19       (thread Main)
================================================================================================
Got `7` responses in `0.3 sec`, `21 op/sec`, 28.5 kb, 87.5 kb/sec

## success (7)
Mean: 97.1 ms, min: 71.8 ms, max: 0.2 sec

## fail
`...no fails...`

## by request name:
### postsList (1)
Mean: 0.1 sec, min: 0.1 sec, max: 0.1 sec

### getPost (6)
Mean: 95.5 ms, min: 71.8 ms, max: 0.2 sec
================================================================================================
```

CHAPTER 5

Source code

GitHub

CHAPTER 6

Documentation

6.1 Installation

```
pip install bombard --upgrade
```

If you want to use specific Python version you can use something like that

```
python3.7 -m pip install bombard --upgrade
```

6.2 Bootstrapping

To create your own `bomard.yaml` use command `--init`. By default it copy example `easy.yaml`

```
bombard --init
```

So now command `bombard` will use this local `bomard.yaml`. Edit it to adapt to your server.

If you want to use another example as base just add `--example <name>` with the example name you want:

```
bombard --init --example simple
```

To list all available examples use `--examples` like that:

```
bombard --examples
```

6.3 Campaign file

All sections are optional.

But you need section `prepare` or `ammo` so Bombard will fire some requests.

Anywhere you can user Python expressions `{}` like

```
repeat: "{args.repeat * 2}"
```

Command line arguments available as `args` in this expressions. All supply variables - as globals.

6.3.1 HTTP parameters

All HTTP parameters but URL are optional.

```
url: "{host}auth"  # fully specified URL
method: POST  # by default GET
body:  # below is JSON object for request body
    email: name@example.com
    password: admin
headers:
  json:  # the same as Content-Type: "application/json"
  Authorization: "Bearer {token}"
```

6.3.2 supply

Variables you use like `{name}` in your requests. Also you can (re)define this variable using `--supply` like:

```
bombard -s name=value,name2=value2
```

Also you can (re)define it from requests.

If you have `extract` section in a request description, it will (re)define `supply` variable with the name from this section.

And `script` section in request also can (re)define variables.

6.3.3 Request description

You use this descriptions in sections `prepare` and `ammo` described below.

Each request should have `URL` and basically that's it. If you need to, you can add other elements like that:

```
getToken:  # Name of request by your choice
  repeat: "{args.repeat * 2}"  # default - option --repeat
  url: "{host}auth"  # we use supply.base var
  method: POST  # by default GET
  headers: json  # shortcut for Content-Type: application/json
  body:  # JSON object for the request body
    email: admin@example.com
    password: admin
  extract:  # extract from request result and add to supply
    token:
```

Bombard automatically adds `application/json` to headers if in the request some JSON body specified. If you need another `Content-Type` specification just add it to `headers` section and it will redefine that default.

repeat

Override `--repeat` command line option. Number of repetitions for the request.

script

In request you can add section `script` with Python3 code. It runs after request.

It can use `supply` object and fire requests with `reload` function. Requests definitions from `ammo` section available as `ammo.request_name`.

Response to the request is available in `resp` object.

In example below we fire requests `getPost` from `ammo` section for 1st three posts we get in the response:

```
for post in resp[:3]:
  reload(ammo.getPost, id=post['id'])
```

Also you can place Python code to separate file and use it like this:

```
script: !include get_token.py
```

If you add this line it mocks all necessary objects and you can use code autocomplete in your IDE:

```
from bombard.mock_globals import *; master('path/to/you/yaml')
```

extract

Instead of script you can use section `extract` in request. It can contain map of `name: extract` pairs. For each pair Bombard will (re)define `supply` var with name `name` with value extracted from the request response as `['extract']`.

```
extract:
    name: extract
    name2: extract2
```

If `extract` is empty Bombard will use the `name`, so `name:` is the same as `name: name`.

Also you can use any custom indices you want like that

```
extract:
    token: "['data']['JWT']"  # place resp['data']['JWT'] to supply.token
```

so `name: ['name']` is the same as `name:`.

dry

If you run Bombard with `--dry` it do not make actual HTTP requests. And if you have `dry` section in request Bombard will use it as result of this `dry` request.

6.3.4 prepare

If campaign file has this section, Bombard will start fire with requests from this section.

Requests in this section can fire requests from `ammo` section, like this:

```
prepare:
  postsList:  # Get ids from posts
    url: "{host}posts"
    script: |
      for post in resp[:3]:  # fire ammo.getPost for 1st three posts in the list
        reload(ammo.getPost, id=post['id'])
```

As you see above you can send some variable not only to global `supply` but just to the request you fire.

If `prepare` section did not fire any `ammo` requests, Bombard after `prepare` will fire all requests from `ammo` section.

So, if you have only `extract` sections in `prepare` requests. Or if `scripts` in `prepare` requests do not call `reload` to fire requests from `ammo`. Then Bombard will fire all `ammo` requests after `prepare` requests.

6.3.5 ammo

If campaign file do not have `prepare` section, Bombard will just fire all requests from this section.

Each request will be repeated `--repeat` times as defined in command line (or by default value for this option).

Otherwise bombard will fire `prepare` section and after that if `prepare` requests did not fire any requests from `ammo`, bombard will fire all requests from `ammo`.

Example of `ammo` request for the request that you see in `prepare` section:

```
ammo:
  getPost:
    url: "{host}posts/{id}"  # use {host} from global supply and {id} in local supply␣
↪just for this request - see script above
```

6.4 Customizing report

To color in red request that take longer than 100ms

```
bombard --threshold
```

You can reduce output to console with `--quiet` or output all the information with `--verbose`.

There are a number of other options please look at `--help`.