bombard Documentation

Выпуск 1.18.1

Andrey Sorokin

нояб. 13, 2019

Оглавление

Глава 1

Вступление

Bombard - это инструмент для стресс-тестирования вашего HTTP-сервера. С его помощью легко и быстро описывать логику формирования запросов.

Он разработан, чтобы быть простым, но мощным инструментом для нагрузочного функционального тестирования.

Благодаря возможности исплоьзовать фрагменты кода Python вы можете легко и быстро описать сложную логику для тестов.

Отчет о тестировании показывает, сколько запросов в секунду способен обслуживать ваш сервер и с какой задержкой.

Глава 2

Описание запросов

Минимально в запросе достаточно указать URL, но легко описать и JSON

```
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:
```

В первом запросе вы можете получить токен, как в примере выше.

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'])
```

В приложение втроены примеры. Список встроенных примеров

```
bombard --examples
```

Глава 3

Командная строка

Из командной строки вы можете изменить количество потоков, количество повторов, переопределить пемеренные, настроить вид отчета и так далее.

Также вы можете загрузить свой собственный файл `bombard.yaml` из любого понравившегося вам примера

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

Глава 4

отчет

Пример отчета для команды

```
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
================================================================================================
```

Глава 5

Исходный код

GitHub

Глава 6

Документация

6.1 Установка

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

Если необходимо использовать конкретную версию Python, это можно сделать так

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

6.2 Быстрый старт

Чтобы создать свой `bomard.yaml`, используйте команду `--init`. По умолчанию она копирует пример `easy.yaml`

```
bombard --init
```

Теперь команда `bombard` будет использовать этот локальный `bomard.yaml`. Отредактируйте его под свой сервер.

Если вы хотите использовать другой пример как основу для своего файла, добавьте `--example <name>` с именем нужного примера:

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

Чтобы увидеть все доступные примеры, используйте `--examples`:

```
bombard --examples
```

6.3 Файл кампании

Все разделы не являются обязательными.

Но вам нужен раздел `prepare` или `ammo`, так что Bombard выполнит несколько запросов.

Везде, где вы можете использовать выражения Python `{}` как

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

Аргументы командной строки доступны как `args` в этих выражениях. Все переменные предложения - как глобальные.

6.3.1 Параметры HTTP

Все параметры HTTP, кроме URL, являются необязательными.

```
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

Переменные, которые вы используете как `{имя}` в своих запросах. Также вы можете (пере) определить эту переменную используя `--supply` наподобие

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

Также вы можете (пере) определить его из запросов.

Если в описании запроса есть раздел `extract`, он (пере) определит переменную `supply` с именем из этого раздела.

Раздел `script` в запросе также может (пере) определять переменные.

6.3.3 Описание запроса

Вы используете это описание в разделах `prepare` и `ammo`, описанных ниже.

Каждый запрос должен иметь `URL` и в принципе этого достаточно. Если вам нужно, вы можете добавить другие элементы, например:

```
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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    password: admin
  extract:  # extract from request result and add to supply
    token:
```

Bombard автоматически добавляет `application/json` в http headers, если в запросе есть JSON body. Если вам нужно переопределить этот `Content-Type`, то просто укажите свой в разделе `headers`.

repeat

Переопределить `--repeat` параметр командной строки. Количество повторений для запроса.

script

В запросе вы можете добавить раздел `script` с кодом Python3. Он запускается после запроса.

Он может использовать объект `supply` и запускать запросы с функцией `reload`. Запрашивает определения из раздела `ammo`, доступные как `ammo.request_name`.

Ответ на запрос доступен в объекте `resp`.

В приведенном ниже примере мы запускаем запросы `getPost` из раздела `ammo` для первых трех постов, которые мы получаем в ответе:

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

Также вы можете поместить код Python в отдельный файл и использовать его следующим образом:

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

Если вы добавите эту строку, она проверяет все необходимые объекты, и вы можете использовать автозаполнение кода в вашей IDE:

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

extract

Вместо скрипта вы можете использовать раздел `extract` в запросе. Может содержать карту пар `name: extract`. Для каждой пары Bombard будет (пере) определять `supply` var с именем `name` со значением, извлеченным из ответа на запрос как `['extract']`.

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

Если `extract` пусто, Bombard будет использовать `name`, поэтому `name:` совпадает с `name: name`.

Также вы можете использовать любые пользовательские индексы, которые вы хотите, например

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

поэтому `name: ['name']` совпадает с `name:`.

dry

Если вы запускаете Bombard с `--dry`, он не делает реальных HTTP-запросов. И если у вас есть раздел `dry` в запросе, Bombard будет использовать его как результат этого запроса `dry`.

6.3.4 prepare

Если в файле кампании есть этот раздел, Bombard начнет стрелять по запросам из этого раздела.

Запросы в этом разделе могут запускать запросы из раздела `ammo`, например:

```
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'])
```

Как вы видите выше, вы можете отправить некоторую переменную не только глобальному `supply`, но и просто на ваш запрос.

Если в разделе `prepare` не было выполнено ни одного запроса `ammo`, Bombard после `prepare` будет запускать все запросы из раздела `ammo`.

Итак, если у вас есть только `extract` разделы в `prepare` запросах. Или если `scripts` в запросах `prepare` не вызывает `reload` для запуска запросов из `ammo`. Затем Bombard будет запускать все запросы `ammo` после запросов `prepare`.

6.3.5 ammo

Если в файле кампании нет раздела `prepare`, Bombard просто запустит все запросы из этого раздела.

Каждый запрос будет повторяться `--repeat` раз, как указано в командной строке (или по умолчанию для этой опции).

В противном случае бомбард будет запускать раздел `prepare`, и после этого, если запросы `prepare` не сработали ни одного запроса от `ammo`, то бомбардировка будет запускать все запросы из `ammo`.

Пример запроса `ammo` для запроса, который вы видите в разделе `prepare`:

```
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 Настройка отчета

Для окрашивания в красный запрос, который занимает более 100 мс

```
bombard --threshold
```

Вы можете уменьшить вывод на консоль с помощью `--quiet` или вывести всю информацию с помощью `--verbose`.

Есть ряд других опций, пожалуйста, посмотрите на `--help`.