

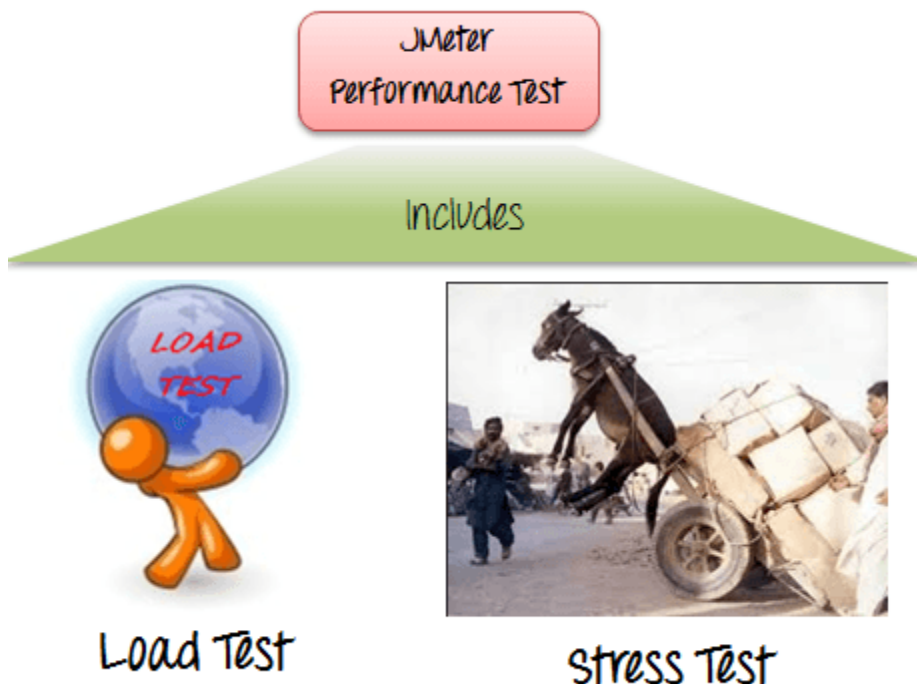
Инструменти за стрес-тестове на уеб приложения

Разработено от Виктор Христов, 62151

Contents

Инструменти за стрес-тестове на уеб приложения.....	1
Основни понятия	2
Тестване на натовареност (Load Testing)	2
Стрес тестване (Stress Testing)	2
JMeter	3
Locust	6
Gatling	8
NeoLoad	12
WebLOAD	14
Rational Performance Tester	17
Silk Performer	20
LoadRunner Cloud.....	22
WAPT.....	24
Цитирана литература.....	27
Текстово съдържание	27
Фигури	28
Списък с фигури.....	29

ОСНОВНИ ПОНЯТИЯ



Фигура 1. Тестване на натовареност

Тестване на натовареност (Load Testing)

Тестването на натовареност е вид нефункционално тестване и е подмножество на тестването на производителност. При този вид тестване се изчислява производителността на едно уеб приложение когато конкретен брой потребители го използват едновременно.

Целта на тестването на натовареност е да се осигури плавното функциониране на приложението при натовареност, отговаряща на тази в реалността. То помага за откриване на проблеми, засягащи:

- Време за отговор на заявка
- Производителността на системните компоненти при различни натоварености
- Производителността на компонентите на базата данни при различни натоварености
- Връзката между клиент и сървър
- Дизайна на софтуера
- Конфигурация на сървъра
- Ограничения на хардуера [\[1\]](#)

Стрес тестване (Stress Testing)

Подобно на тестването на натовареност, стрес тестването също проверява поведението на уеб приложение при определен товар. Това, което ги разграничава е, че докато тестването на натовареност набляга върху производителността на една система, стрес тестването набляга върху нейната устойчивост и способност да обработва грешки.

Една от целите на стрес тестването е да осигури, че системата няма да се срина при непредвидени ситуации. Чрез огромен брой входни данни то определя границата на натовареността, при която ще се счупи системата.

Друга цел е на стрес тестването да осигури, че системата правилно се възстановява след като се премине границата на натовареност, т.е да се провери поведението на системата след като вече е настъпил срив.

Стрес тестването е нужно поради следните причини:

- То проверява до каква степен работи система при голяма натовареност
- То проверява дали правилно се представят съобщенията за грешки при голяма натовареност
- Срыв на системата при критични ситуации ще доведе до значителни финансови загуби
- То подобрява като цяло колко подготвена е една система да справи с огромни натоварености [\[2\]](#)

JMeter



Фигура 2. Лого на JMeter

JMeter е инструмент за стрес тестване с отворен код, разработено изцяло на Java. Той е създаден за тестване на натовареността на функционалностите и изчисляване на производителност. Приложението позволява тестване на производителността както на статични, така и на динамични ресурси. Може да се симулира натовареност върху обект, сървър, група от сървъри или мрежа. Така то изследва тяхната издръжливост и ефективност при различни степени на натовареност. [\[3\]](#)

Тестване на натовареност чрез JMeter се извършва по следния начин:

1. JMeter създава и изпраща заявка към сървъра като нормален уеб браузър
2. JMeter получава отговор от сървъра
3. Тези отговори се събират и визуализират под формата на диаграма
4. JMeter обработва информацията, получена от сървъра
5. JMeter генерира тестов резултат в различни формати (текст, XML, JSON)
6. Тестерът анализира получения резултат [\[4\]](#)

Test report



Create request
to target server

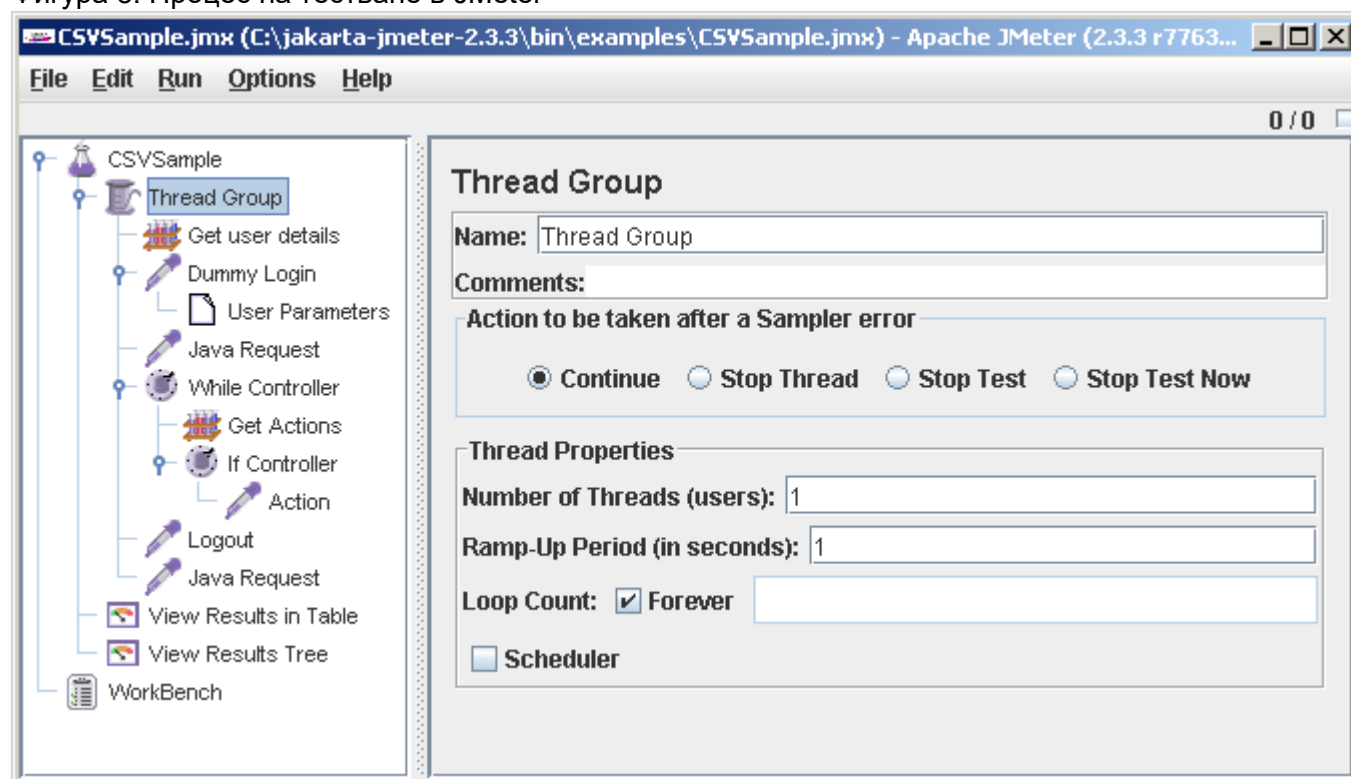
Collect and
calculate
statistical info

*Jmeter simulates
multiple users
sending request to
target server, and
returns the
performance result
of the target*

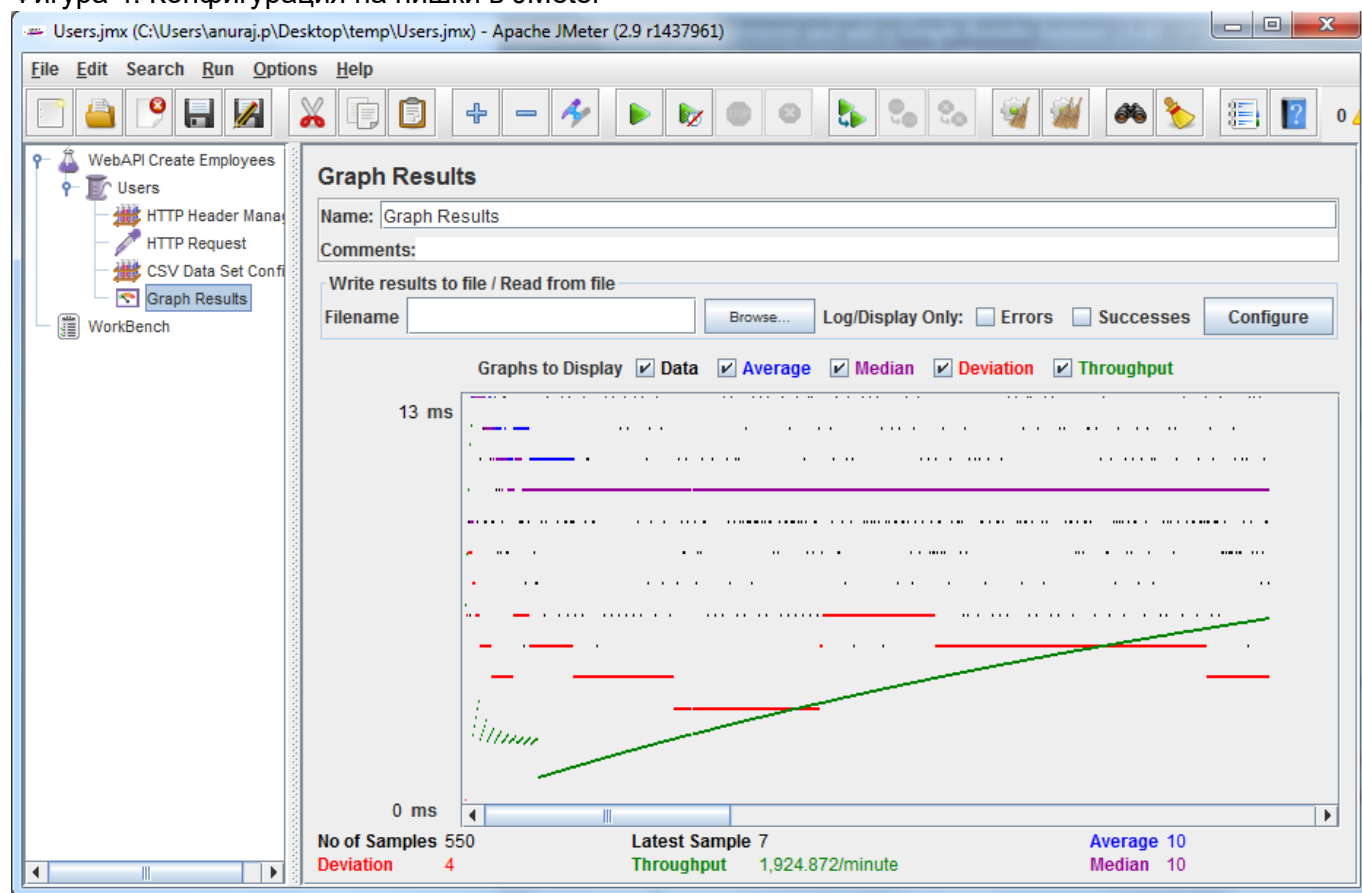
Server
responds

Saves all
responses

Фигура 3. Процес на тестване в JMeter



Фигура 4. Конфигурация на нишки в JMeter



Фигура 5. Резултат от тестване в JMeter

Предимства и функционалности на JMeter:

1. Приложението е с отворен код, което означава, че е напълно безплатно

2. Независим откъм платформата, поради факта, че JMeter е десктоп приложение
3. Лесна инсталация, като е нужно единствено да се копира и изпълни .bat файла
4. Чрез симулиране на различни потребители, които работят на паралелни нишки JMeter може да достигне желаната натовареност
5. Тестовите резултати могат да се представят по четири различни начина: като таблица, лог файл, дърво или диаграма
6. Поддържат се всички базови протоколи: HTTP, SOAP, LDAP, JDBC, JMS, and FTP
7. Интеграция с библиотеките Jenkins, Maven, and Gradle

Недостатъци на JMeter:

1. Симулациите изискват много памет за изпълнение и генериране на тестов отчет
2. JMeter е добър инструмент за тестване на уеб приложения, но не е удобен за десктоп приложения
3. JMeter е десктоп приложение, а не браузърно. Поради това то има трудности при обработката на информация от JavaScript и AJAX, което може да доведе до неточности при симулацията

Locust



Фигура 6. Лого на Locust

Locust е инструмент за стрес тестване с отворен код. Той е предназначен за тестване на уеб сайтове и изчисляване на колко потребителя системата може да поддържа паралелно. Идеята зад инструмента е всеки тест да симулира „рояк от скакалци“, които ще нападнат тестваната система. Поведението на всеки „скакалец“ (тестов потребител) се определя от тестера и той може да наблюдава как се развива теста в реално време чрез потребителски интерфейс.

Друга причина за разработването на Locust е да се избегне чупливостта, от която страдат подобни инструменти за стрес тестване. Locust е базиран на изпълнението на събития, което дава възможност да се поддържат потенциално хиляди конкурентни потребителя на една машина. За разлика от други приложения, които използват събития, Locust не използва callback функции. Вместо тях използва леки процеси, реализирани чрез `gevent`, библиотека на Python за доставяне на синхронизационен приложен програмен интерфейс на високо ниво. Всеки отделен „скакалец“ се изпълнява в собствен процес, което позволява силна гъвкавост при писането на тестове с Python без да се усложнява кода чрез прилагане на callback функции. [\[5\]](#)

Start new Locust swarm

Number of users to simulate

Hatch rate (users spawned/second)

Фигура 7. Задаване на максимален брой потребители и колко потребителя да се добавят всяка секунда към теста в Locust

[Statistics](#) [Failures](#) [Exceptions](#) [Download Data](#)

Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/	40	0	170	256	114	953	13263	0.2
GET	/5/campaign/	68	0	190	304	103	1335	28006	0.6
GET	/accounts/login/	0	0	0	0	0	0	0	0
POST	/accounts/login/	1	0	1600	1599	1599	1599	13263	0
GET	/api/advertiser/	125	0	150	194	88	1870	119	1.5
POST	/auth/login/	277	0	250	378	124	4002	188	3
Total		511	0	250	316	0	4002	4922	5.3

Фигура 8. Изпълнение на тест и отчитане на брой грешки в Locust

Statistics Failures Exceptions Download Data

# fails	Method	Name	Type
6	GET	/	HTTPError(u'500 Server Error: Internal Server Error for url: http://127.0.0.1:8000/accounts/login/?next=/',)
1	GET	/	ConnectionError(MaxRetryError("HTTPConnectionPool(host='127.0.0.1', port=8000): Max retries exceeded with url: /accounts/login/?next=/ (Caused by NewConnectionError('<requests.packages.urllib3.connection.HTTPConnection object at 0x7fda8f45ca50>: Failed to establish a new connection: [Errno 110] Connection timed out',))",))
1	GET	/	ConnectionError(MaxRetryError("HTTPConnectionPool(host='127.0.0.1', port=8000): Max retries exceeded with url: / (Caused by NewConnectionError('<requests.packages.urllib3.connection.HTTPConnection object at 0x7fda8d4ffb10>: Failed to establish a new connection: [Errno 110] Connection timed out',))",))
38	GET	/	ConnectionError(ProtocolError('Connection aborted.', error(104, 'Connection reset by peer')))
1	GET	/	ConnectionError(MaxRetryError("HTTPConnectionPool(host='127.0.0.1', port=8000): Max retries exceeded with url: /accounts/login/?next=/ (Caused by NewConnectionError('<requests.packages.urllib3.connection.HTTPConnection object at 0x7fda8ca00c10>: Failed to establish a new connection: [Errno 110] Connection timed out',))",))
1	GET	/	ConnectionError(MaxRetryError("HTTPConnectionPool(host='127.0.0.1', port=8000): Max retries exceeded with url: / (Caused by NewConnectionError('<requests.packages.urllib3.connection.HTTPConnection object at 0x7fda8df2a6d0>: Failed to establish a new connection: [Errno 110] Connection timed out',))",))
12	GET	/5/campaign/	HTTPError(u'500 Server Error: Internal Server Error for url: http://127.0.0.1:8000/accounts/login/?next=/5/campaign/',)
1	GET	/5/campaign/	ConnectionError(MaxRetryError("HTTPConnectionPool(host='127.0.0.1', port=8000): Max retries exceeded with url: /accounts/login/?next=/5/campaign/ (Caused by NewConnectionError('<requests.packages.urllib3.connection.HTTPConnection object at 0x7fda94030150>: Failed to establish a new connection: [Errno 110] Connection timed out',))",))

Фигура 9. Информация за настъпили грешки по време на тестване в Locust

Предимства и функционалности на Locust:

1. Инструментът е написан предимно за тестване на уеб приложения, но също позволява тестване и на всяка друга система
2. Тестовите изглеждат и се изпълняват като обикновен код, написан на Python
3. Поддържат се стрес тестове, разпределени на няколко машини
4. Всяка инстанция на Locust позволява хиляди виртуални потребители в един процес
5. Интерфейсът на Locust е разработен с JavaScript и HTML
6. Показва важна за тестера информация в реално време
7. Поради факта, че е базиран на уеб, той е лесно разширяем и може да се приложи в множество платформи
8. Всичките сложни операции на инструмента се поемат от библиотека `gevent`

Недостатъци на Locust:

1. След като се достигне определен брой потребители в тест статистиките се изтриват и се започва отново събирането на отчети, което води до загуба на информация
2. Инструментът няма да поиска някои ресурси, например URL адрес, без изрично да се зададе от потребителя [\[3\]](#)

Gatling



Фигура 10. Лого на Gatling

Gatling е инструмент за стрес-тестване, базиран на Scala. Предлага се версия с отворен код, Gatling, както и лицензирана версия, Gatling Frontline. Разработен е с идеята да е инструмент, който е лесен за използване, с лесна поддръжка и висока производителност.

Структурата на Gatling се разделя на четири компонента:

1. Конфигурация на HTTP протокол

Помага да се дефинира базовият URL , който ще се тества. Също така позволява да се дефинира тестовия потребител, заглавна част на езика и връзката.

2. Дефиниция на заглавна част

Изпълнява дефинирането на заглавната част на заявките, които се изпращат към сървъра.


3. Дефиниция на тестов сценарий

Уточнява какви действия ще извърши тестовият потребител, за да се симулира по-точно реално взаимодействие между клиент и сървър в приложението.

4. Дефиниция на симулация

Уточнява колко на брой тестови потребителя ще се изпълнят паралелно.

Gatling Recorder - Configuration

 **Gatling**
STRESS TOOL

Recorder mode: HTTP Proxy

Network

Listening port*: localhost HTTP/HTTPS 8000 HTTPS mode: Self-signed Certificate

Outgoing proxy: host: HTTP Username Password

Simulation Information

Package: default Class Name*: RecordedSimulation

☒ Follow Redirects? ☒ Infer html resources? ☒ Automatic Referers?
☒ Remove conditional cache headers? ☐ Save & check response bodies?

Output

Output folder*: /Users/pdalpra/Work/Gatling/testing/sbt-test/src/test/scala Browse

Encoding: Unicode (UTF-8)

Filters

Java regular expressions that matches the entire URI Strategy: Disabled

Whitelist

Blacklist

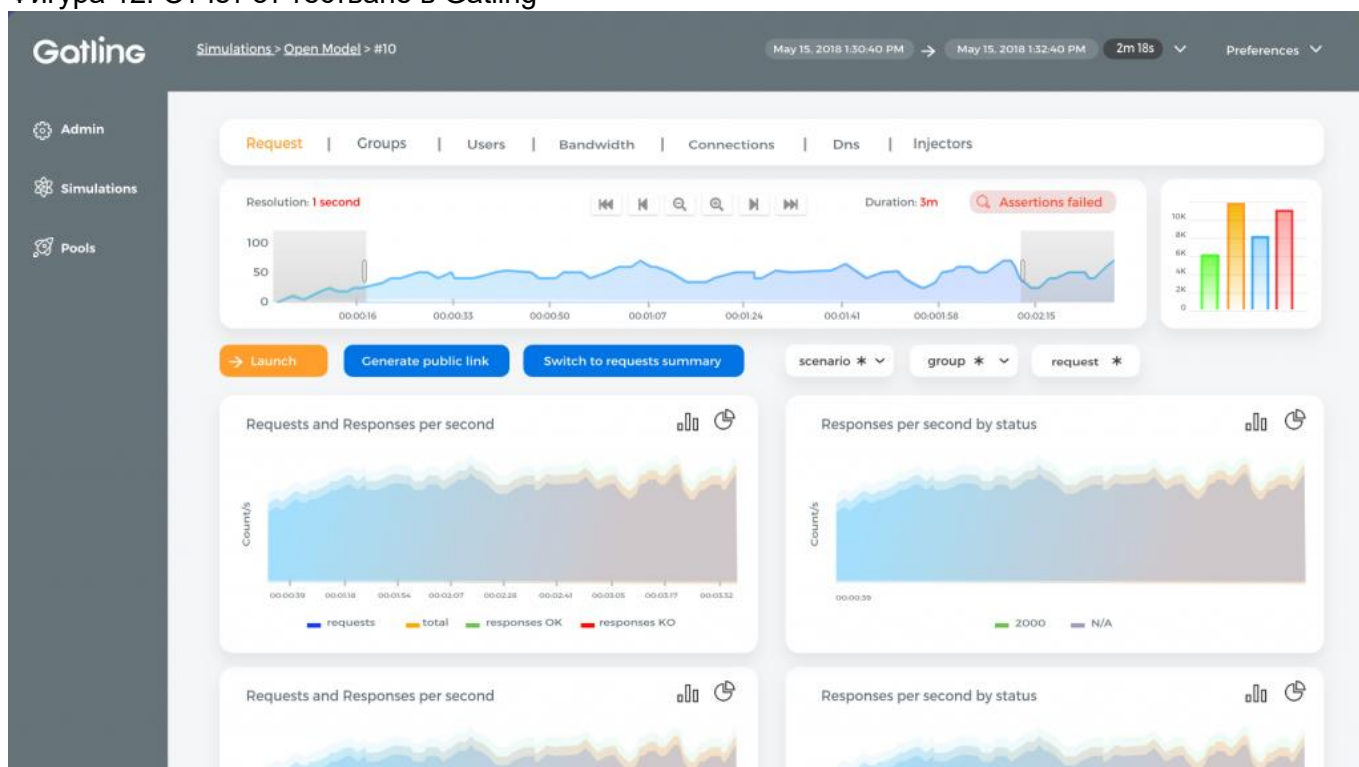
+ - Clear + - Clear No static resources

Save preferences Start !

Фигура 11. Конфигурация на протоколи в Gatling



Фигура 12. Отчет от тестване в Gatling



Фигура 13. Статистика на заявки и отговори в Gatling

Предимства и функционалности на Gatling:

1. Поддържа се от всяка операционна система и всеки браузър
2. Не се нуждае от много памет, за да изпълни тестовите сценарии

3. Тестовите сценарии могат да се изпълняват от различни тестови облаци
4. Тестовите сценарии могат да се изпълняват чрез Jenkins, Gradle и Maven чрез съответните им плъгини
5. Предлага графични отчети с полезна информация за анализ

Недостатъци на Gatling:

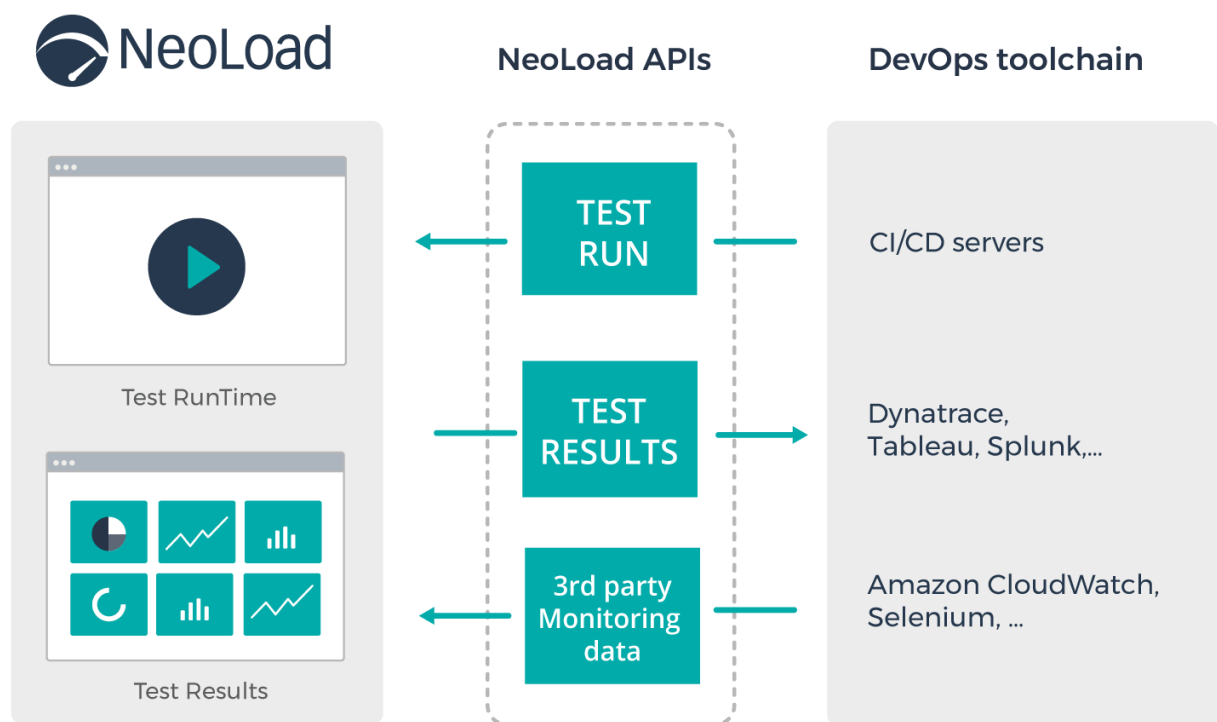
1. Не позволява хоризонтална скалируемост
2. Не позволява да се разпределя товара между различни машини
3. Приложният програмен интерфейс се е променил коренно. Това би довело до проблеми на изпълнение на тестови сценарии, написани на стари версии. [\[3\]](#)

NeoLoad



Фигура 14. Лого на NeoLoad

NeoLoad е инструмент за стрес тестване, написан на Java. Ефективен е при тестване както на уеб сайтове, така и на мобилни и десктоп приложения. Neotys, неговите създатели, предлагат безплатна и лицензирана версия, но безплатната поддържа максимум 50 паралелни потребителя.



Фигура 15. Приложен програмен интерфейс на NeoLoad


Cloud Session Creation Wizard

Create a new on-demand Cloud session

1. Create or use a session

2. Create a new session

3. Validate information



Schedule

Start date: In a few minutes

Duration: 2 hours

Load Generators

Type: Medium

☐ Use reserved IP addresses

Enter the number of required Load Generators:

Zone	Load Generators		Observed startup times	
	Required	Available	Median	Max
USA				
North-America				
South-America				
Brazil (Sao Paulo 1)	0	25	4 min	5 min
Brazil (Sao Paulo 2)	0	25	4 min	6 min
Brazil (Sao Paulo 3)	0	25	2 min	3 min
Europe				
Belgium (St. Ghislain)	0	25	2 min	3 min
Finland (Hamina)	0	25	2 min	3 min
France (Gravelines)	0	25	3 min	4 min
France (Paris 1)	0	25	5 min	9 min
France (Paris 2)	0	25	2 min	4 min
France (Paris 3)	0	25	3 min	6 min
France (Strasbourg)	0	25	3 min	4 min
Germany (Frankfurt 1)	0	25	4 min	5 min
Germany (Frankfurt 2)	0	25	8 min	12 min
Germany (Frankfurt 3)	5	25	2 min	3 min

Total number of Load Generators: 15 Medium

Reserved IP addresses: 0

1 Load Generator for 1 hour = 1 credit

Cost

Available: 33 credits

Reserved for this session: 30 credits

< Back

Next >

Finish

Cancel

Фигура 16. Създаване на облачна сесия в NeoLoad

NeoLoad

Home

Test result

Trends

Dashboard

Search

Run a Test

Resources

Documentation

Users

Account

Profile

Pipeline API Limit Test (build 108)

Overview

Values

Events

SLA

Element type

Transaction

Page

Request

Status

Failed

Warning

Passed

Global indicators

No SLAs available

If SLAs are configured in NeoLoad, they will be available at the end of the test.

Per run

User Path	Status	Element	Parent	Element type	SLA name	Value	Failed threshold	Warning threshold
Component Testing_MySQL	✓	Update OAuthSession	Init	Transaction	Error Rate (%)	0	>= 5	>= 1
Component Testing_REST	✓	Update OAuthSession	Init	Transaction	Average Transaction Response Time (s)	0.061	>= 5	>= 2
Component Testing_MySQL	✓	Update OAuthSession	Init	Transaction	Average Transaction Response Time (s)	0.06	>= 5	>= 2
Component Testing_REST	✓	Update OAuthSession	Init	Transaction	Error Rate (%)	0	>= 5	>= 1
Component Testing_REST	✓	/platform/api/v3/posts/\${RandomPost}	REST API call	Request	Error Rate (%)	0	>= 5	>= 1
Component Testing_REST	✓	/platform/api/v3/posts/\${RandomPost}	REST API call	Request	Average Request Response Time (s)	0.736	>= 3	>= 0.8
Component Testing_MySQL	✓	MySQL	Actions	Transaction	Error Rate (%)	0	>= 5	>= 1
Component Testing_MySQL	✓	MySQL	Actions	Transaction	Average Transaction Response Time (s)	0.896	>= 5	>= 2

Per time interval

User Path	Status	Element	Parent	Element type	SLA name	Failed threshold	% Failed	Warning threshold	% Warning
Component Testing_REST	✗	/platform/api/v3/posts...	REST API call	Request	Average Response Time (s)	>= 3	1.871%	>= 1	22.453%
Component Testing_MySQL	✗	MySQL	Actions	Transaction	Average Response Time (s)	>= 3	11.168%	>= 1	1.777%
Component Testing_REST	✓	Update OAuthSession	Init	Transaction	Average Response Time (s)	>= 3	0%	>= 1	0%
Component Testing_MySQL	✓	Update OAuthSession	Init	Transaction	Average Response Time (s)	>= 3	0%	>= 1	0%

Фигура 17. Валидация на тестови сценарии в NeoLoad

Предимства и функционалности на NeoLoad:

1. Удобен графичен интерфейс, който улеснява процеса на писане на тестови сценарии
2. Предлага опция за дефиниране на поведението на тестовия потребител, както и с колко на брой потребителя да се тестват
3. Позволява следене на производителността на сървъра чрез монитори за употреба на процесора, паметта и т.н
4. Обобщава резултата от тестовите сценарии под формата на диаграми и статистически таблици
5. Записва HTTP натовареност между клиента и сървъра
6. Поддържа трансформация на Selenium скриптове

Недостатъци на NeoLoad:

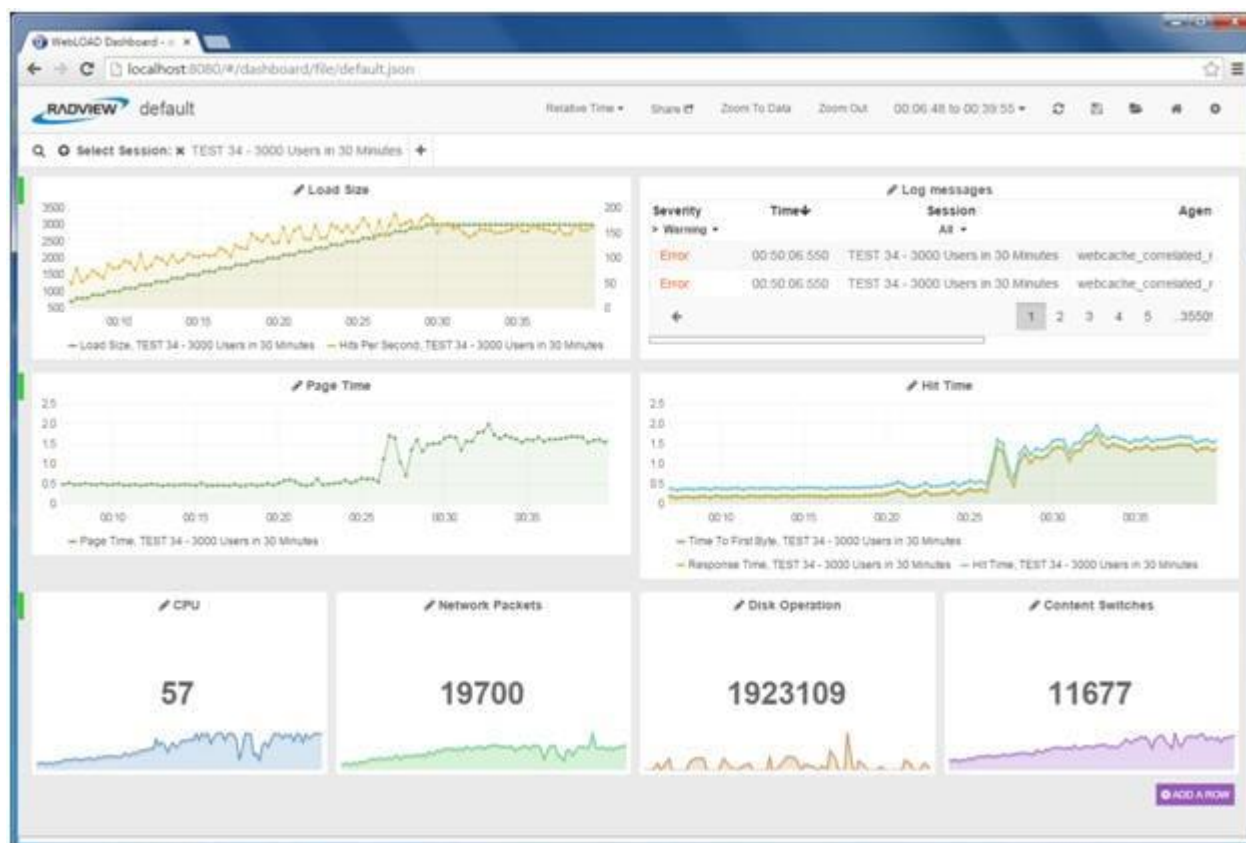
1. Не се поддържа на Mac OSX
2. Не може да достъпи DOM дървото на уеб страницата
3. Липсва опция за запис на времето за отговор на заявка [\[3\]](#)

WebLOAD



Фигура 18. Лого на WebLOAD

WebLOAD е инструмент за стрес тестване и анализи, произведено от RadView Software. Това е мощен инструмент за тестване на уеб и мобилни приложения. Неговите тестови сценарии се генерират на JavaScript и чрез опциите в средата за разработка те могат допълнително да се редактират. Инструмента се предлага като безплатна версия, която съдържа почти всички функционалности, но ограничава максималния брой конкурентни потребители до 50, и като лицензиран продукт или облачна услуга.

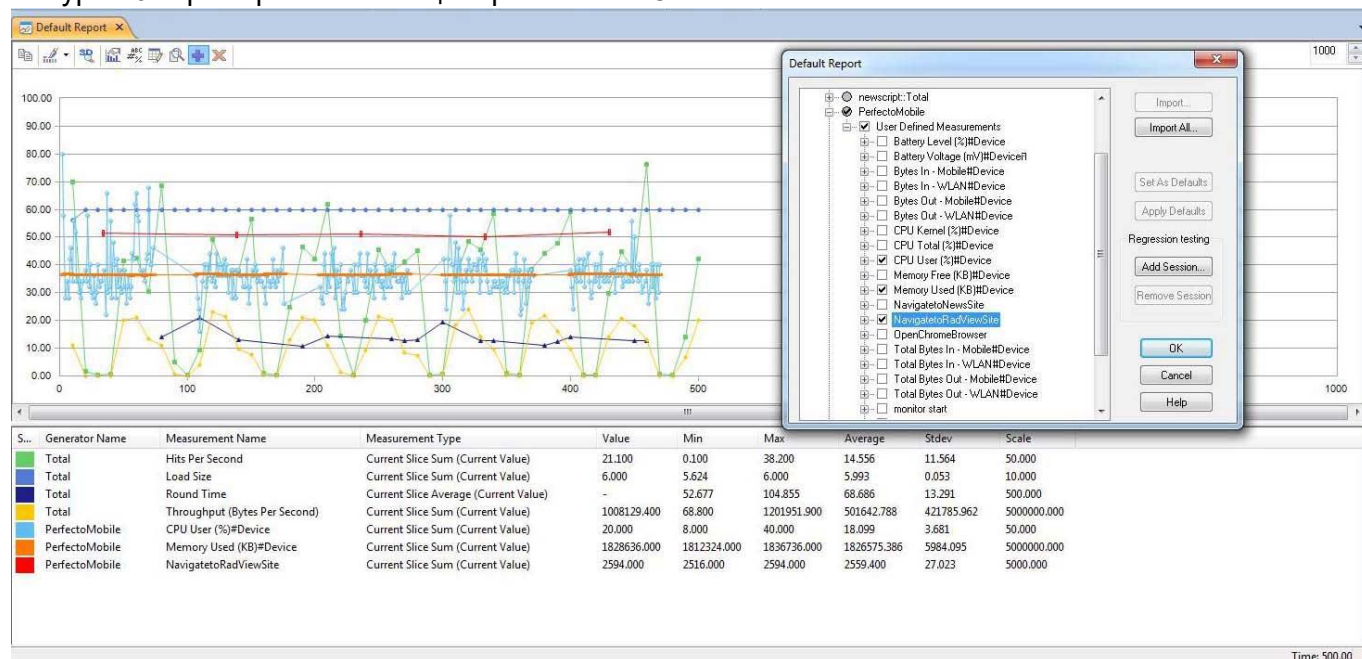


Фигура

19. Начално табло на WebLOAD

```
Page View X JavaScript View X HTTP Headers View X
Function Name: NodeScript
16 wHttp.Header["Upgrade-Insecure-Requests"] = "1"
17 wHttp.Header["Referer"] = "http://10.0.1.100:8080/"
18 //StartAuthentication
19 wHttp.UserName = "admin"
20 wHttp.Password = decrypt("JTOH6LKwg+A=")
21 //EndAuthentication
22 wHttp.Get("http://10.0.1.100:8080/manager/html")
23
24 // END WLUDE
25
26 /* ***** WLUDE - JavaScriptObject - ID 5 ***** */
27 if (document.wSource.indexOf("Jenkins")>0) {
28     if (document.wTables.length>=4) {
29         var applicationsTable = document.wTables[4];
30         var index=0;
31         for ( index=2; index< applicationsTable.rows.length ; index+=2) {
32             var currentRow = applicationsTable.rows[index];
33             var appName = currentRow.cells[0].innerText;
34             var isRunning = currentRow.cells[3].innerText;
35             DebugMessage(appName+" "+isRunning)
36
37             if (appName.indexOf("jenkins")>=0) {
38                 if (isRunning == "true") {
39                     InfoMessage("Jenkins is running, no action is required")
40                 }
41                 else {
42                     InfoMessage("Send SMS to Jim, and ask if we can start Jenkins up")
43                     //send SMS using our internal server that we have Java API to send SMS
44                     String host = "*****";
45                     int port = 9500;
46                     String username = "admin";
47                     String password = "*****";
48
49                     osc = new Packages.hu.ozekisms.MyOzSmsClient(host, port);
50                     osc.login(username, password);
51
52                     String line = "Hi Jim, Jenkins on dev is down, can I start it up?";
53
54                     if (osc.isLoggedIn()) {
55                         osc.sendMessage("*****", line);
56                         osc.logout();
57                     }
58                     break;
59                 }
60             }
61         }
62     }
}
```


Фигура 20. Примерен тестов сценарий в WebLOAD



Фигура 21. Отчет в WebLOAD

Предимства и функционалности на WebLOAD:

1. Потребителят може да симулира товар на машина или в облака
2. Потребителят има възможност да сподели резултати от тестовете, когато е част от разпределени екипи
3. Помощен потребителски интерфейс, който автоматично търси начини да подобри тестовия скрипт
4. Над 80 шаблона за отчет на резултат
5. Автоматично маркиране на потенциални места на задръстване
6. Лесна интеграция с други известни инструменти, например Selenium, Jenkins, Perfecto Mobile, New Relic, и Dynatrace

Недостатъци на WebLOAD:

1. Не поддържа Citrix
2. Не поддържа SAP GUI
3. Нужни се известни технически знания, за да се използва инструмента [\[3\]](#)

Rational Performance Tester



Фигура 22. Лого на RPT

Rational Performance Tester (RPT) е инструмент за тестване на производителност, разработен от IBM. Може да се използва за тестване на уеб приложения, както и сървърно-базирани приложения. Често се използва при подхода DevOps. Някои от целите му са да валидира скалируемостта на приложения, да разпознава силно натоварени участъци на система и да намали нуждата от стрес тестване.

Add EndPoint

Configure Protocol
Choose protocol and set the corresponding configuration options.

☒ HTTP ☐ JMS ☐ WebSphere MQ ☐ WebSphere Java MQ ☐ Microsoft .NET

Protocol configuration: Default HTTP Protocol New...

URL:

☒ Rest mode

Resource:

Parameter Separator:

Parameters:

Name	Value
BatteryStrength	99
CourierNo	655455

Add Edit Remove

Method:

Header:

Name	Value
Content-Type	application/x-www-form-urlencoded

Add Edit Remove

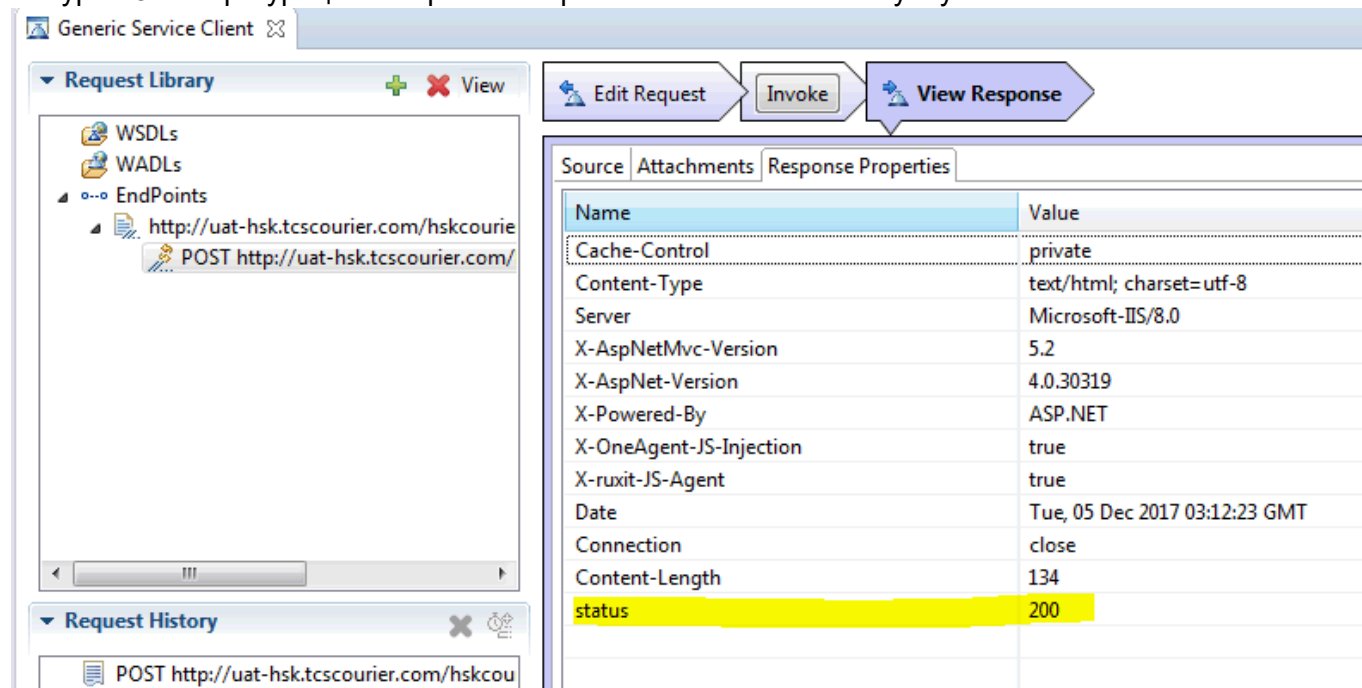
Cookies:

Name	Value
------	-------

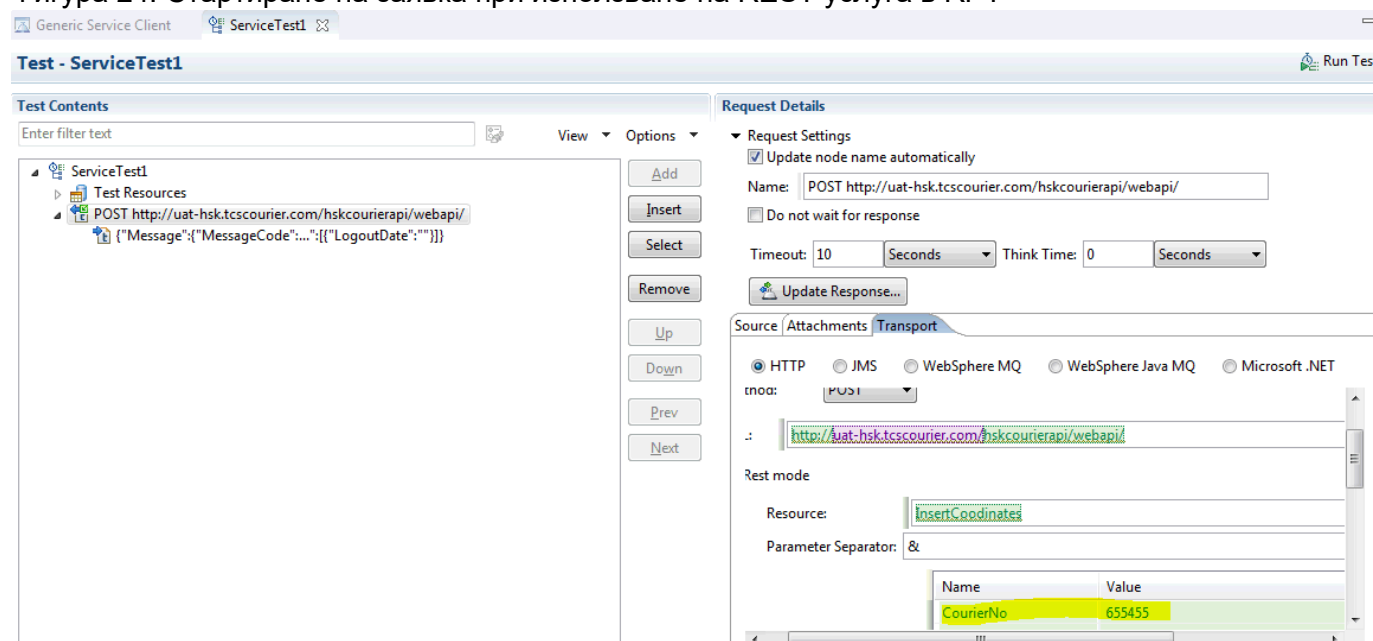
Add Edit Remove

? Finish Cancel

Фигура 23. Конфигурация на протокол при използване на REST услуга в RPT



Фигура 24. Стартиране на заявка при използване на REST услуга в RPT



Фигура 25. Информация за заявка при използване на REST услуга в RPT

Предимства и функционалности на Rational Performance Tester:

1. Не изисква владение на програмиране
2. Поддържа широка база от приложения като HTTP, SAP, Siebel, SIP, TCP Socket и Citrix
3. Докладване на срещнати проблеми при тестване в реално време
4. Позволяване на диагностика на Websphere и Weblogic приложения сървъри
5. Предлага поддръжка за множество среди и платформи

Недостатъци на Rational Performance Tester:

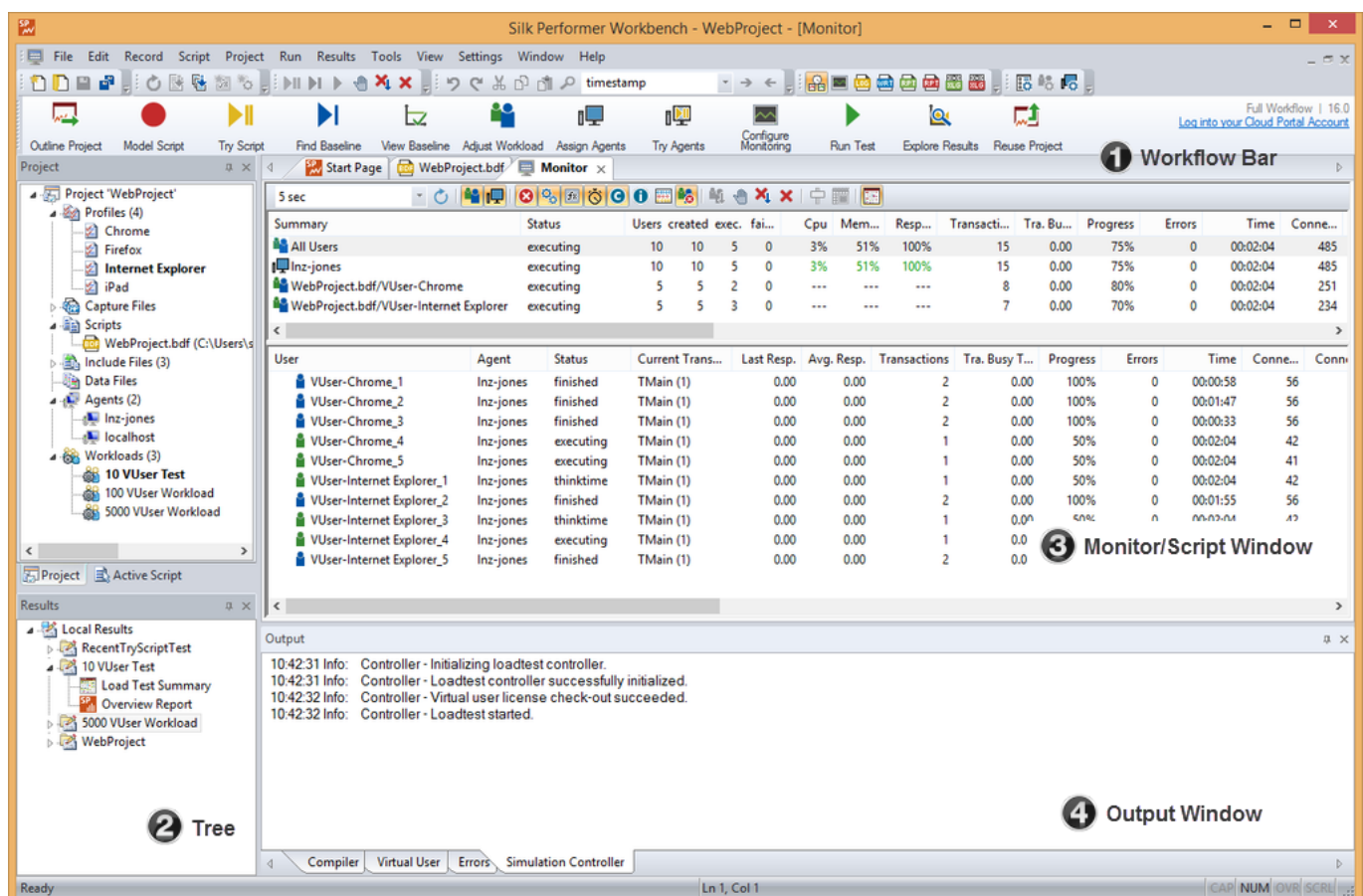
1. Не поддържа приложения, базирани на Java Applet
2. Не позволява динамично променяне на използваните ресурси [3]

Silk Performer

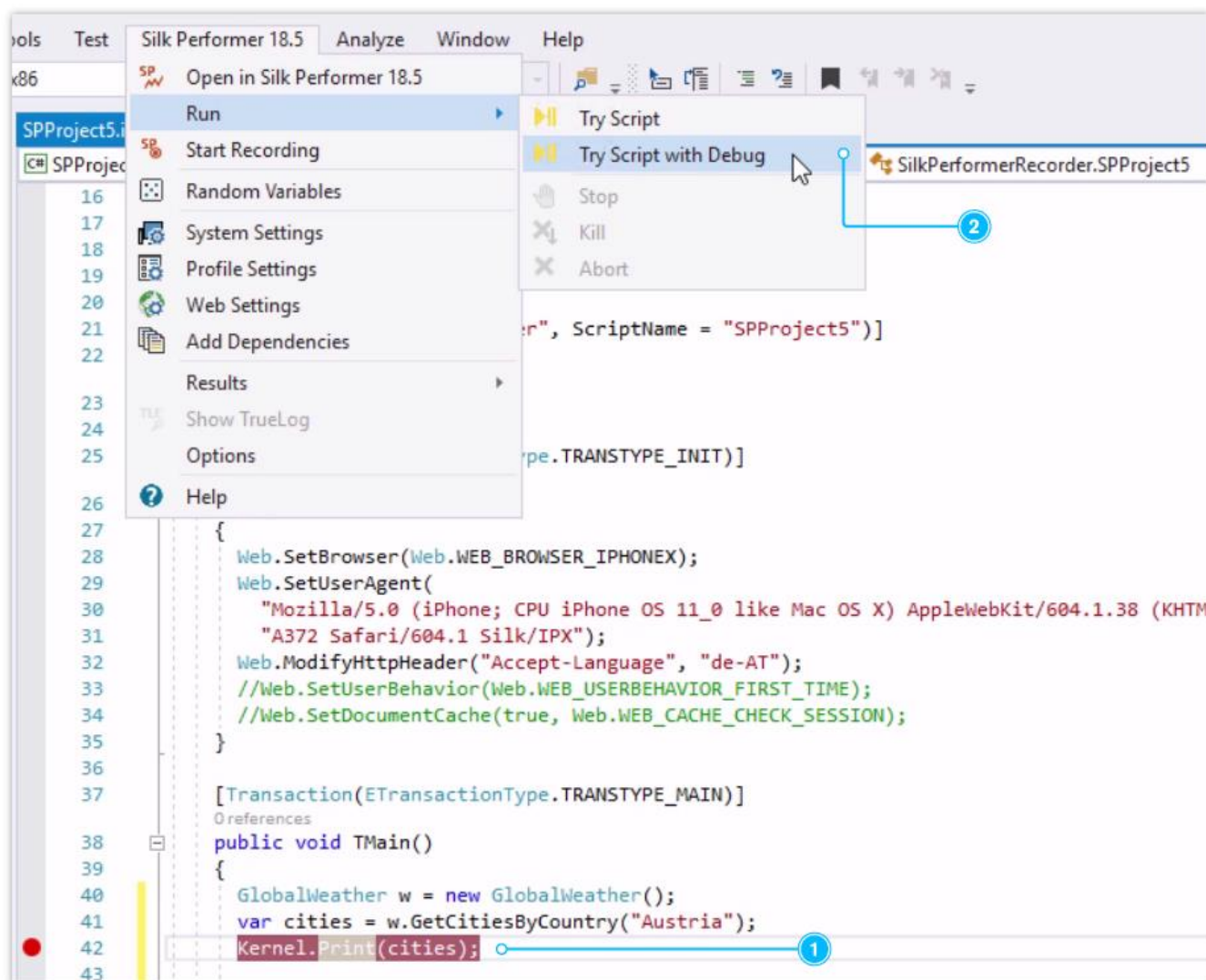


Фигура 26. Лого на Silk Performer

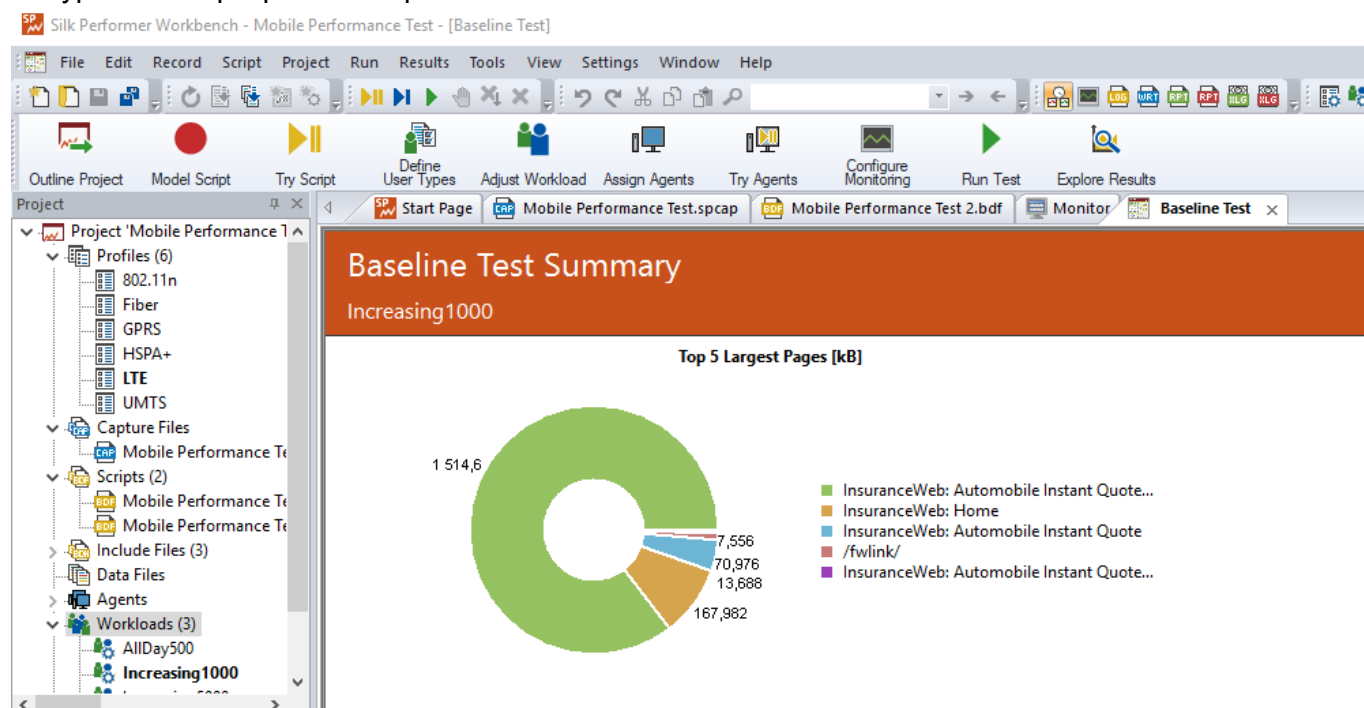
Silk Performer е инструмент за стрес тестване, разработен от Micro Focus. Използва се за тестване на уеб, мобилни и Enterprise системи. Той помага при търсенето на слабости в приложението по време на пикова натовареност и позволява да се анализират резултатите чрез вградени опции за диагностика и отчети. Софтуерът е напълно платен.



Фигура 27. Потребителски интерфейс на Silk Performer



Фигура 28. Стартиране на скрипт в Silk Performer



Фигура 29. Тестов резултат в Silk Performer

Предимства и функционалности на Silk Performer:

1. Поддържа всички основни уеб среди за разработка: HTML5, AJAX, Responsive Web Design, Adobe Flash/Flex, и Microsoft Silverlight
2. Поддържа тестване на приложение върху множество мобилни платформи и стандарти за връзка: iOS, Android, BlackBerry, GPRS, HSPA+, EDGE и LTE
3. Поддържа следните Enterprise приложения: Citrix, SAP, Oracle, MBC Remedy и Mainframe
4. Позволява вградено следене на сървъра
5. Създава информативни отчети чрез таблици и диаграми
6. Безкрайна скалируемост на облака

Недостатъци на Silk Performer:

1. Разработен е единствено за Windows
2. Лоша интеграция с инструменти за дълбока диагностика за следене на производителност от типа „край към край“
3. Стрес тестването консумира голямо количество памет [\[3\]](#)

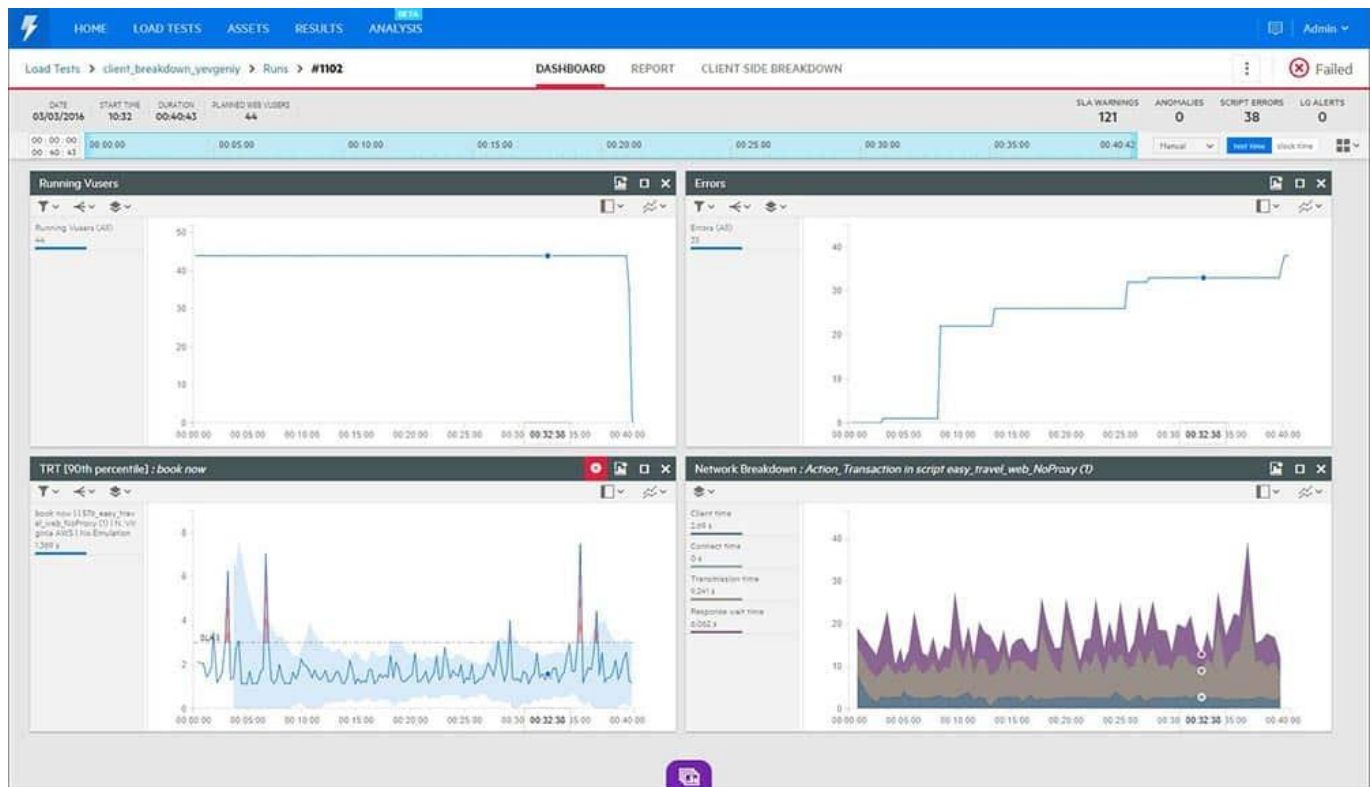
LoadRunner Cloud



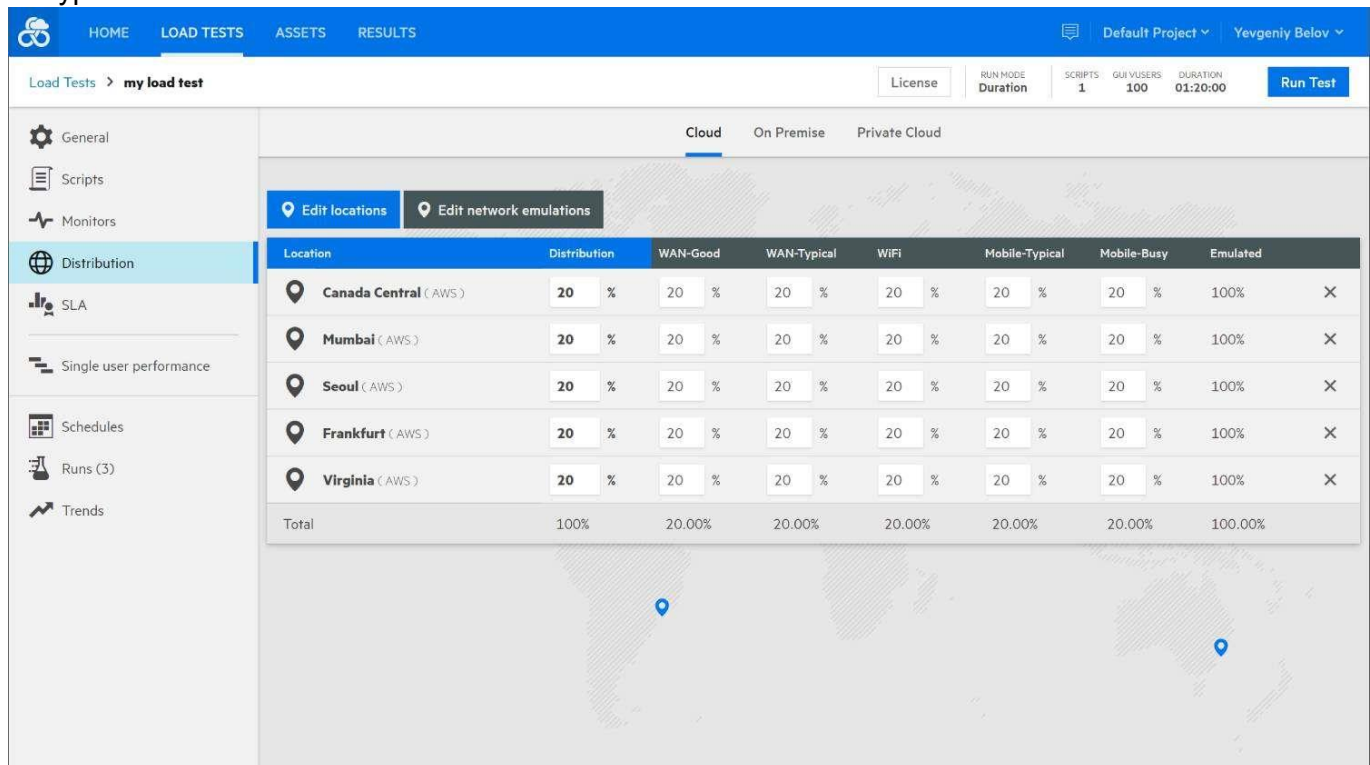
LoadRunner Cloud

Фигура 30. Лого на LoadRunner Cloud

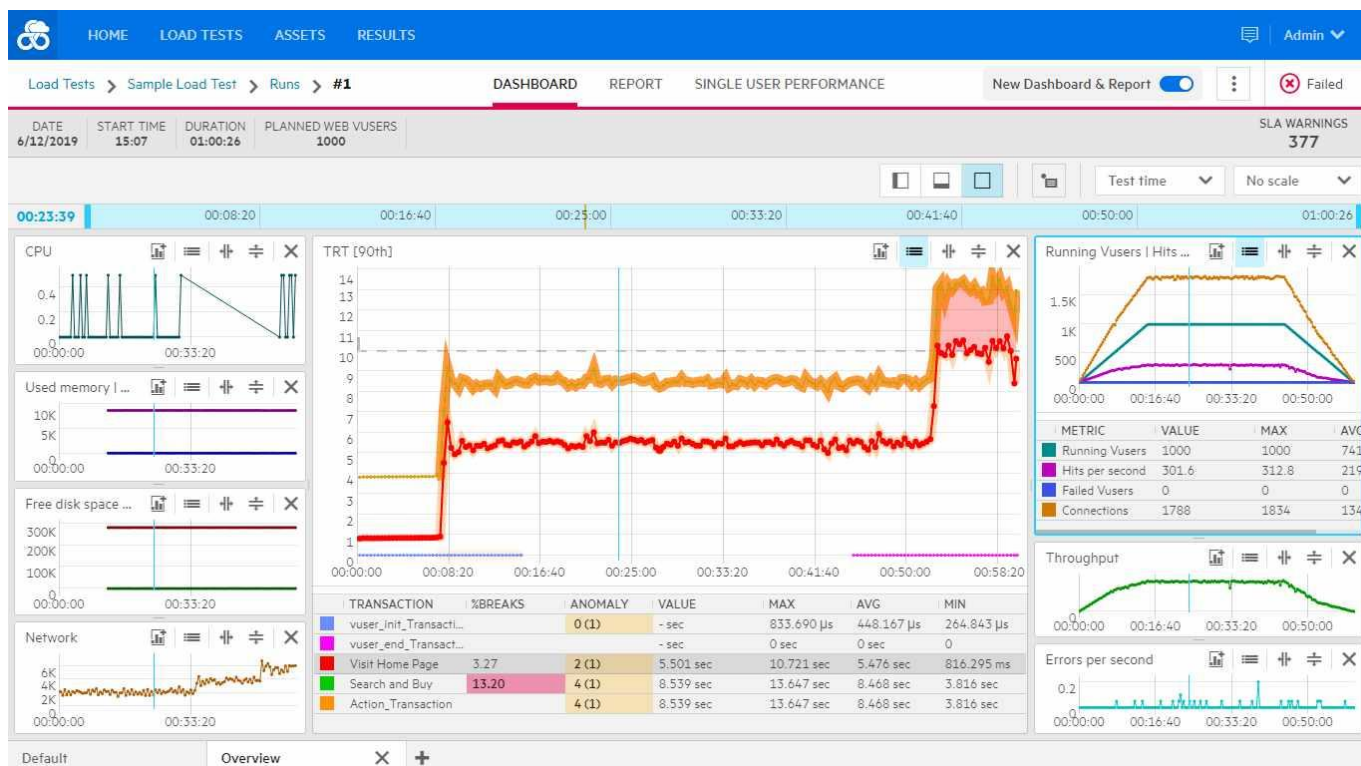
LoadRunner Cloud (познат също като StormRunner Load) е облачно-базиран инструмент за тестване на производителност, разработен от Hewlett Packard Enterprise (които в 2017 се сливат с компания Micro Focus). Той е напълно платен софтуер и се предлага като услуга. Той е подходящ за уеб и мобилни системи, които прилагат DevOps, гъвкави методологии и модела на водопада. [\[6\]](#)



Фигура 31. Начално табло на LoadRunner Cloud



Фигура 32. Разпределение на натовареността в LoadRunner Cloud



Фигура 33. Резултат от тестване в LoadRunner Cloud

Предимства и функционалности на LoadRunner Cloud:

1. Лесен за употреба
2. Поддържа HTTP/HTML, SAP Web, Java, Flex, TruClient Web, TruClient Native Mobile и TruClient Mobile Web протоколи
3. Тясна интеграция с Jenkins, Dynatrace, Gatling, Docker, AWS Code Pipeline и New Relic
4. Може да генерира натовареност от различни географски точки
5. Висока скалируемост, като позволява да се създават тестови сценарии с повече от един милион уеб или мобилни потребители
6. Виртуалните потребители могат да бъдат добавяни или премахвани по време на изпълнението на тестов сценарий
7. Бързо откриване на проблеми с тестовите сценарии
8. Добре персонализирани аналитични отчети, създадени чрез мрежова виртуализация

Недостатъци на LoadRunner Cloud:

1. Не поддържа FTP
2. Няма опция за мрежов анализ чрез Jmeter скрипт
3. Няма опция за създаване на график за тестване
4. Позволява да се прикачват единствено скриптове [3]

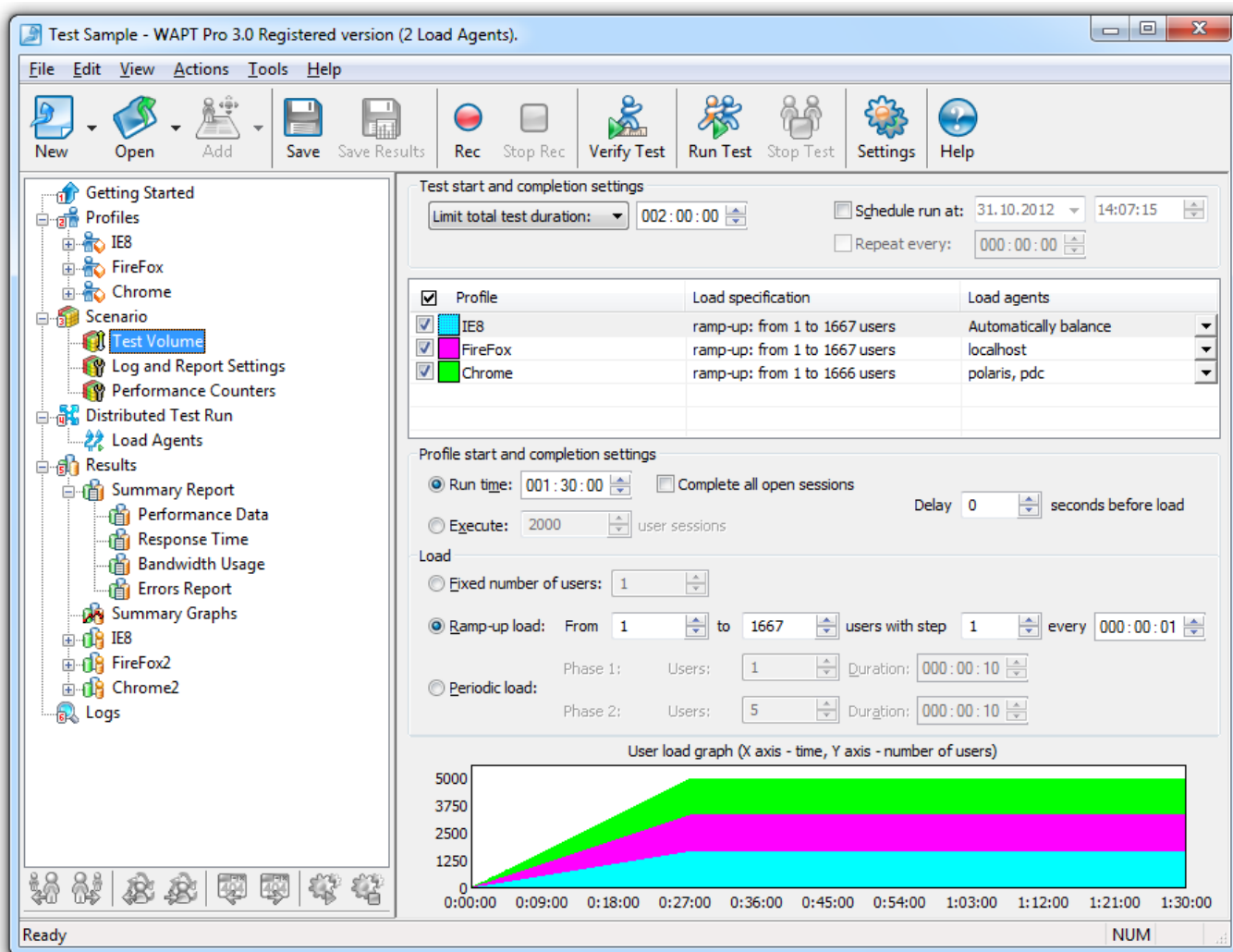
WAPT



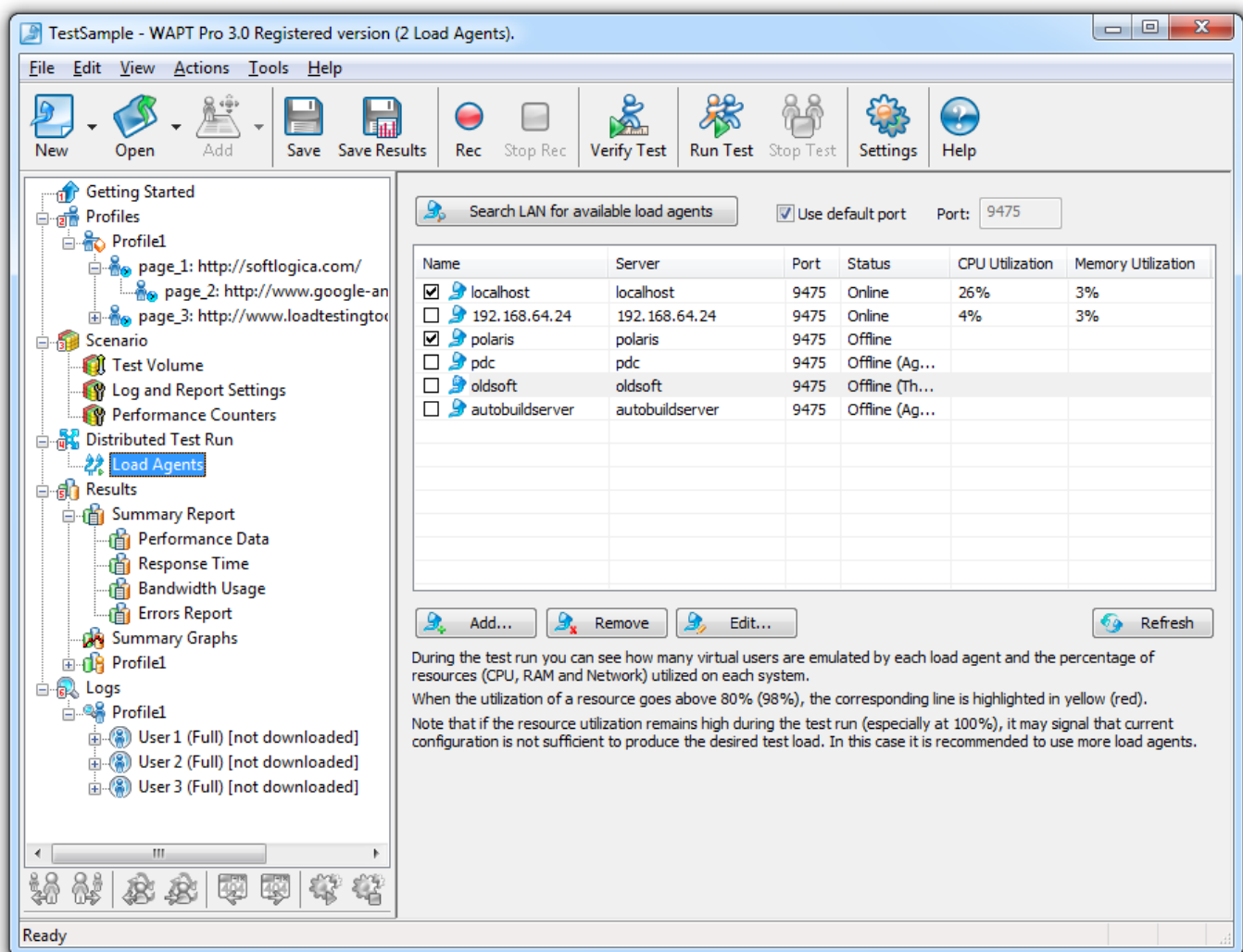
Фигура 34. Лого на WAPT

WAPT е платен софтуер за тестване на производителност, предназначен за екипи от уеб разработчици и QA специалисти. Инструментът цели да комбинира леснота за употреба, ефективност и гъвкавост. Приложим е при тестване на уеб сайтове, мобилни приложения, ERP и CRM системи, IOT платформи и мрежови API услуги. Инструментът се предлага по три начина:

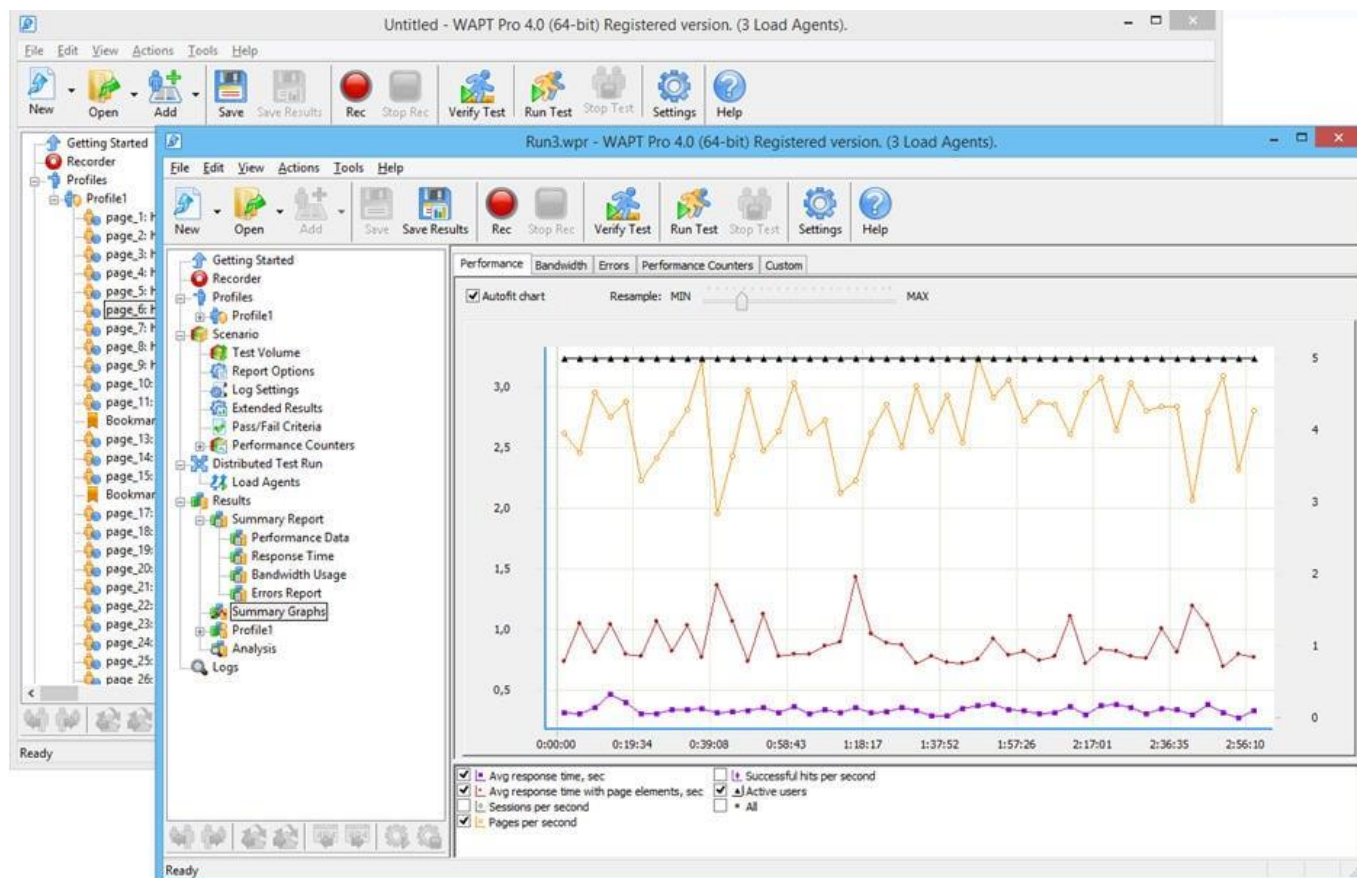
- WAPT - Базова версия с годишен лиценз, която допуска тестване с до 2000 паралелни потребителя [\[7\]](#)
- WAPT Pro - Разширена версия с годишен лиценз, която предлага допълнителни функционалности и допуска тестване с до един милион паралелни потребителя [\[8\]](#)
- WAPT Cloud - Облачна услуга, която се заплаща на час в зависимост от размера на провежданите тестове [\[9\]](#)



Фигура 35. Конфигурация на тест в WAPT



Фигура 36. Разпределяне на виртуални потребители в WAPT



Фигура 37. Тестов резултат в WAPT

Предимства и функционалности на WAPT:

1. Лесен за инсталиране, а при използване на облачната услуга дори няма нужда да се инсталира
2. Поддържа също и RIA технологии
3. Позволява тестване на сигурността на уеб сайтове
4. Бързо проектиране и записване на тестови сценарии
5. Позволява да се включва изпълнение на JavaScript код в поведението на виртуалните потребители
6. Позволява разглеждане на добре-илюстрирани и подробни отчети дори по време на изпълнение на тестови сценарии
7. Добре развито докладване на грешки спрямо различни фактори

Недостатъци на WAPT:

1. Може да се инсталира единствено на Windows OS
2. Няма опция за писане на скриптове [3]

Цитирана литература

Текстово съдържание

- [1] Guru99 Tech Pvt Ltd, "Load Testing Tutorial: What is? How to? (with Examples)", [<https://www.guru99.com/load-testing-tutorial.html>], последно посетен на 24-04-2020
- [2] Guru99 Tech Pvt Ltd, "What is STRESS Testing in Software Testing? Tools, Types, Examples",

- [<https://www.guru99.com/stress-testing-tutorial.html>],
последно посетен на 24-04-2020
- [3] Яна Густы, "TOP 20 tools for load testing in 2018", публикувано на 08-05-2017,
[<https://geteasyqa.com/blog/best-tools-load-testing/>],
последно посетен на 24-04-2020
- [4] ElysiumAcademy Private Limited, "What is JMeter? Why use JMeter? Advantages and disadvantages", публикувано на 03-06-2017,
[<https://www.linkedin.com/pulse/what-jmeter-why-use-advantages-disadvantages-private-limited>],
последно посетен на 24-04-2020
- [5] Джонатан Хейман и други, "Locust Documentation, Getting started, What is Locust?",
[<https://docs.locust.io/en/latest/what-is-locust.html>],
последно посетен на 24-04-2020
- [6] Calleo Consultants Ltd, "StormRunner Load (also known as LoadRunner Cloud)",
[<https://www.calleosoftware.co.uk/products/performance-testing/stormrunner-load>],
последно посетен на 24-04-2020
- [7] SoftLogica, "WAPT 10: Performance testing tool for web and mobile applications",
[<https://www.loadtestingtool.com/product.shtml>],
последно посетен на 24-04-2020
- [8] SoftLogica, "WAPT Pro 5: Scalable performance testing solution",
[<https://www.loadtestingtool.com/pro.shtml>],
последно посетен на 24-04-2020
- [9] SoftLogica, "WAPT Cloud: On-demand performance testing solution in the cloud",
[<https://www.loadtestingtool.com/cloud-testing.shtml>],
последно посетен на 24-04-2020

Фигури

- Guru99 Tech Pvt Ltd, "How to Use JMeter for Performance & Load Testing",
[<https://www.guru99.com/jmeter-performance-testing.html>],
използван за Фиг. 1
- Яна Густы, "TOP 20 tools for load testing in 2018", публикувано на 08-05-2017,
[<https://geteasyqa.com/blog/best-tools-load-testing/>],
използван за Фиг. 3
- Anuraj, "Load testing Web API using Apache JMeter", публикувано на 05-09-2013
[<https://dotnetthoughts.net/load-testing-web-api-using-apache-jmeter/>],
използван за Фиг. 4 и фиг. 5
- Apcelent, "Load Testing a Django Application using LocustIO", публикувано на 19-05-2017,
[<https://blog.apcelent.com/load-test-django-application-using-locustio.html>],
използван за Фиг. 7, Фиг. 8 и Фиг. 9
- Shasta Digital Technologies, "How to write SIMULATIONS using GATLING",
публикувано на 30-12-2016,
[<http://shastadigitaltechnologies.blogspot.com/search?updated-max=2016-12-30T16:15:00%2B05:30&max-results=20&start=4&by-date=false>],
използван за Фиг. 11
- Марко Стапфнер, "Load Testing using Gatling.io 3.0 for Beginners", публикувано на 08-01-2019
[<https://medium.com/@markostapfner/load-testing-using-gatling-io-3-0-for-beginners-75a9b3f93f62>],
използван за Фиг. 12
- Gatling Corp, "Gatling - Load test as code",

- [<https://gatling.io/>],
използван за Фиг. 13
- Neotys, “NeoLoad Features”,
[<https://www.neotys.com/neoload/features>],
използван за Фиг. 15, Фиг. 16 и Фиг. 17
- Software Testing Help, “WebLOAD Review – Getting Started With WebLOAD Load Testing Tool”, публикувано на 10-11-2019
[<https://www.softwaretestinghelp.com/webload-load-testing-tool-review/>],
използван за Фиг. 19, Фиг. 20 и Фиг. 21
- Киран Бирапа, “How To Test The REST API Service With Rational Performance Tester”, публикувано на 29-01-2018,
[<http://www.testworkbench-community.com/blogs/how-to-test-the-rest-api-service-with-rational-performance-tester>],
използван за Фиг. 23, Фиг. 24 и Фиг. 25
- Micro Focus, “Silk Performer Help, Silk Performer Workbench 20.5, Getting Started, Tour Of The UI”,
[<https://www.microfocus.com/documentation/silk-performer/205/en/silkperformer-205-webhelp-en/SILKPERF-043F6D15-TOUROFTHEUI-CON.html>],
използван за Фиг. 27
- davidko, “Creating Tests with Visual Studio”, публикувано на 11-01-2018,
[<https://community.microfocus.com/t5/Application-Delivery-Management/Creating-Tests-with-Visual-Studio/ba-p/1689228>],
използван за Фиг. 28
- Micro Focus, “Powerful, Realistic Load and Stress Testing”,
[<https://www.microfocus.com/en-us/products/silk-performer/overview>],
използван за Фиг. 29
- Orasi Software, “Cloud Load Testing: Micro Focus LoadRunner Cloud”,
[<https://www.orasi.com/continuous-testing/performance-testing/cloud-load-testing-micro-focus-loadrunner-cloud/>],
използван за Фиг. 31
- Micro Focus, “LoadRunner Cloud”,
[<https://www.microfocus.com/en-us/products/loadrunner-cloud/overview>],
използван за Фиг. 32 и Фиг. 33
- Web Design Dev, “WAPT – Web Application Load, Stress and Performance Testing Review”,
[<https://www.webdesigndev.com/wapt-web-application-load-stress-and-performance-testing-review/>],
използван за Фиг. 35 и Фиг. 36
- Software Testing Help, “The Beginner’s Guide To Web Application Performance Testing Using WAPT Pro”,
[<https://www.softwaretestinghelp.com/wapt-pro-load-test-tool-review/>],
използван за Фиг. 37

Списък с фигури

- [Фигура 1. Тестване на натовареност](#)
- [Фигура 2. Лого на JMeter](#)
- [Фигура 3. Процес на тестване в JMeter](#)
- [Фигура 4. Конфигурация на нишки в JMeter](#)
- [Фигура 5. Резултат от тестване в JMeter](#)
- [Фигура 6. Лого на Locust](#)

- [Фигура 7. Задаване на максимален брой потребители и колко потребителя да се добавят всяка секунда към теста в Locust](#)
- [Фигура 8. Изпълнение на тест и отчитане на брой грешки в Locust](#)
- [Фигура 9. Информация за настъпили грешки по време на тестване в Locust](#)
- [Фигура 10. Лого на Gatling](#)
- [Фигура 11. Конфигурация на протоколи в Gatling](#)
- [Фигура 12. Отчет от тестване в Gatling](#)
- [Фигура 13. Статистика на заявки и отговори в Gatling](#)
- [Фигура 14. Лого на NeoLoad](#)
- [Фигура 15. Приложен програмен интерфейс на NeoLoad](#)
- [Фигура 16. Създаване на облачна сесия в NeoLoad](#)
- [Фигура 17. Валидация на тестови сценарии в NeoLoad](#)
- [Фигура 18. Лого на WebLOAD](#)
- [Фигура 19. Начално табло на WebLOAD](#)
- [Фигура 20. Примерен тестов сценарий в WebLOAD](#)
- [Фигура 21. Отчет в WebLOAD](#)
- [Фигура 22. Лого на RPT](#)
- [Фигура 23. Конфигурация на протокол при използване на REST услуга в RPT](#)
- [Фигура 24. Стартиране на заявка при използване на REST услуга в RPT](#)
- [Фигура 25. Информация за заявка при използване на REST услуга в RPT](#)
- [Фигура 26. Лого на Silk Performer](#)
- [Фигура 27. Потребителски интерфейс на Silk Performer](#)
- [Фигура 28. Стартиране на скрипт в Silk Performer](#)
- [Фигура 29. Тестов резултат в Silk Performer](#)
- [Фигура 30. Лого на LoadRunner Cloud](#)
- [Фигура 31. Начално табло на LoadRunner Cloud](#)
- [Фигура 32. Разпределение на натовареността в LoadRunner Cloud](#)
- [Фигура 33. Резултат от тестване в LoadRunner Cloud](#)
- [Фигура 34. Лого на WAPT](#)
- [Фигура 35. Конфигурация на тест в WAPT](#)
- [Фигура 36. Разпределяне на виртуални потребители в WAPT](#)
- [Фигура 37. Тестов резултат в WAPT](#)