**Инструменти за стрес-тестове на уеб приложения**

1. Съдържание

TBD when finished

1. Основни понятия
   1. Тестване на товар (Load Testing)

Тестването на товар е вид нефункционално тестване и е под-множество 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 товар 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;
* Ограниченията на хардуера;
  1. Стрес тестване (Stress Testing)

Подобно на тестването на товар, стрес тестването също проверява поведението на уеб приложение при определен товар. Това, което ги разграничава е, че докато тестването на товар 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 срине 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;
* Срив 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

1. Apache JMeter

Apache JMeter е инструмент за стрес тестване с отворен код, разработено изцяло на Java. То е създадено за тестване на натовареността на функционалностите и изчисляване на производителност. Приложението позволява тестване на производителността на както статични, така и динамични ресурси. Може да симулира натовареност върху обект, сървър, група от сървъри или мрежа. Така то изследва тяхната издържливост и ефективност при различни степени на натовареност.

Тестване на товар чрез JMeter се извършва по следният 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

Предимства на Apache JMeter:

1. Отворен код

JMeter е приложение с отворен код, което означава, че е напълно безплатен инструмент.

1. Независим откъм платформата

Поради факта, че JMeter е десктоп приложение.

1. Лесна инсталация

Нужно е единствено да се копира и изпълни .bat файла.

1. Ефикасна симулация

Чрез симулиране на различни потребители, които работят на паралелни нишки JMeter може да достигне желаната натовареност.

1. Визуализация от резултати на тестването

Резултатите от тест в JMeter може да представи по четири различни начина: като таблица, лог файл, дърво или диаграма.

1. Поддръжка на протоколи

JMeter поддържа всички базови протоколи: HTTP, SOAP, LDAP, JDBC, JMS, and FTP.

1. Интеграция с библиотеки с отворен код

JMeter е интегрирано с библиотеките Jenkins, Maven, and Gradle.

Недостатъци на Apache JMeter:

1. Висока консумация на памет

JMeter симулациите изискват много памет за изпълнение и генериране на тестов отчет.

1. Единствено за уеб приложения

JMeter е добър инструмент за тестване на уеб приложения, но не е удобен за десктоп приложения.

1. Не поддържа JavaScript

JMeter е десктоп приложение, а не браузърно. Поради това то има трудности при обработката на информация от JavaScript и AJAX, което може да доведе то неточности при симулацията.

1. 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 нападат 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.

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 синхронизационни 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.

Предимства на Locust:

1. Може да се тества всяка система

Инструментът е написан предимно за тестване на уеб приложения, но също позволява тестване и на други типове приложения.

1. Писане на тестове на Python

Тестовете изглеждат и се изпълняват като обикновен код, написан на Python. Това е благодарение на факта, че инструмента прилага coroutine функции, вместо callback.

1. Инструментът е разпределен и скалируем

Locust поддържа стрес тестове, разпределени на няколко машини. А всяка инстанция на Locust позволява хиляди тестови потребители в един процес.

1. Уеб-базиран потребителски интерфейс

Интерфейсът на Locust е разработен с JavaScript и HTML. Той показва важна за тестера информация в реално време. И поради факта, че е базиран на уеб, той е лесно разширяем и може да се приложи в множество платформи.

1. Лесно променлив код

Всичките сложни операции на инструмента се поемат от библиотека gevent. Locust е създаден с идеята да е лесно променлив и има за цел да избегне чупливостта, от която страдат други подобни инструменти за стрес тестване.

Недостатъци на Locust:

1. Загуба на статистики

Инструмента събира и представя статистика докато определен брой потребители не е достигнат. След това статистиката се изтрива и се започва събирането от начало.

1. Липсващи ресурси

Например, инструментът няма да поиска URL адрес без изрично да се зададе да го търси.

1. 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.

1. Дефиниция на заглавна част

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

1. 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.

1. Дефиниция на симулация

Уточнява колко на брой тестови потребителя ще се изпълнят паралелно.

Предимства на Gatling:

1. Поддържа се от всяка операционна система и всеки браузър;
2. Инструментът 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. Инструментът предлага графични отчети с полезна информация за анализ

Недостатъци на Gatling:

1. Инструмента не позволява хоризонтална скалируемост;
2. Инструмента 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.
4. NeoLoad
5. WebLoad
6. Rational Performance Tester
7. Silk Performer
8. StormRunner
9. WAPT
10. Литературни източници