Stress testing with Gatling

Gustavo de Lima

About Me

Gustavo de Lima

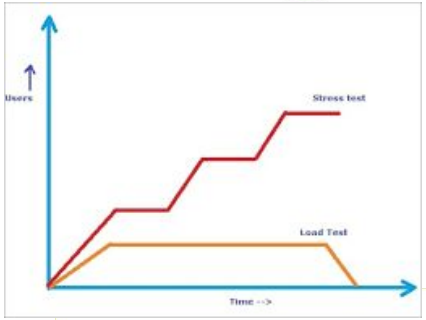
Software Architect

ilegra

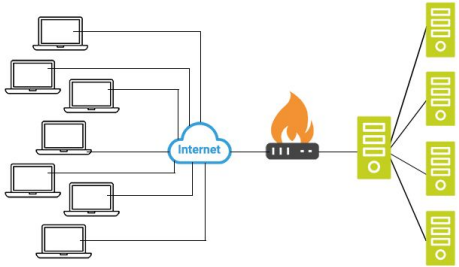
https://github.com/glimsil

https://www.linkedin.com/in/gustavo-de-lima-56a498ba/

Stress Test?



Stress Testing



– Software Testing

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is Defect free.

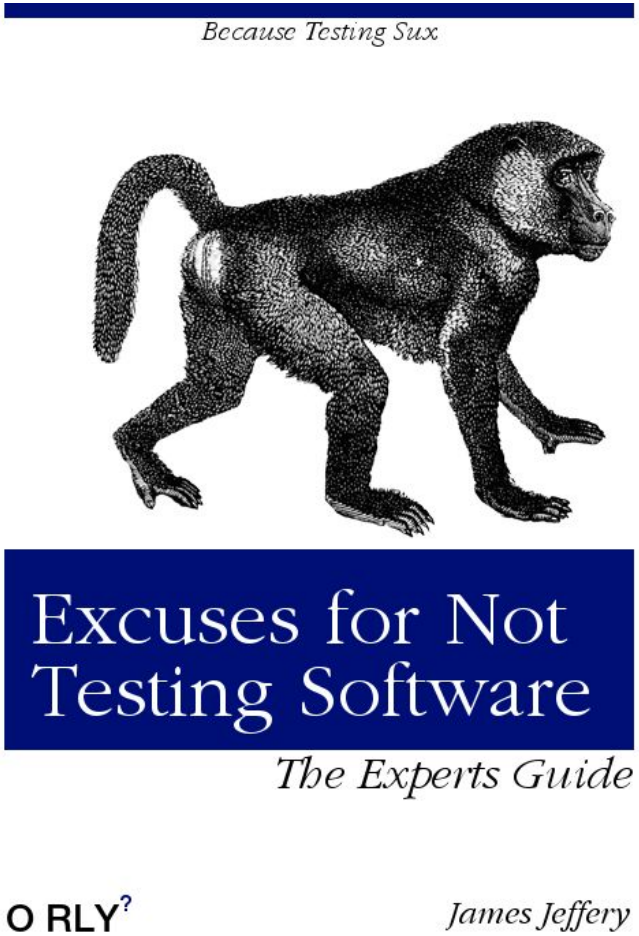
NO WAY

"As a Senior developer I guarantee my own code. I don't need to do tests."

"I have no time to do testing, I have to deliver code. Testing increases the development time!!!"

"My code runs perfectly. Why do I need to test?"

"But, I don't know what to test..."



Why do we need it?

- ➔ Guarantees that our features works.
- ➔ Guarantees that our code has no side effects.
- ➔ Allow safety on refactoring.
- ➔ Preventing bugs is CHEAPER than fixing.
- ➔ Ensures good quality, an agile/lean foundation.

– Types of Testing

PERFORMANCE & STRESS TESTING

WHY??

How can we guarantee both quality and efficiency?

WHY??

How can we guarantee both quality and efficiency?

PERFORMANCE & STRESS TESTING FUNDAMENTALS

- Ensures that the product being tested fits for purpose (is the product performant enough).

- Understand your system and its limitations.

WHY PERFORMANCE TESTING are so hard?

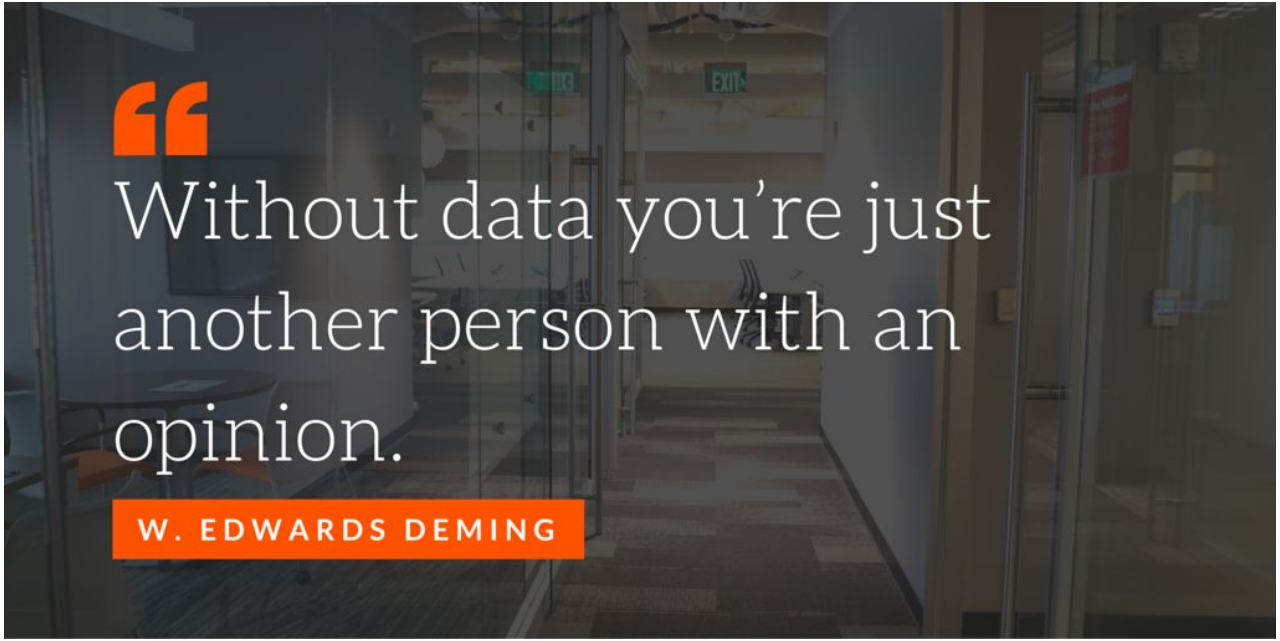
Simulates production as close as possible:

- Hardware
  - CPU, RAM, Storage,...
- Software
  - OS, Virtualization, DB,..
- Isolation

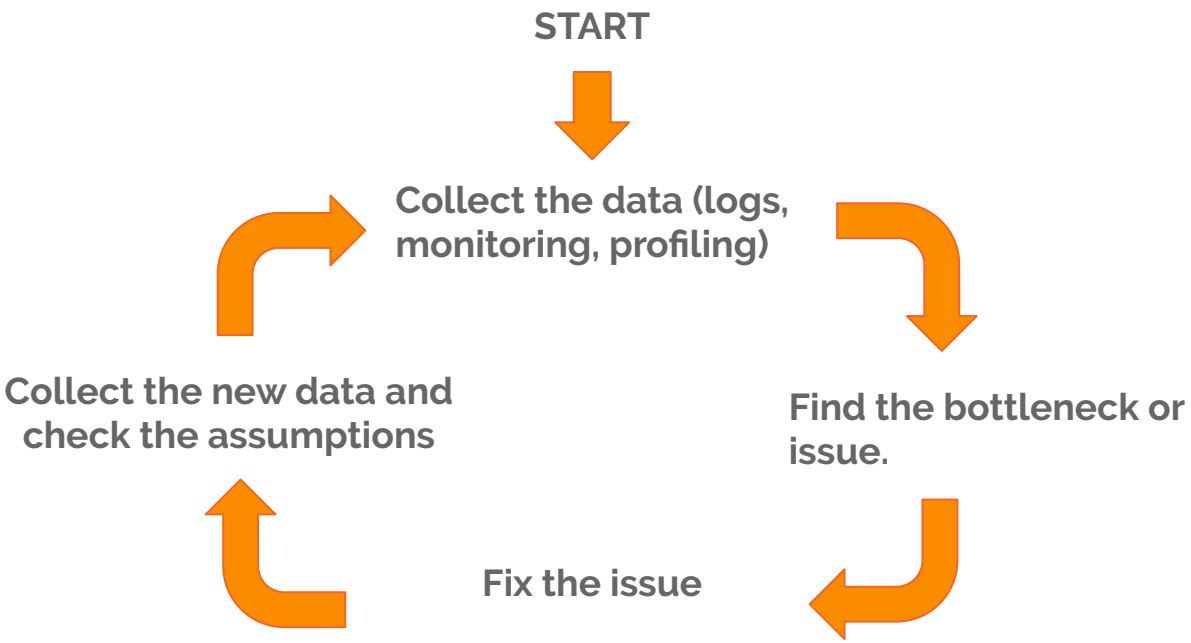
You need to have:

- Infrastructure
- Monitoring
- Logs

YOUR INTUITION IS POSSIBLY WRONG



What we need to do?



Stressing the application

We need to test the upper limits of our application, over chaotic situation (extreme loads, thousands simultaneous accesses, lots of data, ...)

Metrics:

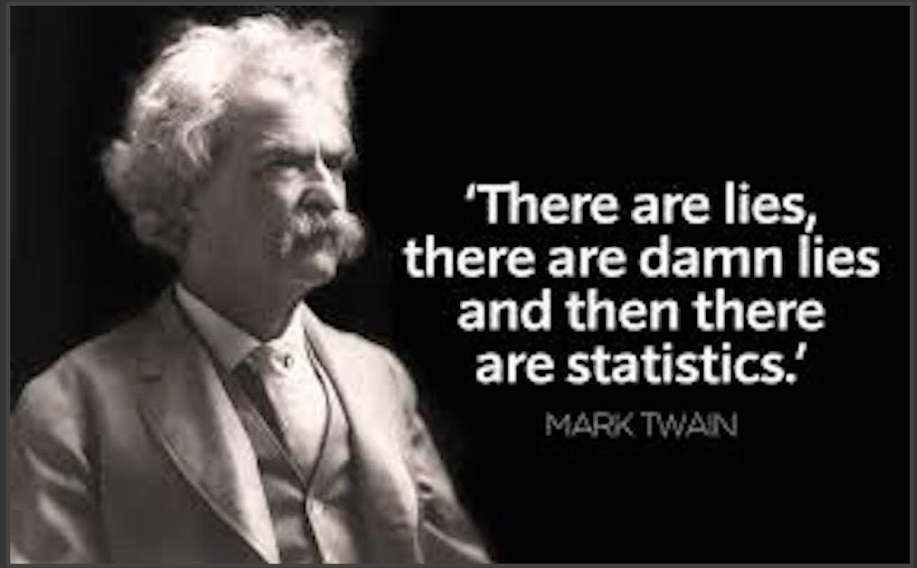
- Response per second, throughput.
- Hit time, response time, etc
- Failures

Planning Tests

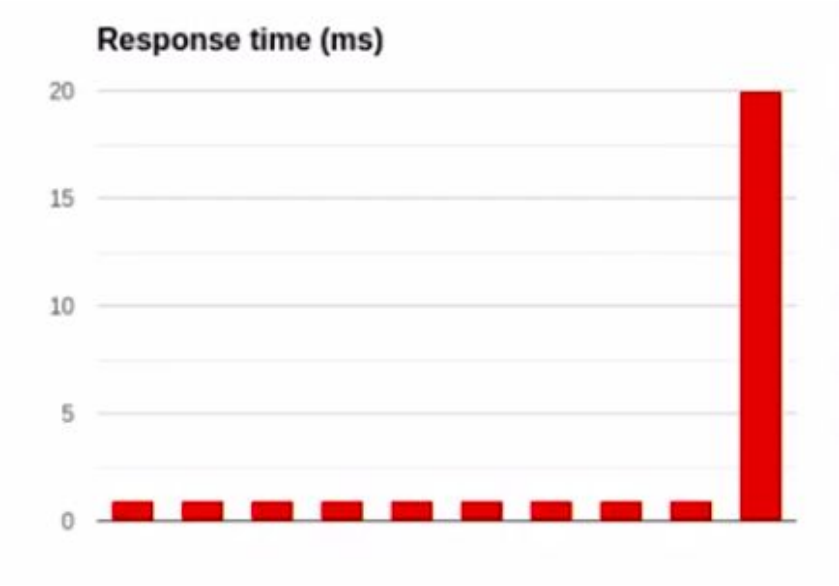
Solid planning up front are essential to have a great testing. Be sure your test plan meets the following objectives:

- ➔ Testing objectives.
- ➔ Test Scope.
- ➔ Acceptance Criteria.
- ➔ Test Approach.
- ➔ Entry and Exit Criteria.
- ➔ Risks, issues, assumptions, dependencies

– Measuring results



Measuring results

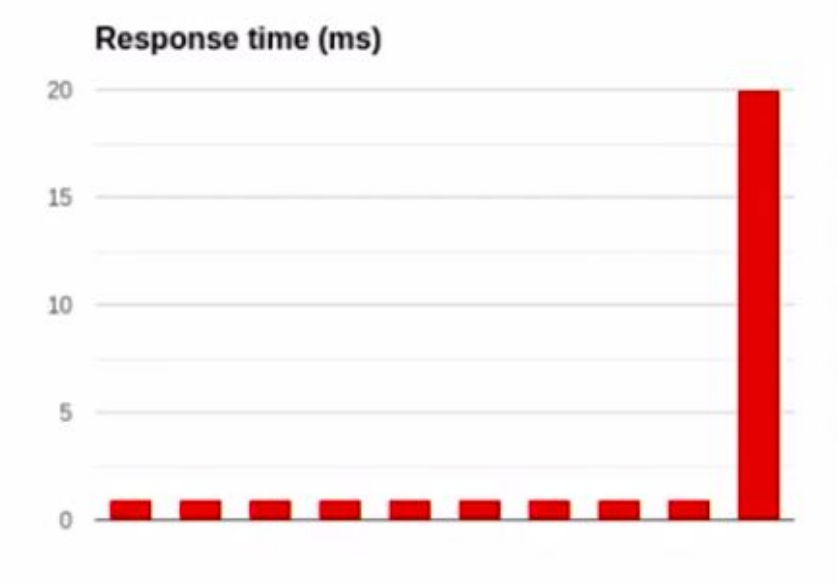


Arithmetic mean = 2.9ms

Median = 1ms

Standard deviation = 6ms

Measuring results



~~Arithmetic mean = 2.9ms~~

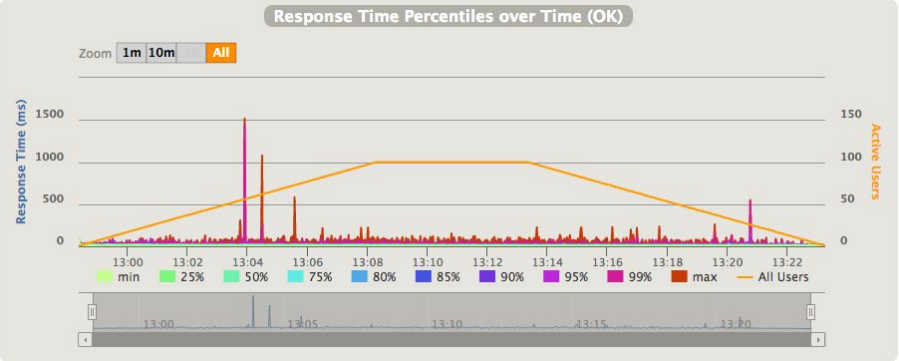
~~Median = 1ms~~

~~Standard deviation = 6ms~~

USE Percentiles!!!

- 20th = 1ms
- 40th = 12ms
- 60th = 22ms
- 80th = 35ms
- 90th = 35.9ms

Measuring results



Coordinated Omission Problem

USE Percentiles?? How To
Lie With Percentiles

Gatling

– Gatling.io

- Open source
- Scala
- Non blocking / async stack (scala, akka, netty)
- DSL
- Recorder
- Reports

– Gatling.io

Open source

gatling / gatling

Watch 237 | Star 4.5k | Fork 949

Code | Issues 36 | Pull requests 1 | Actions | Security | Insights

Async Scala-Akka-Netty based Load Test Tool https://gatling.io

8,314 commits | 10 branches | 0 packages | 60 releases | 143 contributors | Apache-2.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

slandelle Normalize consecutive slashes, close #3826 ··· Latest commit 854650e 6 days ago

gatling-app/src/main	...	2 months ago
gatling-benchmarks	drop ahc package	8 months ago
gatling-bundle/src	Remove -XX:+AggressiveOpts from JVM options, close #3807	last month
gatling-charts/src	nit: explicit return types	last month
gatling-commons/src	ByteBufs#byteBufsToByteArray doesn't honor ByteBuf#readerIndex, close #…	21 days ago

– Gatling.io

Scala

```scala
import java.io.File
import scala.io.Source
import scala.collection.mutable.Map

class SpellCorrector {
  var wordCounts : Map[String, Int] = Map()
  val alphabets = ('a' to 'z').toSet

  def train(trainFile : File) = {
    val lines = Source.fromFile(trainFile) mkString
    val wordREPattern = "[A-Za-z]+"
    wordREPattern.r.findAllIn(lines).foreach( txtWord => {
      val word = txtWord.toLowerCase
      if (wordCounts.keySet contains(word)) {
        wordCounts(word) = wordCounts(word)+1
      } else {
        wordCounts += (word -> 1)
      }
    })
  }

  def getSplittedCombinations(word : String) : Set[(String, String)
    (0 to word.length).map( idx => (word.substring(0, idx), word.su

  def getEditOneSpellings(word: String) : Set[String] = {
    val splits = getSplittedCombinations(word)
    val deletes = splits.map( s => if (s._2.length>0) {s._1+s._2.su
```

Scala

– Gatling.io

Non blocking / async stack (scala, akka, netty)

Netty

– Gatling.io

DSL

```
scenario("DSL demo")
    .exec(http("go to page")
        .get("/computers")
        .check(regex("computers")
          .find(1)
          .exists)
```

Scala

– Gatling.io

Recorder

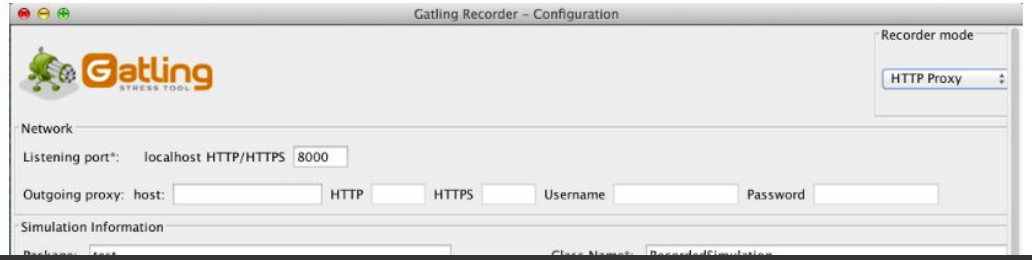
Gatling

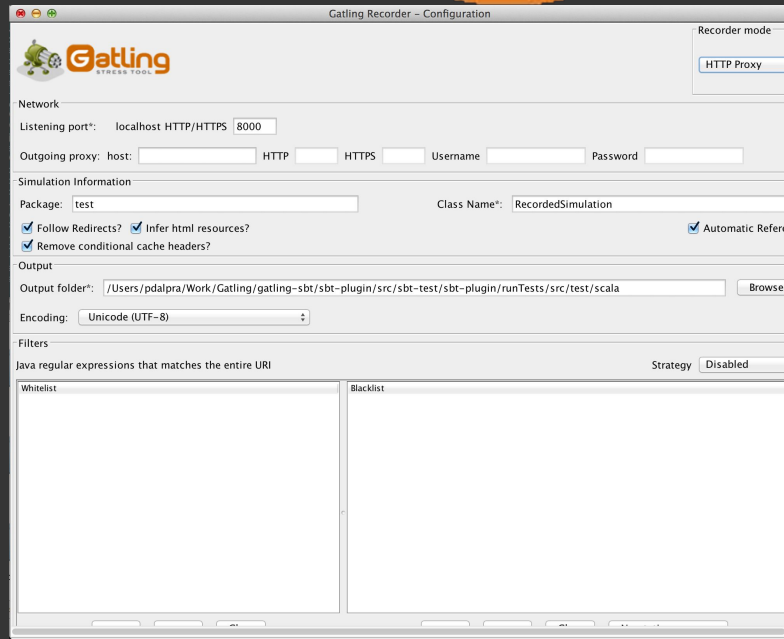
Gatling documentation / 2.2 / HTTP / **Recorder**

RECORDER

The Gatling Recorder helps you to quickly generate scenarios, by either acting as a HTTP proxy between the browser and the HTTP server or converting HAR (Http ARchive) files. Either way, the Recorder generates a simple simulation that mimics your recorded navigation.

If you're using the bundle, you can launch it with the following script `$GATLING_HOME/bin/recorder.sh`. You will get a window that looks like this one:





– Gatling.io

Reports


```
---- RecordedSimulation ------------------------------------------------------------
[##########################################################################]100%
          waiting: 0      / active: 0      / done: 10
================================================================================

Simulation MySimulation completed in 19 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
---- Global Information ------------------------------------------------------------
> request count                                          90 (OK=90     KO=0     )
> min response time                                     247 (OK=247    KO=-     )
> max response time                                     673 (OK=673    KO=-     )
> mean response time                                    347 (OK=347    KO=-     )
> std deviation                                         132 (OK=132    KO=-     )
> response time 50th percentile                         270 (OK=270    KO=-     )
> response time 75th percentile                         506 (OK=506    KO=-     )
> response time 95th percentile                         582 (OK=582    KO=-     )
> response time 99th percentile                         615 (OK=615    KO=-     )
> mean requests/sec                                     4.5 (OK=4.5    KO=-     )
---- Response Time Distribution ----------------------------------------------------
> t < 800 ms                                            90 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

– Gatling.io

Plugins and extensions

- Official
  - SBT
  - Maven
  - Jenkins
  - JMS
- Third Party
  - Gradle
  - Kafka
  - Cassandra
  - JDBC
  - ...

First Simulation - DSL

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration._

class MySimulation extends Simulation {

val httpConfig = http
    .baseUrl("https://jsonplaceholder.typicode.com/posts/1") //A REST API to call
    .acceptHeader("application/json")

val scn = scenario("Test Scenario") // A scenario is a chain of requests and pauses
    .exec(http("request_1")
      .get("/"))

setUp(scn.inject(atOnceUsers(1))
    .protocols(httpConfig))

}
```

DSL

```
val httpProtocol = http
    .baseUrl("http://computer-database.gatling.io")
    .inferHtmlResources(BlackList("""".*\.css""", """.*\.js""", """.*\.ico""""), WhiteList())
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8")
    .acceptEncodingHeader("gzip, deflate")
    .acceptLanguageHeader("it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3")
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; WOW64; rv:46.0) Gecko/20100101 Firefox/46.0")

val scn = scenario("RecordedSimulation")
    .exec(http("request_0")
        .get("/"))
    .pause(5)
    .exec(http("request_1")
        .get("/computers?f=amstrad"))
    .pause(4)
    .exec(http("request_2")
        .get("/computers/412"))
    .pause(2)
    .exec(http("request_3")
        .get("/"))
    .pause(2)
    .exec(http("request_4")
        .get("/computers?p=1"))
    .pause(1)
    .exec(http("request_5")
        .get("/computers?p=2"))
    .pause(2)
    .exec(http("request_6")
        .get("/computers?p=3"))

setUp(scn.inject(atOnceUsers(1))).protocols(httpProtocol)
```

DSL - Structure

```
val httpProtocol = http
    .baseUrl("http://computer-database.gatling.io")
    .inferHtmlResources(BlackList("""".*\.css""", """.*\.js""", """.*\.ico""""), WhiteList())
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8")
    .acceptEncodingHeader("gzip, deflate")
    .acceptLanguageHeader("it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3")
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; WOW64; rv:46.0) Gecko/20100101 Firefox/46.0")

val scn = scenario("RecordedSimulation")
    .exec(http("request_0")
        .get("/"))
    .pause(5)
    .exec(http("request_1")
        .get("/computers?f=amstrad"))
    .pause(4)
    .exec(http("request_2")
        .get("/computers/412"))
    .pause(2)
    .exec(http("request_3")
        .get("/"))
    .pause(2)
    .exec(http("request_4")
        .get("/computers?p=1"))
    .pause(1)
    .exec(http("request_5")
        .get("/computers?p=2"))
    .pause(2)
    .exec(http("request_6")
        .get("/computers?p=3"))

setUp(scn.inject(atOnceUsers(1))).protocols(httpProtocol)
```

Protocol definition

Test Scenario

Behavior Injection

DSL - Injection

```
setUp(scn.inject(
    nothingFor(5 seconds),
    atOnceUsers(1000),
    rampUsers(50) during (10 seconds),
    constantUsersPerSec(50) during (20 seconds),
    rampUsersPerSec(10) to 75 during (2 minutes) randomized
)).protocols(httpProtocol)
```

DSL - Injection

```
setUp(myScenario
      .inject(atOnceUsers(10))
      .protocols(httpConf))
    .throttle(
      reachRps(100) in (30 second),
      holdFor(1 minute),
      jumpToRps(50),
      holdFor(2 hours)
    )
```

Feeders

Feeder is a type alias for `Iterator[Map[String, T]]`, meaning that the component created by the feed method will poll `Map[String, T]` records and inject its content.

```
feed(feeder)
```

This defines a workflow step where every virtual user feed on the same Feeder.
Every time a virtual user reaches this step, it will pop a record out of the Feeder, which will be injected into the user's Session, resulting in a new Session instance.

If the Feeder can't produce enough records, Gatling will complain about it and your simulation will stop.

Feeders - Strategies

Gatling provides multiple strategies for the built-in feeders:

```
.queue // default behavior: use an Iterator on the underlying sequence
.random // randomly pick an entry in the sequence
.shuffle // shuffle entries, then behave like queue
.circular // go back to the top of the sequence once the end is reached
```

Feeders - CSV

```
val csvFeeder = csv("foo.csv") // use a comma separator
val tsvFeeder = tsv("foo.tsv") // use a tabulation separator
val ssvFeeder = ssv("foo.ssv") // use a semicolon separator
val customSeparatorFeeder = separatedValues("foo.txt", '#') // use your own separator
```

Feeders - CSV - Loading Mode

CSV files feeders provide several options for how data should be loaded in memory.

```
val csvFeeder = csv("foo.csv").eager.random
val csvFeeder = csv("foo.csv").batch.random
val csvFeeder2 = csv("foo.csv").batch(200).random // tune internal buffer size
```

Default behavior is an adaptive policy based on (unzipped, sharded) file size, see `gatling.core.feederAdaptiveLoadModeThreshold` in config file. Gatling will use `eager` below threshold and batch above.

Feeders - CSV

```
object Search {

  val feeder = csv("search.csv").random // 1, 2

  val search = exec(http("Home")
    .get("/"))
    .pause(1)
    .feed(feeder) // 3
    .exec(http("Search")
      .get("/computers?f=${searchCriterion}") // 4
      .check(css("a:contains('${searchComputerName}')", "href").saveAs("computerURL"))) // 5
    .pause(1)
    .exec(http("Select")
      .get("${computerURL}")) // 6
    .pause(1)
}
```

Feeders - Redis

This feature was originally contributed by Krishnen Chedambarum.

Gatling can read data from Redis using one of the following Redis commands.

- **LPOP** - remove and return the first element of the list
- **SPOP** - remove and return a random element from the set
- **SRANDMEMBER** - return a random element from the set

Feeders - Redis

By default RedisFeeder uses LPOP command:

```
import com.redis._
import io.gatling.redis.feeder.RedisFeeder

val redisPool = new RedisClientPool("localhost", 6379)

// use a list, so there's one single value per record, which is here named "foo"
val feeder = RedisFeeder(redisPool, "foo")
```

An optional third parameter is used to specify desired Redis command:

```
// read data using SPOP command from a set named "foo"
val feeder = RedisFeeder(clientPool, "foo", RedisFeeder.SPOP)
```

Feeders - JDBC

Gatling also provide a builtin that reads from a JDBC connection.

```
// beware: you need to import the jdbc module

import io.gatling.jdbc.Predef._

jdbcFeeder("databaseUrl", "username", "password", "SELECT * FROM users")
```

Just like File parser built-ins, this return a `RecordSeqFeederBuilder` instance.

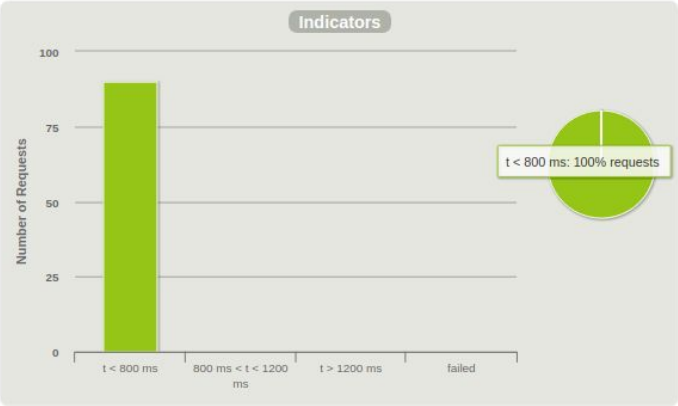
- The databaseURL must be a JDBC URL (e.g. `jdbc:postgresql:gatling`),
- the username and password are the credentials to access the database,
- sql is the query that will get the values needed.

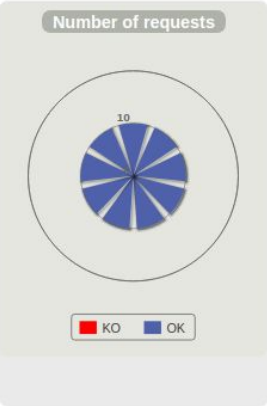
Output

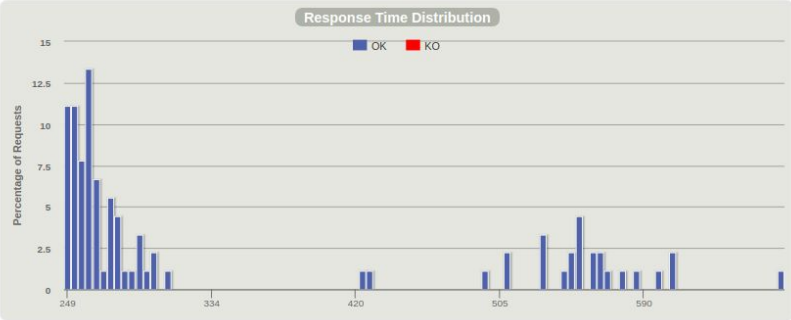
```
---- RecordedSimulation ------------------------------------------------------
[#########################################################################]100%
          waiting: 0      / active: 0      / done: 10
================================================================================

Simulation MySimulation completed in 19 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
---- Global Information --------------------------------------------------------
> request count                                          90 (OK=90     KO=0     )
> min response time                                     247 (OK=247    KO=-     )
> max response time                                     673 (OK=673    KO=-     )
> mean response time                                    347 (OK=347    KO=-     )
> std deviation                                         132 (OK=132    KO=-     )
> response time 50th percentile                         270 (OK=270    KO=-     )
> response time 75th percentile                         506 (OK=506    KO=-     )
> response time 95th percentile                         582 (OK=582    KO=-     )
> response time 99th percentile                         615 (OK=615    KO=-     )
> mean requests/sec                                     4.5 (OK=4.5    KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                             90 (100%)
> 800 ms < t < 1200 ms                                    0 (  0%)
> t > 1200 ms                                             0 (  0%)
> failed                                                  0 (  0%)
================================================================================
```

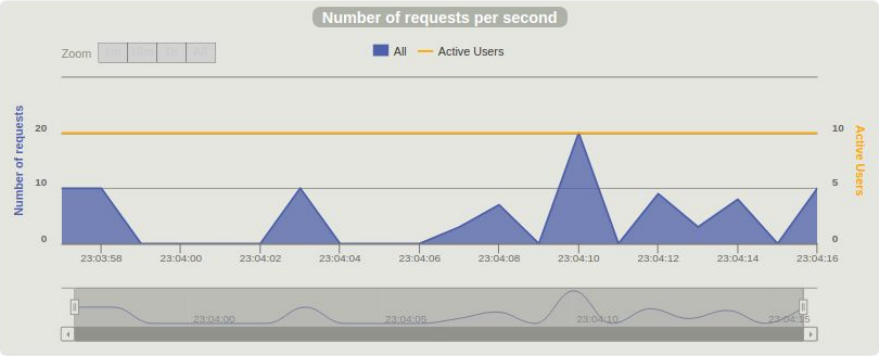
Output

> Global Information





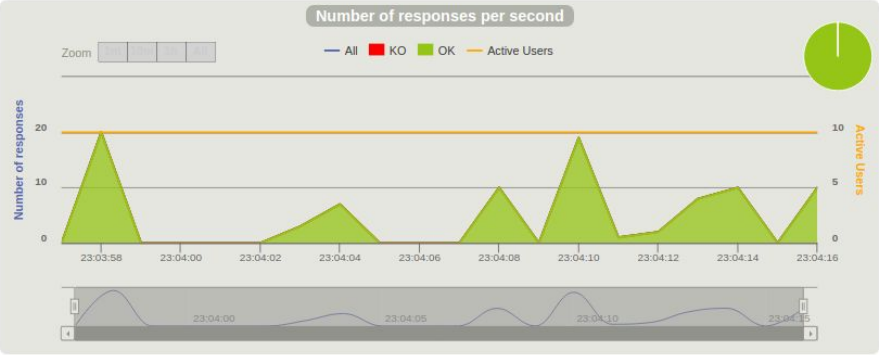




STATISTICS

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	90	90	0	0%	4.5	247	270	506	582	615	673	347	132
request_0	10	10	0	0%	0.5	545	560	567	582	585	586	561	13
request_...direct 1	10	10	0	0%	0.5	254	301	396	562	651	673	353	122
request_1	10	10	0	0%	0.5	498	531	552	560	564	565	533	21
request_2	10	10	0	0%	0.5	248	257	259	262	263	263	256	5
request_3	10	10	0	0%	0.5	250	271	286	291	291	291	270	16
request_...direct 1	10	10	0	0%	0.5	247	263	268	279	280	280	263	10
request_4	10	10	0	0%	0.5	250	261	264	271	274	275	260	7
request_5	10	10	0	0%	0.5	265	276	518	607	608	608	372	152
request_6	10	10	0	0%	0.5	250	255	263	275	282	284	259	10



Exercises

- Groups of 3 people
- Github: https://github.com/glimsil/stress-testing-gatling-training

What we are dealing with

We have a simple user service, used by a blog. The service is simple but bad implemented and with weird patterns and dont have the best tech stack. There are lot of features missing, so we will need to implement some.

What do we need todo

1. Understand the system and it limitations.(code, running tests, etc)
2. Gather information with discovery team about expected performance.
3. Analyse data and find bottlenecks and issues.
4. Fix it
5. Analyse the new data and compare the results.
6. We will add some features and stress them.

Stress testing with Gatling

Gustavo de Lima