Torturing Wildflies with Gatling Stress Tool

Table of Contents

Introduction

This document will describe how to perform **performance** and **load** tests using **Gatling** stress tool.

Objectives

- Setup Gatling in our CDI Crud application;
- Creating the simulation;
  - Stressing the REST API;
  - Stressing the web application;
- Continuous performance testing
- Torturing wildfly application server through load test

A note on performance x load

Before reading this post it has to be clear the difference between both kind of tests:

- **Performance test**: This kind of test aims to constantly check/monitor the system's performance. The test must fail when the system performance (under normal peak) is not considered good or is below expectations (eg: response time or throughput got worse when compared to a predefined limit).

- **Load test**: is meant to test the system by constantly and steadily increasing the load on the system until it reaches a threshold limit. The goal of load testing is to expose defects (eg: memory leaks) when application is under abnormal situation.

Why Gatling?

The main advantage of **Gatling**, in my opinion, is that it is intuitive. When you write performance tests with gatling API you're really describing user steps so it is easy to **understand**, **write** and **maintain** the test.

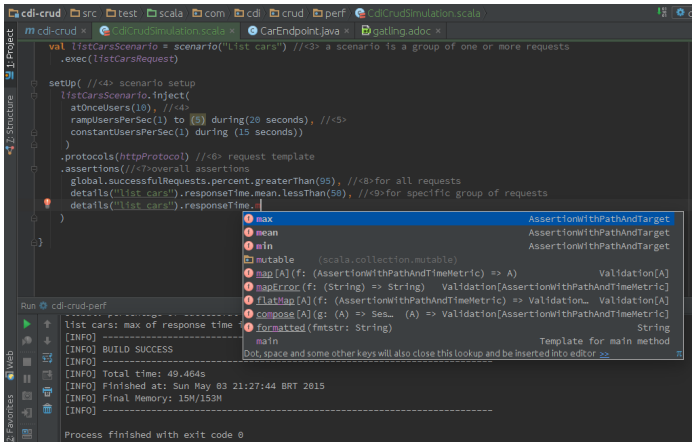
For example, when using JMeter you hardly will write a simulation without the help of its GUI to record the test.

TIP	XML is not a programming language ;)

Another advantage is that gatling is based on actors model, netty and NIO. So what that means?

- it is asynchronous
- non blocking I/O
- each user runs on its thread
- probably perform better then other stress tools
- produces more reliable simulation (it's more approximated to real application behaviour - mainly due to its threading model)

Also IDE support is a big advantage



Setup Gatling in our project

To configure the tool on our project we basically will use the gatling maven plugin as follows:

pom.xml

```
<plugin>
<groupId>io.gatling</groupId>
<artifactId>gatling-maven-plugin</artifactId>
<version>2.1.5</version>
<configuration>
<dataFolder>src/test/resources/data</dataFolder>
</configuration>
</plugin>
```

① External data (samples, user logins etc...) to be used in the simulation.

Also include the following maven dependency:

pom.xml

```
<dependency>
<groupId>io.gatling.highcharts</groupId>
<artifactId>gatling-charts-highcharts</artifactId>
<version>2.1.5</version>
<scope>test</scope>
</dependency>
```

Creating the simulation

Simulation is the name (usually) given to performance tests because this kind of test try to simulate the application's usage under real circumstances or even abnormal (in case of load tests) situations like e.g putting/simulating a lot of users using the app at the same time.

A Gatling simulation is basically a scala file using an specific DSL which will performe calls (usually http) to application under test.

A simple example

Here is a simple simulation which makes http call to our application REST API:

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

class CdiCrudSimulation extends Simulation {


  val httpProtocol = http <1>
    .baseURL("http://localhost:8080/cdi-crud/")
    .acceptHeader("application/json;charset=utf-8")
    .contentTypeHeader("application/json; charset=UTF-8")

  val listCarsRequest = http("list cars") <2>
    .get("rest/cars/")
    .check(status.is(200)) <3>

  val listCarsScenario = scenario("List cars") //<4> A scenario is a group of one or more
requests
    .exec(listCarsRequest)

  setUp( //<4> scenario setup
      listCarsScenario.inject(
        atOnceUsers(10),  <4>
        rampUsersPerSec(1) to (10) during(20 seconds),  <5>
        constantUsersPerSec(2) during (15 seconds))
       )
      .protocols(httpProtocol)  <6>
      .assertions( <7>
        global.successfulRequests.percent.greaterThan(95), // for all requests
        details("list cars").responseTime.mean.lessThan(50), // for specific group of
requests
        details("list cars").responseTime.max.lessThan(300)
      )

}
```

① Template for all http requests;

② Stores this request in a local variable;

③ Request assertion;

④ Add 5 users at the same time (each on its on thread). They will fire one request (wait its response) each one.

⑤ scale from 1 to 10 users during 20 seconds (on the last second the 10 users will fire requests simultaneously)

⑥ 2 users per second during 15 seconds (i fell quite dummy explaining this because the DSL is really

self explanatory)

⑦ this section makes assertions on all or a group of requests

> **NOTE** I’ve already talked about this REST API here

This simulation fires a total of 150 request, here is the output in the console:

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                        150 (OK=150    KO=0     )
> min response time                                      8 (OK=8      KO=-     )
> max response time                                     38 (OK=38     KO=-     )
> mean response time                                    21 (OK=21     KO=-     )
> std deviation                                          5 (OK=5      KO=-     )
> response time 50th percentile                         22 (OK=22     KO=-     )
> response time 75th percentile                         24 (OK=24     KO=-     )
> mean requests/sec                                  4.343 (OK=4.343  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                           150 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================

Reports generated in 0s.
Please open the following file: /home/pestano/projects/cdi-
crud/target/gatling/results/cdicrudsimulation-1430707109729/index.html
Global: percentage of successful requests is greater than 95 : true
list cars: mean of response time is less than 50 : true
list cars: max of response time is less than 300 : true
```

And here are some graphical report:

References

1. http://pt.slideshare.net/swapnilvkotwal/gatling