

Gatling is a truly awesome open source tool for performance and stress testing that is well worth adding to your personal tool repository.

what exactly is the Gatling stress testing tool ?

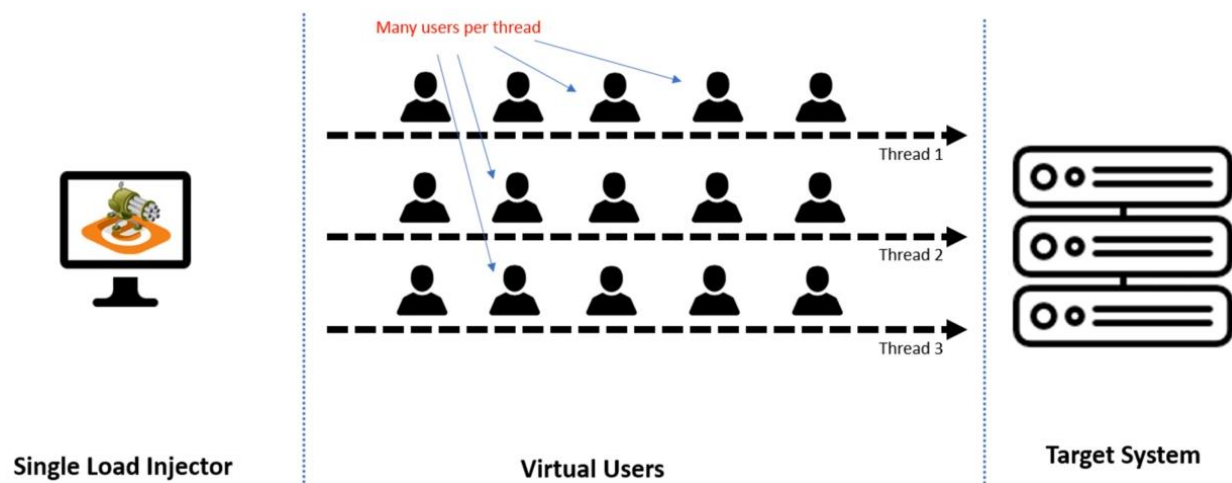
- Gatling is designed to treat your performance tests as production code
- Maintenance and automation are made easier Gatling integrates with your development tools integrated. (Scala code.)
 - o development environment version
 - o control systems build tools
 - o continuous integration solutions.

It build on software of akka, which is tool kit for building giggly concurrent distributed and resilient message driven applications for java and scholar

Akka is distributed framework based on the Actor model allowing for fully asynchronous computing.

Akka are small entities within code communicating with each other through messaging using this mode Gatling can simulate multiple virtual users with a single thread, because akka override the Jvm limitation of handling many thread.

Gatling is bale to run thousands of virtual users on a single machine something that Jmeter and most other commercial tools are not (these gives Gatling a huge advantage over these tools)



Key Features of Gatling

- 1- Excellent to support for HTTP Protocol (This makes getting a great choice for low testing , Any hate HTTP server or for calling API directly.
- 2- Scala based- but with a very expressive DSL (if you have some knowledge of Scala and you can make your tests even more flexible and robust as you have access to all the desired Scala and Java code)
- 3- Gui Recorder included for recording User journey. (convert it into a Gatling scenario.)
- 4- Detailed load test Reports (run as part of continuous integration for a tool such as Jenkins as)

When Gatling be a good tool to use for your project

- Stress / Load Test your application (Catching is a great tool when you just want to stress test your system without worrying about where the traffic originated from)
- Simulate 1000`s of users on a single instance
- Work great in a CL / CD environment or pipeline (it a great tool to add as part of your delivery pipeline or continuous integration as the performance test can be run whenever you create a new build of your project)
- Learn about writing code & write more expressive tests (is also great for when you want to learn more about performance testing and writing test code)
- Improve your skills as a software engineer

Installing Gatling

- Open terminal
- Gatling.bat
- After install check the JDK java is installed and the environment variable with sets.



```
GATLING_HOME is set to "c:\Gatling\gatling-charts-highcharts-bundle-2.3.1"
JAVA = "C:\Program Files\Java\jdk1.8.0_151\bin\java.exe"
```

We can see that getting ships with a few example simulations press 0 to choose the basic simulation.

Run mysimid

```
Select simulation id (default is 'basicsimulation'). Accepted characters are a-z, A-Z, 0-9, - and _
mysimid
```

Gatling one to execute the basic simulation

```
Select run description (optional)
run 1
```

Run 1

(it makes a series of request to an example website hosted by Gatling)

```
---- Requests -----
> Global                                (OK=3      KO=0      )
> request_1                            (OK=1      KO=0      )
> request_1 Redirect 1                  (OK=1      KO=0      )
> request_2                             (OK=1      KO=0      )

---- Scenario Name -----
[-----] 0%
      waiting: 0      / active: 1      / done:0
=====

2018-08-01 13:42:50                                15s elapsed
---- Requests -----
> Global                                (OK=6      KO=0      )
> request_1                            (OK=1      KO=0      )
> request_1 Redirect 1                  (OK=1      KO=0      )
> request_2                             (OK=1      KO=0      )
> request_3                             (OK=1      KO=0      )
> request_4                             (OK=1      KO=0      )
> request_4 Redirect 1                  (OK=1      KO=0      )

---- Scenario Name -----
[-----] 0%
      waiting: 0      / active: 1      / done:0
=====
```

Gatling Recorder

There are two main ways to create performance test script in Gatling either from

- 1- Writing up the code manually
- 2- Or using Gatling recorder

(Gatling recorder useful for getting a quick basic script in place and the recorder is particularly useful when you `re first starting out.)

Command line

- Recorder.bat

Gatling Recorder - Configuration

Recorder mode: HTTP Proxy

Network
 Listening port*: localhost HTTP/HTTPS 8000 HTTPS mode: Self-signed Certificate
 Outgoing proxy: host: HTTP HTTPS Username Password

Simulation Information
 Package: Class Name*: RecordedSimulation
☒ Follow Redirects? ☒ Infer html resources? ☒ Automatic Referers?
☒ Remove cache headers? ☐ Save & check response bodies?

Output
 Output folder*: c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\user-files\simulations 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!

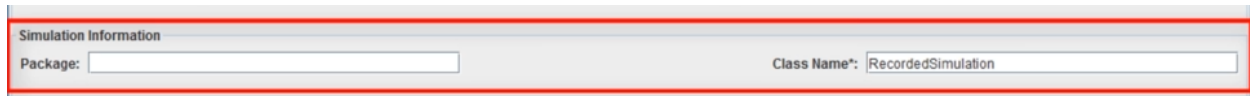
There are two recorder model

- HTTP proxy (which capture all traffic from your browser)
- HAR Converter (Which convert HTTP to ocv file)

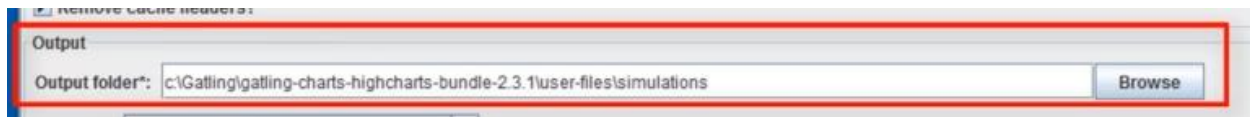
(if you choose HAR converter we simple need to provide a a half hearted recorder)



There are a few other options we can change here such as the package and the class name for our script.



We can also configure whether to follow redirects infer HTML resource

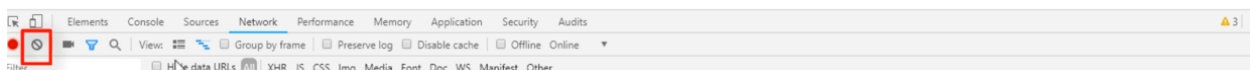


This is output folder where our scripts will be saved again.

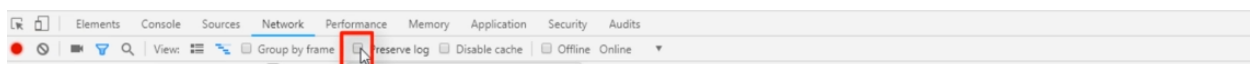
How to create HTTP archive file in google chrome and how to use to create Gatling script.

- Open google chrome and enabled the developer tools.
- Click the network tap (the site we are going to record off first scripts against sky news.com, the captured is loaded now and we load the page what we will do now is reload the page and capture the traffic again.

First, click a button to clear all the previous traffic captured



Then click the preserve lock button and make sure the recording is red

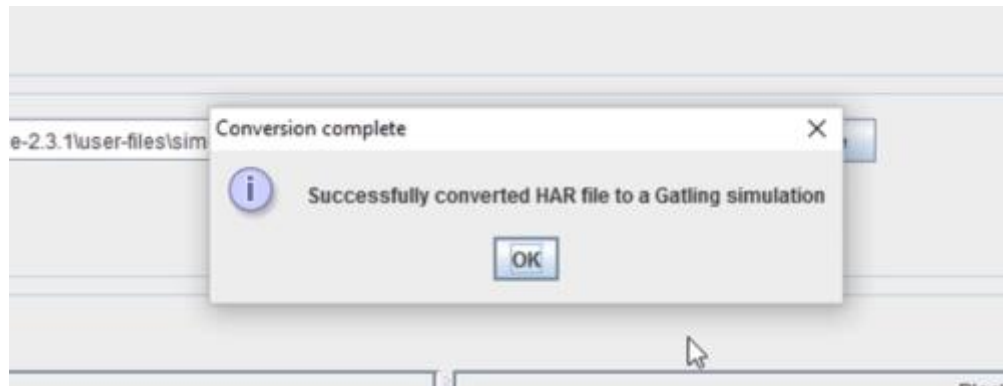


Now refresh the home page to load it again and capture all the traffic

Finally, right click within developer tools and choose save with HAR content.

Open the Gatling record

- Launch the recorder again on command line
- Change record mode to HAR converter and browse the file and click start



- Open and browse to gatling installation folder

```
cmd

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1
λ ls
LICENSE  bin/  conf/  lib/  results/  target/  user-files/

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1
λ cd user-files\

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\user-files
λ ls
bodies/  data/  simulations/

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\user-files
λ cd simulations\
```

- Open the file that we just created called a recorded simulation at Scala which supplied

```
c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\user-files\simulations
λ subl RecordedSimulation.scala |
```

This is our simulation file for loading the home page and code

```

RecordedSimulation.scala x
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class RecordedSimulation extends Simulation {

  val httpProtocol = http
    .baseUrl("https://px.moatads.com")
    .inferHtmlResources()

  val headers_0 = Map(
    "accept" -> "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "accept-encoding" -> "gzip, deflate, br",
    "accept-language" -> "en-GB,en-US;q=0.9,en;q=0.8",
    "cache-control" -> "max-age=0",
    "cookie" -> "_cb_lis=1; _cb=DwzIq9DA7BLMCU91GB; optimizelyEndUserId=oeu15294088801216r0.775631137068107; __gads=ID=9006faf2fb0ee6d2:T=1529408766:S=
    "upgrade-insecure-requests" -> "1",
    "user-agent" -> "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36")

  val headers_5 = Map("User-Agent" -> "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36")

  val headers_32 = Map(
    "Accept" -> "image/webp,image/apng,image/*,*/*;q=0.8",
    "Accept-Encoding" -> "gzip, deflate, br",
    "Accept-Language" -> "en-GB,en-US;q=0.9,en;q=0.8",
    "Connection" -> "keep-alive",
    "User-Agent" -> "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36")

  val headers_39 = Map(
    "Origin" -> "https://news.sky.com",
    "User-Agent" -> "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36")
}

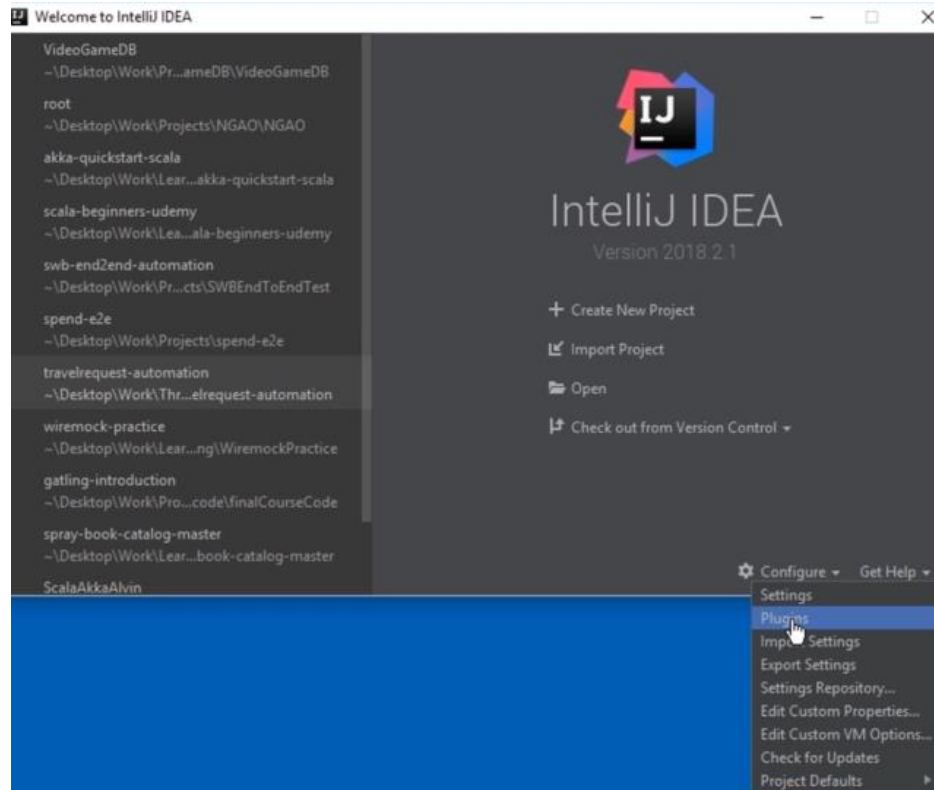
```

Dev Environment pre-requisites

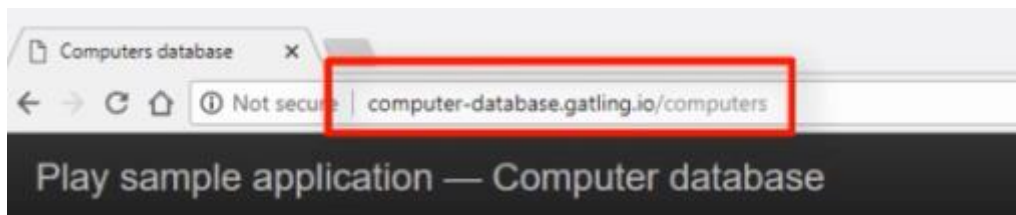
- 1- Java 8 JDK
- 2- IntelliJ
- 3- Scala SDK



- 4- Once the installer has finished check that scala was installed by cmd scala
- 5- Install intellij scala plugin (click configure)



Run website



- 1- Refresh the page and to capture the traffic and then click add a new computer we will fill in details a click create this computer

Play sample application — Computer database

Add a computer

Computer name Required

Introduced date Date ('yyyy-MM-dd')

Discontinued date Date ('yyyy-MM-dd')

Company ▼

[Create this computer](#) or [Cancel](#)

Then we will search for the computer that was created.

Play sample application

2,189 computers found

Done! Computer abc123 has been created

[Filter](#)

Computer name

And click the computer to view status

One computer found

[Filter by name](#) [Add a new computer](#)

Computer name	Introduced	Discontinued	Company
abc123	01 Jan 2018	01 Jan 2019	Apple Inc.

← Previous Displaying 1 to 1 of 1 Next →

We start to recorder

On the top script

```
import scala.concurrent.duration._

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class GatlingDatabase extends Simulation {

  val httpProtocol = http
    .baseUrl("http://computer-database.gatling.io")
    .inferHtmlResources()

  val headers_0 = Map(
    "Accept" -> "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "Accept-Encoding" -> "gzip, deflate",
    "Accept-Language" -> "en-GB,en-US;q=0.9,en;q=0.8",
    "Cache-Control" -> "max-age=0",
    "Connection" -> "keep-alive",
    "Upgrade-Insecure-Requests" -> "1",
    "User-Agent" -> "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36")

  val headers_3 = Map(
    "Accept" -> "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "Accept-Encoding" -> "gzip, deflate",
    "Accept-Language" -> "en-GB,en-US;q=0.9,en;q=0.8",
    "Connection" -> "keep-alive",
    "Upgrade-Insecure-Requests" -> "1",
    "User-Agent" -> "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36")

  val headers_6 = Map(
    "Accept" -> "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "Accept-Encoding" -> "gzip, deflate",
    "Accept-Language" -> "en-GB,en-US;q=0.9,en;q=0.8",
    "Cache-Control" -> "max-age=0",
    "Connection" -> "keep-alive",
    "Origin" -> "http://computer-database.gatling.io",
    "Upgrade-Insecure-Requests" -> "1",
    "User-Agent" -> "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36")
}
```

was captured

The down main scenario

```
val scn = scenario("GatlingDatabase")
  .exec(http("request_0")
    .get("/computers")
    .headers(headers_0)
    .resources(http("request_1")
      .get("/assets/stylesheets/bootstrap.min.css"),
      http("request_2")
        .get("/assets/stylesheets/main.css")))
    .pause(3)
  .exec(http("request_3")
    .get("/computers/new")
    .headers(headers_3)
    .resources(http("request_4")
      .get("/assets/stylesheets/bootstrap.min.css"),
      http("request_5")
        .get("/assets/stylesheets/main.css")))
    .pause(12)
  .exec(http("request_6")
    .post("/computers")
    .headers(headers_6)
    .formParam("name", "abc123")
    .formParam("introduced", "2018-01-01")
    .formParam("discontinued", "2019-01-01")
    .formParam("company", "1")
    .resources(http("request_7")
      .get("/assets/stylesheets/bootstrap.min.css"),
      http("request_8")
        .get("/assets/stylesheets/main.css")))
    .pause(6)
  .exec(http("request_9")
    .get("/computers?f=abc123")
    .headers(headers_3)
    .resources(http("request_10")
      .get("/assets/stylesheets/bootstrap.min.css"),
      http("request_11")
        .get("/assets/stylesheets/main.css")))
    .pause(2)
```

Submit the computer database

```
.exec(http("request_6")
  .post("/computers")
  .headers(headers_6)
  .formParam("name", "abc123")
  .formParam("introduced", "2018-01-01")
  .formParam("discontinued", "2019-01-01")
  .formParam("company", "1")
  .resources(http("request_7")
    .get("/assets/stylesheets/bootstrap.min.css"),
    http("request_8")
    .get("/assets/stylesheets/main.css"))))
```

Then search the computer

```
.pause(6)
.exec(http("request_9")
  .get("/computers?f=abc123")
  .headers(headers_3)
  .resources(http("request_10")
    .get("/assets/stylesheets/bootstrap.min.css"),
    http("request_11")
    .get("/assets/stylesheets/main.css"))))
```

Load the details the computer

```
.exec(http("request_12")
  .get("/computers/2353")
  .headers(headers_3))
```

Right at the bottom of the script is where the load profile is created

It run for single user

Run script and load home page

```
val scn = scenario("GatlingDatabase")
  .exec(http("LOAD_HOME_PAGE")
    .get("/computers")
    .headers(headers_0)
    .resources(http("request_1")
      .get("/assets/stylesheets/bootstrap.min.css"),
      http("request_2")
        .get("/assets/stylesheets/main.css"))))
```

Change to new computer page

```
.exec(http("LOAD_NEW_COMPUTER_PAGE")
  .get("/computers/new")
  .headers(headers_3)
  .resources(http("request_4")
    .get("/assets/stylesheets/bootstrap.min.css"),
    http("request_5")
      .get("/assets/stylesheets/main.css"))))
```

To create new computer

```
.exec(http("CREATE_NEW_COMPUTER")
  .post("/computers")
  .headers(headers_6)
  .formParam("name", "abc123")
  .formParam("introduced", "2018-01-01")
  .formParam("discontinued", "2019-01-01")
  .formParam("company", "1")
  .resources(http("request_7")
    .get("/assets/stylesheets/bootstrap.min.css")
    http("request_8"))
```


To search for computer

```
.exec(http("SEARCH_COMPUTER")
      .get("/computers?f=abc123")
      .headers(headers_3)
      .resources(http("request_10")
                  .get("/assets/stylesheets/bootstrap.min.css"),
                  http("request_11")
                    .get("/assets/stylesheets/main.css"))))
save(2)
```

To view computer page

Change pause time to 1 sec and save the script

On command line and run and press 0 which is created

```
c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\bin
λ gatling.bat
GATLING_HOME is set to "c:\Gatling\gatling-charts-highcharts-bundle-2.3.1"
JAVA = ""C:\Program Files\Java\jdk1.8.0_151\bin\java.exe""
|
```

The result

```
=====
---- Global Information -----
> request count                10 (OK=10    KO=0    )
> min response time            21 (OK=21    KO=-    )
> max response time            106 (OK=106   KO=-    )
> mean response time           50 (OK=50    KO=-    )
> std deviation                 30 (OK=30    KO=-    )
> response time 50th percentile 42 (OK=42    KO=-    )
> response time 75th percentile 72 (OK=72    KO=-    )
> response time 95th percentile 97 (OK=97    KO=-    )
> response time 99th percentile 104 (OK=104   KO=-    )
> mean requests/sec            2 (OK=2     KO=-    )
---- Response Time Distribution -----
> t < 800 ms                   10 (100%)
> 800 ms < t < 1200 ms        0 ( 0%)
> t > 1200 ms                  0 ( 0%)
> failed                       0 ( 0%)
=====
```

Go onto the computer database and search for computer

Let's take a look at the result report

```
c:\Gatling\gatling-charts-highcharts-bundle-2.3.1
λ ls
LICENSE  bin/  conf/  lib/  results/  target/  user-files/

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1
λ cd results\

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\results
λ ls
gatlingdatabase-1534201123744/  mysimid-1533127355487/

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\results
λ cd gatlingdatabase-1534201123744\

c:\Gatling\gatling-charts-highcharts-bundle-2.3.1\results\gatlingdatabase-1534201123744
λ ls
index.html          req_create-new-comp-bbe39.html  req_request-1-46da4.html  simulation.log
js/                 req_load-homepage-d662d.html   req_request-2-93baf.html  style/
req_bootstrap-min-c-5b8a7.html  req_load-new-comput-483b0.html  req_search-computer-64460.html
req_create-new-comp-2f48e.html  req_main-css-4abed.html        req_view-computer-p-c9306.html
```

Write

Index.html (test result report)

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	10	10	0	0%	2	21	42	72	97	104	106	50	30
LOAD_HOMEPAGE	1	1	0	0%	0.2	86	86	86	86	86	86	86	0
bootstrap.min.css	1	1	0	0%	0.2	60	60	60	60	60	60	60	0
main.css	1	1	0	0%	0.2	60	60	60	60	60	60	60	0
request_1	1	1	0	0%	0.2	106	106	106	106	106	106	106	0
request_2	1	1	0	0%	0.2	76	76	76	76	76	76	76	0
LOAD_NEW...TER_PAGE	1	1	0	0%	0.2	23	23	23	23	23	23	23	0
CREATE_NEW_COMPUTER	1	1	0	0%	0.2	23	23	23	23	23	23	23	0
CREATE_N...direct 1	1	1	0	0%	0.2	22	22	22	22	22	22	22	0
SEARCH_COMPUTER	1	1	0	0%	0.2	21	21	21	21	21	21	21	0
VIEW_COMPUTER_PAGE	1	1	0	0%	0.2	23	23	23	23	23	23	23	0

Some transaction recorded

bootstrap.min.css	1	1	0	0%	0.2	60	60	60	60	60	60	60	0
main.css	1	1	0	0%	0.2	60	60	60	60	60	60	60	0
request_1	1	1	0	0%	0.2	106	106	106	106	106	106	106	0
request_2	1	1	0	0%	0.2	76	76	76	76	76	76	76	0

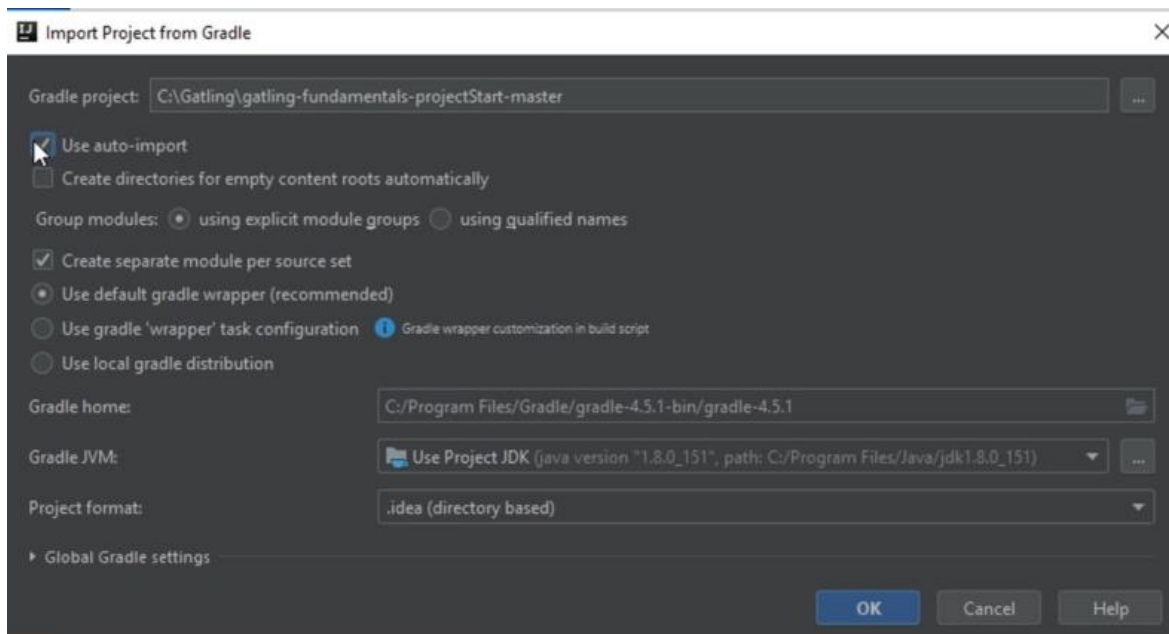
Pre-request the course

- Installation of the video game database
- Installation of Gatling Dev environment pre-req
- Create configure project in intellij
- Write our first Gatling script
- Creating a Gatling Runner class
- Create a base class

Template project



Open intellij and import and select build.gradle and make sign of check box



Create a script is simply is going to call the GET all games end-points of our video game db inside the source

Library

```
import io.gatling.core.Predef._  
import io.gatling.http.Predef._
```

First

```
import io.gatling.core.Predef._  
import io.gatling.http.Predef._  
  
class MyFirstTest extends Simulation {  
  // 1 Common HTTP Configuration  
  // 2 Scenario Definition  
  // 3 Load Scenario  
}
```

E.g. BaseURL,
Common HTTP Headers

Second

```
// 2 Scenario Definition  
// 3 Load Scenario  
}
```

User Journey Steps, i.e:

- Go to Homepage
- Call this API
- Go to this page

Three

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._

class MyFirstTest extends Simulation {

  // 1 Common HTTP Configuration

  // 2 Scenario Definition

  // 3 Load Scenario

}
```

How many users to run?
How long to run for?

that will be executed

```
class MyFirstTest extends Simulation {

  // 1 Common HTTP Configuration
  val httpConf = http
    .baseUrl(url = "http://localhost:8080/app/")
    .header(name = "Accept", value = "application/json")

  // 2 Scenario Definition

  // 3 Load Scenario

}
```

User Journey = 1 call to
`http://localhost:8080/app/
videogames`

```
// 2 Scenario Definition
val scn = scenario(scenarioName = "My First Test")
  .exec(http(requestName = "Get All Games")
    .get("videogames"))
// 3 Load Scenario
}
```

GET
http://localhost:8080/app/
videogames

We assign GET call to the video game end-point

```
// 3 Load Scenario
setUp(
  scn.inject(atOnceUsers(users = 1))
).protocols(httpConf)
}
```

The above is telling our load scenario to run a single user and to execute the scenario called

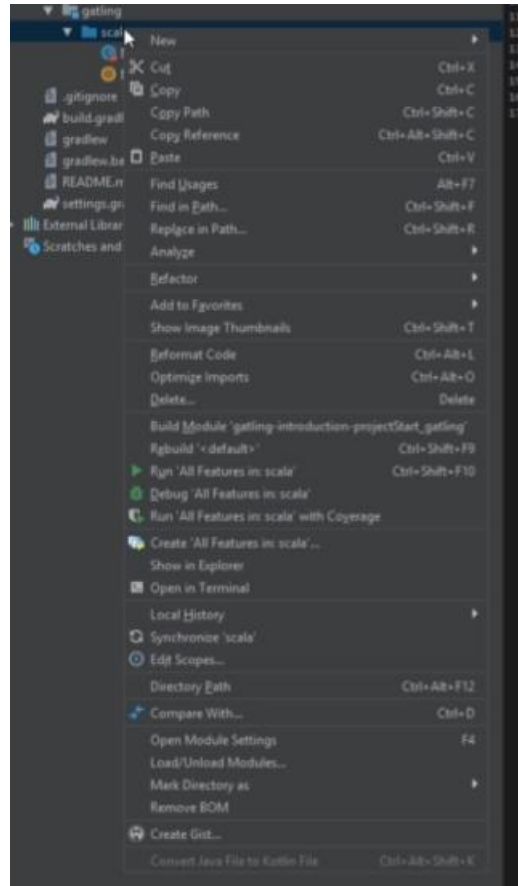
To run script open folder Mygatlingrunner

```
import io.gatling.app.Gatling
import io.gatling.core.config.GatlingPropertiesBuilder

object MyGatlingRunner {
  def main(args: Array[String]): Unit = {
    val simClass = classOf[MyFirstTest].getName
    val props = new GatlingPropertiesBuilder
    props.simulationClass(simClass)
    Gatling.fromMap(props.build)
  }
}
```

A good practice to implement now therefore is the creation of a base class that all tests can extend.

This will allow us to at code that can be reused in all our of our tests.



Create a new package get a name a baseConfig after created right click and choose new class.

Its an optional step but I do find it helpful with debugging as I am on windows.

The proxy to say i`m going to use Fiddler

.proxy ()

This will capture all the traffic on port 8888 when the proxy is running again.

```

package baseConfig

import io.gatling.core.Predef._
import io.gatling.http.Predef._

class BaseSimulation extends Simulation {

  val httpConf = http
    .baseUrl( url = "http://localhost:8080/app/")
    .header( name = "Accept", value = "application/json")
    .proxy(Proxy("localhost", 8888).httpsPort( port = 8888))

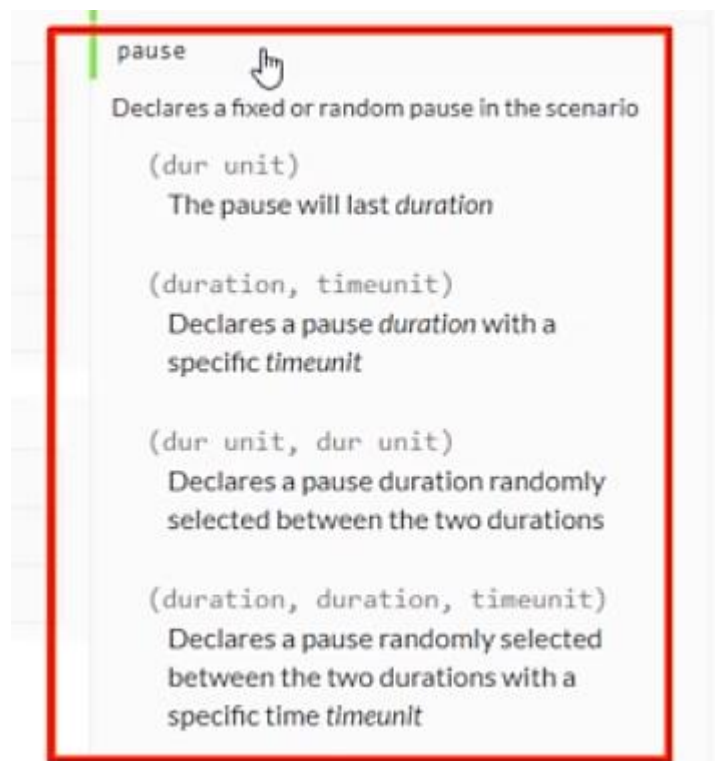
}

```

Task

- 1- GET all request
- 2- GET specific request
- 3- Get videogame API

First, create a new package and create class with AddPauseTime



1. HTTP Configuration

2. Scenario Definition

3. Load Scenario

Make GET request

`.pause(1,20)` => this is add random pause of between 1 and 20 sec to finish the script

Scenario

```
package simulations

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration.DurationInt
import baseConfig.BaseSimulation

class AddPauseTime extends BaseSimulation{

  val scn = scenario( scenarioName = "Video Game DB")

  .exec(http( requestName = "Get All Video Games - 1st call")
    .get("videogames"))
  .pause( duration = 5)

  .exec(http( requestName = "Get specific game")
    .get("videogames/1"))
  .pause(1, 20)

  .exec(http( requestName = "Get All Video Games - 2nd call")
    .get("videogames"))
  .pause(3000.milliseconds)

}
```

2. Scenario Definition

Load one user

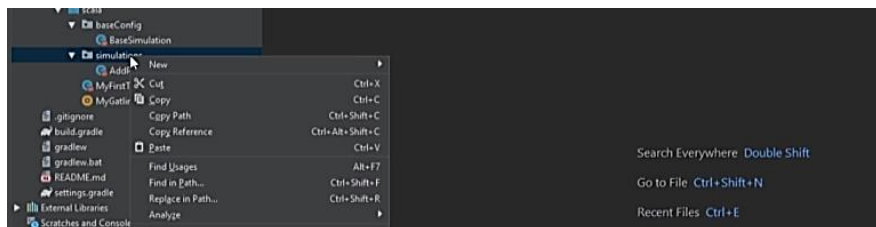
```
setUp(
  scn.inject(atOnceUsers( users = 1))
).protocols(httpConf)
```

What is the above code do is taking the scenario inject a single virtual user at once

Check the HTTP response code we get back

- Check the specific code
- Check code in range
- Check code is 'NOT'

Right on simulation to create a new class (Check ResponseCode)



Snap

```
package simulations

import baseConfig.BaseSimulation
import io.gatling.core.Predef._
import io.gatling.http.Predef._

class CheckResponseCode extends BaseSimulation {

  val scn = scenario( scenarioName = "Video Game DB")

  .exec(http( requestName = "Get All Video Games - 1st call")
    .get("videogames")
    .check(status.is( expected = 200)))

  .exec(http( requestName = "Get specific game")
    .get("videogames/1")
    .check(status.in( expected = 200 to 210)))

  .exec(http( requestName = "Get All Video Games - 2nd call")
    .get("videogames")
    .check(status.not( expected = 404), status.not( expected = 500)))

  setUp(
    scn.inject(atOnceUsers(1))
  ).protocols(httpConf)
}
```


Task

Check for the response body back from GET id = 1 is

- To make check write simple JSON PATH query

Inputs

Output paths

ONPath Syntax

\$.name

ample '\$.phoneNumbers[*]' type See also JSONPath expressions

ON

```

1 id
2 "id": 1,
3 "name": "Resident Evil 4",
4 "releaseDate": "2005-10-01",
5 "reviewScore": 85,
6 "category": "Shooter",
7 "rating": "Universal"
8 }

```

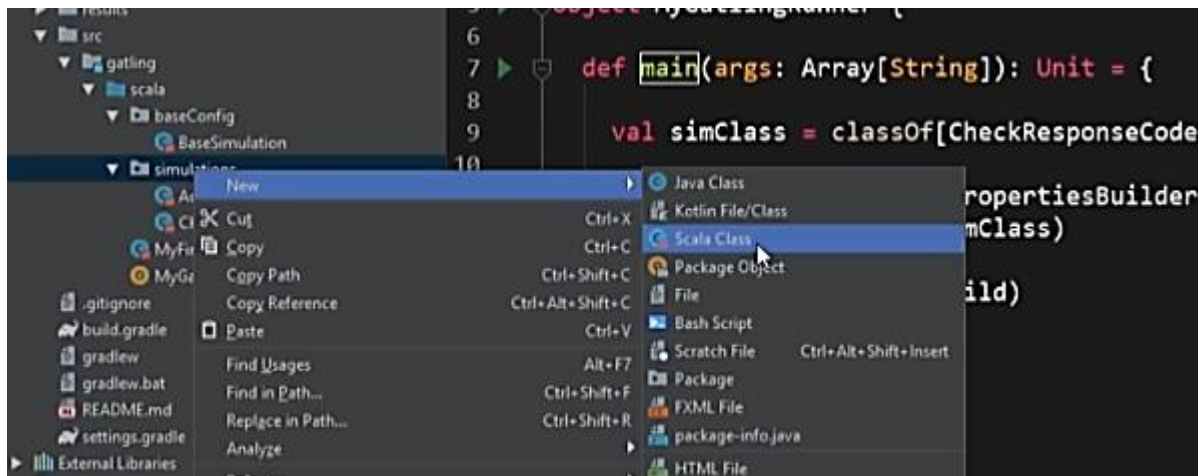
Evaluation Results

```

1 {
2   "Resident Evil 4"
3 }

```

Create a class on simulation



CheckResponseBodyAndExtract

```

Val scn = scenario (scenarioName = 'Video game ')
.exec(http(requestName="Get specific game ")
.get ("videogame/1")
.check(jsonPath("$.name").is (expected = "Resident Evil 4")))

```

```

import io.gatling.core.Predef._
import io.gatling.http.Predef._

class CheckResponseBodyAndExtract extends BaseSimulation {

  val scn = scenario( scenarioName = "Video Game DB")

  .exec(http( requestName = "Get specific game")
    .get("videogames/1")
    .check(jsonPath( path = "$.name").is( expected = "Resident Evil 4"))))

  setUp(
    scn.inject(atOnceUsers( users = 1))
  ).protocols(httpConf)
}

```

Task

Make

- GET video game and using JSON Path to extract response id of second game
- GET specific game , we use extract value as parameter in our call

Using JSON XPath

Inputs

☐ Output paths

JSONPath Syntax

I

Example '\$.phoneNumbers[0].type' See also JSONPath expressions

JSON

```

1- [
2- {
3-   "id": 1,
4-   "name": "Resident Evil 4",
5-   "releaseDate": "2005-10-01",
6-   "reviewScore": 85,
7-   "category": "Shooter",
8-   "rating": "Universal"
9- },
10- {
11-   "id": 2,
12-   "name": "Gran Turismo 3",
13-   "releaseDate": "2001-03-10",
14-   "reviewScore": 91,
15-   "category": "Driving",
16-   "rating": "Universal"
17- },
18- {
19-   "id": 3,
20-   "name": "Tetris",
21-   "releaseDate": "1984-06-25",
22-   "reviewScore": 86,
23-   "category": "Puzzle",
24-   "rating": "Universal"
25- }
26- ]

```

Evaluation Results

```

1- id
2- 2
3- ]

```

Look at the data and save the value of 2 in gameId as parameter

```
.exec(http( requestName = "Get All Video Games - 2nd call")
  .get("videogames")
  .check(jsonPath( path = "$[1].id").saveAs( key = "gameId"))))
```

Using variable and match the with game name

```
.exec(http( requestName = "Get specific game - 2nd call with parameter")
  .get("videogames/${gameId}")
  .check(jsonPath( path = "$.name").is( expected = "Gran Turismo 3")))
```

Result

```
----- Response Time Distribution -----
> t < 800 ms                                2 ( 67%)
> 800 ms < t < 1200 ms                      1 ( 33%)
> t > 1200 ms                               0 (  0%)
> failed                                    0 (  0%)
=====
```

(save data in parameter and use that data in a subsequent request this is commonly referred to as **correlation in performance testing**)

How we can capture the entire response body and view that for debugging purposes

```
.exec(http( requestName = "Get All Video Games - 2nd call")
  .get("videogames")
  .check(jsonPath( path = "$[1].id").saveAs( key = "gameId")))
.exec { session => println(session); session }
```

Console output

```
Simulation: Simulation#CheckResponseBodyAndExtract started...
Session(Video Game DB,1,Map(gatling.http.cache.dns -> io.gatling.http.resolver.ShuffleJdkNameResolver@2ce4925b, gameId -> 2),1534975081422,102,OK,List(),io.gatling.core.protocol.ProtocolComponen
```

Let`s now look at how we can capture and display the entire http response code when debugging, we adding code to the last API call in our script

```
.exec(http( requestName = "Get specific game - 2nd call with parameter")
  .get("videogames/${gameId}")
  .check(jsonPath( path = "$.name").is( expected = "Gran Turismo 3"))
  .check(bodyString.saveAs( key = "responseBody")))
  .exec { session => println(session("responseBody").as[String]); session }
```

Console

```
Simulation simulations.CheckResponseBodyAndExtract started...
Session(Video Game DB 1.Man(gatling http cache dns -> io.gatling.http.resolver.ShuffleIdkNameResolver@584f523f gameId -> 2),153497
{"id":2,"name":"Gran Turismo 3","releaseDate":"2001-03-10","reviewScore":91,"category":"Driving","rating":"Universal"}
```

Task

Good abstract the information for each call into its own method or object.

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._

class CodeReuseWithObjects extends BaseSimulation {

  val scn = scenario( scenarioName = "Video Game DB")

  .exec(http( requestName = "Get All Video Games")
    .get("videogames")
    .check(status.is( expected = 200)))

  .exec(http( requestName = "Get specific game")
    .get("videogames/1")
    .check(status.in( expected = 200 to 210)))

  .exec(http( requestName = "Get All Video Games")
    .get("videogames")
    .check(status.is( expected = 200)))

  def getAllVideoGames(): ChainBuilder = {
    exec(http( requestName = "Get All Video Games")
      .get("videogames")
      .check(status.is( expected = 200)))
  }

  def getSpecificVideoGames(): ChainBuilder = {
    exec(http( requestName = "Get specific game")
      .get("videogames/1")
      .check(status.in( expected = 200 to 210)))
  }

  val scn = scenario( scenarioName = "Video Game DB")
  .exec(getAllVideoGames())
}
```

Good structure

```
def getAllVideoGames() : ChainBuilder = {
  exec(http( requestName = "Get All Video Games")
    .get("videogames")
    .check(status.is( expected = 200)))
}

def getSpecificVideoGames() : ChainBuilder = {
  exec(http( requestName = "Get specific game")
    .get("videogames/1")
    .check(status.in( expected = 200 to 210)))
}

val scn = scenario( scenarioName = "Video Game DB")
  .exec(getAllVideoGames())
  .pause( duration = 5)
  .exec(getSpecificVideoGames())
  .pause( duration = 5)
  .exec(getAllVideoGames())

setUp(
  scn.inject(atOnceUsers( users = 1))
).protocols(httpConf)
```

We can make those method calls loop.

How to can make a method called loop a set number of times.

Gatling cheat

Loops	
forever	Gr
repeat	pi
Repeats a part of the scenario	
(times){chain}	De
The chain is repeated the specified number of times	
(times, counterName){chain}	
The chain is repeated the specified number of times with a forced counter name	

Make snip of code

```
def getAllVideoGames() : ChainBuilder = {
  repeat( times = 3 ) {
    exec(http( requestName = "Get All Video Games")
      .get("videogames")
      .check(status.is( expected = 200)))
  }
}
```

We can see in report

```
2018-08-22 23:15:36 11s elapsed
---- Requests -----
> Global (OK=11 KO=0 )
> Get All Video Games (OK=6 KO=0 )
> Get specific game (OK=5 KO=0 )
Video Game DB
```

Cover section 5

- CSV Feeders
- Basic custom Feeders
- Complex custom feeders [filters used both in a row string and ask groups as well as for template file.]

When you come to running Gatling performance test groups for many users you will typically want different to input different data.

Like

videoGameId = 1

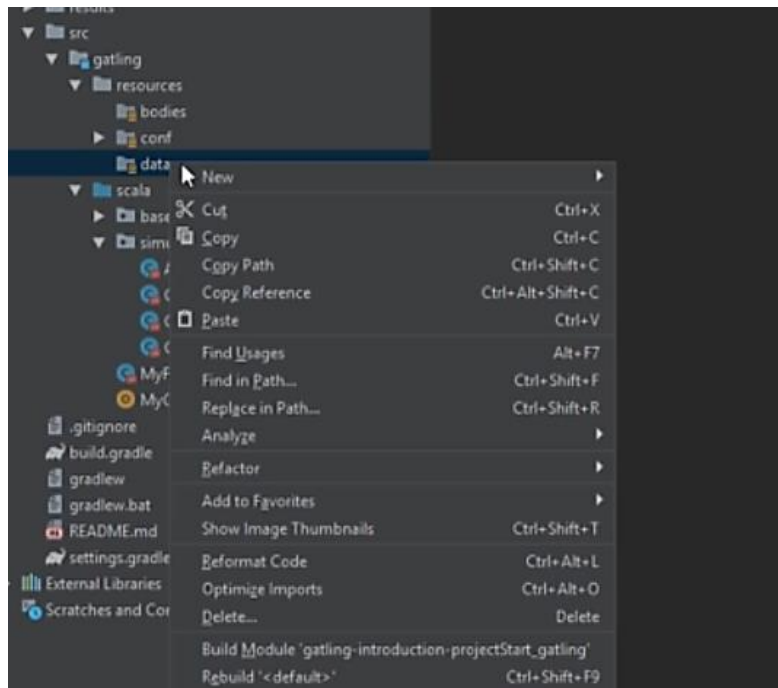
videoGameId = 2

videoGameId = 3

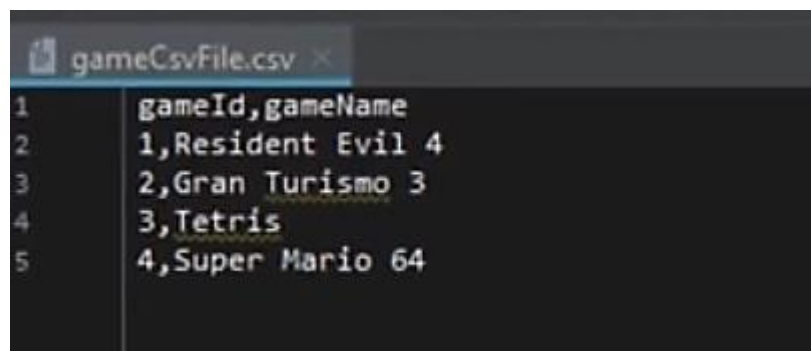
1- One is for use of CSV Feeders which we will look at => Comma Separated value

(it is simple file that is used to hold data typically used for parameter isolation)

- Add csv file in project



New file = gameCSVFile.csv



Right click on simulation and create a class Scala class

- 1- First declare CSV feeder

Feeder definition

Inject data in your scenario

Feeder types

csv

Declares a feeder from a CSV file

(fileName)

fileName is the name of the file under the *data* folder containing the comma separated values to be injected

We have four strategies

Feeder strategies

queue

This is the default strategy (you don't have to specify it). It takes the values in the feeder as in a queue. If the queue is not long enough, you'll get an error at execution.

shuffle

This strategy shuffles the values in the feeder but then works like the *queue* strategy.

random

This strategy chooses values randomly in the feeder. In contrast to a *shuffle* strategy the same values can be chosen several times.

circular

Using Circular in example

```
val csvFeeder = csv( fileName = "gameCsvFile.csv").circular
```

Task

We want to call the same point GET specific game 10 times using our CSV feeder

Snip

```

import baseConfig.BaseSimulation

import io.gatling.core.Predef._
import io.gatling.http.Predef._

class CsvFeeder extends BaseSimulation {

  val csvFeeder = csv( fileName = "gameCsvFile.csv").circular

  def getSpecificVideoGame() : ChainBuilder = {
    repeat( times = 10 ) {
      feed(csvFeeder).
        exec(http( requestName = "Get Specific Video Game")
          .get("videogames/${gameId}")
          .check(jsonPath( path = "$.name").is( expected = "${gameName}") )
          .check(status.is( expected = 200))
          .pause( duration = 1 )
        )
    }
  }
}

```

Using fiddler to check the data

The screenshot shows the Fiddler Web Debugger interface. The top pane displays a list of HTTP requests with columns for #, Result, Protocol, Host, URL, Body, Caching, and Content-Type. The bottom pane shows the details of a selected request, including the raw HTTP data and the decrypted content. A large watermark text "Fiddler HTTP Proxy" is overlaid on the right side of the interface.

The endpoint change each time

One of the most powerful features of Gatling is the ability to write custom feeder's. Doesn't require an external file like a CSV file and it will give you the ability to generate an infinite stream value.

Task

Create a feeder that just changes the gameId each time.

- Create a variable to hold the id number

```
var idNumbers = (1 to 10).iterator
```

- Create a custom feeder and map with the idNumber

```
val customFeeder = Iterator.continually(Map("gameId" -> idNumbers.next()))
```

Continually : iterate over this mean every time you ask this feature for a new entry it will execute the code within the brackets in the brackets and return map.

The Map has just one key value pair.

Snip of Code

```

import baseConfig.BaseSimulation
import io.gatling.core.Predef._
import io.gatling.http.Predef._

class CsvFeederToCustom extends BaseSimulation {

  var idNumbers = (1 to 10).iterator

  val customFeeder = Iterator.continually(Map("gameId" -> idNumbers.next()))

  def getSpecificVideoGame() : ChainBuilder = {
    repeat( times = 10 ) {
      feed(customFeeder).
      exec(http( requestName = "Get Specific Video Game")
        .get("videogames/${gameId}")
        .check(status.is( expected = 200)))
      .pause( duration = 1)
    }
  }

  val scn = scenario( scenarioName = "Video Game DB")
    .exec(getSpecificVideoGame())

  setUp(
    scn.inject(atOnceUsers( users = 1))
  ).protocols(httpConf)
}

```

The test make calls

```

=====
---- Global Information -----
> request count                10 (OK=10   KO=0   )
> min response time            5 (OK=5     KO=-   )
> max response time           1022 (OK=1022 KO=-   )
> mean response time          109 (OK=109  KO=-   )
> std deviation                304 (OK=304  KO=-   )
> response time 50th percentile    6 (OK=6     KO=-   )
> response time 75th percentile    9 (OK=9     KO=-   )
> response time 95th percentile   570 (OK=570  KO=-   )
> response time 99th percentile   932 (OK=932  KO=-   )
> mean requests/sec             0.833 (OK=0.833 KO=-   )
---- Response Time Distribution -----
> t < 800 ms                   9 ( 90%)
> 800 ms < t < 1200 ms         1 ( 10%)
> t > 1200 ms                   0 (  0%)
> failed                        0 (  0%)
=====

```

You can make this similar another way

```
def getNextGameId() = Map("gameId" -> idNumbers.next())
val customFeeder = Iterator.continually(getNextGameId())
```

Learn to POST data and create game API to create a new game in the db for us.

Task POST data using Gatling

Click on simulation and right click and choose new Scala class with name "CustomFeed"

- Create a map to data json

```
val customFeeder = Iterator.continually(Map(
  "gameId" -> ???,
  "name" -> ???,
  "releaseDate" -> ???,
  "reviewScore" -> ???,
  "category" -> ???,
  "rating" -> ???
))
```

- Create a idNumber variable

```
var idNumbers = (11 to 20).iterator
```

- Need to give our games a name (there a several ways we could do this create a name = Game- xxxxx => it refer to 5 random character
- Create a random variable

```
val rnd = new Random()
```

- Make method take random character

```
def randomString(length: Int): String = {
  rnd.alphanumeric.filter(_._isLetter).take(length).mkString
}
```

- Create a random data


```
def getRandomDate(startDate: LocalDate, random: Random):String = {
  startDate.minusDays(random.nextInt(30)).format(pattern)
}
```

```
val customFeeder = Iterator.continually(Map(
  "gameId" -> idNumbers.next(),
  "name" -> ("Game-" + randomString( length = 5)),
  "releaseDate" -> getRandomDate(now, rnd),
  "reviewScore" -> rnd.nextInt(100),
  "category" -> ("Category-" + randomString( length = 6)),
  "rating" -> ("Rating-" + randomString( length = 4))
))
```

- Create a POST method

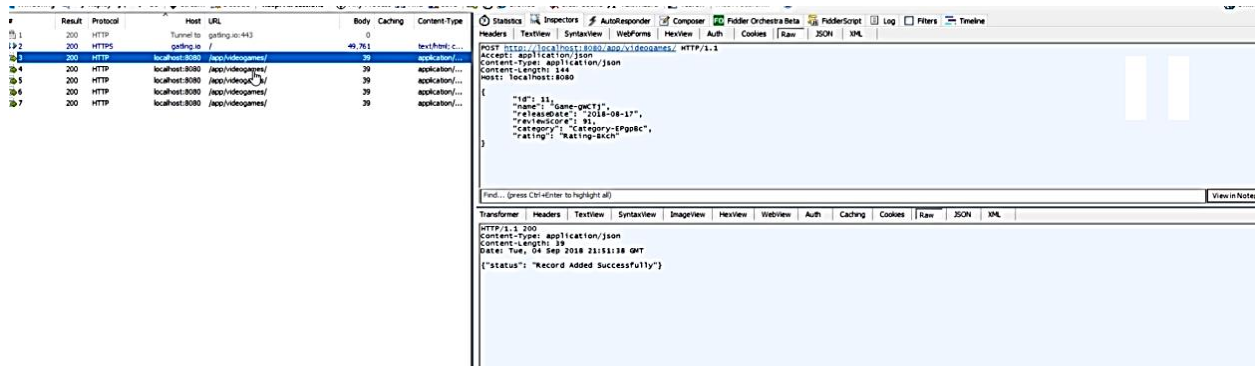
```
def postNewGame() : ChainBuilder = {
  repeat( times = 5 ) {
    feed(customFeeder).
    exec(http( requestName = "Post New Game")
      .post( url = "videogames/")
      .body(StringBody(
        string = "{" +
          "\n\t\"id\": ${gameId}," +
          "\n\t\"name\": \"${name}\" +
          "\n\t\"releaseDate\": \"${releaseDate}\" +
          "\n\t\"reviewScore\": ${reviewScore}," +
          "\n\t\"category\": \"${category}\" +
          "\n\t\"rating\": \"${rating}\""
        ).asJSON
      ).check(status.is( expected = 200)))
    .pause( duration = 1)
  }
}
```

- Add scenario

```
val scn = scenario( scenarioName = "Video Game DB")
  .exec(postNewGame())

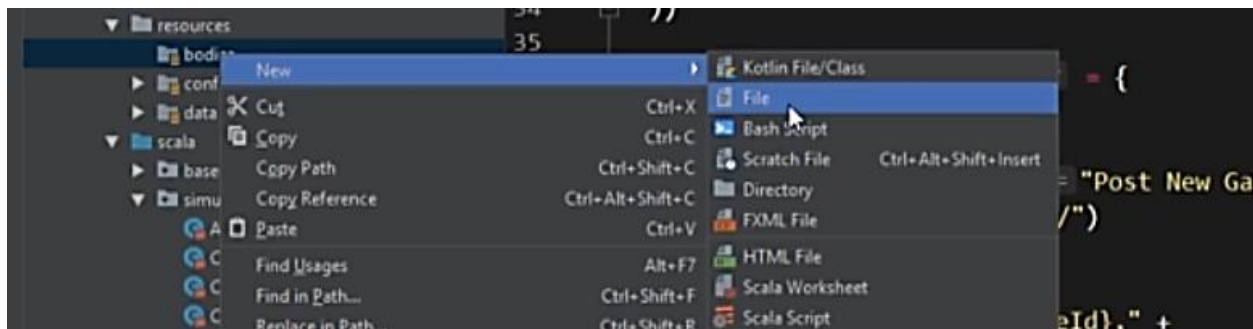
setUp(
  scn.inject(atOnceUsers( users = 1))
).protocols(httpConf)
```

- Open proxy to show created POST data

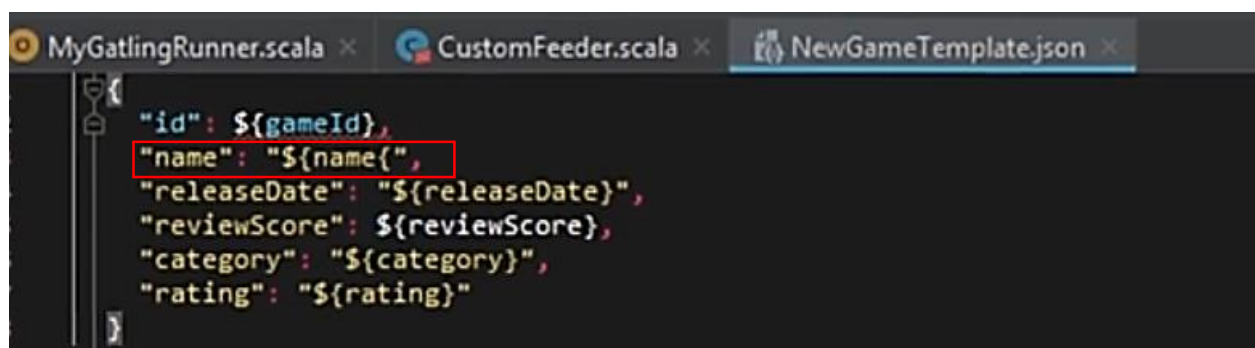


Now show clean way to write code for using template file.

Template file or matched the request body that we send to the create new game call but each of the values will be parameterized.



Give name newGameTemplate.json



Change method to =>

```
def postNewGame() : ChainBuilder = {
  repeat( times = 5 ) {
    feed(customFeeder).
    exec(http( requestName = "Post New Game")
      .post( url = "videogames/" )
      .body(ElFileBody("NewGameTemplate.json")) .asJSON
      .check(status.is( expected = 200 )))
    .pause( duration = 1 )
  }
}
```

To delete the data API

```
2018-09-04 22:52:54.989 DEBUG 9708 --- [nio-8080-exec-7] o.s.jdbc.datasource.DataSourceUtils
ing JDBC Connection to DataSource
2018-09-04 22:52:54.991 DEBUG 9708 --- [nio-8080-exec-7] o.s.web.filter.RequestContextFilter
d thread-bound request context: org.apache.catalina.connector.RequestFacade@79faa390
> Building 80% > :bootRunTerminate batch job (Y/N)? |
```

The template file is common practice

Load simulation design

Section cover

- Basic load simulation Design
- Ramp up users Per second
- Running for fixed duration

(the examples that you see in this section should help you to design a board low test simulations for your application)

(the time has come to start designing our load simulations to have more than one user and run four different periods of time)

Design simulation

Injection profile

Control how users are injected in your scenario

Injection steps

nothingFor

Pauses for a specific duration

(dur unit)

Pause for a given duration

atOnceUsers

Injects a specific number of users at the same time

(nbUsers)

Injects nbUsers

rampUsers

Injects a given number of users with a linear ramp over a given duration

(nbUsers) over(dur unit)

Injects nbUsers over a duration. Eg :
rampUsers(10) over(5 seconds)

constantUsersPerSec

Injects users at a constant rate, defined in users per second, during a given duration

(nbUsers) during(dur unit)

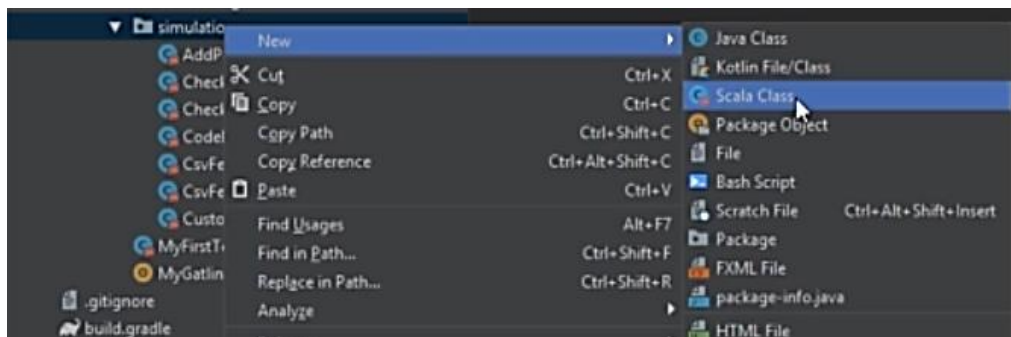
Injects nbUsers each second for duration. Eg :
constantUsersPerSec(10) during(5 seconds)

rampUsersPerSec

splitUsers

heavisideUsers

Create a simulation



Task

- Create a two method and scenario
 - GET API
 - GET Specific API
 - Scenario for two GET

Snap

```
def getAllVideoGames() : ChainBuilder = {
  exec(
    http( requestName = "Get All Video Games")
      .get("videogames")
      .check(status.is( expected = 200)))
  }

def getSpecificVideoGame() : ChainBuilder = {
  exec(http( requestName = "Get Specific Video Game")
    .get("videogames/2")
    .check(status.is( expected = 200)))
  }
}
```

Scenario

```
val scn = scenario( scenarioName = "Video Game DB")
  .exec(getAllVideoGames())
  .pause( duration = 5)
  .exec(getSpecificVideoGame())
  .pause( duration = 5)
  .exec(getAllVideoGames())
```

Now, we are injecting user to our scenario

- Make a stop for five users at once at once users 5 and add 10 users over a period of 10 sec
- Make HTML infer resource (this mean is that Gatling will fetch everything on the page like CSS , JavaScript, images and extra)

(this is a good thing to do if you wish stress testing a page rich in resources)

```
setUp(
  scn.inject(
    nothingFor(5 seconds),
    atOnceUsers( users = 5),
    rampUsers( users = 10) over (10)
  ).protocols(httpConf.inferHtmlResources())
)
```

Result

- 5 few sec nothing happen
- After 5 user at once

(1/2) 2	200	HTTP	localhost:8080	/app/videogames	1,183	application/...
(1/3) 3	200	HTTP	localhost:8080	/app/videogames	1,183	application/...
(1/4) 4	200	HTTP	localhost:8080	/app/videogames	1,183	application/...
(1/5) 5	200	HTTP	localhost:8080	/app/videogames	1,183	application/...
(1/6) 6	200	HTTP	localhost:8080	/app/videogames	1,183	application/...

Calculation

Each user and makes three calls for a total of 45 cells

15 users x 3 calls each = 45 transactions

Task make a Scala class of name “RampUserLoadSimulation”

- 1- The first option we will look at is adding a constant number of users every sec for a fixed duration.
 - a. Make at 10 users a second for 10 second (Add 10 users every 10 seconds = 100 user a total)

```
setUp(
  scn.inject(
    nothingFor(5 seconds),
    constantUsersPerSec( rate = 10) during (10 seconds)
  ).protocols(httpConf.inferHtmlResources())
)
```

As doc, let`s look at another method of ramping users ramp users per second as per the documentation

(we will be providing an **initial starting rate** to inject and **a target rate to reach**)

- Make a start adding 1 new user per second and then increase that to 5 new users per second, over a period of 20 second.

```
setUp(
  scn.inject(
    nothingFor(5 seconds),
    / constantUsersPerSec(10) during (10 seconds)
    rampUsersPerSec( rate1 = 1) to (5) during(20 seconds)
  ).protocols(httpConf.inferHtmlResources())
)
```

In all the scenario so far each virtual user will execute their scenario flow

- Call (games wait five sec , call specific game wait 5 sec and call GET all game and then finish a common use case in performance test script is for the users to loop their journey such that performance test runs for a fixed duration.

Scenario

- GetAllGames()
- GetSpecificGame()
- GetAllGames()

*Run the above scenario
continuously for 60 seconds*

- Make a fixed duration load simulation make a method to loop a scenario forever so we can design a load simulation that will run for a fixed duration.

```
val scn = scenario( scenarioName = "Video Game DB")
    .forever() {
        exec(getAllVideoGames())
        .pause( duration = 5)
        .exec(getSpecificVideoGame())
        .pause( duration = 5)
        .exec(getAllVideoGames())
    }
```

(Now that said 10 users at once and let me add another 50 users over a period of 30 sec at the end of our load simulations and make maximum duration at 1 min

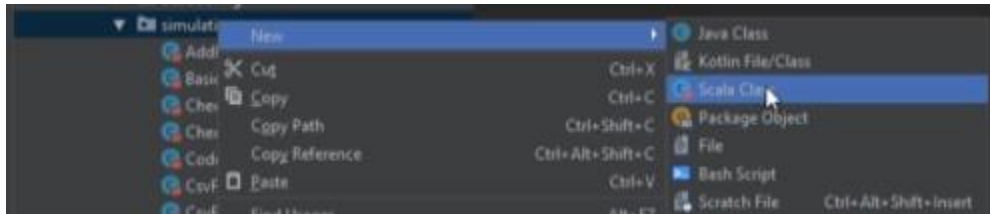
```
setUp(
    scn.inject(
        nothingFor(5 seconds),
        atOnceUsers( users = 10),
        rampUsers( users = 50) over (30 seconds)
    ).protocols(httpConf.inferHtmlResources())
).maxDuration( duration = 1 minute)
```

Cover

- Run test script from console
- How to enable runtime parameters in our Gatling scripts to customize our low testing simulates at execution time.

(so far we have been running all of our gatling scripts by executing then in feddle galling that`s fine for debugging but when com to runner to run our performance test in production and test environments we want to trick them form the command line)

Create a scala class of “RuntimeParameters”



Snap of Code

```
package simulations

import baseConfig.BaseSimulation
import io.gatling.core.Predef._
import io.gatling.http.Predef._

class RuntimeParameters extends BaseSimulation {

  def getAllVideoGames() : ChainBuilder = {
    exec(
      http(requestName = "Get All Video Games")
        .get("videogames")
        .check(status.is(expected = 200)))
    )
  }

  val scn = scenario( scenarioName = "Video Game DB")
    .forever() {
      exec(getAllVideoGames())
    }

  setUp(
    scn.inject(
      nothingFor(5 seconds),
      rampUsers( users = 1) over (1 second))
    )
    .protocols(httpConf).maxDuration(20 seconds)
}
```

Navigate your project folder

- To run install plugin

<https://github.com/lkishalmi/gradle-gatling-plugin>

```
c:\Gatling\gatling-fundamentals-projectStart-master
\ gradlew gatlingRun-simulations.RuntimeParameters
```

In our case it's in the simulations package and it's called runtime parameters

(A Technique that you will find indispensable when you come to add performance testing to your larger testing project and build pipeline)

Now, pass a parameter on command line

Task => First at a help and method to our script that will even read the runtime parameter from the command line or fall back to a default parameter when no parameter is passed at the top of the script declared

- Make a private method take two parameter propertyName and defaultValue
- If the Gatling can't find the environment variable this will make Gatling look for a system property instead
- If Gatling can't find either environment variable or system property with specific name the method will instead return the default value that you specified

```
private def getProperty(propertyName: String, defaultValue: String): String = {
  Option(System.getenv(propertyName))
    .orElse(Option(System.getProperty(propertyName)))
    .getOrElse(defaultValue)
}
```

Make a variable

```
def userCount: Int = getProperty( propertyName = "USERS", defaultValue = "5").toInt
def rampDuration: Int = getProperty( propertyName = "RAMP_DURATION", defaultValue = "10").toInt
def testDuration: Int = getProperty( propertyName = "DURATION", defaultValue = "60").toInt
```

Add debug code that will print out the properties to command line at the start test before print line running

```
before {  
  println(s"Running test with ${userCount} users")  
  println(s"Ramping users over ${rampDuration} seconds")  
  println(s"Total Test duration: ${testDuration} seconds")  
}
```

Modify the scenario

```
setUp(  
  scn.inject(  
    nothingFor(5 seconds),  
    rampUsers(userCount) over (rampDuration)  
  )  
  .protocols(httpConf).maxDuration(testDuration)
```

Snap of Code

```

class RuntimeParameters extends BaseSimulation {

  private def getProperty(propertyName: String, defaultValue: String): String = {
    Option(System.getenv(propertyName))
      .orElse(Option(System.getProperty(propertyName)))
      .getOrElse(defaultValue)
  }

  def userCount: Int = getProperty(propertyName = "USERS", defaultValue = "5").toInt
  def rampDuration: Int = getProperty(propertyName = "RAMP_DURATION", defaultValue = "10").toInt
  def testDuration: Int = getProperty(propertyName = "DURATION", defaultValue = "60").toInt

  before {
    println(s"Running test with ${userCount} users")
    println(s"Ramping users over ${rampDuration} seconds")
    println(s"Total Test duration: ${testDuration} seconds")
  }

  def getAllVideoGames(): ChainBuilder = {
    exec(
      http(requestName = "Get All Video Games")
        .get("videogames")
        .check(status.is(expected = 200)))
  }

  val scn = scenario(scenarioName = "Video Game DB")
    .forever() {
      exec(getAllVideoGames())
    }

  setUp(
    scn.inject(
      nothingFor(5 seconds),
      rampUsers(userCount) over (rampDuration))
  )
  .protocols(httpConf).maxDuration(testDuration)
}

```

Run from command line

```

c:\Gatling\gatling-fundamentals-projectStart-master
λ gradlew gatlingRun-simulations.RuntimeParameters -DUSERS=3 -DRAMPDURATION=5 -DDURATION=30

```

Don't supply any parameter

```

Running test with 5 users
Ramping users over 10 seconds
Total Test duration: 60 seconds
Simulation simulations.RuntimeParameters started...
<=====--> 92% EXECUTING [4s]
> :gatlingRun-simulations.RuntimeParameters

```

Task

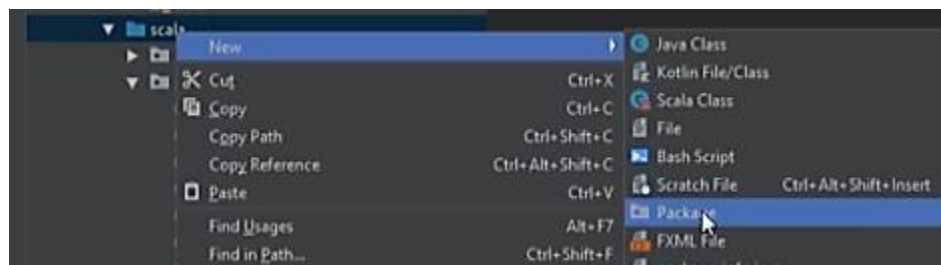
- 1- Call all GET All Games
- 2- Call a New Game (use a custom feeder)
- 3- Call GET Single a Game (of the game created above) => you need to save variables from the previous request to do this)
- 4- Call Delete Game (delete the game just created)

Pre-request

- 1- Create a method for the API calls so that
- 2- they can be reused at appropriate checks and assertions after
- 3- Add pause Time
- 4- Create a load scenario that can be customized by runtime parameters input on the command line
 - a. Number of users
 - b. Test duration
 - c. Make default if there are none specified print messages to console at the start of the test needs
 - d. Print the message to console At Start / End of Test (This test is running with X users for y seconds and print out a message once the test finishes

Create a template

- Create a package



- Make a new Scala class (video game for test)

Snap of solution

1.

```
import java.time.LocalDate
import java.time.format.DateTimeFormatter

import baseConfig.BaseSimulation
import io.gatling.core.Predef._
import io.gatling.http.Predef._

import scala.concurrent.duration.DurationInt
import scala.util.Random

class VideoGameFullTest extends BaseSimulation {

  /** Variables */
  // runtime variables
  def userCount: Int = getProperty( propertyName = "USERS", defaultValue = "3").toInt
  def rampDuration: Int = getProperty( propertyName = "RAMP_DURATION", defaultValue = "10").toInt
  def testDuration: Int = getProperty( propertyName = "DURATION", defaultValue = "60").toInt

  // other variables
  var idNumbers = (20 to 1000).iterator
  val rnd = new Random()
  val now = LocalDate.now()
  val pattern = DateTimeFormatter.ofPattern( pattern = "yyyy-MM-dd")

  /** Helper Methods */
  private def getProperty(propertyName: String, defaultValue: String): String = {
    Option(System.getenv(propertyName))
      .orElse(Option(System.getProperty(propertyName)))
      .getOrElse(defaultValue)
  }

  def randomString(length: Int): String = {
    rnd.alphanumeric.filter(_._isLetter).take(length).mkString
  }

  def getRandomDate(startDate: LocalDate, random: Random): String = {
    startDate.minusDays(random.nextInt(30)).format(pattern)
  }
}
```


2.

```

def postNewGame() : ChainBuilder = {
  feed(customFeeder).
  exec(http( requestName = "Post New Game")
    .post( url = "videogames")
    .body(ElFileBody("NewGameTemplate.json")).asJSON //template file goes in gating/resources/bodies
    .check(status.is( expected = 200)))
}

def getLastPostedGame() : ChainBuilder = {
  exec(http( requestName = "Get Last Posted Game")
    .get("videogames/${gameId}")
    .check(jsonPath( path = "$.name").is( expected = "${name}"))
    .check(status.is( expected = 200)))
}

def deleteLastPostedGame() : ChainBuilder = {
  exec(http( requestName = "Delete Last Posted Game")
    .delete( url = "videogames/${gameId}")
    .check(status.is( expected = 200)))
}

/** Scenario Design */
val scn = scenario( scenarioName = "Video Game DB")
  .forever() {
    exec(getAllVideoGames())
    .pause( duration = 2)
    .exec(postNewGame())
    .pause( duration = 2)
    .exec(getLastPostedGame())
    .pause( duration = 2)
    .exec(deleteLastPostedGame())
  }
}

```

3.

```

/** Setup Load Simulation */
setUp(
  scn.inject(
    nothingFor(5 seconds),
    rampUsers(userCount) over (rampDuration seconds))
)
  .protocols(httpConf)
  .maxDuration(testDuration seconds)

/** After */
after {
  println("Stress test completed")
}
}

```

Covering section

- 1- Monitoring Test execution in Real Time
- 2- Analyze the Gatling Test Report

```
=====
2018-09-24 07:04:42                                     10s elapsed
---- Requests ----
> Global                                           (OK=3    KO=0    )
> Get All Video Games                             (OK=2    KO=0    )
> Post New Game                                   (OK=1    KO=0    )
----- Video Game DB -----
```

It shows the time elapsed since the test started

```
2018-09-24 07:04:47                                     15s elapsed
---- Requests ----
> Global                                           (OK=9    KO=0    )
> Get All Video Games                             (OK=4    KO=0    )
> Post New Game                                   (OK=2    KO=0    )
> Get Last Posted Game                           (OK=2    KO=0    )
> Delete Last Posted Game                         (OK=1    KO=0    )
----- Video Game DB -----
```

The name of test

```
=====
2018-09-24 07:04:47                                     15s elapsed
---- Requests ----
```

The timestamp

```
=====
waiting: 0      / active: 3      / done:0
=====
```

The number of waiting / active / done virtual user

```
---- Requests ----
> Global                                           (OK=21    KO=0    )
> Get All Video Games                             (OK=7     KO=0    )
> Post New Game                                   (OK=5     KO=0    )
> Get Last Posted Game                           (OK=5     KO=0    )
> Delete Last Posted Game                         (OK=4     KO=0    )
----- Video Game DB -----
```

The transaction name of the request that have been made.

```

-----
(OK=21      KO=0      )
(OK=7       KO=0      )
(OK=5       KO=0      )
(OK=5       KO=0      )
(OK=4       KO=0      )
-----

```

Done of pass ok

```

----- Errors -----
> status.find.is(200), but actually found 500                2 (50.00%)
> j.n.ConnectException: Connection refused: no further informati  2 (50.00%)
on: localhost/127.0.0.1:8888
-----

```

If the script an error details

```

----- Global Information -----
> request count                95 (OK=87      KO=8      )
> min response time            0 (OK=2       KO=0      )
> max response time           5148 (OK=5148    KO=173   )
> mean response time          104 (OK=110     KO=35    )
> std deviation                558 (OK=583     KO=56    )
> response time 50th percentile  4 (OK=4      KO=11    )
> response time 75th percentile  5 (OK=5      KO=26    )
> response time 95th percentile 423 (OK=710   KO=136   )
> response time 99th percentile 1264 (OK=1594  KO=166   )
> mean requests/sec           1.727 (OK=1.582   KO=0.145 )
----- Response Time Distribution -----
> t < 800 ms                  82 ( 86%)
> 800 ms < t < 1200 ms        4 (  4%)
> t > 1200 ms                  1 (  1%)
> failed                       8 (  8%)
----- Errors -----
> status.find.is(200), but actually found 500                5 (62.50%)
> j.n.ConnectException: Connection refused: no further informati  2 (25.00%)
on: localhost/127.0.0.1:8888
> jsonPath($.name).find.is(Game-dKAU0), but actually found Game-  1 (12.50%)
ihXsH
=====

```

Additional test information for the run.

Response time percentage mean the percentage of response times that completed within specified time.

```

----- Response Time Distribution -----
> t < 800 ms                               82 ( 86%)
> 800 ms < t < 1200 ms                     4 (  4%)
> t > 1200 ms                              1 (  1%)
> failed                                    8 (  8%)
----- Errors -----

```

We also see breakdown of the response time distribution and again a summary of any errors encountered.

- We can find the location of the report by looking at the message that gets output on the console at the end of the test execution.

```

Reports generated in 0s.
Please open the following file: C:\Gatling\gatling-fundamentals-projectStart-master\build\reports\gatling\videogamefulltest-1537422390217\index.html

```

The total number of request broken down to response time of less than

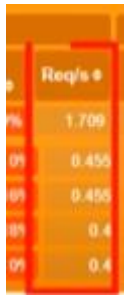
Pie chart number of request is a pie-chart that shows the different type of requests made which one failed.

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	94	86	8	9%	1.709	0	4	5	55	4704	5028	168	840
Get All Video Games	25	25	0	0%	0.455	2	3	4	16	771	1009	44	197
Post New Game	25	21	4	16%	0.455	0	4	4	11	98	126	9	24
Get Last...ted Game	22	18	4	18%	0.4	0	5	6	4330	4654	4680	424	1327
Delete L...ted Game	22	22	0	0%	0.4	2	3	4	10	3974	5028	232	1047

Statistics give a a breakdown of the different transaction whether they passed or failed

Execution	
OK	KO
86	8
25	0
21	4
18	4
22	0

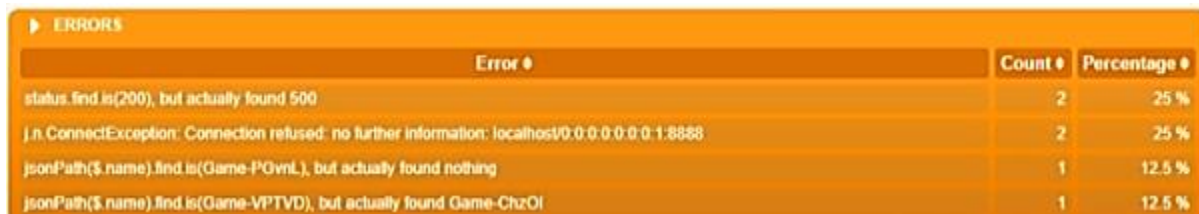
Request per second



Req/s #
1.709
0.455
0.455
0.4
0.4

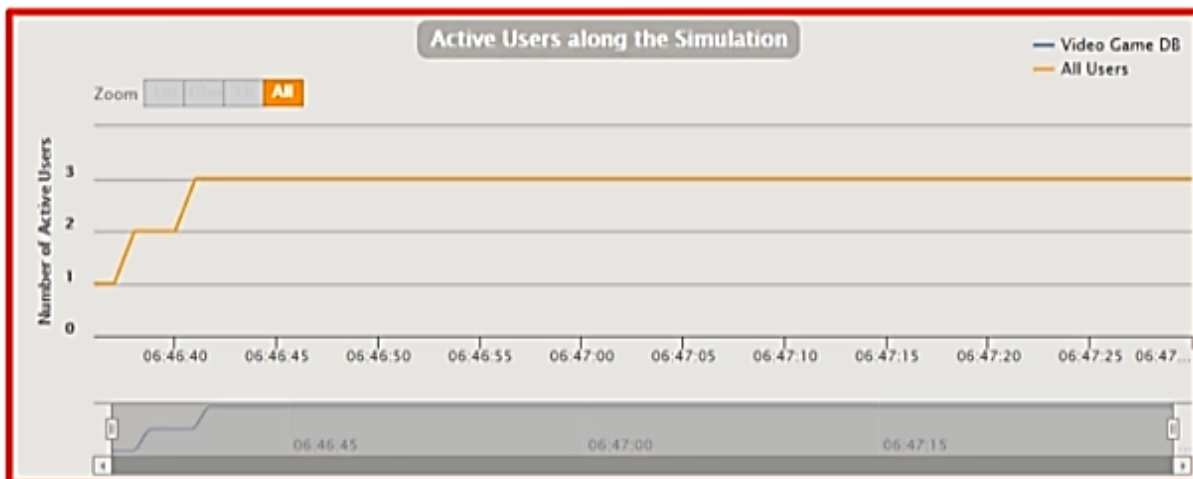
And various response time metric such min and max

Error show details and count of any error identified in the test.



Error #	Count #	Percentage #
status.find.is(200), but actually found 500	2	25 %
j.n.ConnectException: Connection refused: no further information: localhost/0.0.0.0:0.0.0.0:18888	2	25 %
jsonPath(\$.name).find.is(Game-PGvML), but actually found nothing	1	12.5 %
jsonPath(\$.name).find.is(Game-VPTVD), but actually found Game-ChzOI	1	12.5 %

Growing for active user



- Learn more Scala
- Continuous Integration
- Gatling Google Forums -

<https://groups.google.com/forum/#!forum/gatling>