Performance Testing with Gatling + Jenkins Integration

Ashish Khole
Director, QA at ICE Mortgage Technology
Published Apr 20, 2021

Performance Testing has become a crucial part of SDLC. Performance of an application has become a key as it can impact the user experience badly. A performance test let you measure the performance and robustness of your application. The aim is to simulate a high number of concurrent users, as close as possible to real cases. You can then anticipate crashes and have a more precise vision of the response times of all your users, especially those with the worst response times.

There are different types of performance tests:

- **Load test:** Simulating an expected load of users
- **Capacity test:** Increasing step by step the number of users during the simulation, in order to know the limits of your system
- **Stress test:** Simulating a very high number of users using your application at the same time

performance degradations over time

There are lots of tools available in market to perform performance tests. Widely used is **JMeter.** For our performance test we are going to use **Gatling. Gatling** is an open source load testing framework based on Scala and is a very powerful tool for Load Testing. To learn about why we should consider Gatling for Performance Testing, **click here.**

What we will achieve by end of this post?

- Setup Gatling with Maven
- Performance Test a load scenario with help of some online APIs
- Analyze the results & Report
- Integrate the performance script with Jenkins

Application under test

For this tutorial I will be using the open source JsonPlaceholder APIs: **https://jsonplaceholder.typicode.com/** for our test.

Performance Test Scenario

Let's plan the following API actions user will perform in test scenario :

User will browse through some posts

User will open a random post

User will check the comments for that post

User will add comments of his own to this post

Defining load pattern

Simulate 2 users starting, with 3 new ones adding at random intervals. I want the test to run for 2 minutes

So lets get started with the setup. To get started you will need some pre-requisites

Pre-Requisites

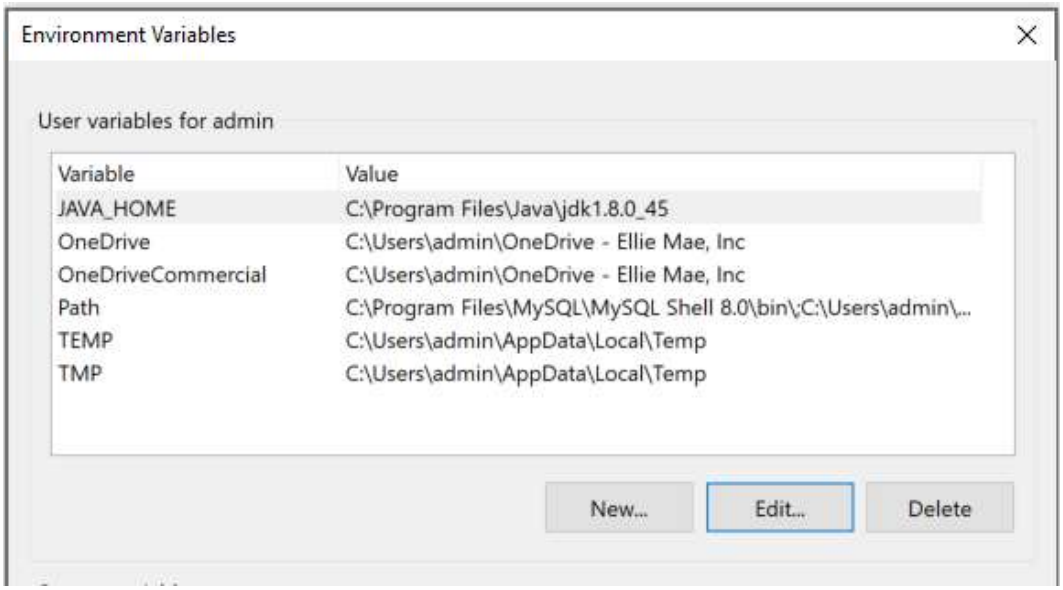
- Java 8 JDK

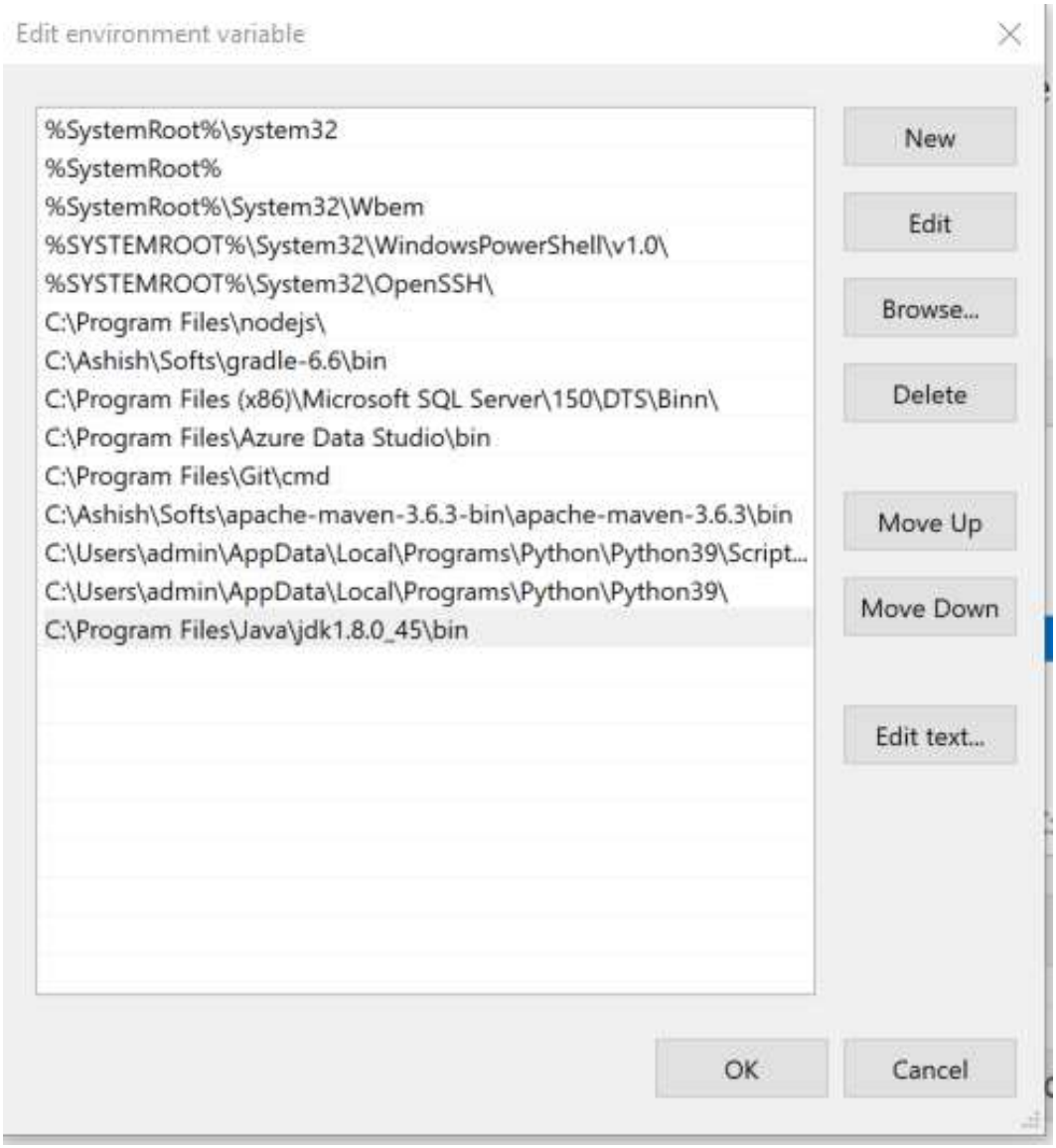
- Gatling
- Intellij IDE & Scala Plugin

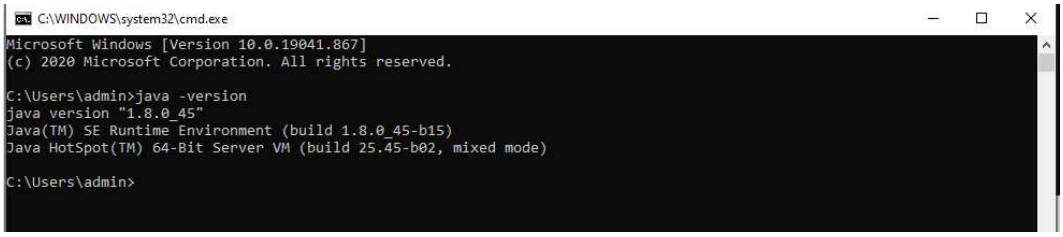
STEP 1: Install Java 8 JDK

Download the Java 8 JDK (64 bit) package from **here** and install it. Default installation will be usually **"C:\Program Files\ Java"**

Set the environment variables by: Start > type in environment variables > click 'Edit the system environment variables'. Add JAVA_HOME and add java bin to the PATH variable.



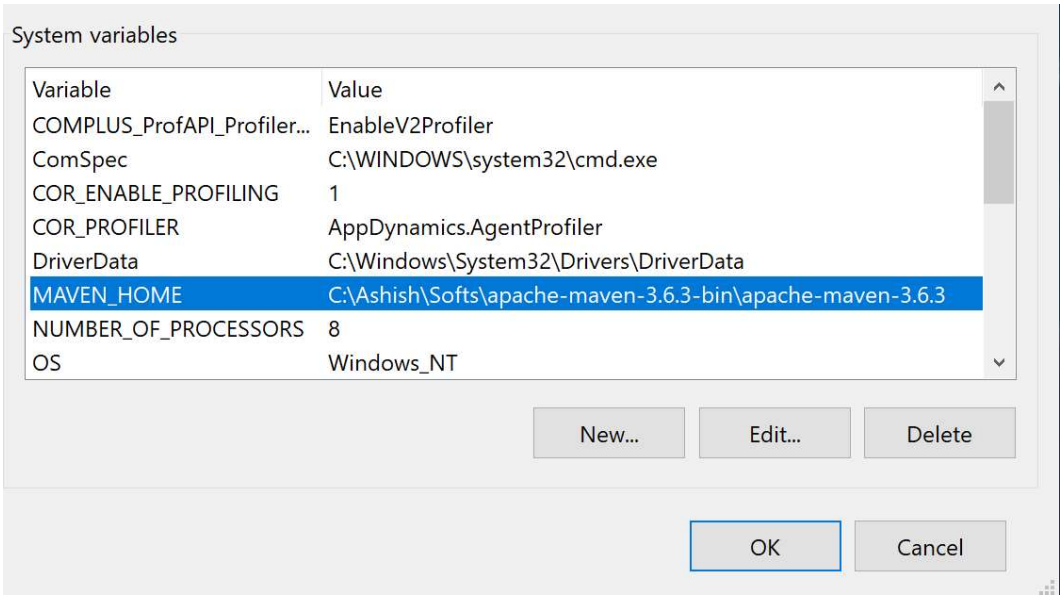




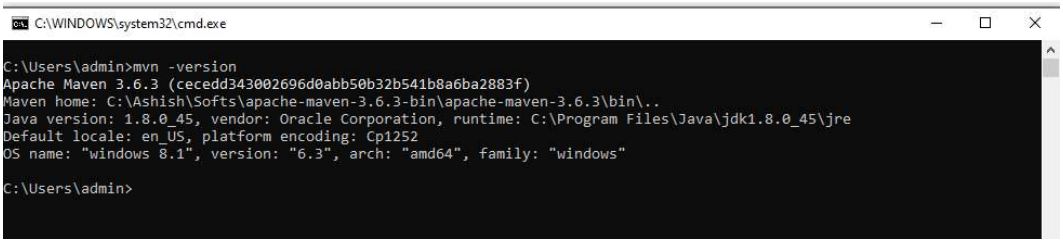
Verify your java installation. Open a command prompt and enter command “java -version”.

STEP 2: Install Apache Maven

PATH similar to the steps we followed for JDK installation.

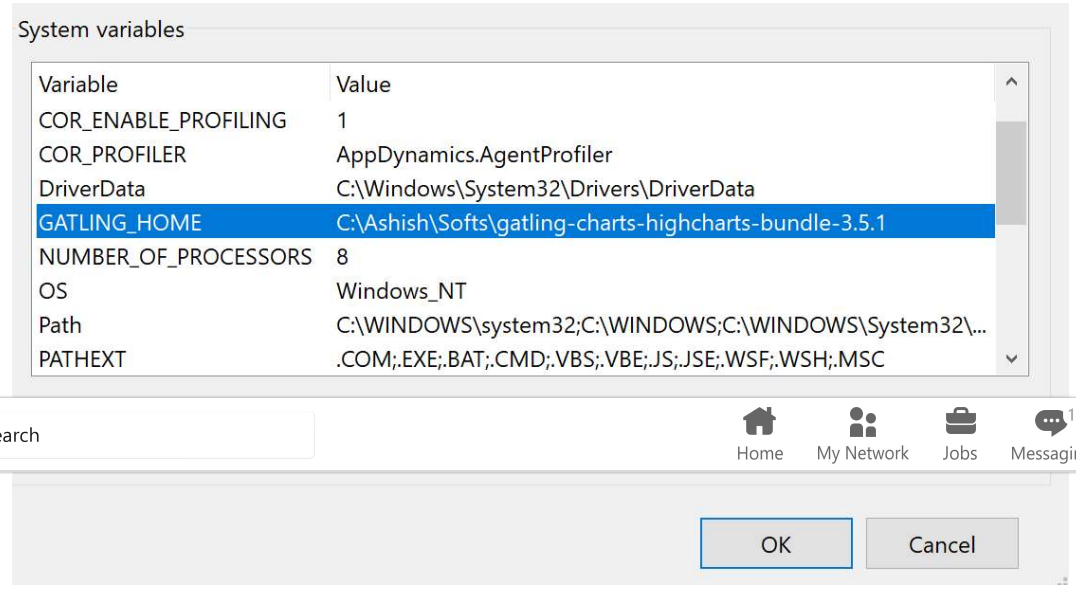


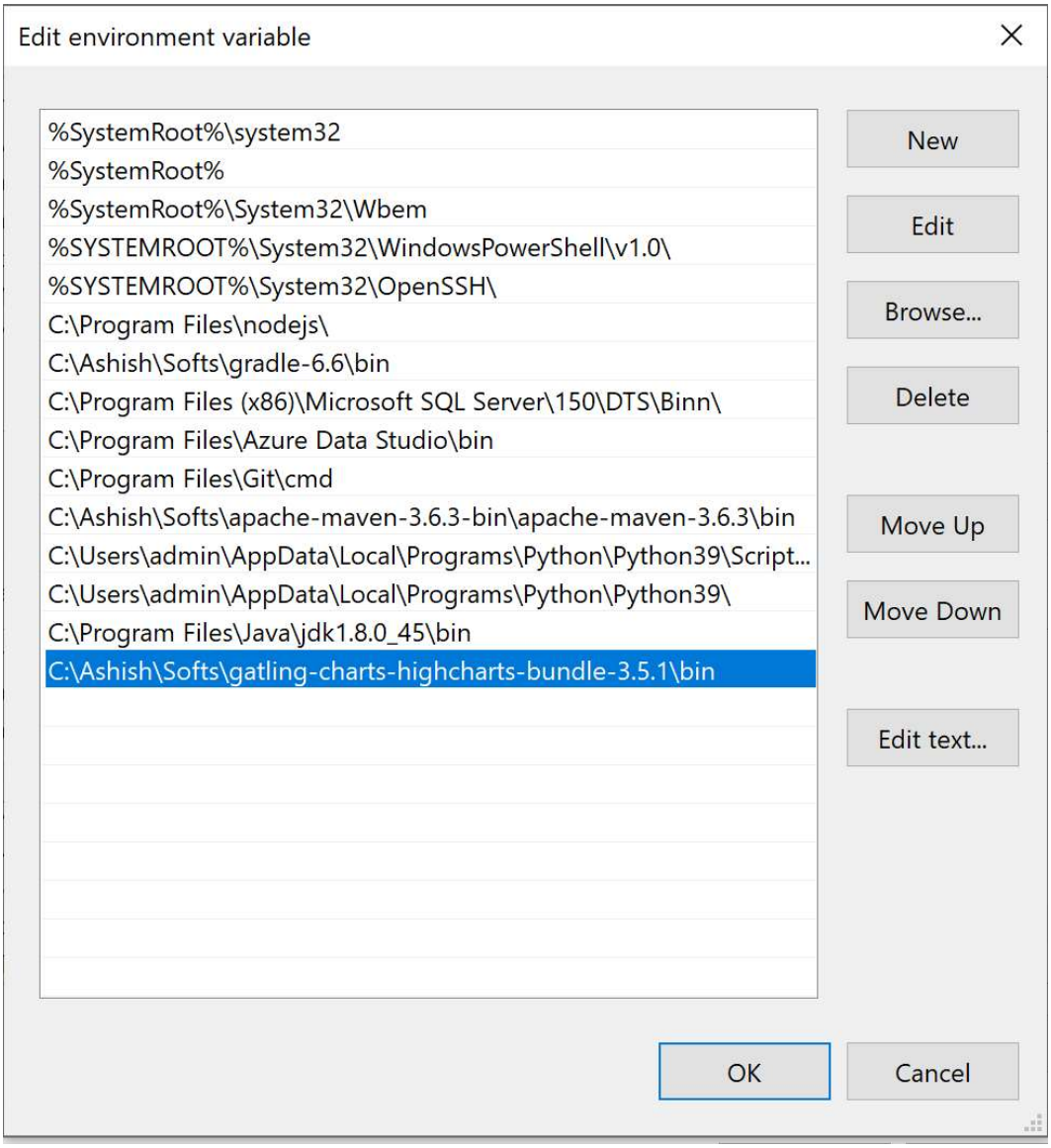
Verify your maven installation. Open a command prompt and enter command “mvn -v”. The result should be something like



STEP 3: Install Gatling

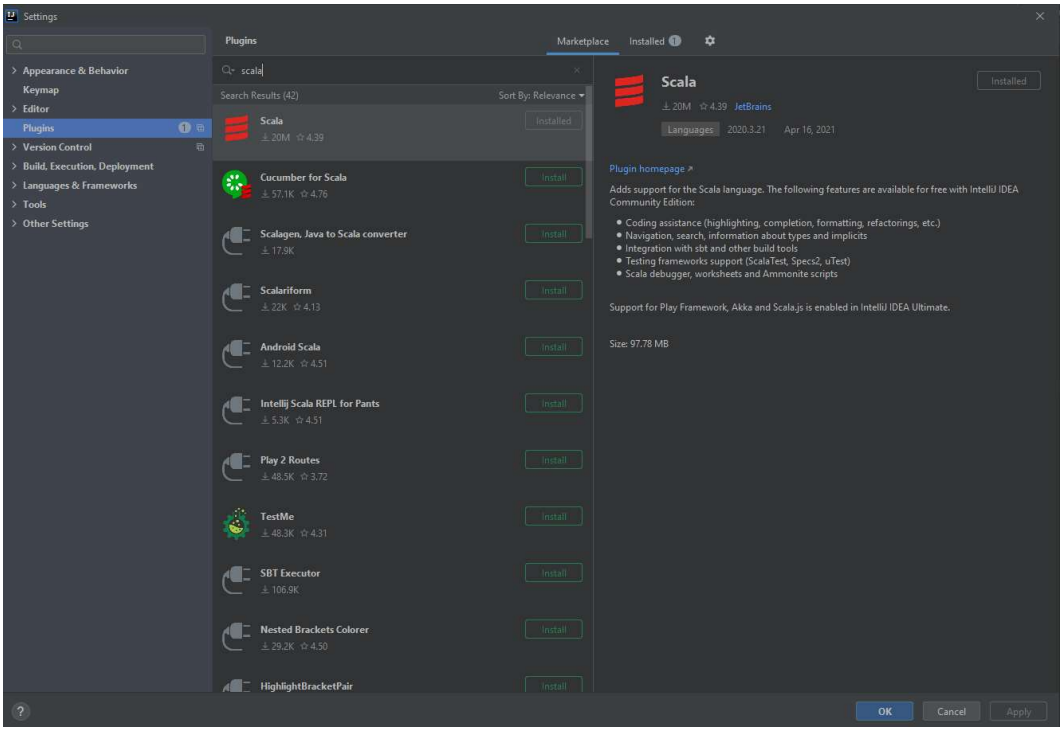
Download the Gatling community version from **here** . Unzip the downloaded zip in your desired location. Once unzipped, edit the environment variables to set GATLING_HOME and PATH.





STEP 4: Install IntelliJ with Scala Plugin

Download the Intellij IDE from **here** & complete the installation. When IntelliJ is first opened after install, the user is prompted to set display settings and to select plugins. Choose Scala plugin and install OR go to **settings -> plugin section** and search for "**scala**" and install the plugin.

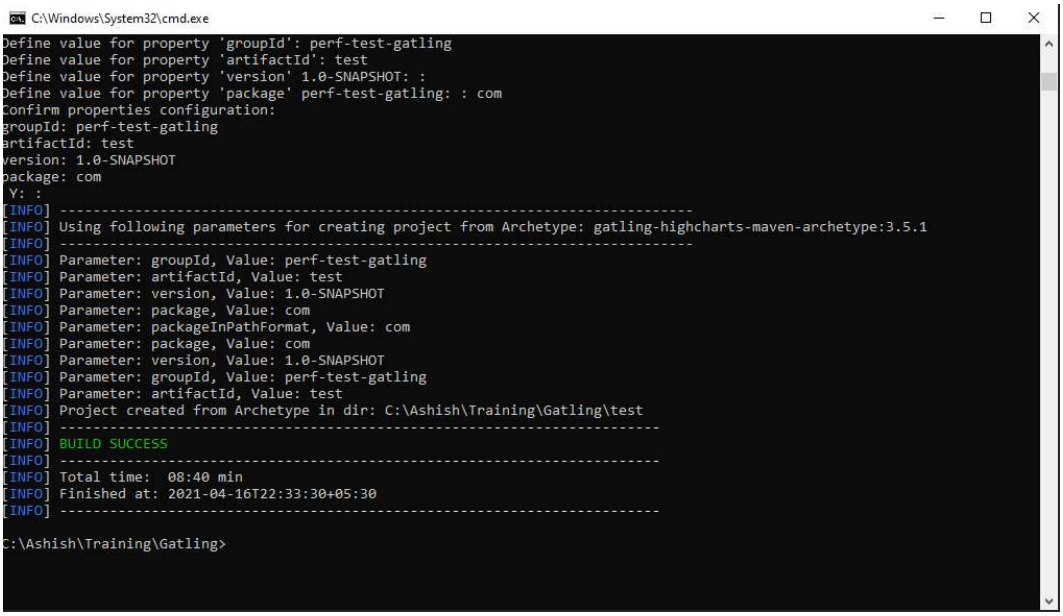


STEP 5: Creating first Gatling Performance Project

Let's create a project using maven archetype. Open your command prompt and navigate to folder where you want to create your project(I'm using "C:\Ashish\Training\Gatling"). Use following command

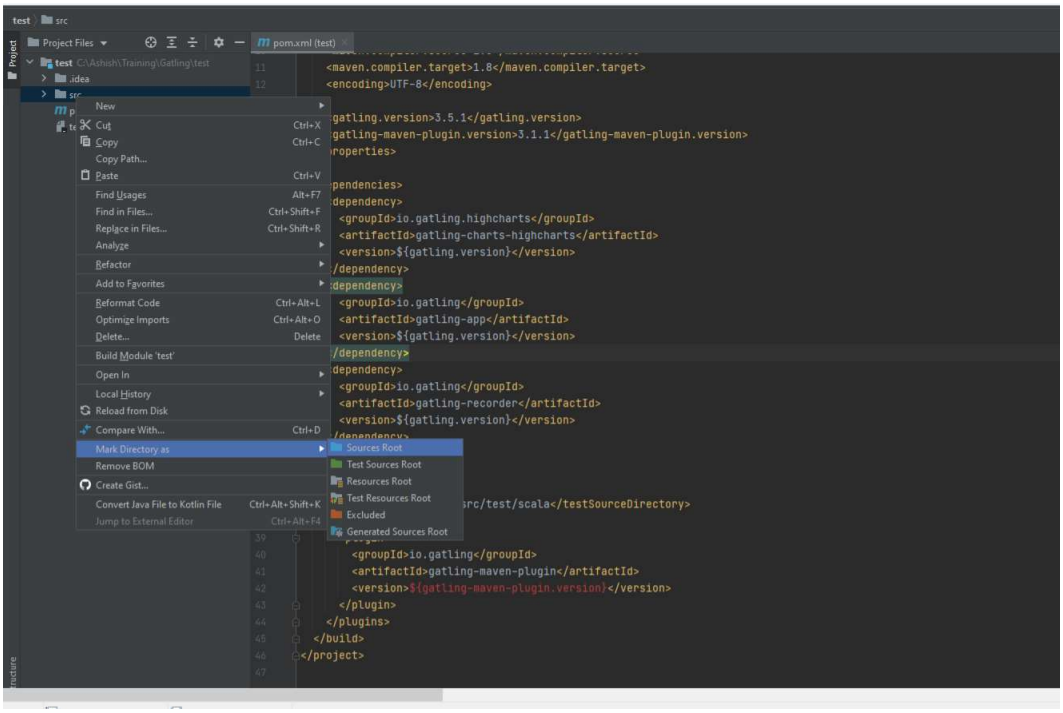
mvn archetype:generate -DarchetypeGroupId=io.gatling.highcharts -DarchetypeArtifactId=gatling-highcharts-mav

This command will download all the required dependencies for the project and at the end will prompt to define the group id, artifact id, version, package details for the project. This should look something like this:



Now open your project in Intellij and you should see following project structure:

The src folder that is created should be set to sources root so IntelliJ knows to compile this code. To achieve this right click on src directory and select **"Mark directory as Sources Root"** option



Search Home My Network Jobs Messaging Notifications Me

Before we start here is a skeleton of the script which it looks like:

```
//define an optional package to hold the Scala class
package simulations

//required imports
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration._

//class declaration
class BasicSimulation extends Simulation {
//Step1: Define Common HTTP protocol configuration
//Step2: Define Headers
//Step3: Define Feeders
//Step4: Define HTTP Requests
//Step5: Define Scenario
//Step6: Load Injection pattern
//Step7 (optional): Define before hook
//Step8 (optional): Define after hook
}
```

Create a package named "**Simulations**" under the scala folder. Right click on **"scala" -> New -> Package .** Create new file "**TestFirstBasicAPISimulation.scala**" under "**Simulations**" package and add the following code snippet.

```
package Simulations

//Import required object and references

import io.gatling.core.Predef._
import io.gatling.core.scenario.Simulation
import io.gatling.core.structure.ScenarioBuilder
import io.gatling.http.Predef._
import io.gatling.http.protocol.HttpProtocolBuilder
import scala.concurrent.duration._


//Extend Testclass with Simulation class
```


```
//Step1: Define a common http protocol config

  val httpConf: HttpProtocolBuilder = http.baseUrl("https://jsonplaceholder.typicode.com")
```

```
    .header("Accept","application/json")

//Step 2: Define variable
  val postIdNumbers = scala.util.Random

//Step 3: Define the scenario

  val scn: ScenarioBuilder = scenario("PerfTest Scenario - 4 calls")
    .exec(http("Get all Posts - 1st Call")
      .get("/posts")
      .check(status.is(200))
    )
    .pause(5)

    .exec(http("Get Specific Post - 2nd Call")
      .get("/posts/"+postIdNumbers.nextInt(20))
      .check(status.is(200))
      .check(jsonPath("$.id").saveAs("postId"))
    )
    .pause(1, 20)

    .exec(http("Get Comments for above post - 3rd call ")
      .get("/posts/${postId}/comments")
      .check(status.is(200))
    )
    .pause(2)

    .exec(http("Add Comment to the Post - 4th Call")
      .post("/posts/${postId}/comments")
      .body(StringBody(
        """{
            "name": "gatling",
            "email": "gatling@test.com",
            "body": "This is a simple comment"
          }"""))
      .check(status.is(201))
    )
```


```
  setUp(
    scn.inject(
      atOnceUsers(2),
      rampUsersPerSec(2) to(3) during(120 seconds) randomized
    )
```

```
 ).protocols(httpConf)
}
```

This script will make 4 different api calls to the endpoint "https://jsonplaceholder.typicode.com/" as per the defined load.

To learn more about Simulation class, ***click here****.*

Lets understand each part of the script.

Import statements

We added these import statements at the top of the script which are required for our script:

Extend Simulation

Next we extended our Scala class with the Gatling ***Simulation*** class. We must always extend from the Simulation class of the Gatling package, to make a Gatling script.

HTTP Configuration

The first thing that we do is setup the **HTTP configuration** for our Gatling script. The HTTP configuration for our class looks like this:

There are many options to configure in HTTP protocol and all are defined in the **documentation**.

Scenario Definition

The **Scenario Definition** is where we define our user journey in our Gatling script. These are the steps that the user will take when interacting with our application, for example:

- User will browse through some posts (Will call GetAllPosts API) and wait for 5 seconds
- User will open a random post (Will call a GetPost API) and wait for random seconds between 1 to 20
- User will check the comments for that post (Will call GetComments API) and wait for 2 seconds
- User will add comments of his own to this post (Will call PostComment API)

Execute Call : *In above snippet we are making 4 API calls. One to "posts", one to* "posts/RandomNumber" , one to retrieve comments of the posts "posts/${POSTID}/comments and one to add the comments POST "posts/${postid}/comments

Check Response Code: For each API calls we are checking the response code returned. If the response code does not match Gatling will throw error.

Pause Time: After every API call we see that we have added different type of pause interval. Firstly on line 25 we did **.pause(5)**- this will pause for 5 seconds. Then on line 32 we did **.pause(1,20)** - this will pause for a random time between 1 to 20 seconds. And on line 38 **.pause(2)** - This will pause for 2 seconds.

Extracting data from Response: On line 30 we have used jsonPath to extract the value of id and store it in variable "postId"

Generate Random Value: In code snippet we have defined a object reference of Random class as "postIdNumbers" and used on line 28 of the code snippet to use random postId.

Load Scenario

The final part of the Gatling script is the **Load Scenario**. This is where we set the load profile (such as the number of virtual users, how long to run for etc.) for our Gatling test. Each of the virtual users will execute the scenario that we defined above. Here, we are starting with **2 users** and **ramping** up the load from **2 to 3 users per second randomly** for **2 mins.**

Additional Ingestion options can be found **here**.

STEP 6: Executing your script

In Intellij terminal execute the following command:

C:\Ashish\Training\Gatling\test>mvn clean gatling:test -Dgatling.simulationClass=Simulations.TestFirstBasicAPIS

Gatling will run and should produce a report that will look like shown below which show some basic information of the test:

Gatling also generates HTML reports. The path is shown at the end of result. In our case it is **"C:\Ashish\Training\Gatling\test\target\gatling\testfirstbasicapisimulation-20210419134211657\index.html"**

Here we can see more detailed information about response times for each call. We can see 95% of users response time is less than 800ms. We can see min and max response times for

can read more about reports **here**.

We can also see different charts for overall and individual requests:

- Active Users along the Simulation
- Response Time Distribution
- Response Time Percentiles over Time (OK)
- Number of requests per second

STEP 7: Integrate with Jenkins

Assumption: Jenkins is already installed.

Install Plugin:

Login with the Jenkins admin user as we need to add gatling plugin in jenkins . Navigate to **"Manage Jenkins" -> "Manage Plugins"** and search for "**gatling**" plugin and install . (Restart Jenkins if required)

Create a Free Style Job:

Click on **"New Item"** and create a Free Style Project named **"gatling-perf-test"**

Configure Job

Configure the job with following:

- Workspace location (Here we are not using Source Code Management, but you can commit your project and give the GIT URL by selecting source code management as Git)
- Build Trigger
- Maven command to execute the script.
- Post Build action to track the Gatling simulation

Click on Save and then build the Job by clicking on "Build". Open the console and check for logs.

For every execution you should see a summary on the job with the mean time response details of each execution

This actually completes our integration setup for Gatling with Maven and Integration with Jenkins for performance testing. I hope you will find this article helpful.

Thanks for reading!! Keep Learning & Sharing!!!

Report this

54 · 1 Comment

Like Comment Share

Add a comment...

Mayur Sawant 1y

Nikhil Chaudhari

Like Reply

More articles by this author

How to Build a COVID-19 Dashboard with ELK stack.
Jun 1, 2020

JMeter Integration with Grafana+InfluxDB for Rea...
May 17, 2020