International Journal of Marketing and Technology
Vol. 8 Issue 5, May 2018,
ISSN: 2249-1058 Impact Factor: 6.559
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

CONFIGURE JMETER IN VAADIN APPLICATION

Sukhendu Mukherjee[*]

Abstract

This article describes how we can use JMeter for any vaadin application.JMeter may be used to test performance both on static and dynamic resources,Webdynamicapplications. It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types
Apache JMeter features include:

- Ability to load and performance test many different applications/server/protocol types:
    - Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, …)
    - SOAP / REST Webservices
    - FTP
    - Database via JDBC
    - LDAP
    - Message-oriented middleware (MOM) via JMS
    - Mail - SMTP(S), POP3(S) and IMAP(S)
    - Native commands or shell scripts
    - TCP
    - Java Objects

.

Keywords:

Vaadin;

JMeter;

Load Testing;

Performance Testing;

[*]**CTO Tiny Planet Inc**

1. Introduction

Vaadin is primarily a server-side framework. To test vaadin application performances we can use JMter for that.UsingJMeter we can do load test the application.It will help us to find out how the vaadin application will be performed in production with heavy load of data.

2. Research Method

As a part of load testing we took our existing vaadin application and configured accordingly.

Get the latest JMeter

Download JMeter from http://jmeter.apache.org/download_jmeter.cgi

Configure JMeter

Unzip the apache-jmeter-x.x.x.zip file.

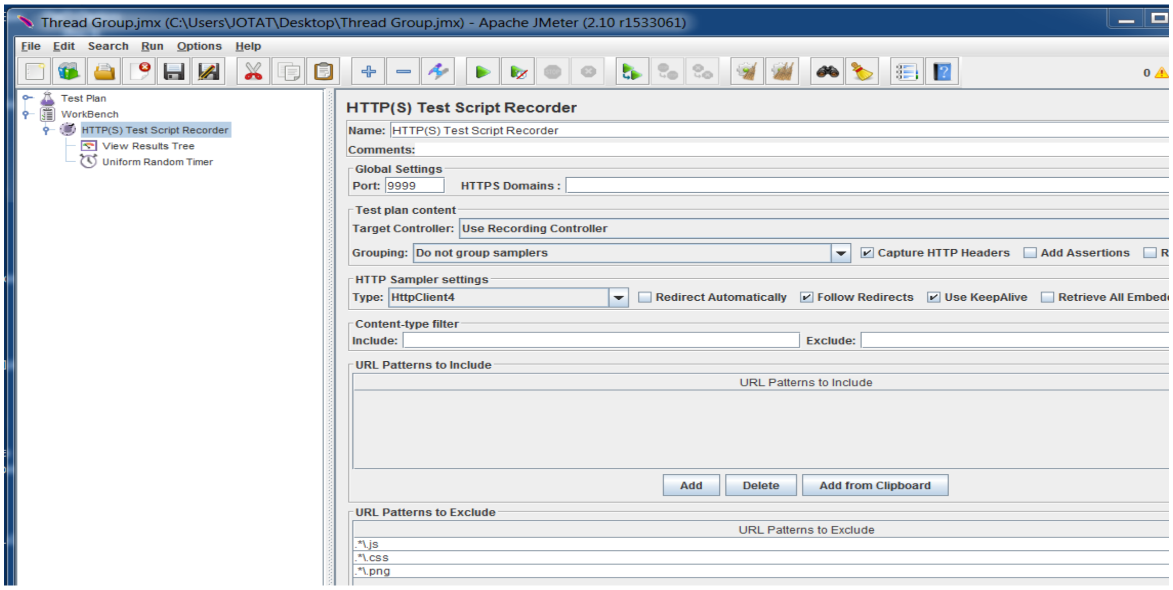
Edit JMETERHOME/bin/jmeter.bat (or jmeter.sh) and check that the JVM memory parameters are ok (e.g. set HEAP=-Xms512m -Xmx1500m -Xss128k). The maximum heap size (-Xmx) should be at least 1024m. I would also recommend that the thread stack size is set to 512k or below if a large number of threads is used in testing.

Start JMeter

E.g. double clicking jmeter.bat

Configure Test Plan and WorkBench

Right click the WorkBench icon and select 'Add' → 'Non-Test Elements' → 'HTTP(S) Test Script Recorder'. Edit the Recorder parameters and set Port: to e.g. 9999 (you could leave it to 8080 if your web application servers do not use the port 8080). Typically requests related to loading static web content like css files and images are excluded from the load test. You can use 'Url Patterns to Exclude' section of the recorder to define what content is excluded. For instance to exclude css files add following pattern: .*\.css


Thread Group.jmx (C:\Users\JOTAT\Desktop\Thread Group.jmx) - Apache JMeter (2.10 r1533061)
File Edit Search Run Options Help
Test Plan
WorkBench
HTTP(S) Test Script Recorder
View Results Tree
Uniform Random Timer
Add
Delete
Add from Clipboard

HTTP(S) Test Script Recorder

Name: HTTP(S) Test Script Recorder

Comments:

Global Settings

Port: 9999 HTTPS Domains :

Test plan content

Target Controller: Use Recording Controller

Grouping: Do not group samplers ✓ Capture HTTP Headers ☐ Add Assertions ☐ R

HTTP Sampler settings

Type: HttpClient4 ☐ Redirect Automatically ✓ Follow Redirects ✓ Use KeepAlive ☐ Retrieve All Embed

Content-type filter

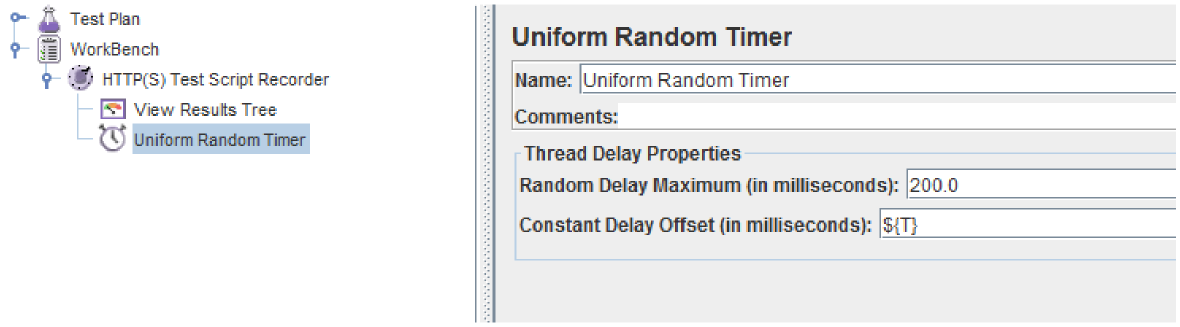
Include: Exclude:

URL Patterns to Include

URL Patterns to Exclude

.*\.js
.*\.css
.*\.png

Right click the Recorder icon and select 'Add' → 'Timer' → 'Uniform random timer'. Configure timer by setting the **Constant Delays Offset into ${T}**. This setting means that JMeter records also the delays between the http requests. You could also test without the timer but with the timer your test is more realistic.



Uniform Random Timer

Name: Uniform Random Timer

Comments:

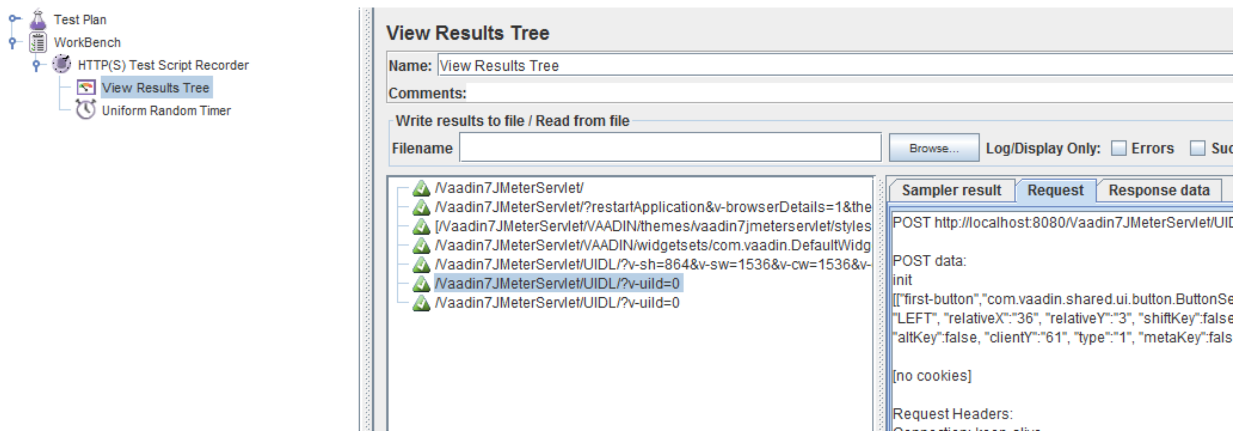
Thread Delay Properties

Random Delay Maximum (in milliseconds): 200.0

Constant Delay Offset (in milliseconds): ${T}

Optionally we could also add 'View Result Tree' listener under the Recorder. With 'View Result Tree' listener it is possible to inspect every recorded request and response.

Note since JMeter you can do this in one step using menu item "Templates…" and selecting "Recording" template.



View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Suc

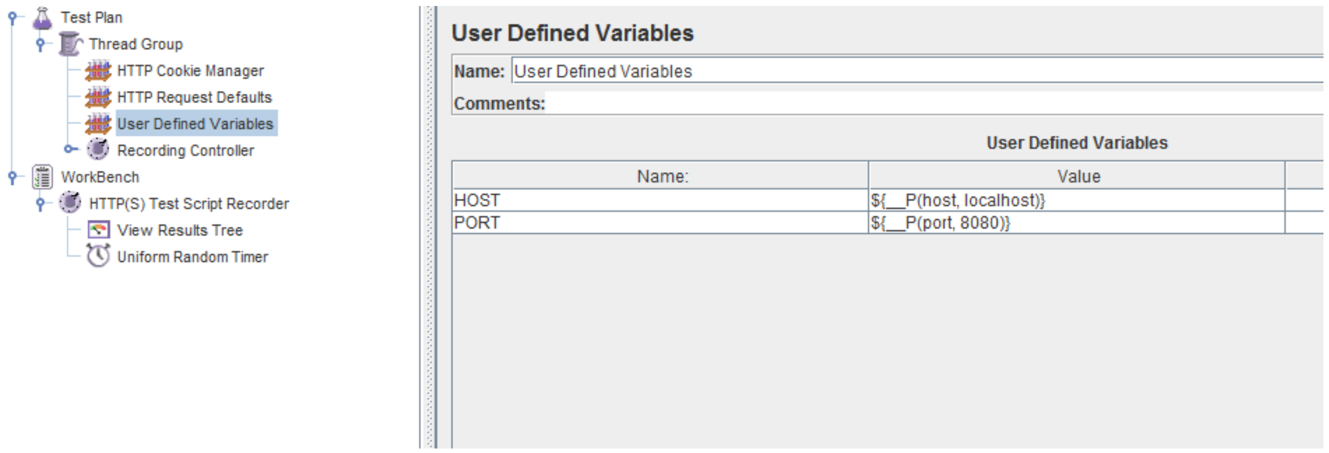
```
/Vaadin7JMeterServlet/
/Vaadin7JMeterServlet/?restartApplication&v-browserDetails=1&the
[/Vaadin7JMeterServlet/VAADIN/themes/vaadin7jmeterservlet/styles
/Vaadin7JMeterServlet/VAADIN/widgetsets/com.vaadin.DefaultWidg
/Vaadin7JMeterServlet/UIDL/?v-sh=864&v-sw=1536&v-cw=1536&v-
/Vaadin7JMeterServlet/UIDL/?v-uiId=0
/Vaadin7JMeterServlet/UIDL/?v-uiId=0
```

```
POST http://localhost:8080/Vaadin7JMeterServlet/UIE

POST data:
init
[["first-button","com.vaadin.shared.ui.button.ButtonSe
"LEFT", "relativeX":"36", "relativeY":"3", "shiftKey":false
"altKey":false, "clientY":"61", "type":"1", "metaKey":fals

[no cookies]

Request Headers:
```

Next, configure the Test Plan. Add a 'Thread Group' to it and then add a 'Config Element' → 'HTTP Cookie Manager' to the thread group. Set Cookie policy of the cookie manager to be 'compatibility'. **Remember also to set the "Clear cookies each iteration" setting to 'checked'**. Add also a 'Config Element' → 'HTTP Request Defaults' into Thread group.

We could also add a 'Config Element' → 'User Defined Variables' and a 'Logic Controller' → 'Recording Controller' into Thread Group.

JMeter should now looks something like the screenshot below:



User Defined Variables

Name: User Defined Variables

Comments:

User Defined Variables

Name:	Value
HOST	${__P(host, localhost)}
PORT	${__P(port, 8080)}

Configure Vaadin application

Disable the xsrf-protection

In Vaadin have to disable the xsrf-protection of your application or otherwise the JMeter test may fail.

If we use web.xml in your Vaadin 7 project, add the following context-parameter in the web.xml or optionally add it as an init parameter just like in the Vaadin 6 project below.

```
<context-param>
<param-name>disable-xsrf-protection</param-name>
```

```
<param-value>true</param-value>
</context-param>
```

If we use annotation based (Servlet 3.0) Vaadin servlet configuration, you can currently (in Vaadin 7.1.11) either fall back to Servlet 2.4 web.xml configuration or set the parameter value for 'disable-xsrf-protection' as a java.lang.System property before the Vaadin'sDefaultDeploymentConfiguration is loaded. This can be done for example by extending VaadinServlet class. At the end of this Wiki article there is an example servlet (JMeterServlet) that implements this functionality. See the example code below for how to replace the default VaadinServlet with your custom VaadinServlet in the UI class.

```
public class YourUI extends UI {

@WebServlet(value = "/*", asyncSupported = true)
@VaadinServletConfiguration(productionMode = false, ui = YourUI.class)
public static class Servlet extends JMeterServlet {
  }

  @Override
protected void init(VaadinRequest request) {
   //...
  }
```

Important! Remember to enable the protection after the testing is done!

Disabling syncId happens with similar parameter

```
<context-param>
<param-name>syncId</param-name>
<param-value>false</param-value>
</context-param>
```

If we want to do the above with Java Servlet 3.0 annotations, use the following:

```
initParams = {
@WebInitParam(name = "disable-xsrf-protection", value = "true"),
@WebInitParam(name = "syncIdCheck", value = "false")}
```

Use debug id:s within your Vaadin application

Normally a Vaadin application sets a sequential id for each user interface component of the application. These ids are used in the ajax-requests when the component state is synchronized between the server and the client side. The aforementioned id sequence is likely the same between different runs of the application, but this is not guaranteed.

There no more exists a setDebugId() method; instead there is setId() method. Unfortunately this method won't set component ids used in the ajax-request. Therefore, by default, JMeter tests of a Vaadin 7 application are not stable to UI changes. To overcome this problem you can use our JMeterServlet (see the end of this article) instead of the default VaadinServlet. When using the JMeterServlet component ids are again used in the ajax requests. See example above for how to replace default VaadinServlet with JMeterServlet. For additional information, see the Vaadin ticket #13396.

Use named windows in your application

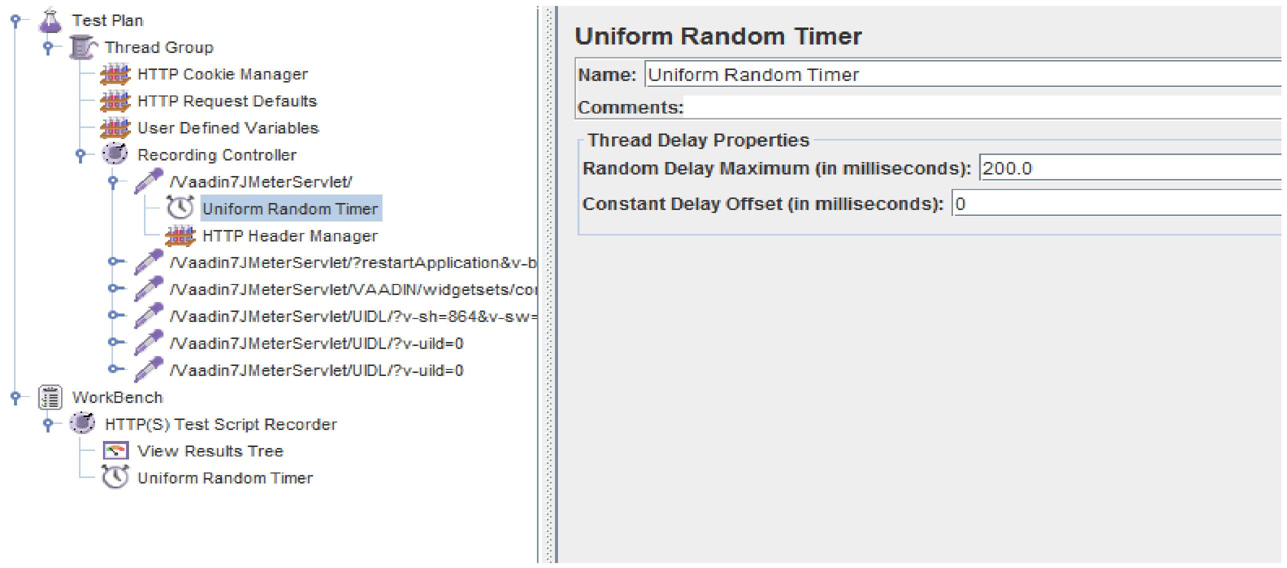
Setting the name for the Windows **in the Vaadin (< 6.4.X)** application is important since otherwise these names are randomly generated. Window name could be set using the setName()-method.

Configure your browser

Since JMeter is used as a proxy server, you have to configure the proxy settings of your browser. You can find the proxy settings of Firefox from Tools → Options → Connections → Settings: 'Manual proxy configuration'. Set the correct IP of your computer (or 'localhost' string) and the same port that you set into proxy server settings above.

Start recording

Start web application server. Start the proxy server from the JMeter. Open the URL of your web application into the browser configured above. You should append ?restartApplication to the URL used when recording the tests to make sure that the UI gets initialized properly. Thus the URL becomes something like (http://localhost:8080/test/TestApplication/?restartApplication). If everything is ok your web application opens normally and you can see how the different HTTP requests appear into JMeter's thread group (see screenshot below). When you have done the recording, stop the proxy server.



Test Plan
Thread Group
HTTP Cookie Manager
HTTP Request Defaults
User Defined Variables
Recording Controller
/Vaadin7JMeterServlet/
Uniform Random Timer
HTTP Header Manager
/Vaadin7JMeterServlet/?restartApplication&v-b
/Vaadin7JMeterServlet/VAADIN/widgetsets/coı
/Vaadin7JMeterServlet/UIDL/?v-sh=864&v-sw=
/Vaadin7JMeterServlet/UIDL/?v-uiId=0
/Vaadin7JMeterServlet/UIDL/?v-uiId=0
WorkBench
HTTP(S) Test Script Recorder
View Results Tree
Uniform Random Timer

Uniform Random Timer
Name: Uniform Random Timer
Comments:
Thread Delay Properties
Random Delay Maximum (in milliseconds): 200.0
Constant Delay Offset (in milliseconds): 0

Performance testing

Clean up the recorded request

Before you start the test, you may have to delete the first timer object which is located below the first HTTP request in the thread group since its time delay may be unrealistically big (see the screenshot above). **It is also very much recommended to check the recorded data and delete all unessential requests.**

Detecting Out of Sync errors

If test results in the application being in an Out of Sync error state it is not by default detected by JMeter (because the response code is still HTTP/1.1 200 OK). To make an assertion for detecting this kind of error you should add a Response Assertion to your test plan. Right-click on the thread group and select Add → Assertions → Response Assertion. Configure the assertion to assert that the Text Response does NOT contain a pattern "Out of sync".

Optional parameterization of the request

Sometimes, it is useful to parameterize the recorded requests. Parameterization of a request is easily done in JMeter:

1. add a "User Defined Variables"-element into the first place of your Test Plan.
2. Copy paste the whole parameter value of wanted UIDL-request into the newly made variable (e.g. PARAM1).
3. Replace the value of the UIDL-request with the parameter reference (e.g. ${PARAM1}).

Start testing

Now, it is time to do the actual testing. Configure the thread group with proper 'Number of Threads' (e.g. 100) and set also the 'Ramp-Up Period' to some realistic value (e.g. 120). Then, add e.g. 'Listener' → 'Graph Results' to monitor how your application is performing. Finally, start the test from the Run → Start.

Stop on Error

When you are pushing your Vaadin application to the limits, you might get into a situation where some of the UIDL requests fail. Because of the server-driven nature of Vaadin, it's likely that subsequent requests will cause errors like "*Warning: Ignoring variable change for non-existent component*", as the state stored on the server-side is no longer in sync with the JMeter test script. In these cases, it's often best to configure your JMeter thread group to stop the thread on sampler error. However, if you have configured your test to loop, you might want to still continue (and ignore the errors), if the next iteration will start all over again with fresh state.

3. Results and Analysis

As a result we will be able to setup for any vaadin application that will help users to measure application scalability and sustainability.Users can check the output of the load test and report also.

4. Conclusion

Implementing JMeter for vaadin application to get load testing and scalability report for application to how the application will run in production.