**What is performance testing?**

Performance testing is a non-functional type of testing which is used to evaluate application under test (AUT) performance in terms of its responsiveness as well as behavior of AUT under various workload.

A slow website results into a bad user experience and have negative financial impact. Even delay of second, for longer period of time, may result into huge revenue loss. Thus, performance test of websites comes into picture. To performance test a website there are tools available, for example: JMeter, LoadRunner, WebLoad, LoadView, NeoLoad and many more.

**Importance of Performance Testing:**

* Helps in evaluating potential bottlenecks of an AUT.
* Slowness of an application or web service can be evaluated under heavy load.
* We can find out how many parallel users an AUT can handle.
* Helps in finding out impact of changes in each release in terms of performance.

**What is JMeter?**

Apache JMeter is an open source, pure Java platform software which is designed to load test functional behavior and measure performance.

**Key features of JMeter includes:**

**License:** Since JMeter is open source, it is free and easily available.

**Graphical User Interface:** Simple, user friendly and easy to learn as compared to other performance testing tools

**Server/ Protocol Support:** JMeter has ability to load and performance test different applications/server/protocols. A few protocols includes HTTP, HTTPS, FTP, SOAP/REST, Database via JDBS, LDAP, JMS, SMTP(S), POP(3) and IMAP(S), Native Commands/ Shell Scripts and TCP.

**Platform:** JMeter is pure java software. Therefore, it is platform independent and supports all environment.

**Simulation:** Simulate multiple users by using virtual users or unique users in order to generate heavy traffic on web server or services.

**Supports Distributed Testing:** It has master slave for distributed testing where master will distribute tests among all slaves and slaves will execute scripts against your server.

**Test Result Visualization:** Test result can b view in different formats like Graph, Table, Tree, and Report, etc.

**Reporting:** By default, JMeter provides XML and CVS Report Formats only. We can use JMeter and ANT together to obtain HTML report as per the requirement.

**Testing Types:** Apart from just Performance, Load, Stress testing JMeter works well for Functional, Regression and Soak/Endurance testing too.

**Record and Playback:** Record user scenario/action in Firefox Browser and play scripts

**Framework:** Multi-threading framework allows concurrent and simultaneous sampling of different functions by many or separate thread groups.

**Installation:** No Complex installation required – Just run JMeter.bat on windows / run Jmeter.sh on Linux

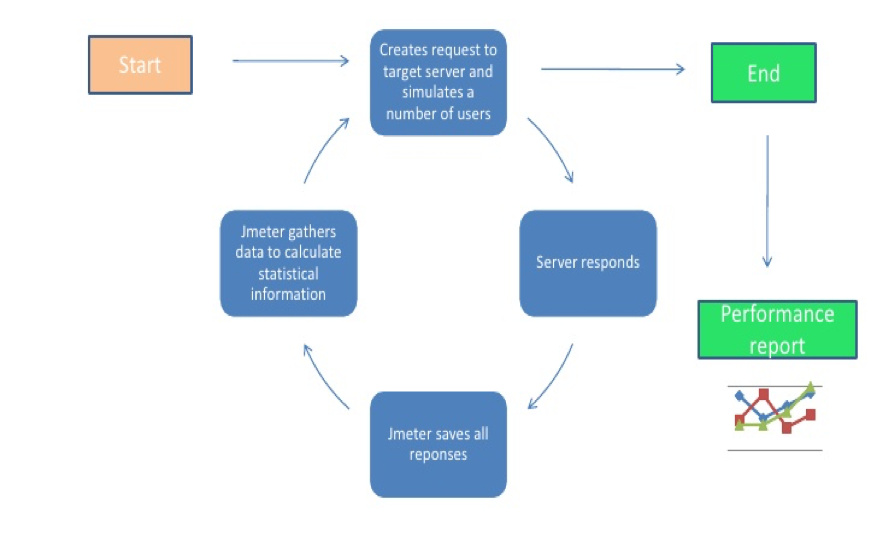
**Knowledge:** JMeter does not require extensive programming knowledge. Only prior knowledge of Java language is preferred.

**JMeter Workflow**

When we start load or perform test of an application, JMeter creates requests to target server and simulates number of users sending requests to the target server.

As soon as server starts responding to the requests, JMeter starts saving all the responses. On the basis of data/response JMeter gathers data to calculate statistical information.

Finally, using this statistical information JMeter prepares a report which tells about performance of the AUT.



**Pre-Requisites**

**Software Requirements:**

Since JMeter is pure Java software, Java should be on your machine. JMeter 5.0 supports only java 8 and higher versions. It does not support any prior version of java 8.

**Operating System Requirements:**

If your Operating system supports java, then JMeter should run correctly on your system. Below mentioned operating systems and the file required to launch JMeter on the respective operating systems.

Windows: jmeter.bat (Launch JMeter on windows)

MAC: jmeter.sh (Launch JMeter on MAC)

Linux: jmeter.sh (Launch JMeter On Linux)

**Installation:**

1. Go to Apache JMeter’s website to download JMeter.

<https://jmeter.apache.org/download_jmeter.cgi>

1. Click apache-jmeter-5.1.zip for windows
2. Once download is complete, Unzip the File and save it your preferred location.

Bin folder contains templates, .bat, .sh, .jar files to start JMeter. It also contains User & JMeter’s properties files.

Lib folder contains all required jar files.

jmeter.bat is required to launch jmeter GUI on windows

jmeter.sh is required to launch jmeter GUI on MAC and Linux

**HTTP(S) Test Script Recorder (was: HTTP Proxy Server ):**

The HTTP(S) Test Script Recorder allows JMeter to intercept and record your actions while you browse your web application with your normal browser.

JMeter will create test sample objects and store them directly into your test plan.

**Test Script Recorder Settings:**

Let’s now examine the most interesting Script Recorder Settings:

**Global Settings:**

Port: HTTP and HTTPS proxy port. The proxy will be running on this TCP port, listening for http requests,

HTTPS Domains: List of domain (or host) names for HTTPS. Use this to pre-generate certificates for all servers you wish to record. Example: httpbin.org

**Test Plan Content:**

Target Controller: by default, everything is recorded within Recording Controller. But you can change it to whatever you want (like the Thread Group directly),

Grouping: groups http request by trying to determine which requests belong to the same page,

Capture HTTP Headers: creates HTTP Headers Manager as http request child, capturing the headers sent by the web browser. Leave checked as most http headers are important.

**HTTP Sampler Settings:**

Use Keep-Alive: enable HTTP Keep-Alive to signal that the connection should be kept open for further messages (by default for HTTP 1.1). Most sites are HTTP 1.1 compliant now,

Follow Redirects: generates only a single HTTP Request when multiple HTTP redirections are detected, and enables Follow Redirects on it.

Let’s start JMeter’s Proxy Recorder:

Select HTTP(s) Test Script Recorder within the tree on the left,

Click on Start button: a dialog should show up asking to install CA Root Certificate. This certificate must be installed within the web browser to support recording HTTPs traffic.

By default, JMeter generates a Root CA valid only 7 days. That’s annoying since we need to redo the procedures below every week. But there is a way to hack this.

Open <JMETER\_HOME>/bin/jmeter.properties and locate the following lines:

# The default validity for certificates created by JMeter

#proxy.cert.validity=7

For example, to have certificates valid for a year, change it to:

# The default validity for certificates created by JMeter

proxy.cert.validity=365

Remember to:

Stop JMeter,

Change the jmeter.properties,

Delete any existing <JMETER\_HOME> / bin / ApacheJMeterTemporaryRootCA.crt,

And restart JMeter.

That should save you from repeating the browser installation procedures below too often.

**Proxy Setup:**

To configure JMeter’s Proxy in Firefox:

* Open Firefox
* Open main menu and select Options
* Select General section, then scroll-down to Network Settings and click on Settings button
* Select 'Manual Proxy Configuration' radio button and enter both http and https proxies with hostname localhost and port 8888 (default proxy port).
* Click on Ok.

Firefox is now configured with JMeter’s proxy settings and ready to record http and https requests.

**Recording the Scenario:**

* Select Target Controller 'Test Plan > HTTP(S) Test Script Recorder'
* Click on the Start button within the HTTP(S) Test Script Recorder to start JMeter’s proxy. Click on OK when the Root CA dialog appears.
* Now browse to AUT and You should see requests filling in within the HTTP(S) Test Script Recorder.

**HTTP Cookie Manager:**

HTTP Cookie Manager is used to store cookies which targeted server sends in the response of your Http request. The saved cookies can be used in other samplers/requests of the Test Plan.

**What is Thread Group?**

A Thread Group is a set of threads executing the same scenario. Set the number of iterations in the configuration. Thread behavior is defined according to ramp up and destroyed once the number of iterations per thread has elapsed.

A number of threads(users) can be defined in a Thread Group. Each thread simulates a real user requesting to the server under a test.

If you set the number of threads as 20; JMeter will create and simulate 20 virtual users during the load test.

To create a Thread Group,

Right Click on Test Plan and then go to **Add >> Threads >> Thread Group**.

**Components of Thread Group:**

The Thread Group Panel holds the following components.

**1.** **Action to be taken after a Sampler Error**

If JMeter catch any sampler error during test execution, you can tell it how to react in that scenario from the following available options.

**Continue:** to ignore error and move to the next element in the tree

**Start Next Thread Loop:** to stop current Thread and Start Next

**Stop Thread:** to stop the current Thread execution.

**Stop Test:** to stop entire test execution.

**Stop Test Now:** the entire test will be stopped abruptly.

Default is Continue.

**2. Thread Properties**

**Number of Threads (users):** Simulates the number of users or connections to your server application.

**Ramp-Up Period (in seconds):** Tells JMeter how long to take ramp-up to the full number of threads chosen. For Example: If you set “Number of Threads” to “20”, and “Ramp-Up Period to “40” seconds, then JMeter will wait till 40 seconds to make all threads up and running. That means each thread will start 2 seconds late after the previous thread was started.

Formula: Ramp-Up Period / Number of Threads i.e. 40 / 20 = 2 (seconds)

**Loop Count:** the number of times the test to be executed. If you need to run the test forever, then select the “Forever” check box.

**Scheduler:** To schedule test execution. Scheduler Configuration bottom panel will get enabled when you select this checkbox. The schedule feature is also very helpful in soak/endurance testing.

**3. Scheduler Configuration**

You can configure test start time, end time, duration and start up delay of your load test plan using Scheduler Configuration section. To enable this area of configuration, Scheduler check box must be selected from the above Section of Thread Properties.

**Start Time:** This plans the test to start at scheduled time. Pre-Condition is that the JMeter should be running on given date and time in “Start Time” field.

**End Time:** This command JMeter to end the test at the mentioned time. End time override and stop execution in between. Means End Time is maximum allowed time to finish execution of the test plan. JMeter ends the execution immediately as soon as End Time is occurred.

**Duration (seconds):** This tell the JMeter to execute the test for the specific duration of time. If the duration is set to 60 secs, JMeter will keep the execution on for 60 secs and ends it once the time is elapsed. It also ignores or override the End Time and All threads has completed its test or not.

**Startup delay (seconds):** This tells JMeter to wait for specified time before starting the test. If the Startup time is set to 10 secs, JMeter will not start loading the Users till the time 10 secs are over.

**Stepping Thread Group:**

* Preview graph showing estimated load
* Initial thread group delay to combine several thread group activities
* Increase load by portions of threads (users) with ramp-up period
* Configurable hold time after all threads started
* Decrease load by portions

**Example:**

* Total Thread count should be 500.
* The Test should begin with delay of 2 seconds.
* Then kick start 10 threads immediately.
* Now add 50 threads every 5 seconds with a ramp-up of 10 seconds until all the threads (i.e 500 users) are up and running.
* After reaching 500 threads, all of them should continue running for 180 seconds.
* At the end, ramp-down should be achieved with 20 threads every 20 seconds.

**Concurrency Thread Group:**

- 100 threads

- 20 minutes Ramp Up Time

- 10 Ramp-Up Steps

- 30 minutes holding the target rate

This means that:

Every 2 minutes 10 users will be added until we reach 100 users.

(20 minutes divided by 10 steps equals 2 minutes per step. 100 users divided by 10 steps equals 10 users per step. Totaling 10 users for every 2 minutes).

After reaching 100 threads, all of them will continue running and hitting the server together for 30 minutes.

**Listeners:**

Listeners are the JMeter component that displays test results.

Listener allows us to view and analyze the Sampler request and response in the form of tables, graphs, trees etc. You can also save results in a file and read saved results later.

You can access the sampler result in an ongoing load/stress test without waiting till end.

The variety of JMeter listeners enables to monitor the requests sent from JMeter as well as to analyze the responses received from the system under test.

Listeners aggregate time and capacity related information that JMeter collects from requests and responses, or deal with statistical information like distribution and percentage.

All Listeners can be found by clicking Add->Listeners. A JMeter Listener will only collect data from JMeter elements at or above its level.

If a listener is added to the script as a child element, it will only show the data related to its parent.

If a listener is added under a thread group of a script that has a few thread groups, that listener will display the data of all the samplers that belong to that thread group.

If you need to review reports of all the samplers in a script, place the listener at the same level of all thread groups in the script.

**View Results Tree**

The View Results Tree listener displays samples that the JMeter samplers generate, and the assertion results that are related to these samples.

This listener displays the samples in the order they are generated by the JMeter script, and provides parameters and data for each of them.

View Results Tree listener provides the request parameters, response parameters and the response data. This is displayed under the corresponding tabs: sampler result, request, and response data.

The Sampler result tab contains the response code, headers, cookies and information about time, latency, response size in bytes - separately for the headers, the body and the error count.

The Request tab contains information about headers that the JMeter script added, the URL, the HTTP method and cookie information.

The Response data tab contains the response body of the sample.

One of the most useful elements of the View Results Tree listener is a tab that renders information transmitted in the sample body into the required format:

simple text, HTML, or XML. It also filters substrings, by using RegExp patterns, specific XPath, JSON Path or CSS.

This helps constructing and validating the following corresponding PostProcessors in the script.

**View Results in Table**

The View Result in Table listener displays information about each sample in the form of a table. The table shows time related data for each sample, the thread number and the sample execution result.

Results in the table are in the order the time samples were issued. This listener is useful for getting a quick estimation of the behaviour of the system under test.

**Response Time Graph**

An additional listener that plots a graph of the samples’ response time over the course of the test is the Response Time Graph listener.

This listener is able to plot a graph either for all samples or for selected ones.

**Aggregate Graph**

Aggregate graphs allow us to generate bar graphs easily and let us select graph display settings. The generated graph and table data can be saved explicitly in the form of PNG and CSV.

**Aggregate Report**

Aggregate report shows a separate table row for each differently named sampler request in the test.

**Assertion Results**

Assertion Results displays the results of assertions applied on the Sampler.

Note that Assertion Results SHOULD NOT BE USED during Load/Stress test as it consumes a lot of resources like memory and CPU.

It is recommended to use it with functional testing or debugging purposes.

**Session# 2**

**Logic Controllers:**

Logic Controllers handle the order of processing Samplers/Requests in a Thread.

Logic Controllers determine the order in which user request is executed.

Logic Controllers will decide “When & How” to send a request to a web server.

**1. Simple Controller:**

Simple Controller doesn’t provide any functionality and it is just a container that contains user request.

**2. Transaction Controller:**

The Transaction Controller generates an additional sample which measures the overall time taken to perform the nested test elements.

The generated sample time includes all the times for the nested samplers excluding by default (since 2.11) timers and processing time of pre/post processors unless checkbox "Include duration of timer and pre-post processors in generated sample" is checked.

The generated sample is only regarded as successful if all its sub-samples are successful.

**Generate Parent Sample:** If checked, then the sample is generated as a parent of the other samples, otherwise the sample is generated as an independent sample.

**Include duration of timer and pre-post processors in generated sample:** Whether to include timer, pre- and post-processing delays in the generated sample. Default is false.

**3. Interleave Controller:**

JMeter will alternate among each of the requests for each loop iteration.

It picks up and makes one of user request run in each loop of the thread.

Let’s consider an example of Interleave Controller, where Thread Group contains “Number of Threads” to “1”, “Loop Count” to 5, and a total of “4” Samplers (Http requests).

Total “5” requests will be sent to web server under test by using Interleave Controller in sequential order i.e. 1 request per loop.

**4. Runtime Controller:**

Runtime Controller controls the execution of its samplers/requests for the given time.

For example, if we have specified Runtime(seconds) to “20”, JMeter will run test for 20 seconds only.

**5. Random Controller:**

Random Controller provides functionality to run user request in random order for each loop i.e. One random user request in each loop.

Random Controller is similar to “Interleave Controller” except that it makes samplers/requests to run in random order in each loop.

For Example, you have defined “4” Http requests under “Random Controller”, then these requests will be sent in random order and their order of execution will be decided at rum-time.

**6. Random Order Controller:**

Random Order controller is similar to “Simple Controller”, it will make all samplers to run in random order in each thread loop.

Let’s consider an example of Random Order Controller, where Thread Group contains “Number of Threads” to “1”, “Loop Count” to 5, and a total of “4” Samplers (Http requests).

A total of 1 \* 5 \* 4 = “20” requests will be sent to the web server under test. The order of execution will be random.

**7. Recording Controller:**

Recording Controller is a place holder where proxy server can save recorded requests. It has no effects on test execution.

**8. If Controller:**

The If Controller allows the user to control whether the test elements below it (its children) are run or not.

By default, the condition is evaluated only once on initial entry, but you have the option to have it evaluated for every runnable element contained in the controller.

If Controller will internally use javascript to evaluate the condition but this can have a performance penalty.

A better option (default one) is to check Interpret Condition as Variable Expression?, then in the condition field you have 2 options:

Option 1 : Use a variable that contains true or false

If you want to test if last sample was successful, you can use ${JMeterThread.last\_sample\_ok}

Option 2 : Use a function (${\_\_jexl3()} is advised) to evaluate an expression that must return true or false.

For example, ${\_\_jexl3(${UserName} == "nkasu@xtglobal.com")}

this would be evaluated as true/false, the result would then be passed to JavaScript which would then return true/false.

If the Variable Expression option is selected, then the expression is evaluated and compared with "true", without needing to use JavaScript.

**9. Loop Controller:**

JMeter will loop through them a certain number of times, in addition to the loop value you specified for the Thread Group.

For Example, Thread Group loop count is set to “2”, Loop Controller loop count is set to “2”, and “3” requests are present under Loop Controller node,

then JMeter will send a total of “12” Http Requests to the web server under test.

* Select Forever checkbox to run tests for infinite times
* Enter loop count number to run tests for a fixed number of times

**Assertion:**

Assertion in JMeter is used to validate response of the request, that you have sent to the server.

Assertion is a process where you verify expected result with the actual result of the request at run time. If you need to apply assertion on a particular Sampler, then add it as a child of that Sampler.

You can view assertion results by adding “Assertion Listener” to the Thread Group. Failed assertions will be displayed in other listeners as well.

**1. Response Assertion:**

Response Assertion can be used to add and compare pattern strings against one or many values of server response.

For Example, when you send a request to the URL: https://www.google.com and get the server response. Here, you can verify this response by using Response Assertion.

You can insert “<title>Google</title>” as a “Pattern to Test” field value in Response Assertion. If response doesn’t contain this string, it will fail the sampler.

* Add the pattern string according to your expected response

**2. Size Assertion:**

Size Assertion is used to verify the server response contains expected number of Bytes or not.

* Select Response Size Field to Test
* Enter expected response size in bytes
* Choose type of comparison

**3.Duration Assertion:**

Duration Assertion is used to verify that server response is received within a specified time constraint or not.

It will mark the sample request as failed, if response takes longer time duration than the given amount of time.

* Enter duration period of response in milliseconds.

**4. HTML Assertion:**

HTML Assertion is used to verify that the response contains correct HTML syntax or not.

It means the response data must be met the HTML syntax. It will fail the test in case of improper HTML syntax response.

**Timers:**

JMeter sends requests without applying any delay between each sampler/request. If you perform load/stress testing on your server without any delay, it will be overloaded. Then, it won’t be able to give you realistic results and fail to simulate real world user traffic experience.

JMeter Timers are the solution to all these problems. Timer element can be added in a test plan to apply wait between each sampler/request.

**1. Constant Timer:**

Constant Timer element delays each request in a Thread Group for the same amount of time.

Test Action sampler is similar to Constant Timer.

**2. Uniform Random Timer:**

Uniform Random Timer element is used to delay each request for a random period of time.

Total amount of delay = Random Delay Maximum + Constant Delay Offset

* Enter any Maximum Random Delay value in milliseconds.
* Enter constant Delay Offset value in milliseconds in addition to random delay.

**3. Gaussian Random Timer:**

Gaussian Random Timer element is used to delay each user request for a random period of time.

It has a random deviation around the Constant Delay Offset based on Gaussian curve distribution.

**4. Constant Throughput Timer:**

A Constant Throughput Timer can pause the threads in order that a goal-oriented scenario could be implemented.

Using constant throughput timer, you can decide how many samples should be executed per minute.

Constant throughput timer will add random pauses between requests during test execution to match required throughput figure (samples per minute).

**Session# 3:**