Introduction to Performance Testing

Performance Testing : Performance testing is a practice of evaluating how a system performs in terms of responsiveness and stability under a particular workload.

Apache JMeter

Jmeter is used to perform the performance testing, functional testing and load testing of Web applications.

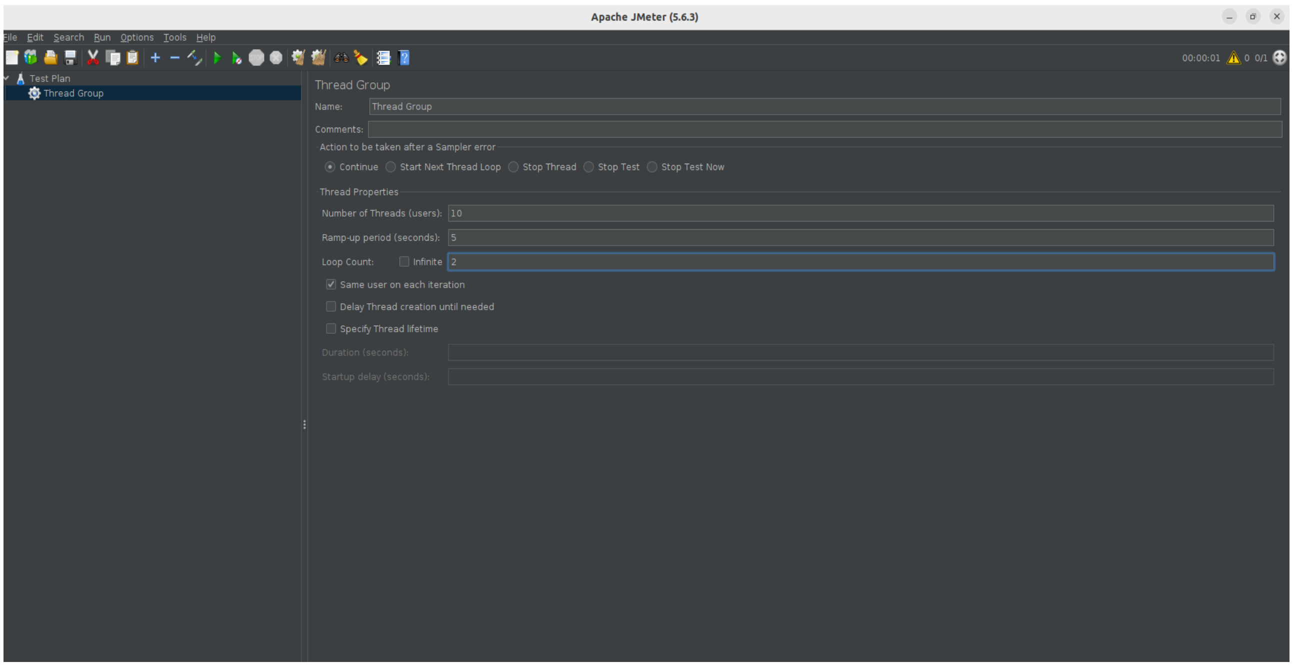
Test Plan

A test plan describes a series of steps Jmeter will execute when run. A complete test plan consists of one or more Thread Group , logic controllers , sample generating controllers , listeners , timers , assertions and configuration elements .

Purpose of each component added in the Test Plan :

1. **Thread Group**
   Thread group elements are the begining points of any Test plan . All controllers and samplers must be under a thread group. Other elements ex. Listeners may be placed directly under the test plan , in which they will apply to all thread groups. As the name implies , the thread group element controls the number of threads Jmeter will use to execute the test .

   The controls for a thread group allows us to :
   - Set the number of threads
   - Set the ramp-up period
   - Set the number of times to execute the test



Thread Group
Name: Thread Group
Comments:
Action to be taken after a Sampler error
[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now
Thread Properties
Number of Threads (users): 10
Ramp-up period (seconds): 5
Loop Count: Infinite 2
[✓ Same user on each iteration]
Delay Thread creation until needed
Specify Thread lifetime
Duration (seconds):
Startup delay (seconds):

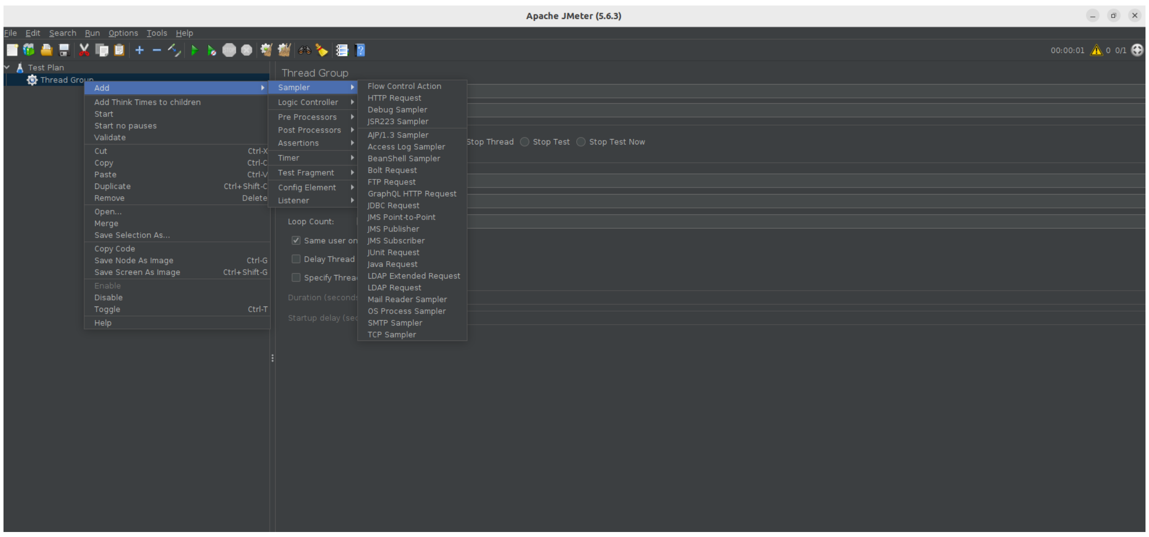
- Each thread will execute the test plan in its entirely and completely indepnedently of other test plan.
- The Ramp-up period tells Jmeter how long to take to ramp-up to the full number of threads chosen.

2. Controllers

Jmeter has two types of controllers

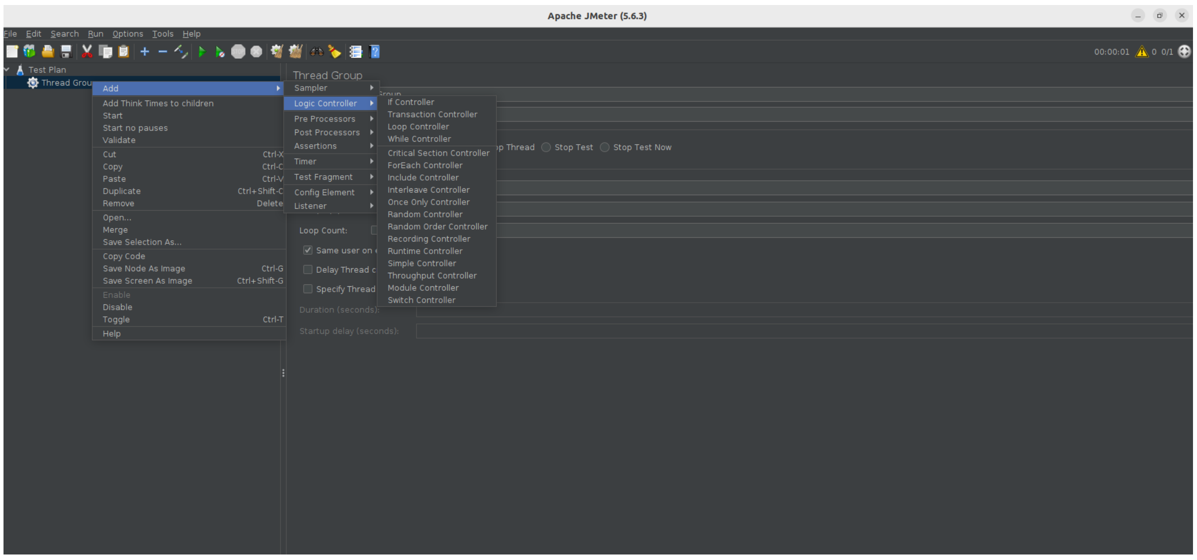
- **Samplers**

  Samplers tell Jmeter to send requests to a server and wait for the a response. They are processed in the order they appear in the tree . Controllers can be used to modify the number of repetitions of a sampler.



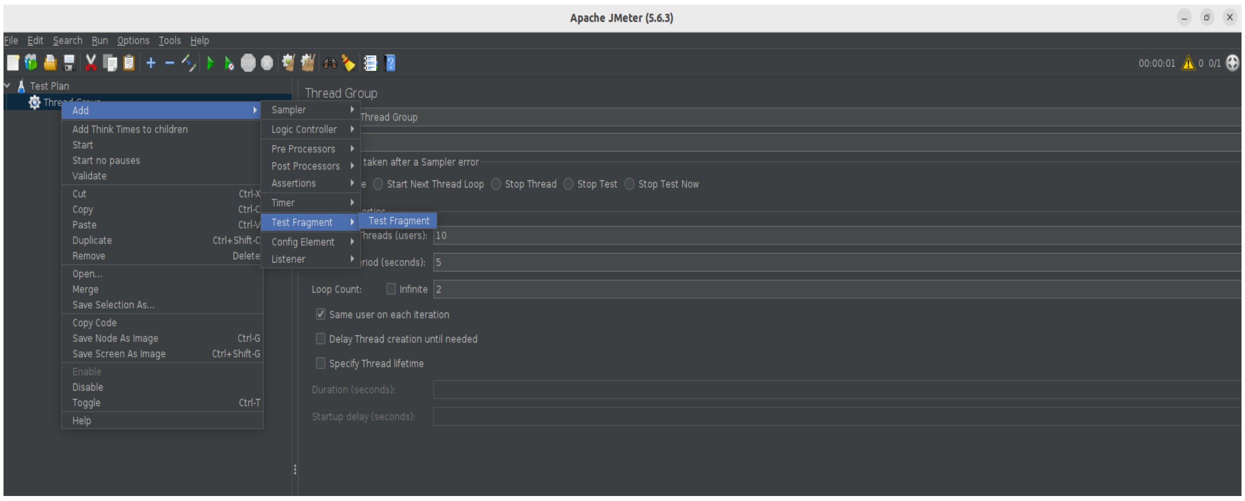
- **Logic Controllers**

  Logic controllers let us customize the logic that jmeter uses to decide when to send requests. Logic controllers can change the order of requests coming from their child elements. They can modify the requests themselves , cause Jmeter to repeat requests , etc.



3. Test Fragments

The test fragment element is a special type of controller that exist on the test plan tree at the same level as the Thread group element. It is distinguished from a Thread group in that it is not executed unless it is referenced by either a Module controller and Include controller . This element is purely for code re-use with in Test Plans.



Thread Group

Thread Group

taken after a Sampler error

Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

hreads (users): 10

riod (seconds): 5

Loop Count: ☐ Infinite 2

☑ Same user on each iteration

☐ Delay Thread creation until needed

☐ Specify Thread lifetime

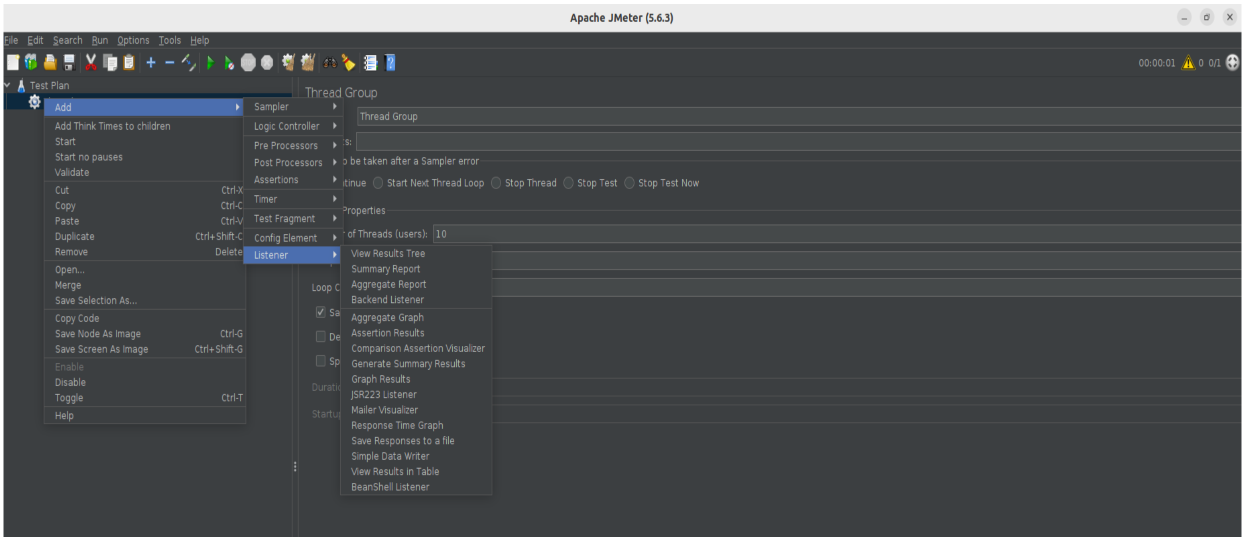
Duration (seconds):

Startup delay (seconds):

4. Listeners

Listeners provide access to the information Jmeter gathers about the test cases while Jmeter runs. The Graph Results listener plots the response times on a graph. The "View Results Tree" Listener shows details of sampler requests and responses, and can display basic HTML and XML representations of the response. Other listeners provide summary or aggregation information.

Listeners can be added anywhere in the test, including directly under the test plan. They will collect data only from elements at or below their level.



Thread Group

Thread Group

be taken after a Sampler error

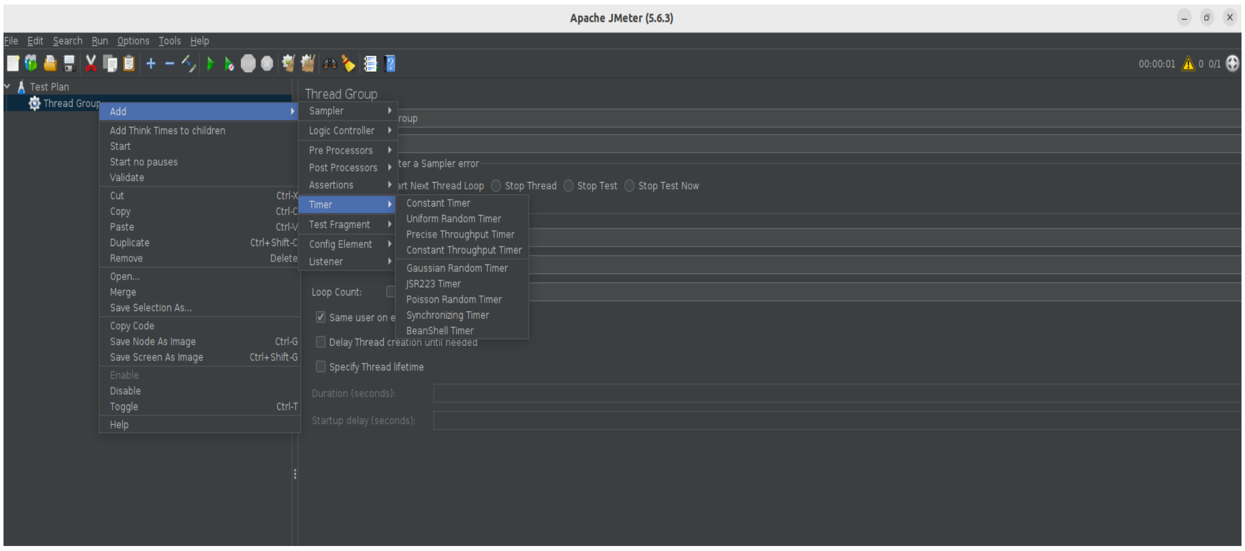
tinue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Properties

of Threads (users): 10

5. Timers

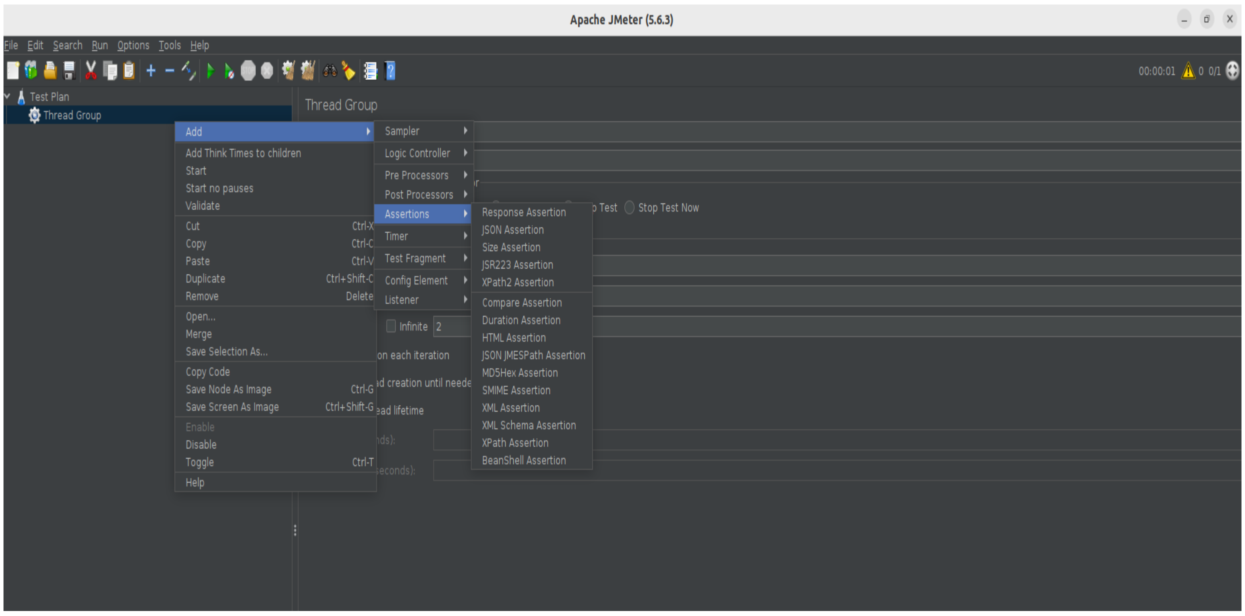
The purpose of the Timer element is to pause a Jmeter Thread for a certain amount of time. Timers allow Jmeter to delay each request that a thread makes. A timer can solve the server overload problem.



6. Assertions

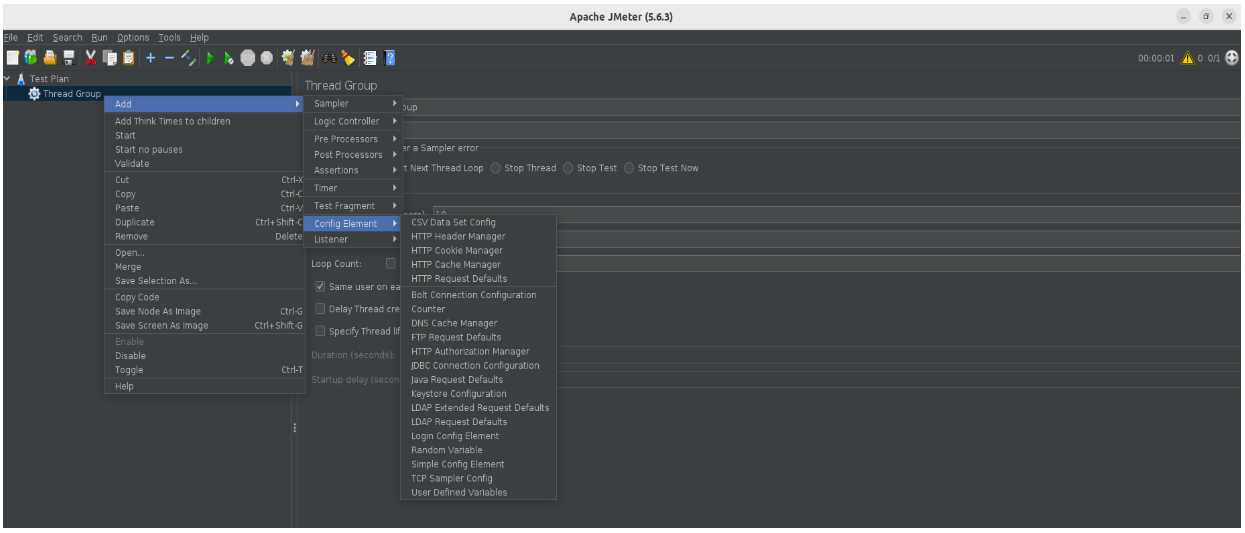
Assertions allow us to assert facts about responses recieved from the server being tested. Using an assertion , we can essentially test that our application is returning the results we expect it to.

**We can add assertions to any sampler.



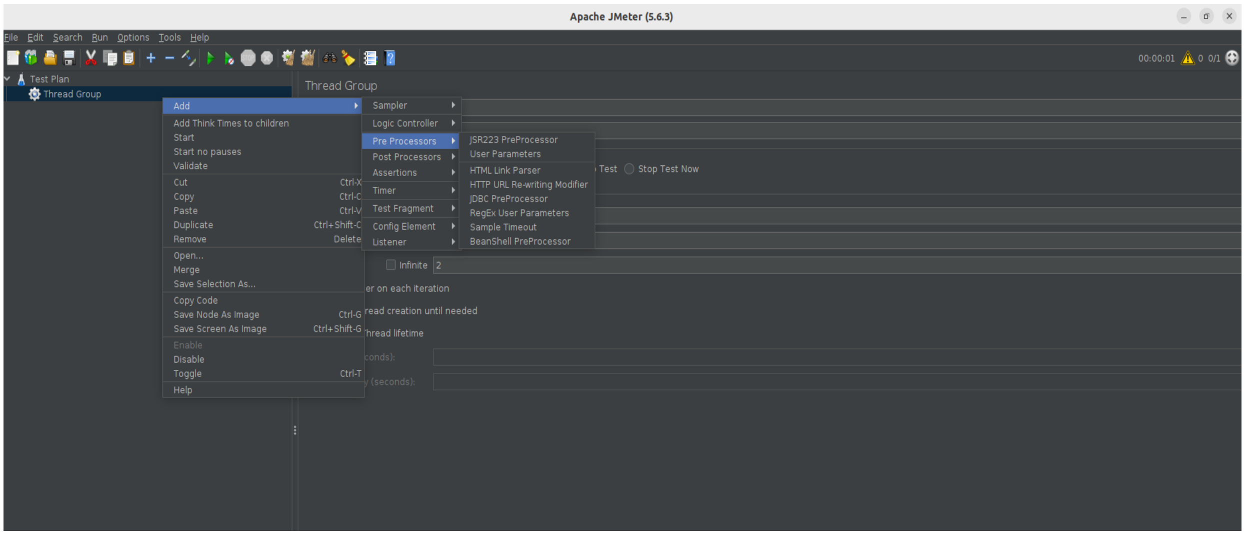
7. Configuration Elements

It is a simple element where we can collect the configuration values of all samplers like webserver’s hostname or database url etc. A conf element is accessible from only inside the tree branch where we place the element.



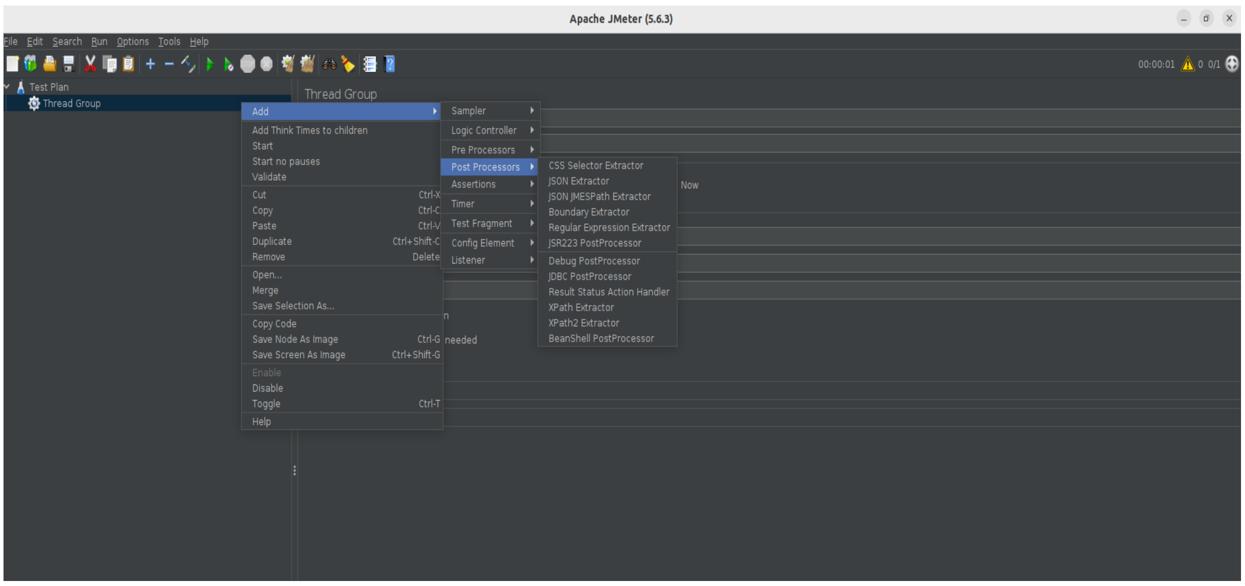
8. Pre-Processor Elements

A Pre-Processor executes some action prior to a Sampler Request being made. If a Pre-Processor is attached to a Sampler element, then it will execute just prior to that sampler element running. A Pre-Processor is most often used to modify the settings of a Sample Request just before it runs, or to update variables that aren't extracted from response text. See the scoping rules for more details on when Pre-Processors are executed.



9. Post – Processor Elements

A Post-Processor executes some action after a Sampler Request has been made. If a Post-Processor is attached to a Sampler element, then it will execute just after that sampler element runs. A Post-Processor is most often used to process the response data, often to extract values from it. See the scoping rules for more details on when Post-Processors are executed.



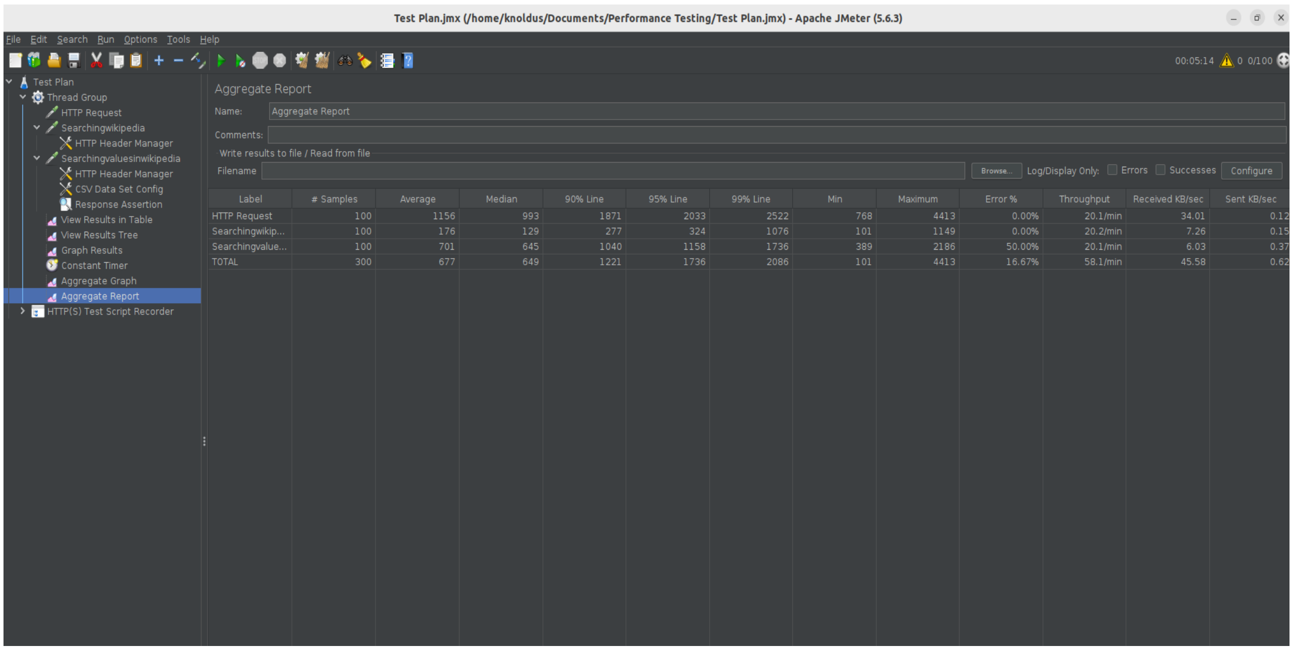
10. Non – Test Elements

Thses elements do not directly participate in the test execution, but have a significant contribution to Jmeter scripting. HTTP(S) Test Script Recorder is specially used to record the test script.

11. Jmeter aggregate report analysis

- **Label** - Label: Basically it is the name of the URL that we provided, in the above example, we provide two HTTP requests that are nothing but the Label of the report.
- **Sample -** It is used to show the number of virtual users per request that we added to the Thread Group.
- **Average -** It is used to specify how much time is taken by a specific request or Label. In the above example, the total average time is 2682 milliseconds as shown in the above screenshot.
- **Median -** It is the time in a bunch of test results. It demonstrates that half of the examples took something like this time i.e. the rest of at minimum as long.
- **90% Line -** It is the Value under which 90 Percent of the examples fall. The rest tests took essentially as long as they were worth. (90th percentile)To work out 90%, Make a rundown of all the exchange values and yet again orchestrate their qualities in dropping requests. Presently wipe out the top 10% exchanges of your all-out list. Staying most noteworthy is the 90th percentile.
- **95% Line -** It is the Value under which 95 Percent of the examples fall. The rest tests took essentially as long as they were worth. (95th percentile) To compute 95%, Make a rundown of all the exchange values and once again orchestrate their qualities in plummeting requests. Presently kill the top 5% exchanges of your all-out list. Staying most elevated worth is the 95th percentile.

- **99% Line -** It is the Value under which 99 Percent of the examples fall. The rest of the tests took basically as long as they were worth. (99th percentile) To compute close to 100%, Make a rundown of all the exchange values and yet again organize by their qualities in plummeting requests. Presently kill the top 1% exchanges of your absolute rundown. Staying most elevated worth is the 99th percentile.
- **Min -** It is used to show the minimum amount of time taken by the label and it is shown in milliseconds.
- **Max -** It is used to show the maximum amount of time taken by the label or we can say that HTTP request.
- **Error% -** It shows failed HTTP requests per label.
- **Throughput -** It shows a number of requests are processed per second.
- **KB/Sec -** It is used to show the average amount of data received from the server during Test Plan execution.



Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request	100	1156	993	1871	2033	2522	768	4413	0.00%	20.1/min	34.01	0.12
Searchingwikip...	100	176	129	277	324	1076	101	1149	0.00%	20.2/min	7.26	0.15
Searchingvalue...	100	701	645	1040	1158	1736	389	2186	50.00%	20.1/min	6.03	0.37
TOTAL	300	677	649	1221	1736	2086	101	4413	16.67%	58.1/min	45.58	0.62

Note : I have attached a CSV file named as “values.csv” in the test plan into that i have taken two values from it , one is a valid input and another one is the Invalid Input and applied assertions on it.