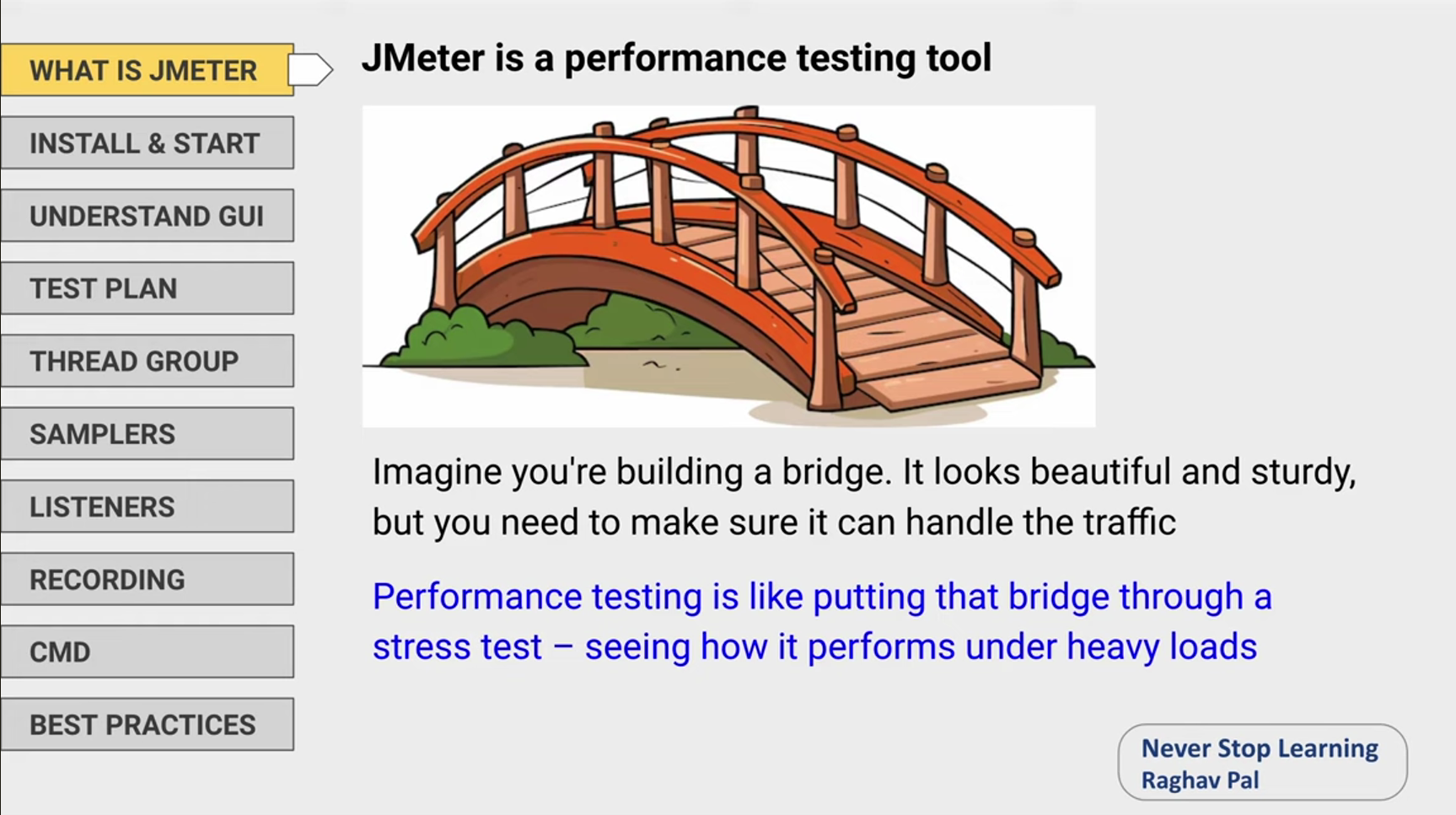
**JMeter**



**View results tree**

When we needed to do real performance test it is advised to disable view result tree. It may consume lot of resources because it has showing lot of details and getting so many data.

**Connect time, Latency, Response time**



**Connect time**- The amount of time spent by a computer user in being connected to a network, **note:** that connect time is not automatically subtracted from latency.

**Latency**- (time to first byte)the number of milliseconds between the time JMeter sent the request and when an initial response was received. The time from just before sending the request to just after receiving the first part of the response

Time that we started getting the response- **Latency**

**Latency is the time until we receive the first byte and connect time is how long it took to initiate the HTTP connection**

1. **Format to Initiate the test in Command Prompt along with csv/jtl report:**

****

**jmeter -n -t "location of your test file" -I "location of results file"**

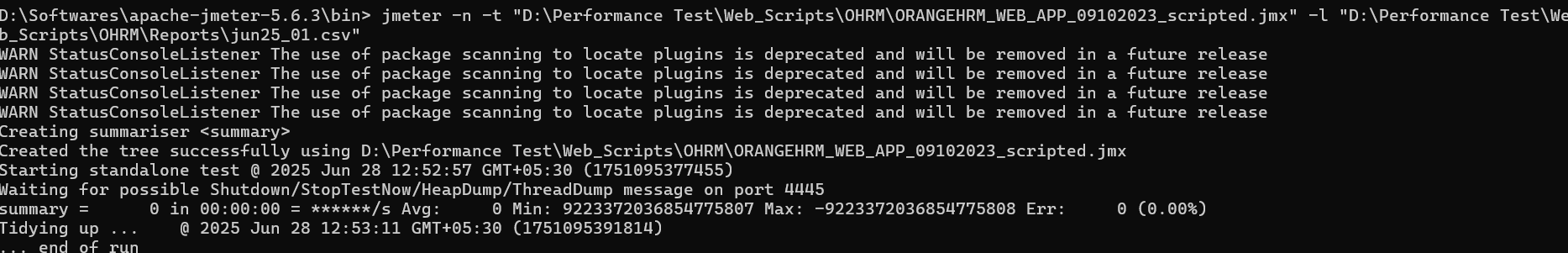
**Explanation:**

* -n → Non-GUI mode
* -t → Path to your .jmx file
* -J → Set JMeter properties (used as variables)
* -l → Path to save the results .jtl

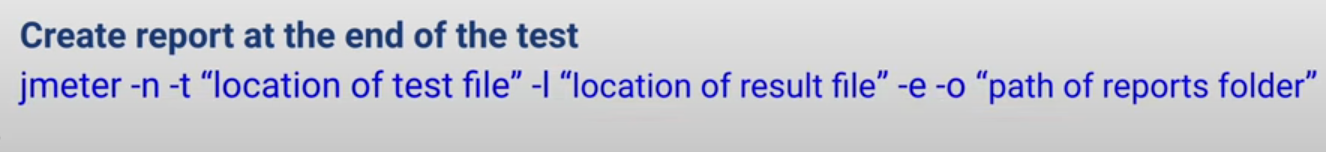
**Example:**

jmeter -n -t "D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_09102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Reports\jun25\_01.csv"

**Execution in CMD:**



1. **Format to Initiate the test with csv/jtl report and HTML report:**

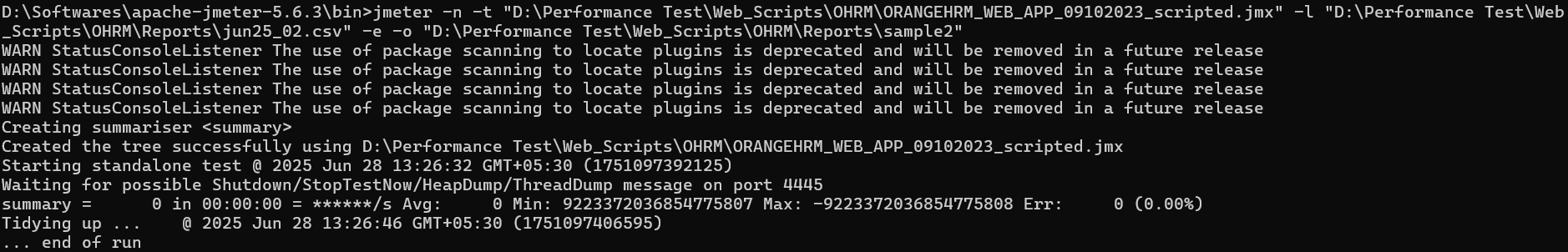


**jmeter -n -t "location of test file" -I "location of result file" -e -o "path of reports folder"**

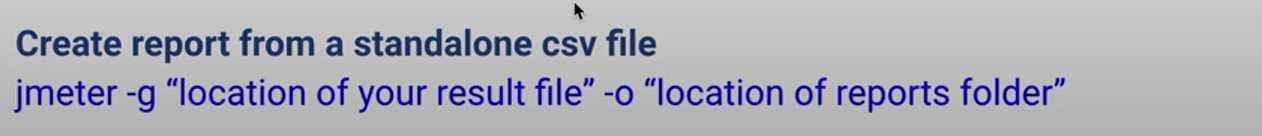
**Example:**

jmeter -n -t "D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_09102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Reports\jun25\_02.csv" -e -o "D:\Performance Test\Web\_Scripts\OHRM\Reports\sample2"

**Execution in CMD:**

****

1. **Create HTML report from existing standalone csv file,**

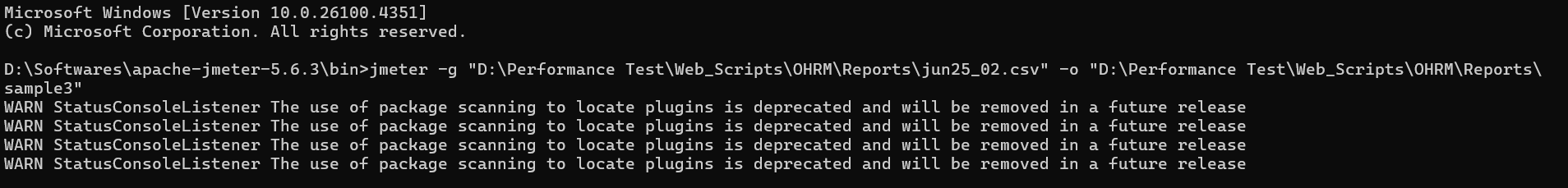
****

**jmeter -g "location of your result file" -o "location of reports folder"**

**Example:**

jmeter -g "D:\Performance Test\Web\_Scripts\OHRM\Reports\jun25\_02.csv" -o "D:\Performance Test\Web\_Scripts\OHRM\Reports\sample3"

**Execution in CMD:**



1. **Execute test by providing thread,rampup & duration from CMD**

**jmeter -n -t "location of test file" -Jthreads=20 -JrampUpTime=60 -Jduration=60 -JloopCount=1 -Jdelay=1 -l "location of result file" -e -o "path of HTML reports folder"**

**Update the parameters in JMeter JMX file:**

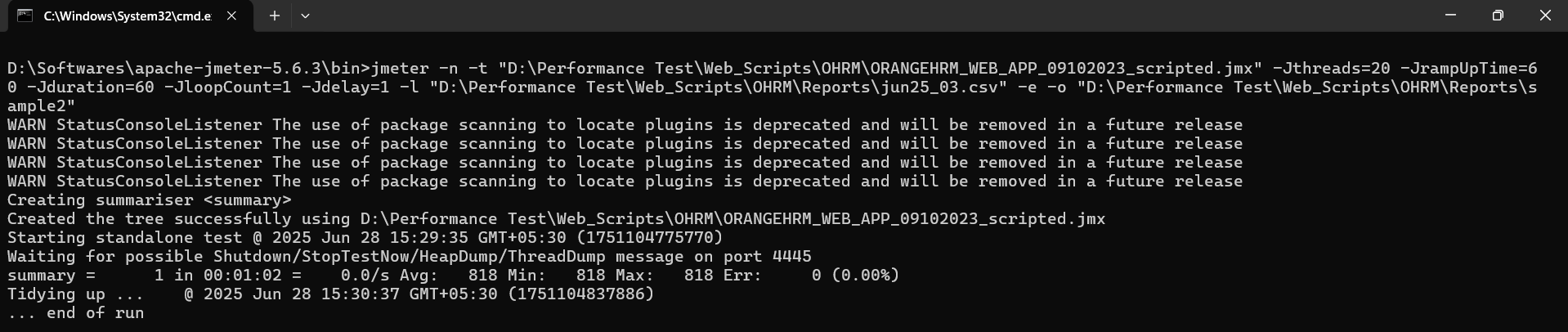
|  |  |
| --- | --- |
| **Thread Count** | ${\_\_P(threads,1)} |
| **Ramp-up time** | ${\_\_P(rampUpTime,1)} |
| **Loop Count** | ${\_\_P(loopCount,1)} |
| **Duration** | ${\_\_P(duration,60)} |
| **Start-up Delay** | ${\_\_P(delay,1)} |
| **Access Token** | ${\_\_P(AccessToken,1)} |

Instead of **${\_\_P(threads,1)}** this format you can use this **${threads}** as well.

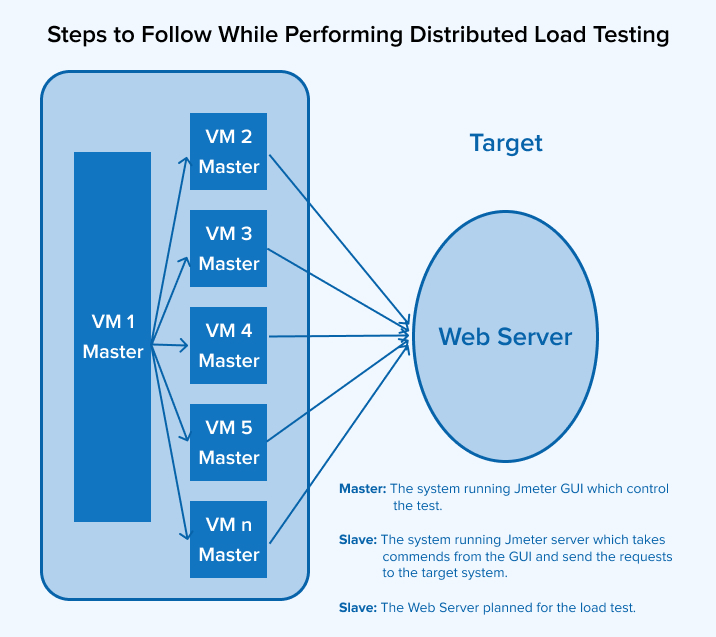
**Example:**

jmeter -n -t "D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_09102023\_scripted.jmx" -Jthreads=20 -JrampUpTime=60 -Jduration=60 -JloopCount=1 -Jdelay=1 -l "D:\Performance Test\Web\_Scripts\OHRM\Reports\jun25\_03.csv" -e -o "D:\Performance Test\Web\_Scripts\OHRM\Reports\sample2"

**Execution in CMD:**

****

**Remote test or Distributed test**



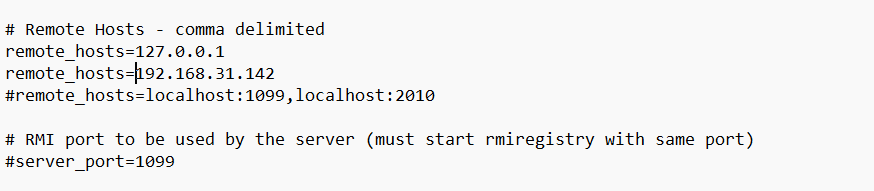
**Step 1 :** Setup Master added remote system’s IP in jmeter.properties

**Step 2 :** Create keystore file run create-rmi-keystore.bat / create-rmi-keystore.sh name : rmi password : changeit

**Step 3 :** run jmeter-server file on slave (remote) system

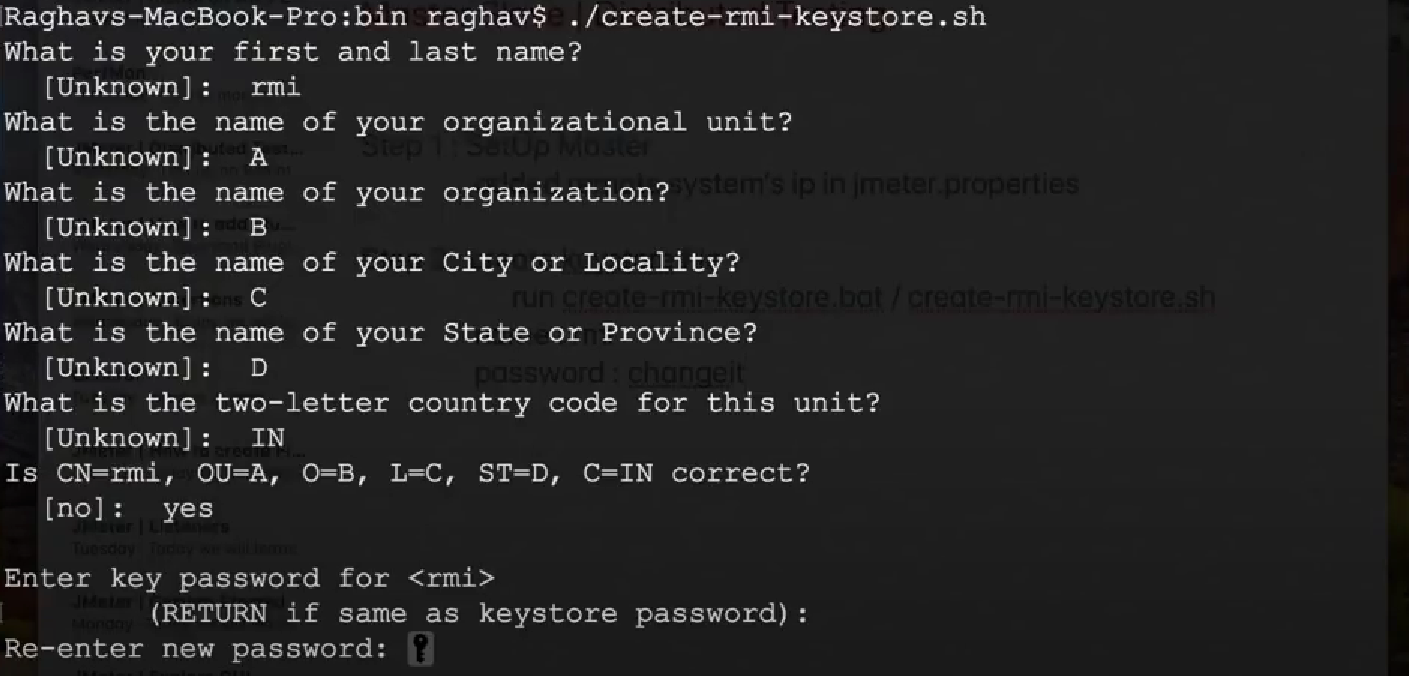
**Illustration:**

Step1: Add the remote machine ip address

****

Step 2: Open **Create keystore file** in bin folder and provide the answers

**Illustration**



Keep password as: **changeit** (if its asked)

Once its id done we can able to find: **rmi\_keystore.jks**

**Step 3:** Now you need to start **run jmeter-server file on slave (remote) system** in bin folder

Step 4: Remote test will work fine

* If you want to run test with **100 Users** then give **thread Count as 50**.
* All systems (master and slaves) have same Version of JMeter
* All systems have java (preferably same Version)
* All systems can connect to each other (are in same subnet)
* no need to copy JMeter script (JMX) to slave systems

**To run the distributed test in CMD and saving the results**

Open the JMeter server batch file as well, while running

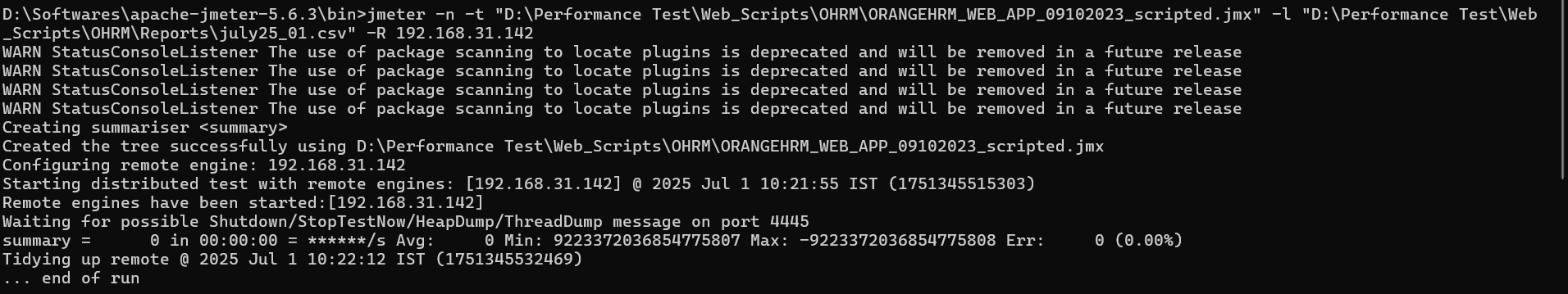
**Command:** Syntax

**jmeter -n -t "location of test file" -I "location of result file" -R 192.168.31.142**

**Example:**

**jmeter -n -t "D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_09102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Reports\july25\_01.csv" -R 192.168.31.142**

**Example in CMD:**

****

**JMeter Intermediate**

1. **Execute the script from Jenkins,**

(Pre-requests- Add the Jenkins performance plugin (Performance.hpi) to Jenkins local machine plugin folder)

* Goto Jmeter/bin user.properties - add line **jmeter.save.saveservice.output\_format=xml**
* **Create a Jmeter test and Run** JMeter test from command line to check every thing is fine
* **Changing the directory and executing the test**

&& is used to integrate two commands in a single command:

**Example:**

cd /d D:\Softwares\apache-jmeter-5.6.3\bin && jmeter.bat -Jjmeter.save.saveservice.output\_format=xml -n -t "D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_10102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Jenkins\_Report\Jenkins\_OHRM.jtl"

* Add a job in Jenkins - add build step **Execute Windows batch command**

**Command**:

cd /d D:\Softwares\apache-jmeter-5.6.3\bin && jmeter.bat -Jjmeter.save.saveservice.output\_format=xml -n -t "D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_10102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Jenkins\_Report\Jenkins\_OHRM.jtl"

* **Post build actions-** Source data files

D:\Performance Test\Web\_Scripts\OHRM\Jenkins\_Report\Jenkins\_OHRM.jtl

* Save and execute. ( **JenkinsJobIntermediate\_1**- Job Name)

1. **SMTP sampler to send email**

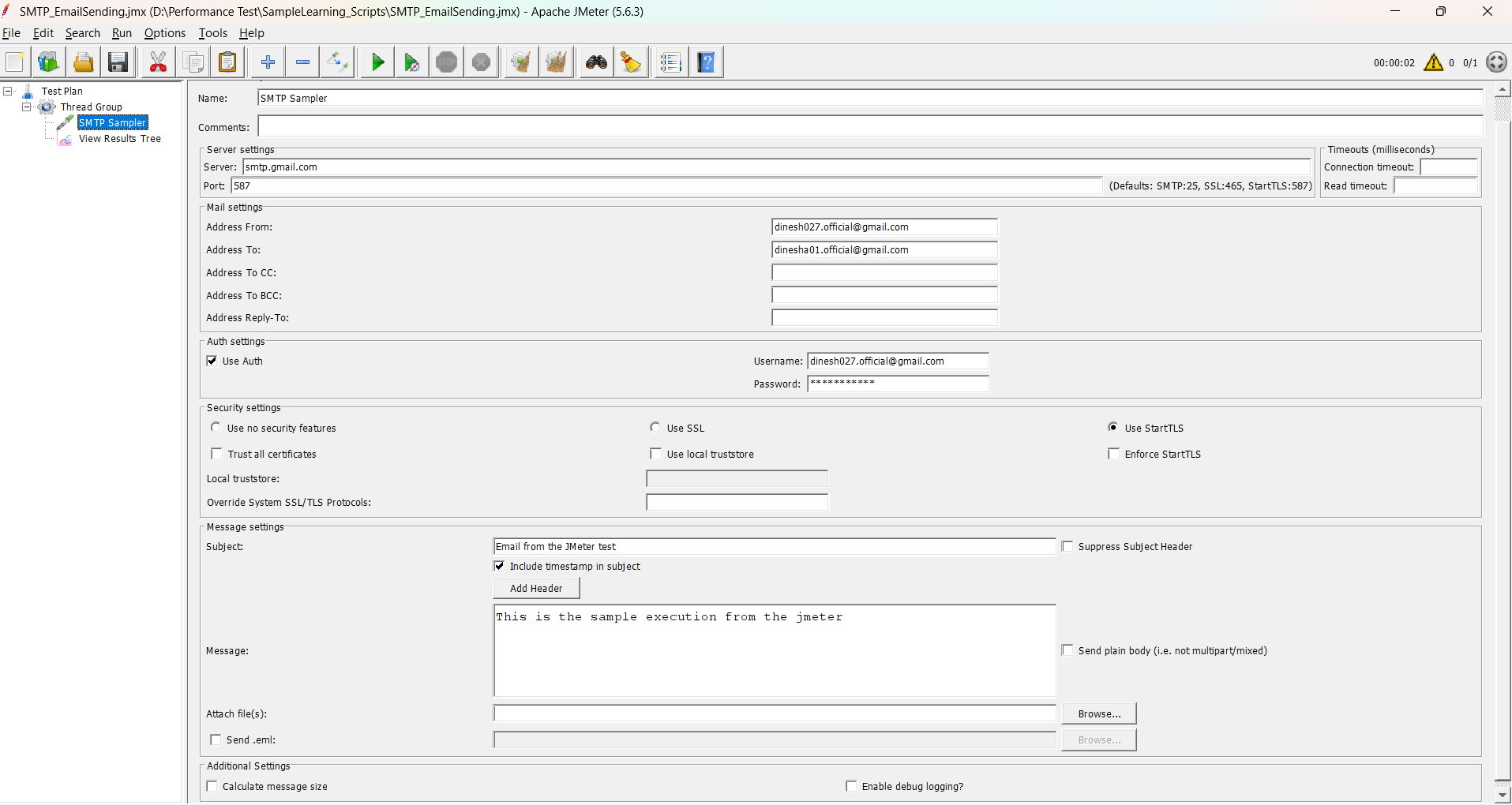
**Step 1** : Download javamail jar and put in jmeter lib folder - restart jmeter**.**

**Step 2** : Create TestPlan - add ThreadGroup - add Sampler SMTP

**Step 3 :** Add data in sampler smtp.googlemail.com / smtp.gmail.com 587 / 465

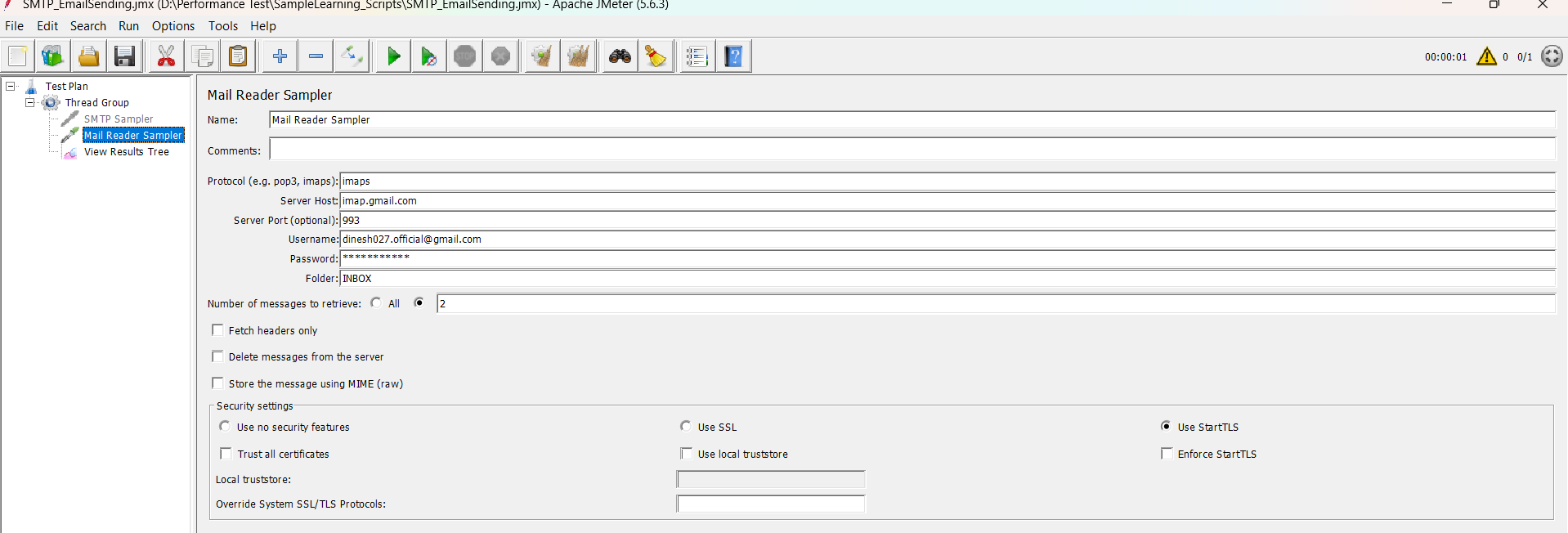
**Step 4 :** Add listeners to view test results

**Step 5 :** Run and validate

****

1. **Mail reader Sampler:**

The Mail Reader Sampler can read (and optionAlly delete) mail messages using POP3(S) or IMAP(S) protocols.



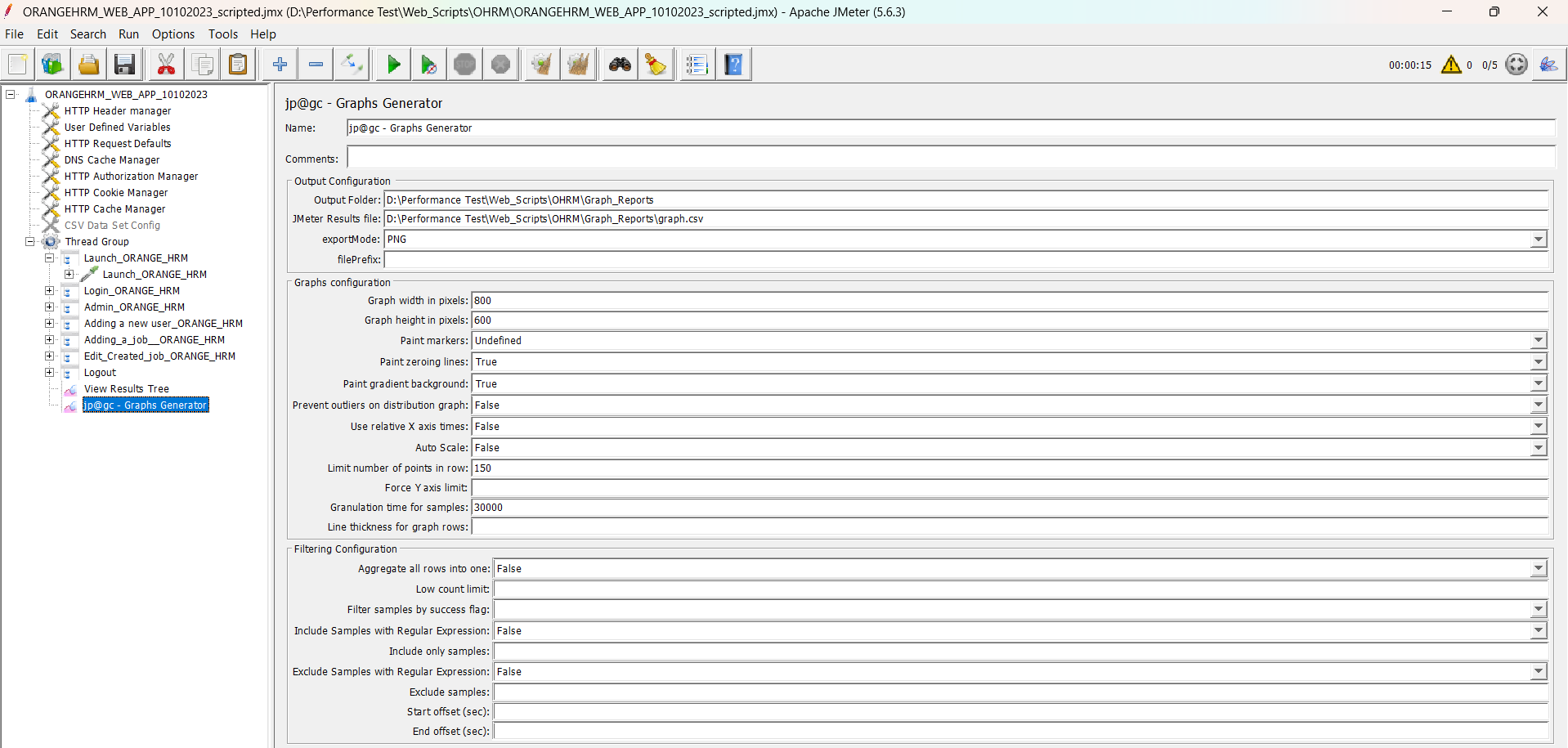
1. **Graphical Results- Graph Generator.**

Important plugins to be downloaded for Graphical results

| **Plugin Name** | **JAR File** |
| --- | --- |
| CMD Tool | jmeter-plugins-cmd.jar |
| Graphs Generator | jmeter-plugins-graphs-ggl.jar |
| Graphs Basic | jmeter-plugins-graphs-basic.jar |
| Synthesis Report | jmeter-plugins-synthesis.jar |
| Response Times Distribution | jmeter-plugins-response-times-distribution.jar |
| Percentiles Graph | jmeter-plugins-perfmon.jar *(or part of basic/graphs)* |
| Times vs Threads | jmeter-plugins-tst.jar |

**Note:** All ways add jp@gc - Graphs Generator under view results tree

Add the csv file to write the response in View result tree

****

**Graphical Results Execution in Non GUI Mode:**

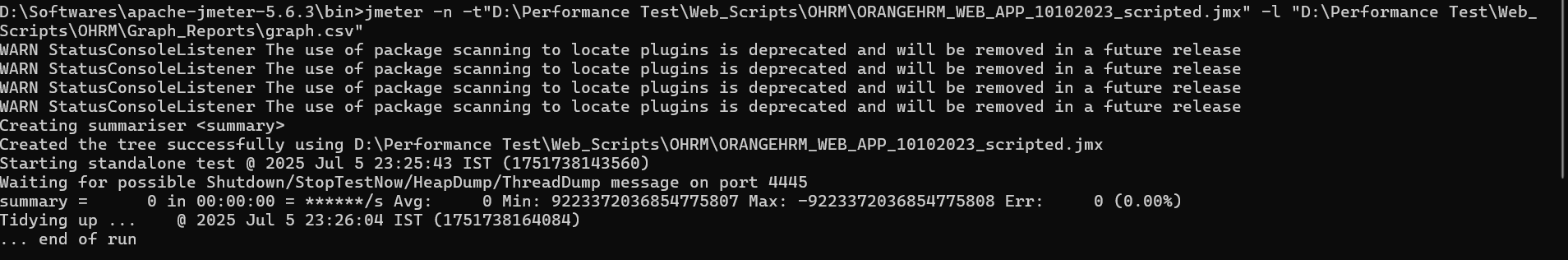
**Example:**

**jmeter -n -t "location of test file" -I "location of result file"**

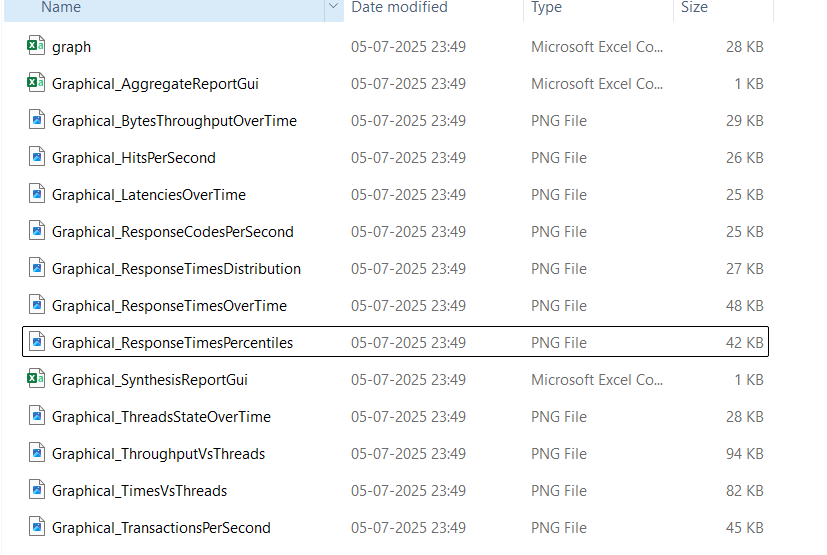
**Command:**

jmeter -n -t"D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_10102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Graph\_Reports\graph.csv"

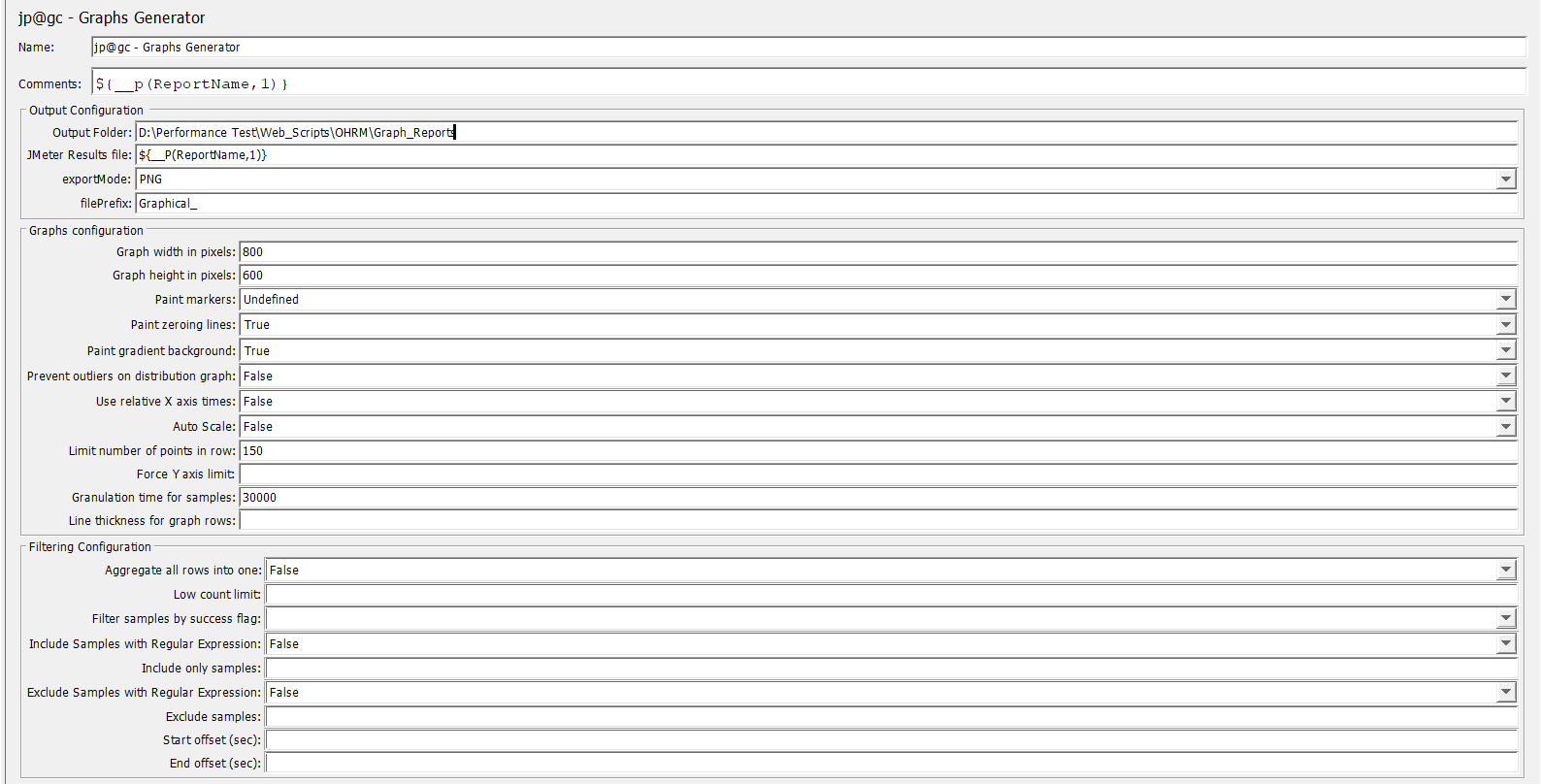
**Execution in CMD:**

****

Reports Created:



Now in script parametarized **${\_\_P(ReportName,1)}** and passing from CMD



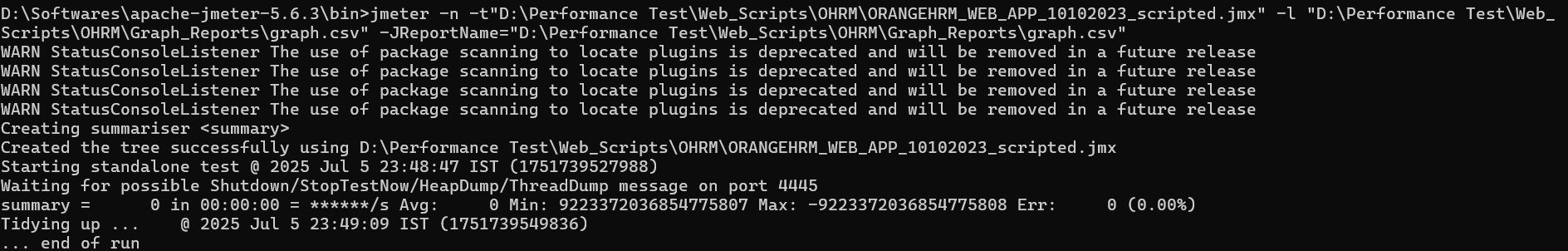
**Example:**

**jmeter -n -t "location of test file" -I "location of result file" -J ReportName=” location of result file”**

**Command:**

jmeter -n -t"D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_10102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Graph\_Reports\graph.csv" -JReportName="D:\Performance Test\Web\_Scripts\OHRM\Graph\_Reports\graph.csv"

**Execution in CMD:**



1. **Generate graphical results from NON-GUI test run:(JMeter execution in CMD)**

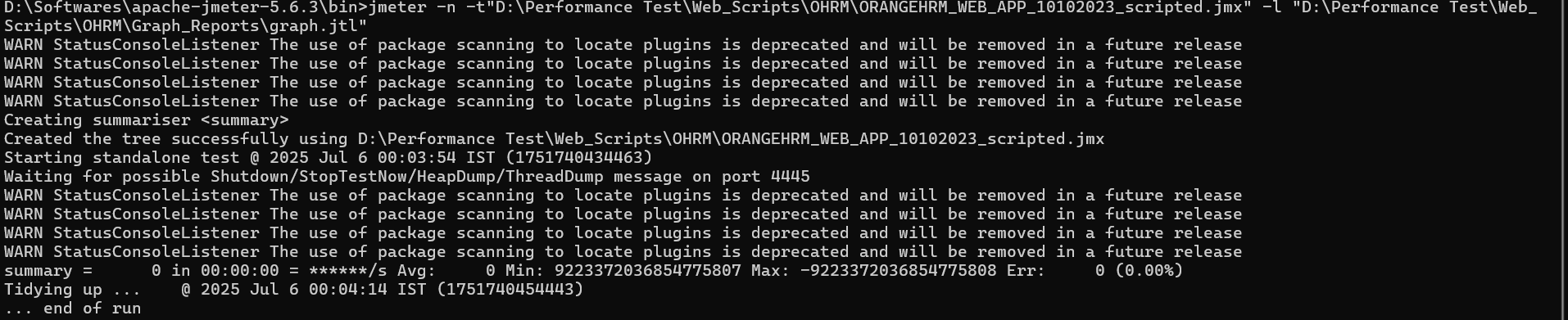
**Example:**

**jmeter -n -t “location of jmeter script .jmx” -l “location of output file”**

**Command:**

jmeter -n -t"D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_10102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Graph\_Reports\graph.jtl"

**Execution in CMD:**

****

Now the JTL file we can open in All the listeners and browses we can able to see the detailed report format.

1. **Summarizer – Logs while executing in CMD:**

* **JMeter summarizer-** Open JMeter properties

Uncommand the summarizer things

Now you can able to see the logs in NON-GUI mode execution.

* **Add Console status logger:**

And add the Console Status Logger in test plan and execute in NON-GUI mode

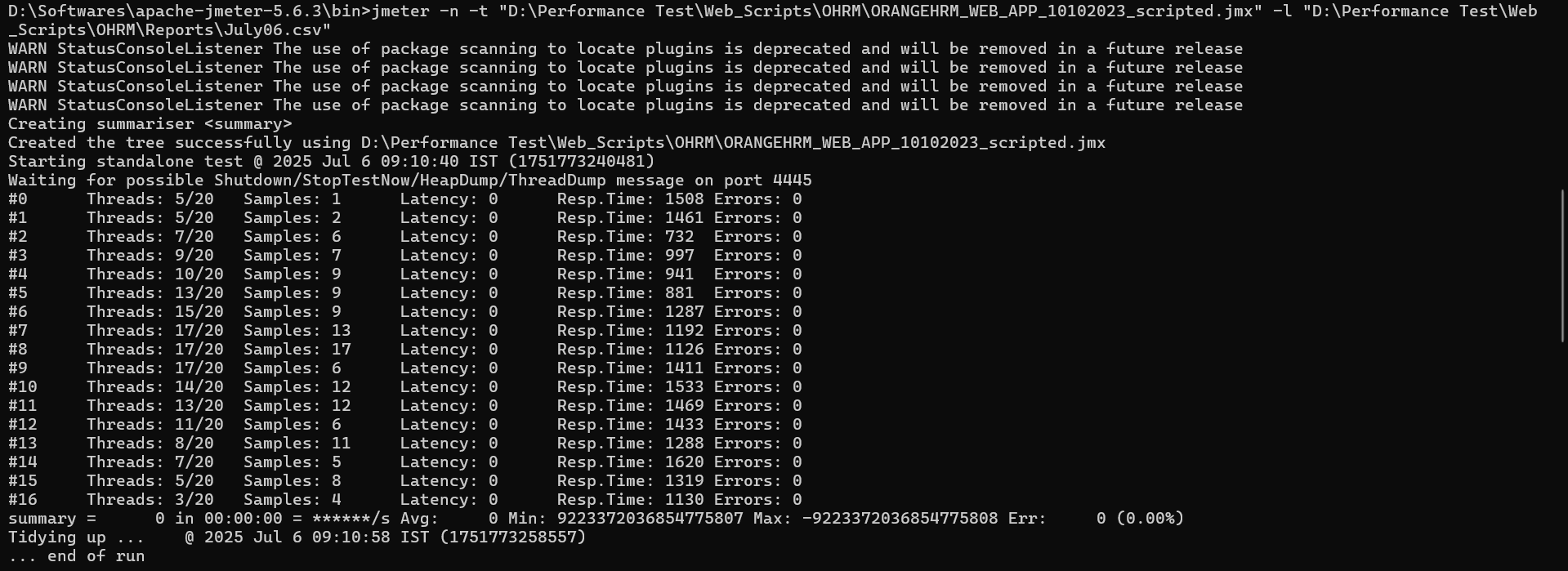
**Example:**

**jmeter -n -t “location of jmeter script .jmx” -l “location of output file”**

**Command:**

jmeter -n -t "D:\Performance Test\Web\_Scripts\OHRM\ORANGEHRM\_WEB\_APP\_10102023\_scripted.jmx" -l "D:\Performance Test\Web\_Scripts\OHRM\Reports\July06.csv"

**Execution in CMD:**

****

**Controllers:**

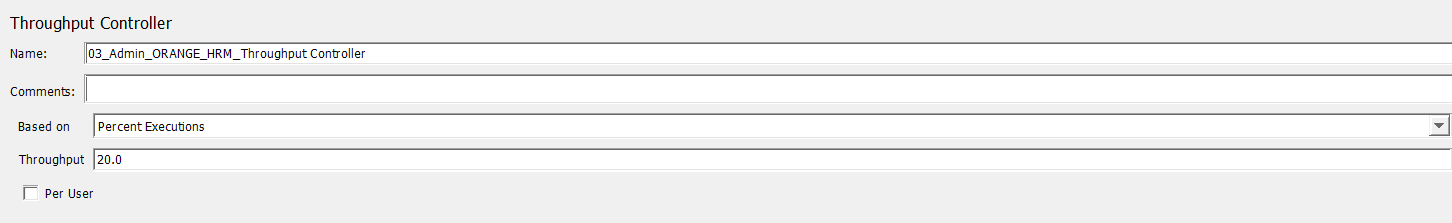
1. **Throughput Controller**

This controller is badly named, as it does not control throughput. Please refer to the [Constant Throughput Timer](https://jmeter.apache.org/usermanual/component_reference.html#Constant_Throughput_Timer) for an element that can be used to adjust the throughput.

The Throughput Controller can yield very complex behavior when combined with other controllers - in particular with interleave or random controllers as parents (also very useful).

1. **Percent executions**

causes the controller to execute a certain percentage of the iterations through the test plan.



1. **Total executions**

causes the controller to stop executing after a certain number of executions have occurred.



The **Throughput Controller** in JMeter controls **how often its child samplers are executed** during the test.

The **"Per User"** checkbox determines **how the "Throughput" value is interpreted**:

| **Per User Option** | **Meaning** |
| --- | --- |
| ✅ Checked | Each **virtual user (thread)** will run the child samplers "Throughput" number of times. |
| ❌ Unchecked | All users **combined** will run the child samplers "Throughput" number of times. |

1. **Module controller.**

The module controller provides a list of All controllers loaded into the gui. Select the one you want to substitute in at runtime.

**Scenario: Testing a Web Application (like OrangeHRM)**

Your application has these 3 business actions:

1. ✅ **Login**
2. 👤 **Add Employee**
3. 📋 **Generate Report**

In a real-time test, each user must:

* First log in
* Then perform any one or more of the above actions
* FinAlly log out

**💡 Problem (Without Module Controller)**

If you write the **Login** steps under every test flow manuAlly:

* Login → Add Employee
* Login → Generate Report
* Login → View Dashboard

You are **repeating the Login script multiple times**.  
If the login API changes, you have to update **everywhere** → Not maintainable!

**✅ Solution: Use Module Controller**

**🧱 You define modules like this (in a separate part of the test plan):**

* **Login Module** – handles authentication
* **Add Employee Module** – adds a new employee
* **Generate Report Module** – triggers a report generation
* **Logout Module** – logs out the user

**🧠 Then in your Thread Group:**

You add **Module Controllers** that point to each module:

[Thread Group]

├── Module Controller → Login Module

├── Module Controller → Add Employee Module

├── Module Controller → Logout Module

So during the test, it executes only the referenced modules — like a **shortcut or link**.

**🔁 Real-Time Execution Flow Example**

Let’s say 3 test flows are needed:

| **Scenario Name** | **Modules Used** |
| --- | --- |
| Add Employee | Login → Add Employee → Logout |
| View Report | Login → Generate Report → Logout |
| Just Login Check | Login → Logout |

You can build All this in **one JMX file**, using **Module Controllers** instead of copying samplers.

**Other Examples:**

Test Plan

├── Thread Group: Actual Execution

│ └── Module Controller → selects Login Module

│

└── Thread Group: Modules (Disabled)

├── Simple Controller (Login Module)

│ └── HTTP Request → /login

├── Simple Controller (Add Employee Module)

│ └── HTTP Request → /employee/add

└── Simple Controller (Logout Module)

└── HTTP Request → /logout

In this example: above

* You only run **Thread Group 1**
* Module Controller fetches the desired module from the **disabled Thread Group 2**

**✅ Why This Is Used in Real Projects**

* ✅ **Modular design** → like coding functions
* ✅ Easy to update (change login in one place)
* ✅ Reduces errors (less copy-pasting)
* ✅ Great for large teams or shared test scripts
* ✅ Works very well with **Test Fragments** for better reusability

| **Concept** | **Explanation** |
| --- | --- |
| Module Controller | Points to and runs a specific test fragment (controller + child samplers) |
| Test Fragment | A block (controller + samplers) that can be reused |
| Fragment Location | Can be in any thread group, but preferably in a **disabled one** |
| Unique Names Required | To avoid conflicts in dropdown selection and misexecution |
| Real Use Cases | Reuse login, logout, common flows, switch modules easily |

1. **Include Controller.**

**JMeter recording:**

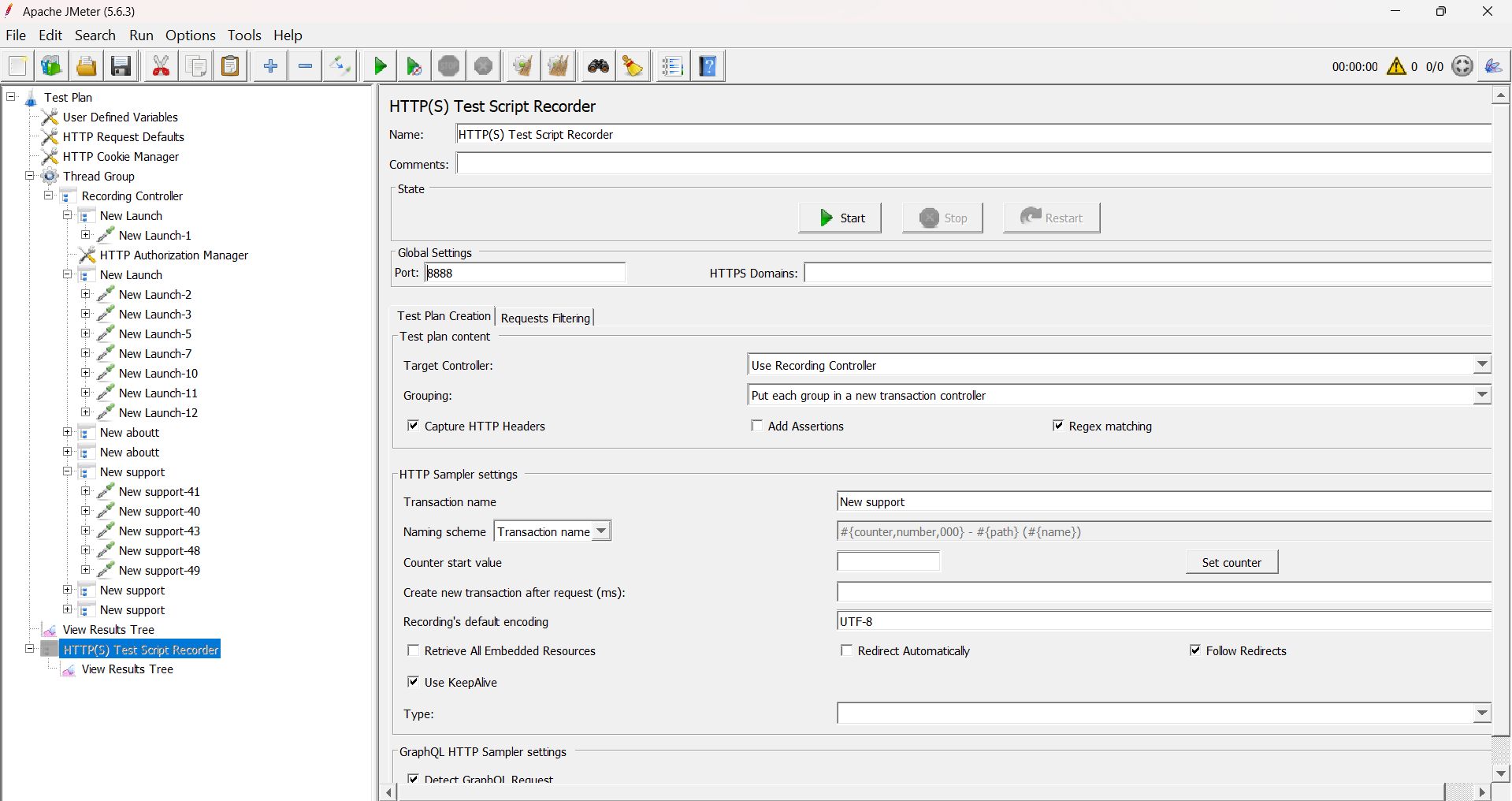
**STEP 1:** Add thread group and add http test script recorder.

**STEP 2:** Do the manual proxy settings in the system and disable automatic proxy setting.

**STEP:** Trigger the **Start** button in the JMeter Root CA Certificate will generate in JMeter bin folder, import that in to system certificate.

(Open manage Computer Certificate 🡪 Select Trusted root certification🡪 Certificates 🡪 File 🡪options🡪Locate and import the certificate)

Then you can able to start the recording.



**Best Practices,**

* Select **Grouping** as **Put each group in a new transaction controller**
* **Naming Scheme** as **Transaction Name or Prefix or Sufix.**
* **Target Controller** you can select while recording where the particular request to store
* **Under Requests filters:** You can include and exclude the necessary or unwanted URLS

#### Setup Elements in JMeter:

1. **Add Thread Group** under Test Plan
2. Add **HTTP Request Defaults** (to define server/domain)
3. Add **Recording Controller** (to store recorded steps)
4. Add **HTTP(S) Test Script Recorder**  
   Right-click Test Plan > Add > Non-Test Elements > HTTP(S) Test Script Recorder

| **Setting** | **Value** |
| --- | --- |
| **Port** | 8888 (default) |
| **Target Controller** | Recording Controller |
| **Group by** | Put each group in a new transaction controller (recommended) |
| **HTTPS Domains** | Add target domains (for filtering) |
| **Capture HTTP Headers** | Enable, but filter unnecessary headers |
| **Filter URL Patterns** | Exclude unwanted URLs (e.g., .css, .png, .js) |
| **Add Response Time** | Optional for debug |
| **Content Type Filter** | e.g., `text/html |
| **Correlation** | **Regex/JSON Extractor, JMESPath** |
| **Parametrization** | **CSV Data Set Config** |

### ****Post Recording Clean-Up (Mandatory)****

**Advanced tip**: JMeter records a lot of noise. Clean the script using:

* Remove static resources (.js, .png, .css)
* Parametrize values (correlation):
  + Extract CSRF tokens, Session IDs using **Regular Expression Extractor** or **JSON Extractor**
* Replace hardcoded data with **CSV Data Set Config**
* Add **Assertions**, **Think Times**, and **Timers**

##### **Application Performance Monitoring tools including :**

##### **DX APM**

##### **AppDynamics**

##### **AWS Cloudwatch**

##### **Dynatrace**

##### **Datadog**

##### **New Relic**

**Blazemeter**

**What is BlazeMeter?**

BlazeMeter is a cloud-based performance testing platform that:

Fully supports JMeter and extends its capabilities

Enables testing at scale, collaboration, CI/CD integration, and real-time dashboards

It was originAlly a commercial layer over JMeter (now part of Perforce) and is popular in enterprise-grade performance testing.

| **Feature** | **BlazeMeter** | **JMeter** |
| --- | --- | --- |
| **Scalability** | Run distributed tests from cloud (1000s of users) | Needs manual setup for distributed testing |
| **CI/CD Integration** | Built-in Jenkins, GitHub Actions, Azure DevOps plugins | Manual scripting required |
| **Test from multiple geographies** | Yes (run from different cloud regions) | No (runs locAlly unless set up via remote servers) |
| **Real-time reporting** | Live dashboards, graphs, alerts | Basic listeners + offline HTML reports |
| **API Monitoring** | Built-in (24x7 uptime & response check) | Not available |
| **Script Conversion Tools** | Postman2JMX, Swagger import, HAR converter | Manual or 3rd party |
| **Collaboration** | Share tests and reports within teams | Not natively |
| **SaaS Deployment** | Web-based UI, no instAll needed | Desktop app required |

BlazeMeter is like a **pro** version of **JMeter** in the cloud — ideal for teams, enterprises, and CI/CD pipelines.

It simplifies scaling, sharing, converting, and analysing JMeter tests — especiAlly when your needs go beyond what the local **JMeter GUI** can do.

**Use BlazeMeter when:**

* You need to run large-scale tests or test from multiple regions
* Your team needs centralized test management & collaboration
* You want to automate performance testing as part of your CI/CD process
* You want to test without dealing with complex JMeter setups

1. **Postman to JMeter conversion- To do**

BlazeMeter has a free online tool to convert Postman collections to JMX easily.

👉https://converter.blazemeter.com

2. **Run JMeter tests in Jenkins CI/CD**

Use BlazeMeter plugin to run and report JMeter tests as part of pipelines.

3. **Cloud Load Testing**

* No need to manage JMeter slaves
* Simulate 10k+ users easily
* Ideal for geo-distributed testing

4. **Performance Monitoring**

Integrates with APM tools (e.g., New Relic, Dynatrace)

Offers threshold-based alerts

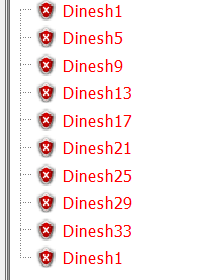
**Counter Config Element:**

Allows the user to create a counter that can be referenced anywhere in the Thread Group. The counter config lets the user configure a starting point, a maximum, and the increment. The counter will loop from the start to the max, and then start over with the start, continuing on like that until the test is ended.

**Counter Example:** For 10 threads

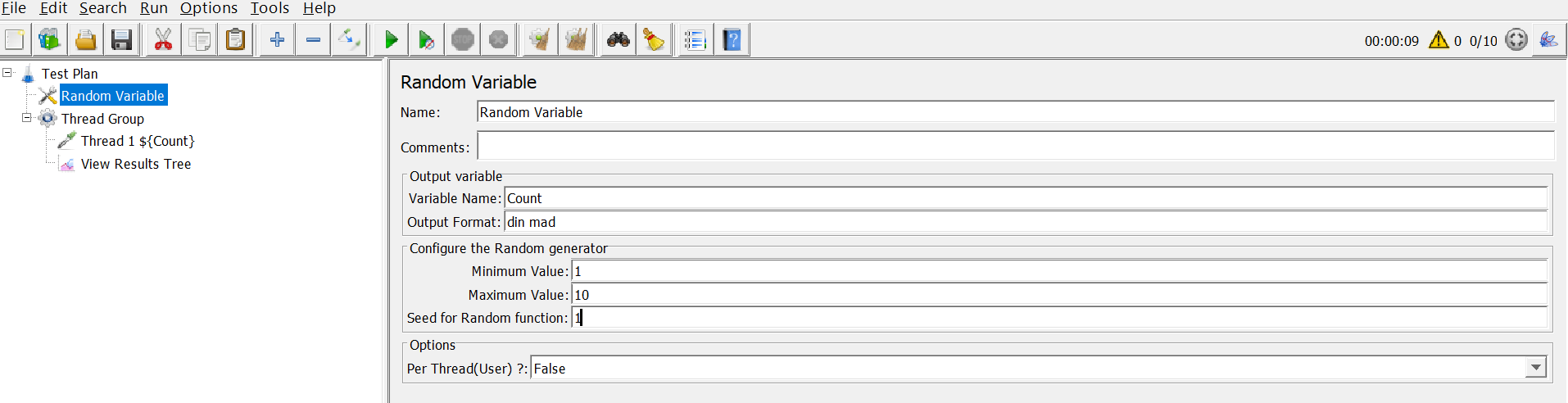


**Results:**



**Random Variable:**

Example: Threads 10 , 2 times ranned

****

Random Values will **Repeat:** By **seed of Random Function**



**Result CSV file- Time stamp Format**

In excel sheet we can able to see the time stamp of the request in different format to change that in to IST time:

**Go🡪 JMeter.properties🡪 Search for timestamp format🡪 Un comment the 🡪 IST type of format**

**Aggregate Report from CMD:**

**Prerequisite:** Need to download JMeterPluginsCMD and CMD runner JAR file **(V.2.2)**

JMeterPluginsCMD.bat --generate-csv "location of csv file" --input-jtl "location of jtl file" --plugin-type AggregateReport

**JMeterPluginsCMD.bat --generate-csv "D:\Performance Test\Web\_Scripts\OHRM\Reports\july0013" --input-jtl "D:\Performance Test\Web\_Scripts\OHRM\Reports\july13.jtl" --plugin-type AggregateReport**

**-For me yet To Do**

**While Controller:**

* **DynamicAlly control iterations** based on response data
* **Repeat** until a specific condition is met (like token expiry, successful login, etc.)
* **Exit early** from looping logic when criteria is fulfilled.
* The While Controller runs its children until the condition is "**false**".
* Loop continues **while the condition is true**

**Example 1:**

Set a counter in **User Defined Variables**:

count = 0

Use a **While Controller**:

Condition: ${\_\_javaScript(${count}++ < 5)}

**Result**: This runs the loop **5 times**.

| **Feature** | **Details** |
| --- | --- |
| Purpose | Repeat actions **until** a condition is false |
| Best For | Dynamic loops, polling, retries |
| Can Use | Variables, functions (\_\_javaScript), logical conditions |
| Common Risk | Infinite loops if not carefully controlled |

**Test Plan**

│

├── Thread Group

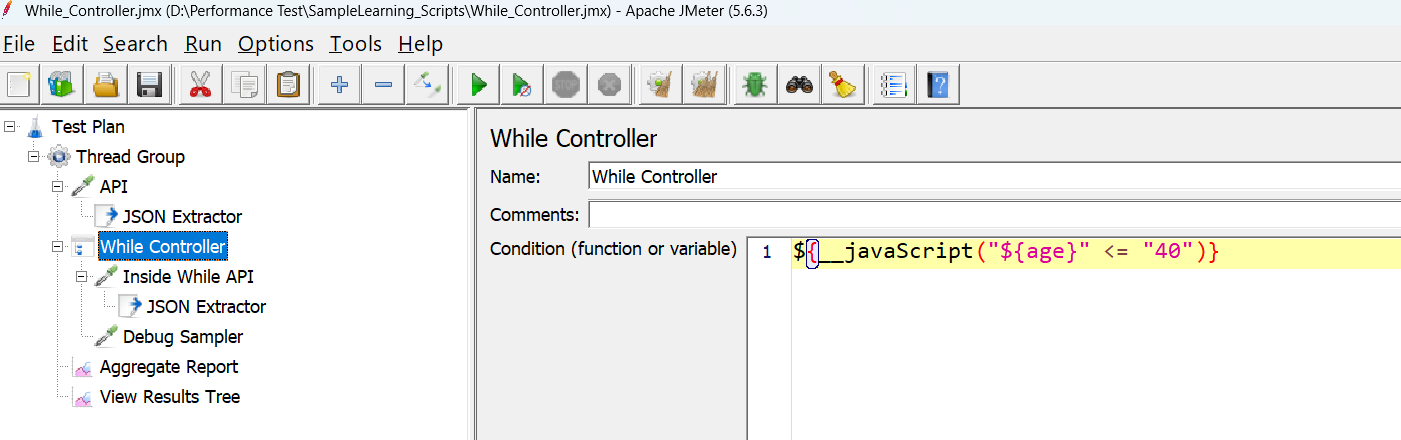
│ ├── HTTP Request → (Initial GET to /**Age**)

│ ├── JSON Extractor → Extract "status" from response

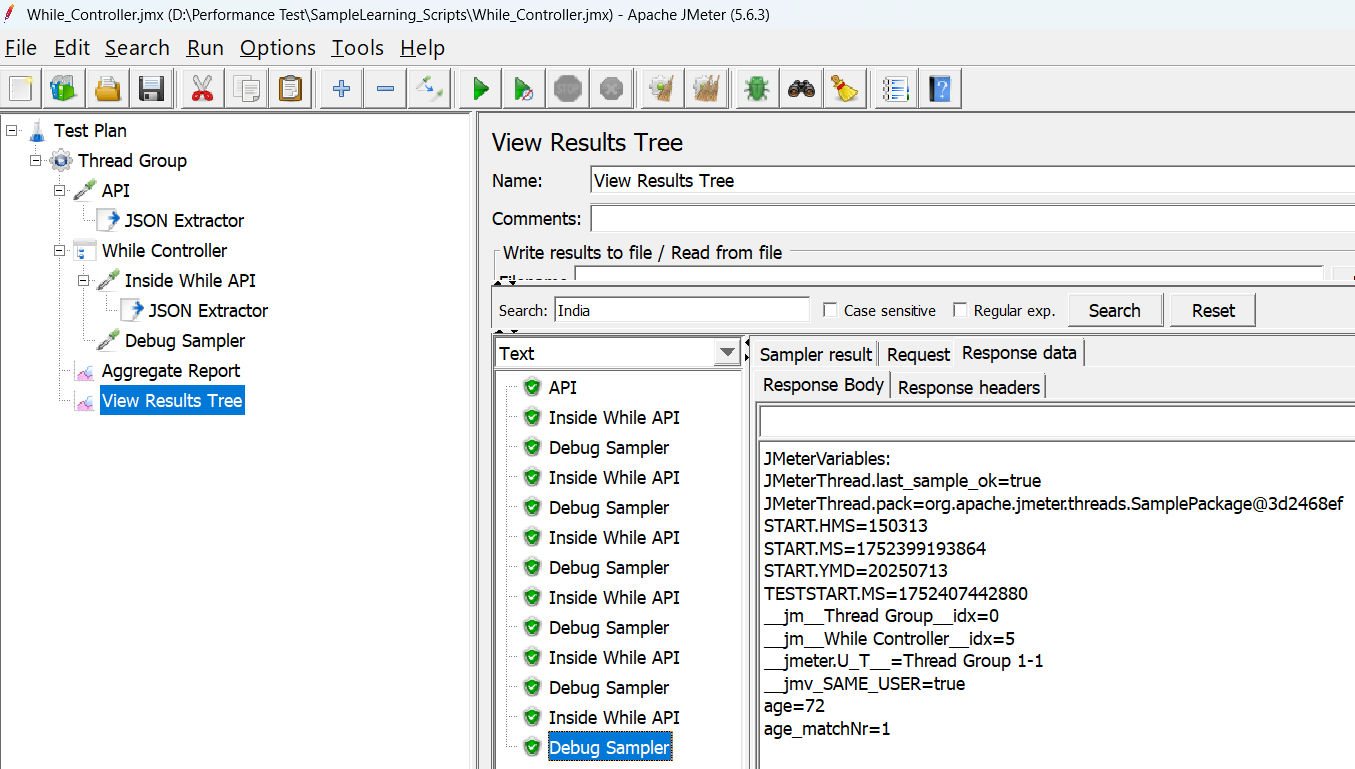
│ └── While Controller (Condition: "${Age}" <= "70")

│ ├── HTTP Request → (GET /status again)

│ └── JSON Extractor → Update



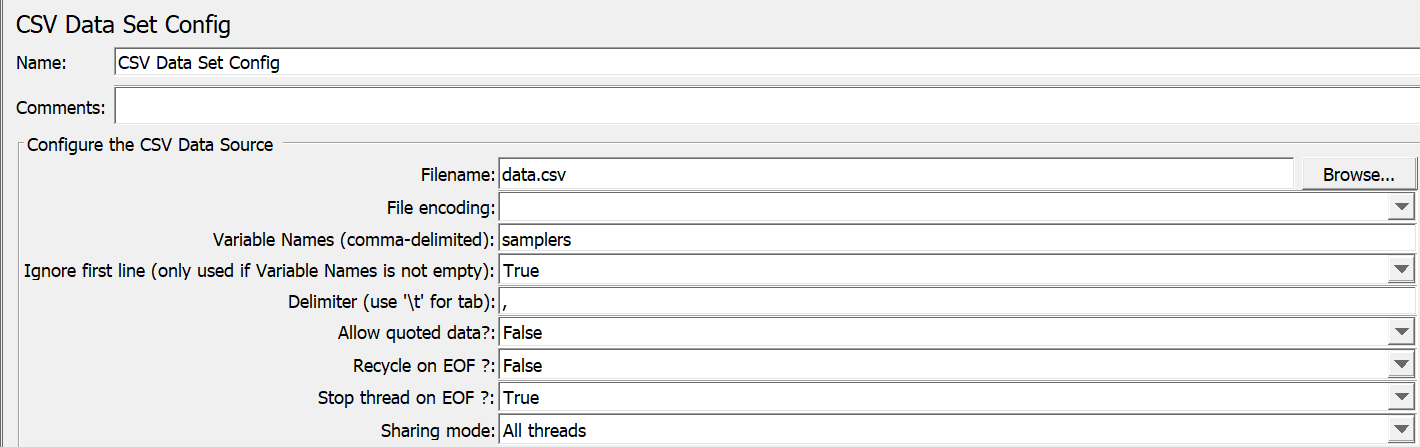
Results:



We can use the While controller; it will expect for the false statement and it will print once then it will stop the loop.

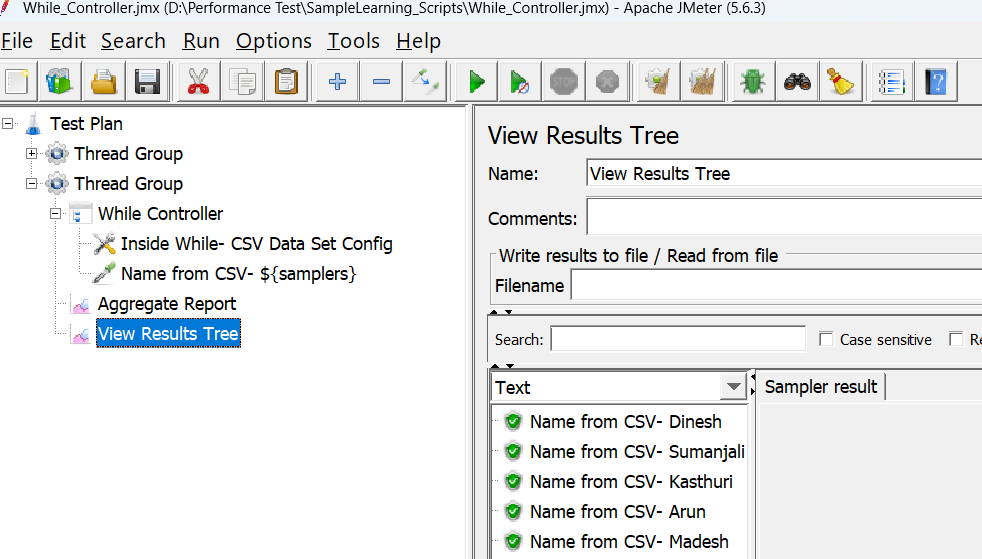
Also, we can use for other purpose like;

Configure the CSV data Set



**Recycle: Fase ; Stop Thread: True;**

It will make the while loop to run All the data in **the csv file till EOF**, even you have mentioned **thread count as 1**.



Make sure CSV is inside of the while controller.

**Multiple Thread Group:**

**If you have multiple thread group, we can able to run a particular thread in both GUI and NON GUI Mode**

**GUI Mode:**

* Give the thread count whatever the count you want to run in one of the thread group.
* And whichever thread you don’t want to run **give count as 0**.

**Non-GUI Mode:**

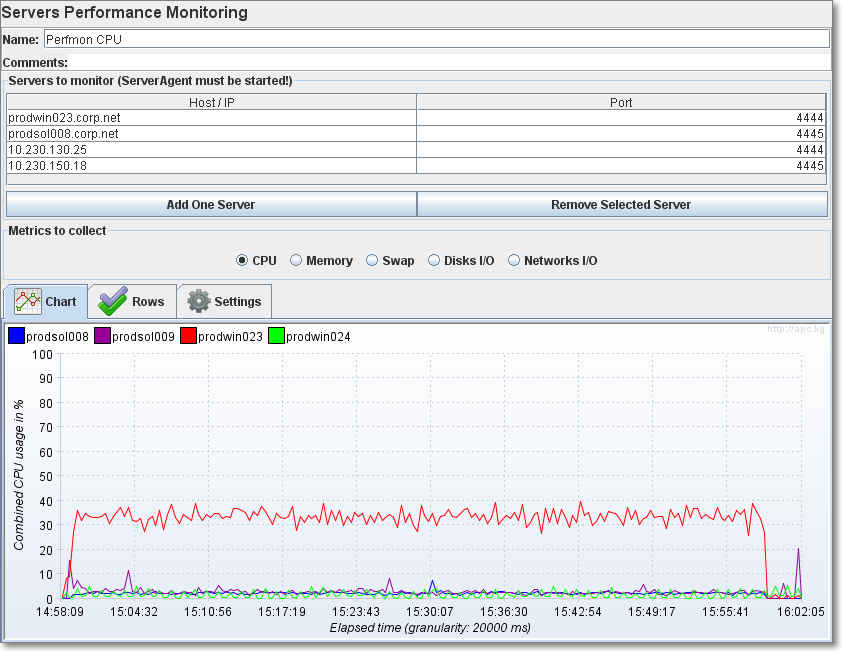
Prerequisite: Give the property function naming in thread group for All the thread group you have. **i.e. {\_\_P(threadCount,1)}**

**jmeter -n -t "location of JMX script " -l "location of csv or JTL file" -Jusers1=2 -Jusers2=0**

**Servers Performance Monitoring** -**Perform Metrics Collector:**

During a load test, it is important to know the health of the servers loaded. It is also nice to see if you are targeting a cluster if the load is correctly dispatched. To address this, the plugin package now supports server monitoring! Using it, you can monitor [**CPU, Memory utilization, Swap, Disks I/O and Networks I/O**](https://github.com/undera/perfmon-agent/blob/master/README.md#supported-metrics)**on**[**almost All platforms**](http://support.hyperic.com/display/SIGAR/Home#Home-overview)**!**

**UI of Perfmon Listener in JMeter**

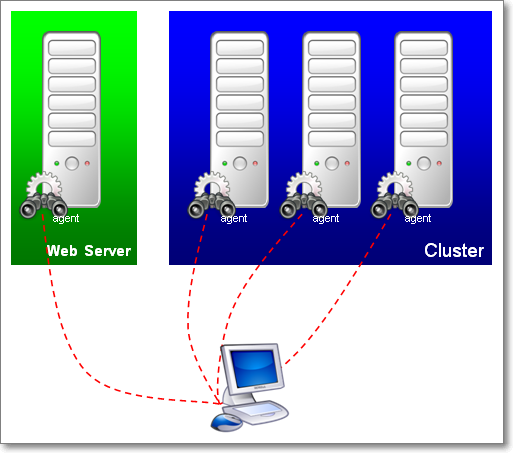


In client side where the application was stored there, we Need to connect with **Server Agent.**

JMeter cannot retrieve by default server metrics except Tomcat ones. To overcome this situation, we have developed a server agent which will get performance data for JMeter. The agent uses the [SIGAR](http://support.hyperic.com/display/SIGAR/Home) open source library. It is composed of a Java common part and native libraries per OS.

**Prerequisite: InstAll Perform Metrics Collector plugin and server agent**

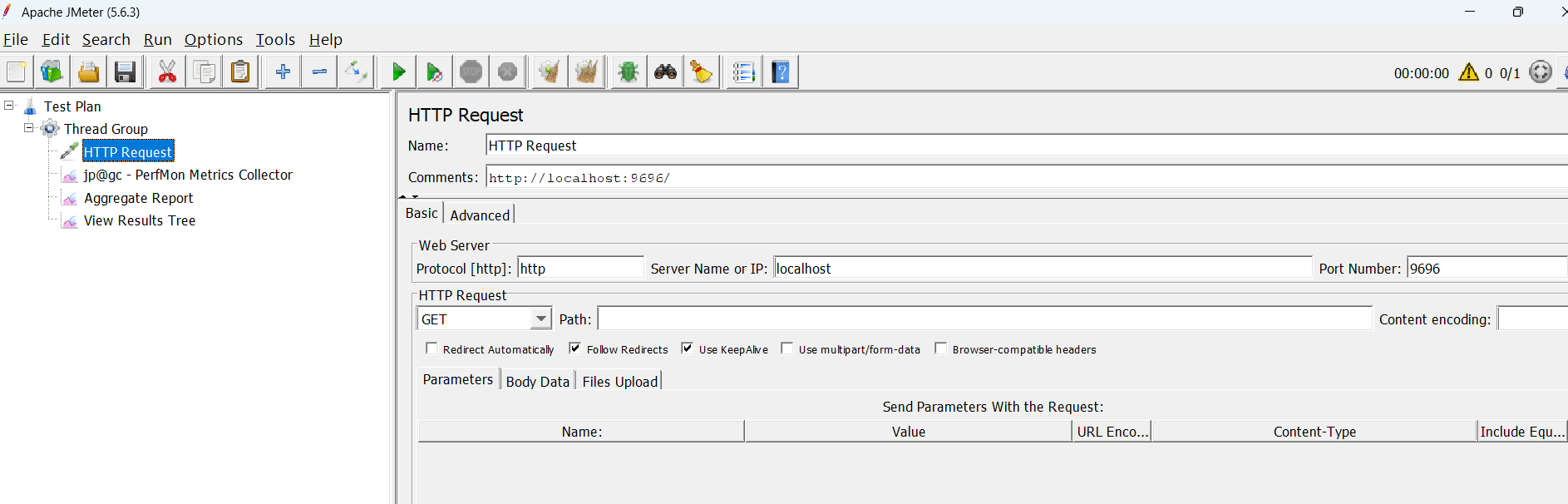
This is how the Server Agent will work:



**Step 1**: Open the startAgent.bat file (it wil show port number as well)

**Step 2**: Open another CMD and give telnet <ip Address of the server> port numer 4444.

**Step 3**: Create a simple test, using local host Jenkins –(For me its not worked)



**Step 4:** Run and Validate, you can able to see the server **CPU and Memory utilization clearly, by the test runs.**

**JMeter- Test plan checker**

Is a tool (available plugin) is used to check the JMX file details and whether its fine or not

Once it downloaded open the TestPlanCheck.bat file and you can check about the JMX file in the UI mode.

* TestPlanCheck.bat --jmx "Thread Group.jmx"
* TestPlanCheck.bat --jmx "Thread Group.jmx" --stas
* TestPlanCheck.bat --jmx "Thread Group.jmx" --tree-dump
* TestPlanCheck.bat --jmx "Thread Group.jmx" --stats --tree-dump