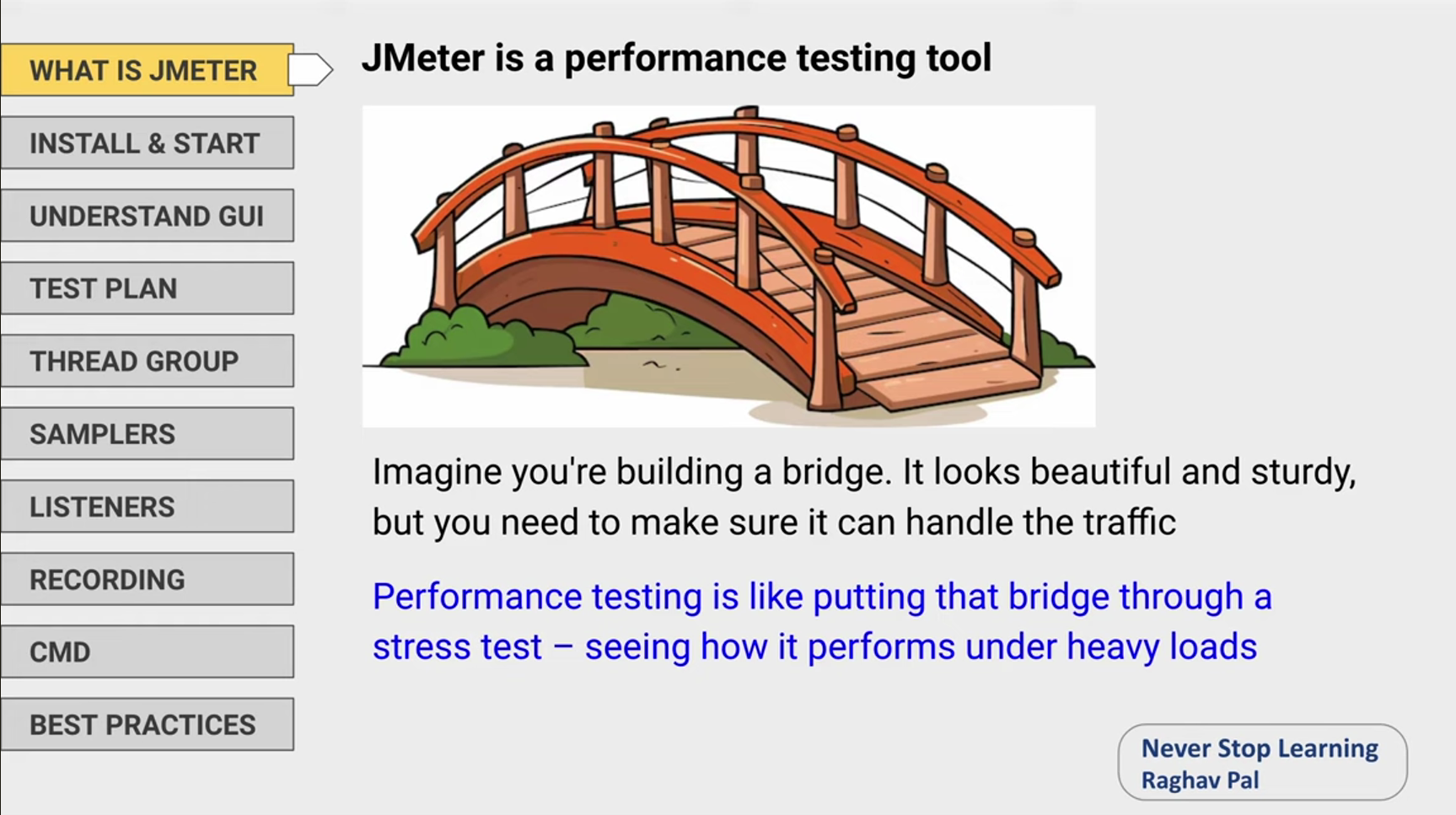
**JMeter**



**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"**

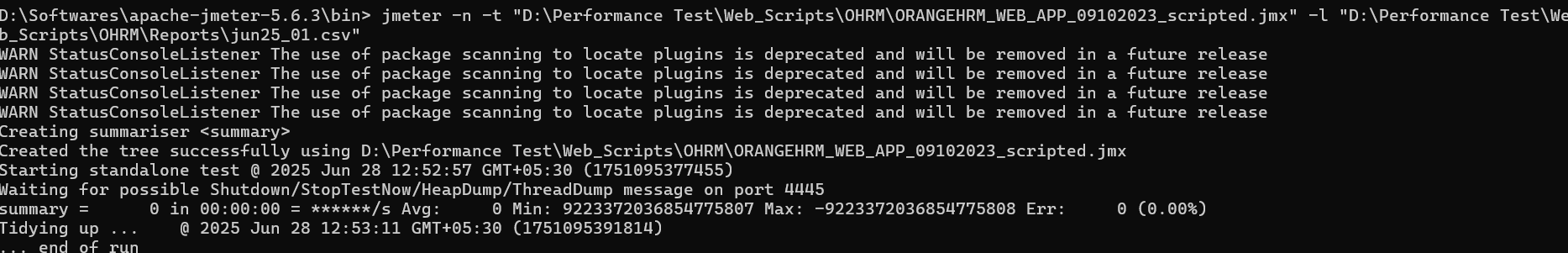
**Explanation:**

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

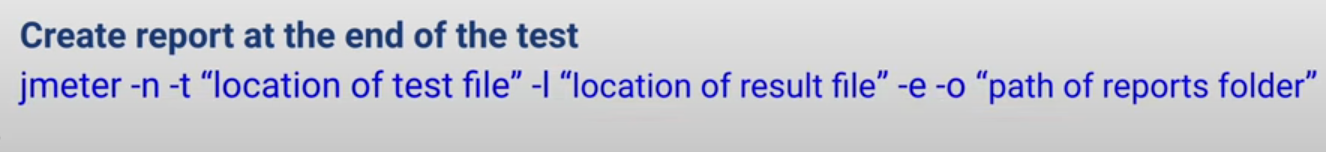
**Example:**

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:**

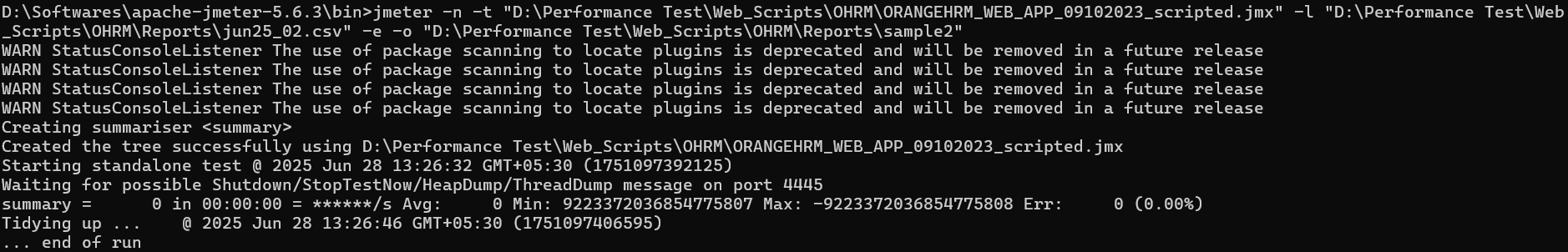


**jmeter -n -t "location of test file" -I "location of result file" -e -o "path of reports folder"**

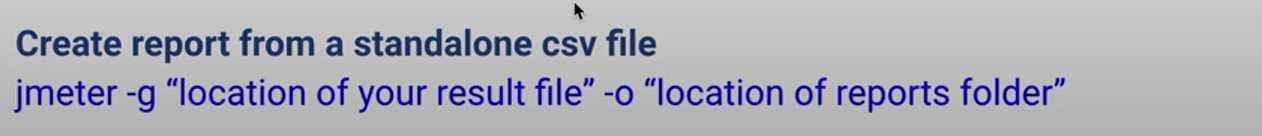
**Example:**

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,**

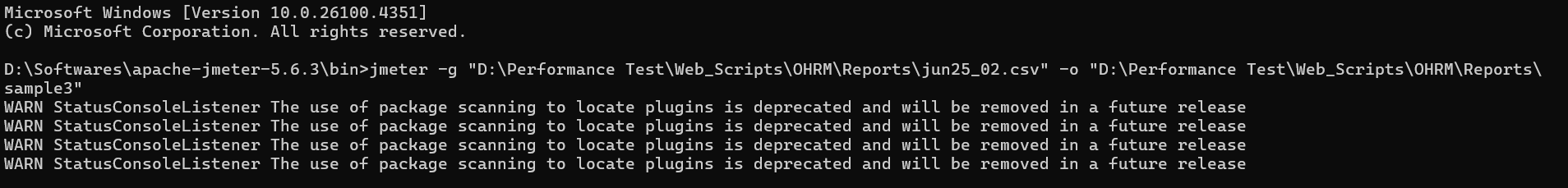
****

**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:**

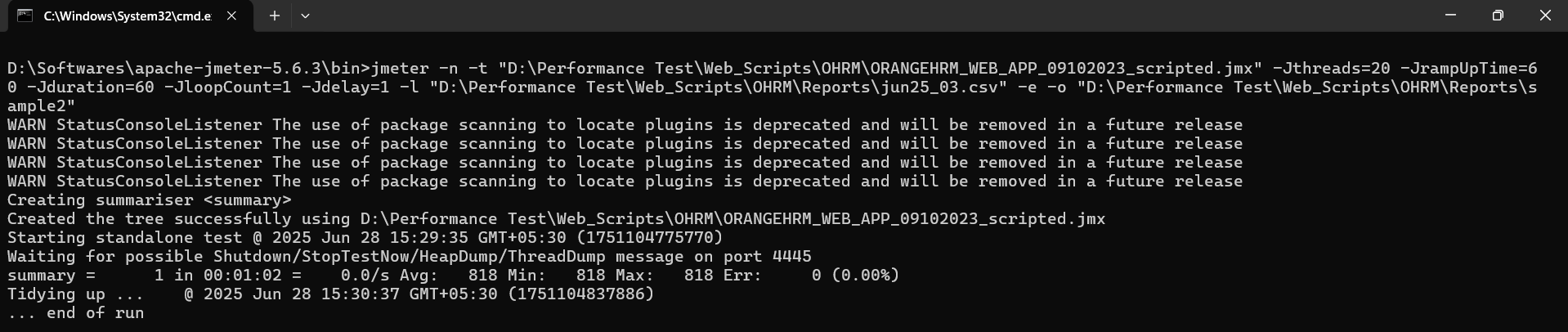
|  |  |
| --- | --- |
| Thread Count | ${\_\_P(threads,1)} |
| Ramp-up time | ${\_\_P(rampUpTime,1)} |
| Loop Count | ${\_\_P(loopCount,1)} |
| Duration | ${\_\_P(duration,60)} |
| Start-up Delay | ${\_\_P(delay,1)} |
| Access Token | ${\_\_P(AccessToken,1)} |

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**

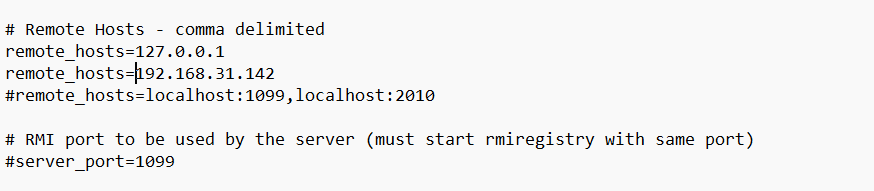
**Step 1 :** SetUp Master added remote system’s ip in jmeter.properties

**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

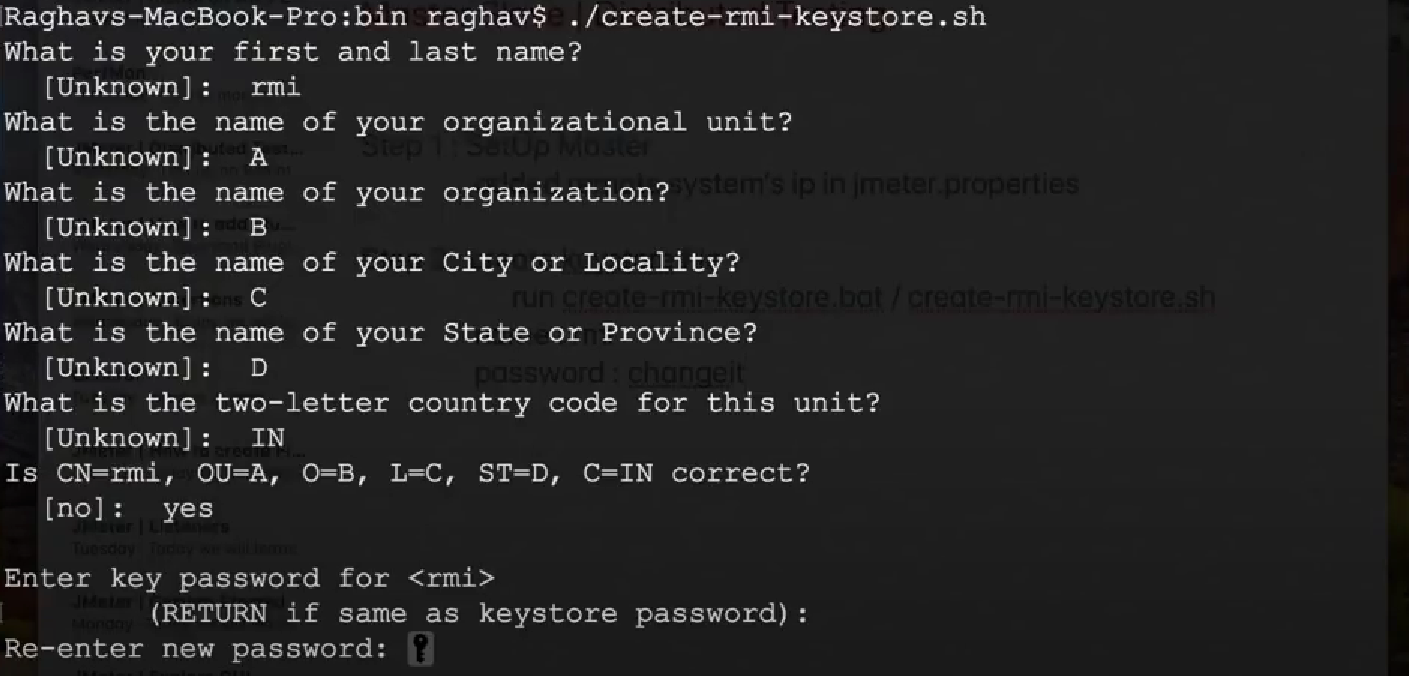
**Illustration:**

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

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

Open the jmeter server batch file as well while running

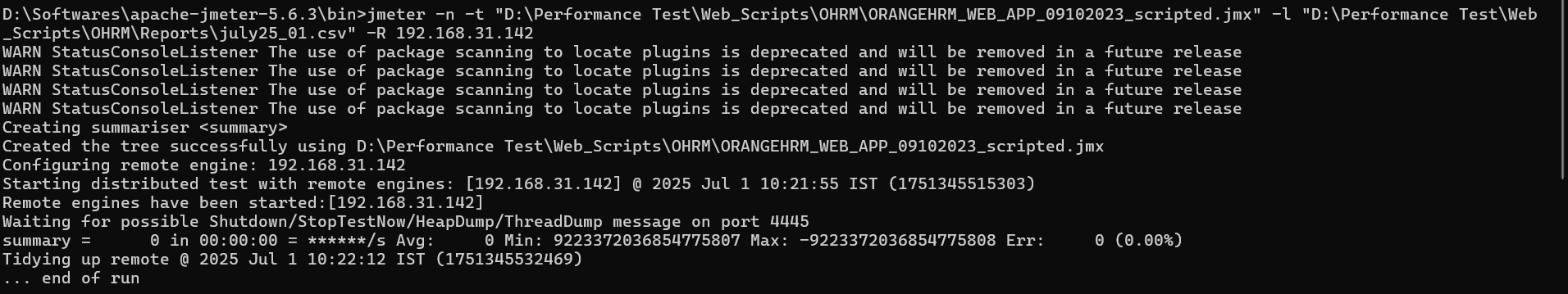
**Command:** Syntax

**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**

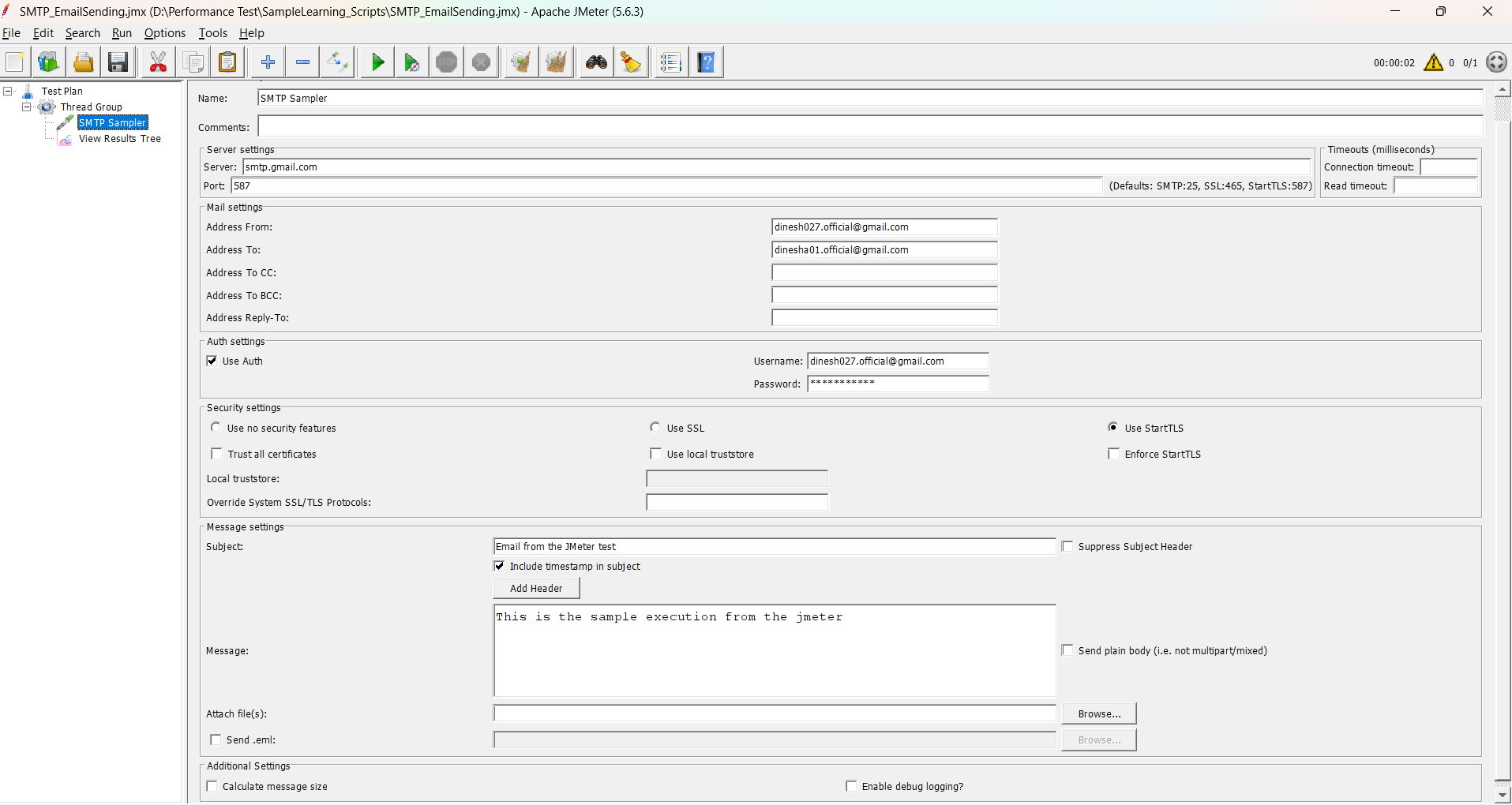
**Step 1** : Download javamail jar and put in jmeter lib folder - restart jmeter**.**

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

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

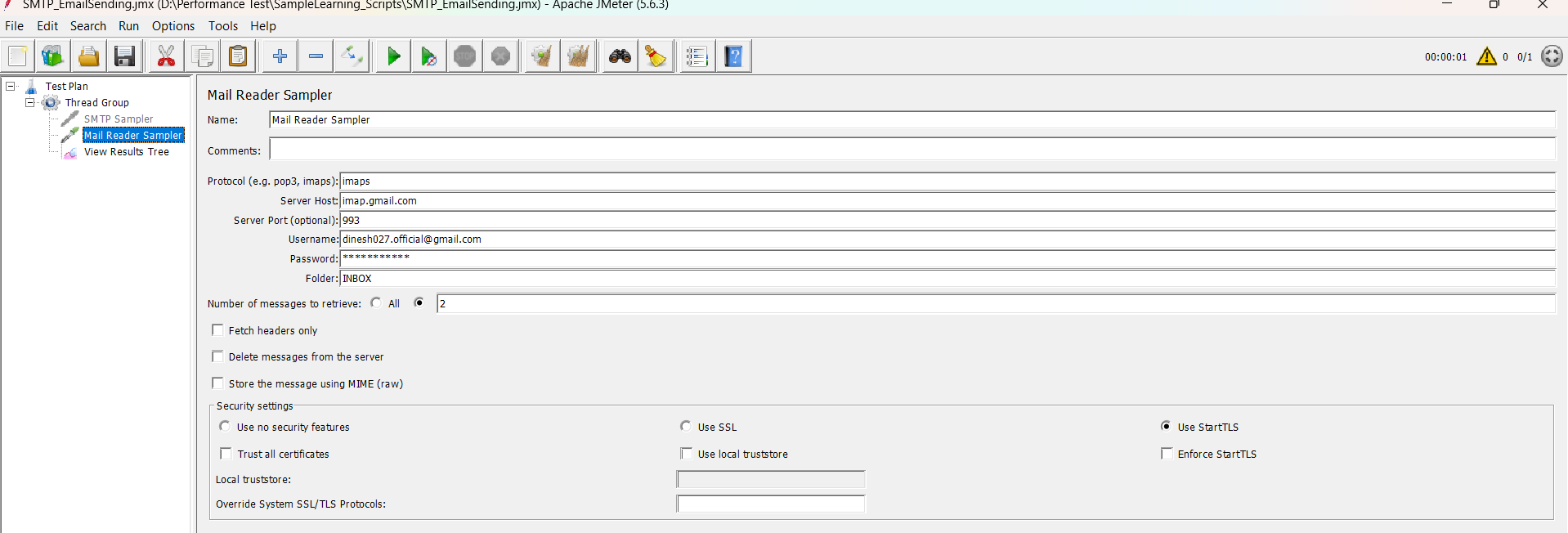
**Step 4 :** Add listeners to view test results

**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.



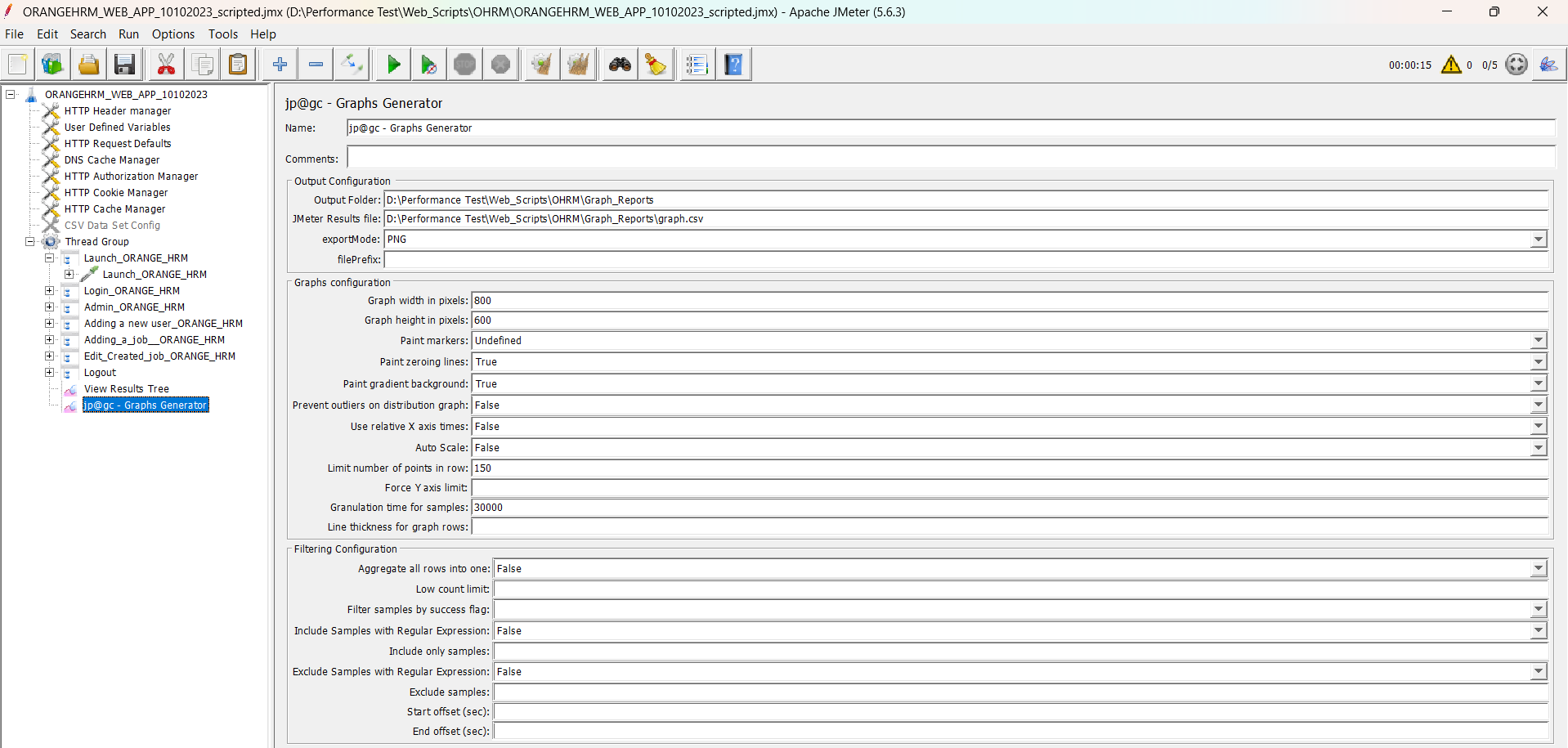
1. **Graphical Results- Graph Generator.**

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:**

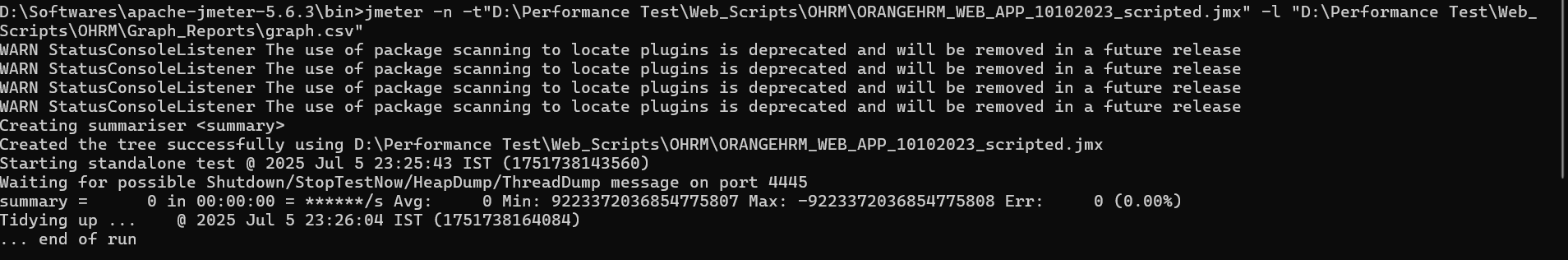
**Example:**

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

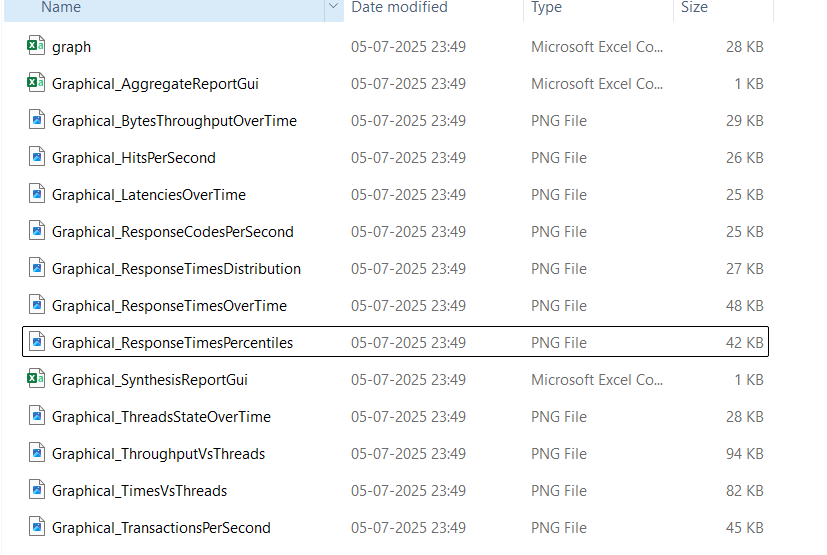
**Command:**

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"

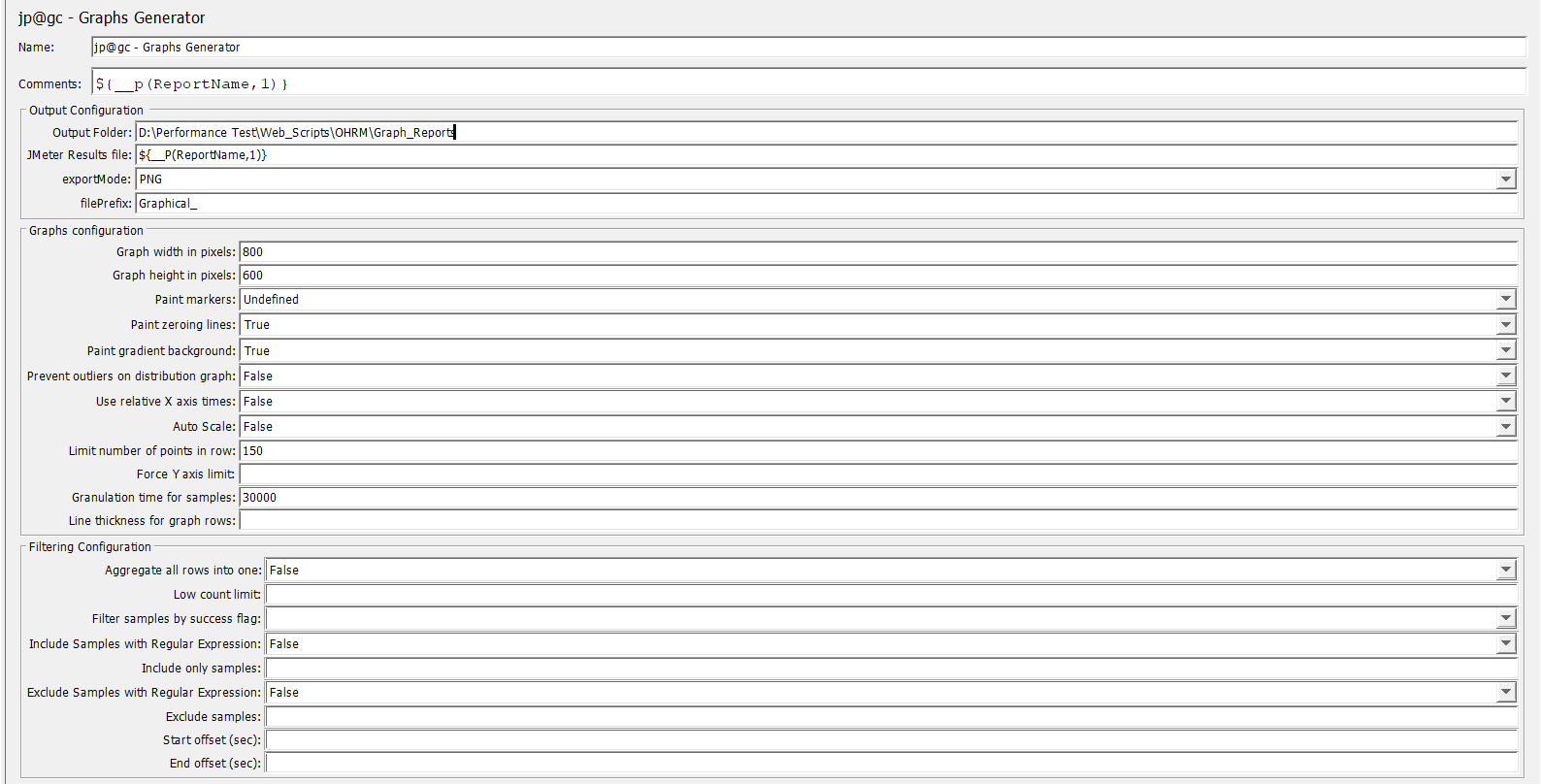
**Execution in CMD:**

****

Reports Created:



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



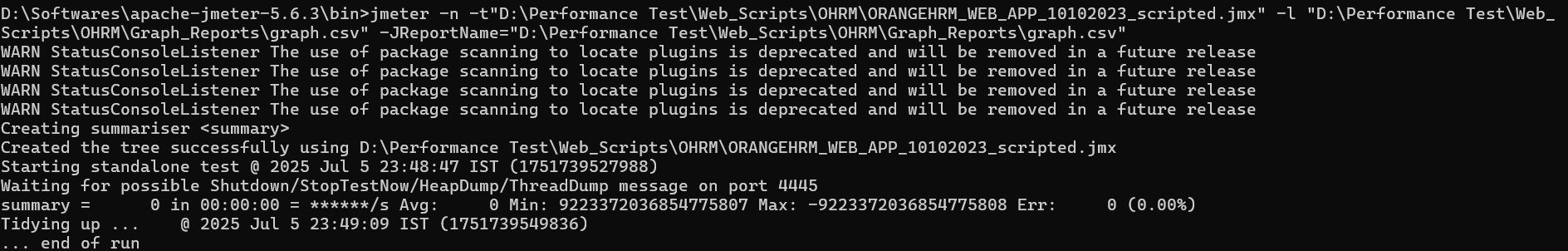
**Example:**

**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)**

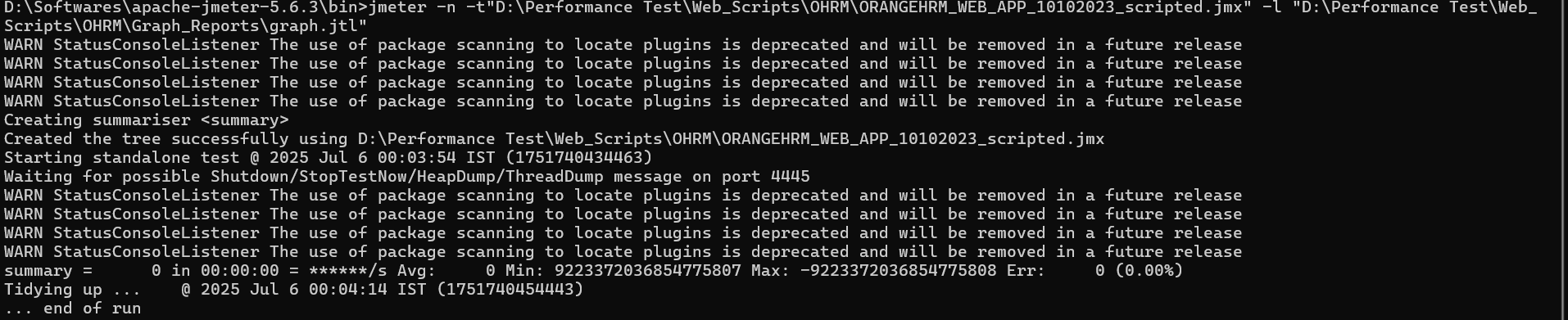
**Example:**

**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

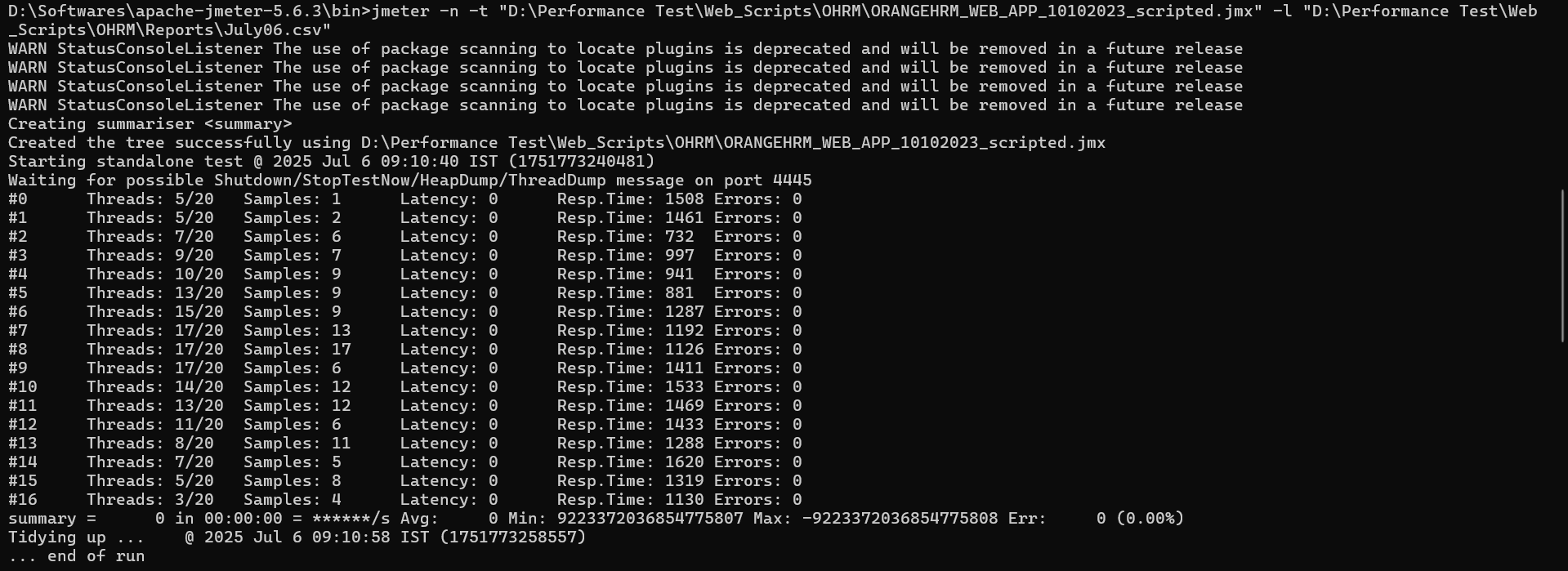
**Example:**

**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:**

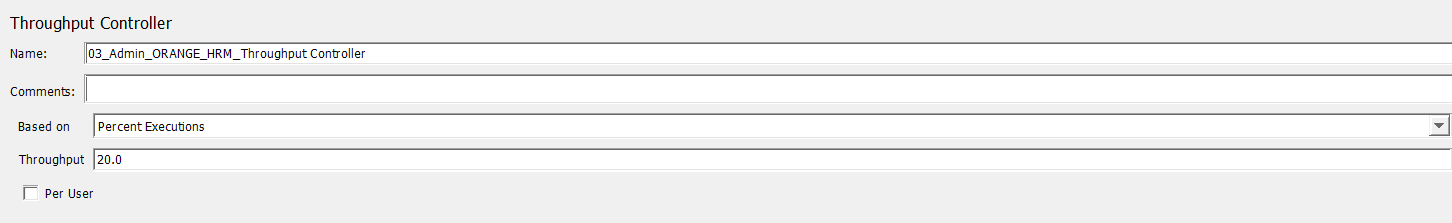
1. **Throughput Controller**

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.**