**TRAINING APACHE JMETER**

**MỤC LỤC**

[**1.** **Overview** 4](#_Toc39646369)

[**2.** **Download Apache Jmeter** 5](#_Toc39646370)

[**3.** **Getting Started** 6](#_Toc39646371)

[**3.1.** **Overview** 6](#_Toc39646372)

[**3.1.1.** **Xây dựng Kịch bản Test** 6](#_Toc39646373)

[**3.1.2.** **Chạy Load Test** 7](#_Toc39646374)

[**3.1.3.** **Phân tích Load Test** 7](#_Toc39646375)

[**3.2.** **Running JMeter** 8](#_Toc39646376)

[**3.2.1.** **Tạo Test Plan từ Template** 8](#_Toc39646377)

[**3.2.2.** **Sử dụng JMeter với Proxy** 9](#_Toc39646378)

[**3.2.3.** **CLI Mode (Non – GUI Mode)** 10](#_Toc39646379)

[**3.2.4.** **Server Mode** 14](#_Toc39646380)

[**4.** **JMeter GUI** 15](#_Toc39646381)

[**5.** **Test Plan** 18](#_Toc39646382)

[5.1. Test Plan là gì 18](#_Toc39646383)

[5.2. Các Element của một Test Plan 18](#_Toc39646384)

[5.2.1. Thread Group 18](#_Toc39646385)

[5.2.2. Controllers 20](#_Toc39646386)

[5.2.3. Listener 32](#_Toc39646387)

[5.2.4. Configuration Elements 36](#_Toc39646388)

[5.2.5. Timers 37](#_Toc39646389)

[5.2.6. Assertions 44](#_Toc39646390)

[5.2.7. Processor: Pre – processor and Post – Processor 51](#_Toc39646391)

[**6.** **Performance Testing** 57](#_Toc39646392)

[**7.** **JMeter Distributed (Remote) Testing: Master Slave Configuration** 64](#_Toc39646393)

[**7.1.** **Distributed Testing** 64](#_Toc39646394)

[**7.2.** **Remote Test Example** 64](#_Toc39646395)

[**8.** **HTTP Proxy Server In JMeter: Record Example Script** 66](#_Toc39646396)

[**8.1.** **Setting HTTP Proxy server** 67](#_Toc39646397)

[**9.** **Recording Web Application with BlazeMeter** 80](#_Toc39646398)

[**9.1.** **BlazeMeter** 80](#_Toc39646399)

[**9.2.** **Recording Web Application with BlazeMeter** 81](#_Toc39646400)

[**10.** **JMeter Config Elements – Variables** 84](#_Toc39646401)

[**10.1.** **User Defined Variables** 84](#_Toc39646402)

[**10.2.** **CSV Data Set Config** 87](#_Toc39646403)

[**10.3.** **JDBC Connection Cofiguration** 90](#_Toc39646404)

[**11.** **Sử dụng hàm trong JMeter** 93](#_Toc39646405)

**TÀI LIỆU THAM KHẢO**

1. Apache JMeter: <https://jmeter.apache.org/>
2. Quick Guide JMeter Tutorials: <https://www.guru99.com/introduction-to-jmeter.html>
3. JMeter VN – Discuss & Sharing about JMeter: <https://jmetervn.com/category/basic/>
4. Start Learning JMeter with sample Test Cases: <https://medium.com/chaya-thilakumara/start-learning-jmeter-with-sample-test-cases-2dc2a4963b62>

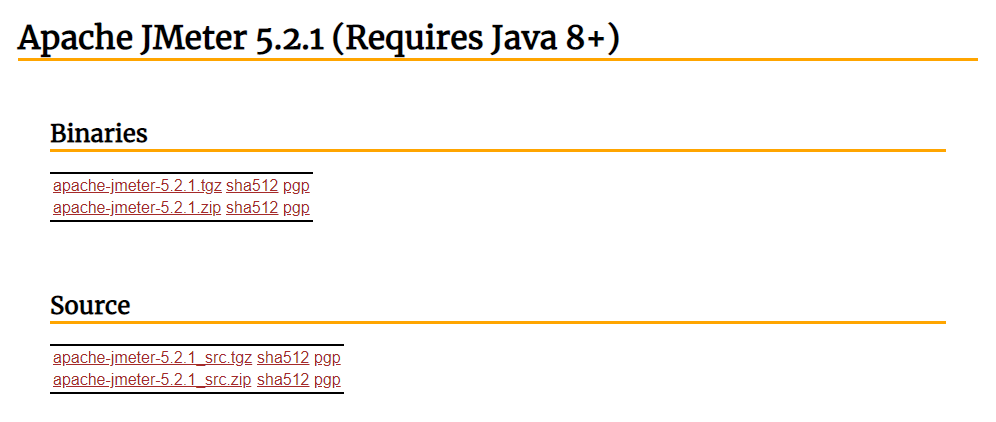
**NỘI DUNG**

# **Overview**

* Apache Jmeter là một phần mềm mã nguồn mở, viết bằng ngôn ngữ Java để chạy test các chức năng và đo hiệu suất của chương trình. Apache Jmeter được thiết kế cơ bản để kiểm tra các Web Applications nhưng cũng đã mở rộng sang các chức năng kiểm tra khác.
* Apache Jmeter có thể sử dụng để test hiệu suất cả web động và web tĩnh. Nó có thể dùng để mô phỏng một nguồn tải lớn trên 1 server, nhóm server, mạng hoặc đối tượng để kiểm tra khả năng của các đối tượng đó, hoặc để phân tích tổng quan hiệu suất dưới nhiều loại nguồn tải khác nhau.
* Apache Jmeter bao gồm các khả năng:
  + Khả năng tải và thực hiện kiểm tra nhiều loại ứng dụng/server/protocol:
    - 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
  + Các đặc trưng đầy đủ của 1 Test IDE như recording (trình duyệt hoặc native applications), build và debug
  + Command – line mode: test các hệ điều hành tương thích với Java (Linux, Windows, Mac OSX,…)
  + Report
  + Khả năng trích xuất dữ liệu từ hầu hết các định dạng phản hồi phổ biến (HTML, JSON, XML,…)
  + Khả năng thực thi đa luồng
  + Sử dụng bộ nhớ Cache để phân tích/chạy lại các kết quả test offline

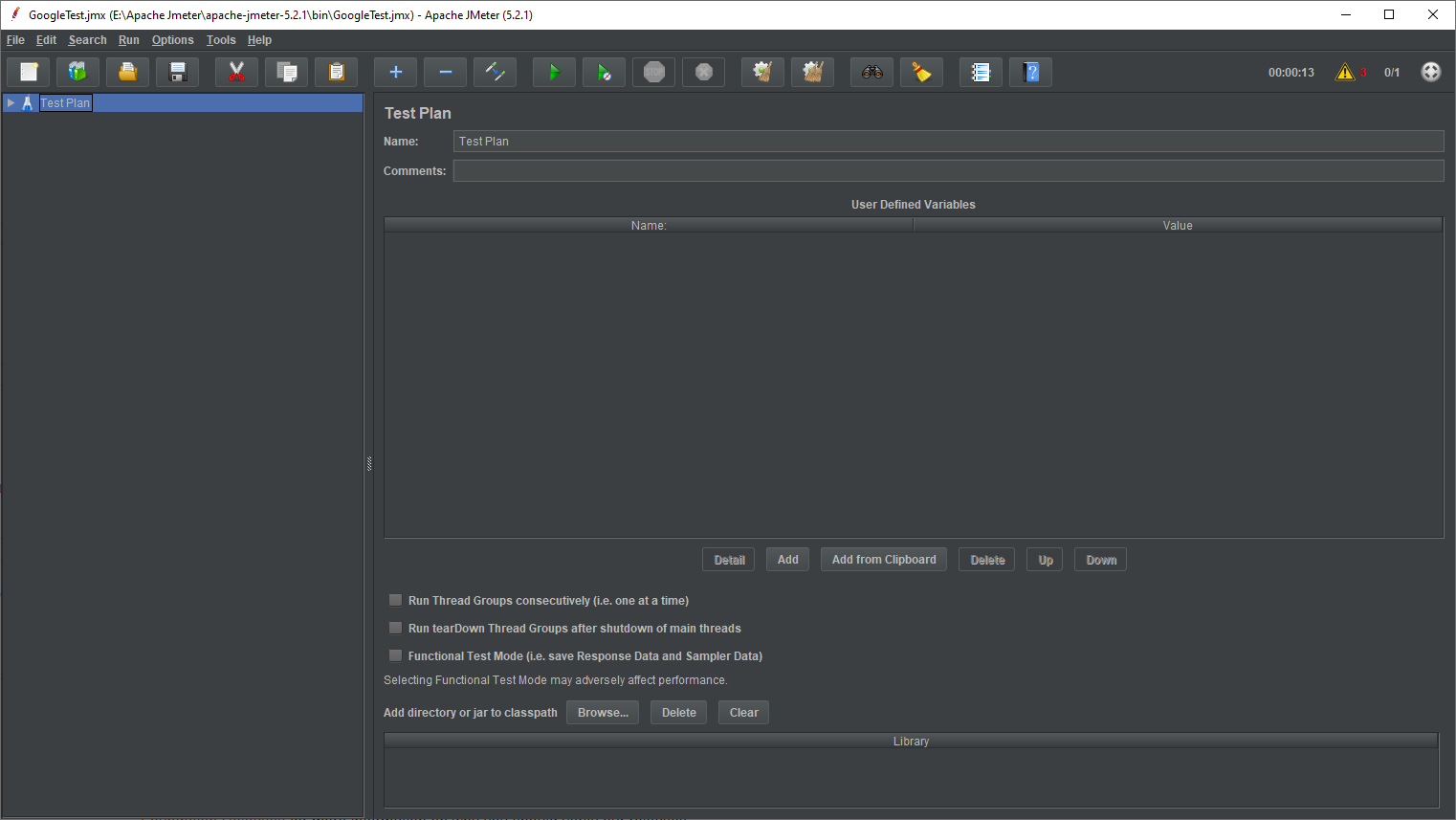
# **Download Apache Jmeter**

* Truy cập đường link download: <https://jmeter.apache.org/download_jmeter.cgi>
* Chọn nguồn download thích hợp (Yêu cầu Java 8+)



Hình 1: Download Apache Jmeter

* Sau khi download thành công, tiến hành giải nén và chạy Jmeter bằng cách chạy file ApacheJMeter.jar trong thưc mục **bin**



Hình 2: Apache JMeter GUI

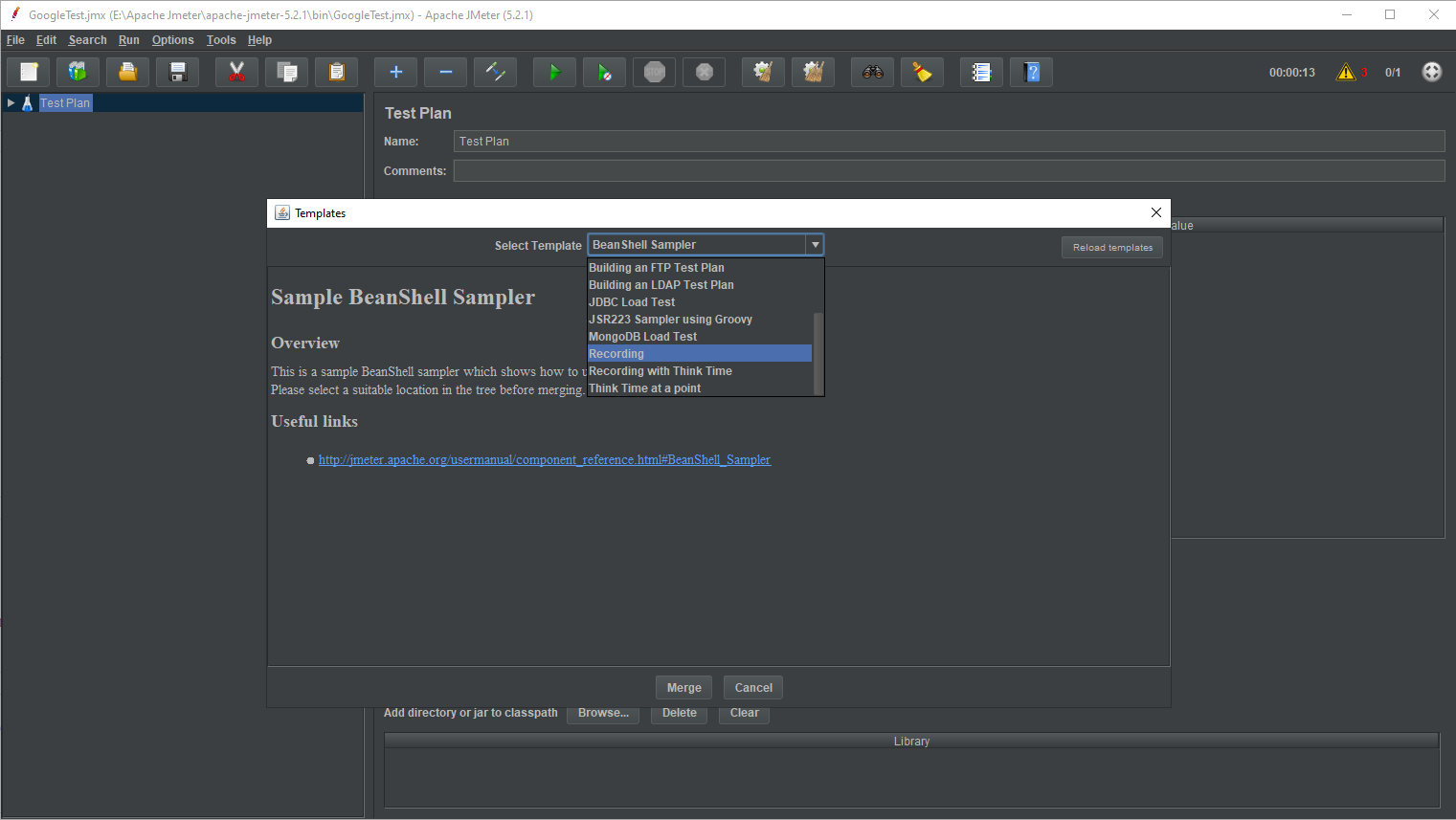
# **Getting Started**

## **Overview**

Khi sử dụng Jmeter, ta thường thực hiện theo một chu trình dưới đây:

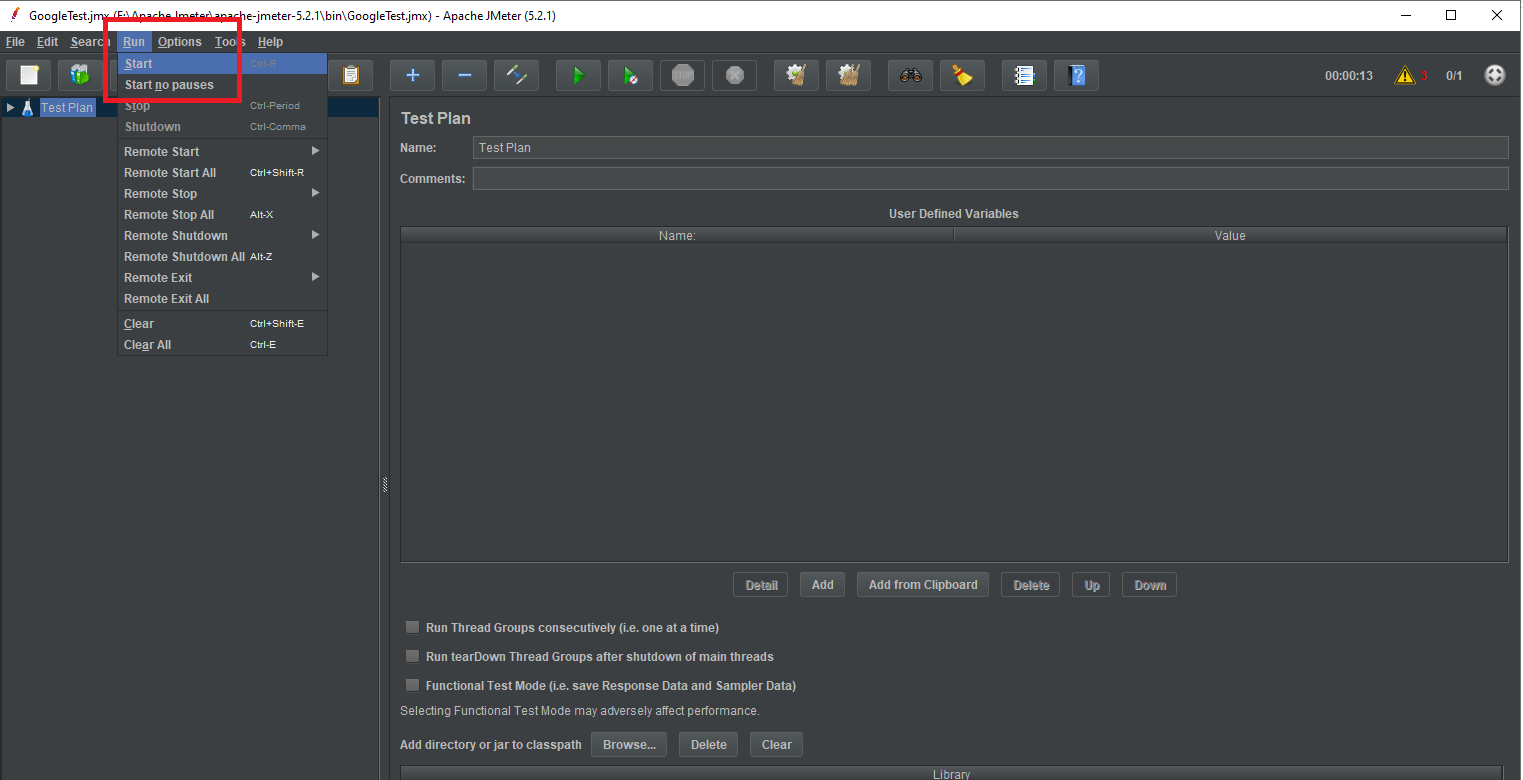
### **Xây dựng Kịch bản Test**

* Để thực hiện, ta chạy Run Jmeter in GUI Mode
* Sau đó ta có thể lựa chọn record ứng dụng từ trình duyệt hoặc native application bằng cách chọn File 🡪 Templates 🡪 Recording



Hình 3: Recording Template

* Ta cũng có thể tự xây dựng một Test plan thủ công, để thực hiện điều đó cần thực hiện các bước cần thiết được trình bày ở phần
* Để debug Test Plan:
  + Chọn Run 🡪 Start no pauses
  + Chọn Run 🡪 Start hoặc dùng phím tắt Ctril + R
  + Validate trên Thread Group



Hình 4: Debug Test plan

* Chi tiết các bước để xây dựng Test plan sẽ được nói đến ở mục…

### **Chạy Load Test**

* Sau khi xây dựng xong Test Plan, ta chay Load Test. Bước đầu tiên là cấu hình môi trường chạy Jmeter:
  + Đảm bảo thích hợp CPU máy, bộ nhớ và mạng
  + Điều chỉnh Hệ điều hành
  + Cài đặt Java phiên bản mới nhất, hỗ trợ Jmeter
  + Tăng kích thước Java Heap. Mặc định Jmeter chạy với kích thước heap là 1 GB, điều này có thể không đủ cho việc test và phụ thuộc vào test plan và số lượng thread ta muốn thực thi
* Sau khi mọi thứ đã sẵn sang, ta có thể sử dụng CLI mode (Command – line mode hay còn gọi là Non – GUI mode) để chay Load Test
* Ta có thể tạo 1 file CSV hoặc XML chứa các kết quả test khi chay Load Test bằng CLI mode. Jmeter sẽ tạo một HTML report sau khi chạy xong Load Test. Jmeter sẽ cung cấp một tổng thể của load test khi thực hiện.
* Ta cũng có thể có một kết quả test với thời gian thực tế trong suốt quá trình test bằng cách sử dụng Backend Listener

### **Phân tích Load Test**

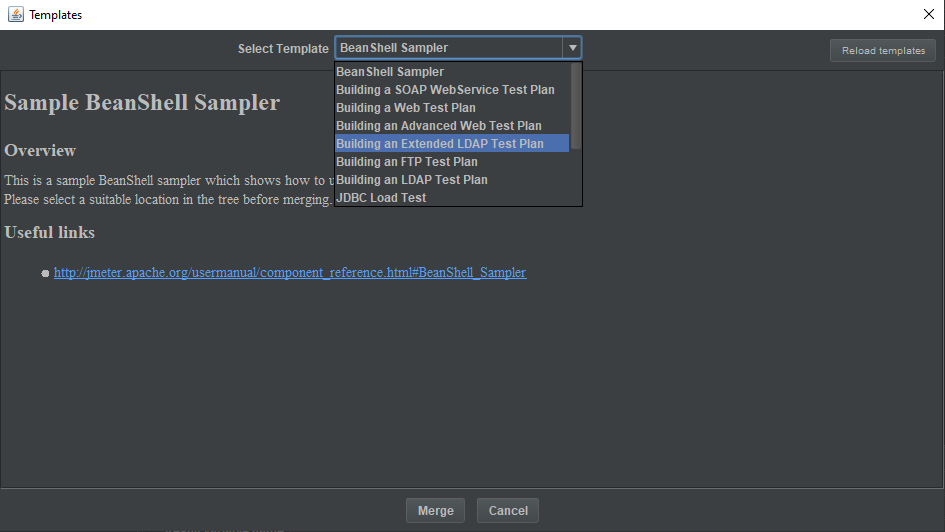
* Khi hoàn thành Load Test, ta sử dụng HTML report để phân tích kết quả.

## **Running JMeter**

* Để chay JMeter, chay file jmeter.bat (Windows) hoặc jmeter (Unix) trong thư mục bin.
  + GUI Mode chỉ nên dùng để tạo test script
  + CLI Mode phải được dùng để chay test
* Có một số script trong thư mục bin mà ta có thể lựa chọn để sử dụng:
  + Jmeter.bat: chạy JMeter (GUI mode)
  + Jmeter.cmd: chay JMeter dùng cửa sổ console (GUI mode)
  + Jmeter-n.cmd: thả một file JMX tại đây để chay một CLI mode test
  + Jmeter-n-r.cmd: thả một file JMX tại đây để chạy CLI mode test từ xa
  + Jmeter-t.cmd: thả một file JMX tại đây để load nó dưới GUI mode
  + Jmeter-server.bat: khởi chạy Jmeter ở chế độ server
  + Mirror-server.bat: chay Jmeter Mirror Server ở CLI mode
  + Shutdown.cmd: dừng CLI mode
  + Stoptest.cmd: dừng đột ngột CLI mode

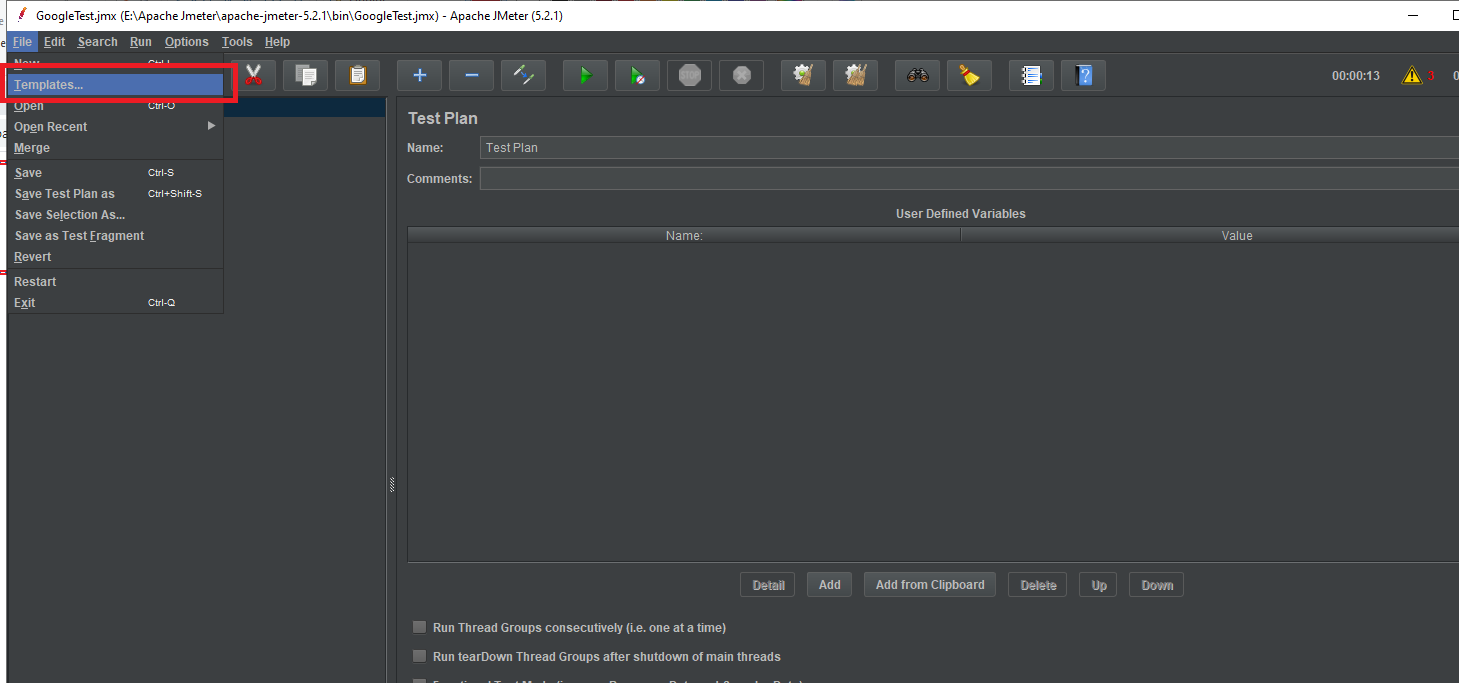
### **Tạo Test Plan từ Template**

* Trong Template của JMeter hỗ trợ tạo nhanh chóng nhiều loại Test Plan khác nhau:

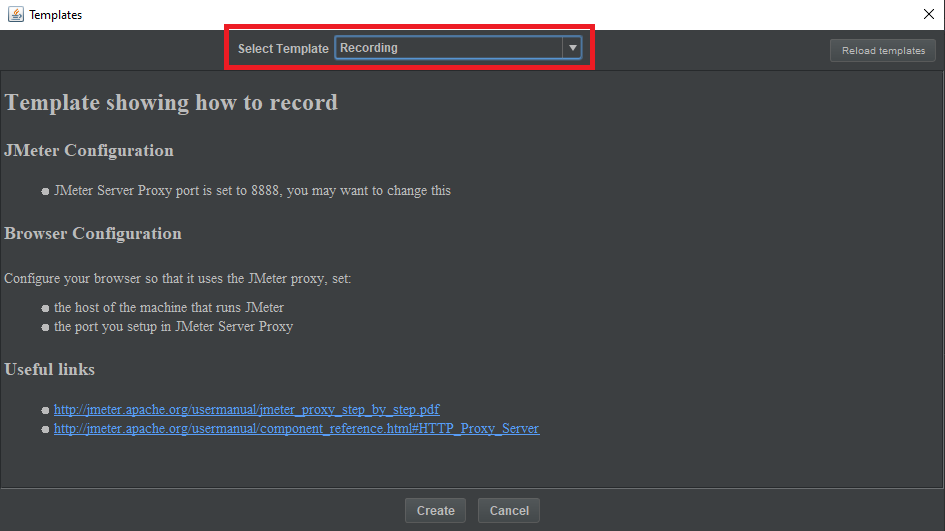


Hình 5:Create Test Plan using Template

* Để tạo Test plan bằng cách record từ template, ta chọn File 🡪 Templates 🡪 Recording 🡪 Create



Hình 6: Select Templates



Hình 7: Select Recording

### **Sử dụng JMeter với Proxy**

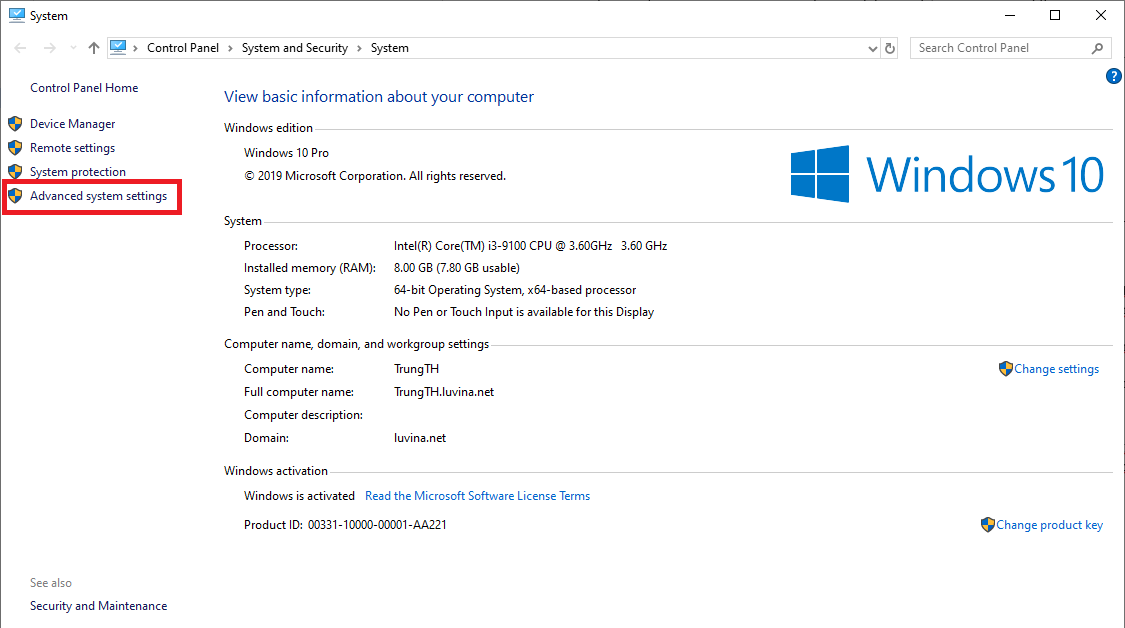
* Trường hợp test from behind a firewall/proxy server, ta cần cung cấp firewall/proxy hostname và port number cho Jmeter. Để làm điều này, ta chay jmeter.bat từ commond line với các parameter sau:
  + **–E** [proxy scheme to use - optional - for non-http]
  + **–H** [proxy server hostname or ip address]
  + **–P** [proxy server port]
  + **–N** [non proxy hosts] (e.g. **\*.apache.org|localhost**)
  + **–u** [username for proxy authentication - if required]
  + **–a** [password for proxy authentication - if required]
* Ví dụ:

jmeter -E https -H my.proxy.server -P 8000 -u username -a password -N localhost

* Nếu proxy được cung cấp, JMeter sẽ set cấu hình các property sau:
  + http.proxyScheme
* Nếu proxy host và port được cung cấp, JMeter sẽ set cấu hình các property sau:
  + http.proxyHost
  + http.proxyPort
  + https.proxyHost
  + https.proxyPort
* Nếu 1 danh sách nonproxy host được cung cấp, JMeter sẽ set các property sau:
  + http.nonProxyHosts
  + https.nonProxyHosts
* Bởi vậy nếu không muốn set cả http và https proxy, ta có thể lựa chọn các property thích hợp trong system.properties thay vì sử dụng các parameter ở command line
* Việc ứng dụng proxy trong JMeter sẽ được trình bày cụ thể ở mục 11 của tài liệu này.

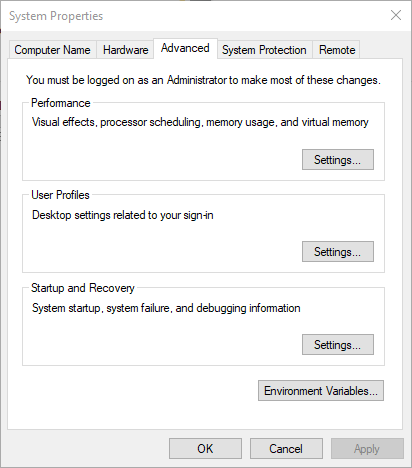
### **CLI Mode (Non – GUI Mode)**

* *Để hiểu được chế độ này, ta cần hiểu rõ cách sử dụng JMeter, thao tác với các element và record web, vì vậy mục 3.2.4 này có thể tìm hiểu sau khi nắm được các nội dung kể trên.*
* Khi load Test, ta phải chạy JMeter ở chế độ này để lấy kết quả tối ưu. Sử dụng các command sau:
  + **–n** [run JMeter in CLI mode]
  + **–t** [name of JMX file that contains the Test Plan].
  + **–l** [name of JTL file to log sample results to].
  + **–j** [name of JMeter run log file].
  + **–r** Run the test in the servers specified by the JMeter property "**remote\_hosts**"
  + **–R** [list of remote servers] Run the test in the specified remote servers
  + **–g** [path to CSV file] generate report dashboard only
  + **–e** generate report dashboard after load test
  + **–o** output folder where to generate the report dashboard after load test. Folder must not exist or be empty. The script also lets you specify the optional firewall/proxy server information:
  + **–H** [proxy server hostname or ip address]
  + **–P** [proxy server port]
* Nếu property jmeterengine.stopfail.system.exit đang được set ở true (mặc định là false) thì JMeter sẽ gọi System.exit trường hợp không thể dừng tất cả các luồng. Thông thường điều này không cần thiết.
* Để có thể load Test với chế độ Non – GUI Mode, ta cần thiết lập môi trường cho JMeter trước:
  + Truy cập Control Panel\System and Security\System, chọn Advanced System Settings



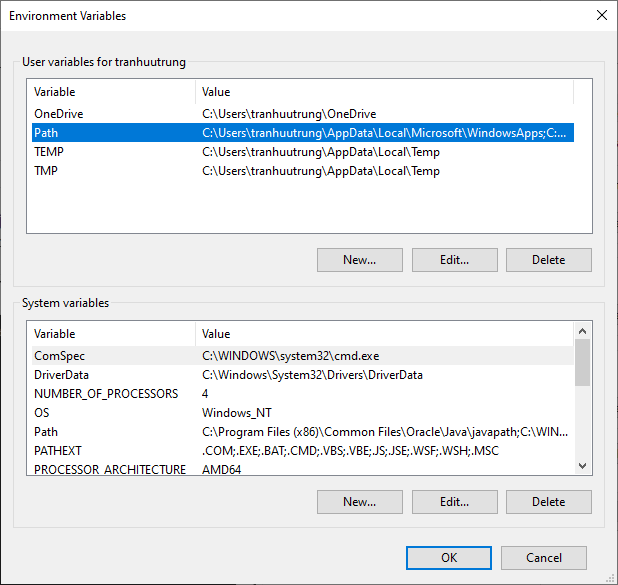
Hình 8: Setting JMeter Environment

* + Click Environment



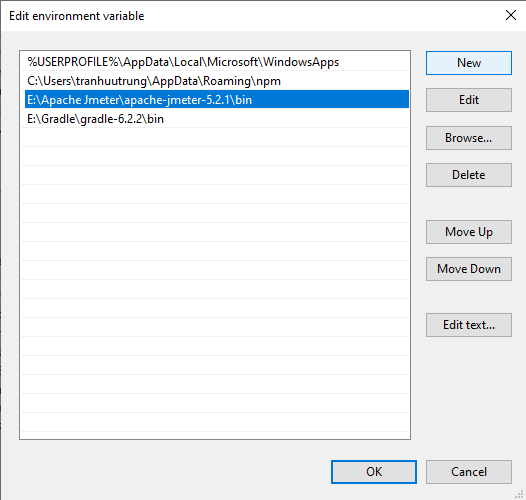
Hình 9:Setting JMeter Environment

* + Chọn Path, click Edit



Hình 10:Setting JMeter Environment

* + Click New, thêm đường dẫn tới thư mục bin của JMeter và click OK.



Hình 11: Setting JMeter Environment

* + Kiểm tra xem JMeter đã được thiết lập môi trường thành công chưa bằng cách mở command prompt và gõ: jmeter –v. Nếu chưa thành công thì cần thiết lập lại.



Hình 12: JMeter Environment is set successfully

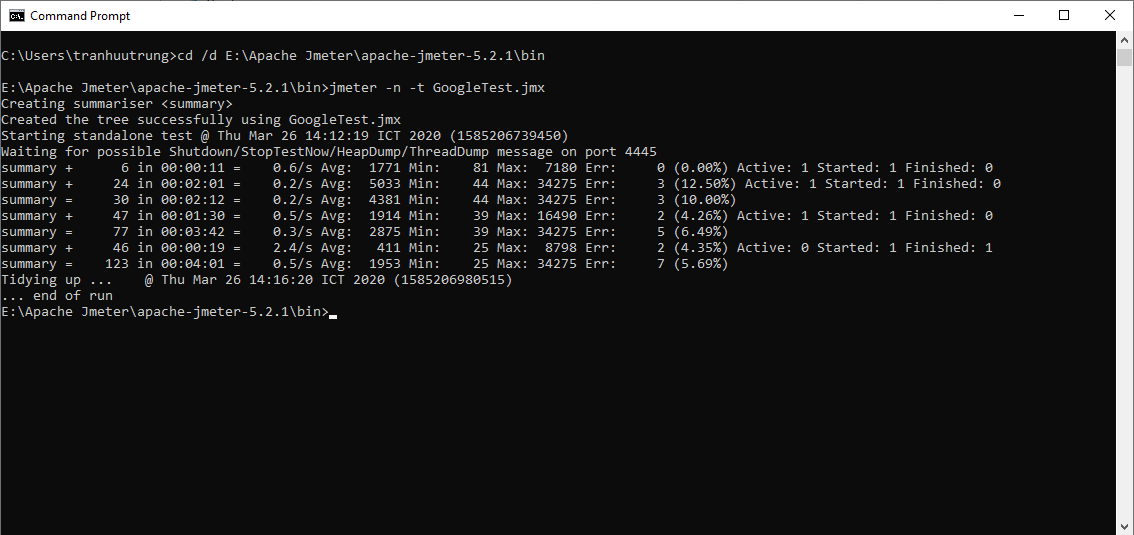
* Sau khi thiết lập môi trường, ta đã có thể chạy load Test ở chể độ Non- GUI:
  + Mở command prompt, áp dụng các câu lệnh đã giới thiệu ở trên, ví dụ ta cần load file google\_test.jmx trong thư mục bin của JMeter (file google\_test.jmx có thể tạo được sau khi tìm hiểu mục 5 trở đi của tài liệu).
  + Trỏ đường dẫn thư mục hiện tại trên cmd tới thực mục bin của JMeter:

cd /d E:\Apache Jmeter\apache-jmeter-5.2.1\bin

* + Chay load Test:

jmeter -n -t GoogleTest.jmx

* + Kết quả test:



Hình 13: Test Results

### **Server Mode**

* Dùng cho việc Chia các phần Test. Chạy JMeter bằng Server Mode trên nút điều khiển và điều khiển server từ GUI. Ta cũng có thể dùng CLI Mode để điều khiển test. Để bắt đầu server, chạy file jmeter-server.bat ở mỗi server host
* Script cũng cho phép chỉ định lựa chọn firewall hay proxy server:
  + –H [proxy server hostname or ip address]
  + –P [proxy server port]
* Ví dụ:

Jmeter-server –H my.proxy.server –P 8080

* Nếu muốn server tự động ngắt sau khi chạy 1 single test, ta cần thiết lập JMeter property server.exitaftertest=true
* Để chạy test từ client trong CLI mode, sử dụng đoạn command sau:

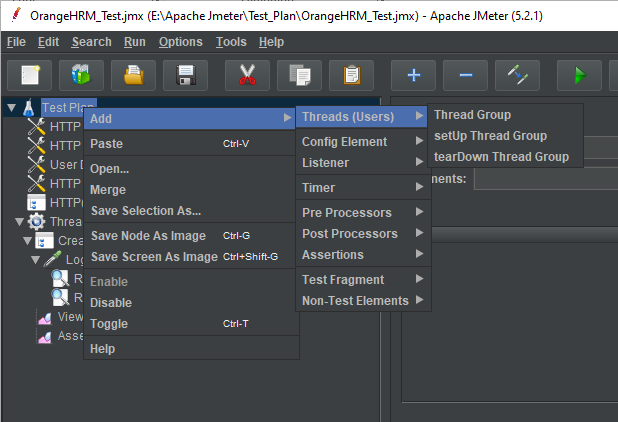
Jmeter –n –t testplan.jmx –r [-Gprop=val] [-Gglobal.properties][-X]

Trong đó:

* + –G [chỉ định JMeter properties cần được cài trong server]
  + –X [thoát server sau khi thực hiện test]

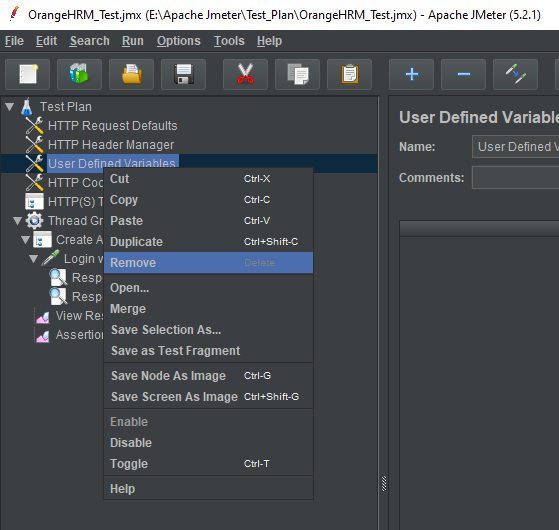
# **JMeter GUI**

* Khi chạy JMeter, ta sẽ thấy element chính của JMeter là Test Plan. Test Plan là vùng chứa các element cần thiết để tạo một Autotest. Các thành phần của Test Plan gồm:
  + Thread Group
    - Samplers
    - Logic Controllers
  + Listeners
  + Configuration Elements
  + Assertions
  + Timers
  + Pre Processors
  + Post-Processors
  + Miscellaneous Features
  + Các chức năng trong mục Non-Test Elements để phục vụ cho việc Recording Web



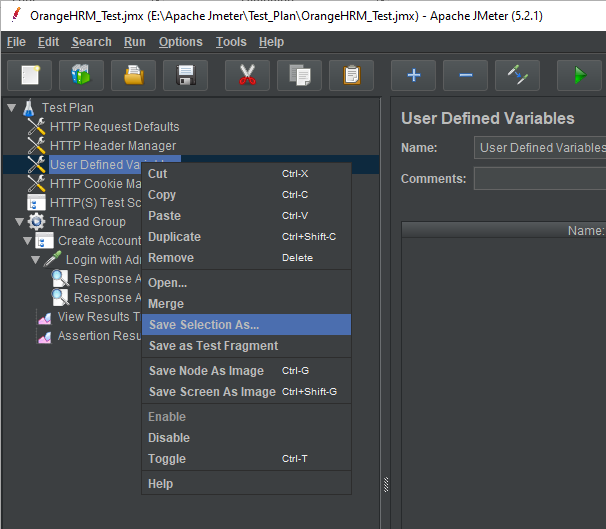
Hình 14: Elements of a Test Plan

* Element là các yếu tố cần thiết để xây dựng một Test Plan, bởi nếu không có chúng, JMeter sẽ không thể thực thi Test Plan. Để thêm element, ta chọn Test Plan 🡪 Add 🡪 lựa chọn element.
* Để Remove Element, ta chọn một element 🡪 remove, hoặc nhấn nút delete và xác nhận



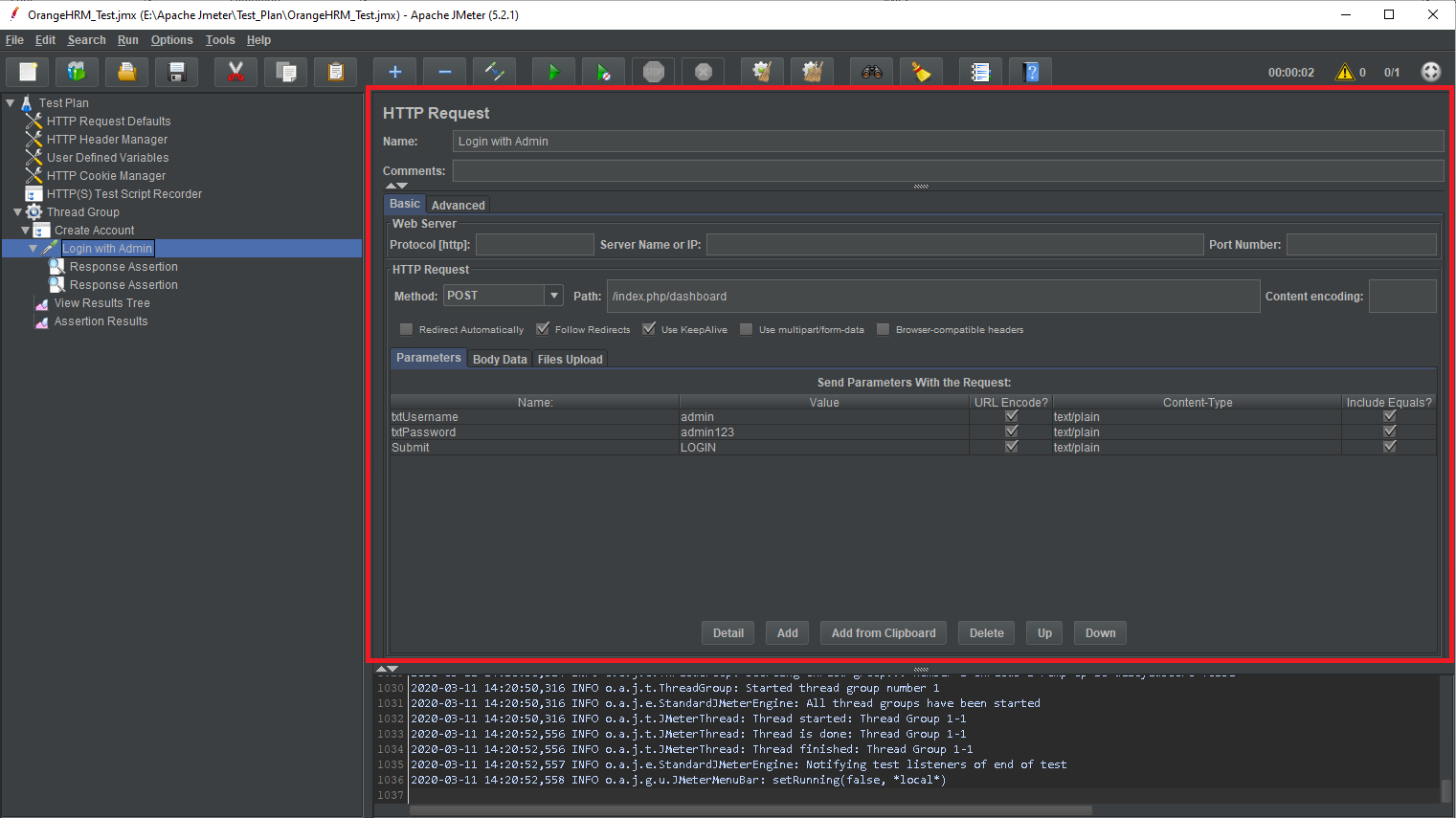
Hình 15: Remove element

* Một element sau khi được tạo cũng có thể lưu lại để dùng sau này. Để lưu lại element, ta chọn element 🡪 Save Selection As, trỏ đường dẫn tới thư mục lưu và click Save



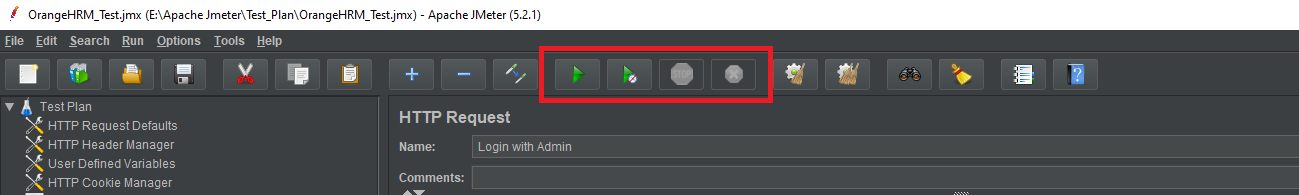
Hình 16: Save Element

* Để load một element đã được lưu từ trước vào Test Plan, ta chọn Một element đang có trong Test Plan 🡪 Merge, trỏ đến file [element].jmx cần load và click open.
* Để cấu hình một element, ta chọn element hiện có trong Test Plan, chỉnh sửa thiết lập tại vùng cửa sổ bên phải



Hình 17: Config Element

* Để chạy Test Plan, ta click nút Run trên thanh công cụ, hoặc nhấn tổ hợp phím Ctrl + R



# **Test Plan**

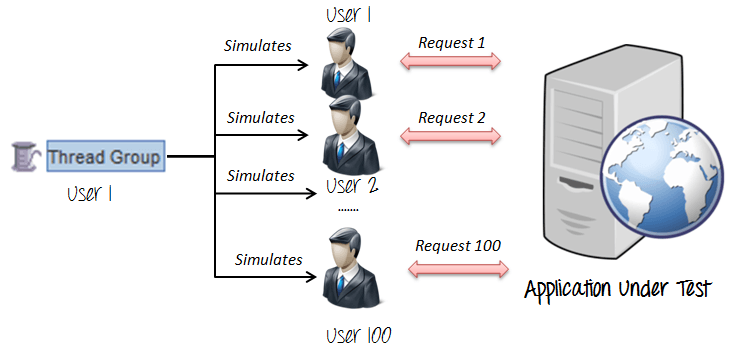
## Test Plan là gì

* Một Test Plan xác định và cung cấp tổng quan một layout những thứ cần có, cần thực hiện để test: ứng dụng web tốt như một ứng dụng client – server. Nó có thể được xem là bộ phận điều khiển các test case. Test Plan cung cấp một framework thực hiện một trình tự hành động liên tục để thực hiện kiểm tra hiệu năng. Một Test Plan bao gồm các thành phần như Thread Groups, Logic Controllers, Samples Controller, Listeners, Timers, Asseartions và Configuration Elements. Một Test Plan phải có ít nhất một Thread Group.

## Các Element của một Test Plan

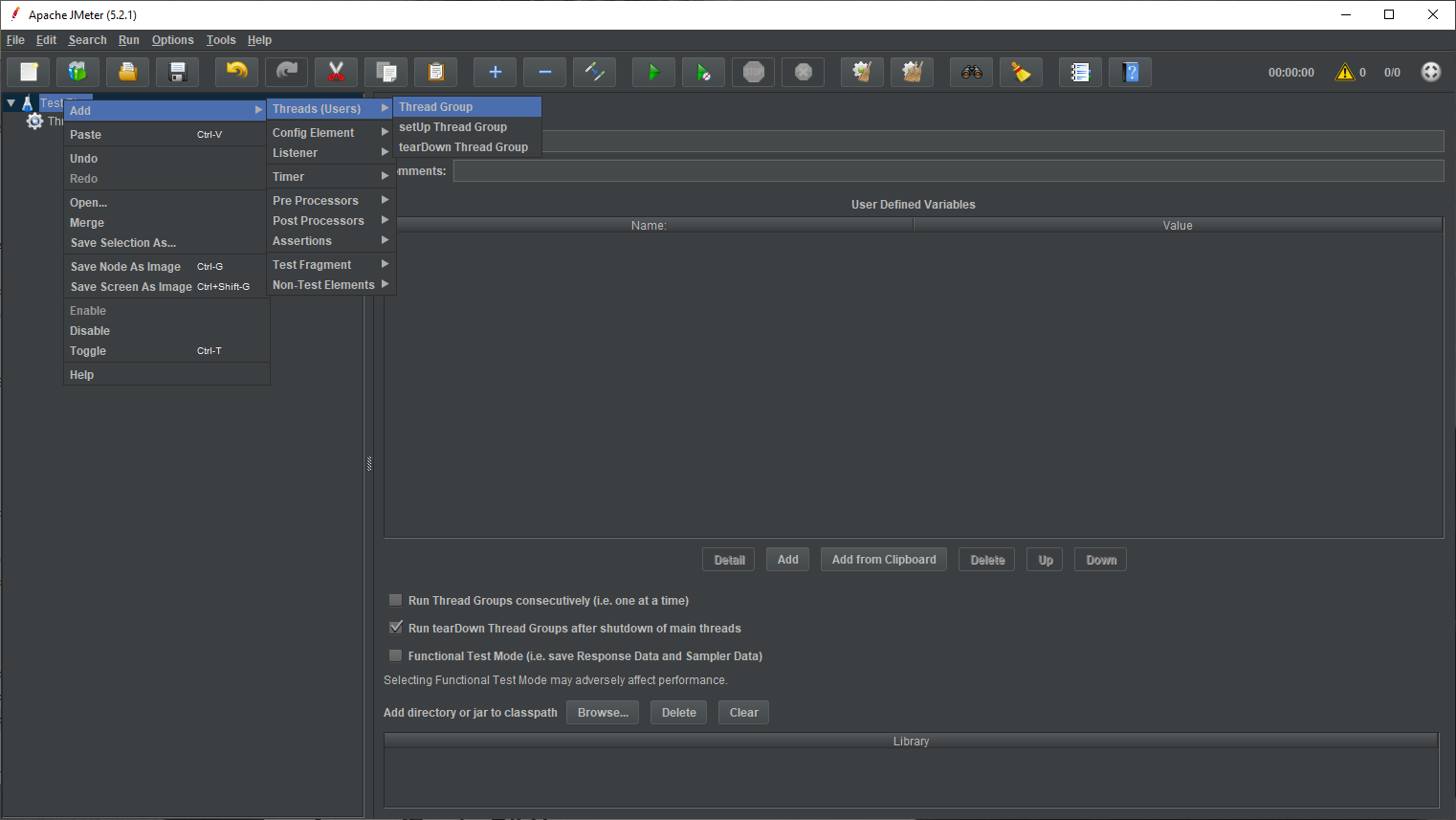
### Thread Group

* Thread Group là một bộ Thread, là điểm bắt đầu của một Test Plan. Nó chứa các thành phần JMeter khác. Mỗi thread thể hiện một user sử dụng ứng dụng khi test. Cơ bản, mỗi thread mô phỏng một user request tới server.
* Một Thread Group tượng trưng cho một nhóm user thực hiện test. Việc điều khiển một thread group cho phép ta thiết lập số lượng thread của mỗi group.
* Ví dụ: nếu thiết lập một lượng thread là 100, JMeter sẽ tạo và mô phỏng 100 user request tới server đang test.

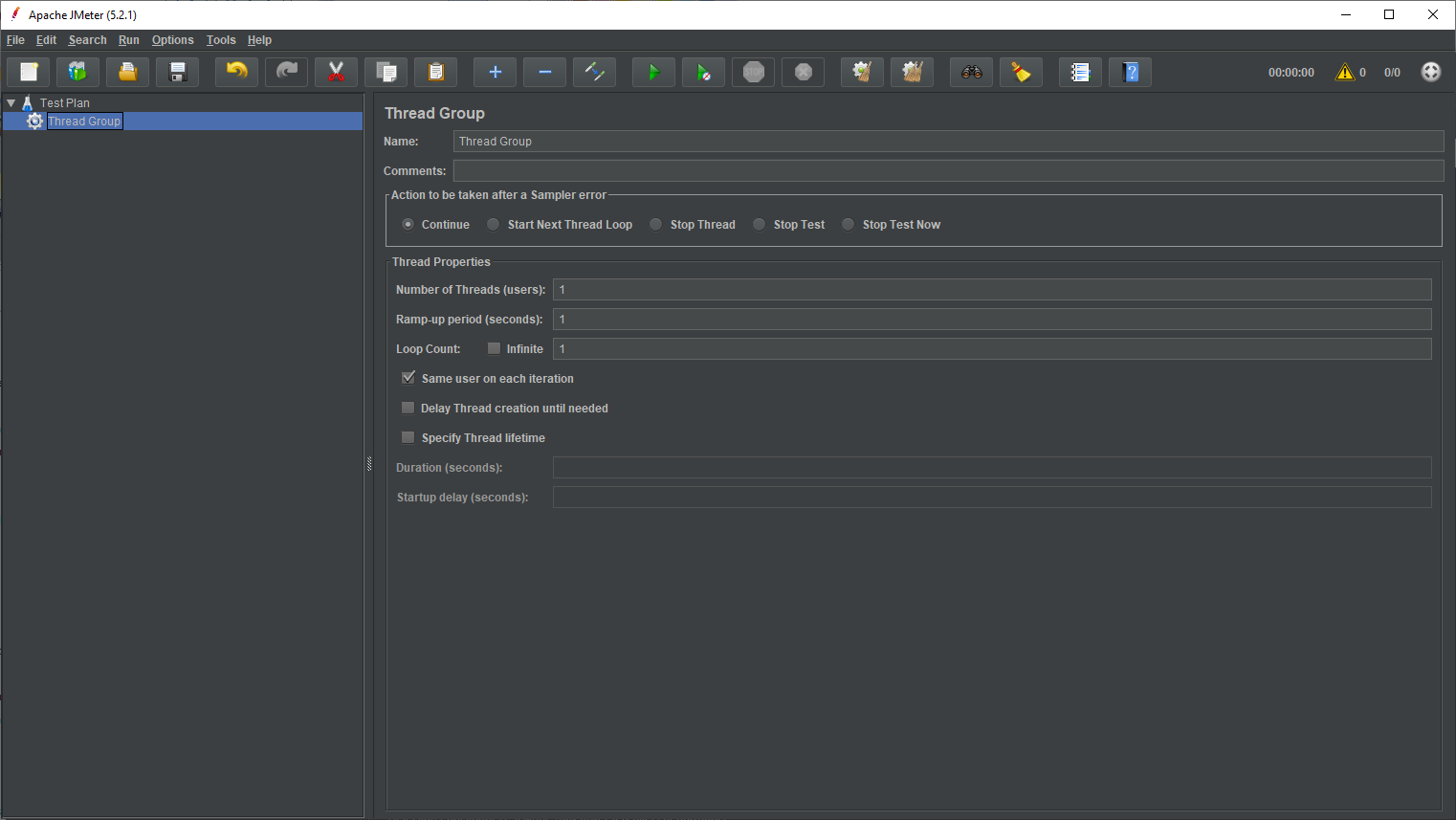


Hình 18: Simulate Thread Group

* Thread Group là thành phần cơ bản của một Test Plan. Để thêm Thread Group, ta chọn Test Plan 🡪 Add 🡪 Thread (Users) 🡪 Thread Group.



Hình 19: Add Thread Group



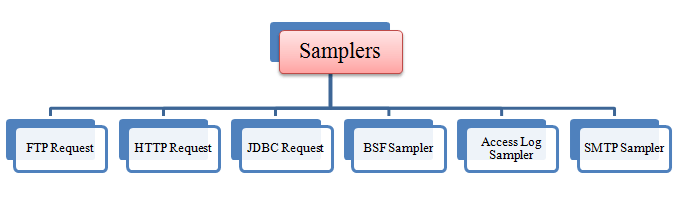
Hình 20: Thread Group interface

* Các chức năng có trong Thread Group bao gồm:
  + Name: Đặt tên cho Thread Group
  + Comments: chú thích, mô tả
  + Action to be taken after a Sampler error: Chọn hành động cho Thread Group khi 1 Sampler bị lỗi (Continue, Start Next Thread Loop, Stop Thread, Stop Test, Stop Test Now).
  + Thread Properties:
    - Number of Threads (users): số lượng Thread mong muốn, mỗi Thread thể hiện một user.
    - Ramp – up period (seconds): thời gian để JMeter chạy tất cả các Thread. Ví dụ nếu set RUP là 60s với NOT là 10 thread, thì mỗi Thread liên tiếp sẽ delay một khoảng thời gian là 6s. Trong 60s, tất cả các Thread sẽ chạy hết. Cần phải chọn thời gian RUP đủ dài để tránh khối lượng hành động lớn khi bắt đầu test, nhưng cũng đủ ngắn để Thread cuối cùng chạy trước khi Thread đầu tiên kết thúc. Có thể đặt RUP bằng với NOT và sau đó điều chỉnh cho phù hợp.
    - Loop Count: số vòng lặp hành động của mỗi 1 user. Click Infinite để chạy test liên tục cho tới khi dừng bằng cách thủ công.

### Controllers

#### Samplers

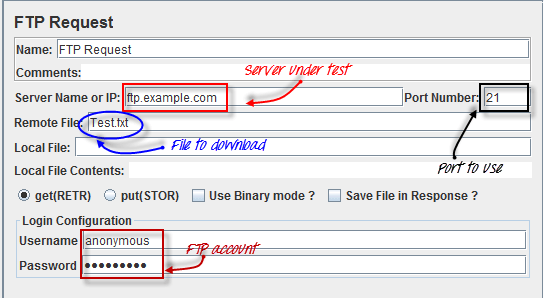
* Như ta đã biết, JMeter hỗ trợ test HTTP, FTP, JDBC và nhiều giao thức khác, và samplers dùng để Thread Group nhận biết loại request (HTTP, FTP, JDBC) và tạo request đó.



Hình 21: Sampler contains requests

##### FTP Request

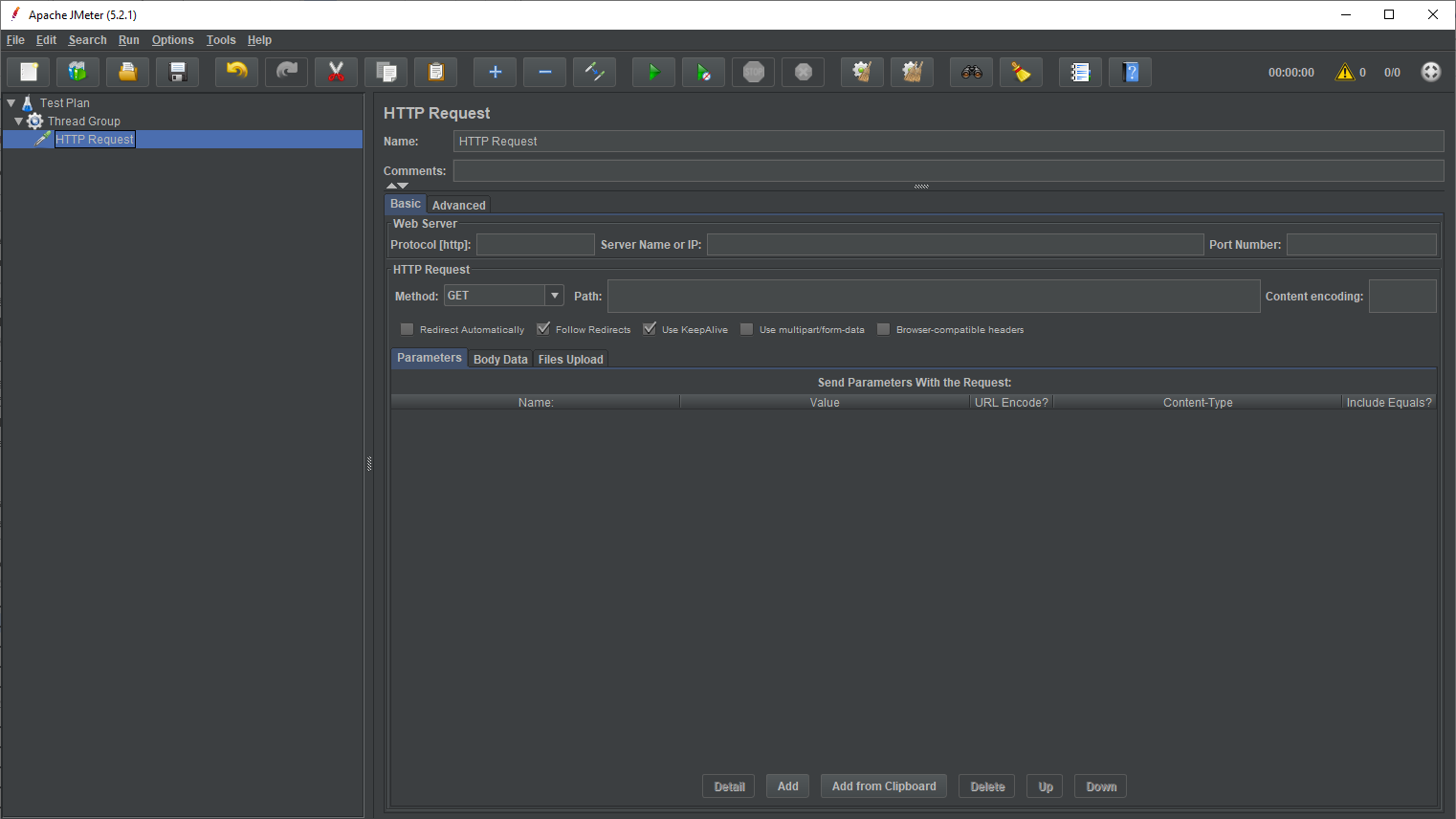
* Sử dụng FTP request để test FTP server. Controller này giúp tai gửi một FTP “download file” hoặc “upload file” tới FTP server.
* Ví dụ: nếu ta muốn tải một file “Test.txt” từ một FTP server, ta cần cấu hình FTP request như sau:



Hình 22: Example of FTP Request

##### HTTP Request

* Sampler này được dùng để gửi một HTTP/HTTPs Request tới một web server. Nó cũng dùng để JMeter phân giải các tệp HTML về hình ảnh hoặc nguồn nhúng khác và gửi HTTP request để truy xuất chúng.



Hình 23: HTTP Request Panel

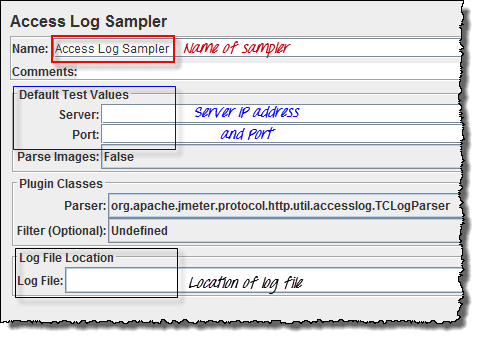
* Các tham số chính của HHTP Request:
  + Protocol: giao thức của server cần gửi request đến – HTTP hoặc HTTPs.
  + Url: địa chỉ đinh danh của tài nguyên, ở đây là đường link tới server cần test.
  + Port: cổng request tới url.
  + Method: là phương thức mà HTTP Request này sử dụng, thường là GET, POST, ngoài ra còn có HEAD, PUT, DELETE, OPTION, CONNECT.
  + Path: đường dẫn tới cây thư mục con của server.
  + Parameter: tạo các data test để truyền giá trị vào các parameter, các parameter này tương ứng với các parameter có trên màn hình trang web
  + Body data: tạo các tham số và truyền giá trị cho các tham số đó, các tham số tương ứng với parameter được lấy từ màn hình request. Cách thức giống với Parameter, nhưng được viết dưới dạng script.
  + File Upload: mục này được sử dụng khi cần gửi HTTP request để upload file.
* Phương thức GET và POST:
  + Với GET, câu truy vấn sẽ được đính kèm vào đường dẫn của HTTP Request. Ví dụ: **/?username=”abc”&password=”def”**
    - Đặc điểm:
      * GET có thể được cached, bookmark hay lưu trữ trong lịch sử trình duyệt.
      * GET request bị giới hạn về chiều dài, do chiều dài của URL là có hạn.
      * GET request không nên dùng với dữ liệu quan trong, chỉ dùng để nhận dữ liệu.
  + Với POST, câu truy vấn sẽ được gửi trong message body của HTTP request:
    - Đặc điểm:
      * POST không thể cached, bookmark hay lưu tronng lịch sử trình duyệt
      * POST không bị giới hạn về chiều dài
      * POST có tính bảo mật, dùng để gửi những dữ liệu cần bảo mật
* Controller này sẽ được sử dụng phổ biến ở các mục sau trong tài liệu.

##### JDBC Request

* JDBC Request dùng để thực hiện Performance testing database. Do khi thực hiện query để lấy dữ liệu, có thể sẽ phải lấy dữ liệu từ nhiều bảng khác nhau, với nhiều điều kiện lọc giới hạn phạm vi dữ liệu sẽ được trích xuất ra nên hay xảy ra vấn đề là tốc độ trả về kết quả hiển thị trên màn hình mất nhiều thời gian, hoặc không hiển thị được kết quả trên màn hình do timeout. Và cũng có trường hợp nhiều user thực hiện việc search dữ liệu trên cùng một trang ở tại cùng một thời điểm, có thể ảnh hưởng đến tốc độ cho kết quả hoặc là bị timeout.
* Vì thế cần phải thực hiện kiểm tra performance, so sánh tốc độ xử lí thực tế với tốc độ kì vọng chênh lệch như nào để có hướng cải thiện tốc độ.
* Để thực hiện gửi JDBC request tới database, ta xem chi tiết ở mục 14.3 trong tài liệu.

##### Access Log Sampler

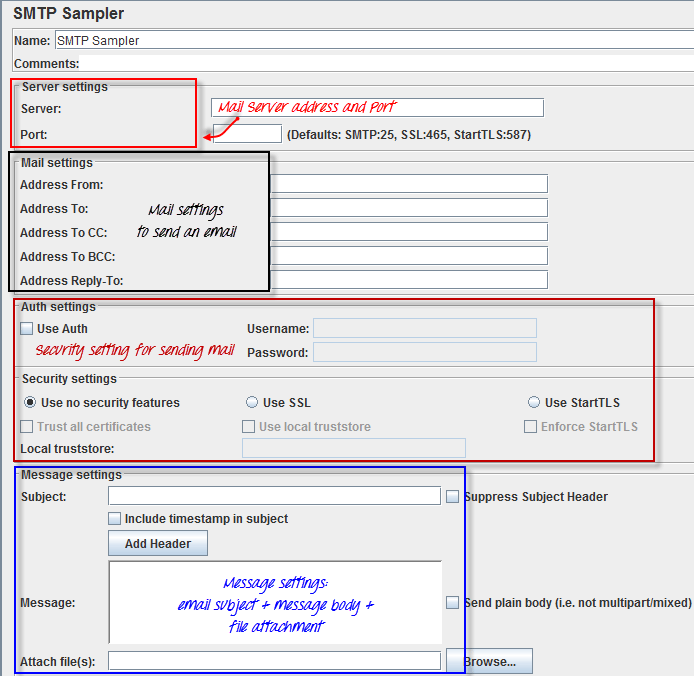
* Sampler này cho phép đọc access logs và tạo HTTP request. Log có thể là ảnh, Html, CSS,…



Hình 24: Access Log Sampler

##### SMTP Sampler

* Sampler này để test mail server, nó sẽ gửi mail thông báo sử dụng giao thức SMTP.



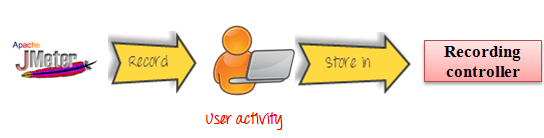
Hình 25:SMTP Sampler

#### Logic Controller

* Logic Controllers xác định thứ tự xử lí yêu cầu trong một Thread. Nó cho phép ta điều khiển khi nào cần gửi một user request tới web server. Ví dụ, ta có thể sử dụng Random Controllers để gửi HTTP request tới server một cách ngẫu nhiên. Hay nói cách khác, Logic Controllers để xác định thứ tự yêu cầu của người dùng, bao gồm: Recording, Simple, Loop, Random, Module,…

##### Recording Controller

* JMeter có thể record các bước Test, recording controller là nơi lưu trữ các bước record. Controller này được sử dụng khi ta record một trang web thông qua cài đặt proxy.

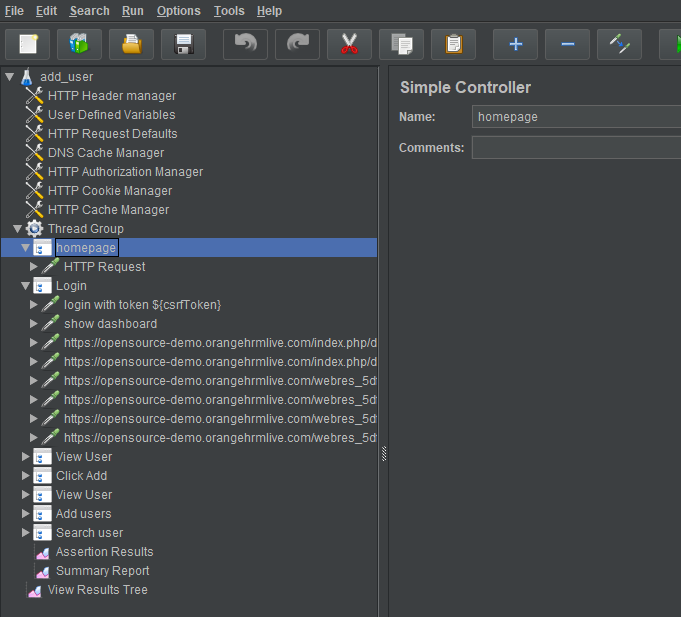


Hình 26: Recording Controller

* Thông thường ta sẽ tạo Recording Controller khi thực hiện record bằng proxy, hoặc cách nhanh hơn là sử dụng Template Record của JMeter, khi đó Test Plan sẽ được tạo mẫu tự động và có sẵn Recording Controller.

##### Simple Controller

* Simple Controller dùng để chứa các User Request. Sử dụng Simple Controller khi tạo Test Plan sẽ giúp ta dễ dàng kiểm tra các khâu thực thi trong Test Plan hơn.



Hình 27: Simple Controllers

##### Loop Controller

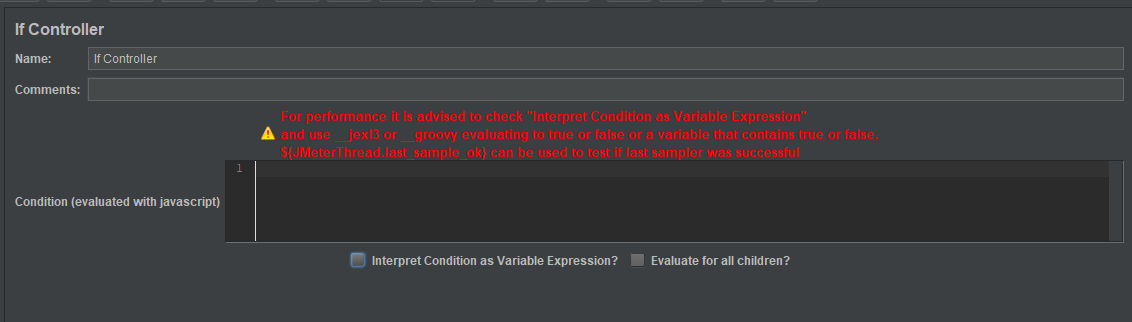
* Loop Controller làm user request chạy trong một khoảng thời gian cố định hoặc chạy vô hạn:



Hình 28: Road map of Loop Controller

##### If Controller

* If Controller tương tự như điều kiện if… else trong ngôn ngữ lập trình. Các request nằm trong controller này chỉ được chạy khi điều kiện ta thiết lập trong If Controller thỏa mãn.

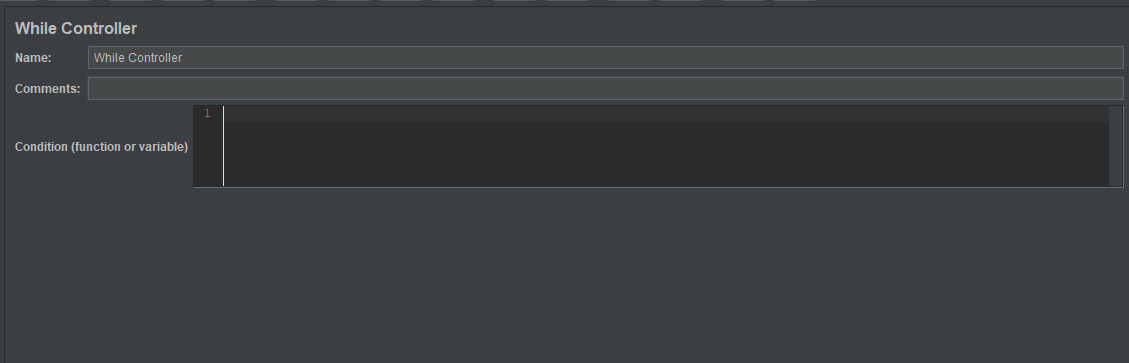


Hình 29: If Controller Panel

* Condition (evaluated with javascript): điều kiện thiết lập cho controller để điều khiển các thành phần trong nó. Mặc định được viết bằng javascript và trả về true hoặc false.
* Interpret Condition as Variable Expression?: Sử dụng khi muốn tạo script điều kiện bằng ngôn ngữ khác không phải Javascript.
* Enable for all children?: Kiểm tra điều kiện cho từng mục con trong If Controller.

##### While Controller

* Các thành phần bên trong Controller này sẽ được lặp lại liên tục cho đến khi điều kiện của controller trả về giá trị là false.

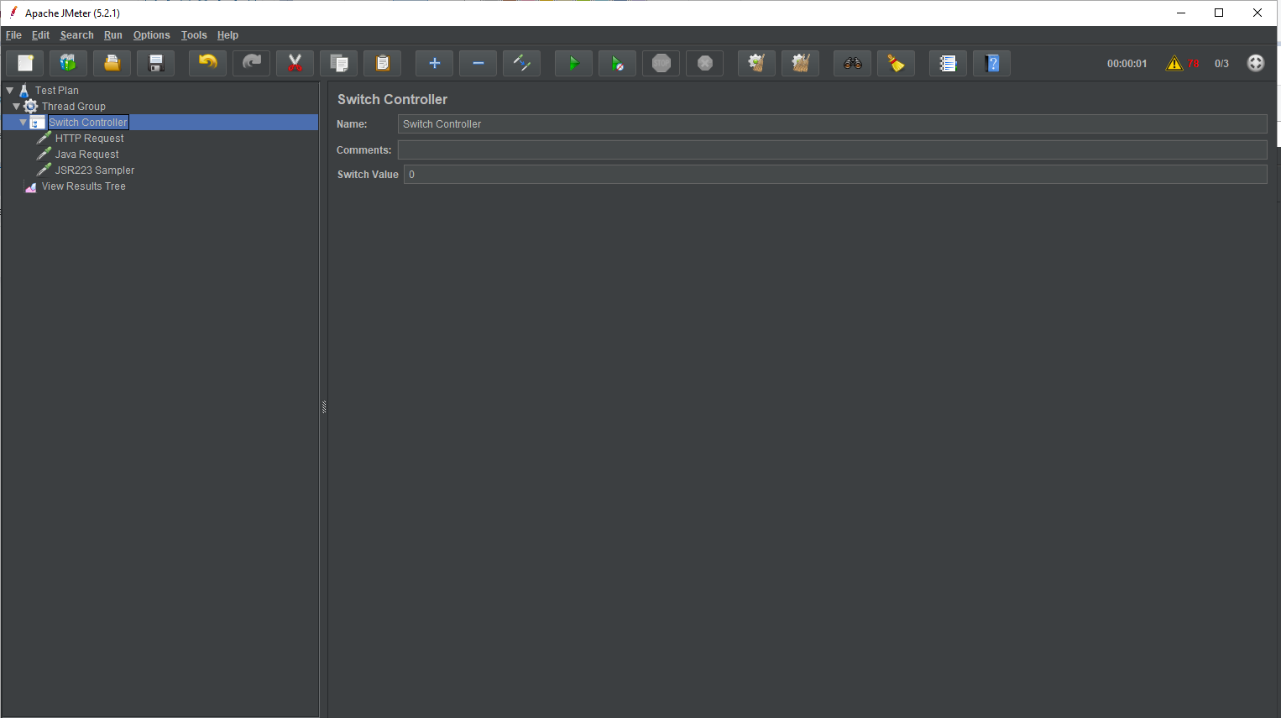


Hình 30: While Controller Panel

* Condition (function or variable): Có thể thiết lập hàm hoặc variable để tạo điều kiện cho vòng lặp. Câu lệnh trong Condition được viết bằng Javascript, ngoài ra cho phép các hàm jexl3 và groovy.

##### Switch Controller

* Switch Controller sẽ thực hiện một request bên trong nó dựa vào tham số “Switch Value”. Tham Giá trị của tham số này tương ứng với vị trí của thành phần trong controller đó (Mặc định thành phần đầu tiên là 0). Nếu Switch Value lớn hơn chỉ số vị trí của thành phần cuối cùng trong Switch thì Switch Value sẽ trả về giá trị 0.



Hình 31: Switch Controller

* Ví dụ: Theo hình 31, nếu set giá trị Switch value là 0, thì Test Plan sẽ chỉ chạy HTTP request, set là 1 thì chạy Java Request, set là 2 chạy JSR223 Sampler. Nếu set giá trị lớn hơn 2 sẽ mặc định chạy HTTP Request.

##### Random Controller

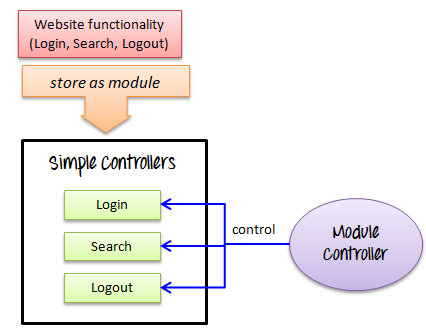
* Random Controller làm tất cả các user request chạy một cách ngẫu nhiên.
* Ví dụ, ta có 3 user request đến [www.google.com](http://www.google.com) theo thứ tự:

1. HTTP request
2. FTP request
3. JDBC request

* Mỗi request cần chạy 5 lần, tổng công 15 user request cần được gửi tới Google bằng JMeter.
* Nếu yêu cầu gửi request theo thứ tự, các request được gửi theo thứ tự: HTTP request 🡪 FTP request 🡪 JDBC request ở mỗi vòng lặp.
* Nếu yêu cầu gửi request một cách ngẫu nhiên, các request sẽ gửi theo 1 thứ tự không xác định: FTP request 🡪 HTTP request 🡪 JDBC request hoặc JDBC request 🡪 FTP request 🡪 HTTP request… ở mỗi vòng lặp

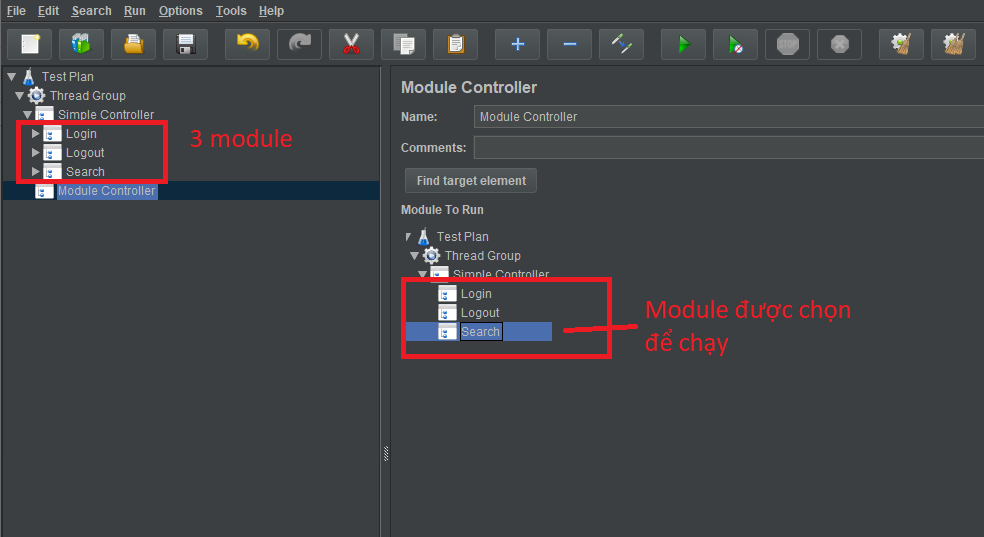
##### Module Controller

* Các Web Application thường bao gồm nhiều hàm chức năng nhỏ (Login, tạo tài khoản, Log out,…). Các hàm chức năng này có thể lưu trong Simple Controller như là 1 Module và được Module Controller chọn để chạy.



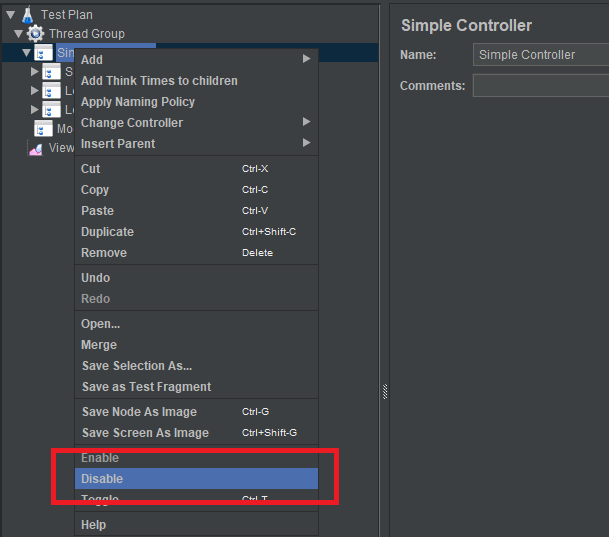
Hình 32: Module Controller example

* Giả sử ta muốn mô phỏng:
  + 50 user đăng xuất
  + 100 user đăng nhập
  + 30 user thực hiện tìm kiếm trên [www.google.com](http://www.google.com)
* Ta có thể tạo 3 module, mỗi module mô phỏng một yêu cầu của người dùng: Login, Logout, Search, sau đó chọn module cần được chạy.



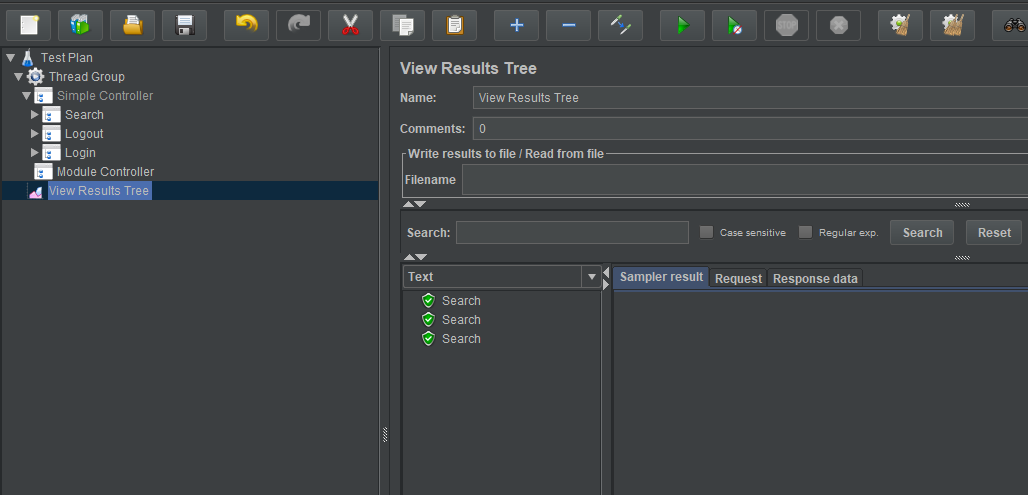
Hình 33: Config Module Controller

* Disable Simple Controller (nếu không Disable, khi run sẽ chạy toàn bộ test plan và bỏ qua Module Controller).

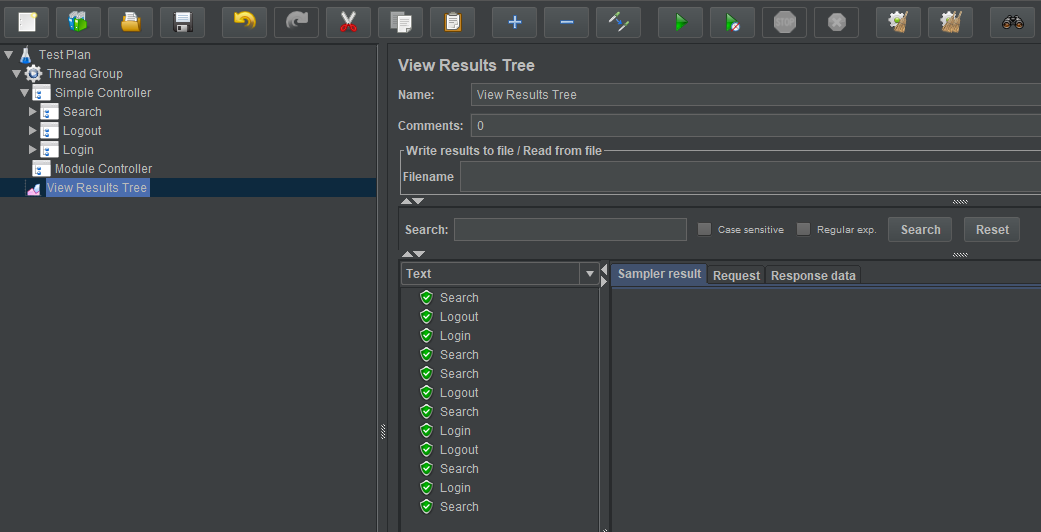


Hình 34: Disable Simple Controller

* Thêm View Results Tree để xem kết quả Test Plan



Hình 35: Kết quả khi Disable Simple Controller

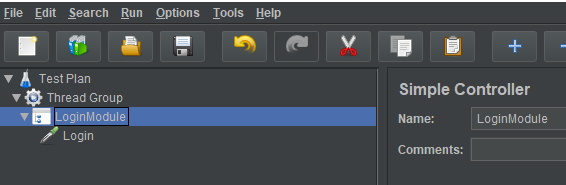


Hình 36: Kết quả khi không Disable Simple Controller

##### Include Controller

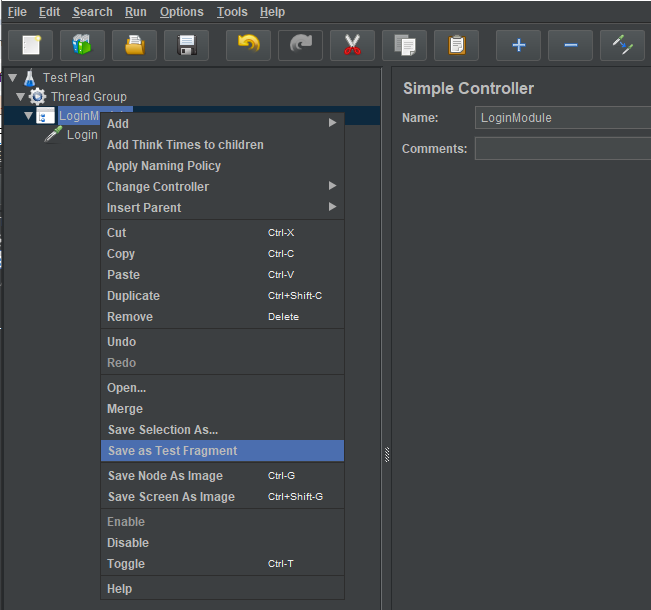
* Include Controller chia nhỏ test plan ra thành nhiều mảng. Nghĩa là trong một test Plan, ta có thểm thêm nhiều Include Controller, và mỗi Controller đó chứa các thành phần thực hiện một chức năng nhất định. Ví dụ: ta có 5 kịch bản load test và mỗi test plan cần login để thực hiện được các bước test tiếp theo. Thay vì phải record/add các bước login ở cả 5 kịch bản test thì ta có thể lưu lại chức năng login ra một fragment và thêm vào mỗi kịch bản thông qua Include Controller. Sau đây là ví dụ mô phỏng:

1. Tạo một Simple Controller thực hiện hành động login



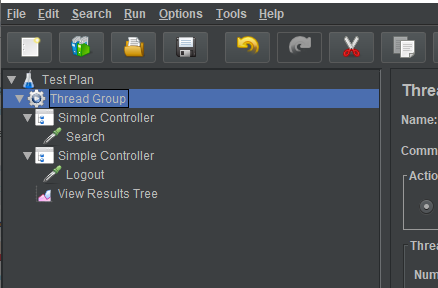
Hình 37: Tạo Module Login

1. Lưu lại Controller dưới dang Test Fragment



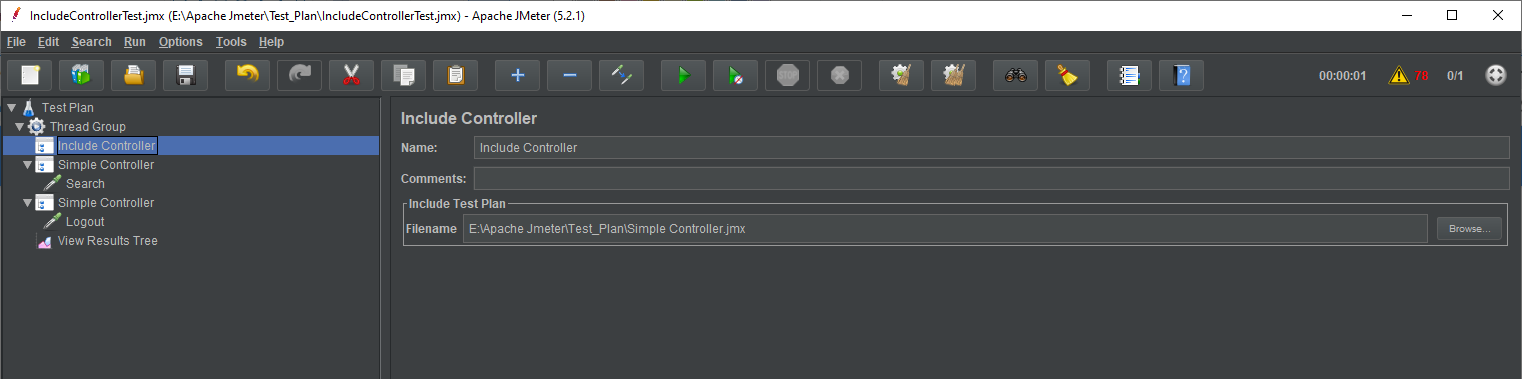
Hình 38: Save Controller as Tes Fragment

1. Tạo một Test Plan



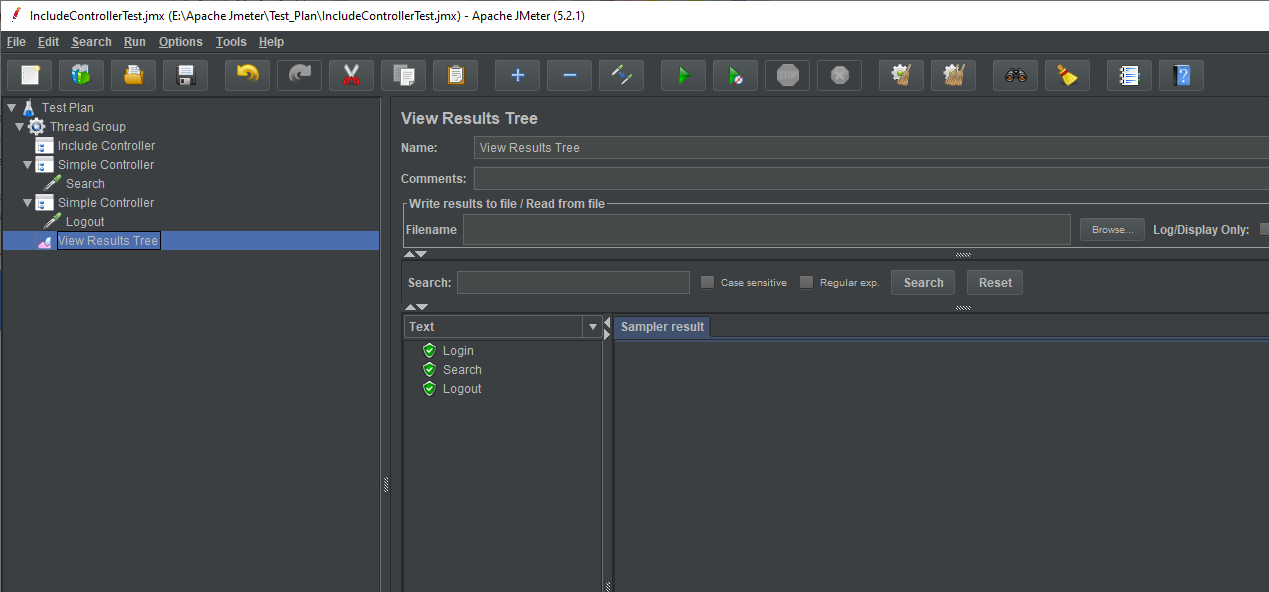
Hình 39: Tạo Test Plan

1. Thêm Include Controller, trong mục File Name, trỏ đường dẫn tới Controller vừa lưu ở bước 2.



Hình 40: Thiết lập Include Controller

1. Run test và xem kết quả trong View Results Tree



Hình 41:Kết quả Test

##### Một số Controller khác

###### Interleave Controller

* Chọn và làm một trong số các user request chạy trong mỗi vòng lặp của Thread.

###### Runtime Controller

* Điều khiển các Module bên trong Controller này trong một khoảng thời gian cố định. Hết thời gian, Runtime Controller sẽ kết thúc.

###### Transaction Controller

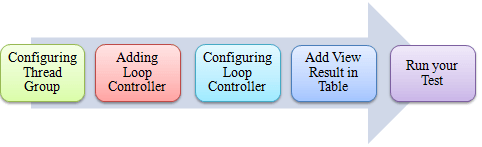
* Đo thời gian tổng thể để hoàn thành một lần thực hiện test

###### Include Controller

* Được thiết kế để sử dụng một Test Plan mở rộng. Controller này cho phép sử dụng nhiều Test Plan trong JMeter.

##### Loop Controller Example

* Mục này sẽ chỉ ra các bước để thực hiện Test Plan có Loop Controller.
* Loop Controller khiến các Sampler chạy trong một số lần nhất định, phụ thuộc vào giá trị lặp ta chọn cho Thread Group. Ví dụ, nếu ta:
  + Thêm một HTTP Request vào một Loop Controller với giá trị là 50 lần
  + Cấu hình Thread Group Loop Count là 2
  + Khi đó, JMeter sẽ gửi tổng cộng là 50 \* 2 = 100 HTTP Request tới server



Hình 42: Road map of The Example

* Bước 1: Configuring Thread Group

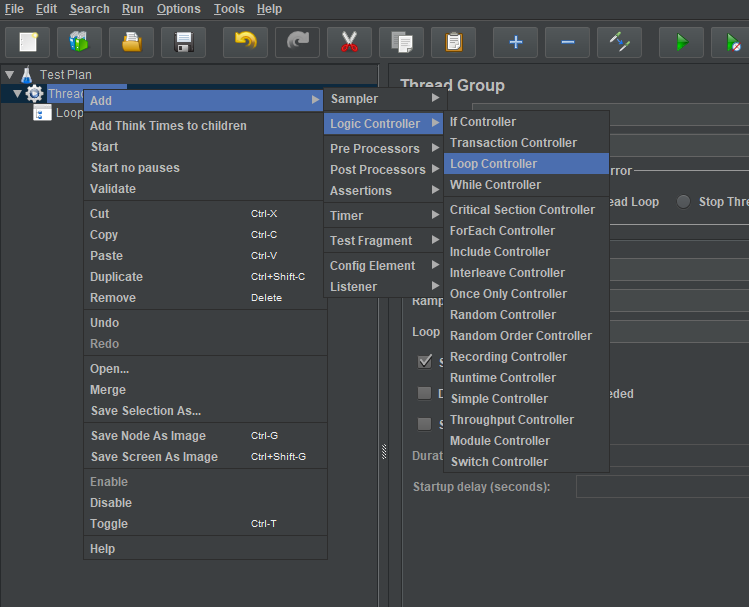
1. Add Thread Group
   * + Chuột phải vào Test Plan, chọn Add 🡪 Thread (Users) 🡪 Thread Group
     + Cấu hình cho Thread Group như sau:



Hình 43: Configure Thread Group

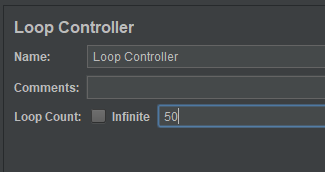
* + - JMeter sẽ gửi 1 user request đến server google 2 lần.

1. Add JMeter elements:
   * + Thêm HTTP Request Defaults, gửi request tới [www.google.com](http://www.google.com) như mục 5.
2. Add Loop Controller:
   * + Chuột phải vào Thread Group 🡪 Logic Controller 🡪 Loop Controller



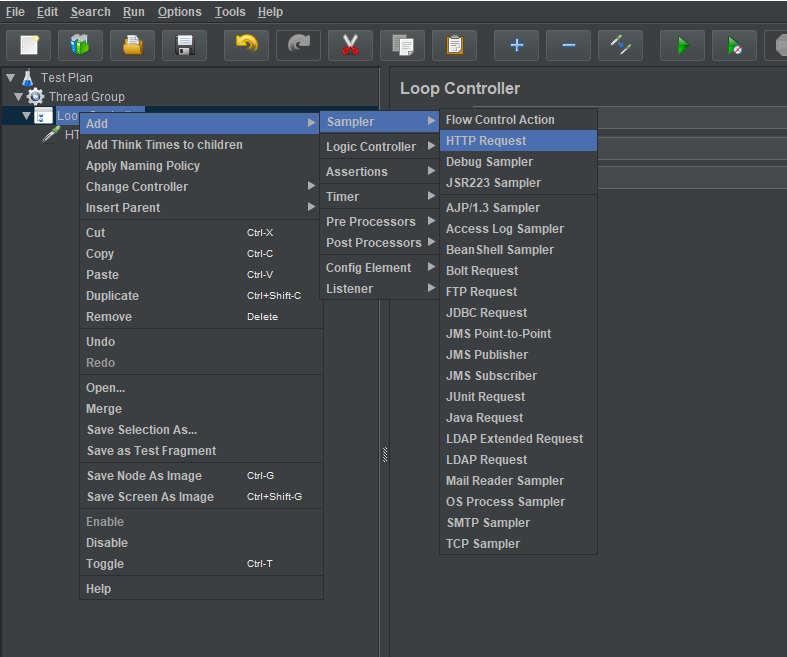
Hình 44: Adding Loop Controller

* Bước 2: Configuring Loop Controller
  + Thêm giá trị cho Loop Count là 50, nó sẽ tạo 1 user request tới google.com và thực hiện 50 lần. Với số vòng lặp trong Thread Group là 2, JMeter sẽ gửi tổng cộng 100 HTTP request tới server.



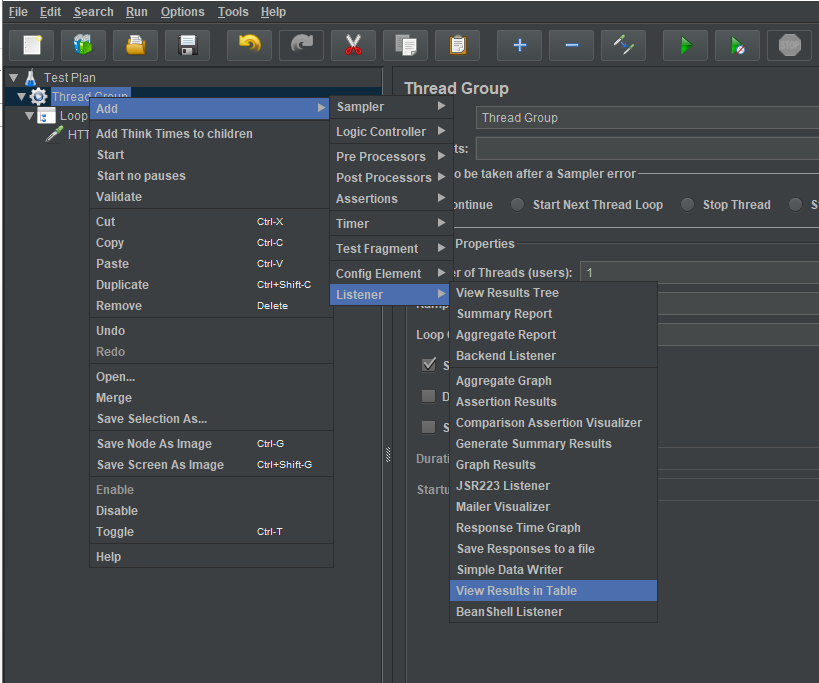
Hình 45: Configuring Loop Controller

* + Chuột phải vào Loop Controller 🡪 Add 🡪 Sampler 🡪 Http Request



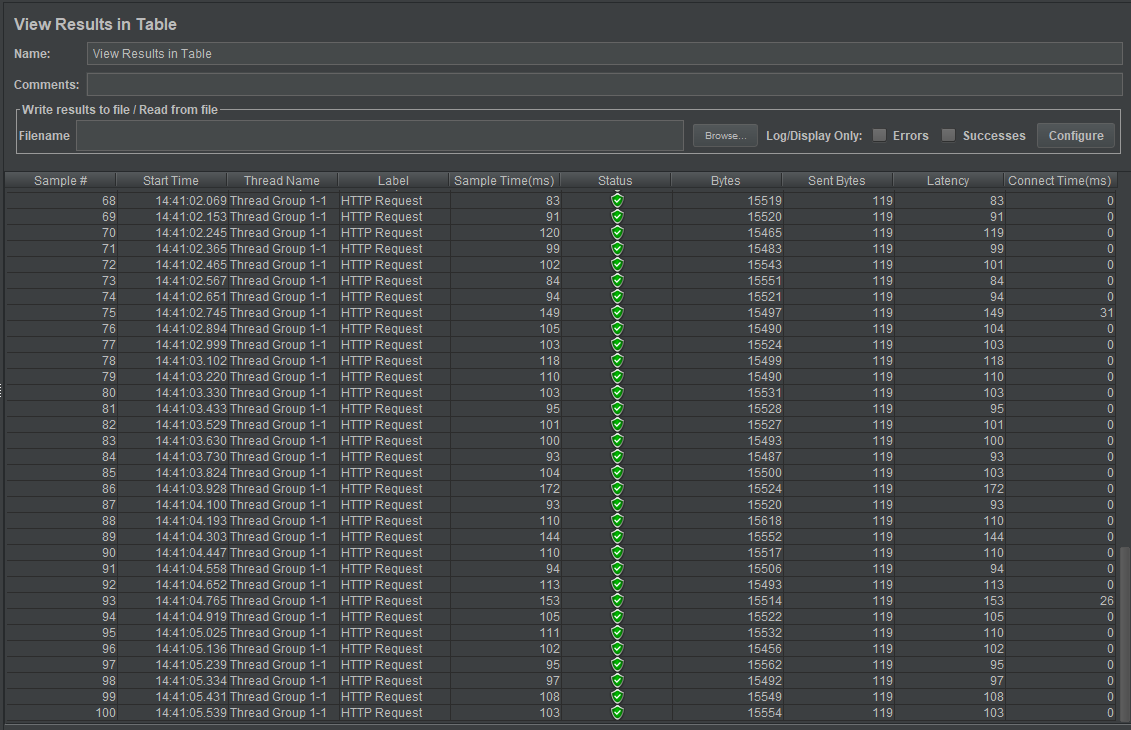
Hình 46: Adding HTTP Request

* Bước 3: Add View Results in Table
  + Chuột phải Thread Group 🡪 Listener 🡪 View Results in Table



Hình 47: Adding View Results in Table

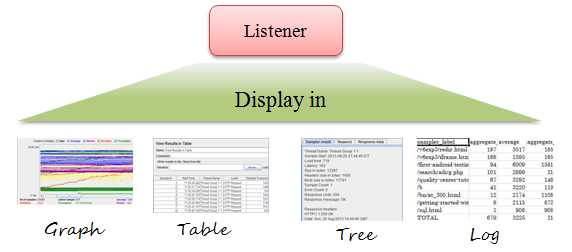
* Bước 4: Run Test Plan
  + Kết quả test sẽ được hiển thị trong View Results in Table. Test sẽ dừng lại khi đủ 100 lần thực hiện gửi request.



Hình 48: Result of the test

### Listener

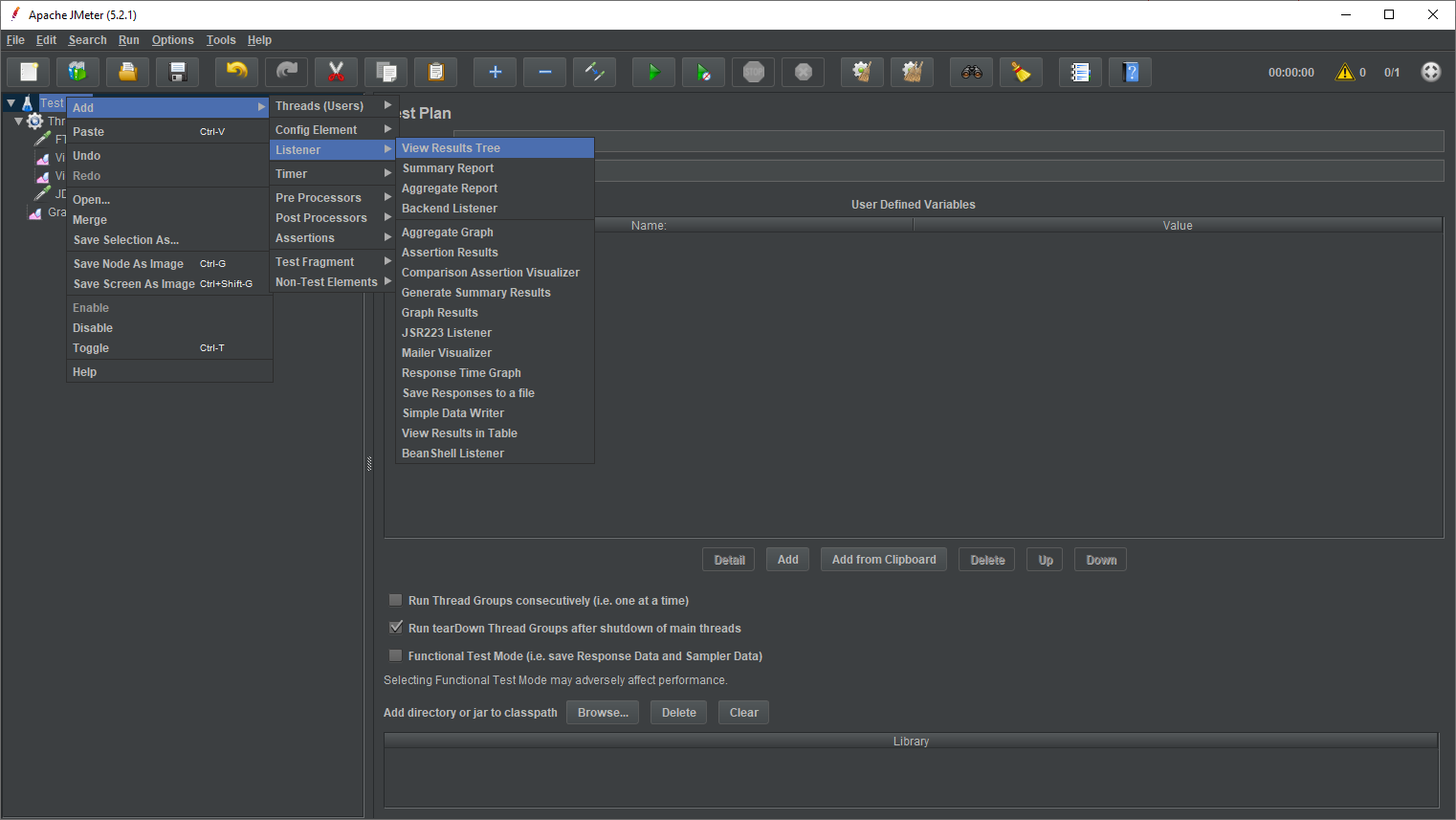
* Dùng để hiển thị kết quả của test. Có thể hiển thị theo kiểu biểu đồ, cây, log, bảng



Hình 49: Types of Listener

#### View Results Tree

* View Results tree hiển thị các sample mà JMeter khởi tạo, và kết quả trả về liên quan tới các sample đó. Listener này hiển thị kết quả theo thứ tự Test Plan và cung cấp các tham số và dữ liệu cho từng sample.
* Ví dụ đối với HTTP Request, View Results Tree sẽ hiển thị kết quả của request parameters, response parameters và response data trong tab Sampler Result, Request và Response data. Tab Sampler Result bao gồm response code, headers, cookies và thông tin về thời gian, độ trễ, kích thước phản hồi theo byte riêng cho từng thành phần: header, body và số lỗi.
* Để thêm View Results Tree, chọn Test Plan 🡪 Add 🡪 Listener 🡪 View Results Tree



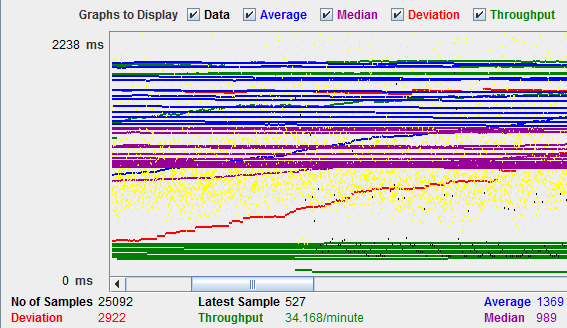
Hình 50: Add View Results Tree

#### Graph Result

* Biểu diễn các kết quả dữ liệu, kết quả trung bình, độ sai lệch, thông lượng ở dạng biểu đồ.
* Để thêm một Graph Result, ta chọn Test Plan 🡪 Add 🡪 Listener 🡪 Graph Results.



Hình 51: Add Graph Results

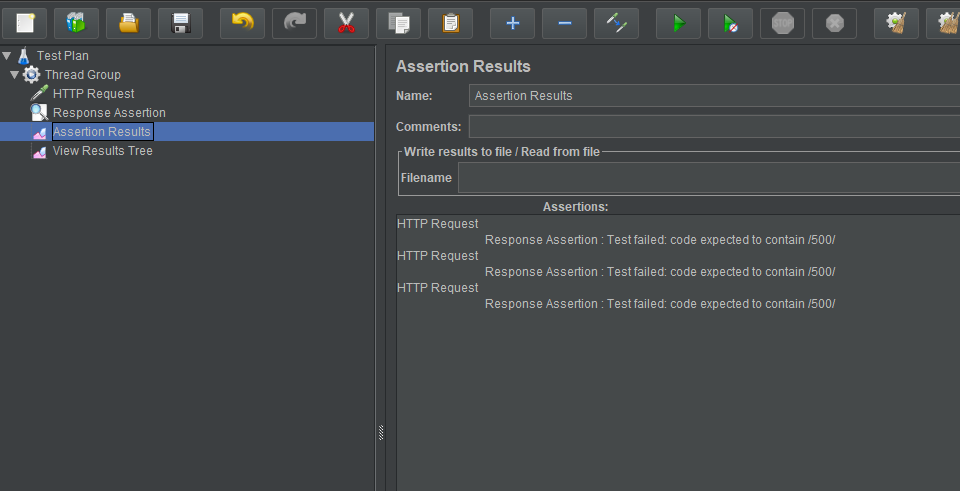


Hình 52: Graph Result

* Ý nghĩa các đại lượng trên biểu đồ:
  + No of Samples: Số lượng Sample đang được truy cập
  + Lastest Sample: Thời gian phản hồi gần nhất (ms)
  + Average Elapsed Time: Thời gian phản hồi trung bình (ms)
  + Deviation: độ sai lệch so với chuẩn (ms)
  + Throughput (KB/s): số lượng request/đơn vị thời gian
  + Median:

#### Assertion Results

* Assertion Results hiển thị kết quả của tất cả các xác nhận trong phạm vi của nó. Listener này hiển thị samples khi có lỗi xảy ra trong quá trình xác nhận. Ví dụ như xác nhận login hệ thống.

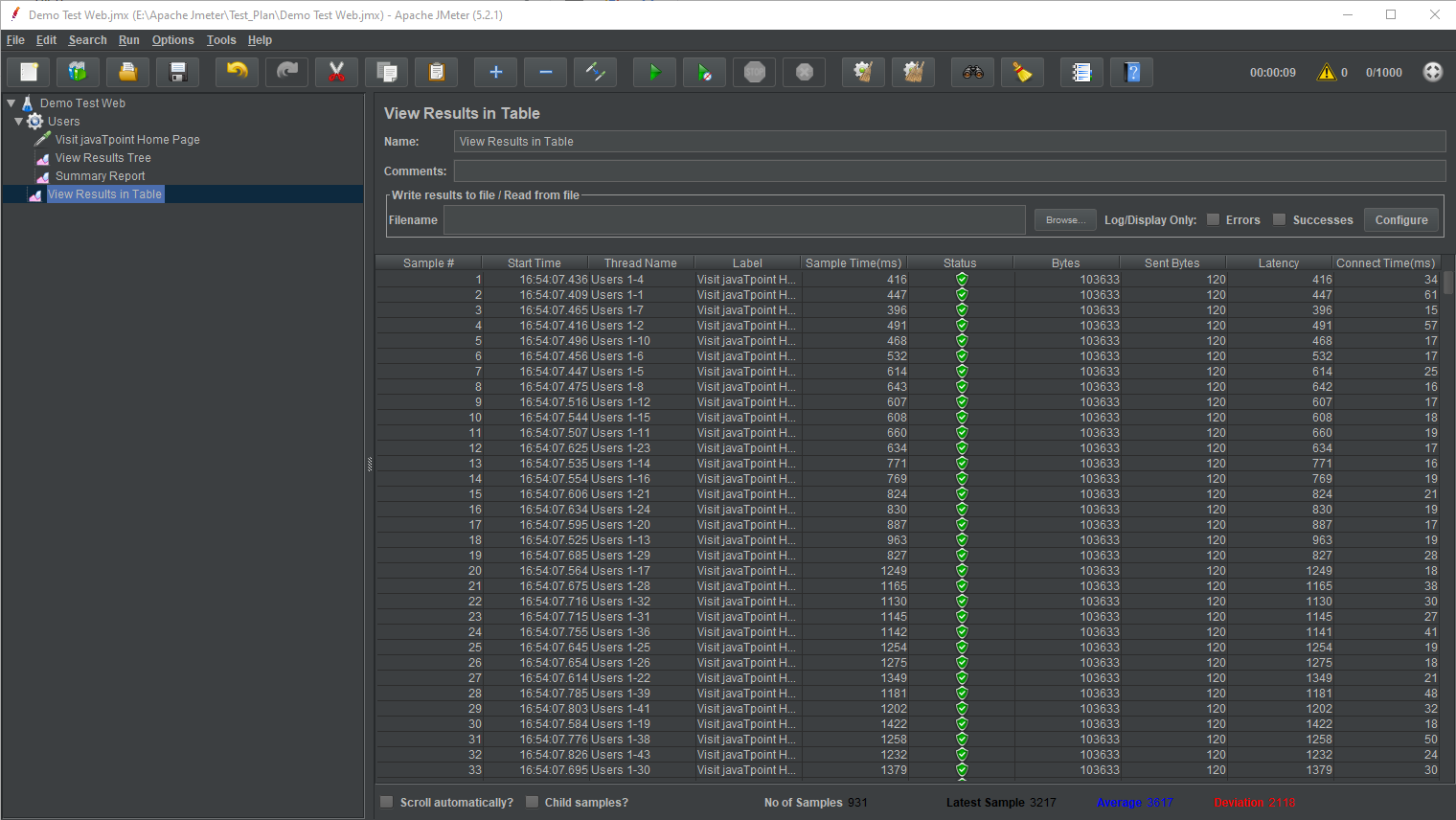


Hình 53: Example of Assertion Results

* Sử dụng Assertion Results khi Test Plan có add Assertions.

#### View Result in Table

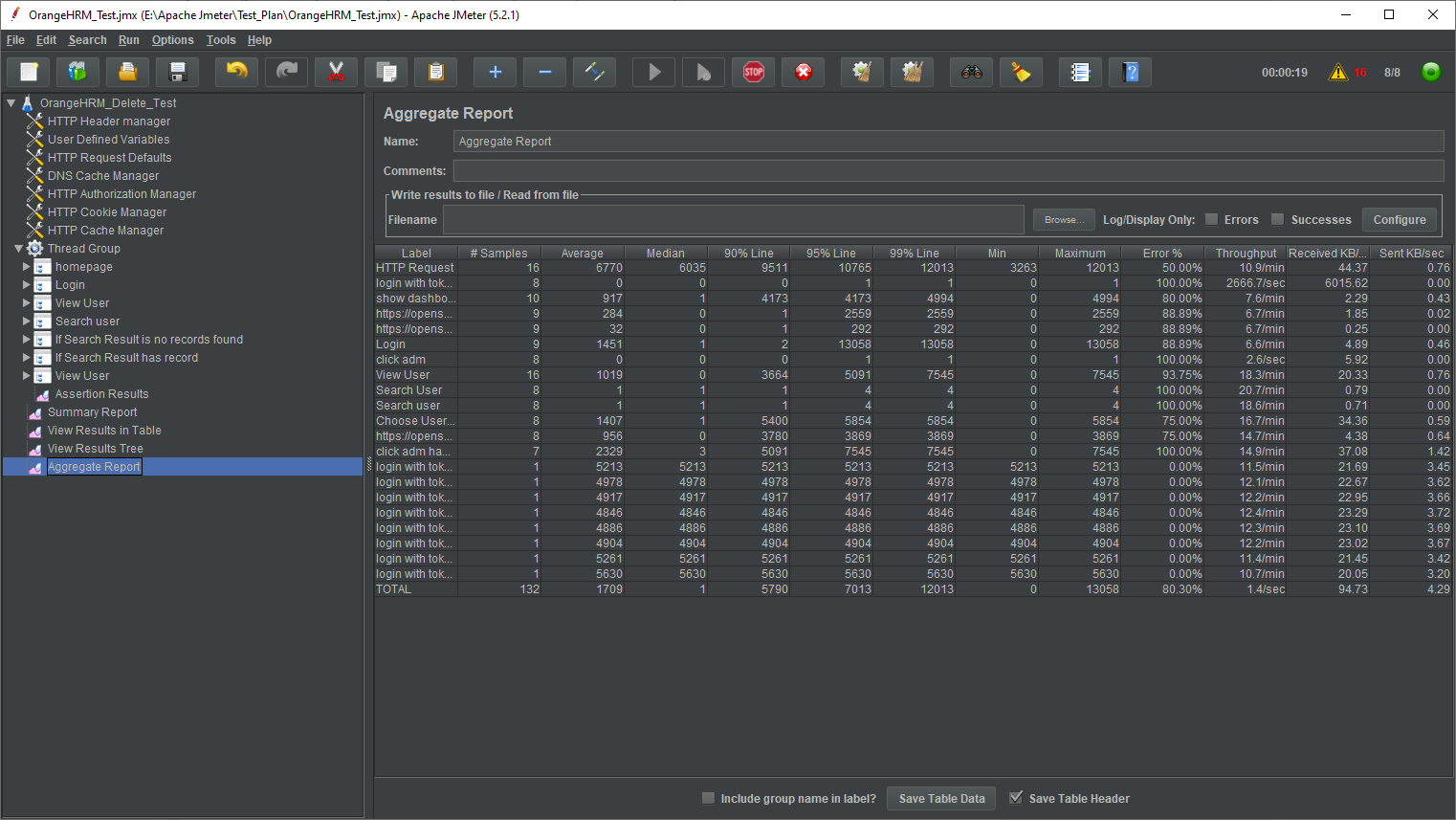
* View Result in Table hiển thị thông tin về mỗi sample dưới dạng bảng, bao gồm các thông tin về thời gian, dữ liệu tải, Thread number, kết quả sample (pass hay fail). Results in Table là Listener hữu ích khi cần thống kê thời gian xử lí các hành động của quá trình test một cách dễ dàng.



Hình 54:View Results in Table

* + Start Time: Thời điểm bắt đầu truy cập
  + Thread Name: Tên của Thread đang thực hiện
  + Label: Các Request đang được thực hiện
  + Sample Time: Thời gian thực hiện Request tương ứng
  + Status: Pass/Fail
  + Bytes: Lượng dữ liệu tải về
  + Sent Byte: Lượng dữ lại tải lên
  + Latency: độ trễ
  + Connect Time: Thời gian kết nối

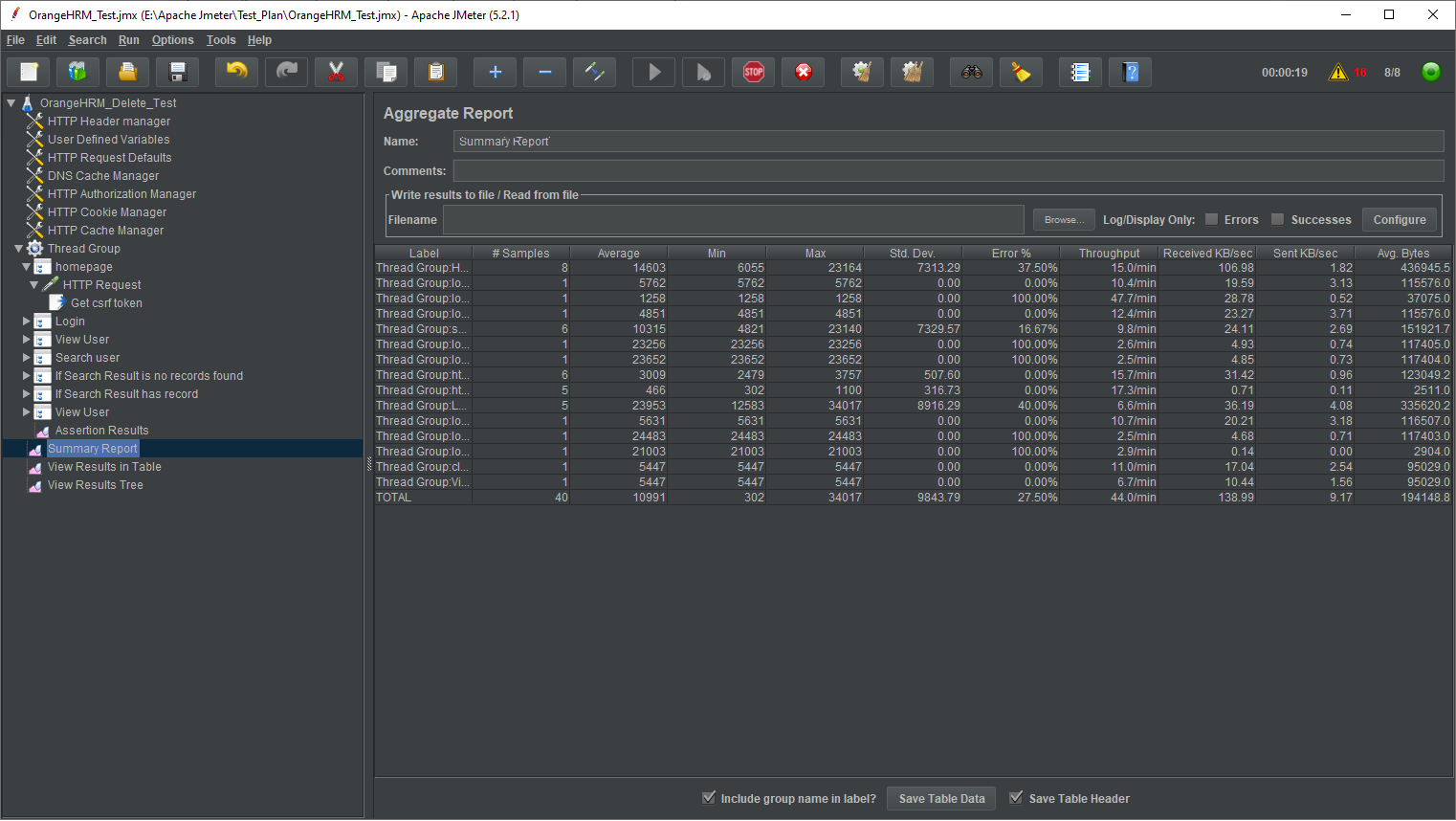
#### Aggregate Report



Hình 55: Aggregate Report

* Aggregate report tạo một bảng thống kê lại các request đã thực hiện trong Test Plan. Các thông số thống kê trong report bao gồm:
  + Label: Tên của từng request có trong Test Plan. Mặc định tất cả những request bị trùng tên trong Test Plan sẽ chỉ hiển thị một dòng duy nhất trong table, nên cần chú ý đặt tên các request phải khác nhau.
  + Samples: Tổng số lần chạy của request.
  + Average (ms): Thời gian phản hồi trung bình (Response time) của request, tính cho đến lần chạy cuối cùng. Ví dụ: Một HTTP request run tổng cộng 4 lần với các kết quả Response time lần lượt là 101ms, 106ms, 153ms, 128ms. Thì Response Time trung bình là 122ms.
  + Min (ms): Response Time thấp nhất của Request tính cho toàn bộ các lần run.
  + Max (ms): Response Time cao nhất của Request tính cho toàn bộ tất cả các lần run.
  + Median (ms): Median + 1 giá trị A, sẽ chia toàn bộ các giá trị thành 2 phần bằng nhau. Một phần sẽ chứa những giá trị < A, phần còn lại sẽ chứa những giá trị > A. Ý nghĩa của con số trong Median: sẽ có 50% số request có response time nhỏ hơn giá trị đó, và 50% request còn lại có response time lớn hơn.
  + 90% Line: 90% số request có Response time nhỏ hơn giá trị trong table, 10% còn lại thì lớn hơn.
  + 95% Line: 95% số request có Response time nhỏ hơn giá trị trong table, 5% còn lại thì lớn hơn.
  + 99% Line: 99% số request có Response time nhỏ hơn giá trị trong table, 1% còn lại thì lớn hơn.
  + Error %: Tỉ lệ số lượng request bị lỗi.
  + Throughput: Thông lượng. Con số này cho biết số lượng request mà server xử lí được trong 1 đơn vị thời gian.
  + Received KB/sec: Thông lượng nhận về mỗi giây.
  + Sent KB/sec: Thông lượng gửi đi mỗi giây.

#### Summary Report



Hình 56: Summary Report

* Tương tự với Aggregate Report, Summary Report cũng tạo bảng thống kê lại các hành động trong Test Plan, nhưng có một vài thông tin thống kê khác:
  + Std.Dev (Standard Deviation): Độ lêch chuẩn đo lường sự thay đổi của 1 tập hợp data, dựa trên thống kê.
  + Avg.Bytes: Dung lượng trung bình của 1 lần response tính bằng bytes.

#### Phân tích kết quả Report

* Dựa vào những thông số trong Aggregate/Summary Report, ta có thể đánh giá được hiệu năng của server test:
  + **Response Time** THẤP và Throughput THẤP 🡪 Trường hợp này sẽ không bao giờ xảy ra. Vì Response Time THẤP nghĩa là thời gian đáp ứng rất nhanh, nhưng Throughput THẤP lại chỉ ra rằng số request được xử lý rất ít. Điều này là vô lý.
  + **Response Time** THẤP and Throughput CAO 🡪 Đây là một kết quả lý tưởng. Thời gian xử lý thấp và số lượng request xử lý cùng đồng thời lại cao. Điều này chứng tỏ rằng Server đang rất tốt.
  + **Response Time** CAO and Throughput THẤP 🡪 Test chỉ ra rằng thời gian xử lý quá cao, và lượng request được xử lý lại rất thấp. Phải xem xét để improve về phía sever.
  + **Response Time** CAO and Throughput CAO 🡪 Throughput cao, tức là server đang làm việc rất tốt, vậy tại sao thời gian xử lý lại cũng cao (không tốt). Có thể vấn đề lúc này đến từ phía Client, hoặc cụ thể là đến từ JMeter, cần xem xét lại script test để đạt được kết quả tốt hơn.

### Configuration Elements

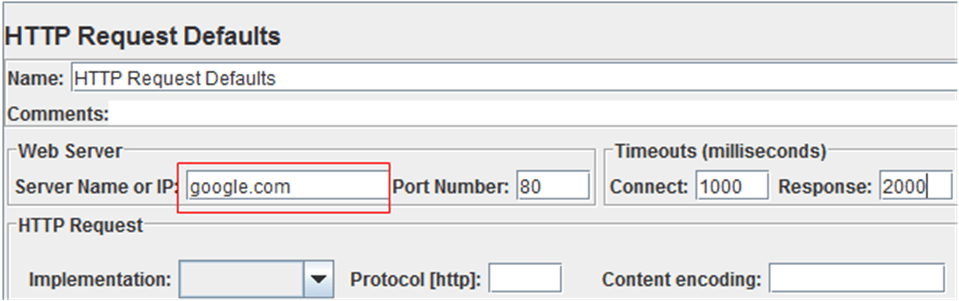
* Thiết lập ban đầu và biến để sử dụng trong Test Plan.

#### HTTP Cookie Manager

* Ví dụ ta sử dụng một trình duyệt để truy cập [www.google.com](http://www.google.com). Ta sẽ phải đăng nhập, khi đó username và password sẽ được lưu trữ trong máy tính dưới dạng cookie. Trong lần sau, khi ta truy cập trang đó sẽ không cần đăng nhập lại nữa, bởi trình duyệ đã sử dụng dữ liệu ở cookie để đăng nhập.
* HTTP Cookie Manager có các tính năng như một trình duyệt web. Nếu ta có một HTTP request và phản hồi có chứa cookie, Cookie Manager sẽ tự động lưu trữ cookie đó và sử dụng cho các request khác của website.

#### HTTP Request Defaults

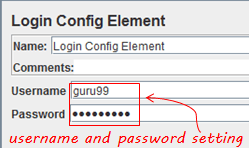
* Element này dùng để thiết lập các giá trị mặc đinh mà HTTP Request sử dụng.
* Ví dụ: Ta cần gửi 100 HTTP request tới google.com. Ta cần nhập link google.com cho 100 request. Thay vì làm điều đó, ta có thể thêm 1 HTTP Request Default với trường Server Name or IP gán là google.com:



Hình 57: HTTP Request Defaults

#### Login Config Element

* Login Config Element giúp ta thêm hoặc ghi đè username và password trong sampler.
* Ví dụ: ta cần mô phỏng 1 user đăng nhập tới [www.facebook.com](http://www.facebook.com). Ta có thể sử dụng Login Config Element để thêm username và password trong một user request.



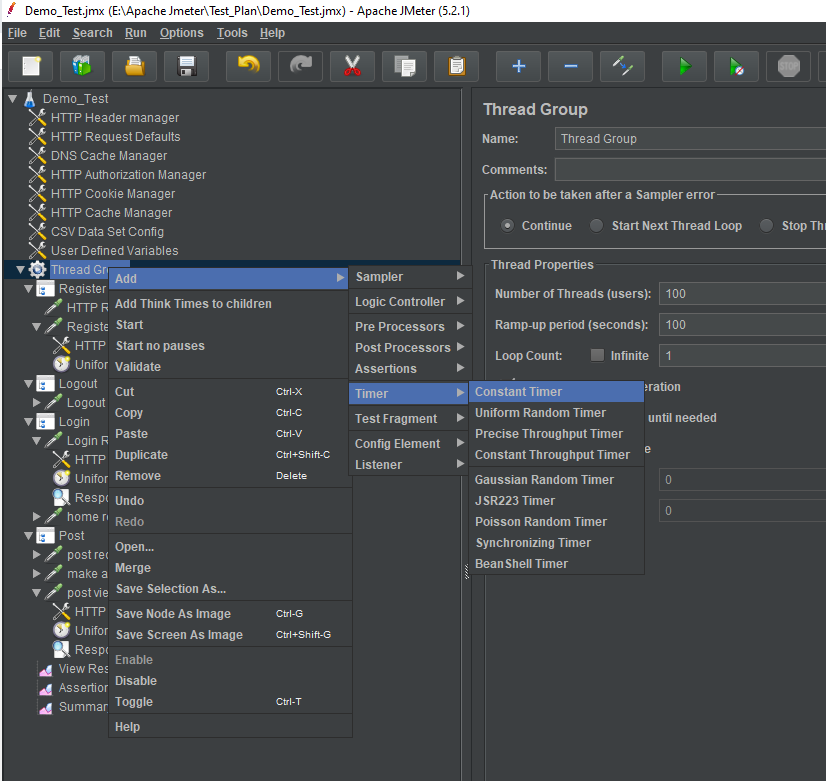
Hình 58: Example Login Config Element

### Timers

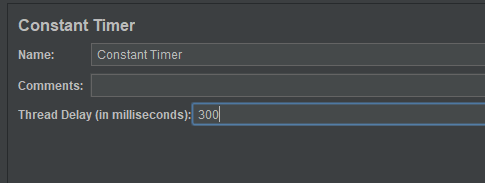
* JMeter mặc định gửi request liên tục mà không trì hoãn giữa mỗi request. Trong trường hợp đó, JMeter có thể làm tràn test server bởi thực hiện quá nhiều request trong một khoảng thời gian ngắn, dẫn đến hiện tượng quá tải trên server.
* Timers cho phép JMeter trì hoãn giữa mỗi lần request mà một thread tạo, đồng thời giải quyết được vấn đề quá tải của server.
* Thực tế rất ít xảy ra trường hợp có nhiều người dùng cùng đăng nhập 1 hệ thống, đúng 1 thời điểm chính xác, nên timer có thể được dùng để mô phỏng khả năng đợi của ngươi dùng, hay còn gọi là “Think time”.

#### Constant Timer

* Constant timer trì hoãn mỗi một user request với một khoảng thời gian cố đinh.
* Để thêm một Constant timer cho Test Plan, chọn Thread Group 🡪 Add 🡪 Timer 🡪 Constant Timer.



Hình 59: Adding Constant Timer

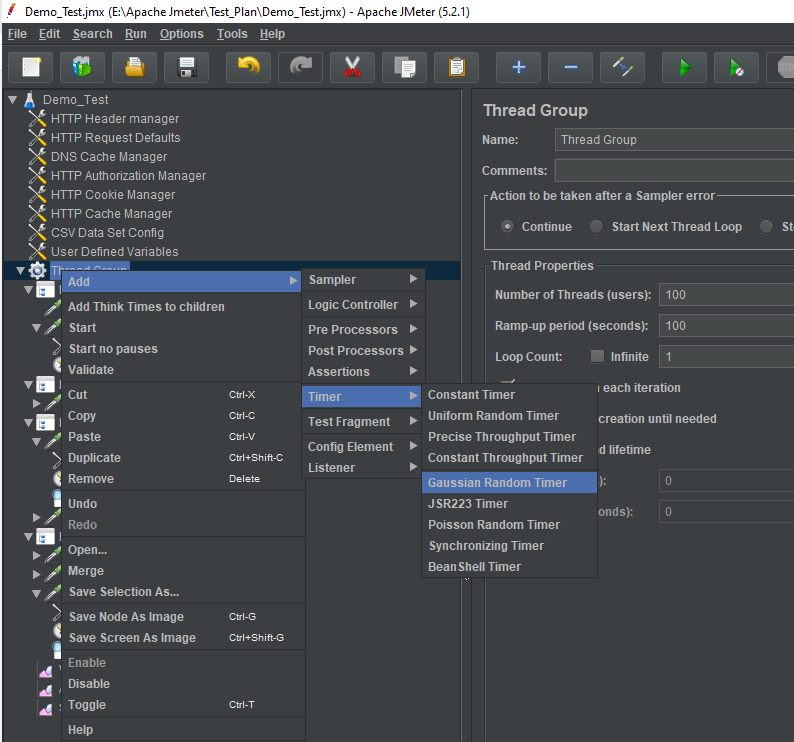


Hình 60: Constant Timer Panel

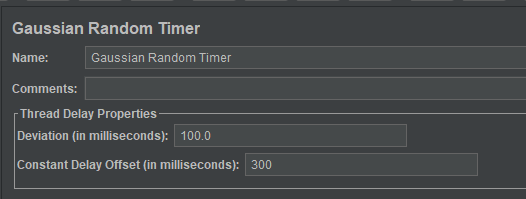
* Để thiết lập Constant Timer, ta chỉ cần đặt giá trị thời gian tại Thread Delay (theo hình 49).

#### Gaussian Random Timer

* Để thêm một Gaussian random timer, ta chon Thread Group 🡪 Add 🡪 Timer 🡪 Gassian random timer



Hình 61: Adding Gaussian Timer



Hình 62: Setting Gaussian Random Timer

* Ý nghĩa của các Parameter:
  + Name: tên hiển thị của timer
  + Deviations (ms): Tham số của hàm phân phối Gaussian
  + Constant Delay Offset (ms): Thời gian bổ sung
* Gaussian random timer trì hoãn mỗi user request bằng một khoảng thời gian ngẫu nhiên. Thời gian trì hoãn này được tính theo công thức:
  + Công thức trên được tính khi U và V nhận giá trị thỏa mãn S = U + V > 1 (U, V = [-1; 1])
* Gaussian Time = X( or Y) \* deviation + Constant Delay Offset

#### Uniform Random Timer

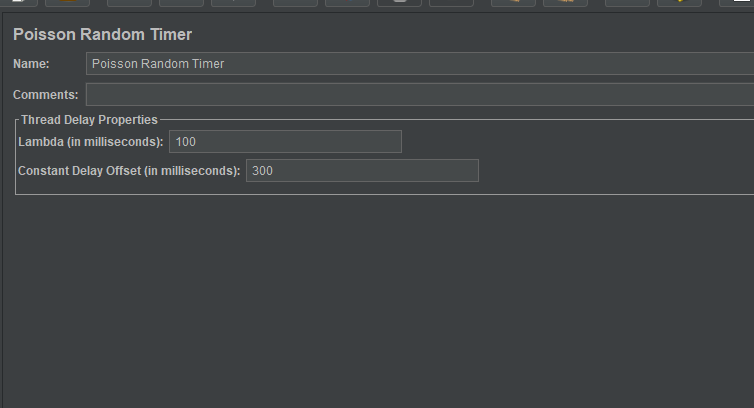


Hình 63: Setting Uniform Random Timer

* Ý nghĩa của các Parameter:
  + Name: Tên hiển thị của timer
  + Random Delay Maximum: Thời gian tối đa dùng để trì hoãn
  + Constant Delay Offset (ms): thời gian bổ sung
  + Tổng thời gian delay = Random Delay Time + Constant Delay Offset
* Uniform Random Timer sẽ khởi tạo một giá trị xác suất (P) phân bố trong khoảng [0; 1) mỗi lần chạy. Khi đó giá trị của Uniform Random Timer sẽ được tính theo công thức:

Time = P \* Random Delay Maximum + Constant Delay Offset.

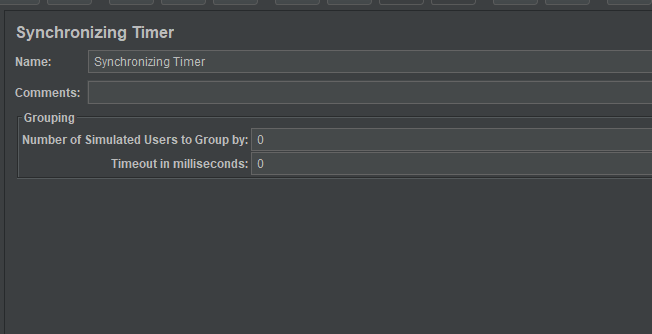
#### Poisson Random Timer



Hình 64: Poisson Random Timer

* Ý nghĩa của các parameter:
  + Lambda: Thời gian xác định hệ số Poisson
  + Constant Delay Offset: Thời gian bổ sung
* Một Poisson Random Timer sử dụng số ngẫu nhiên tiếp theo từ Phân bố Poisson tính toán từ Lambda (λ), cộng với “Constant Delay Offset”:
* Poisson Random Timer = k + Constant Delay Offset

#### Synchronizing Timer

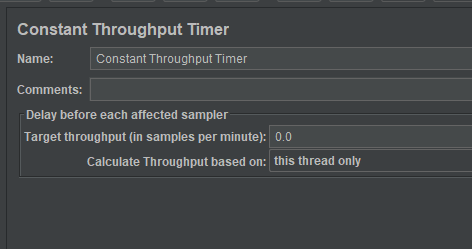


Hình 65: Synchronizing Timer Panel

* Một Synchronizing Timer tạm dừng tất cả các Thread nếu số lượng người dùng giả lập hiện tại nhỏ hơn Number of simulated User to Group by đã set.
* Timeout in miliseconds : dùng để tạo timeout trong trường hợp hệ thống mãi không thỏa mãn điều kiện đã nói ở trên, khi có time out ta sẽ đảm bảo hệ thống không bị dừng.

#### Constant Throughput Timer

* Timer này cho phép ta giữ tổng thông lượng không đổi. Nếu server không chịu được tải lớn, thông lượng sẽ thấp. Thông lượng có thể giam nếu các thời gian khác mâu thuẫn với Throughput Time.
* Giá trị của Throughput time có thể thay đổi chứ không cố định.
* Các tùy chọn có thể sử dụng để thay đổi giá trị throughput time:
  + Sử dụng biến đếm
  + Sử dụng hàm JavaScript hoặc Beanshell
  + Sử dụng điều khiển server Beanshell để thay đổi 1 thuộc tính JMeter

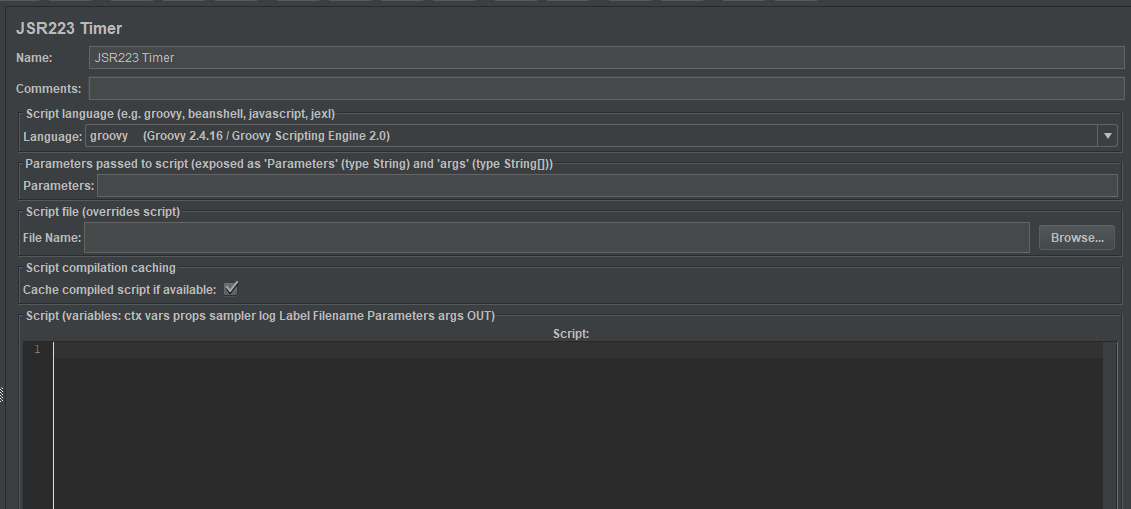


Hình 66: Constant Throughput Timer

* Các tham số trong panel:
  + Name: Tên của Timer
  + Target throughput: Thông lượng mà thời gian cần tạo được
  + Calculate Throughput based on:
    - This thread only: mỗi thread sẽ cố gắng duy trì target throughput. Tổng thể thông lương sẽ tỉ lệ với với số lượng thread đang hoạt động
    - All active threads in current thread group: target throughput được chia ra theo các thread trong group. Mỗi thread sẽ delay khi cần, dựa trên lần cuối nó chạy.
    - All active threads: target throughput được chia ra theo các thread trong Thread Groups, mỗi thread delay dựa trên lần chạy cuối cùng. Trong trường hợp này, mỗi Thread Group sẽ cần một Constant Throughput Timer với các thiết lập như nhau.
    - All active threads in current thread group (shared): giống như trên, nhưng mỗi thread delay dựa trên lần chạy cuối cùng của bất cứ thread nào trong group.
    - All active threads (shared): mỗi thread delay dựa trên lần chạy cuối cùng của bất cứ thread nào trong group.

#### Beanshell, JSR223 Timers

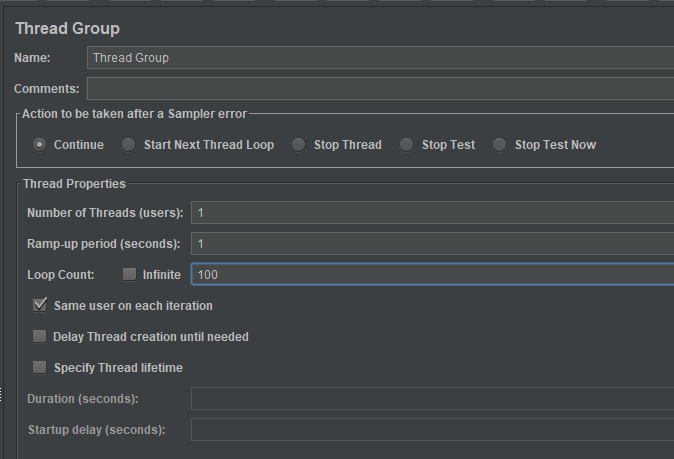
* Các timer này cần phải tự thực hiện logic delay thread bằng một trong các ngôn ngữ lập trình được hỗ trợ. Tuy nhiên, nếu cần xác định “think time” dựa trên một số thuật toán hiện không được cung cấp bởi JMeter, ta có thể tạo triển khai thuật toán của riêng mình bằng một trong những timer này. JMeter hỗ trợ các ngôn ngữ Beanshell, JavaScript và JEXL. ta có thể thêm các thư viện có liên quan để kích hoạt hỗ trợ cho các ngôn ngữ khác. Nói tóm lại, để sử dụng bộ đếm thời gian dựa trên kịch bản, bạn cần phải:
  + Thực hiện tính toán độ trễ bằng thuật toán
  + Điều khiển luồng để “sleep” khoảng thời gian được tính bằng cách sử dụng câu lệnh return, trả về giá trị được tính toán (có thể thực hiện trực tiếp trong timer, tuy nhiên, điều đó không được khuyến khích).



Hình 67: JSR223 Timer

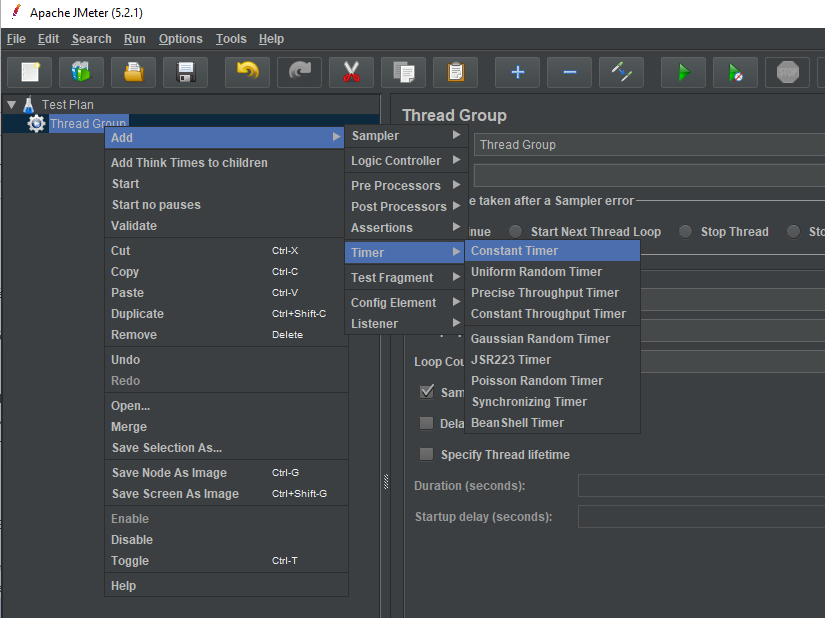
#### Ví dụ sử dụng Constant Timer

* Ta sẽ sử dụng Constant Timer để tạo một khoảng thời gian trì hoãn giữa mỗi user request đến google.com
* Bước 1: Thêm Thread Group
  + Chuột phải vào Test Plan và chọn Add 🡪 Thread (Users) 🡪 Thread Group
  + Set các giá trị cho Thread như hình dưới đây, JMeter sẽ tạo một user request tới <http://www.google.com> 100 lần:



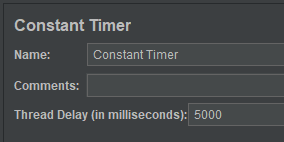
Hình 68: Adding Thread Group

* Bước 2: Thêm JMeter Elements
  + Thêm HTTP Request Default
  + Thêm HTTP Request
* Bước 3: Thêm Constant Timer
  + Chuột phải Thread Group 🡪 Timer 🡪 Constant Timer



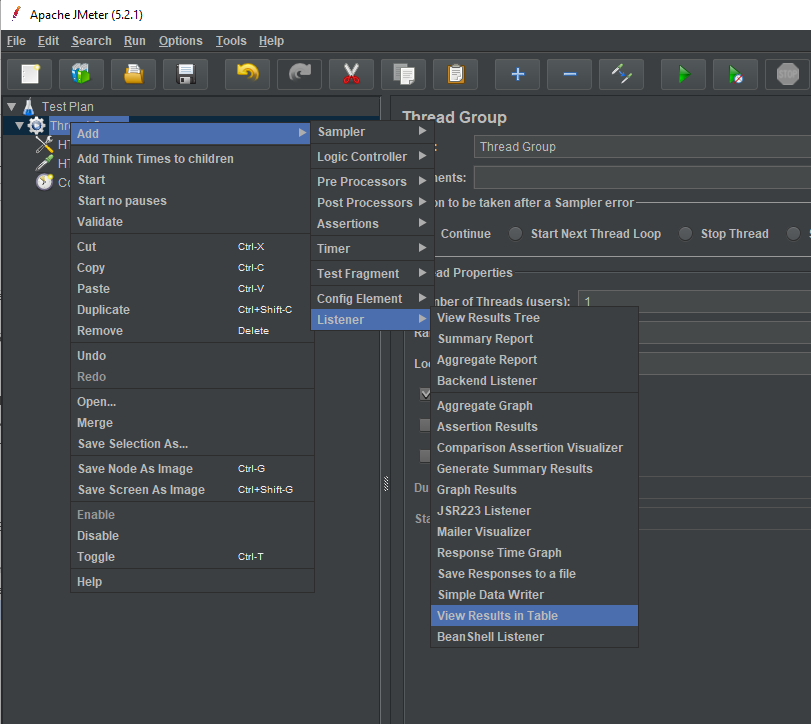
Hình 69: Adding Constant Timer

* + Cấu hình Thread Delay là 5000 ms.



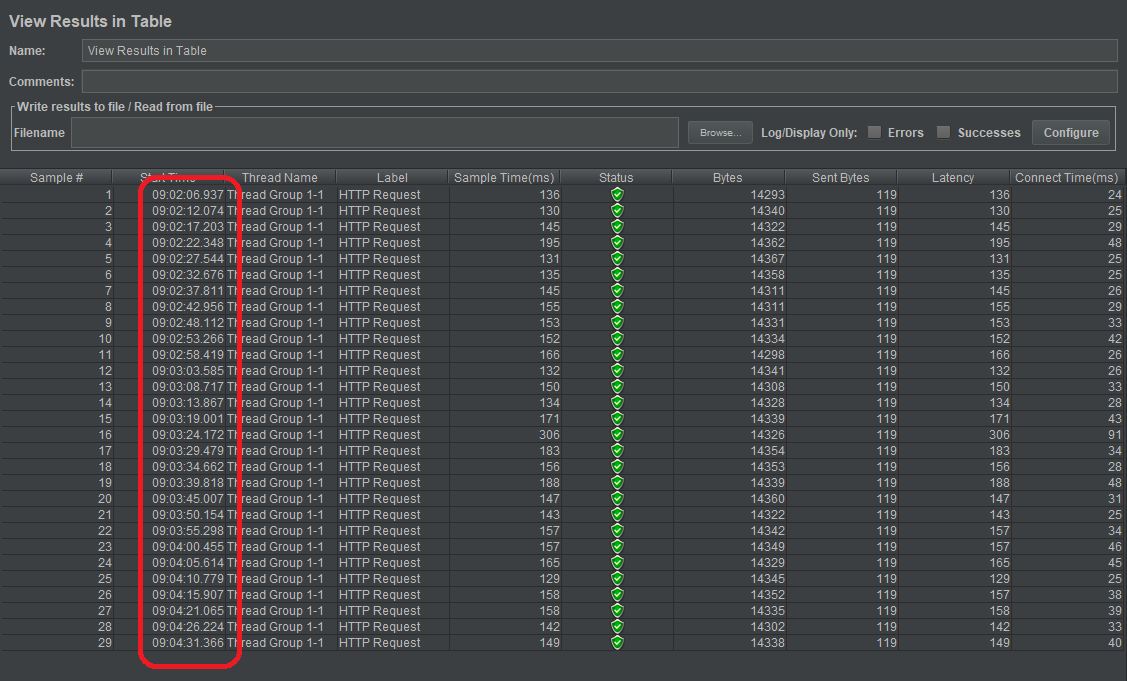
Hình 70: Configuring Constant Timer

* Bước 4: Thêm View Results in Table
  + View Results In Table hiển thị kết quả test ở dạng bảng
  + Chuột phải Thread Group 🡪 Add 🡪 Listener 🡪 View Result in Table



Hình 71: Adding View Results in Table

* Bước 5: Run Test Plan



Hình 72: View Results in Table

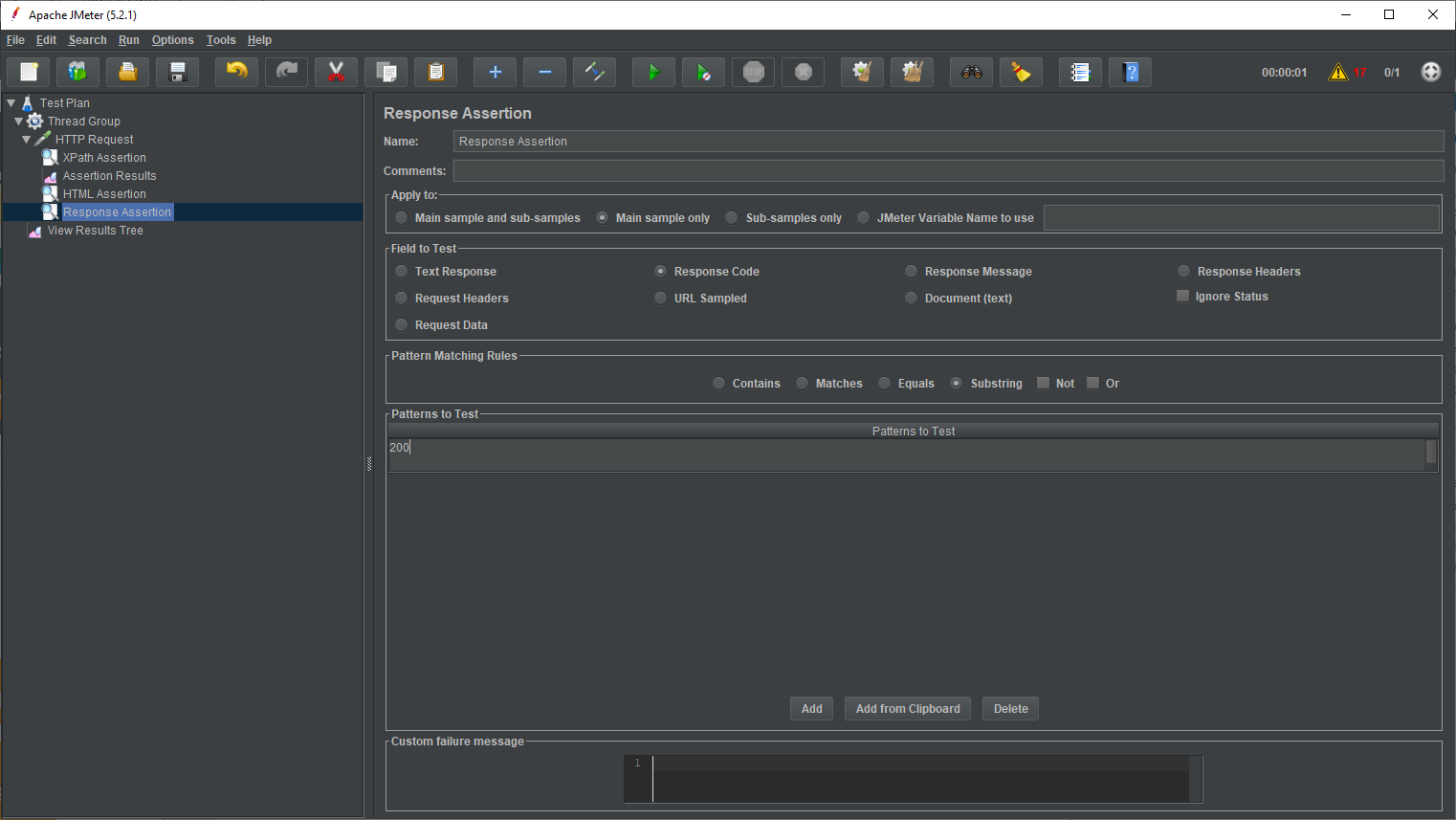
* + Kết quả cho thấy, ở mỗi sample có một khoảng thời gian delay là 5000 ms.

### Assertions

* Assertion giúp ta xác nhận kết quả trả về từ server khi test so với kết quả mong muốn.

#### Response Assertion

* Sử dụng để so sánh kết quả mong muốn và kết quả trả về của một request dựa vào: Text Response, Response code, Response Hearders,…
* Response Assertion cho phép ta thêm các Strings mẫu để so sánh với các trường của server phản hồi
* Ví dụ: khi gửi một user request tới trang chủ Google và lấy được phản hồi từ server, ta có thể sử dụng Response Assertion để xác nhận nếu server phản hồi có chứa String mong muốn (ví dụ như “OK”).

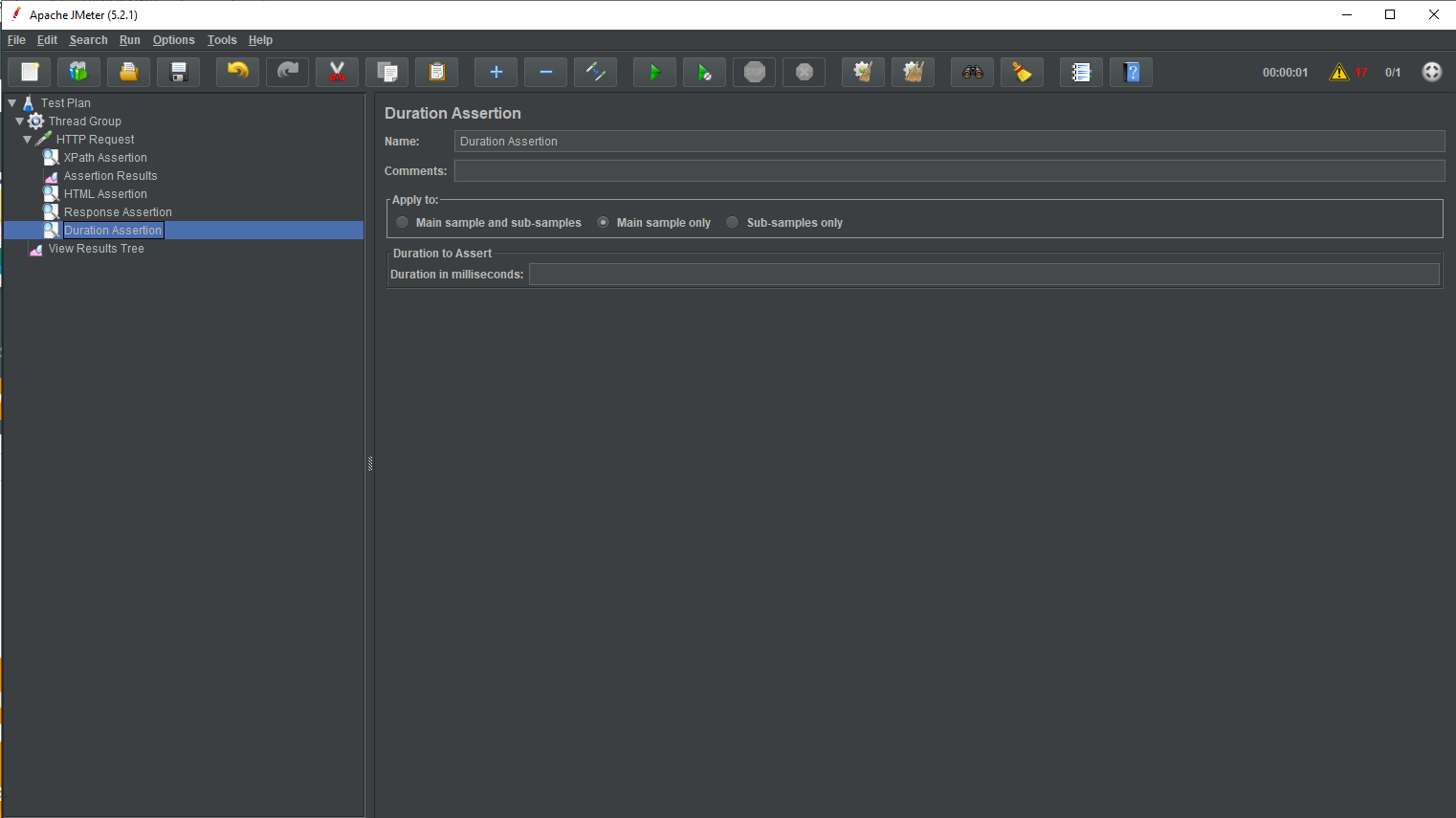


Hình 73: Response Assertion

* Response Assertion là một trong những Assertion được các Performance Tester sử dụng nhiều nhất. Nó bao gồm các chức năng:
  + Field to Test: Lựa chọn kiểu dữ liệu phản hồi từ server để so sánh với kết quả mong muốn (Text Reponse, Response code, Document (text), Response Message, Response Headers, Request Headers, URL Sample,…).
  + Pattern Matching Rules: Kiểm tra tất cả các phản hồi so với mẫu theo điều kiện (Contains, Matches, Equals, Substring, Not, Or).
  + Pattern to Test: Giá trị chuỗi mẫu dùng để so sánh.
* Response Assertion sử dụng CPU tương đối.

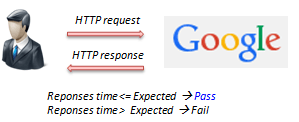
#### Duration Assertion

* Duration Assertion dùng để kiểm tra mỗi lần server phản hồi trong khoảng thời gian cho trước. Bất cứ phản hồi nào mà mất nhiều thời gian hơn thời gian đã cho thì được đánh dấu là phản hồi lỗi.



Hình 74: Duration Assertion

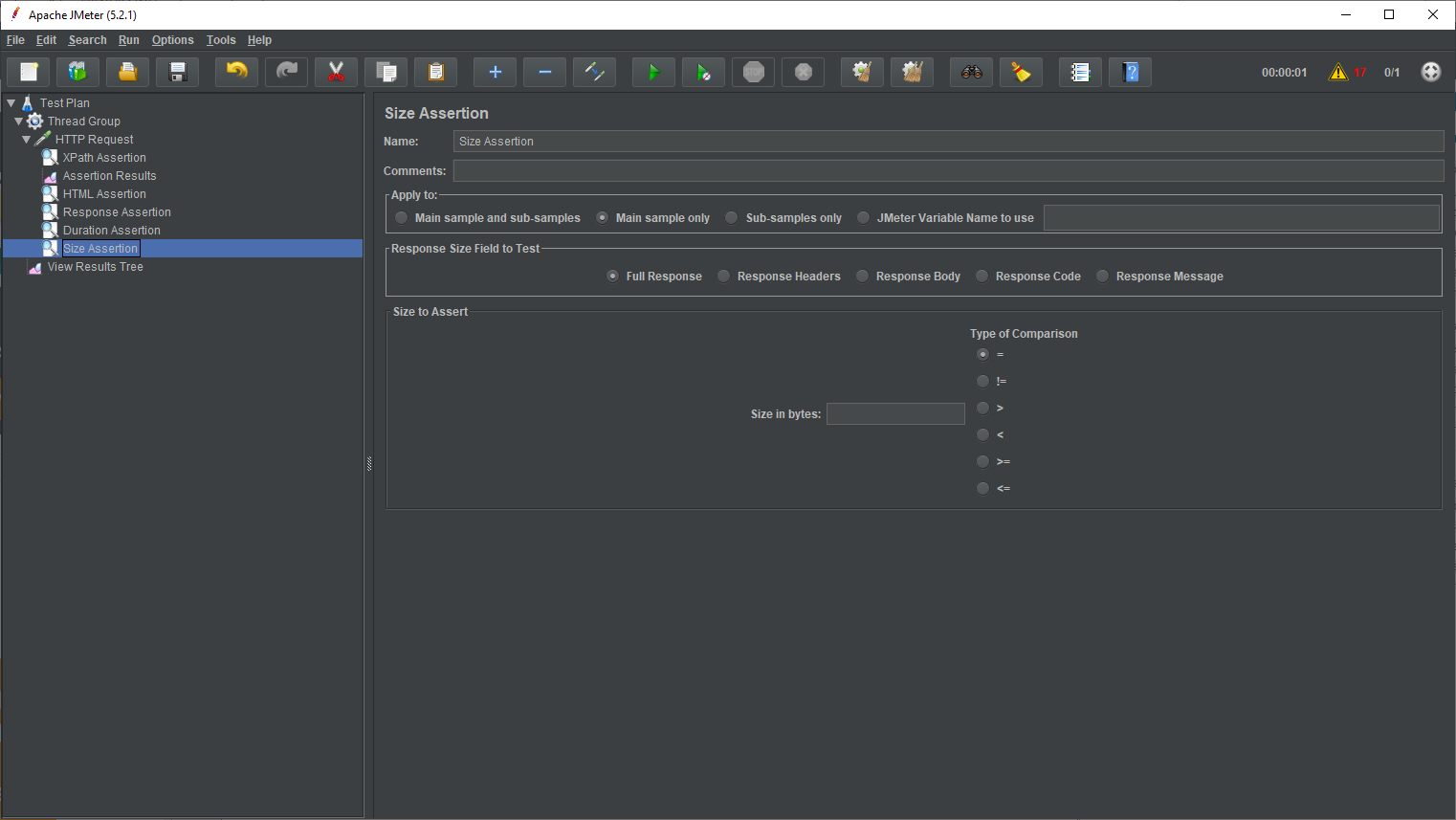
* Duration Assertion sử dụng ít CPU.
* Ví dụ: một user request gửi tới [www.google.com](http://www.google.com) bằng JMeter và lấy một phản hồi trong khoảng thời gian mong đợi là 5ms thì Test Case pass, ngược lại thì test case fail.



Hình 75: Duration Assertion

#### Size Assertion

* Size Assertion kiểm tra xem mỗi lần server phản hồi có chưa số lượng byte mong muốn hay không. Ta có thể cố định kích thước bằng, lớn hơn, nhỏ hơn hoặc không bằng với lượng byte cho trước.
* JMeter gửi một user request tới [www.google.com](http://www.google.com) và lấy gói phần hồi với kích thước nhỏ hơn mong muốn (5000 bytes) thì test case đó pass, ngược lại test case false.

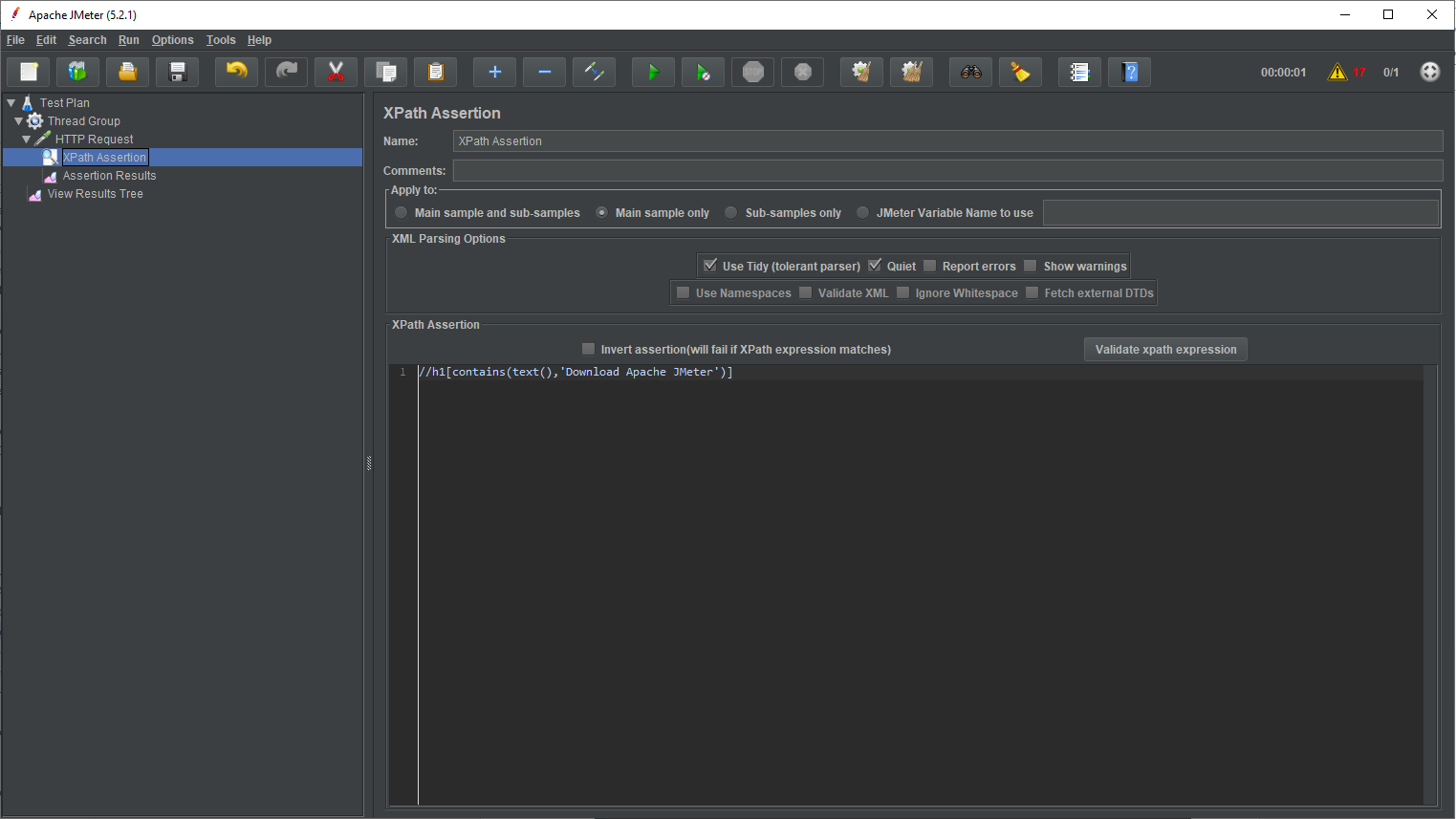


Hình 76: Size Assertion

* Size Assertion nên được sử dụng khi kiểm tra phản hồi lượng dữ liệu tải 1 file từ server. Kiểm tra file chắc chắn đã được tải về đầy đủ dữ liệu.
* Size Assertion sử dụng CPU ít.

#### Xpath Assertion

* Sử dụng để so sánh kết quả mong muốn và kết quả trả về của một Xpath có đúng hay không? Assertion này thích hợp khi thực hiện test SOAP Web Services.
* Xpath Assertion sử dụng lượng CPU cao.

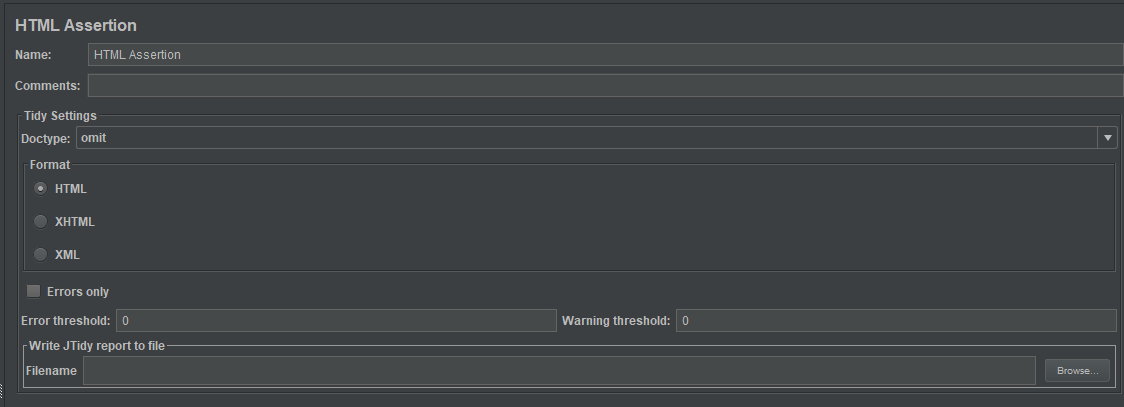


Hình 77: Xpath Assertion Example

* Các chức năng trong panel:
  + XML Parsing Options:
    - Use Tidy: Click chọn nếu XML không đúng cú pháp
    - Use Namespaces:
    - Validate XML: Đảm bảo XML đúng cú pháp, nếu không test sẽ fail
    - Fetch External DTDs: tải xuống

#### HTML Assertion

* HTML Assertion cho phép người dùng kiểm tra cú pháp HTML của dữ liệu phản hồi.
* Dữ liệu phản hồi phải chứa cú pháp HTML



Hình 78: HTML Assertion

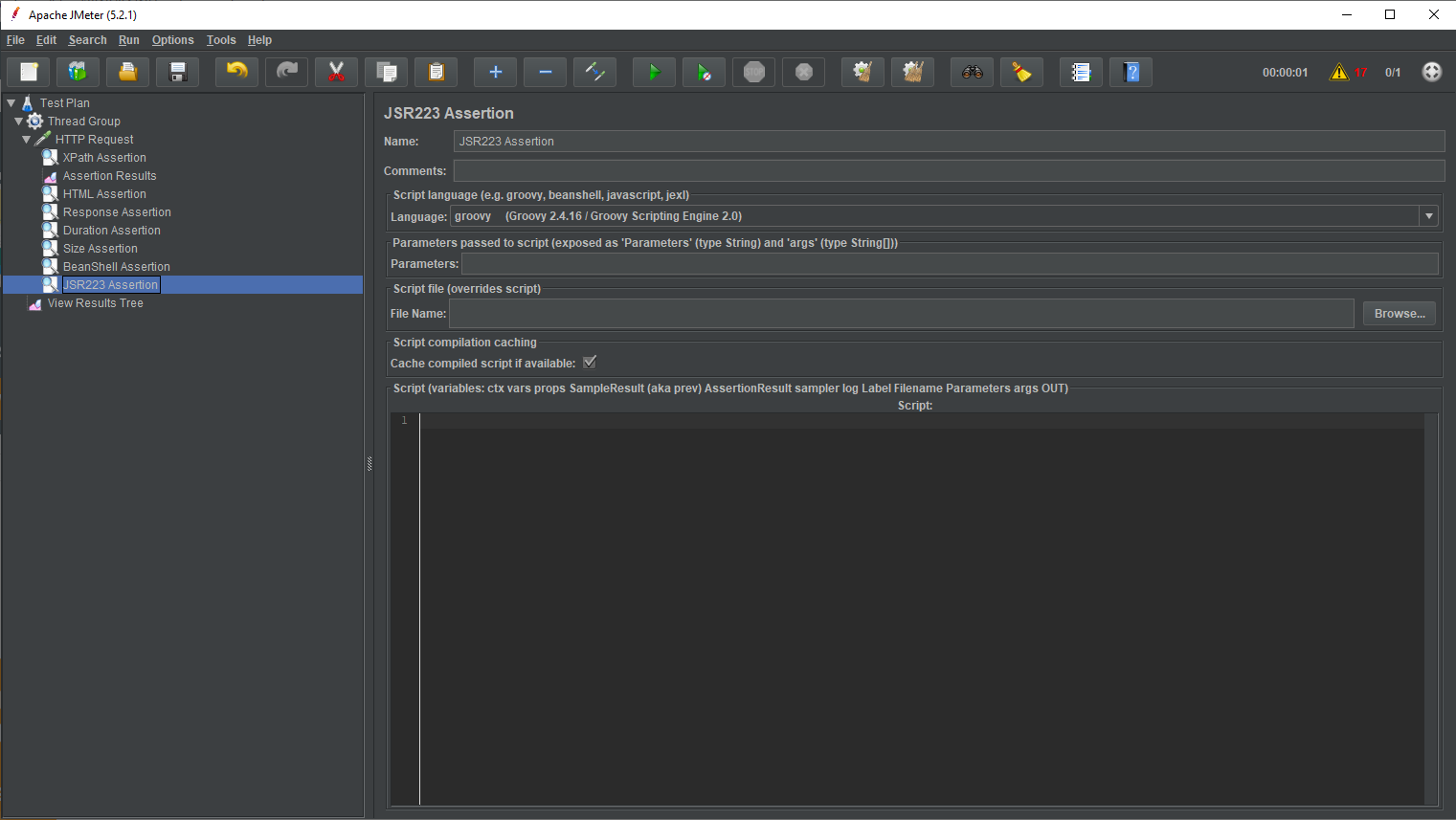
#### Beanshell Assertion

* Beanshell Assertion cho phép thực hiện kiểm tra xác minh một sampler sử dụng script Beanshell. Beanshell cung cấp các biến sau:
  + Failure – Boolean: Cho biết bộ lấy mẫu được coi là thành công.
  + FailureMessage – String: Đây là một thông báo hiển thị khi có lỗi assertion.
  + ResponseData – byte: Một mảng byte biểu diễn dữ liệu phản hồi.
  + ResponseCode – String: Biểu diễn Response code.
  + ResponseMessage – String: Giữ thông báo phản hồi.
  + ResponseHeader – String: Chứa các Response Header.
  + SampleResult: Đây là class JMeter SampleResult chứa các kết quả lấy mẫu. Khi có nhiều sampler cha (ví dụ như Transaction Controller hoặc Retrieve all Embedded Resources), phương thức này trả về một mảng các nested request.
* Ví dụ, đoạn script Beanshell sau sẽ trả về một lỗi nếu từ “blazemeter” không xuất hiện trên URL:

**String path = SampleResult.getURL().getPath();  
  
if (!path.contains("blazemeter")) {  
   Failure = true;  
   FailureMessage = "URL Path: didn't contain \"blazemeter\"" + System.getProperty("line.separator") + "URL Path detected: " + path;  
}**

* Beanshell Assertion sử dụng CPU cao, nên dùng JSR223 Assertion nếu cần tạo các script assertion để tiết kiệm CPU.

#### JSR223 Assertion

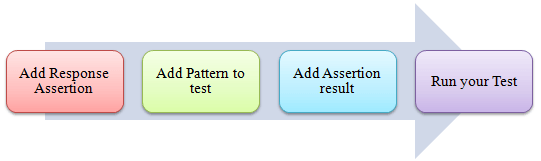


Hình 79: JSR223 Assertion

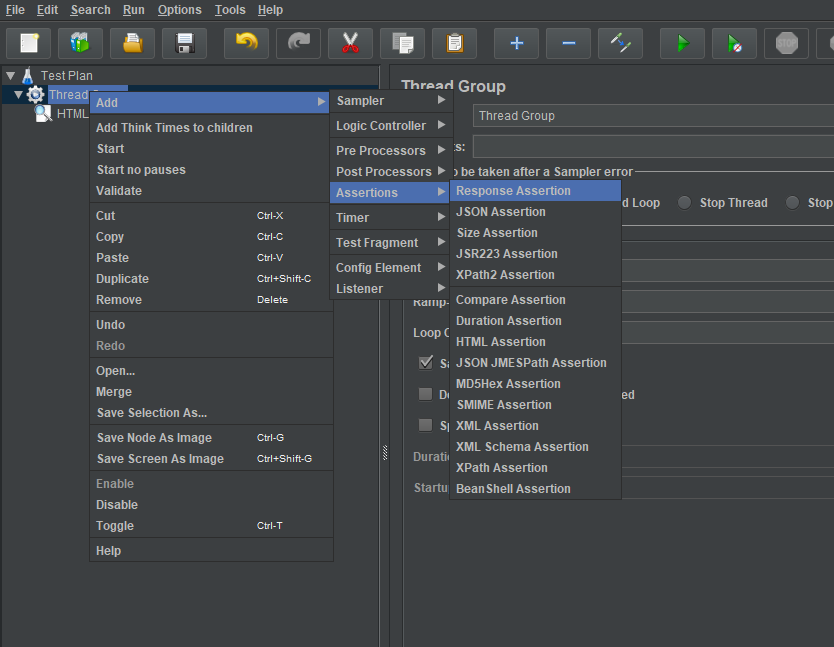
* JSR223 Assertion cung cấp linh hoạt hơn nhiều so với Beanshell Assertion khi viết script (hỗ trợ beanshell, groovy, java, jexl và Javascript và tốc độ xử lí nhanh hơn beanshell.
* Trong hầu hết các trường hợp, ta thường bỏ qua thiết lập Parameter và Script File. Hai thiết lập này đượ sử dụng khi cần chia sẻ script hàm qua nhiều JMX project, để có thể sử dụng lại bằng cách lưu script thành một file rời.
* JSR223 Assertion không tiêu tốn nhiều CPU như Beanshell Assertion nên là lựa chọn hoàn hảo khi cần viết script cho Assertion.

#### Ví dụ sử dụng Response Assertion

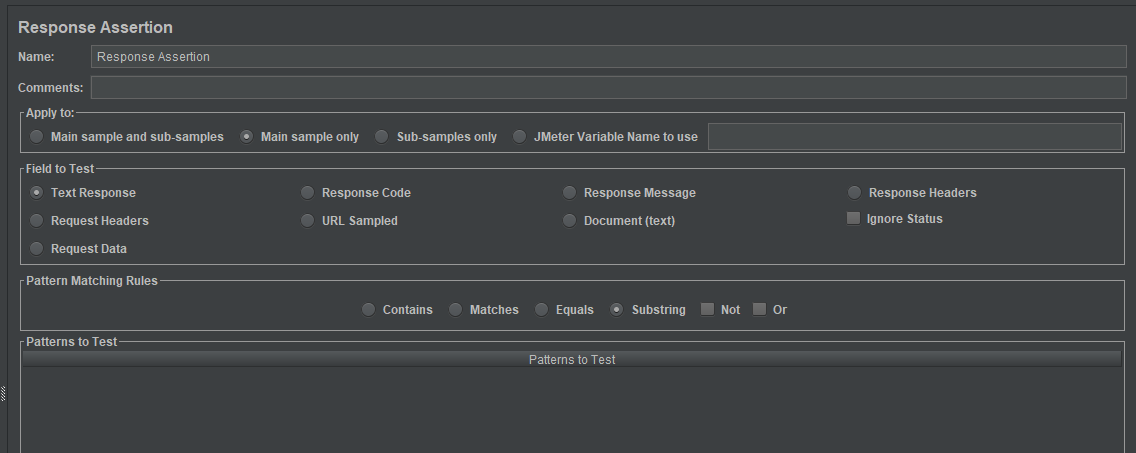
* Ở ví dụ này, ta sử dụng Response Assertion để so sánh gói phản hồi từ [www.google.com](http://www.google.com) có khớp với chuỗi kí tự mong muốn mà ta mong đợi hay không.
* Luồng đi:



* Response Assertion Panel cho phép ta thêm chuỗi kí tự mẫu dùng để so sánh với giá trị các trường của phản hồi.
* Bước 1: Add Response Assertion
  + Thêm Http Request, Http Request Defaults như ở mục 5.
  + Chuột phải vào Thread Group 🡪 Add 🡪 Assertions 🡪 Response Assertion

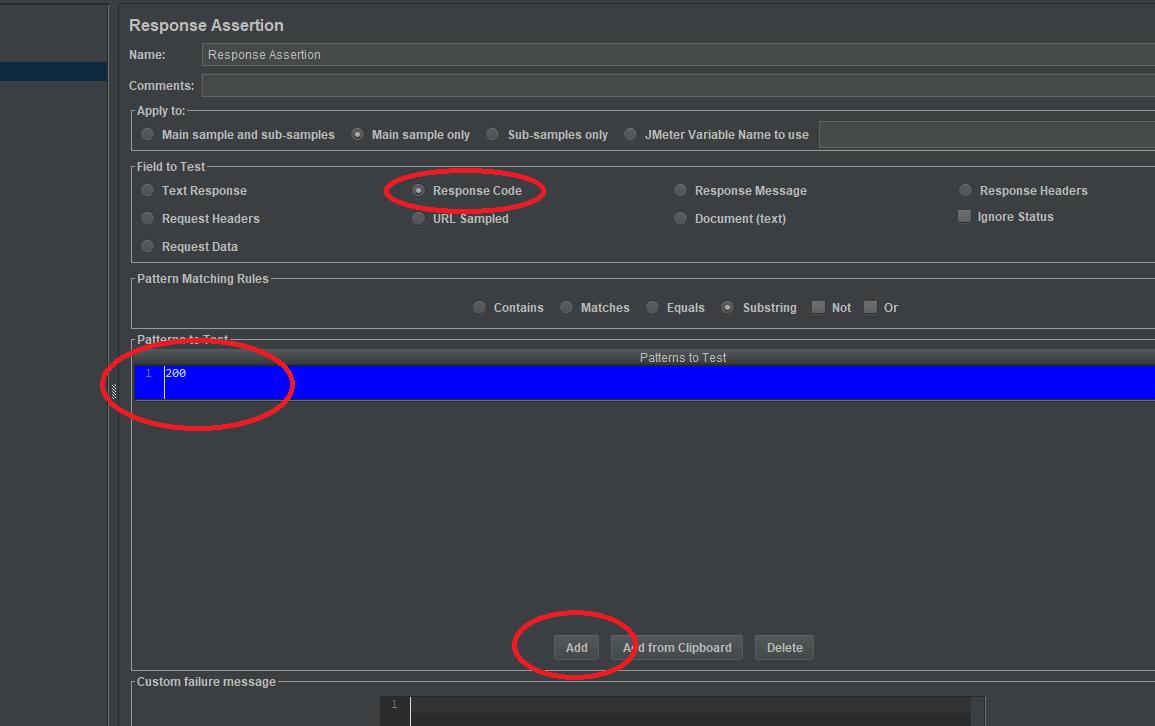


Hình 80: Add Response Assertion



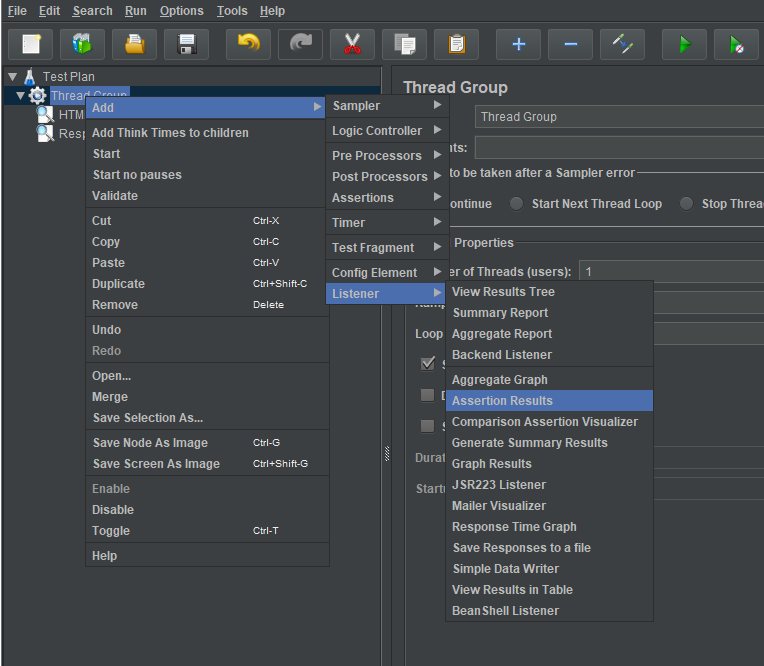
Hình 81: Response Assertion Panel

* + Response Field to Test:
    - Text Response: dữ liệu phản hồi dạng text
    - Document: dữ liệu phản hồi là tài liệu dạng text
    - Response Code: mã phản hồi
    - Response Message: tin nhắn phản hồi
    - Response Header: header phản hồi
  + Partterns To Test: danh sách parttern để kiểm tra (được add bởi user).
* Bước 2: Add Parttern to test
  + Ví dụ này ta kiểm tra sự phản hồi của server, cụ thể là kiểm tra response code.
  + Khi gửi một request đến server Google, server có thể trả về các response code sau:
    - 404: Server Error
    - 200: Server OK
    - 302: Web server redirects to other pages. Kết quả này thường xảy ra khi ta truy cập google ở ngoài khu vực USA. Google sẽ chuyển hướng sang website chỉ định của từng quốc gia. Ví dụ: google.com sẽ chuyển hướng sang google.co.in ở Ấn Độ.
  + Giả sử ta muốn xác nhận server google.com sẽ phản hồi code chứ mã 200:
    - Ở Response Field To Test, chọn Response Code.
    - Tại Response Assertion Panel, chọn Add 🡪 nhập giá trị 200 trong Parttern to Test.

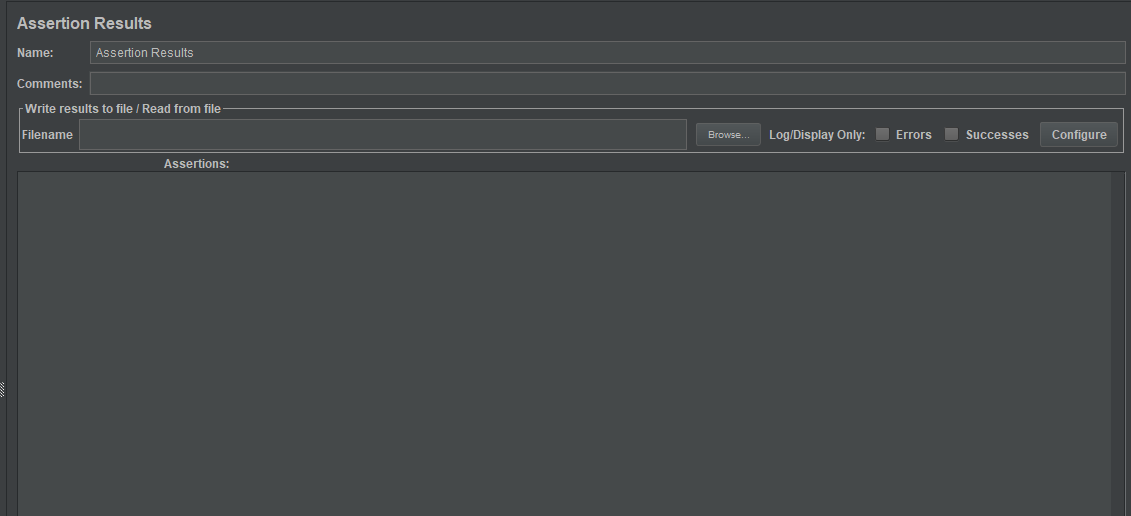


Hình 82: Add Parttern to test

* Bước 3: Add Assertion Results
  + Chuột phải vào Thread Group 🡪 Add 🡪 Listener 🡪 Assertion Results
  + Kết quả trả về sẽ được hiển thị ở vùng Assertions

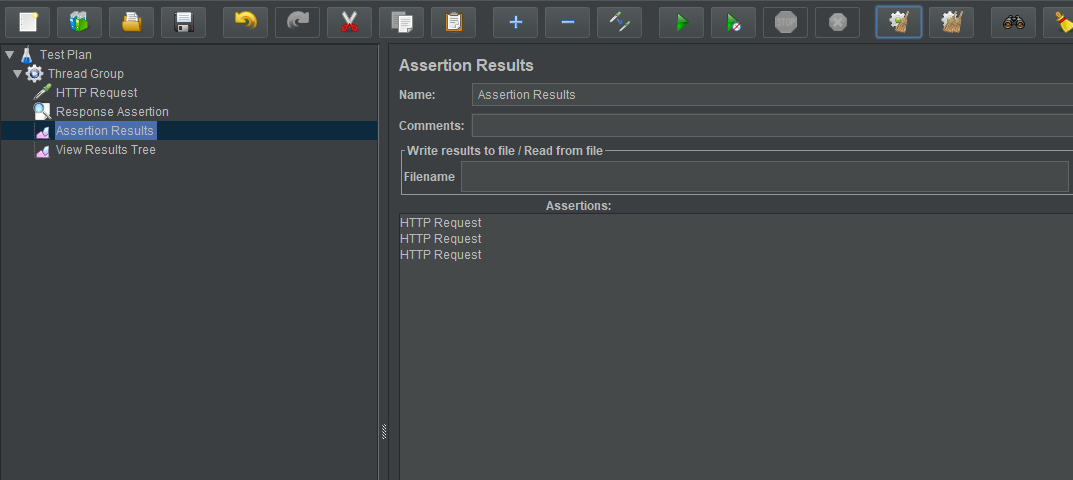


Hình 83: Add Assertion Results



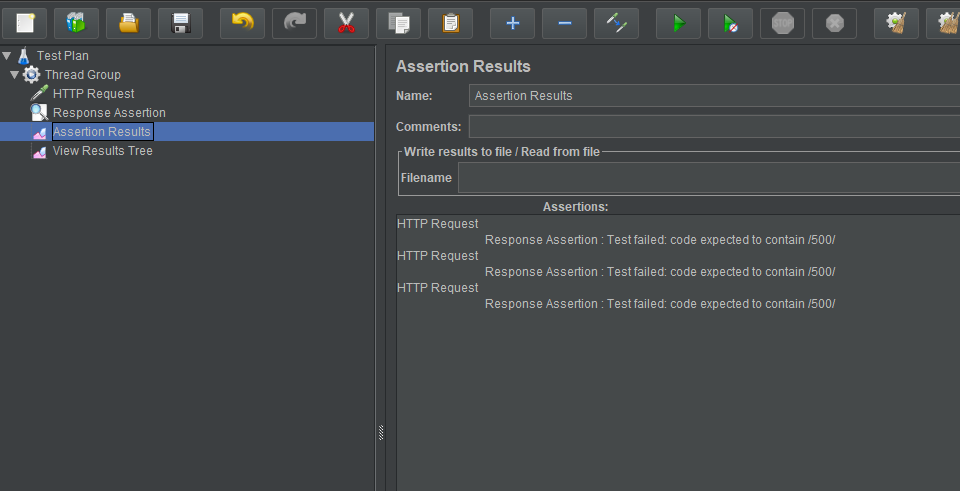
Hình 84: Assertion Result Panel

* Bước 4: Run Test Plan
  + Chọn Run hoặc Ctrl + R, mở cửa sổ Assertion Results để xem kết quả.
  + Nếu Google server phản hồi mã có có chứa “200” thì test case pass:



Hình 85: Response code contains 200 - Server OK

* + Ta sẽ ví dụ một trường hợp Assertion trả về kết quả lỗi. Vì danh sách mã lỗi của server google không có giá trị “500”, ta sẽ thay giá trị ở Patterns to Test trong Response Assertion Panel thành 500 và kiểm tra Assertion Results.



Hình 86: Response code does not contain 500 - Server is not OK

### Processor: Pre – processor and Post – Processor

#### PreProcessor

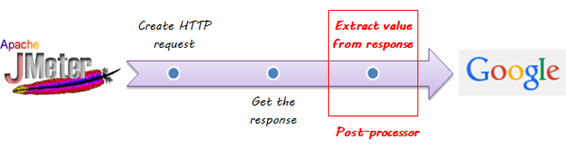
* Pre – Processor thực thi một số hành động trước khi tạo Sampler Request.
* Ví dụ đơn giản: ta muốn JMeter duyệt thông qua trang web đang test, kiểm tra tất cả các link trên page và trả về HTML. Ta thêm một số action như “HTML link parser” trong controller trước khi tạo một HTTP request.



Hình 87: PreProcessor

#### PostProcessor

* PostProcessor thực thi một số hành động sau khi tạo Sampler Request.
* Giả sử: JMeter gửi một HTTP Request tới web server (google) và lấy được phản hồi. Ta muốn JMeter dừng lại việc test nếu phản hồi là lỗi, khi đó ta sẽ sử dụng Post – Processor để thực hiện công việc như mô tả dưới đây:

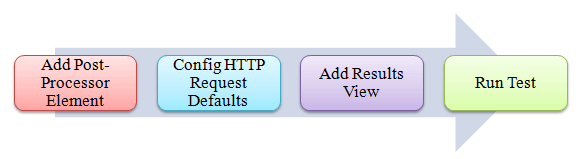


Hình 88: Post Processor

#### Post Processor Example

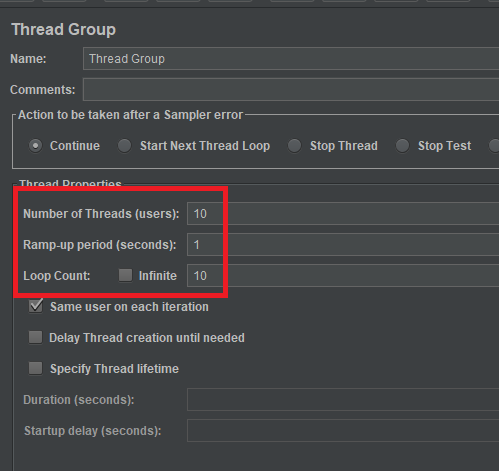
* Hướng dẫn này sẽ trình bày các bước để sử dụng PostProcessor trong JMeter. Bắt đầu với script test đơn giản sau:

1. JMeter gửi một HTTP request tới server google
2. JMeter lấy một phản hồi từ server google
3. Nếu server trả về một phản hồi lỗi, JMeter sẽ dừng test
4. Nếu server phản hồi OK, JMeter sẽ tiếp tục test



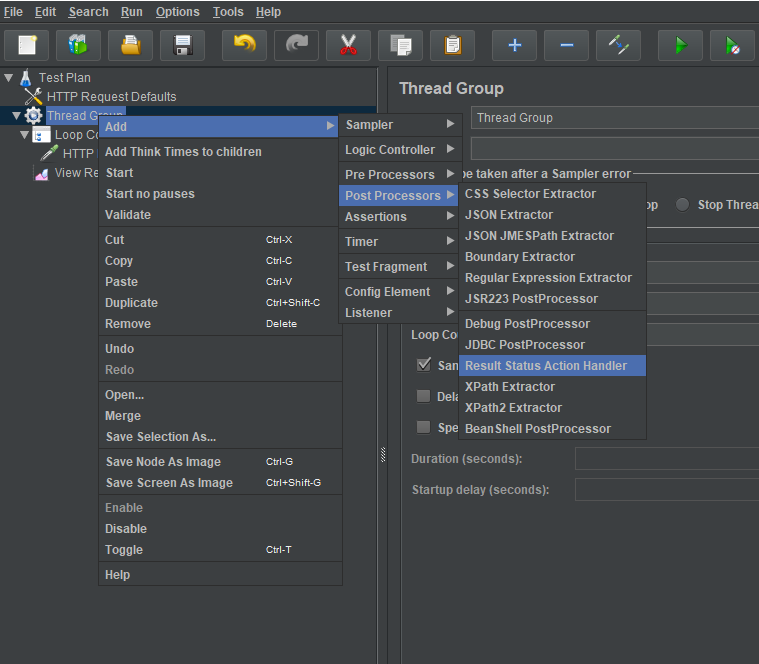
Hình 89: Roadmap of the example

* Bước 1: Add Thread Group
  + Chuột phải vào Test Plan 🡪 Add 🡪 Threads (Users) 🡪 Thread Group
  + Cấu hình Thread Group như sau:



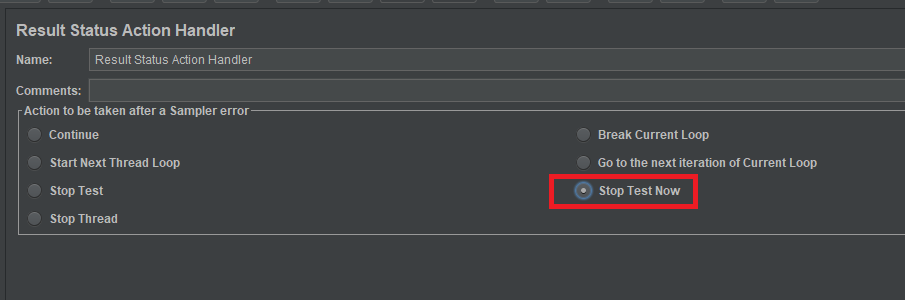
Hình 90: Configuring Thread Group

* + JMeter sẽ tạo 10 user request tới google và lặp lại 10 lần.
* Bước 2: Add JMeter elements
  + Thêm HTTP request Default, request đến google
  + Thêm HTTP request
* Bước 3: Add Post – Processor Element
  + Chuột phải Thread Group 🡪 Add 🡪 Post Processor 🡪 Result Status Action Handler
  + Result Status Action Handler cho phép người dùng dừng thread của test nếu user request thất bại.



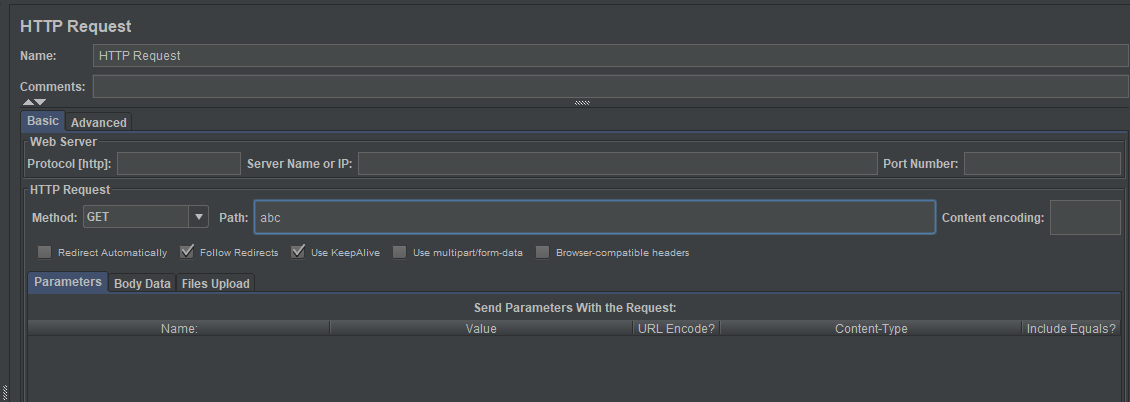
Hình 91: Adding Result Status Action Handler

* + Trong Result Status Action Handler Panelm chọn Stop Test Now. Lựa chọn này sẽ dừng test nếu JMeter nhận được phản hồi lỗi từ server.



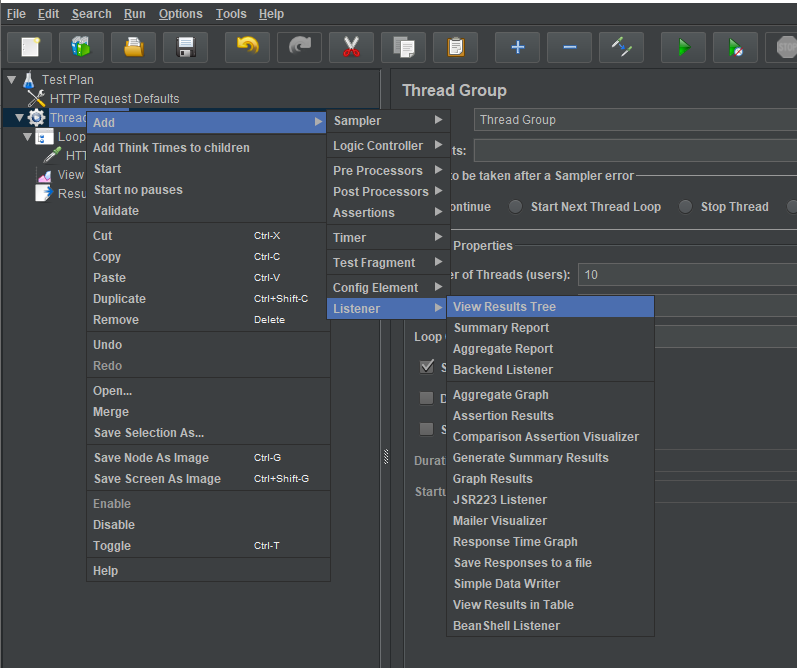
Hình 92: Setting Result Status Action Handler

* Bước 4: Config HTTP Request
  + Mở HTTP Request Panel, nhập giá trị tại path, ví dụ “abc”.
  + Khi ta nhập “abc” tại path, JMeter sẽ tạo một URL request tới Google server: <http://www.google.com/abc>. URL này không tồn tại trên Google server, đây là một URL sai nên server sẽ trả về lỗi.



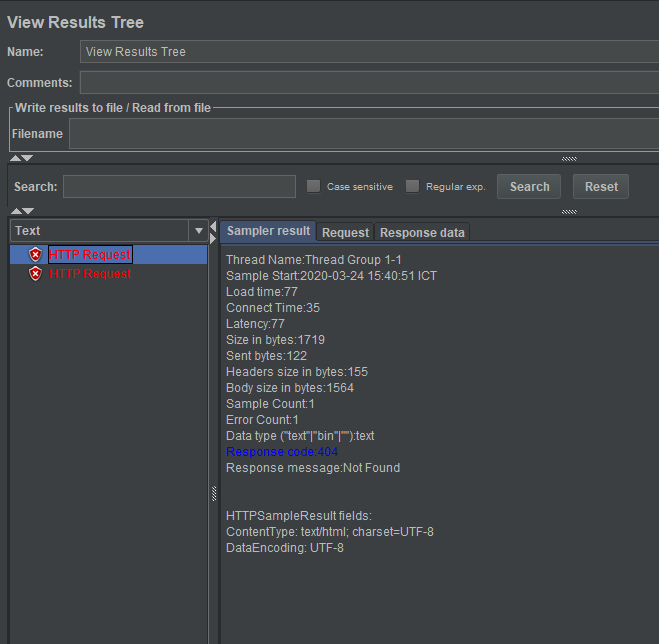
Hình 93: Configuring HTTP Request

* Bước 5: Add View Result Tree
  + Chuột phải vào Thread Group 🡪 Add 🡪 Listener 🡪 View Result Tree



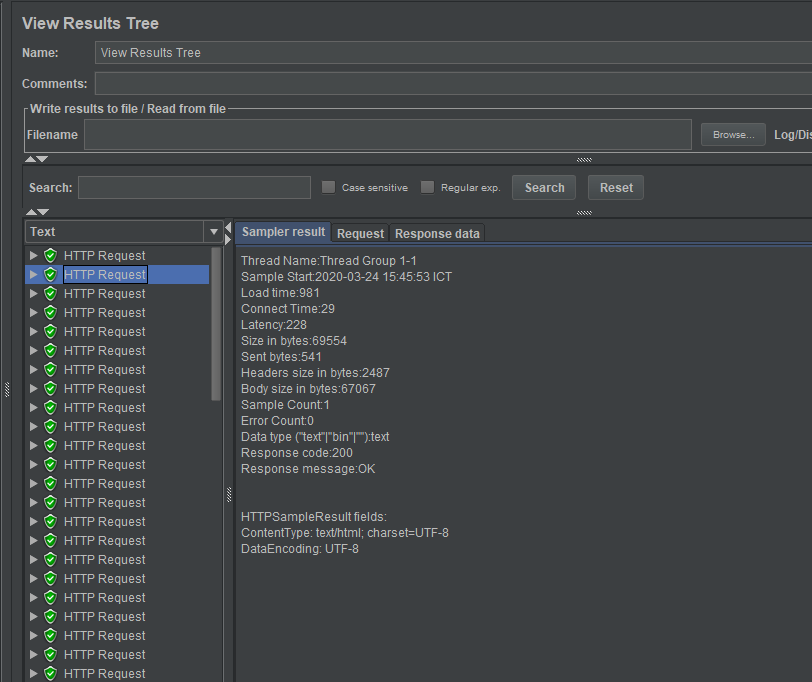
Hình 94: Add View Result Tree

* Bước 6: Run Test Plan
  + Chạy Test Plan và xem kết quả trong View Result Tree. Ta sẽ thấy lỗi phản hồi từ Google và test sẽ dừng lại mà không hoàn thành 100 thread.



Hình 95: Error Result Tree

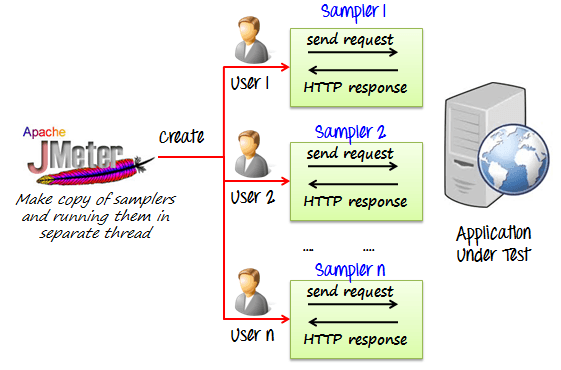
* + Nếu tại bước 4, giá trị tại path sửa thành “calendar”. JMeter sẽ tạo URL request <http://www.google.com/calendar> đến server google. Đây là một request đúng nên Google sẽ phản hồi OK. Khi đó kết quả được hiển thị ở View Results Tree:



Hình 96: Success Result Tree

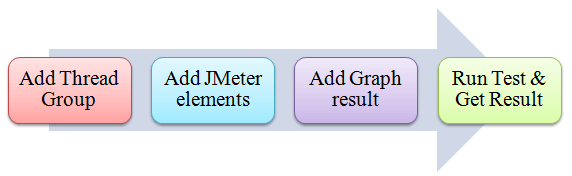
# **Performance Testing**

* Performance Testing là quy trình quan trong để đảm bảo Web Application khi test sẽ đáp ứng yêu cầu tải cao nhất. Nó có thể dùng để phân tích tổng quan hiệu năng của server dưới mức chịu tải cao.
* Apache JMeter cung cấp các lợi ích sau trong Performance Testing:
  + JMeter có thể test hiệu năng của cả nguồn tĩnh như JavaScript, HTML và nguồn động như JSP, Servlets, AJAX
  + Jmeter có thể giả lập tối đa số lượng người dùng tại một thời điểm mà trang web có thể xử lí
  + JMeter cung cấp nhiều kiểu báo cáo phân tích kết quả khác nhau (Report, Graph, Result Tree…)
* JMeter Performance Testing gồm:
  + Load Testing: Mô hình hóa kì vọng sử dụng bằng cách giả lập lượng user truy cập Web service đồng thời.
  + Stress Testing: Mỗi web server sẽ có một dung lượng tải tối đa. Khi lượng tải đạt gần đến giới hạn thì web server sẽ bắt đầu phản hồi chậm dần, có thể gây ra lỗi. Mục tiêu của Stress Testing là tìm ra giá trị tải tối đa mà một web server có thể đạt được.

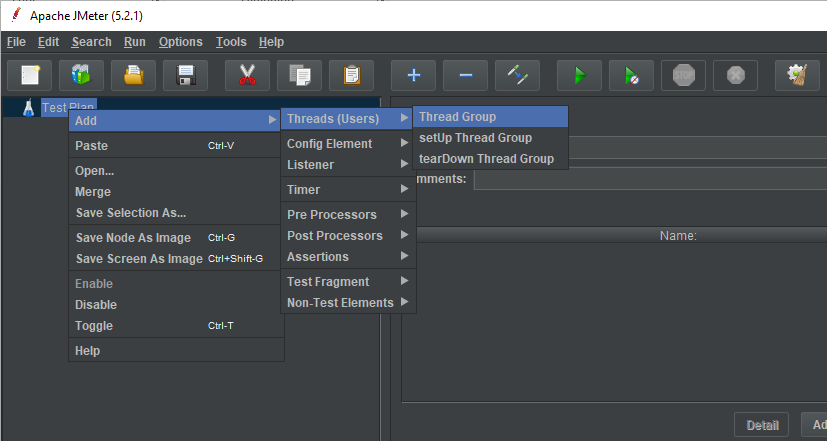


Hình 97: JMeter - Load Testing Performance Simulate

* Tạo một Performance Test Plan với JMeter
  + Ví dụ: Thực hiện một Performance Test mô phỏng khả năng chịu tải của goolgle.com khi có 1000 người truy cập đồng thời
    - Road map:

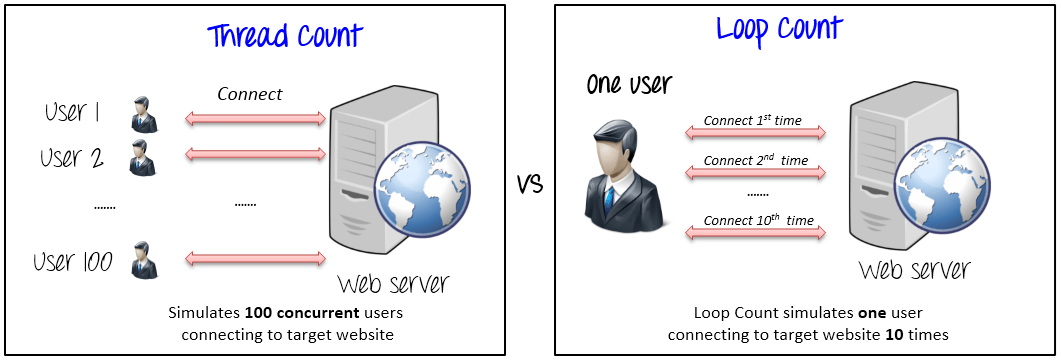


* + Bước 1: Thêm Thread Group bằng cách chuột phải vào Test Plan 🡪 Add 🡪 Thread Group



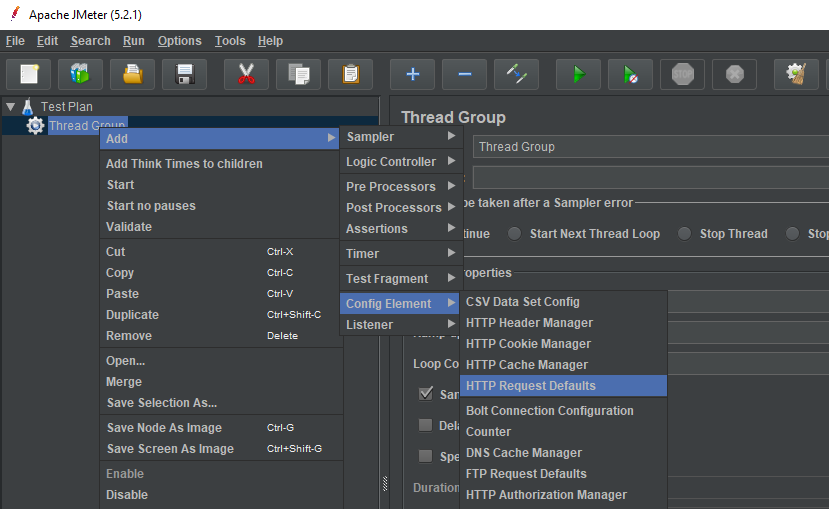
Hình 98: Add Thread Group for Test Plan

* + - Trong Thread Group, cài đặt các thông số Thread Properties như sau:
      * Number of Threads: 100 (số lượng user kết nối tới trang web)
      * Loop Count: 10 (số lần thực hiện test)
      * Ramp – Up Period: 100s (thời gian thực hiện Performance Test)
    - Ý nghĩa của Threads Count và Loop Count được thể hiện ở hình dưới đây:



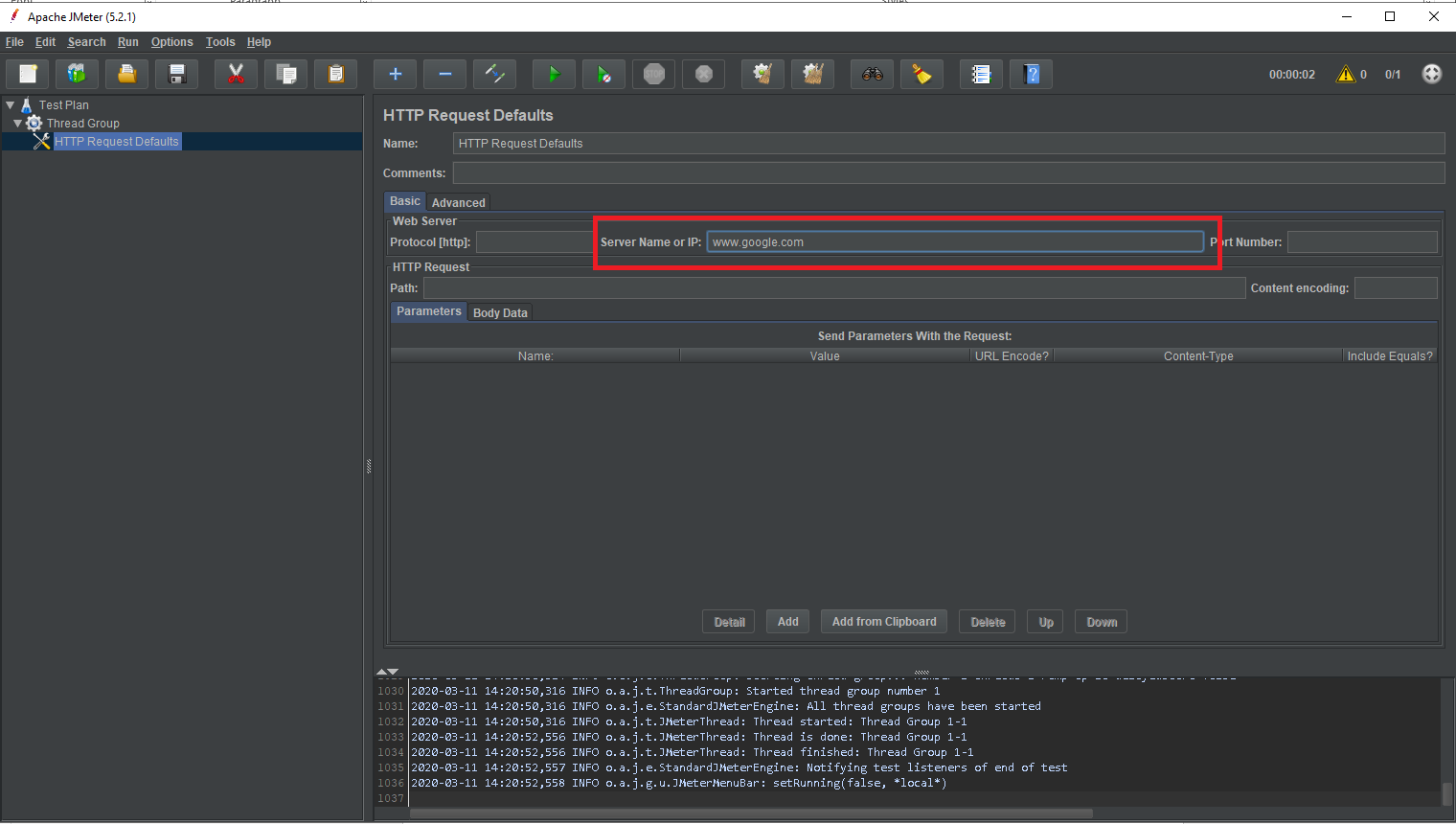
Hình 99: Difference between Thread Count and Loop Count

* + - Ramp – up Period: khoảng thời gian trì hoãn trước khi bắt đầu user tiếp theo. Ví dụ, nếu có 100 user và Ramp – up Period là 100s thì thời gian trì hoãn giữa mỗi user sẽ là 1s (100s / 100 users)
  + Bước 2: Thêm JMeter elements
    - HTTP Request Default: element này có thể được thêm bằng cách nhấp chuột phải vào Thread Group, chọn Add 🡪 Config Element 🡪 HTTP Request Defaults



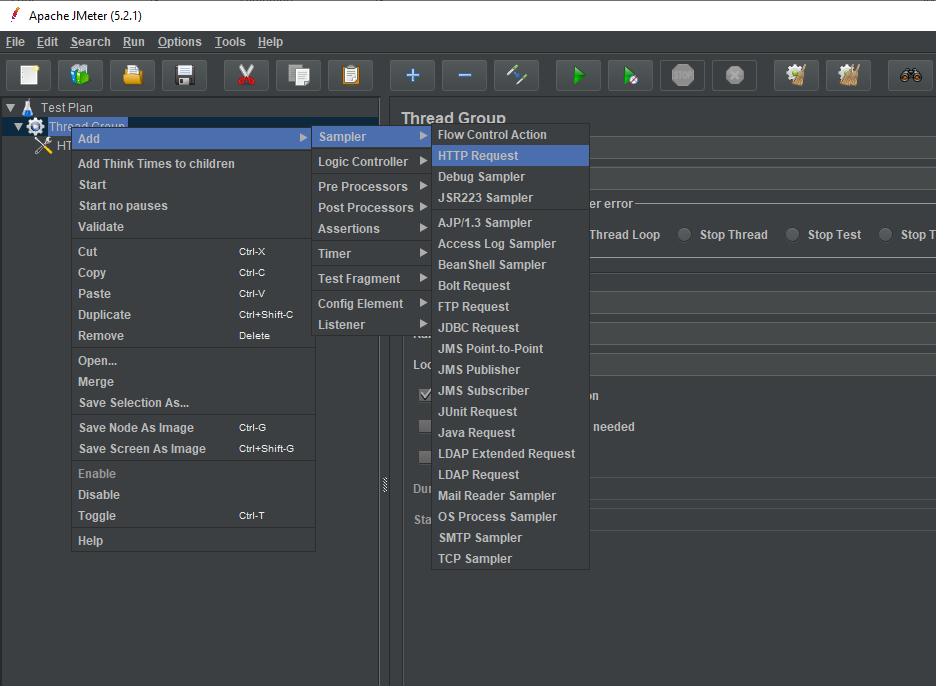
Hình 100: Adding HTTP Request Defaults

* + - Trong HTTP Request Defaults, nhập địa chỉ google.com tại mục Server Name or IP:



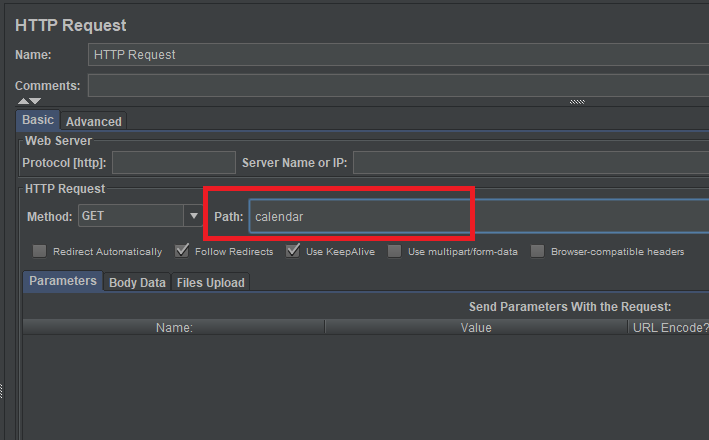
Hình 101: Setting HTTP Request Defaults

* + - HTTP Request: Thread Group 🡪 Add 🡪 Sampler 🡪 HTTP Request



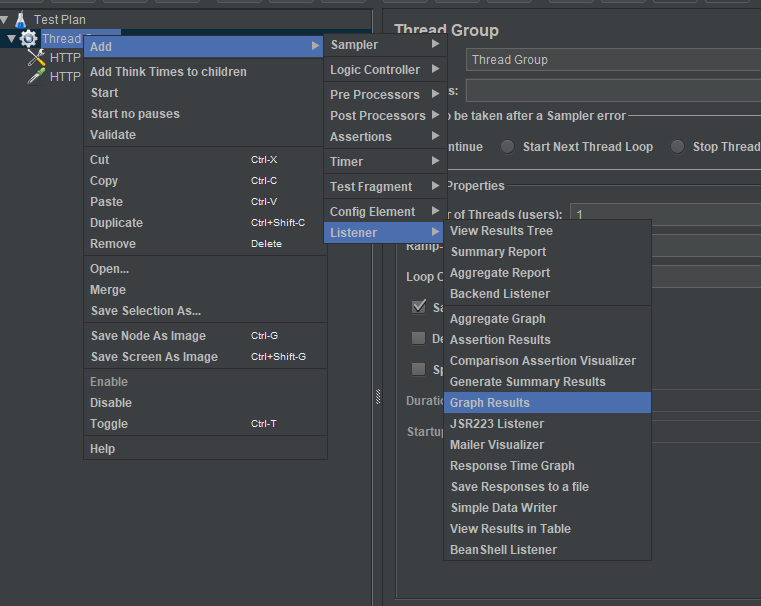
Hình 102: Adding HTTP Request

* + - Tại HTTP Request Control Panel, mục Path chứa URL request mà ta muốn gửi tới Google server. Ví dụ, tại đây nhập “calendar”, JMeter sẽ tạo một URL request <http://www.google.com/calendar> tới Google Server.



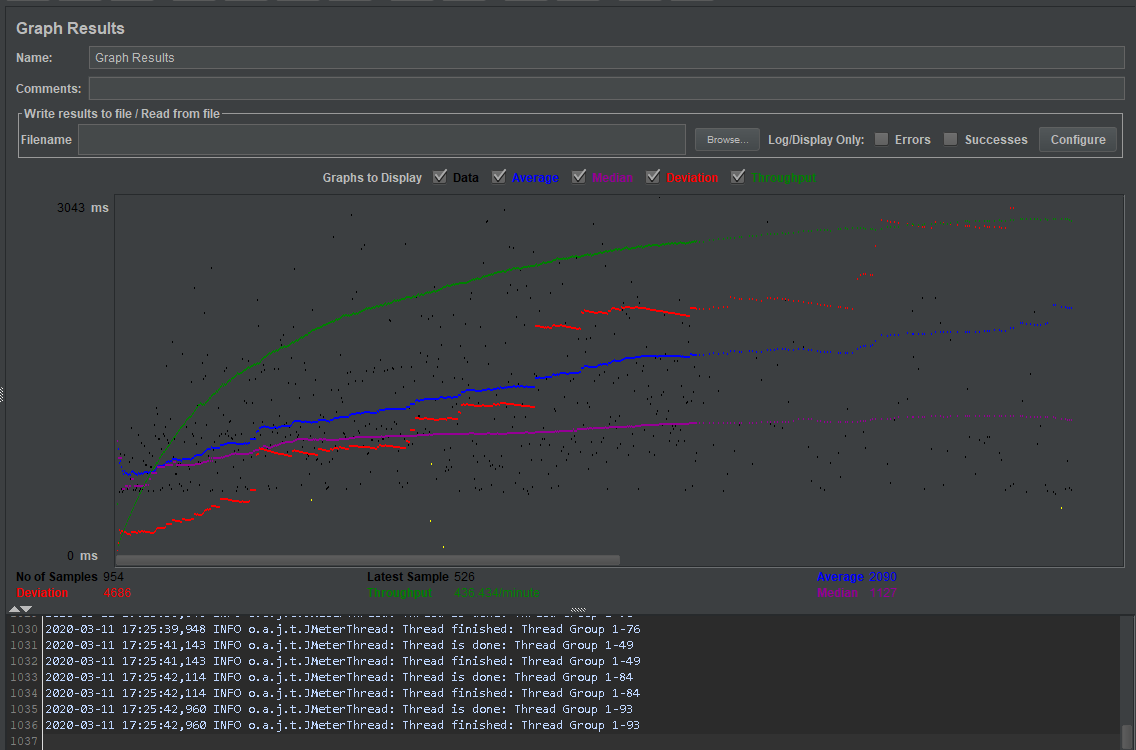
Hình 103: Sending URL Request to Google Server

* + - Nếu mục Path để trống thì JMeter sẽ tự điều hướng một URL request <http://www.google.com> tới Google Server
  + Bước 3: Thêm báo cáo kết quả JMeter
    - JMeter cung cấp nhiều loại báo cáo kết quả khác nhau, có thể ở dạng biểu đồ, bảng,… Ở ví dụ này ta chọn hiển thị kết quả ở dạng biểu đồ. Chuột phải vào Test Plan 🡪 Add 🡪 Listener 🡪 Graph Results



Hình 104: Adding Graph Results

* + Bước 4: Chạy Test Plan và xem kết quả: Chọn Run trên thanh công cụ hoặc nhấn Ctrl + R. Sau đó chọn Element Graph Results để xem kết quả



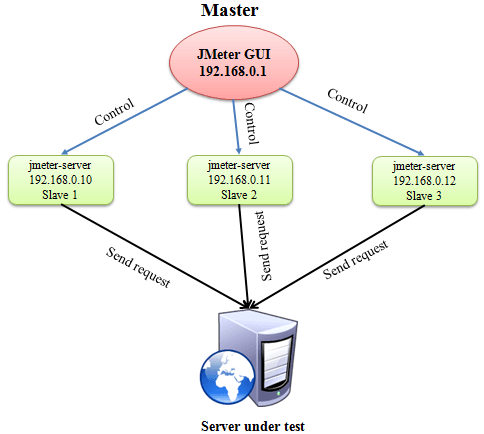
Hình 105: Graph Results

* + - Black: Tổng số samples gửi đi hiện tại
    - Blue: Trung bình hiện tại của tất cả các mẫu được gửi đi
    - Red: Độ sai lệch chuẩn hiện tại
    - Green: Tốc độ thông lượng thể hiện số lượng yêu cầu mà server xử lí
    - Để phân tích hiệu năng của web server khi test, ta quan tâm tới 2 giá trị: Throughput (điểm green) và Deviation (điểm đỏ).
    - Throughput là giá trị quan trong nhất, nó thể hiện khả năng chịu tải lớn của server. Giá trị này càng cao, hiệu năng của server càng lớn. Ở ví dụ trên, giá trị throughput là 436.434/minute, nghĩa là Google server có thể xử lí được 436.434 yêu cầu mỗi phút. Thực tế giá trị này còn phụ thuộc nhiều vào CPU, tốc độ mạng,... nên kết quả thu được sẽ chỉ mang tính chất tương đối.

# **JMeter Distributed (Remote) Testing: Master Slave Configuration**

## **Distributed Testing**

* Distributed Testing là một loại Test sử dụng nhiều hệ thống để kiểm tra hiệu năng Stress Testing. Distributed Testing được áp dụng cho test website và server khi chúng đang làm việc với nhiều client đồng thời.
* Distributes Testing sử dụng mô hình client – server như hình dưới đây:

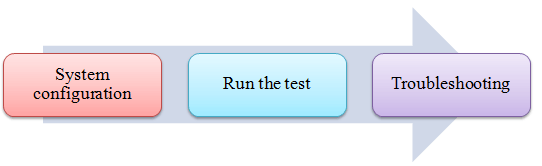


Hình 106: Distributed Testing using model client – server

* + Master: Hệ thống chạy ở JMeter GUI, điều khiển các slave
  + Slave: Hệ thống chạy ở JMeter Server, nhận điều khiển từ master và gửi yêu cầu tới server
  + Target: server đang test, nhận request từ các slave.

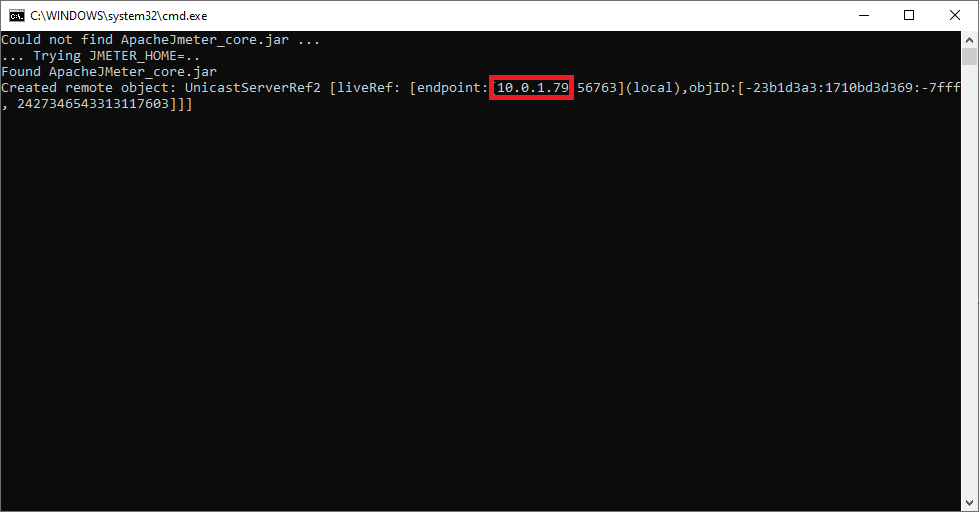
## **Remote Test Example**

* Điều kiện:
  + Tắt Firewall, trong một số trường hợp, firewall sẽ chặn đường truyền.
  + Tất cả các máy phải cùng một subnet. Nếu các máy tính không cùng một subnet, có thể chúng sẽ không kết nối được với nhau trong cùng một mạng.
  + Sử dụng cùng một version của JMeter để tránh các lỗi xảy ra.

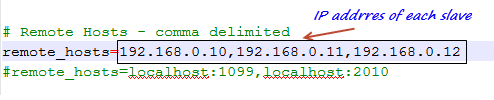


Hình 107: Road map of Remote Test

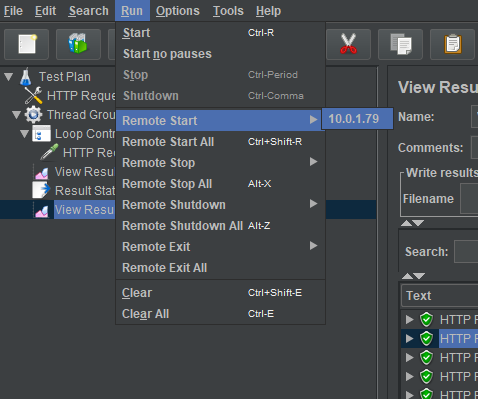
* Bước 1: System Configuration
  + Cài đặt Môi trường cho JMeter (xem ở mục 3.2.3)
  + Cài đặt hệ thống slave, đi đến thư mục jmeter/bin và chạy file “jmeter-server.bat”.
  + Máy Slave sẽ có IP trong phần endpoint:[…].



* + Trên hệ thống Master, đi đến thư mục /bin và sửa file jmeter.properties, tìm đến dòng remote\_hosts và thêm IP của máy slave như sau:



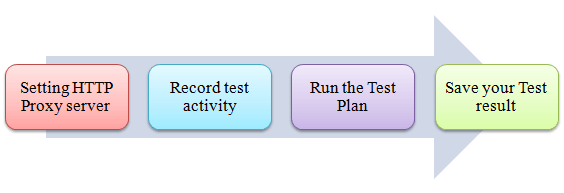
* Bước 2: Run Test
  + Đến thời điểm này, ta đã có thể bắt đầu Distributed Test qua JMeter. Trên máy Master, chạy JMeter GUI và mở test plan. Chon Run 🡪 Remote Start 🡪 Chon IP của máy slave.



Hình 108: Select a Slave to run test

# **HTTP Proxy Server In JMeter: Record Example Script**

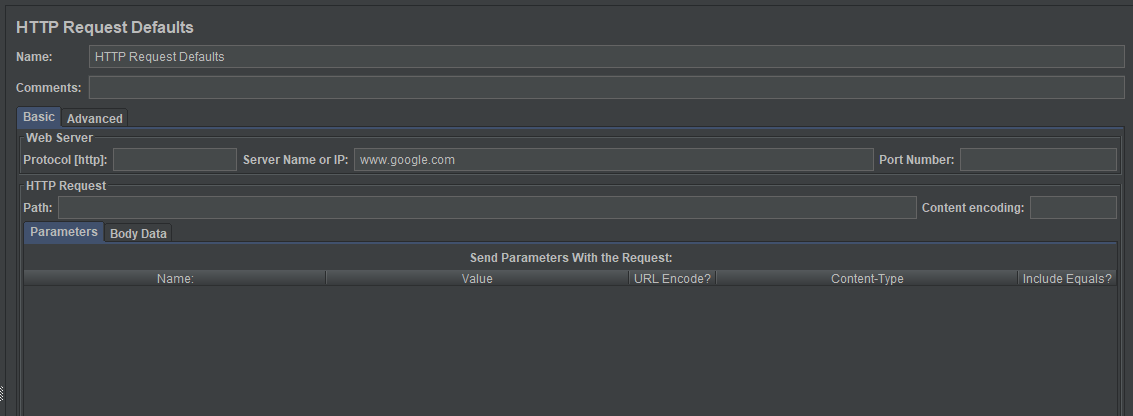
* Record Testing giúp tester record và run các hành động của họ trên đối tượng test. Nó là một loại của test tự động nhưng cho nhiều user. Hướng dẫn này sẽ chỉ cách sử dụng Proxy Server để thực hiện record test.
* Proxy cho phép JMeter xem và record lại các hành động của tester khi họ đang sử dụng web trên trình duyệt thông thường.



Hình 109: Road map of Record

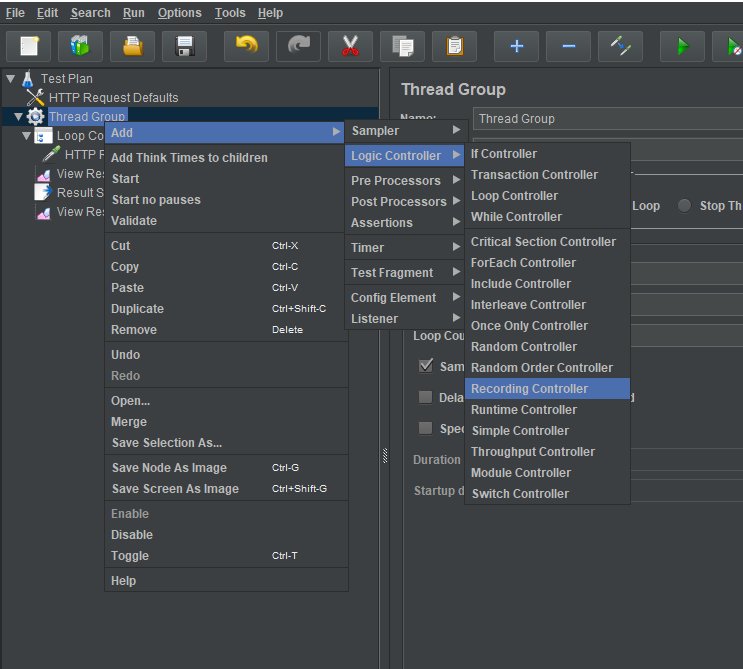
## **Setting HTTP Proxy server**

1. Tạo Thread Group trong Test plan: Test Plan 🡪 Add 🡪 Thread Users 🡪 Thread Group
2. Thêm HTTP Request Defaults: Thread Group 🡪 Add 🡪 Config Element 🡪 HTTP Request Defaults. Cài đặt HTTP Request Defaults như sau:



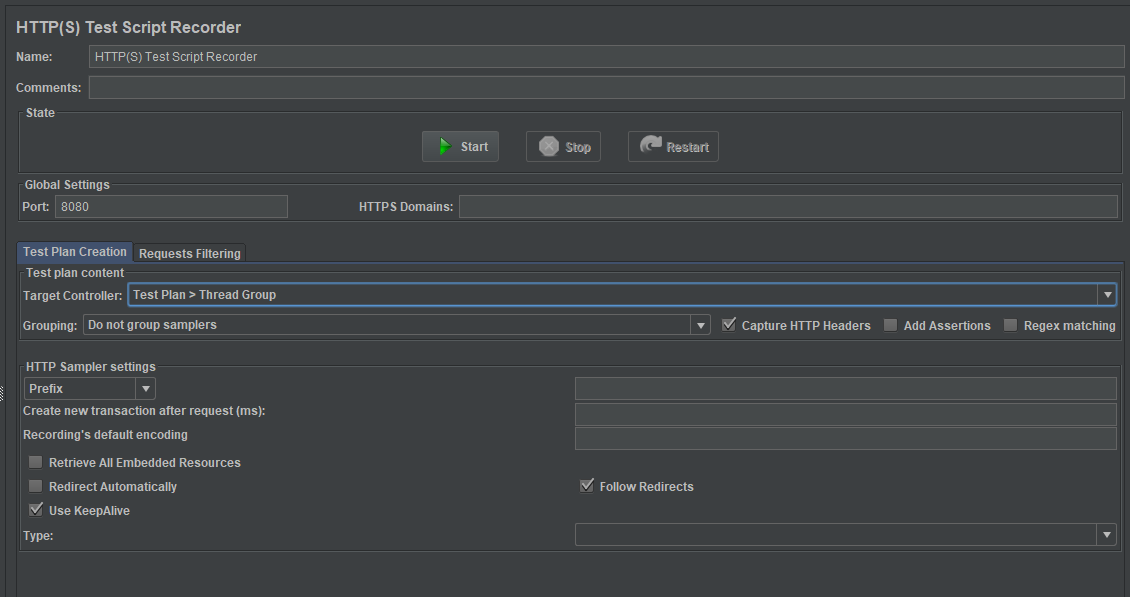
Hình 110: Setting HTTP Request Defaults

1. Thêm Recording Controller: Thread Group 🡪 Add 🡪 Logic Controller 🡪 Recording Controller



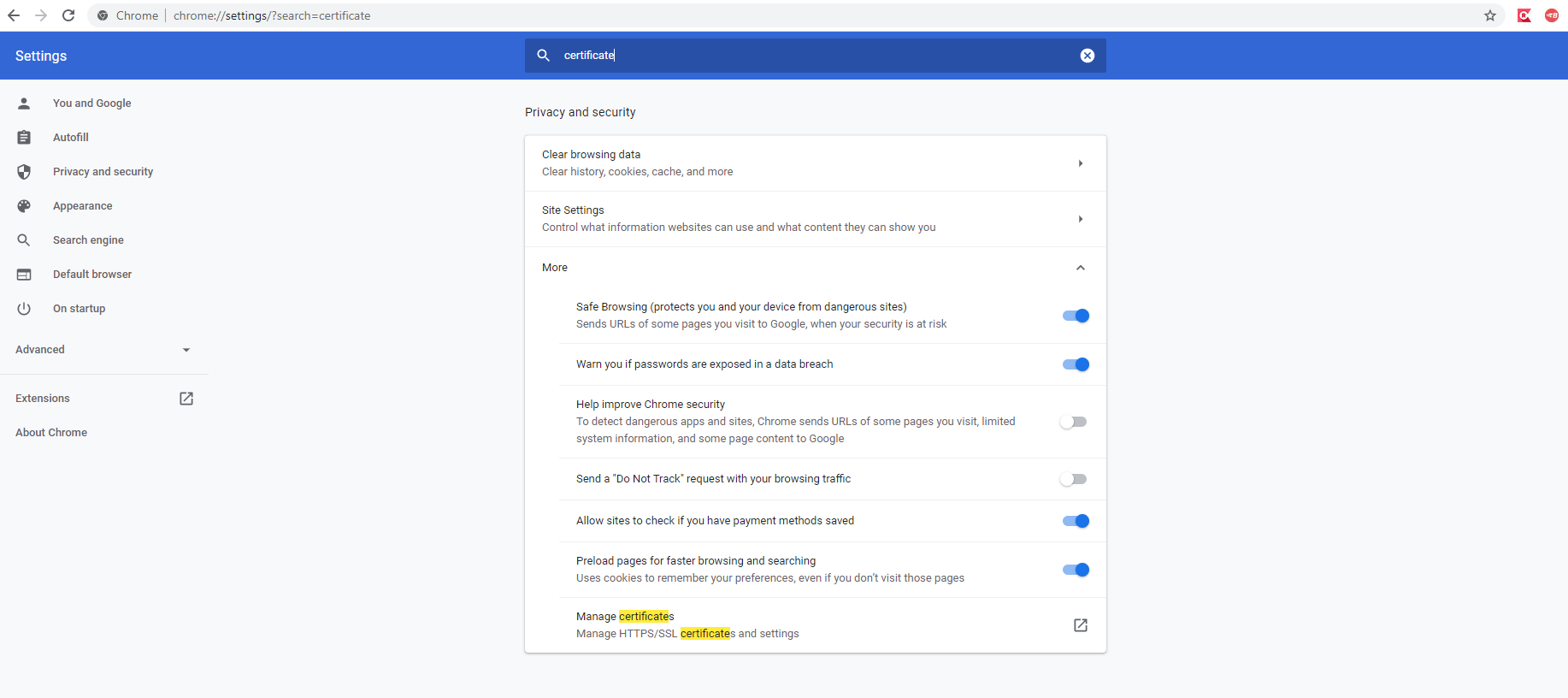
Hình 111: Adding Recording Controller

1. Thêm Proxy Server: Test Plan 🡪 Add 🡪 Non – Test Elements 🡪 HTTP(S) Test Script Recorder
2. Trong HTTP(S) Test Script Recorder, chọn [Test Plan > Thread Group] trong mục Target Controller, cổng 8080

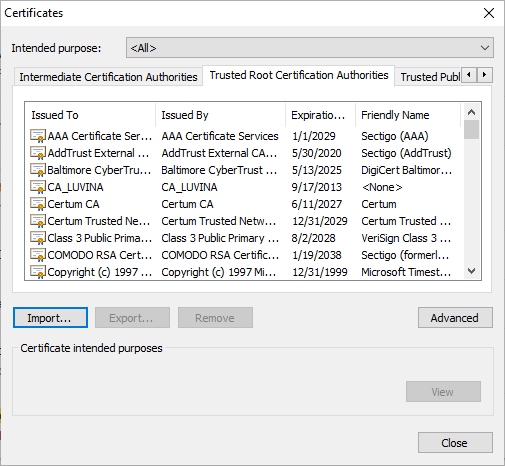


Hình 112: Setting HTTP(s) Test Script Recorder

1. Cấu hình proxy (đây là bước quan trọng để có thể thực hiện record tự động):
   * Đối với CHROME:
     + Trong thư mục /bin của JMeter có file ApacheJMeterTemporaryRootCA, đây là file chứng chỉ quan trọng cần được thêm vào Certificate của Google để có thể record được trên google. Install file này.
       - Mở file , kiểm tra thời gian Valid From, nếu quá hạn thì cần Install lại, click Install Certificate và làm theo hướng dẫn.
2. Mở Google Chrome 🡪 Setting. Tìm kiếm “certificate” trong ô nhập search. Chọn Manage Certificates.

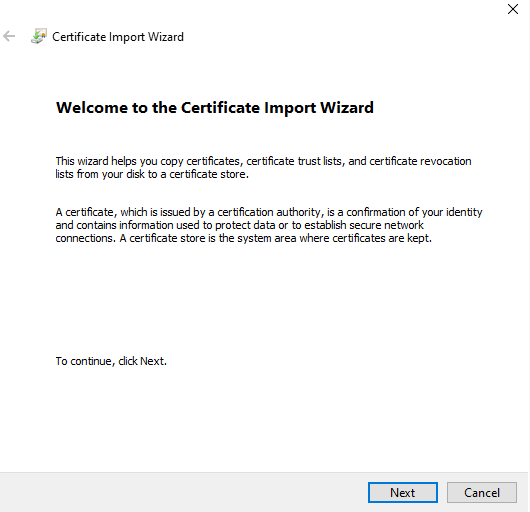


Hình 113: Searching Setting Certificate

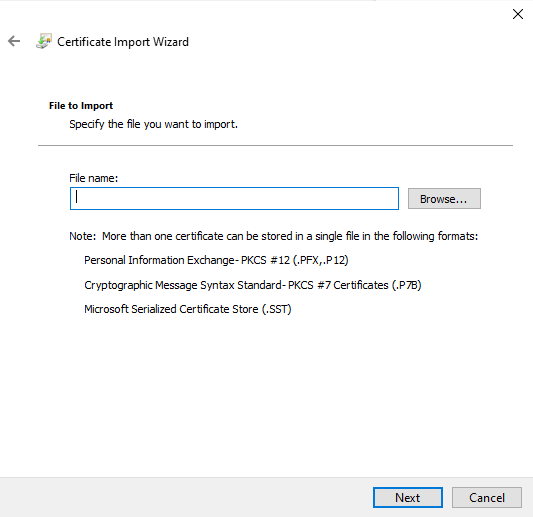


Hình 114: Certificate Panel

1. Tại Tab Trusted Root Certification Authorities, Import ApacheJMeterTemporaryRootCA vừa Install bằng cách chọn Import.

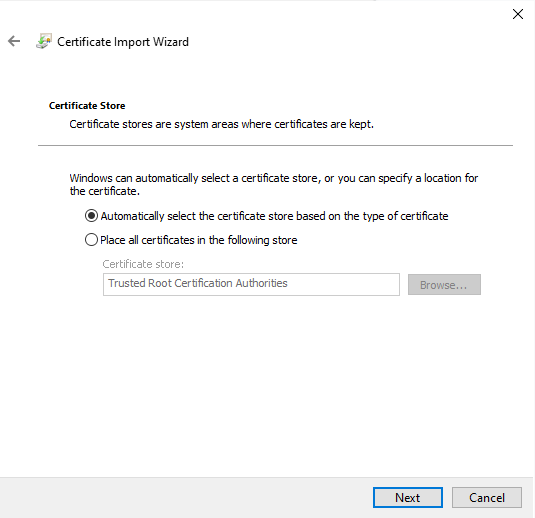


Hình 115: Import Certification

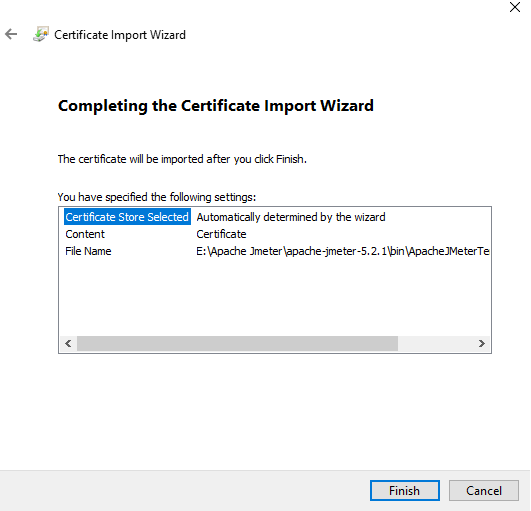


Hình 116: Import Certification

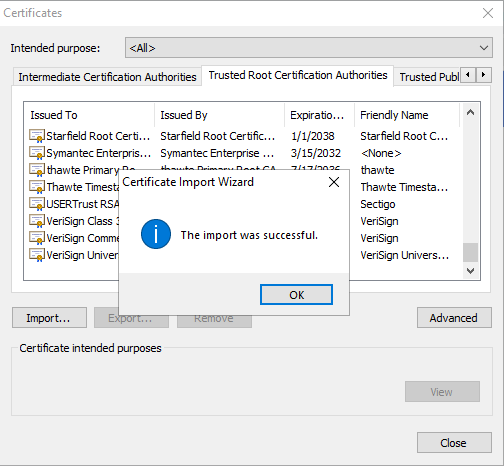
* + - Browse đến file ApacheJMeterTemporaryRootCA của thư mục bin trong JMeter. Click Next.



Hình 117: Import Certificate

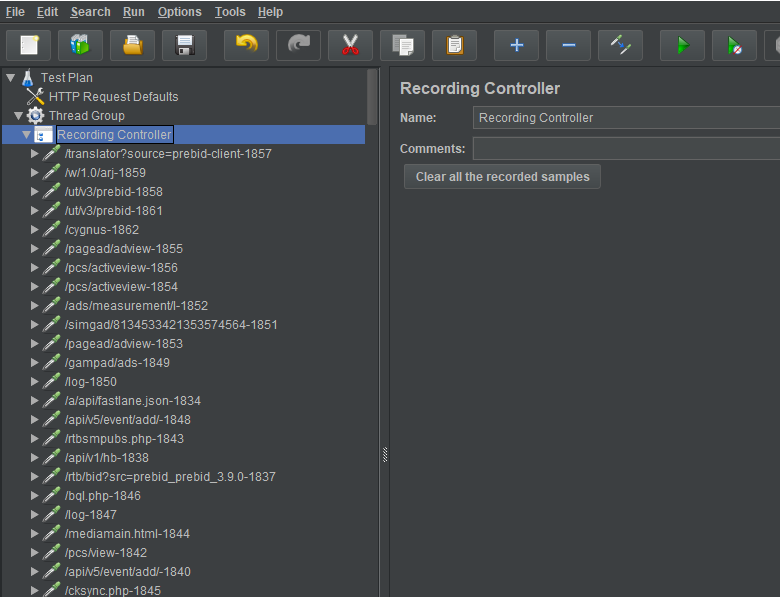


Hình 118: Import Certification



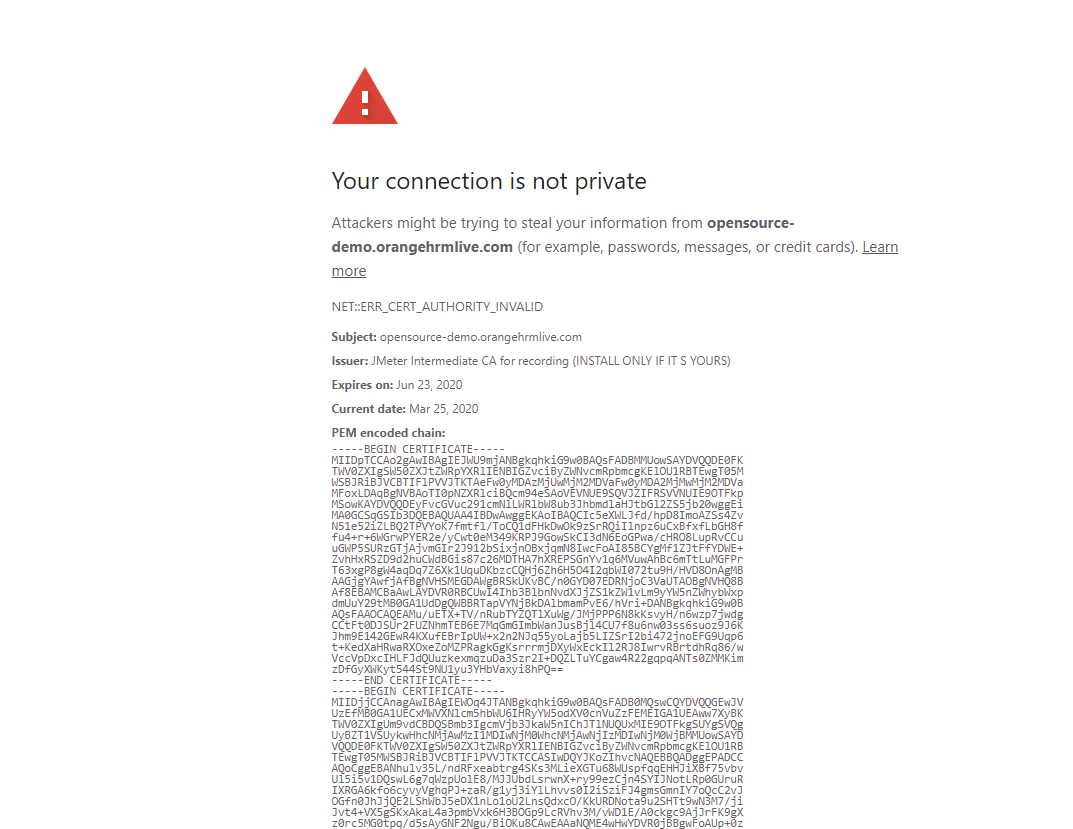
Hình 119: Import Certification successful

1. Sau khi đã Import Certification của JMeter, ta cài đặt proxy trong window. Truy cập Network & Internet trong Setting, chọn Proxy. Bật User a proxy server, nhập Address là <http://localhost>, port là 8888 và Save.
2. Vậy là ta đã có thể record JMeter thông qua proxy server đối với google. Tại HTTP(s) Test Script Recorder Panel, click Start để bắt đầu record và mở trình duyệt Chrome lên, thực hiện các thao tác record. Sau khi thao tác xong, chọn Stop để dừng việc record. Các thao tác record được lưu trong Recording Controller



Hình 120: Recording Controller

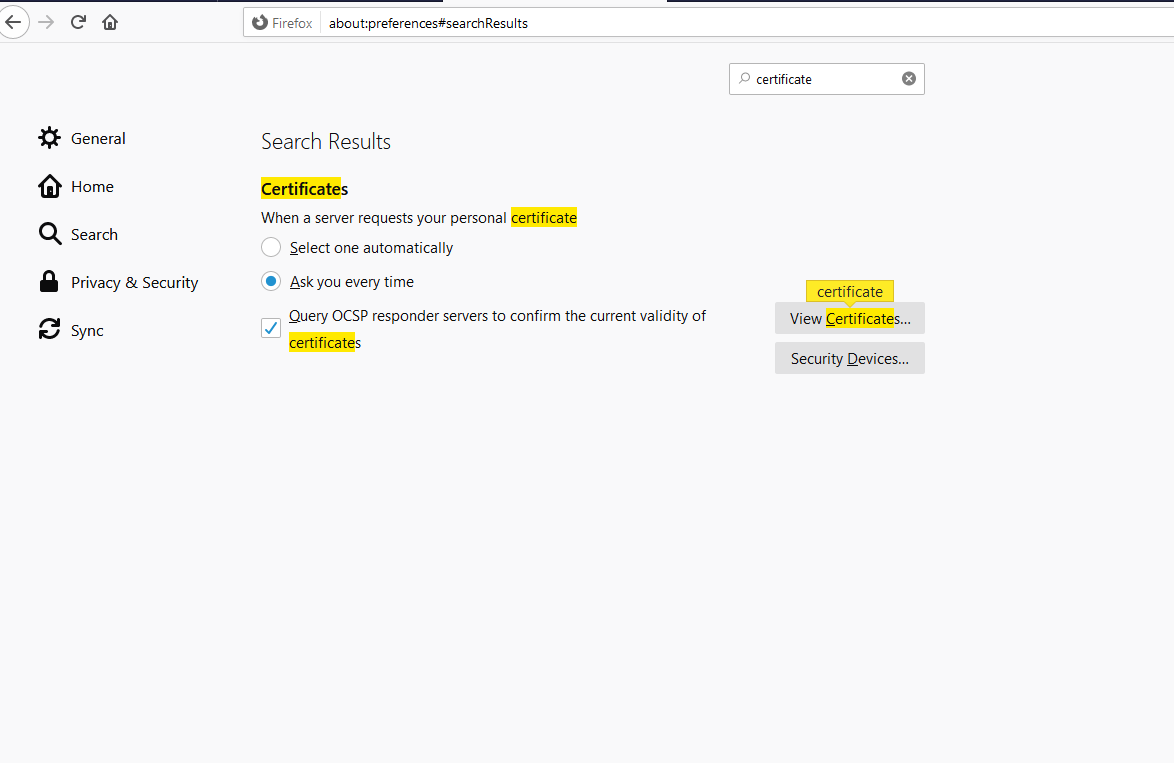
* + - Trường hợp trình duyệt bình thường trên Chrome không chạy được record, ta hãy dùng trình duyệt bảo mật để chạy.



Hình 121: ERROR when record using Chrome

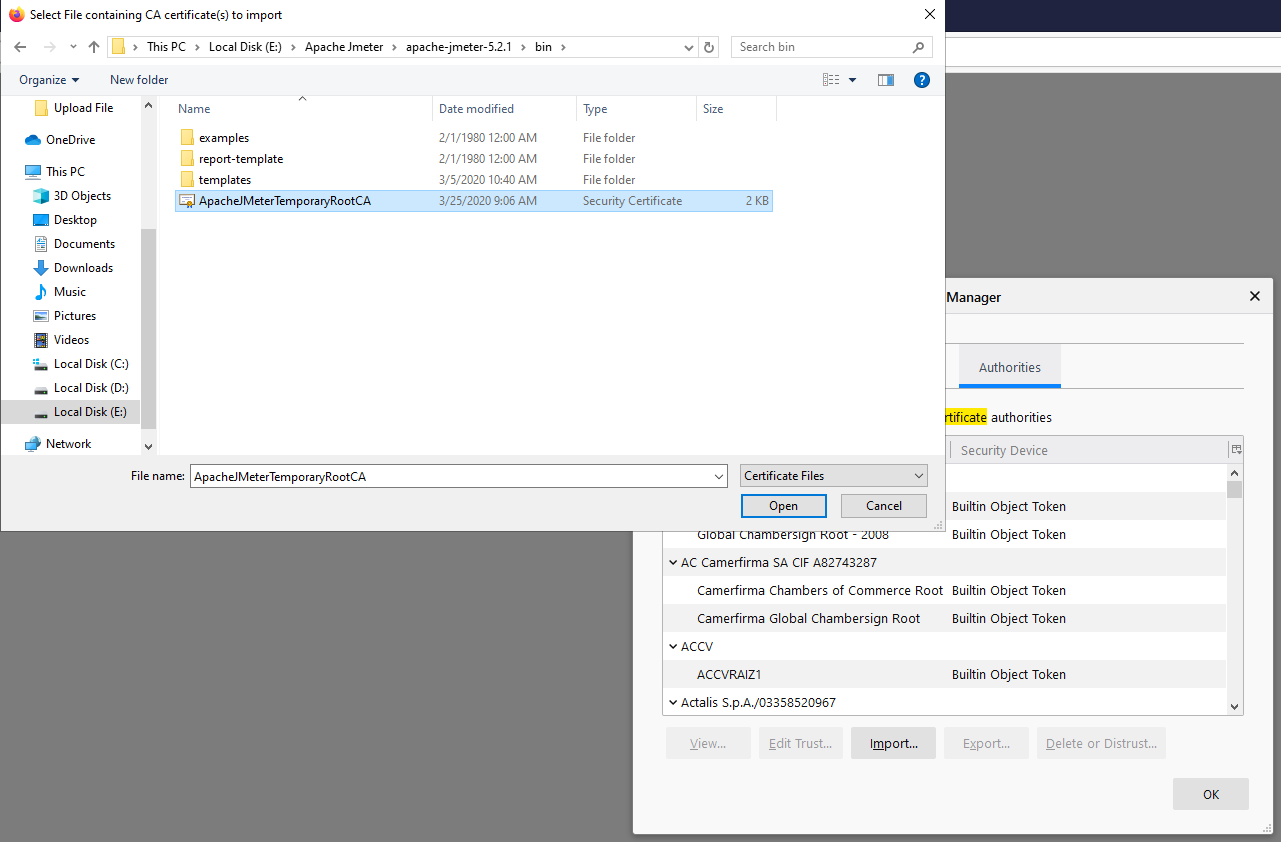
* + Đối với FIREFOX:
    - Tương tự như Chrome, ta cần Install ApacheJMeterTemporaryRootCA trước, sau đó Import Certificate vào Firefox.
    - Để Import vào Firefox, ta thực hiện:

1. Mở trình duyệt Firefox.
2. Truy cập Option, tìm kiếm cho từ khóa “certificate”. Chọn View Certificates.



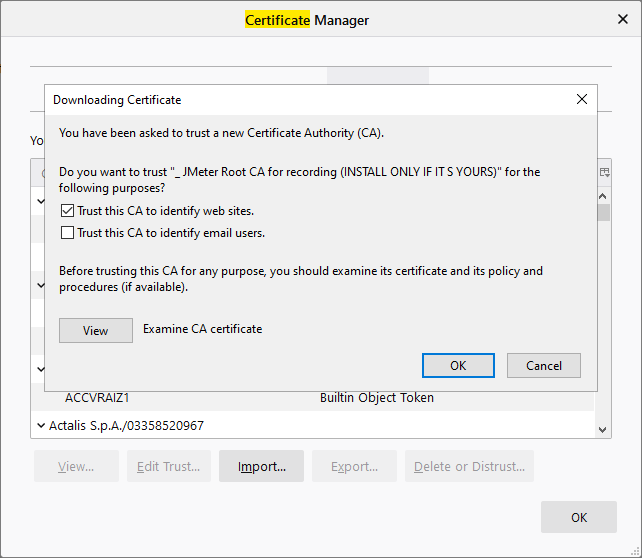
Hình 122: View Certificates

1. Chọn tab Authorities, Import, trỏ đường dẫn tới file certificate của JMeter trong thư mục bin.



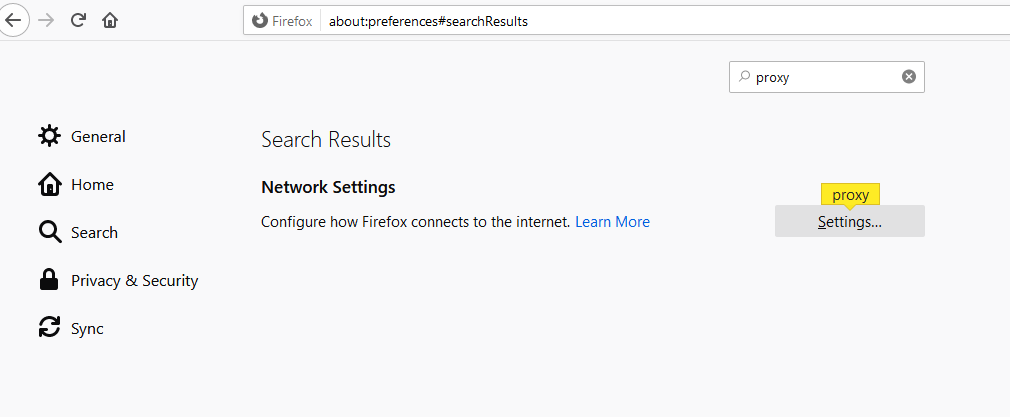
Hình 123: Import Certification

1. Chọn Trust this CA to identify web sites, click OK, certification của JMeter đã được import thành công.



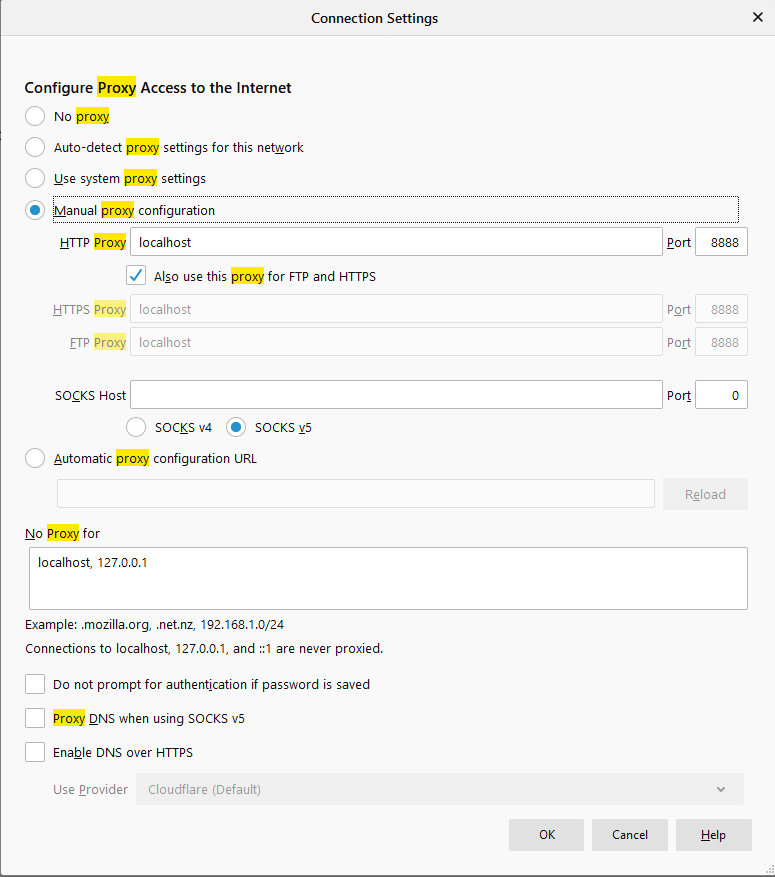
Hình 124: Import Certification

1. Sau khi Import thành công, ta cài đặt proxy. Truy cập option của trình duyệt, tìm kiếm từ khóa “proxy”, chọn Settings khi tìm thấy mục Network Settings.



Hình 125: Searching Network Settings

1. Tiến hành config proxy theo như hình dưới đây, click OK sau khi hoàn thành:



Hình 126: Config Proxy

* + Vậy là ta đã có thể thực hiện được record trên Firefox. Trong trường hợp trình duyệt báo lỗi thì có thể ta đã bỏ qua bước Setting proxy, hoặc Certification bị hết hạn,… Hãy lặp lại các bước như trên để kiểm tra.

# **Recording Web Application with BlazeMeter**

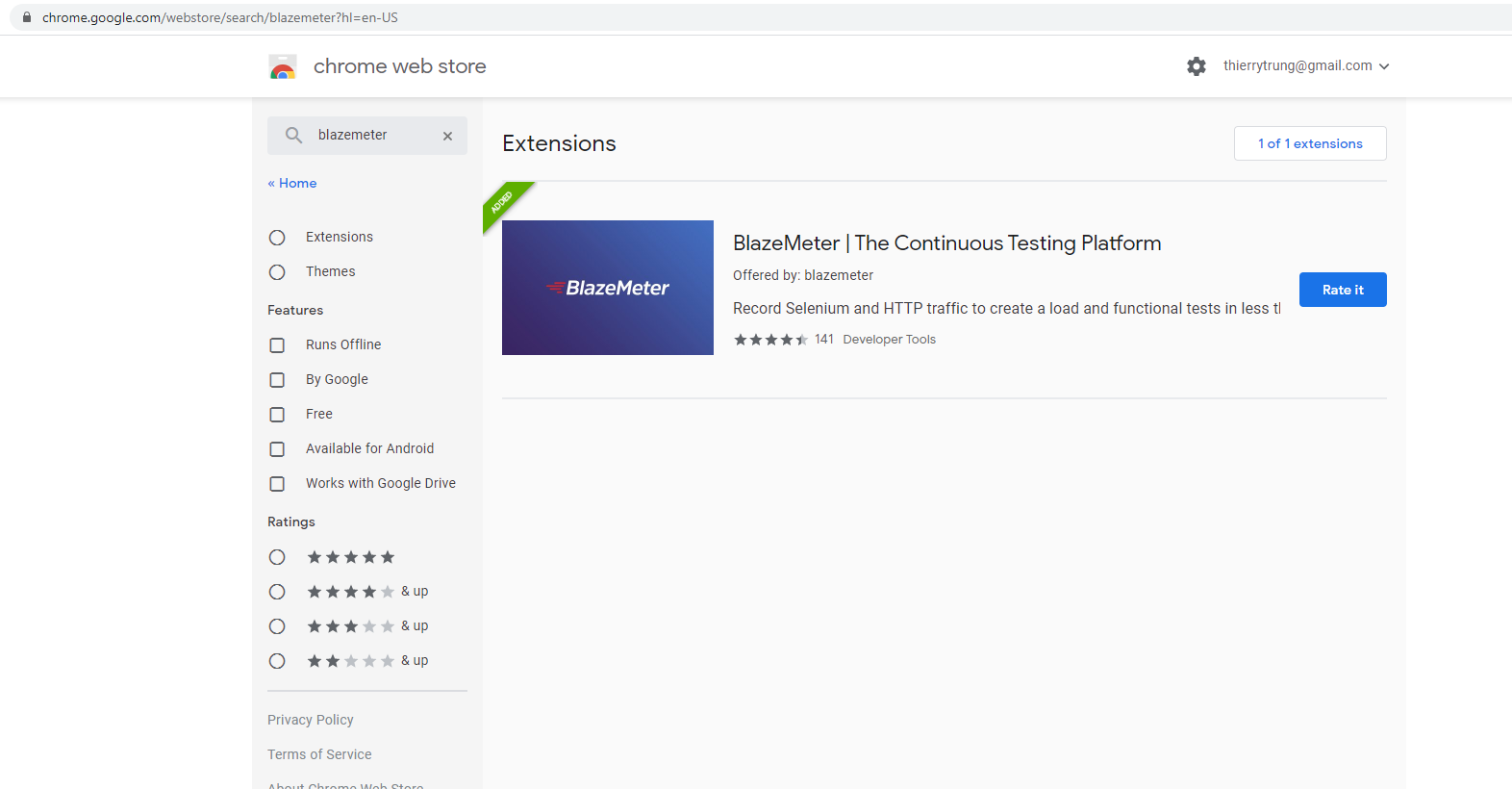
## **BlazeMeter**

* BlazeMeter là một công cụ để thực hiện test performance Công cụ này có khả năng mở rộng cao đến một triệu người dùng ảo.
* Nó có báo cáo dựa trên web toàn diện có thể được truy cập từ mọi nơi.
* BlazeMeter cung cấp plug-in Relic mới giúp theo dõi mắt trên máy chủ. Đối với tài khoản Relic mới (giấy phép) là bắt buộc.
* BlazeMeter sử dụng các máy chủ đám mây của Amazon và tất cả việc thực hiện kiểm tra được thực hiện từ đám mây.
* BlazeMeter cũng cung cấp hỗ trợ trình cắm thêm cho Google Analytics, Apache JMeter và Drupal.
* BlazeMeter có thể dễ dàng ghi lại (record) các hành động khi thực hiện test một cách nhanh chóng và xuất ra file jmx, hỗ trợ chỉnh sửa và chạy trên JMeter.

## **Recording Web Application with BlazeMeter**

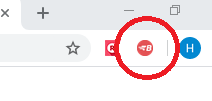
* Như đã nói ở trên, BlazeMeter có thể dễ dàng ghi lại (record) các hành động khi thực hiện test một cách nhanh chóng và xuất ra file jmx, hỗ trợ chỉnh sửa và chạy trên JMeter.
* Đây là cách record web đơn giản, không phức tạp như thiết lập proxy ở mục 11, record cũng chính xác. Các bước thực hiện:

1. Cài đặt tiện ích BlazeMeter (trình duyệt Chrome):
   * Tìm kiếm từ khóa “blazemeter”, đây chính là tiện ích cần cài:



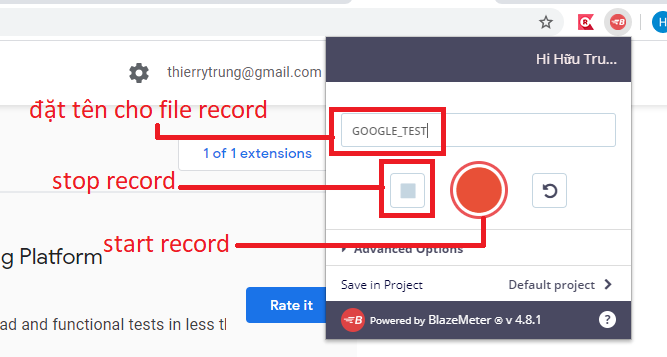
Hình 127: Installing BlazeMeter

1. Sau khi cài đặt thành công vào trình duyệt, ta sẽ thấy icon của tiện ích trên cửa sổ Chrome



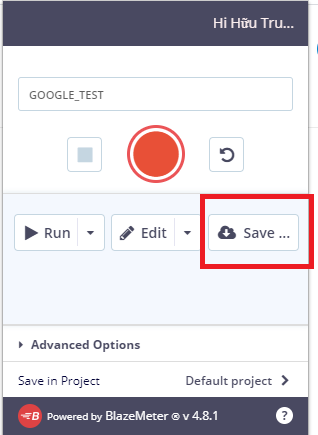
Hình 128: BlazeMeter Icon

1. Để tiến hành record, ta truy cập tới trang web cần test, ví dụ ta cần test thời gian tìm kiếm có nội dung “internet” trên google. Sau đó click icon BlazeMeter, đặt tên và click record.

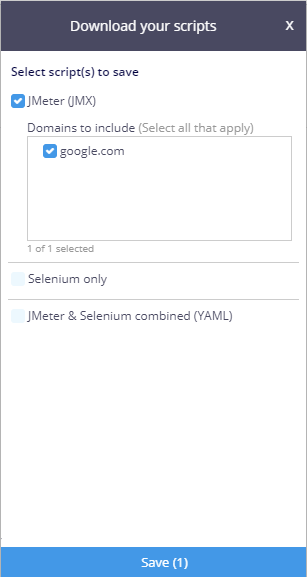


Hình 129: Quick Intruction with BlazeMeter

1. Thực hiện các thao tác trên google, nhập ô tìm kiếm, sau đó click stop ở Blazemeter. Khi đó ta sẽ kết thức việc record. Tải file vừa record dưới dạng jmx để có thể load ở trong JMeter.

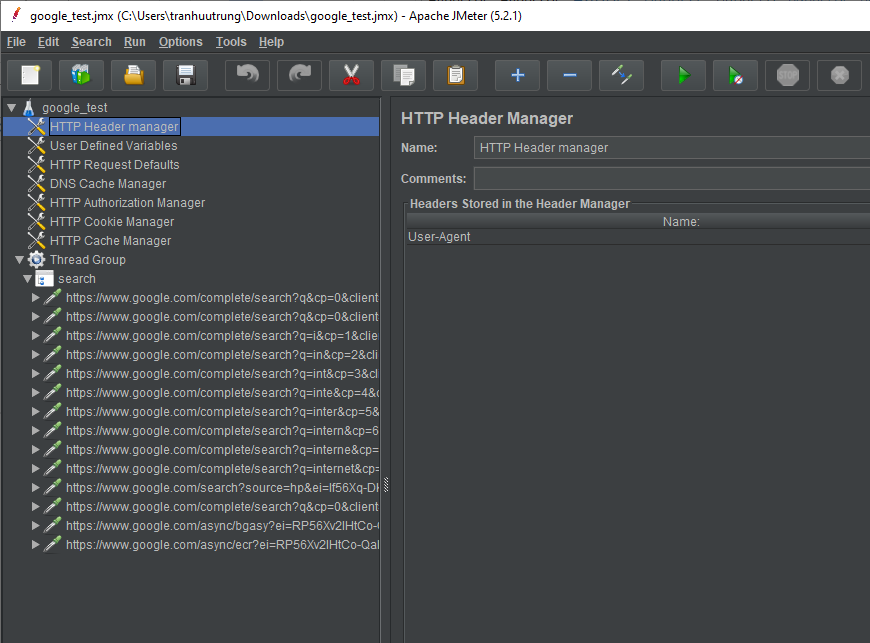


Hình 130: Save the test



Hình 131: Select JMeter (JMX)

1. Sau khi save và tải xuống, ta mở JMeter và load Test Plan vừa tải.



Hình 132: Test Plan is loaded into JMeter

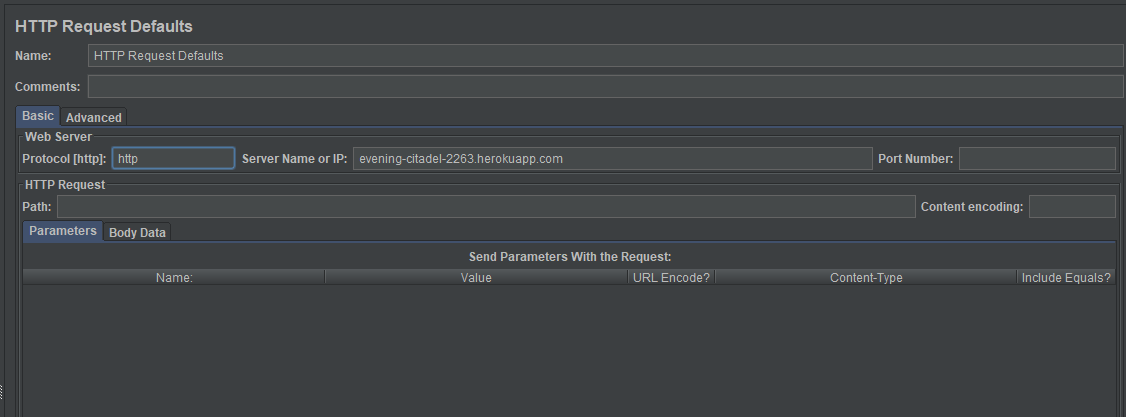
# **JMeter Config Elements – Variables**

* Ở mục này, ta sẽ tiến hành thực hiện các ví dụ đơn giản để hiểu được cách sử dụng của variables, đọc dữ liệu từ file CSV, đọc dữ liệu từ JDBC.
* Ví dụ login một trang web đơn giản. (<http://evening-citadel-2263.herokuapp.com/>)

## **User Defined Variables**

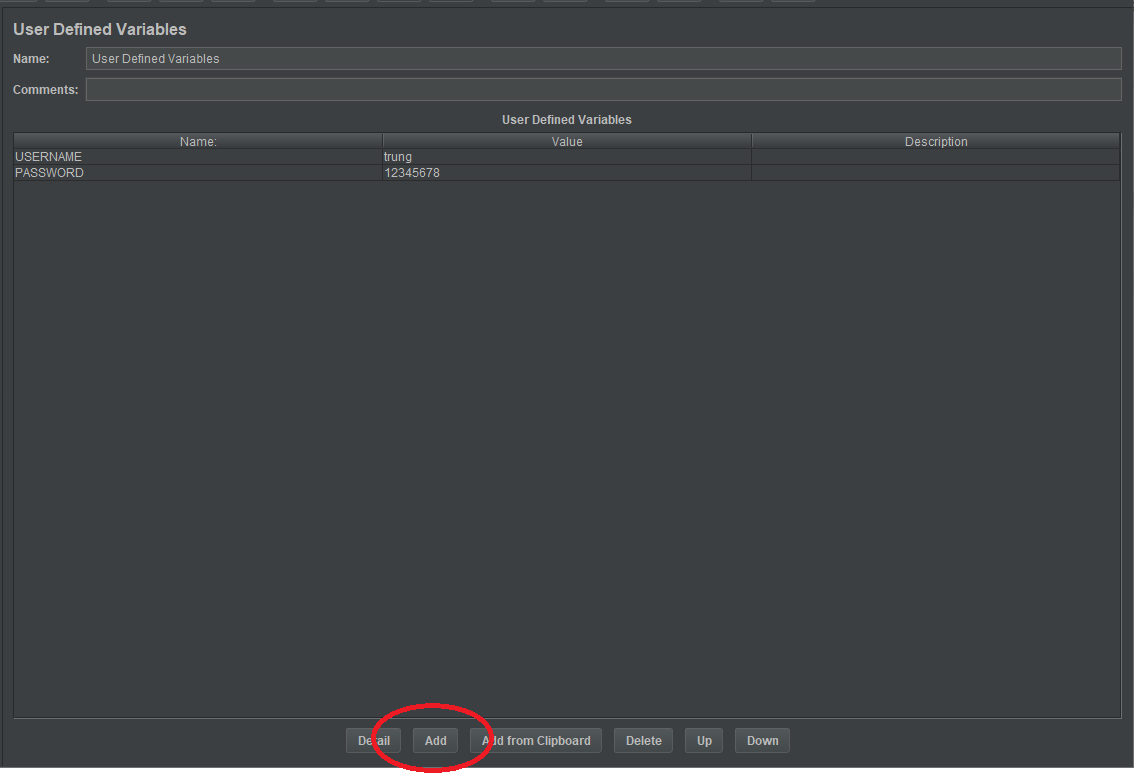
1. Tạo Test Plan thực hiện kiểm tra đăng nhập

* Cách 1: Thực hiện thủ công
  + Test Plan 🡪 Add 🡪 Config Elements 🡪 HTTP Cookie Manager
  + Test Plan 🡪 Add 🡪 Config Elements 🡪 HTTP Request Defaults, thực hiện cấu hình như hình dưới đây:



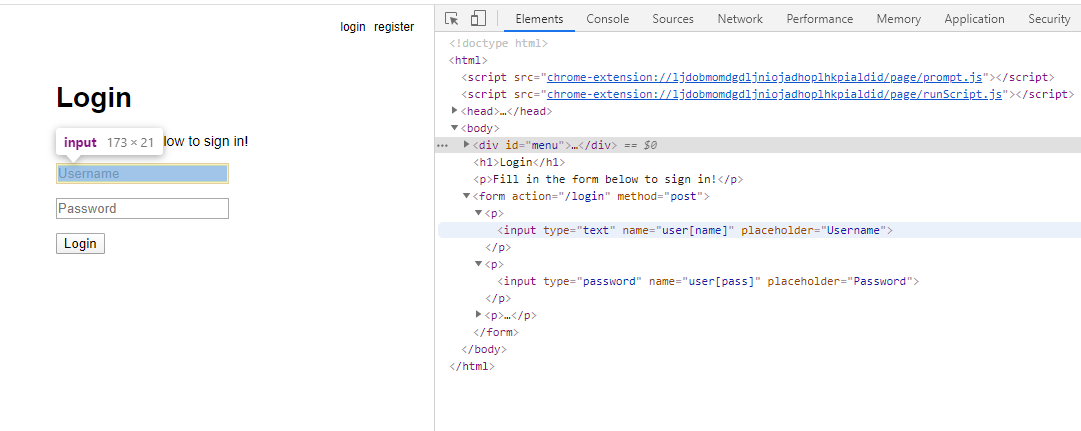
Hình 133: Configuring HTTP Request Defaults

* + Test Plan 🡪 Add 🡪 Config Elements 🡪 User Defined Variables
    - Thêm biến USERNAME và PASSWORD có giá trị như hình dưới bằng cách chọn Add. Các biến này có giá trị cố định, có thể sửa ở cột Value.

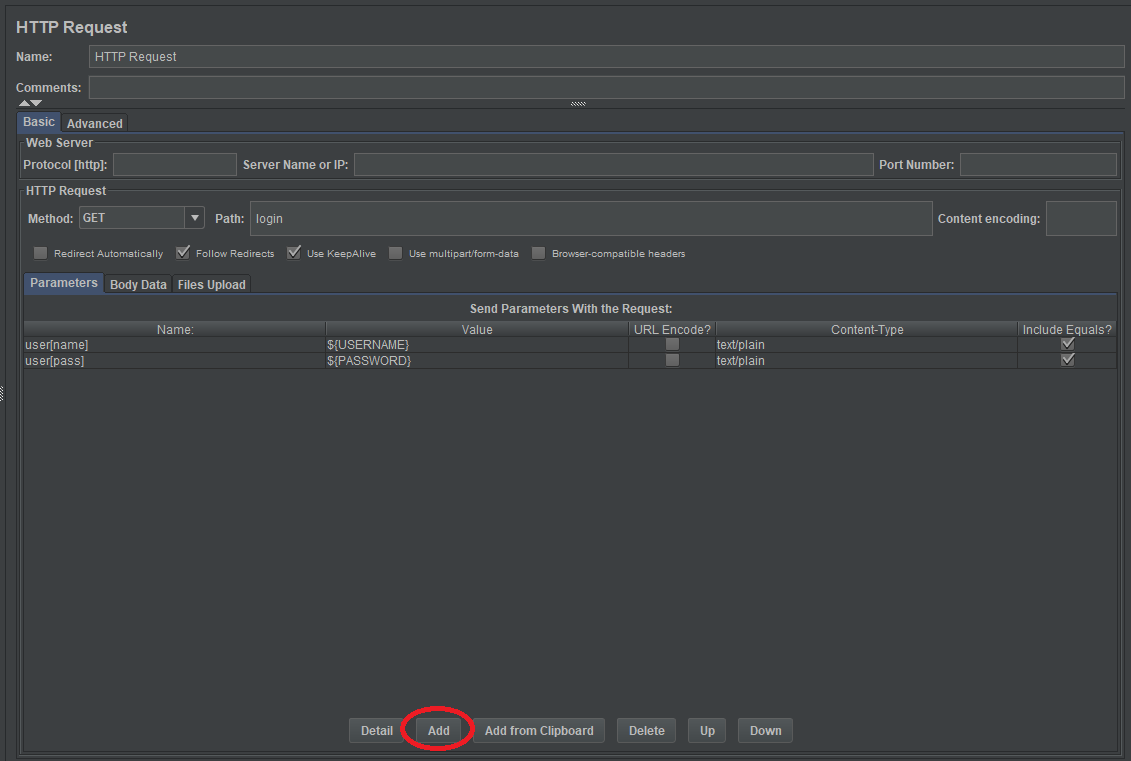


Hình 134: Setting User Defined Variables

* + Test Plan 🡪 Add 🡪 Thread Users 🡪 Thread Group
  + Thread Group 🡪 Add 🡪 Sampler 🡪 HTTP Request, cấu hình đường path là login để trỏ đến [evening-citadel-2263.herokuapp.com/](http://evening-citadel-2263.herokuapp.com/)login
  + Truy cập trang web: <http://evening-citadel-2263.herokuapp.com/> để inspect các parameter cần thiết cho HTTP Request, nhấn F12.

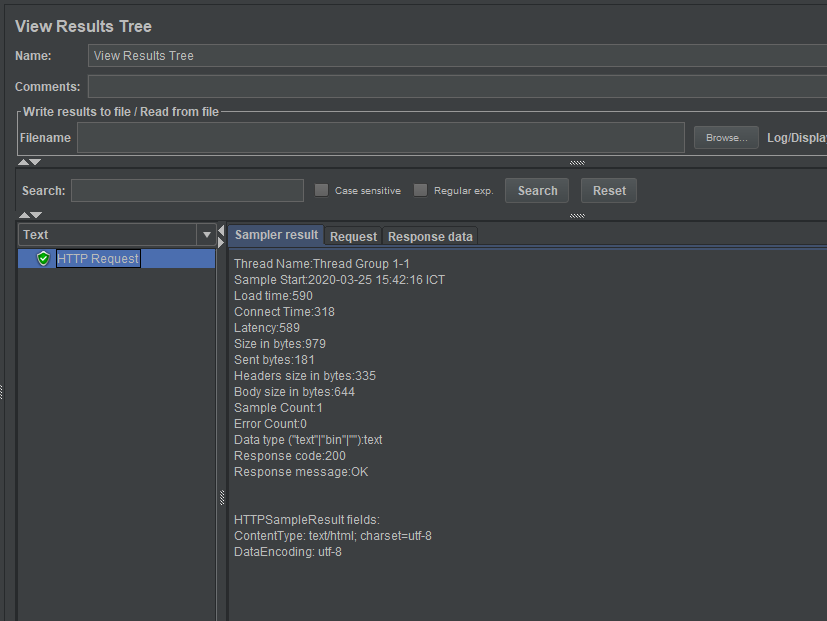


* + Ta thấy 2 parameter là user[name] và user[pass], đây chính là các parameter cần thêm vào HTTP Request. Giá trị của các parameter này chính là các biến đã tạo ở trên. Cụ thể như trong hình dưới đây:



Hình 135: Setting HTTP Request

* + - Muốn lấy được giá trị của biến, thì biến đó cần đặt trong dấu ${..}. Nếu không có, biến sẽ trở thành chuỗi giá trị. Ví dụ như trong hình 88, nếu không có cặp dấu ${…} thì user[name] sẽ có giá trị là “USERNAME” chứ không phải là “trung”.
  + Thêm View Results Tree để xem kết quả. Chạy Test Plan vừa tạo.



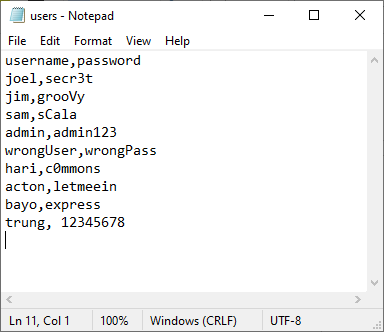
Hình 136: View Results Tree

* Cách 2: Thực hiện record tự động
  + Có thể sử dụng JMeter thông qua proxy, hoặc sử dụng BlazeMeter.
  + Cách này sẽ giúp ta tránh được sự thiếu xót khi thêm các element cho Test Plan, nhưng lại tạo nhiều element thừa (có thể remove khỏi Test Plan). Các bước sau thực hiện như Cách 1.

## **CSV Data Set Config**

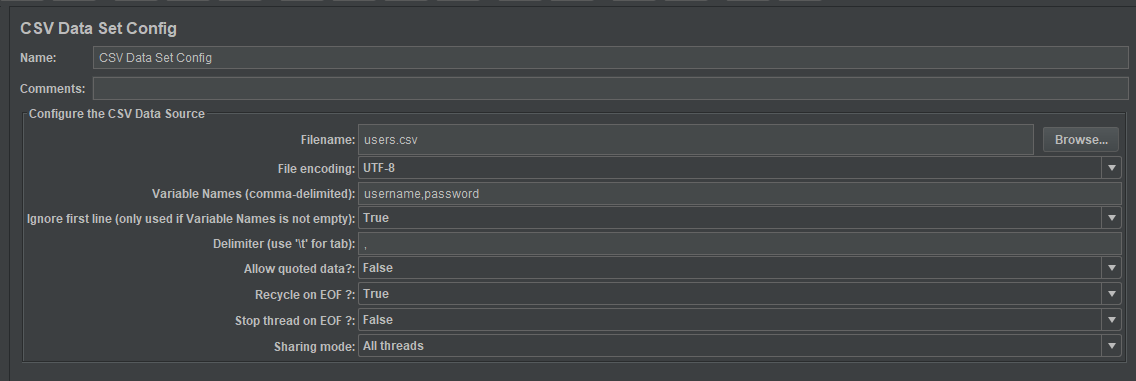
* CSV Data Set Config được dùng để đọc dòng từ file và cắt chuỗi thành các biến. Nó thích hợp để chứa một lượng lớn các biến, ví dụ trong mục này, file CSV sẽ chứa danh sách username và password. CSV Data Set Config sẽ đọc file và lấy giá trị của biến, gán vào các parameter và tiến hành đanh nhập hệ thống.
* Cách thực hiện:

1. Tạo một file CSV chứa danh sách username và password, có thể tạo bằng notepad hoặc excel và lưu dưới dạng csv file và để vào cùng thư mục với Test Plan vừa tạo.



Hình 137: Create CSV file

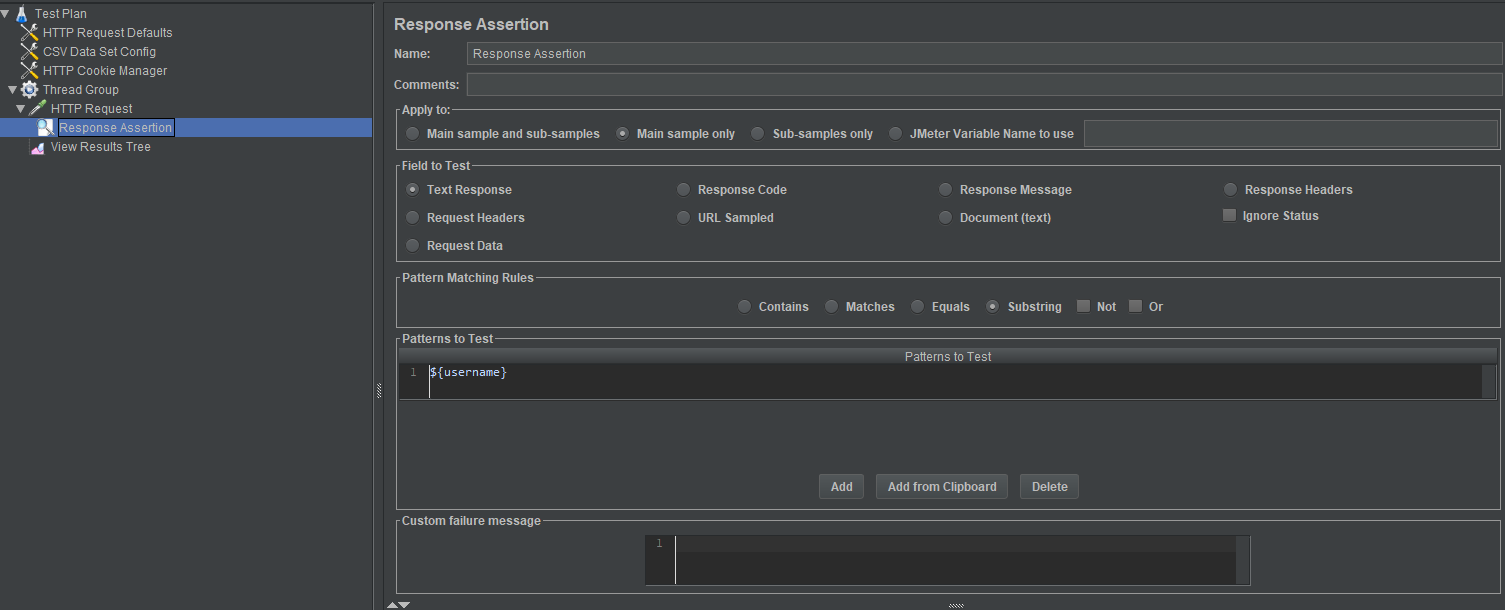
1. Test Plan 🡪 Add 🡪 Config Elements 🡪 CSV Data Set Config (Gỡ User Defined Variables khỏi Test Plan). Cấu hình CSV Data Set Config như sau:



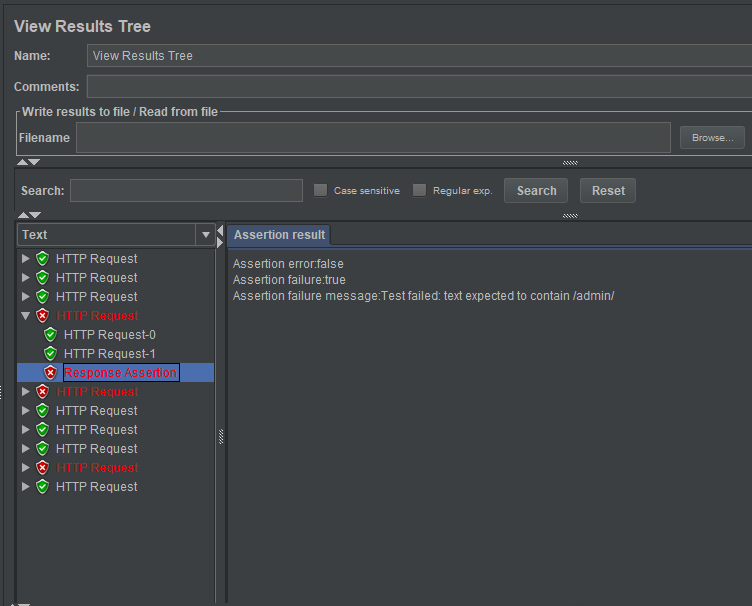
Hình 138: Setting CSV Data Set Config

* + Filename: trỏ tới file csv
  + File endcoding: chọn kiểu mã hóa văn bản, có thể không chọn
  + Variable Names: tên các variable
  + Ignore first line: nếu variable names để trống thì giá trị này là false
  + Delimiter: dấu ngắt chuỗi (thường là “,”)
  + Allow quoted data: cho phép kí tự đặc biệt
  + Recycle on EOF?: lặp lại đọc file khi đã duyệt hết file
  + Stop thread on EOF?: dừng thread khi duyệt hết file

1. Cài đặt Thread Group với số user có thể là 10, ramp – up period là 10s.
2. Chạy Test Plan và xem kết quả trong View Result Tree. Để kiểm tra đăng nhập có thành công hay không, ta có thể thêm Response Assertion ở trong HTTP Request. Trong Response Assertion, ta cài đặt kiểm tra xem response trả về có chứa tên đăng nhập không:



Hình 139: Add Response Assertion



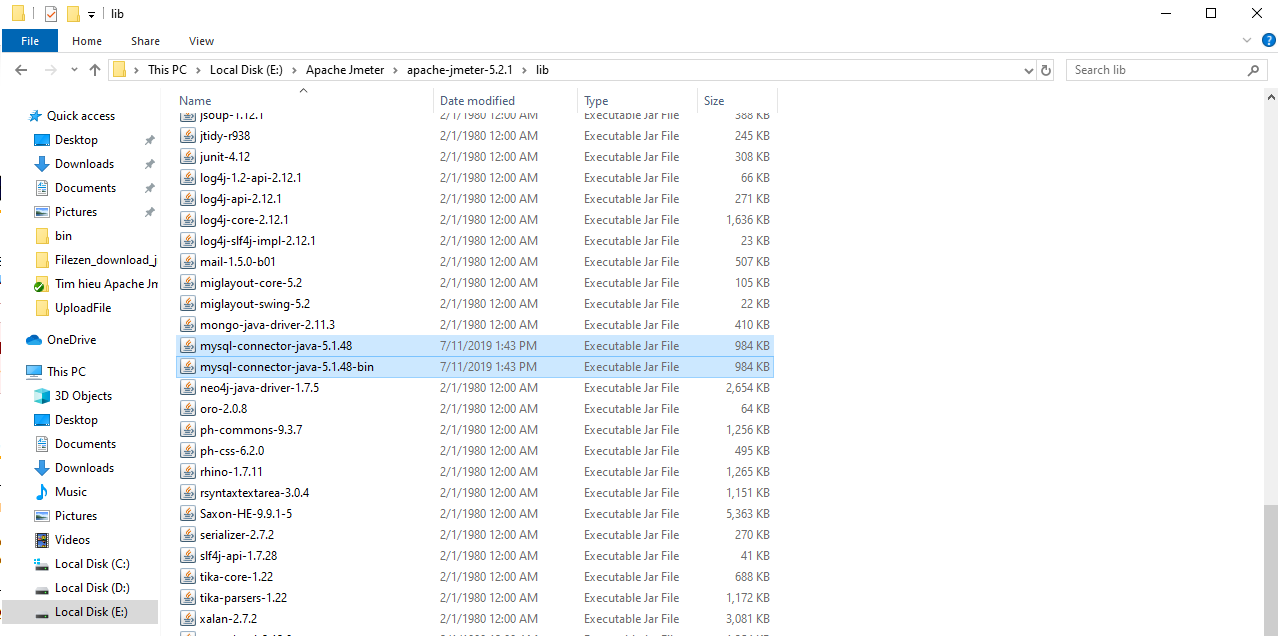
Hình 140: View Results Tree

* + Như hình 93, kết quả phản hồi có cả phản hồi lỗi là do tài khoản đăng nhập không chính xác, nên màn hình vẫn giữ nguyên ở trang đăng nhập và không chứa chuỗi ${username}.

## **JDBC Connection Cofiguration**

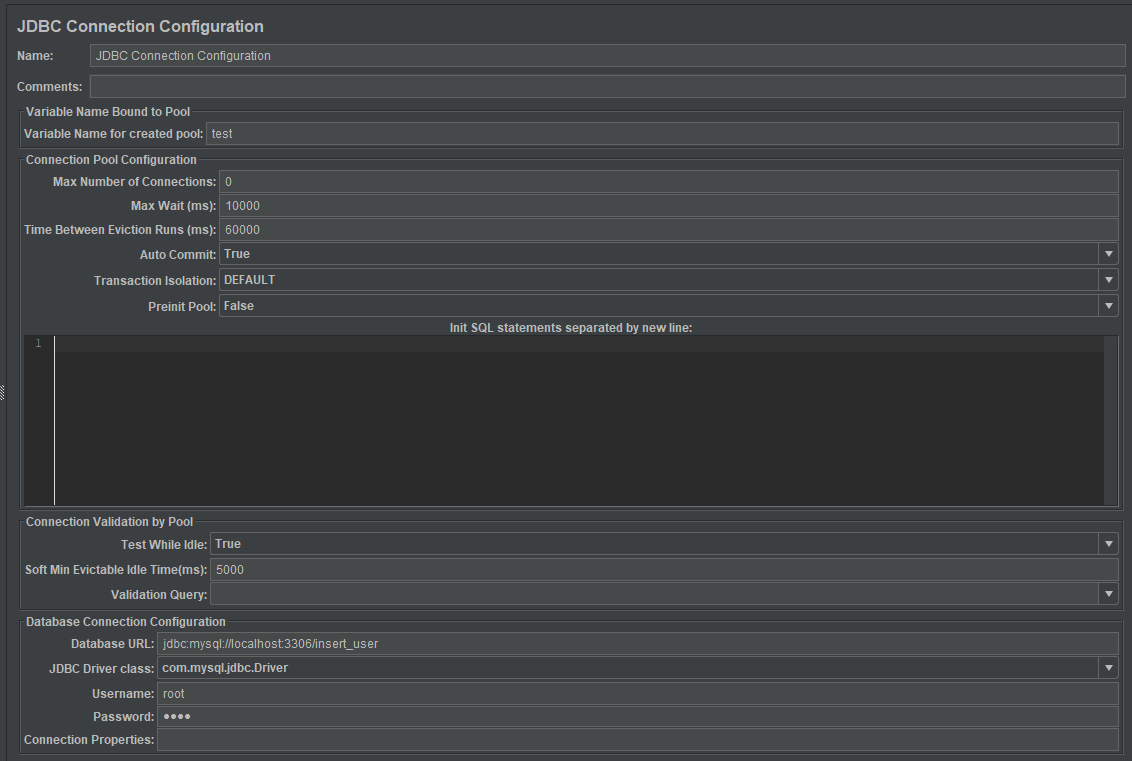
* Ở mục này, ta sẽ tìm hiểu cách để tạo một Test Plan cơ bản kiểm tra một server database. Ta sẽ tạo 50 user gửi 1 request SQL tới database server, và mỗi user thực hiện 100 lần. Tổng cộng sẽ có (50 user) x (1 request) x (100 lần) = 5000 request. Để thực hiện, ta làm như sau:

1. Download thư viện MySQL Connector (đối với JDBC của MySQL) và copy vào thư mục lib của jmeter:



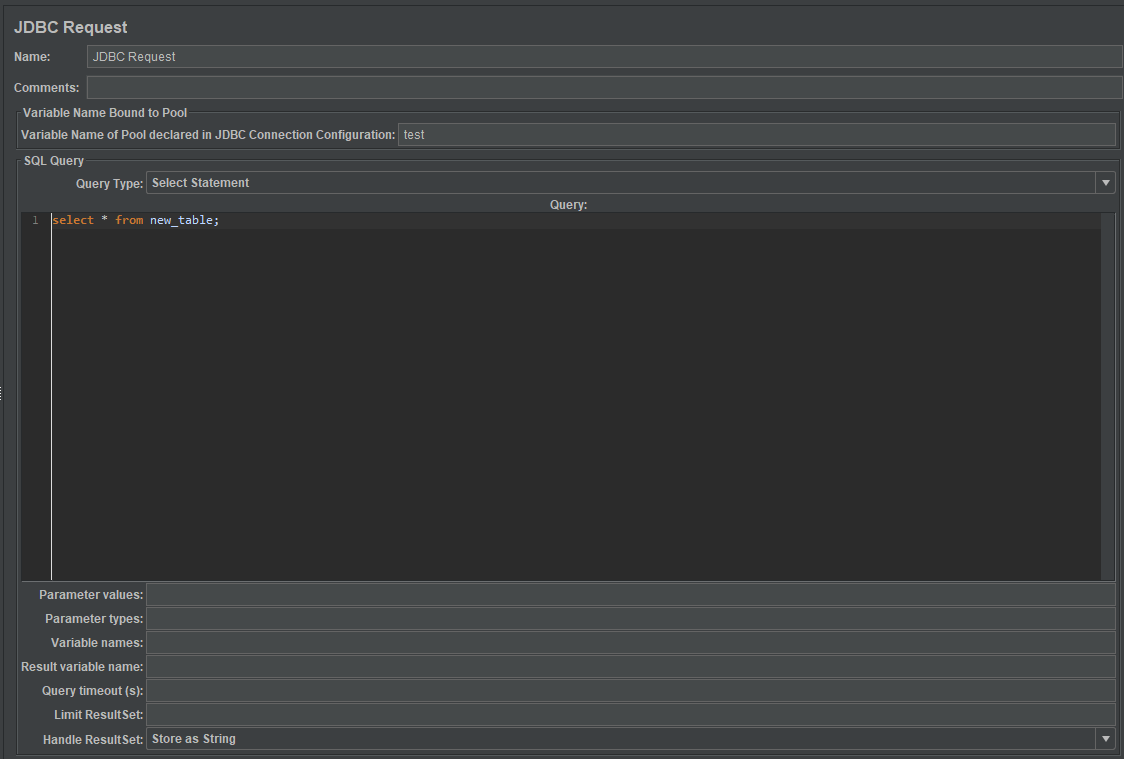
Hình 141: Adding Library Connector

1. Tạo một database, ví dụ tạo database có tên là “insert\_user” và trong đó có bảng “new\_table”.
2. Tạo Test Plan:
   * Thêm Thread Group – 50 user, 100 loop.
   * Thêm JDBC Connection Configuration: Chuột phải vào Thread Group 🡪 Config Elements 🡪 JDBC Connection Configuration
   * Cấu hình cho JDBC Connection Configuration:



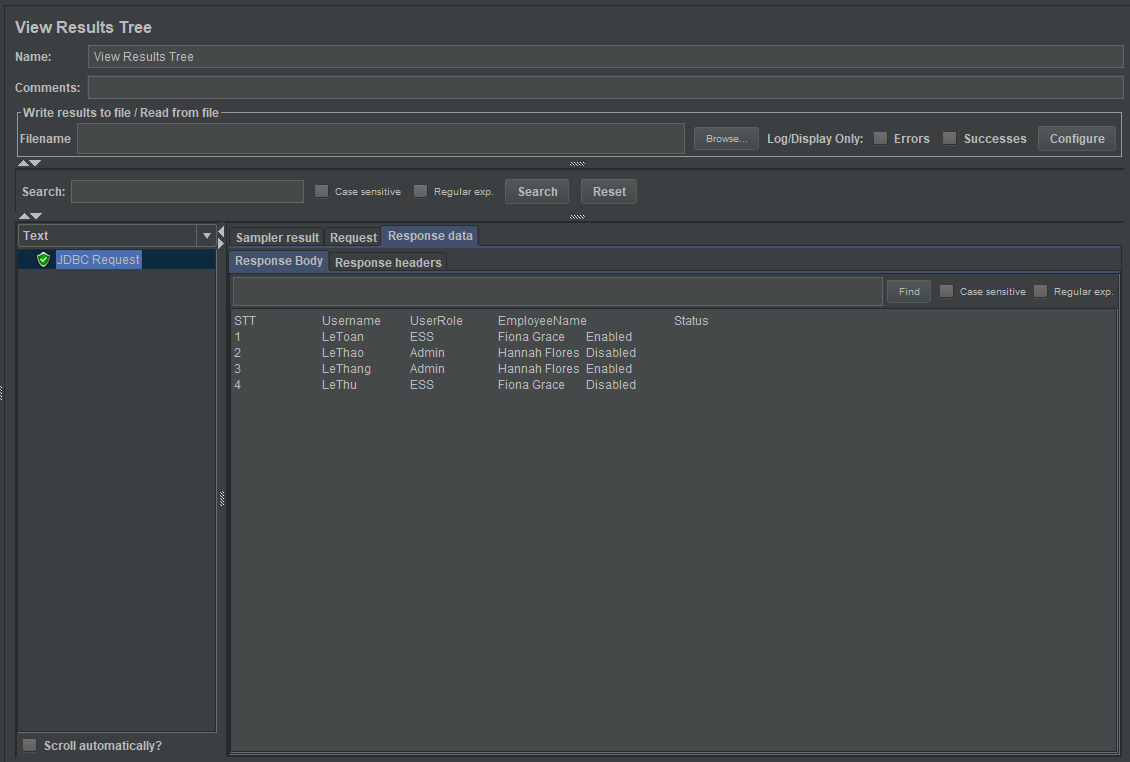
Hình 142: Configuring JDBC Connection

* + Những mục cần chú ý:
    - Database URL: đường trỏ tới database cần test
    - JDBC Class: driver của database
    - Username/password: tên đăng nhập vào mật khẩu của SQL
  + Thêm JDBC Request: Thread Group 🡪 Add 🡪 Sampler 🡪 JDBC Request



Hình 143: Setting JDBC Request

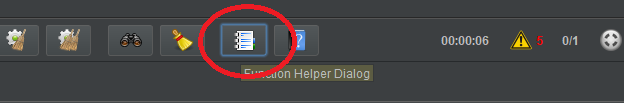
* + - Variable Name of Pool declared in JDBC Connection Configuration: phải có tên trùng với Variable Name for Created Pool trong JDBC Connection Configuration.
    - Query: vùng nhập câu lệnh SQL cần truy vấn.
    - Variable Names: có thể gán dữ liệu các cột vào từng biến, các biến ngăn cách nhau bởi dấu “,”.
  + Thêm View Results Tree để xem kết quả.



Hình 144: View Results Tree

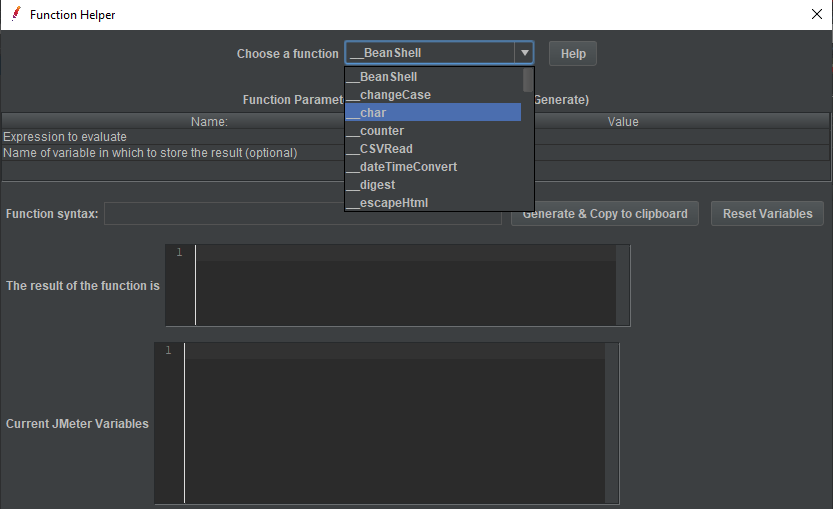
# **Sử dụng hàm trong JMeter**

* JMeter hỗ trợ các hàm để tạo giá trị cho các parameter, hoặc tạo giá trị dùng làm mẫu so sánh. Ví dụ một số hàm như: counter, randomString, escapeHtml,…
* Hàm trong JMeter có dạng: ${\_\_functionName(var1,var2,var3)}
* Để tạo một hàm, ta có thể thực hiện nhanh như sau:
  + Click Function Helper Dialog trên thanh công cụ



Hình 145: Function Helper Dialog

* + Cửa sổ Function Helper hiện lên, chọn hàm cần sử dụng trong mục Choose a function và click Generate & Copy to clipboard để lấy hàm. Có thể điều chỉnh cài đặt tạo hàm sau khi chọn hàm ở vùng Function Parameters.



Hình 146: Choose a Function and generate

**BẢNG**: DANH SÁCH HÀM CÓ THỂ TẠO NHANH TRONG FUNCTION HELPER DIALOG

|  |  |  |
| --- | --- | --- |
| **Kiểu hàm** | **Tên hàm** | **Ý nghĩa** |
| Scripting | beanShell | Chạy một đoạn mã BeanShell |
| String | changeCase | Thay đổi kiểu chuỗi hoa và thường |
| String | char | Tạo giá trị kí tự unicode từ một danh sách số |
| Calculator | counter | Returns the number of current iteration in the thread group. |
| Input | CSVRead | Đọc file CSV |
| Formatting | dateTimeConvert | Chuyển đổi ngày hoặc thời gian tới dạng mong muốn |
| Calculation | digest | Tạo một chuỗi kí tự (SHA-1, SHA-256,…) |
| String | escapeHtml | Mã hóa các kí tự HTML về dạng chuỗi kí tự hợp lệ |
| String | escapeOroRegexpChars | Trích dẫn kí tự metal được sử dụng bởi biểu thức chính quy ORO |
| String | escapeXml | Mã hóa kí tự XML |
| Variables | eval | Đánh giá một biểu thức biến |
| Variables | evalVar | Đánh giá một biểu thức chứa trong một biến |
| Input | FileToString | Đọc toàn bộ file |
| Scripting | groovy | Chạy một đoạn mã Groovy |
| Calculation | intSum | Cộng số nguyên |
| Properties | isPropDefined | Kiểm tra nếu một tài nguyên tồn tại |
| Variables | isVarDefined | Kiểm tra nếu biến tồn tại |
| Scripting | javaScript | Xử lí JavaScript |
| Scripting | jexl2 | Đánh giá 1 biểu thức Jexl2 |
| Scripting | jexl3 | Đánh giá 1 biểu thức Jexl3 |
| Infor | log | Hiển thị thông báo và trả về dữ liệu |
| Infor | logn | Hiển thị thông báo và trả về dữ liệu rỗng |
| Calculation | longSum | Cộng số thực |
| Infor | machineIP | Lấy địa chỉ IP của máy local |
| Infor | machineName | Lấy tên máy local |
| Properties | P | Đọc một tài nguyên (shorthand method) |
| Properties | property | Đọc một tài nguyên |
| Calculation | Random | Tạo một số ngẫu nhiên |
| Calculation | RandomDate | Tạo một ngày ngẫu nhiên trong khoảng ngày cho trước |
| Calculation | RandomFromMultipleVars | Trích xuất một phần tử từ cac giá trị của một tập hợp các biến được phân tách bằng “|” |
| Calculation | RandomString | Tạo một chuỗi kí tự ngẫu nhiên |
| String | regexFunction | Phân tích phản hồi trước đó bằng biểu thức chính quy |
| Infor | samplerName | Lấy tên của sampler (label) |
| Properties | setProperty | Thiết lập 1 JMeter property |
| Variables | split | Cắt một chuỗi thành từng biến |
| Input | StringFromFile | Đọc 1 dòng từ file |
| Input | StringToFile | Ghi dữ liệu vào file |
| Infor | threadGroupName | Lấy tên của thread group |
| Infor | threadNum | Lấy thứ tự thread |
| Infor | time | Trả về thời gian hiện tại ở nhiều đinh dạng |
| Infor | timeShift | Trả về một ngày ở nhiều định dạng với số lượng ngày, giờ, phút, giây được chỉ định |
| String | unescape | Xử lí chuỗi chứa Jave escape (\n, \t,..) |
| String | unescapeHtml | Giải mã kí tự HTML |
| String | urldecode | Giải mã application/x-www-form chuỗi urlencoded |
| String | urlencoded | Mã hóa một chuỗi thành chuỗi application/x-www-form |
| Calculation | UUID | Khởi tạo ngẫu nhiên loại 4 UUID |
| Variables | V | Gán giá trị cho một biến |
| Input | XPath | Sử dụng một XPath dể đọc một file |