Exercise: JMeter

This document defines the exercise assignments for the "Back-End Test Automation" course @ SoftUni

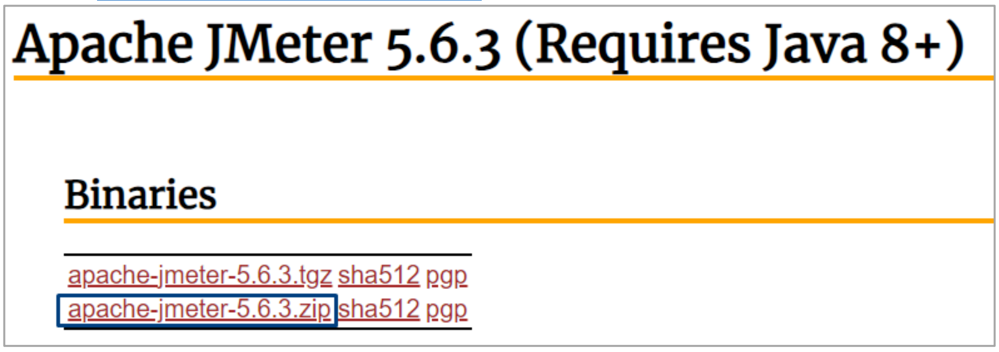
0. Prerequisites

- Since JMeter is Java-based, it runs on the operating systems that are Java compliant. Just make sure your machine has the latest version of **Java (JVM) installed**.

```
PS C:\Users\mddim> java -version
java version "21.0.2" 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
```

1. Installing JMeter

- Download the JMeter binary – "**apache-jmeter-{version}.zip**" from Binaries section of Apache JMeter Official Website

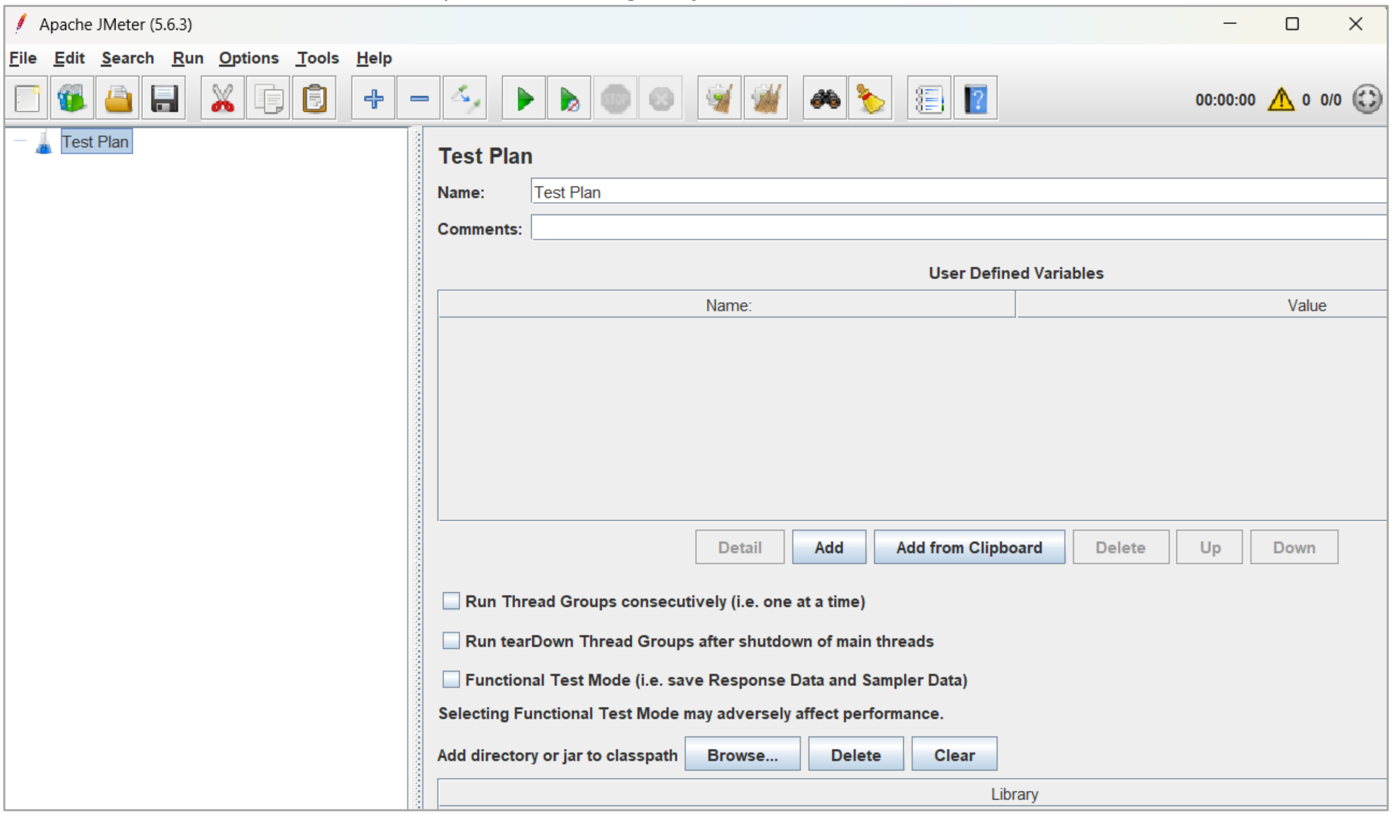


Apache JMeter 5.6.3 (Requires Java 8+)

Binaries

apache-jmeter-5.6.3.tgz sha512 pgp
apache-jmeter-5.6.3.zip sha512 pgp

- **Unzip the JMeter binary to a directory where we want JMeter to be installed**.
- Now we can launch JMeter by double clicking the **jmeter.bat** file inside the **bin folder**.



Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

00:00:00 0 0/0

Test Plan

Test Plan

Name: Test Plan

Comments:

User Defined Variables

Name:	Value

Detail | Add | Add from Clipboard | Delete | Up | Down

☐ Run Thread Groups consecutively (i.e. one at a time)

☐ Run tearDown Thread Groups after shutdown of main threads

☐ Functional Test Mode (i.e. save Response Data and Sampler Data)

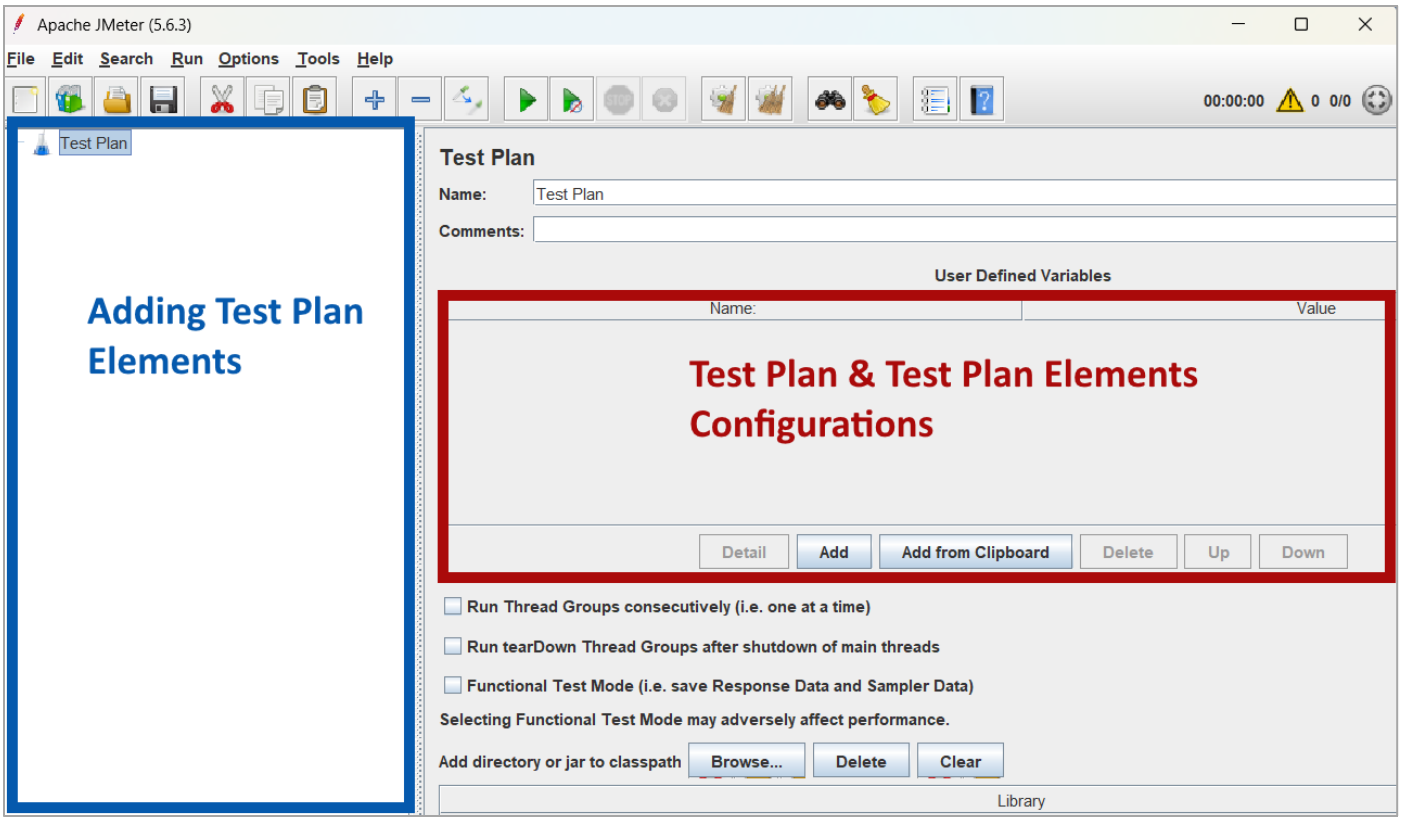
Selecting Functional Test Mode may adversely affect performance.

Add directory or jar to classpath Browse... Delete Clear

Library

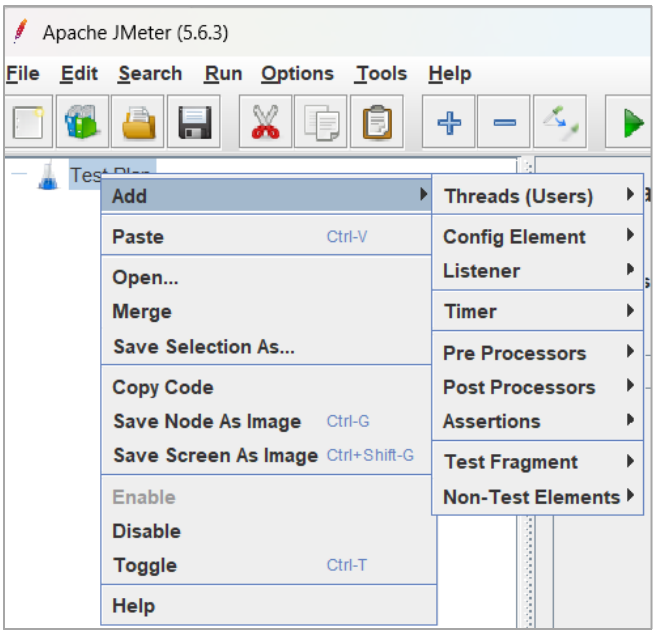
2. Test Plan

Test plan consists of **all actions and components you need to execute your test scripts**. You can set it up with **steps**, like **asking for web pages, waiting**, and **checking answers** from a website or app. Below you can notice **two panes**. The **left pane** is used for **adding** and **removing** of **Test Plan elements**, whereas the **right pane** is used to **configure** either the **Test Plan itself** or its **different elements**.


Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
00:00:00 0 0/0
Test Plan
Adding Test Plan Elements
Test Plan
Name: Test Plan
Comments:
User Defined Variables
Name: Value
Test Plan & Test Plan Elements Configurations
Detail Add Add from Clipboard Delete Up Down
Run Thread Groups consecutively (i.e. one at a time)
Run tearDown Thread Groups after shutdown of main threads
Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath Browse... Delete Clear
Library

3. Test Plan Elements

JMeter offers a **rich array of options and menus** that provide comprehensive testing capabilities.
Let's check some of them.

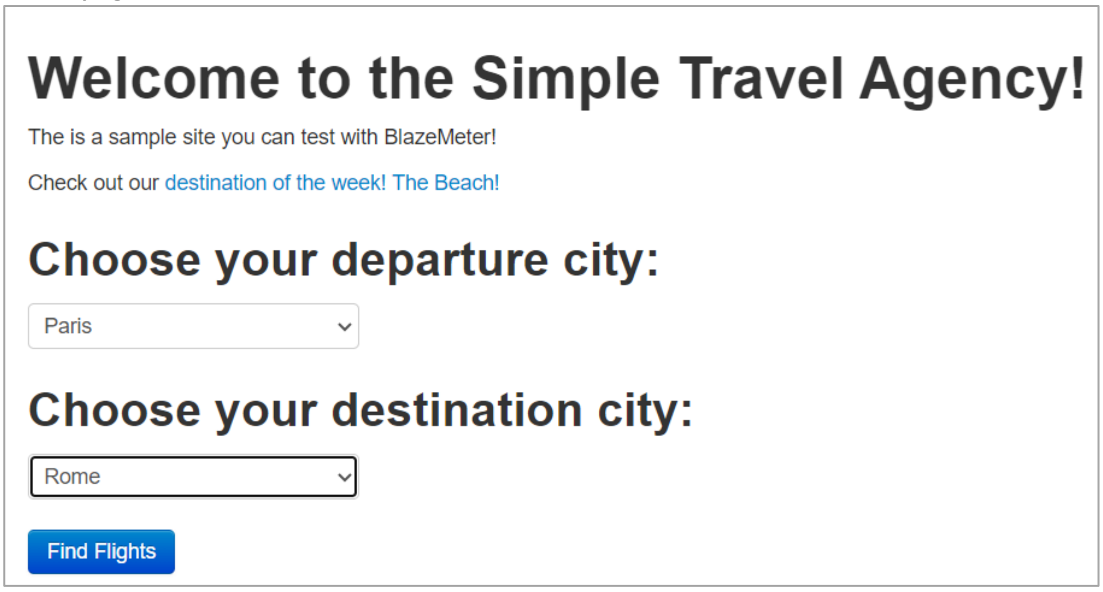


- **Threads (Users)** - Determines how many virtual users will hit the server.
- **Config Element** - Defines settings for requests, like server data or user variables.
- **Listener** - Displays results and test data in various formats.
- **Timer** - Adds delays between requests to mimic real user timing.

- **Pre-Processors** - Modifies or sets up requests before they're made.
- **Post-Processors** - Processes data after a request is made, like extracting values.
- **Assertions** - Checks if the responses are as expected.
- **Test Fragment** - Holds elements for reuse but doesn't run on its own.
- **Non-Test Elements** - Elements for notes or setup actions not part of the actual testing.
- **Samplers** - Determine the type of requests sent to the server and mimic user actions.

4. Writing your First Script – Home Page

We will use https://blazedemo.com/, which simulates a mock travel agency named **Simple Travel Agency**. The site lets us search and choose flights, which we will simulate doing through JMeter. Our **first script** will be **navigating to Home page**.



Welcome to the Simple Travel Agency!

The is a sample site you can test with BlazeMeter!

Check out our destination of the week! The Beach!

Choose your departure city:

Paris

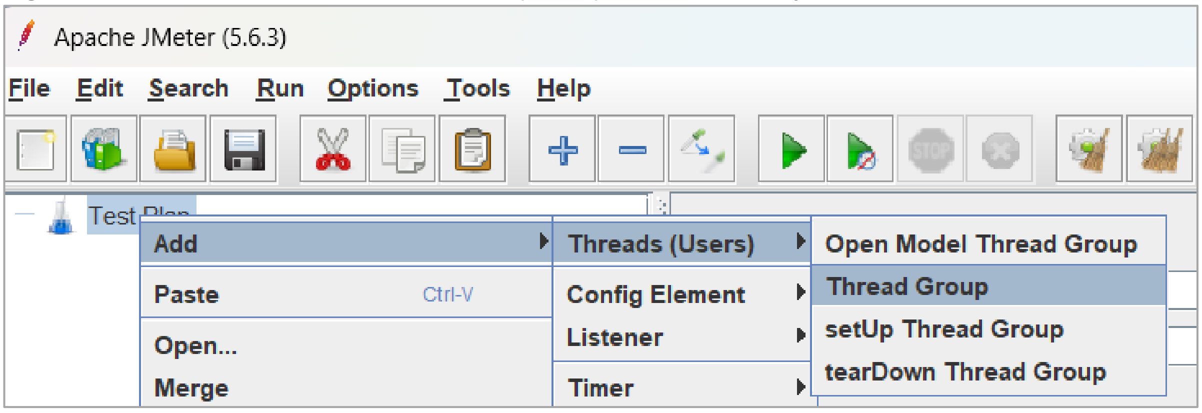
Choose your destination city:

Rome

Find Flights

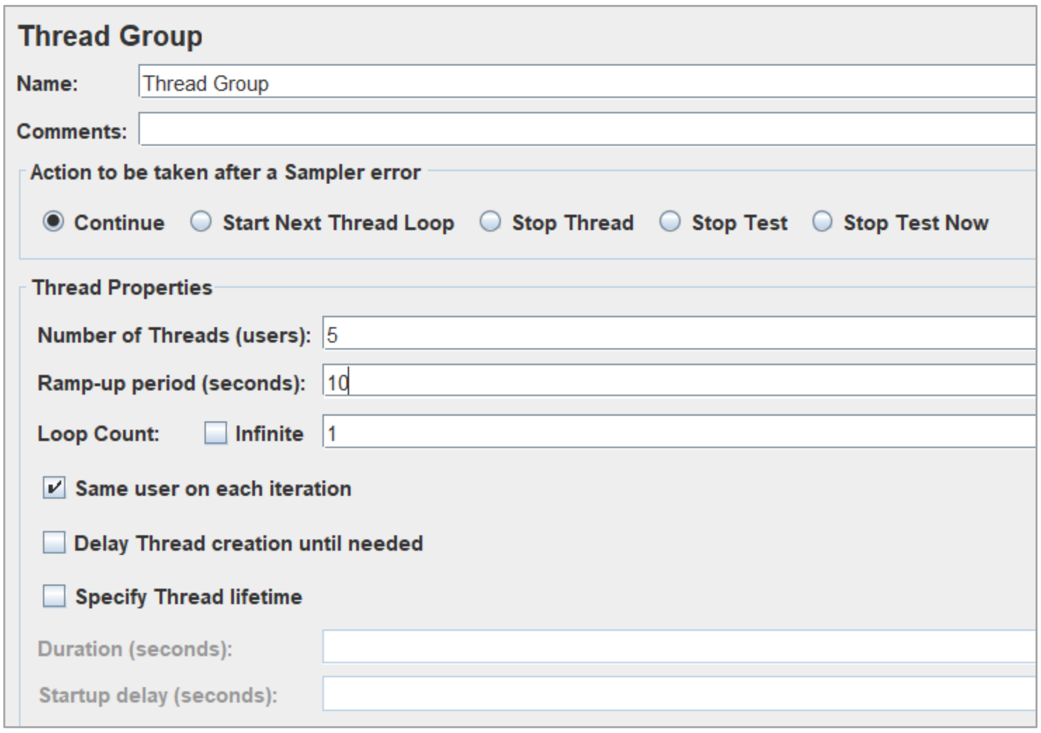
4.1. Thread Groups

To start **building the test script**, add a **Thread Group to your Test Plan**.
Thread groups determine the user flow and simulate how users behave on the app. Each thread represents a user.
Right click the Test Plan -> Add -> Threads (Users) -> Thread Group



Configure the Thread Group by setting:

- **Name:** Leave the default **"Thread Group"** name or provide a custom name if you prefer.
- **Number of Threads:** The number of users (threads) you are testing. Let's say 5.
- **Ramp-up Period:** How quickly (in seconds) you want to add users. For instance, a value of zero means all users start at once. A value of 10 means that users are added one by one, with the total number reached after ten seconds. Consider a ramp-up period of 10 seconds.
- **Loop Count:** How many times the test should repeat. Let's say 1 time (no repeat).


Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 5

Ramp-up period (seconds): 10

Loop Count: Infinite 1

[✓] Same user on each iteration

Delay Thread creation until needed

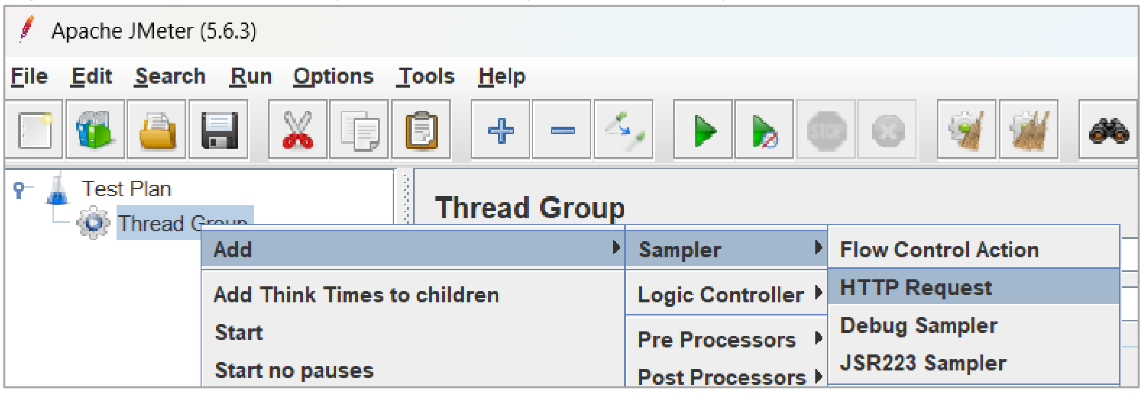
Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

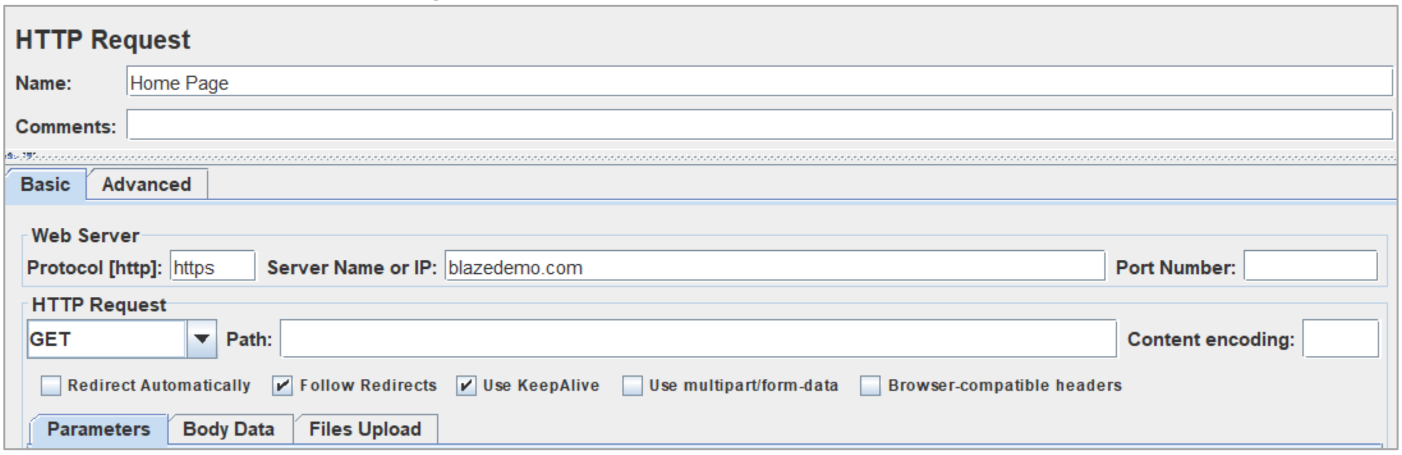
4.2. Samplers

We want to send an HTTP request to our testing site, **so add an HTTP Request Sampler**.
Right-click the Thread Group -> Add -> Sampler -> HTTP Request



Configure the Sampler by setting:

- **Name:** You can either give it a specific name or retain the default name, **"HTTP Request".**
- **Server Name or IP:** Enter the DNS name or IP address of the server. For this example, use **"blazedemo.com".**
- **Protocol:** Type **"https"** (Websites accessible through HTTP are considered insecure by browsers and are often not listed in search engine results)


HTTP Request

Name: Home Page

Comments:

Basic | Advanced

Web Server

Protocol [http]: https Server Name or IP: blazedemo.com Port Number:

HTTP Request

GET Path: Content encoding:

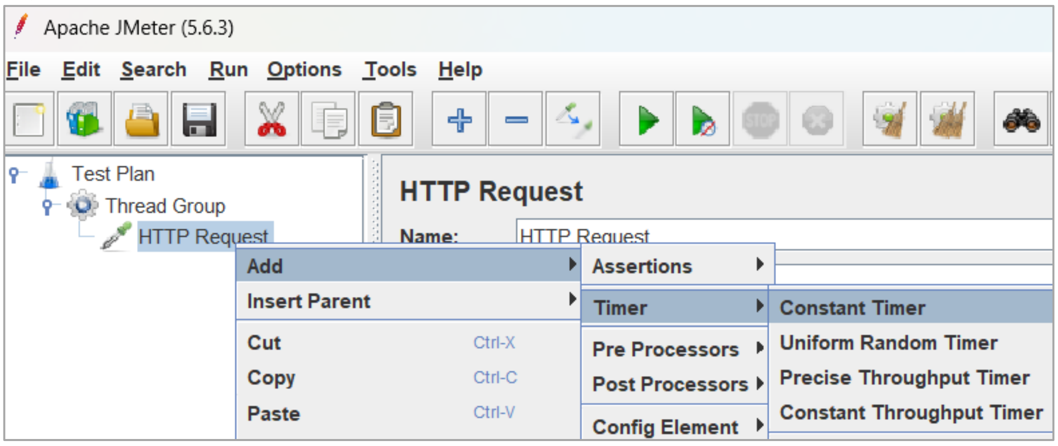
Redirect Automatically [✓] Follow Redirects [✓] Use KeepAlive Use multipart/form-data Browser-compatible headers

Parameters | Body Data | Files Upload

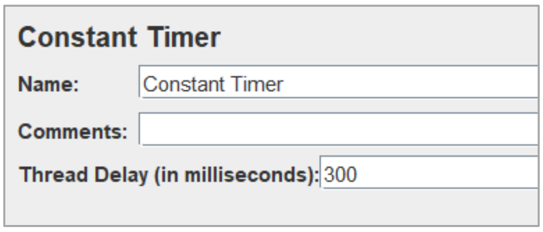
4.3. Timers

When users click on a website or app, they naturally have pauses and delays. These can be **simulated with Timers**. **Constant timers** are the **most common**. They determine **how many milliseconds to wait between requests**.

Right-click your HTTP Request -> Add -> Timer -> Constant Timer



- In our case, we will wait the default **300 milliseconds**.



Constant Timer

Name: Constant Timer

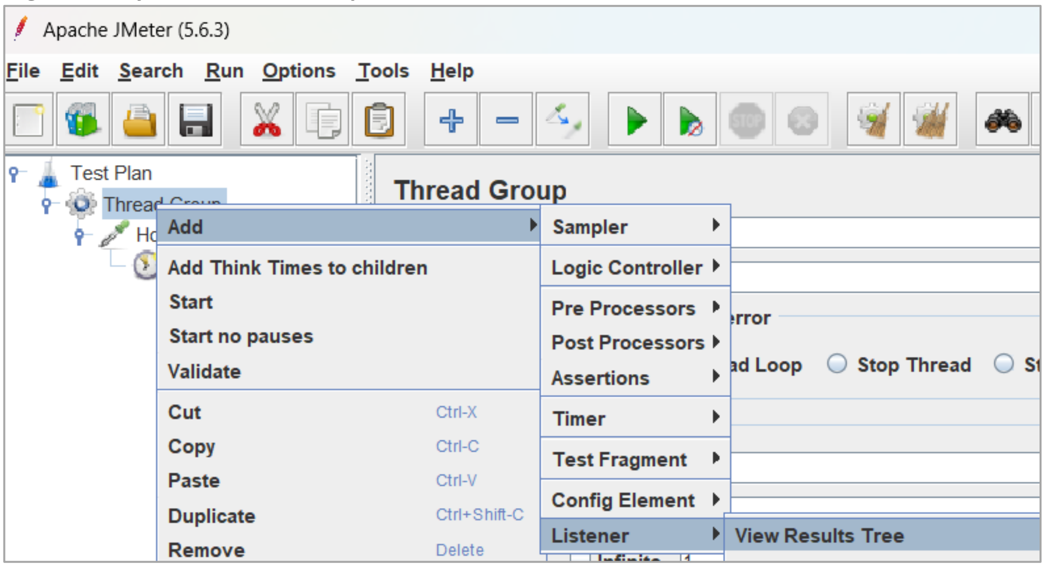
Comments:

Thread Delay (in milliseconds): 300

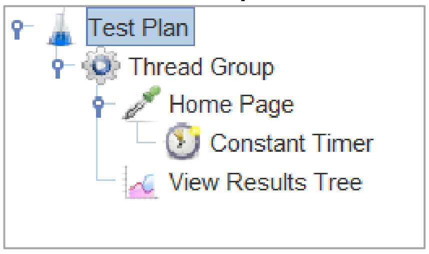
4.4. Listeners

After running the test, we want to see its results. This is done through **Listeners**, a **recording mechanism** that s**hows results**, including **logging** and **debugging** information. The **View Results Tree** is the **most common Listener**.

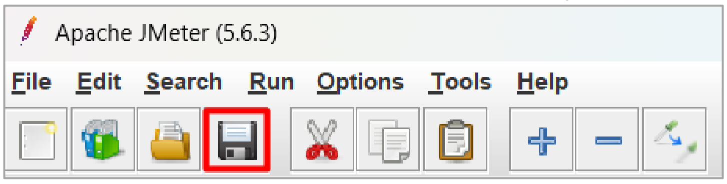
Right-click your Thread Group -> Add -> Listener -> View Results Tree



This is the basic script we created targeting the BlazeDemo page:

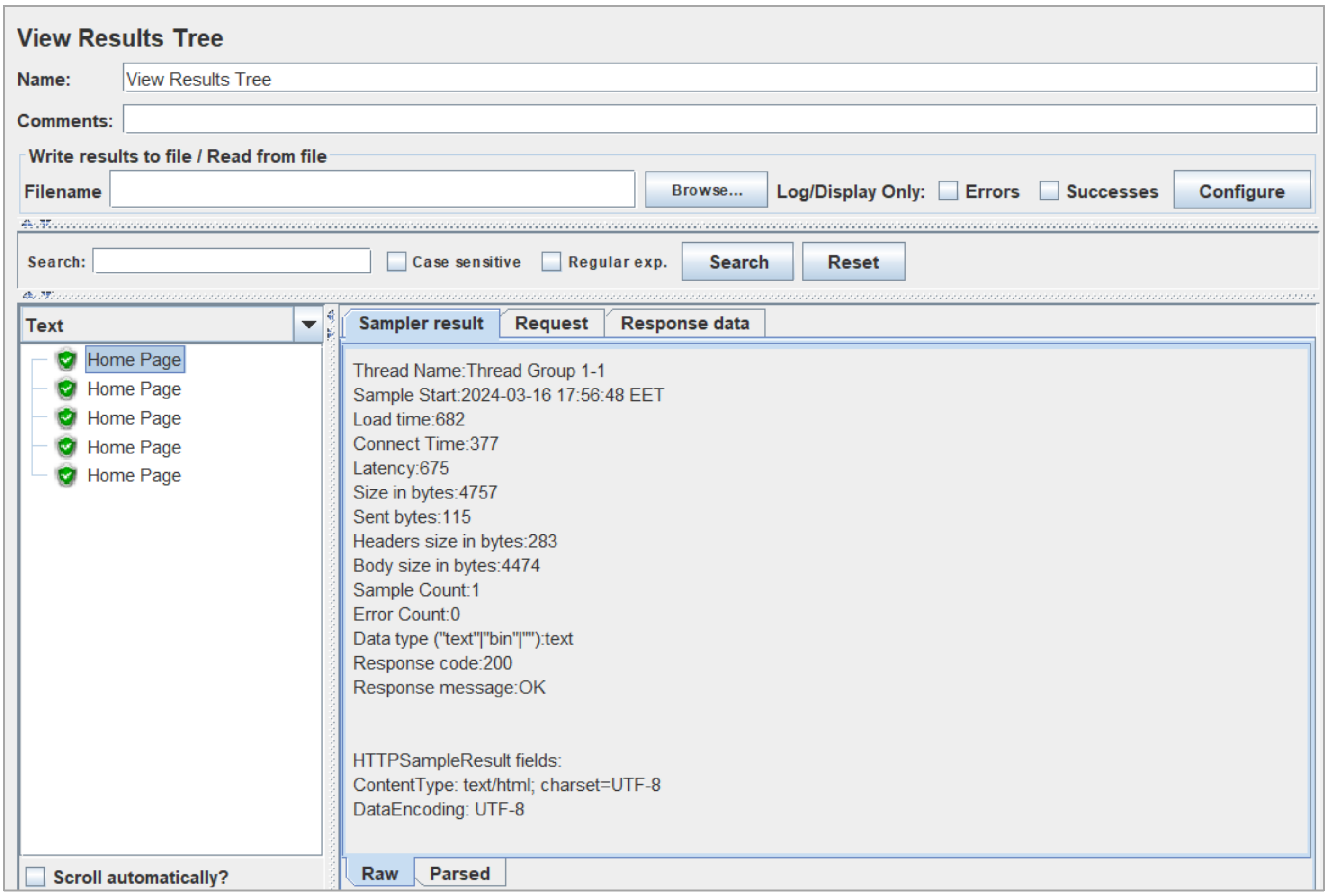


Now click "Save". Your test will be saved as a .jmx file.



4.5. Running the Test

To run the test, click the **green "Play" button** on top.
After the test completes running, you can **view the results on the Listener**.


View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Search: Case sensitive Regular exp. Search Reset
Text
Home Page
Home Page
Home Page
Home Page
Home Page
Scroll automatically?
Sampler result Request Response data
Raw Parsed

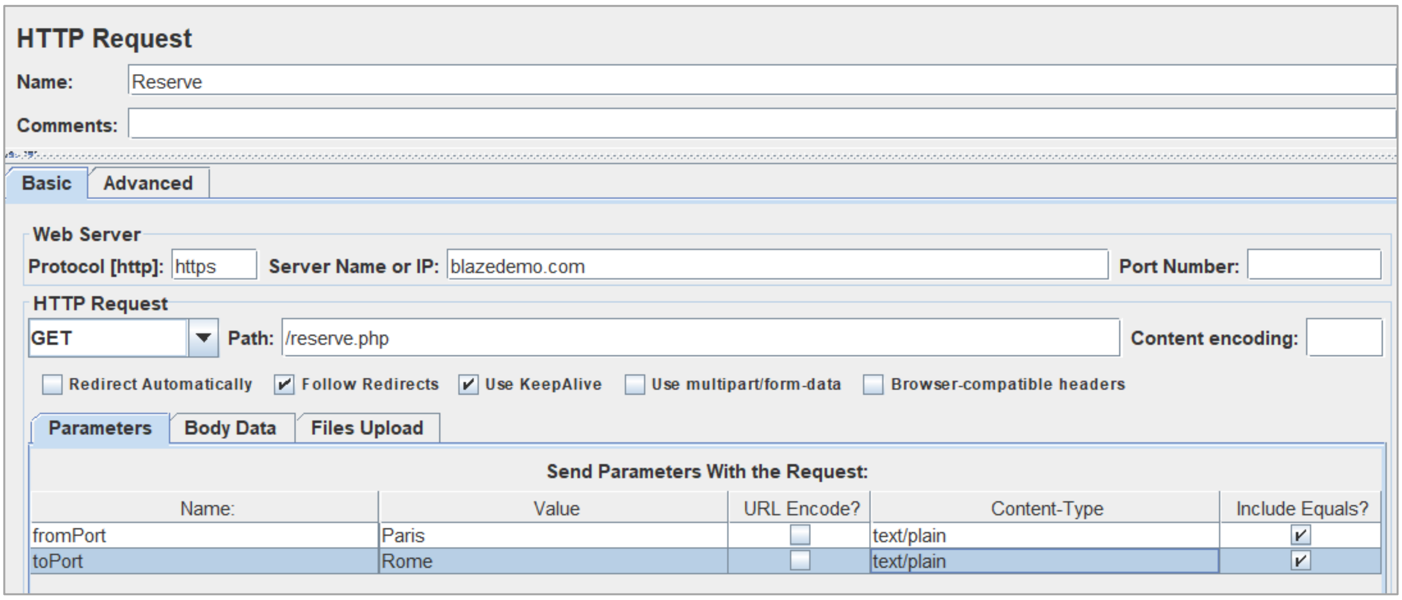
```
Thread Name:Thread Group 1-1
Sample Start:2024-03-16 17:56:48 EET
Load time:682
Connect Time:377
Latency:675
Size in bytes:4757
Sent bytes:115
Headers size in bytes:283
Body size in bytes:4474
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK


HTTPSampleResult fields:
ContentType: text/html; charset=UTF-8
DataEncoding: UTF-8
```

In this example, you can see the tests were successful because the results show green icons. **On the right**, you can see more **detailed results**, like **load time**, **connect time**, **errors**, the **request data**, the **response data**, etc. You can also save the results if you want to.

5. Second Script - Reservations

Now let's continue t by creating a **second script** and **adding parameters to it**. Whereas the first script simply navigates to home page, **this script will search for a flight**.

- When you **search for a flight** in your **web browser**, and click the **"Find Flights" button**, note that the next page adds a "**/reserve.php**" to the URL.
- With this in mind, go back to JMeter and **add another HTTP Sampler**. This time, we'll not only **specify "blazedemo.com"**, but **we'll add "reserve.php"** to the **"Path" field**. Doing so tells the sampler to build a full URL of **"blazedemo.com/reserve.php"**.
- Next, **add the parameters** required to make the selection.
(If you don't already know what parameters this webpage is designed to use for this scenario, you can use your web browser's developer tools to review the page code and find it).
- To complete the choice, this page requires, we specify a **"fromPort"** and **"toPort"**, which we'll designate **"Paris"** and **"Rome"**.



HTTP Request

Name: Reserve

Comments:

Basic | Advanced

Web Server

Protocol [http]: https Server Name or IP: blazedemo.com Port Number:

HTTP Request

GET Path: /reserve.php Content encoding:

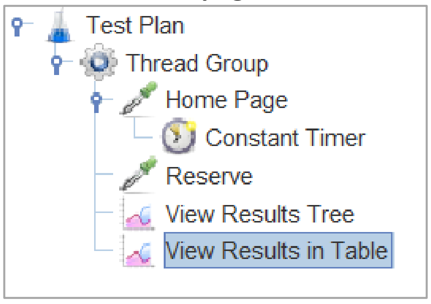
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters | Body Data | Files Upload

Send Parameters With the Request:

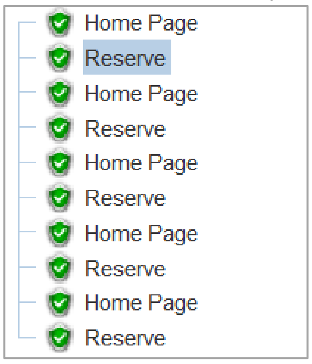
Name:	Value	URL Encode?	Content-Type	Include Equals?
fromPort	Paris	☐	text/plain	☑
toPort	Rome	☐	text/plain	☑

- We'll also add a **"View Results in Table" Listener**.
- This is the Second script, which includes **hitting the Home page**, and then **sending two parameters** to the **Reserve page**.



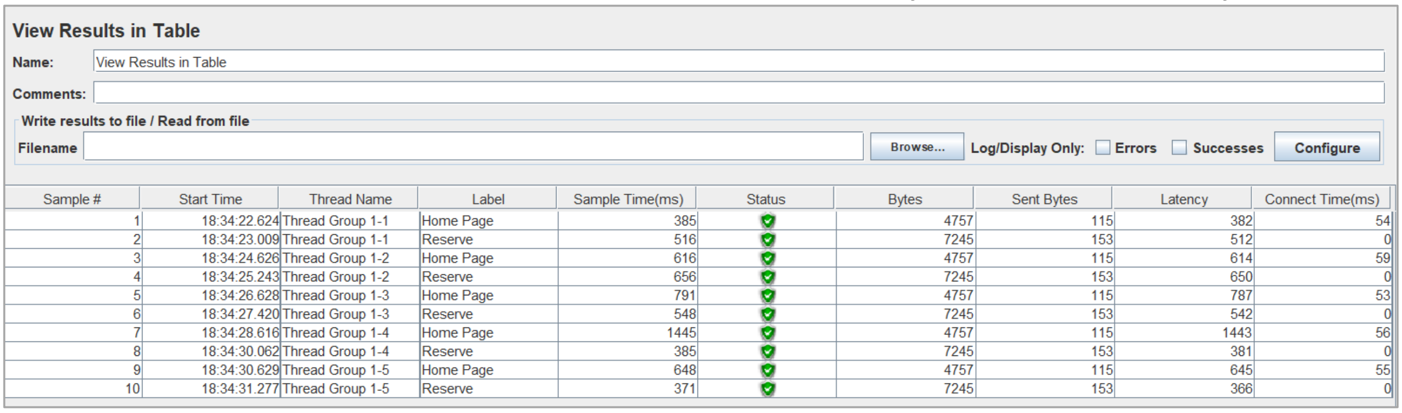
```
Test Plan
  Thread Group
    Home Page
      Constant Timer
    Reserve
    View Results Tree
    View Results in Table
```

- **Save** and **run the test**.
- In the results, you can see that **each page is hit 5 times.**



```
Home Page
Reserve
Home Page
Reserve
Home Page
Reserve
Home Page
Reserve
Home Page
Reserve
```

- The **View Results Table** shows us **additional data**: **start time**, **sent bytes**, **connect time**, **latency**, etc.



View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

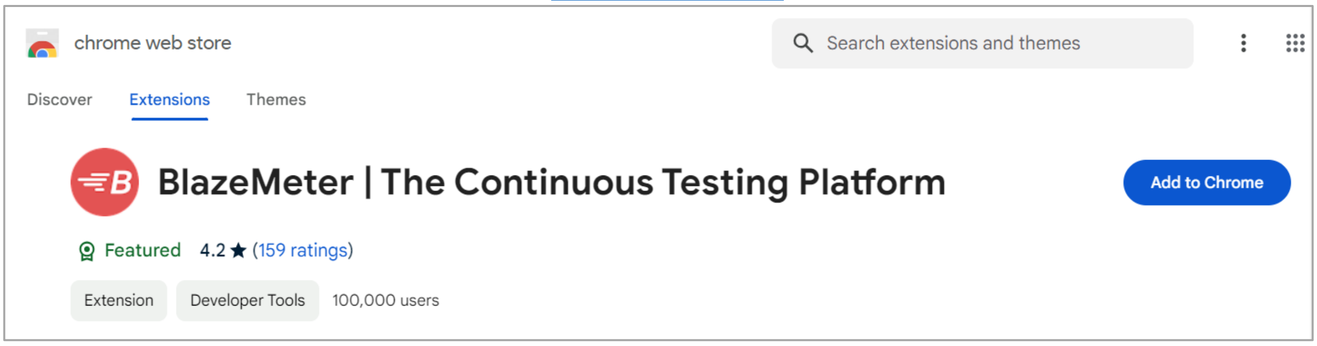
Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	18:34:22.624	Thread Group 1-1	Home Page	385	✓	4757	115	382	54
2	18:34:23.009	Thread Group 1-1	Reserve	516	✓	7245	153	512	0
3	18:34:24.626	Thread Group 1-2	Home Page	616	✓	4757	115	614	59
4	18:34:25.243	Thread Group 1-2	Reserve	656	✓	7245	153	650	0
5	18:34:26.628	Thread Group 1-3	Home Page	791	✓	4757	115	787	53
6	18:34:27.420	Thread Group 1-3	Reserve	548	✓	7245	153	542	0
7	18:34:28.616	Thread Group 1-4	Home Page	1445	✓	4757	115	1443	56
8	18:34:30.062	Thread Group 1-4	Reserve	385	✓	7245	153	381	0
9	18:34:30.629	Thread Group 1-5	Home Page	648	✓	4757	115	645	55
10	18:34:31.277	Thread Group 1-5	Reserve	371	✓	7245	153	366	0

6. Third Script - Blaze Meter Chrome Extension

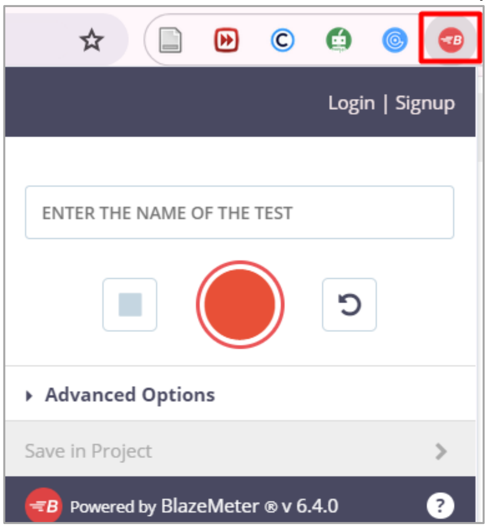
Having learned to craft basic JMeter scripts, you may notice that **constructing extensive user interactions by hand can be quite time-consuming**. An efficient alternative is to **employ recording tools**.
For recording, **options include JMeter's built-in recorder** or the **user-friendly BlazeMeter Chrome Extension**. Available for **free on the Chrome store**, the BlazeMeter extension **bypasses the need for configuring a proxy**, which is necessary for JMeter's own recorder.

6.1. Install BlazeMeter Chrome Extension

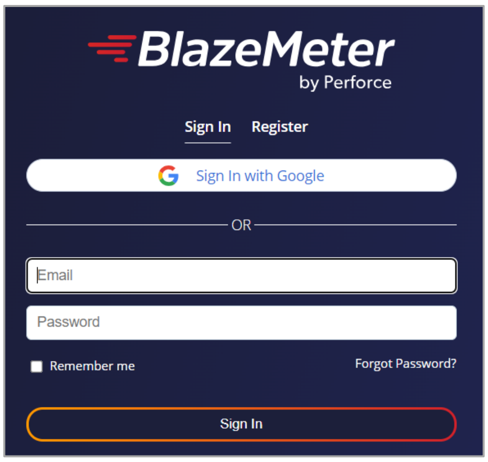
You can find BlazeMeter Chrome Extension in **Chrome Web Store**.


chrome web store
Search extensions and themes
Discover
Extensions
Themes
BlazeMeter | The Continuous Testing Platform
Add to Chrome
Featured 4.2 ★ (159 ratings)
Extension
Developer Tools
100,000 users

- Click on **"Add to Chrome" button.**
- The **Extension** is then **added** to your Chrome browser.



- You can start recording immediately, but **in order to save the recording in .jmx format** you have to **Signup/Login.** This could be done via your Google profile and it is pretty easy to do, so we won't explain it in details.



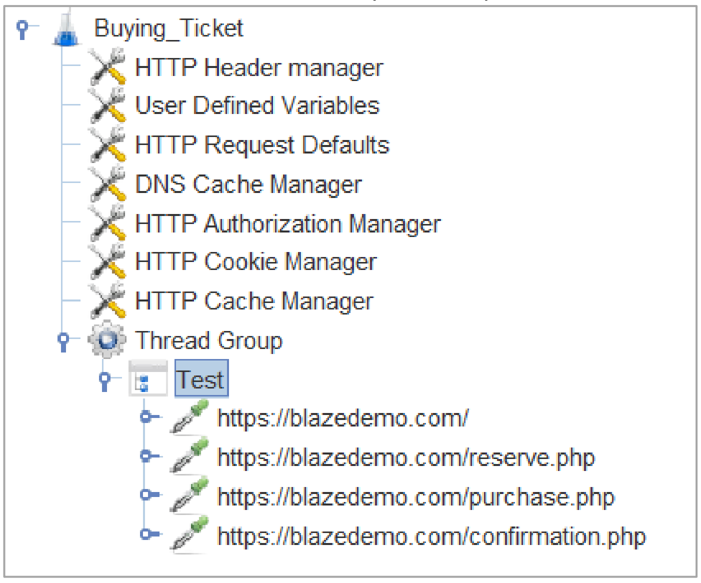
- When you Login **head back** to **Chrome extension**.

6.2. Simulate and Record User Scenario for Testing

- **Add a name for your** test and click the **red Record button**.
- Through your Chrome browser, **simulate the user scenario you want to test by clicking away**. In our case:
    - **Navigate** to https://blazedemo.com
    - **Search** for **flights** from Paris to Berlin
    - **Choose** the **Lufthansa** flight
    - **Fill in** and **submit the form**
    - And **buy the ticket**
- When you're done, stop the recording, and **export the recording to .jmx**.
- Don't forget to move the downloaded script to your preferred folder.

6.3. Edit the Recorded Test Plan in JMeter

- Open the **.jmx file file in JMeter**.
- You will be able to see your test plan, which was created from the recording.



```
Buying_Ticket
  HTTP Header manager
  User Defined Variables
  HTTP Request Defaults
  DNS Cache Manager
  HTTP Authorization Manager
  HTTP Cookie Manager
  HTTP Cache Manager
  Thread Group
    Test
      https://blazedemo.com/
      https://blazedemo.com/reserve.php
      https://blazedemo.com/purchase.php
      https://blazedemo.com/confirmation.php
```

Note: The plan **has new elements**, like **Cookie Manager** and **Cache Manager**. They are here because browsers keep cookies and cache, and they were captured in the recording. You can **clear them up if you need to**. These **entirely optional elements** are present to simulate web browser behavior more closely.

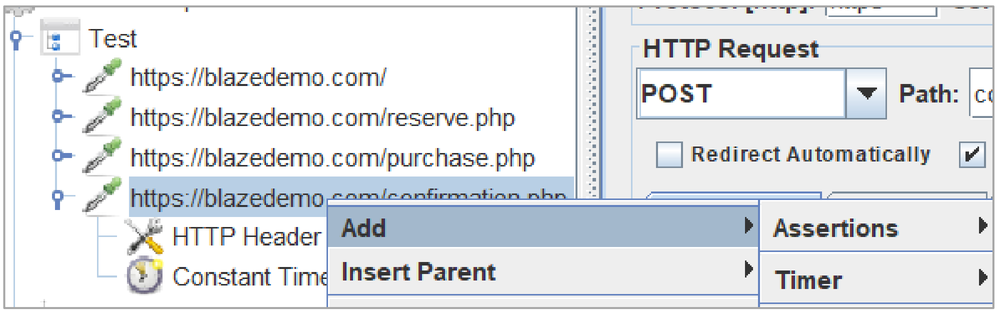
- **Don't delete User Defined Variables. Taka a look what these are doing!**
- **Add a listener** and run the test.

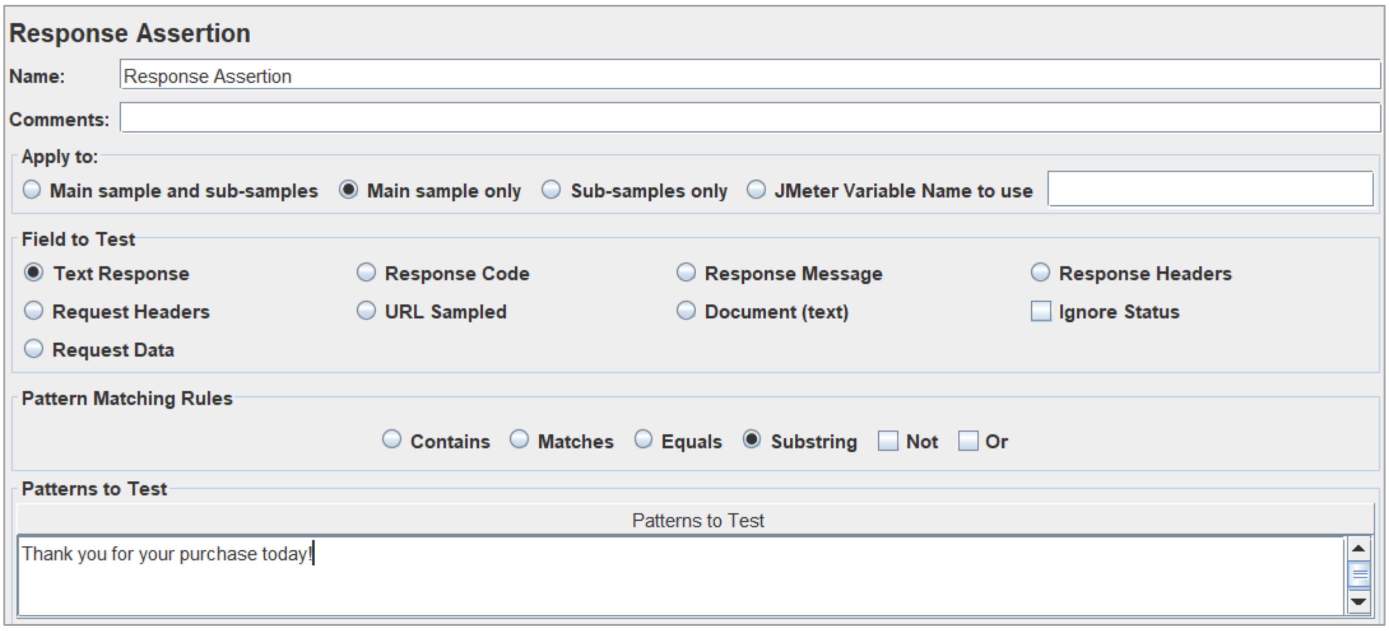
Note 2: Sometimes the **HTTP requests are recorded twice**. For example: You may have two requests to https://blazedemo.com, two requests to https://blazedemo.com/reserve.php, etc. This is normal, don't worry about it. Just delete the second request.

6.4. Assertions

Assertions are **elements that let you check for errors**, or in other words - determine if your **test passes or fails**. JMeter automatically marks requests with HTTP status codes below 400 as successful, but it might be the case that server responded with HTTP status code 200 OK but the response is full of errors or simply absent.

- We want to make sure the webpage returns the information we're looking for.
- In our example, we want to **make sure users who purchase flights**, **receive a message** on the **confirmation page** saying: **Thank you for your purchase today!**
- Go to JMeter. **Add a Response Assertion (you know how to do it).**
- **Add the exact characters you want users to see**.




Response Assertion

Name: Response Assertion

Comments:

Apply to: ( ) Main sample and sub-samples [✓ Main sample only] ( ) Sub-samples only ( ) JMeter Variable Name to use

Field to Test: [✓ Text Response] ( ) Response Code ( ) Response Message ( ) Response Headers ( ) Request Headers ( ) URL Sampled ( ) Document (text) ☐ Ignore Status ( ) Request Data

Pattern Matching Rules: ( ) Contains ( ) Matches ( ) Equals [✓ Substring] ☐ Not ☐ Or

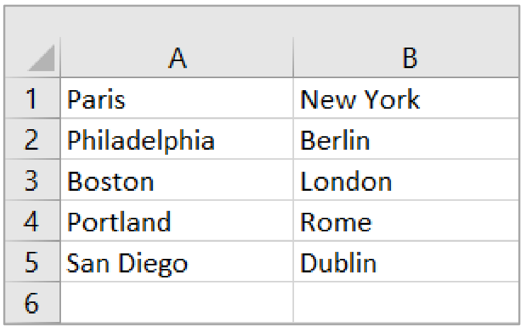
Patterns to Test: Thank you for your purchase today!

- **Save** and **run the test**.
- If the **response contains the string**, the test will pass. If not, the **test will fail, and it will also detail why**.

7. Dynamic Data

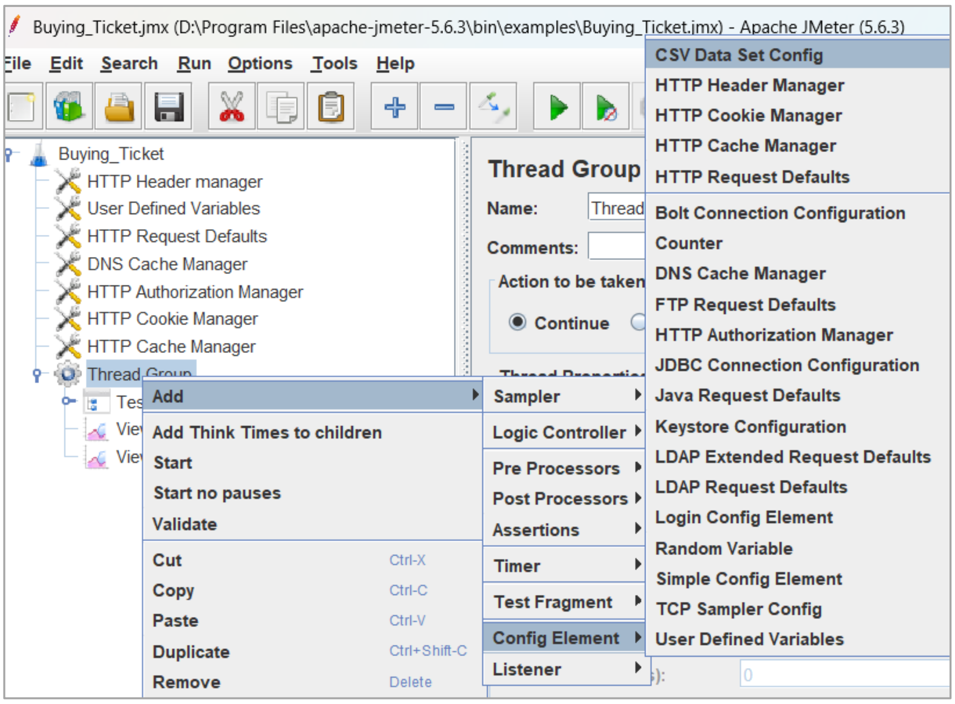
What happens when you want to **create a dynamic script**, which chooses **different parameters each time you test**, like passwords, login information, or search criteria? This is what Dynamic Data through CSV files is for.

- **Create a CSV file on your computer**, consisting of the **different variables you are testing**.
- Put the file in the JMeter folder. In our case, we created a basic one, with departure and destination cities.

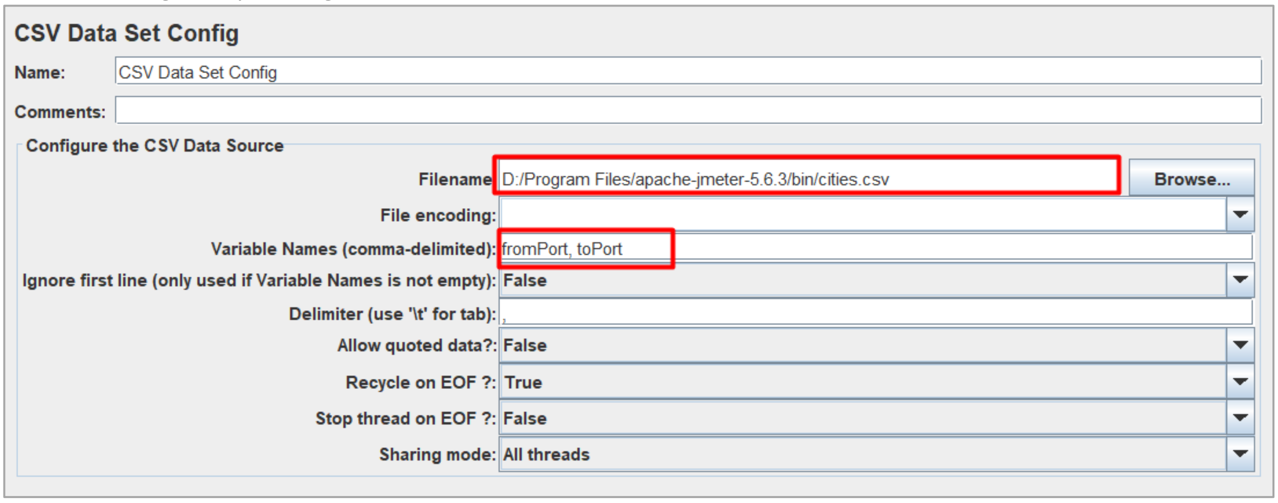


	A	B
1	Paris	New York
2	Philadelphia	Berlin
3	Boston	London
4	Portland	Rome
5	San Diego	Dublin
6		

- **Right-click your thread group -> Add -> Config Element -> CSV Data Set Config**



- Configure by adding the variable names. In our case, **fromPort** and **toPort**.



CSV Data Set Config

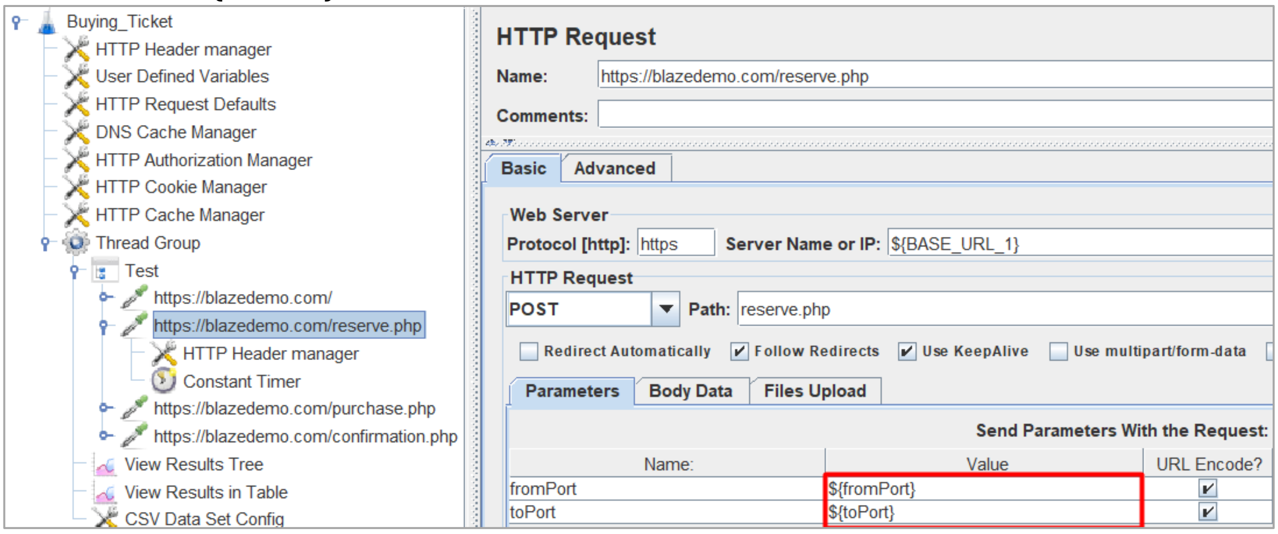
Name: CSV Data Set Config

Comments:

Configure the CSV Data Source

Filename	D:/Program Files/apache-jmeter-5.6.3/bin/cities.csv
File encoding:	
Variable Names (comma-delimited):	fromPort, toPort
Ignore first line (only used if Variable Names is not empty):	False
Delimiter (use '\t' for tab):	,
Allow quoted data?:	False
Recycle on EOF ?:	True
Stop thread on EOF ?:	False
Sharing mode:	All threads

- Go back to the HTTP Request (from second script) and change the variable from the specific name (of the cities) to the general name:
  - `${fromPort}`
  - `${toPort}`



HTTP Request

Name: https://blazedemo.com/reserve.php

Comments:

Web Server

Protocol [http]: https   Server Name or IP: ${BASE_URL_1}

HTTP Request

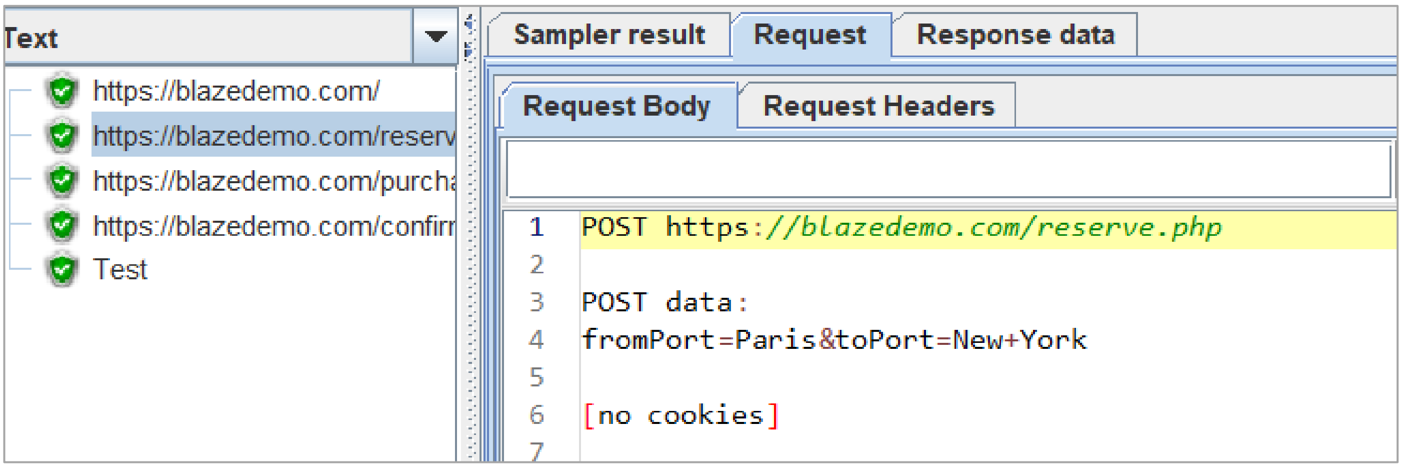
POST   Path: reserve.php

☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data

Send Parameters With the Request:

Name:	Value	URL Encode?
fromPort	${fromPort}	✓
toPort	${toPort}	✓

- The **data tested** will now **come from the CSV file**, and you will be able to see the **dynamic results in the View Results Tree**. In our example, it's no longer Paris and Rome, but Philadelphia and Berlin, Paris and New York, etc.



```
POST https://blazedemo.com/reserve.php

POST data:
fromPort=Paris&toPort=New+York

[no cookies]
```

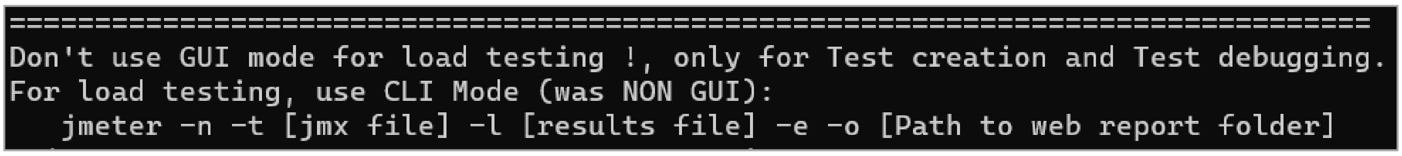
8. Load Testing

Always perform load testing responsibly, with permission, on systems you own or have explicit authority to test!
Targeting a service that you don't own, with an extreme load, such as 10,000 virtual users, firstly will overwhelm your own machine. Your laptop or computer will run out of CPU power, memory, and other resources, causing your system to crash way before reaching the targeted service. Also, if your internet service provider (ISP) detects unusual traffic, it may throttle internet speeds or block access, as such activities are considered a cyber-attack. Legal repercussions are also a risk, since service providers' "Terms of service" typically prohibit unauthorized load testing.

- With all this in mind, let's continue with **JMeter CLI mode**.

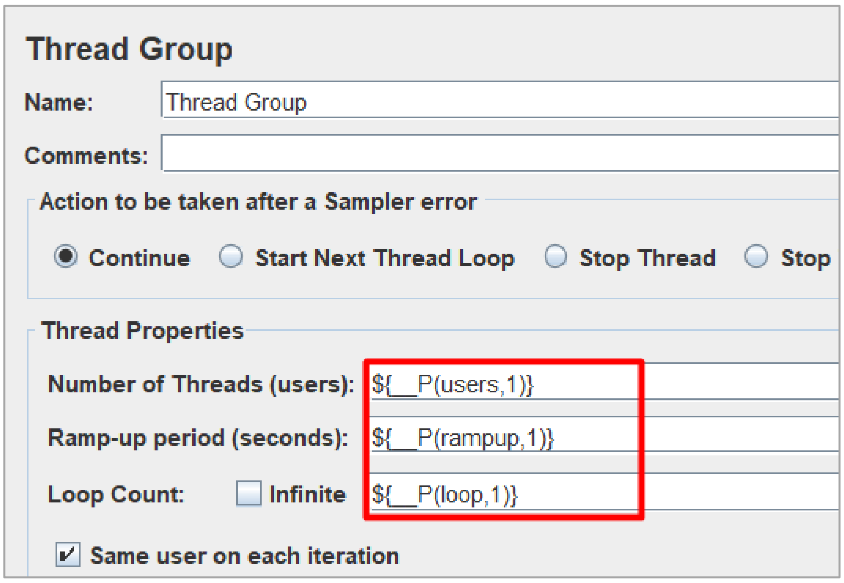
9. JMeter CLI Mode

For load testing, it's highly recommended to use JMeter in CLI mode.
This command-line mode **consumes fewer system resources**, enabling the simulation of a larger number of users and providing more accurate performance results.
JMeter's GUI mode is mainly for creating and debugging your test plans, not for running them under load.



```
================================================================================
Don't use GUI mode for load testing !, only for Test creation and Test debugging.
For load testing, use CLI Mode (was NON GUI):
   jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
```

- Before we continue, we have to make some **changes to your Thread group**.
- Change the **Number of Threads (users), Ramp-up period (seconds) and Loop Count** like this:



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop

Thread Properties

Number of Threads (users): ${__P(users,1)}

Ramp-up period (seconds): ${__P(rampup,1)}

Loop Count: Infinite ${__P(loop,1)}

[✓ Same user on each iteration]

This approach uses JMeter's property function **`${__P(name, default)}`** to **assign variable values through the command line interface (CLI)**, rather than hardcoding them into the test plan. If these values are not specified in the CLI, the test will run with the default values provided in the test plan (which in our case is 1).

- **Save your Test Plan and close JMeter GUI.**
- **Open CMD or PowerShell, or Terminal of your choice** (We are using CMD in this tutorial).
- **Navigate to the JMeter 'bin' folder.**

```
D:\Program Files\apache-jmeter-5.6.3\bin>
```

- To execute a JMeter test from the command line when **your current directory is the JMeter 'bin' folder**, you use the **following syntax**:

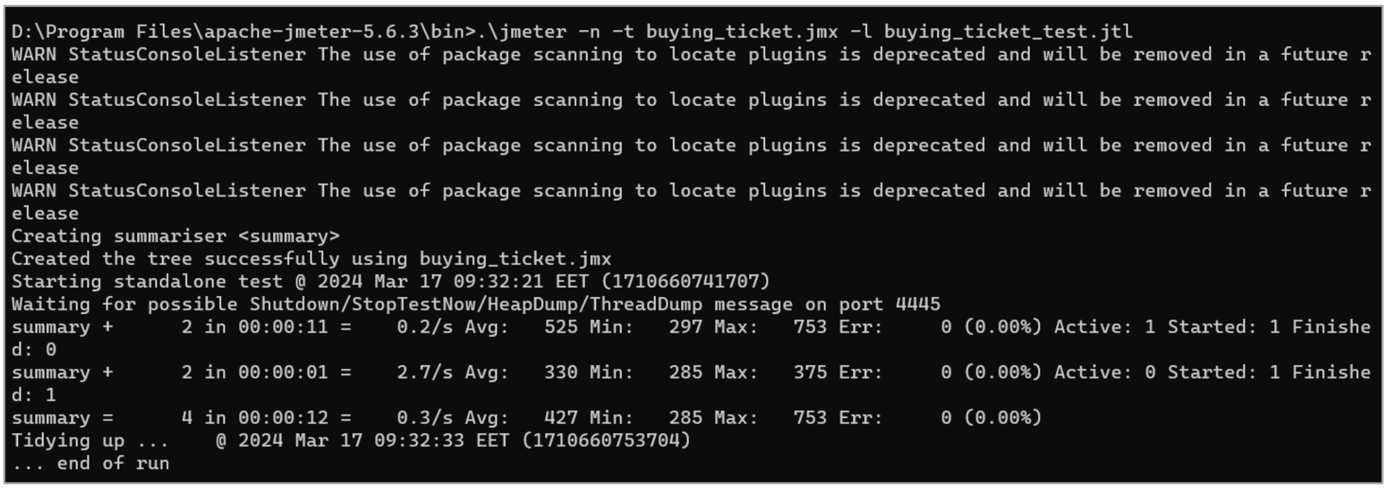
```
.\jmeter -n -t [path_to_test_plan] -l [path_to_results_file]
```

- **`-n:`** This flag tells JMeter to run in non-GUI mode
- **`-t:`** This flag is followed by the path to the test plan file (.jmx file). It tells JMeter which test plan to execute.
- **`-l:`** This flag is followed by the path where you want JMeter to output the results of the test execution (.jtl file). It stands for "**log file**".
- Replace **`[path_to_test_plan]`** with the relative or absolute path to your .jmx file and **`[path_to_results_file]`** with the desired path for your results file (.jtl) that will be created during testing.

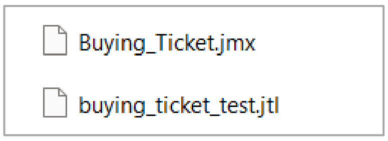
- For example, if your test plan is named test.jmx and you want to save the results in test.jtl within the same directory, you would use:

```
.\jmeter -n -t test.jmx -l test.jtl
```

- In this case our test plan is named **buying_ticket.jmx** and we want to save the results in **buying_ticket_test.jtl** both in JMeter's 'bin' directory:



```
D:\Program Files\apache-jmeter-5.6.3\bin>.\jmeter -n -t buying_ticket.jmx -l buying_ticket_test.jtl
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
Creating summariser <summary>
Created the tree successfully using buying_ticket.jmx
Starting standalone test @ 2024 Mar 17 09:32:21 EET (1710660741707)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary +      2 in 00:00:11 =    0.2/s Avg:   525 Min:   297 Max:   753 Err:     0 (0.00%) Active: 1 Started: 1 Finishe
d: 0
summary +      2 in 00:00:01 =    2.7/s Avg:   330 Min:   285 Max:   375 Err:     0 (0.00%) Active: 0 Started: 1 Finishe
d: 1
summary =      4 in 00:00:12 =    0.3/s Avg:   427 Min:   285 Max:   753 Err:     0 (0.00%)
Tidying up ...    @ 2024 Mar 17 09:32:33 EET (1710660753704)
... end of run
```



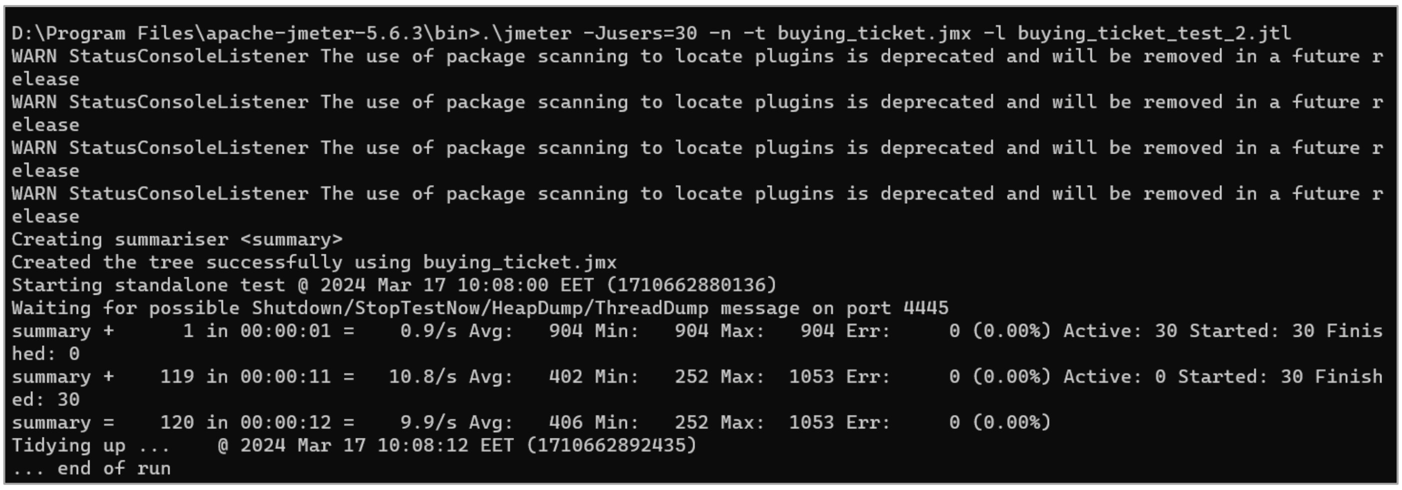
Buying_Ticket.jmx

buying_ticket_test.jtl

- If you want to run the test with a specific number of users, which you can define via the command line, you add a **`-J` parameter** to set that property:

```
.\jmeter -J[property_name]=[value] -n -t [path_to_test_plan] -l [path_to_results_file]
```

- Replace **`[property_name]`** with the name of the **property you're setting (e.g., users)** and **[value]** with the **number** you wish to set it to.
- For example, to **run the test with 30 users**:



```
D:\Program Files\apache-jmeter-5.6.3\bin>.\jmeter -Jusers=30 -n -t buying_ticket.jmx -l buying_ticket_test_2.jtl
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
Creating summariser <summary>
Created the tree successfully using buying_ticket.jmx
Starting standalone test @ 2024 Mar 17 10:08:00 EET (1710662880136)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary +      1 in 00:00:01 =    0.9/s Avg:   904 Min:   904 Max:   904 Err:     0 (0.00%) Active: 30 Started: 30 Finis
hed: 0
summary +    119 in 00:00:11 =   10.8/s Avg:   402 Min:   252 Max:  1053 Err:     0 (0.00%) Active: 0 Started: 30 Finish
ed: 30
summary =    120 in 00:00:12 =    9.9/s Avg:   406 Min:   252 Max:  1053 Err:     0 (0.00%)
Tidying up ...    @ 2024 Mar 17 10:08:12 EET (1710662892435)
... end of run
```

Note that you can't use the same name for the .jtl file as in the previous test, since it already exists.

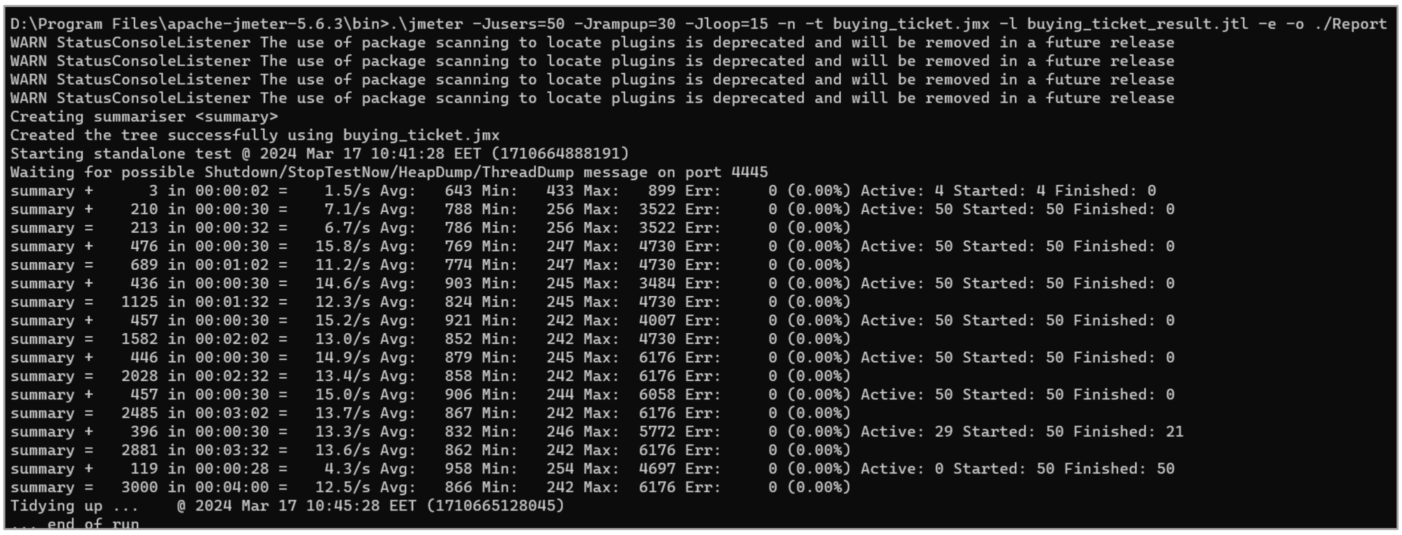
We can **extend the command further** to conduct a more comprehensive test with specific **parameters for users, ramp-up time, and loop count**, and then **generate a report**:

```
.\ jmeter -J[property_name]=[value] -J[another_property_name]=[value] -n -t
[path_to_test_plan] -l [path_to_results_file] -e -o [path_to_output_folder]
```

- **`-J[property_name]=[value]:`** Sets custom values for different test parameters. You replace [property_name] with the name of the variable you want to set (e.g., users, rampup, loop) and [value] with the desired value.
- **`-e:`** Tells JMeter to generate an HTML report at the end of the test.
- **`-o [path_to_output_folder]:`** Specifies the output folder where the HTML report will be saved. If the folder doesn't exist, JMeter will create it.

- For example: Run a test plan named **buying_ticket.jmx with 50 users**, a **ramp-up period of 30 seconds**, **15 iterations of the test loop**, **save the results in buying_ticket_results.jtl**, and **generate a report in a folder named Report** in the current directory.
- **`-Jusers=50:`** This will override the number of users specified in your test plan to 50.
- **`-Jrampup=30:`** This sets the ramp-up period to 30 seconds, spreading out the start of each of the 50 users over 30 seconds.
- **`-Jloop=15:`** This makes the test plan run 15 times for each user.
- **`-e -o ./Report:`** After the test completes, an HTML report will be generated in the ./Report directory.

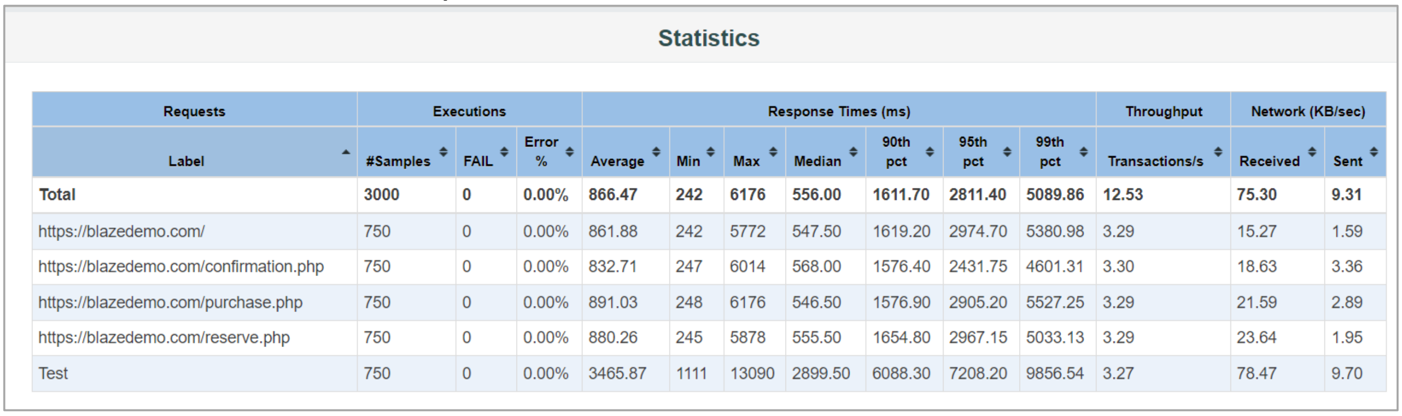
Note: Keep in mind that this test runs for about 4 minutes.



```
D:\Program Files\apache-jmeter-5.6.3\bin>.\jmeter -Jusers=50 -Jrampup=30 -Jloop=15 -n -t buying_ticket.jmx -l buying_ticket_result.jtl -e -o ./Report
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
Creating summariser <summary>
Created the tree successfully using buying_ticket.jmx
Starting standalone test @ 2024 Mar 17 10:41:28 EET (1710664888191)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary +      3 in 00:00:02 =    1.5/s Avg:   643 Min:   433 Max:   899 Err:     0 (0.00%) Active: 4 Started: 4 Finished: 0
summary +    210 in 00:00:30 =    7.1/s Avg:   788 Min:   256 Max:  3522 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    213 in 00:00:32 =    6.7/s Avg:   786 Min:   256 Max:  3522 Err:     0 (0.00%)
summary +    476 in 00:00:30 =   15.8/s Avg:   769 Min:   247 Max:  4730 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    689 in 00:01:02 =   11.2/s Avg:   774 Min:   247 Max:  4730 Err:     0 (0.00%)
summary +    436 in 00:00:30 =   14.6/s Avg:   903 Min:   245 Max:  3484 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =   1125 in 00:01:32 =   12.3/s Avg:   824 Min:   245 Max:  4730 Err:     0 (0.00%)
summary +    457 in 00:00:30 =   15.2/s Avg:   921 Min:   242 Max:  4007 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =   1582 in 00:02:02 =   13.0/s Avg:   852 Min:   242 Max:  4730 Err:     0 (0.00%)
summary +    446 in 00:00:30 =   14.9/s Avg:   879 Min:   245 Max:  6176 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =   2028 in 00:02:32 =   13.4/s Avg:   858 Min:   242 Max:  6176 Err:     0 (0.00%)
summary +    457 in 00:00:30 =   15.0/s Avg:   906 Min:   244 Max:  6058 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =   2485 in 00:03:02 =   13.7/s Avg:   867 Min:   242 Max:  6176 Err:     0 (0.00%)
summary +    396 in 00:00:30 =   13.3/s Avg:   832 Min:   246 Max:  5772 Err:     0 (0.00%) Active: 29 Started: 50 Finished: 21
summary =   2881 in 00:03:32 =   13.6/s Avg:   862 Min:   242 Max:  6176 Err:     0 (0.00%)
summary +    119 in 00:00:28 =    4.3/s Avg:   958 Min:   254 Max:  4697 Err:     0 (0.00%) Active: 0 Started: 50 Finished: 50
summary =   3000 in 00:04:00 =   12.5/s Avg:   866 Min:   242 Max:  6176 Err:     0 (0.00%)
Tidying up ...    @ 2024 Mar 17 10:45:28 EET (1710665128045)
... end of run
```

10. HTML Report

Now let's take a look at the **HTML Report** that JMeter created.



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3000**	**0**	**0.00%**	**866.47**	**242**	**6176**	**556.00**	**1611.70**	**2811.40**	**5089.86**	**12.53**	**75.30**	**9.31**
https://blazedemo.com/	750	0	0.00%	861.88	242	5772	547.50	1619.20	2974.70	5380.98	3.29	15.27	1.59
https://blazedemo.com/confirmation.php	750	0	0.00%	832.71	247	6014	568.00	1576.40	2431.75	4601.31	3.30	18.63	3.36
https://blazedemo.com/purchase.php	750	0	0.00%	891.03	248	6176	546.50	1576.90	2905.20	5527.25	3.29	21.59	2.89
https://blazedemo.com/reserve.php	750	0	0.00%	880.26	245	5878	555.50	1654.80	2967.15	5033.13	3.29	23.64	1.95
Test	750	0	0.00%	3465.87	1111	13090	2899.50	6088.30	7208.20	9856.54	3.27	78.47	9.70

The HTML report generated by JMeter provides a **comprehensive and visual representation of the test results**. Here's what you can typically find in a JMeter HTML report:

- **Dashboard:** An overview of the test including metrics like throughput, the number of requests, and response times. It's designed to give you a quick understanding of the performance test results at a glance.
- **Statistics:** Detailed tables that include metrics for each type of request, such as the average, median, 90th percentile, min, max response times, error percentage, and more. This helps you see how each request within your test plan performs.
- **Charts and Graphs:** Visual representations of key data points such as response times over time, throughput over time, and a distribution of response times. These can be interactive, allowing you to zoom in on specific time intervals or filter by request type.

- **Errors:** Information about any errors that occurred during the test. This section will show you the error count and the types of errors that were encountered.
- **Export Feature:** You can typically export the data from the report into CSV format for further analysis or sharing with your team.
- **Request and Response Details:** Some reports include samples of request and response data, which can be useful for debugging issues.
- **Configuration:** A snapshot of the test configuration, which includes details about the test environment and setup. This is important for replicating the test in the future or for auditing purposes.
- **Top Statistics:** This section summarizes the main statistics like total requests, average response time, min and max response time, and errors in a clear and concise manner.