

# Performance Testing with JMeter

Part I

# Agenda

## Part I:

- Introduction
- JMeter Download and Installation
- Recording Test Script
- Test Plan, Thread Groups, Samplers, Listeners
- Load Testing Example Scenario
- Analyzing the Results
- Approach for Load Testing
- Online Store Example Scenario
- Best Practices

# Agenda

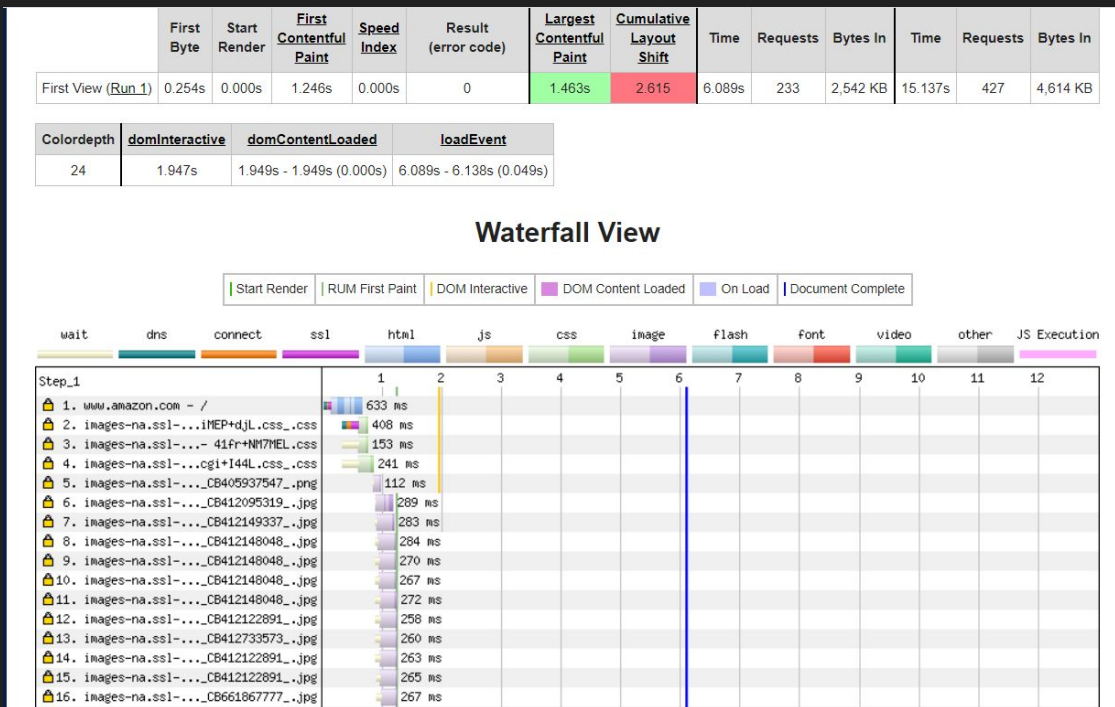
## Part II:

- Data Driven Testing with CSV Data Set Config Element
- Pre/Post Processors
- Assertions
- Pacing
- Database Performance Testing
- JMeter CLI mode
- Best Practices

# Performance Testing

- How well the system performs in terms of responsiveness and stability under workload
  - **Responsiveness:** How fast an application responds
  - **Stability:** How consistent or reliable the application is
- **Load Testing:** The aim of the load testing is to analyze the performance of the application under expected conditions. We can start load testing with 80% of the expected number of users, then slowly climb up to 100%.
- **Stress Testing:** The aim of the stress testing is to find out at what point the application breaks.

# Simple Performance Test



<https://webpagetest.org/>

# Performance Testing Tools

- WebLoad
- SilkPerformer
- LoadRunner (HP Performance Tester)
- LoadView
- NeoLoad
- Rational Performance Tester
- SmartMeter.io
- Apache JMeter

# JMeter

- Open source tool from Apache for performance testing
- Java based desktop application
- Supports many types of protocols/services like HTTP, Web Service, LDAP, JDBC, Java, FTP etc.
- Supports distributed testing
- Allows to record the user activity on the browser
- Simulates the activities with different number of users

# Best Practices

- Java 8 or above required for JMeter
- Always use latest version of JMeter
- Do not practice on public websites, use demo websites created for performance testing with less number of users. Some demo websites:
  - <https://blazedemo.com>
  - <https://www.demoblaze.com>
  - <https://orangehrm-demo-6x.orangehrmlive.com>
  - <https://computer-database.gatling.io/computers>
  - <https://petstore.octoperf.com/actions/Catalog.action>
- Use correct number of users based on your test scenario, hardware capabilities and how fast the application server is.



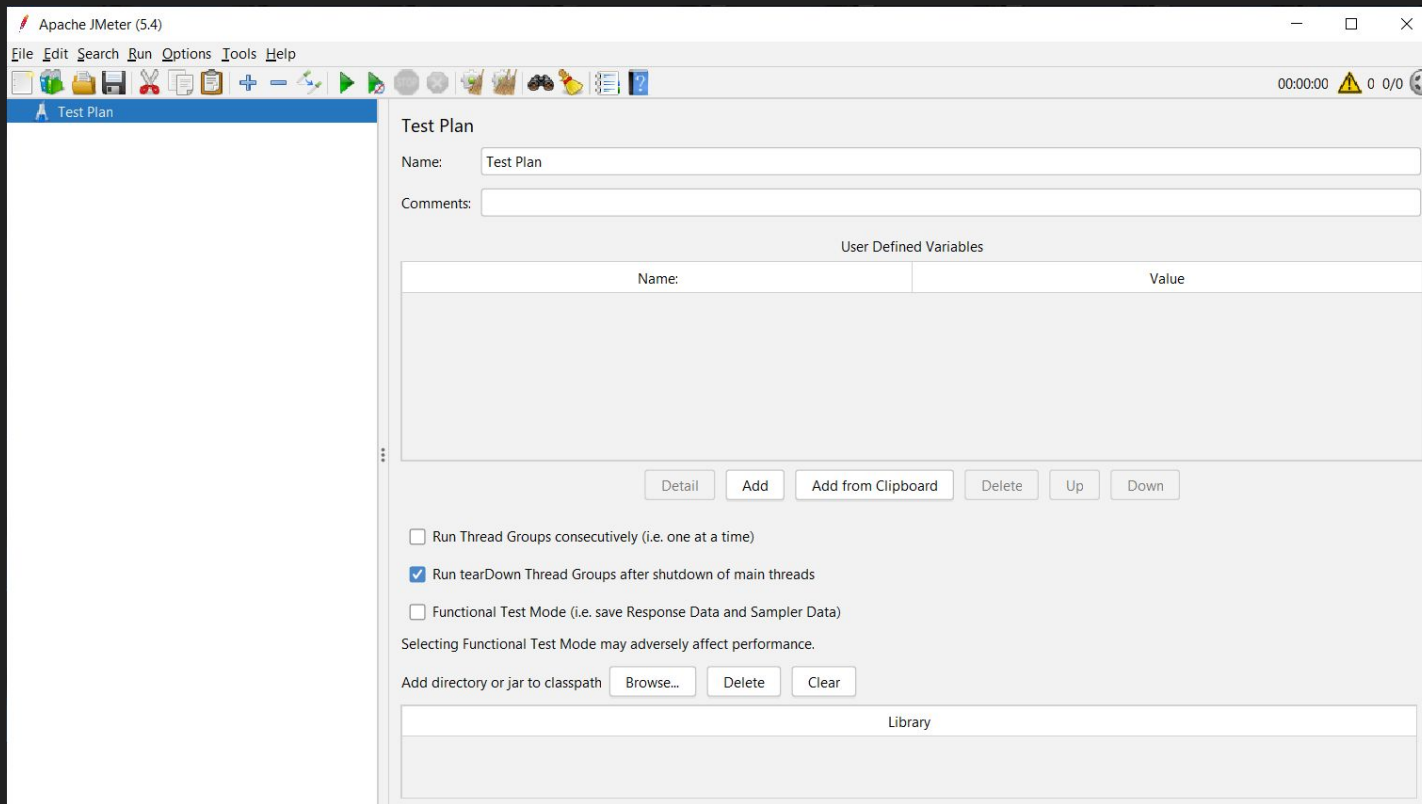
# Downloading JMeter for Windows

1. Download the binary zip file from the following link:  
[https://jmeter.apache.org/download\\_jmeter](https://jmeter.apache.org/download_jmeter)
2. Extract the zip file and browse the JMeter bin directory.
3. Find the `jmeter.bat` file and create a shortcut onto your desktop.
4. JMeter can now be run from the shortcut.

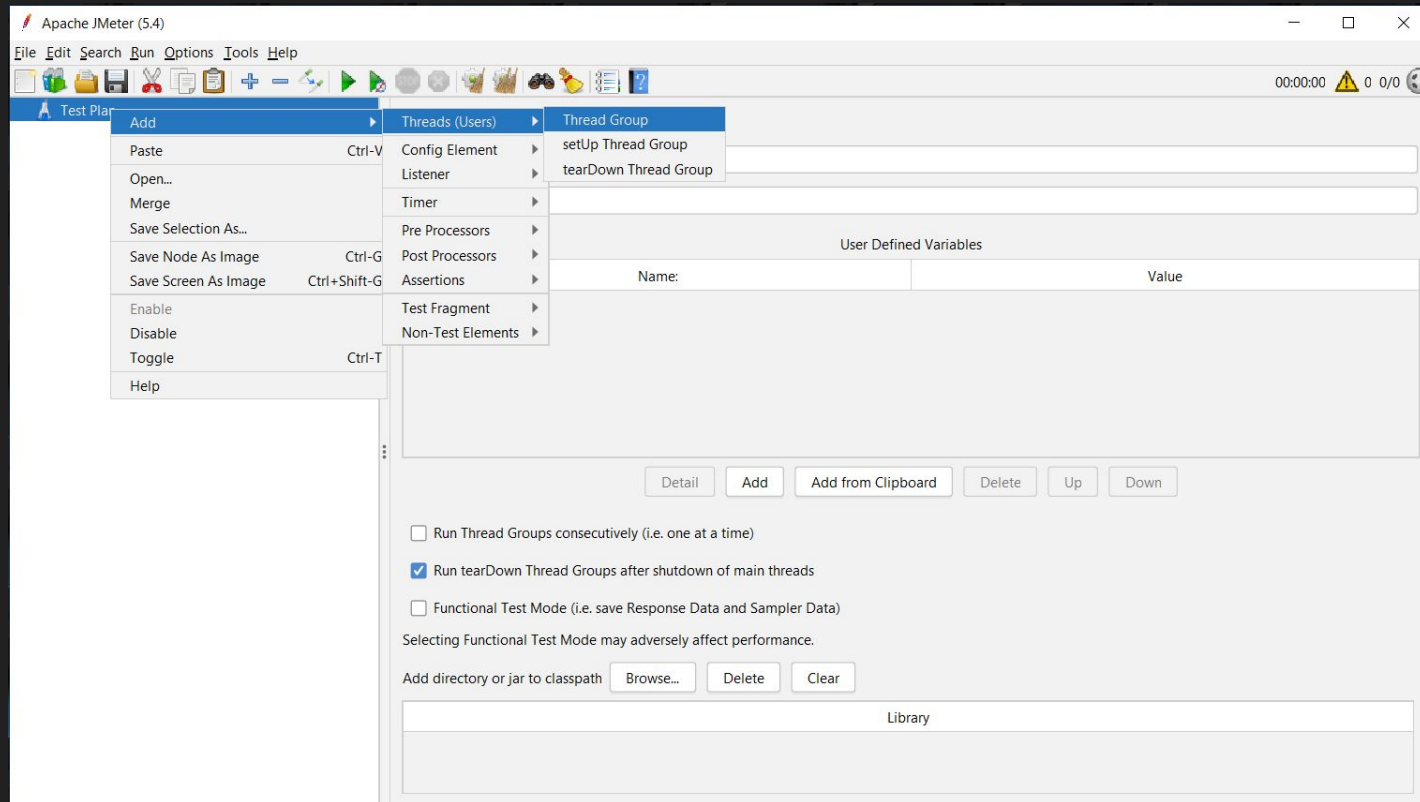
# Downloading JMeter for Mac OS

1. Use homebrew and run the command: `brew install jmeter`
2. For installing with extra plugin set: `brew install jmeter --with-plugins`
3. Run `jmeter -?` for help
4. Type `jmeter` in terminal to launch it. ↵

# Launching JMeter



# Threads (Users)



# Thread Group

- Each thread is a user and we can set the Number of Threads.
- Ramp-up period: 10 threads in 10 seconds means each second a user is added and at the end of the 10 second, we will have 10 users.
- Loop Count: The number of times the test case will iterate for that user.
- Thread life time: The duration that we want the thread to be active.

Thread Properties	
Number of Threads (users):	10
Ramp-up period (seconds):	10
Loop Count:	<input type="checkbox"/> Infinite 2
<input checked="" type="checkbox"/> Same user on each iteration	
<input type="checkbox"/> Delay Thread creation until needed	
<input checked="" type="checkbox"/> Specify Thread lifetime	
Duration (seconds):	20
Startup delay (seconds):	1

# Test Plan

- Test Plan consists of all components and actions to execute the performance test scenario.
- A Test Plan might have multiple thread (user) groups.
- If “Run thread group consecutively” is selected, it means if we have a number of thread groups, only 1 thread group will be executed at a time.
- If “Functional Test Mode” is selected, JMeter will record response of each request sent to server. This option is used in functional testing but degrades performance.
- Global variables such as a URL, a key etc. can be created and these variables can be used throughout the testing.

# Recording Test Script

There are different ways to record a script:

1. JMeter Test Script Recorder
2. Chrome browser BlazeMeter extension

<https://guide.blazemeter.com/hc/en-us/articles/206732579-The-BlazeMeter-Chrome-Extension-Record-JMeter-Selenium-or-Synchronized-JMeter-and-Selenium>

3. Using the chrome dev tools and converting the HTTP Archive (.har) files to JMeter Test Plans (.jmx).

<https://www.flood.io/blog/convert-har-files-to-jmeter-test-plans>

# Record and Playback

- JMeter records the test script based on our actions.
- Recorded script is a starting point to develop test plans.
- We can edit or run the recorded test script.
- In order to record a test script:
  1. Add a HTTPS Test Script Recorder element to the Test Plan
  2. Set the port number in recorder or keep the default (Port: 8888)
  3. Configure the browser proxy with the same port number
  4. Import the JMeter certificate to the browser



# Configuring Browser for JMeter Proxy

Example: Firefox browser

1. Start the Firefox browser.
2. From the toolbar, click Options>Network Settings
3. Check “Manual proxy configuration”
4. Enter for Address: localhost, port:8888
5. Check “Also use this proxy for FTP and HTTPS”
6. Https requires certificate and JMeter provides the certificate for connection. Import the certificate from JMeter bin folder to the browser.

For the other browsers, find the proxy settings and “localhost” for address and “8888” for port number.

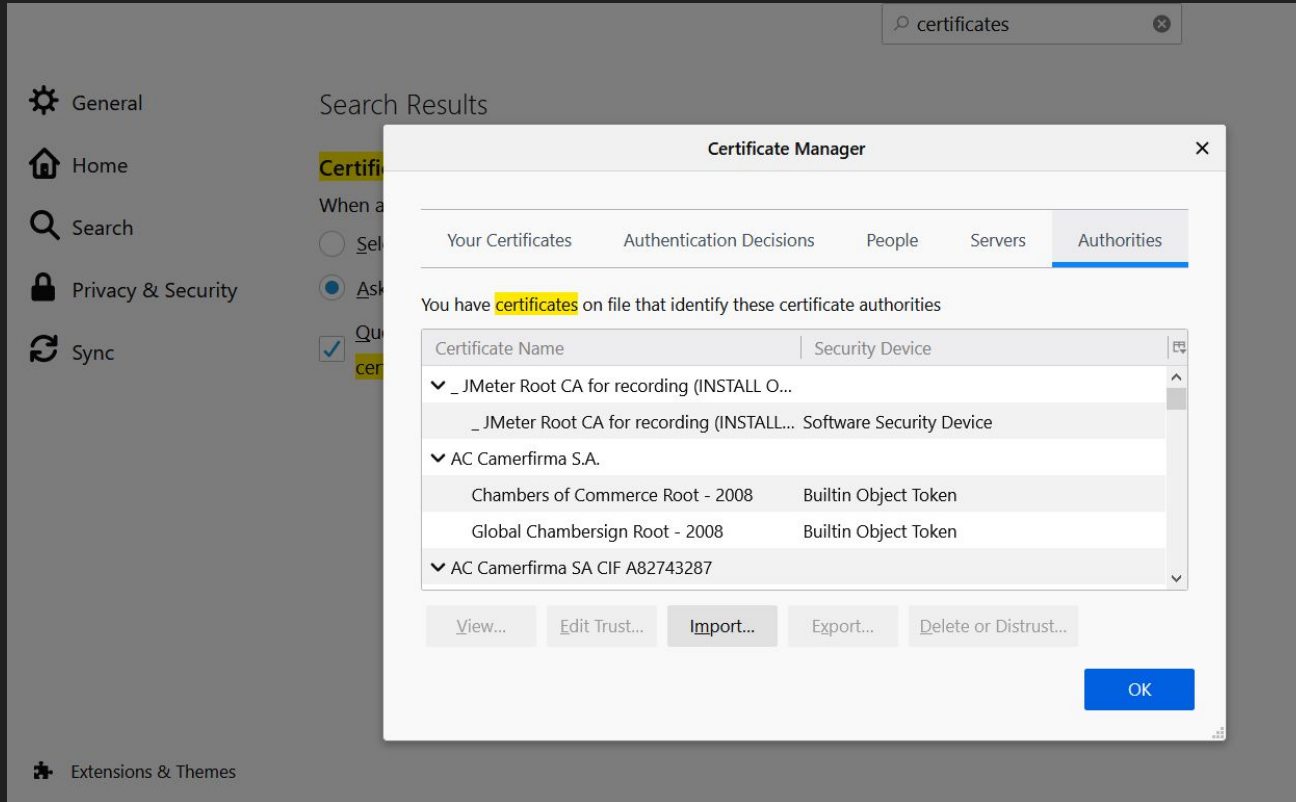
# Configuring Browser for JMeter Proxy

The image shows the 'Connection Settings' dialog box in a web browser. On the left is a sidebar with navigation links: 'General' (selected), 'Home', 'Search', 'Privacy & Security', 'Sync', 'Extensions & Themes', and 'Firefox Support'. The main panel is titled 'Connection Settings' and contains the following sections:

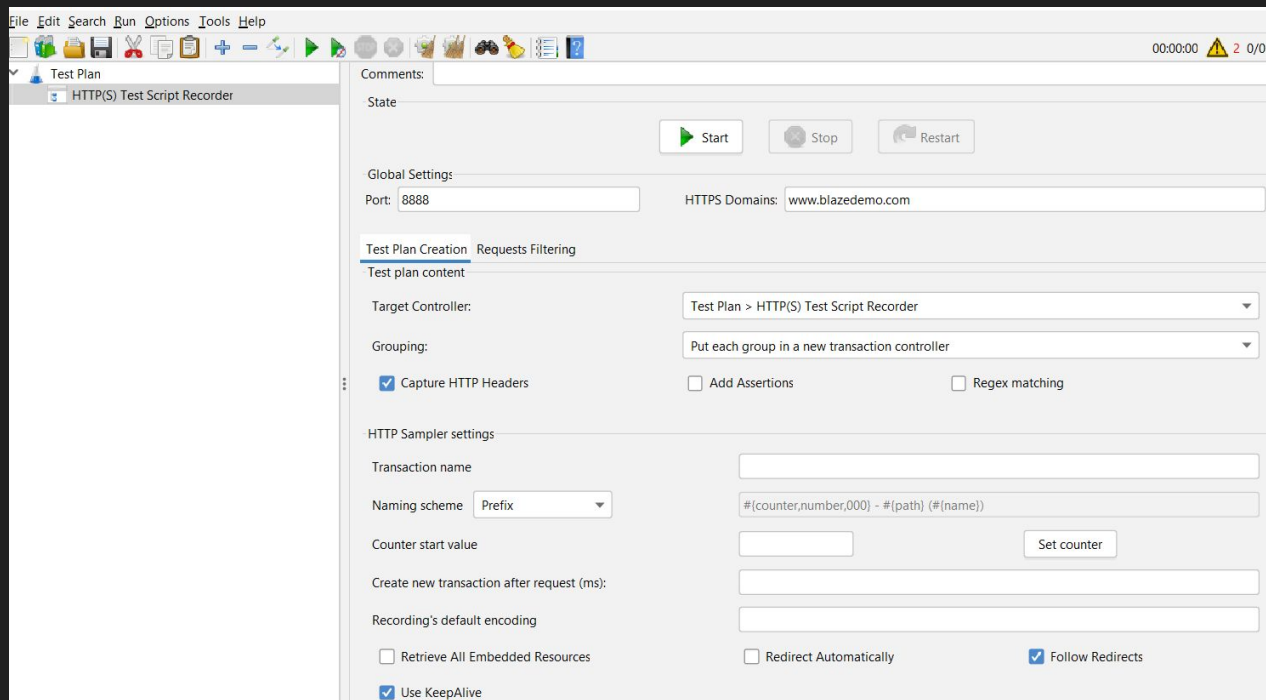
- Configure Proxy Access to the Internet**
  - ☐ No proxy
  - ☐ Auto-detect proxy settings for this network
  - ☐ Use system proxy settings
  - ☒ Manual proxy configuration
- Manual proxy configuration fields:**
  - HTTP Proxy:  Port: 
    - ☒ Also use this proxy for FTP and HTTPS
  - HTTPS Proxy:  Port:
  - FTP Proxy:  Port:
  - SOCKS Host:  Port: 
    - ☐ SOCKS v4
    - ☒ SOCKS v5
  - ☐ Automatic proxy configuration URL:
- No proxy for:**

At the bottom right are three buttons: 'OK' (blue), 'Cancel', and 'Help'.

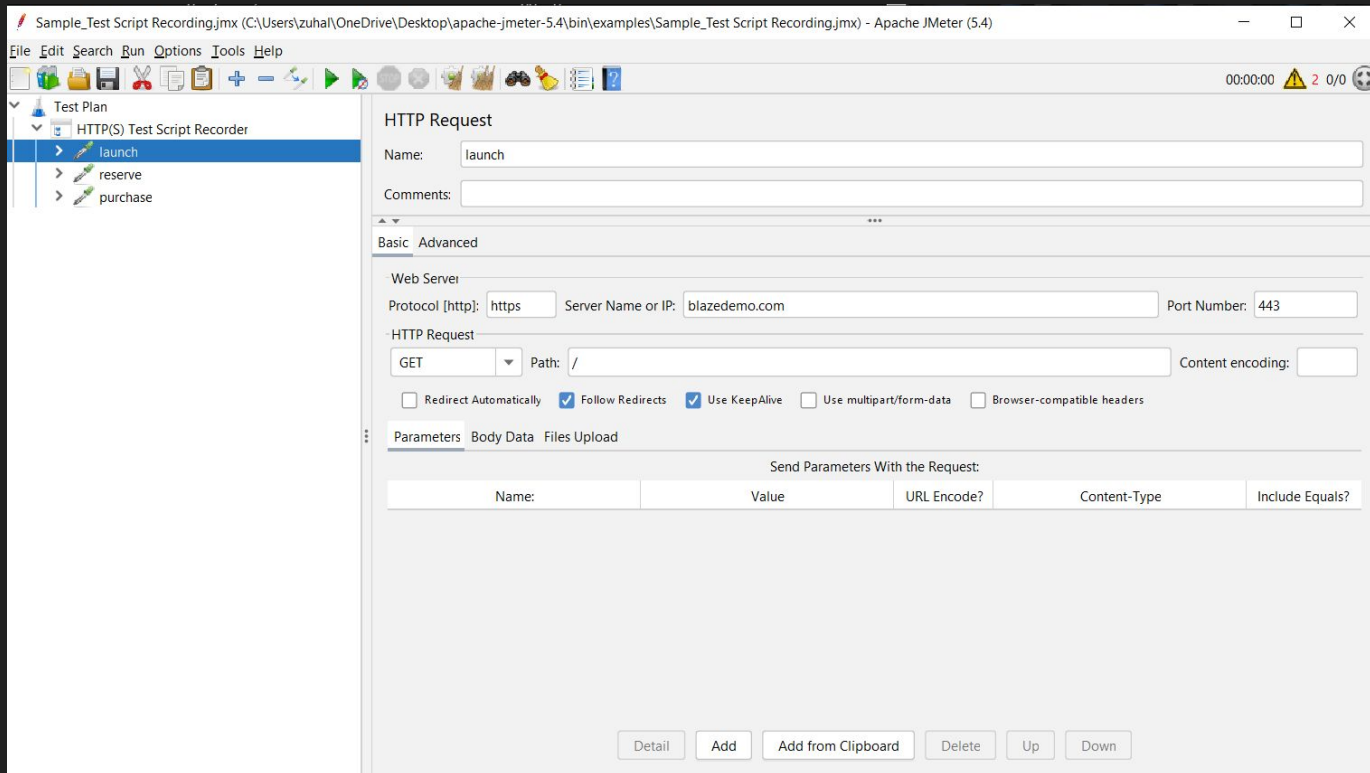
# Adding JMeter Certificate



# HTTPS Test Script Recorder

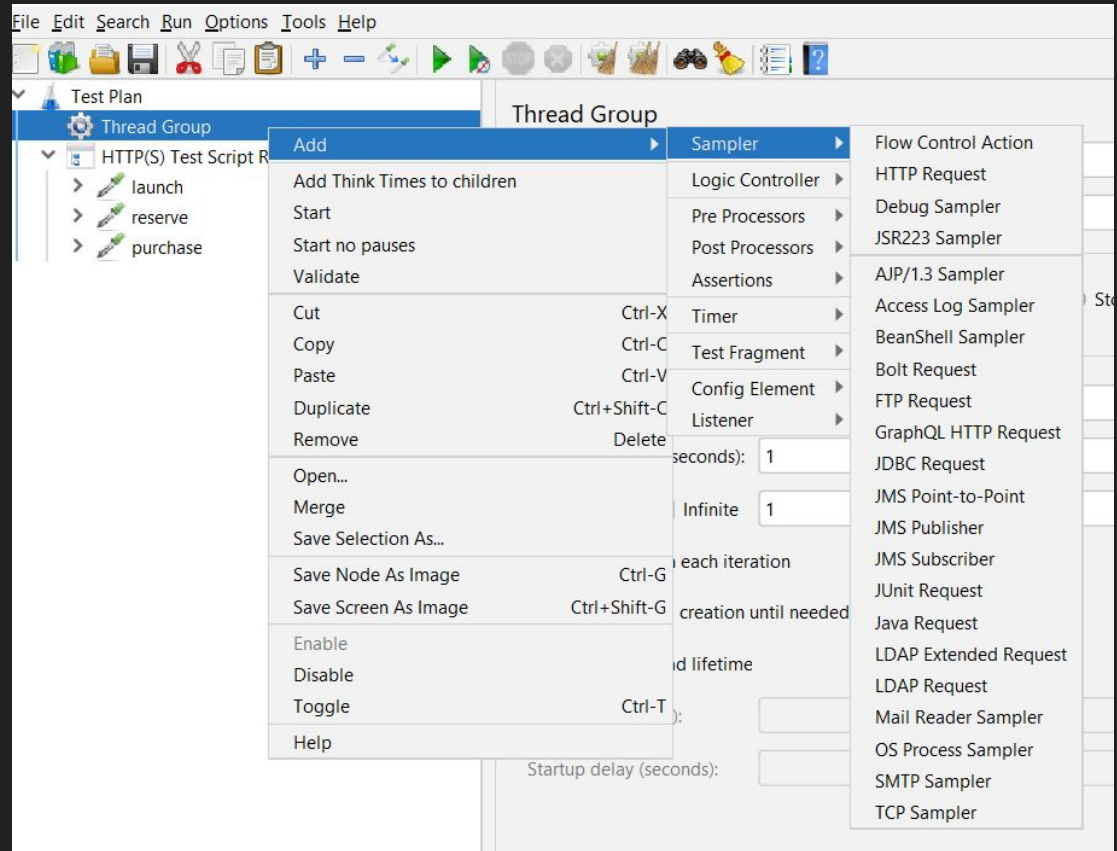


# Sample Scenario - Recording and Playback



# Samplers

- Samplers are used to send requests to servers.
- HTTP Request is used to send requests like GET, POST, PUT, PATCH, DELETE etc.



# HTTP Header Manager

- The HTTP Header Manager allows to customize what information JMeter sends in the HTTP request header. By default it is added while recording a script.
- The HTTP Header Manager, should be added at the Thread Group level, unless for some reason different headers for the different HTTP Request objects exists in the test.

HTTP Header Manager

Name:

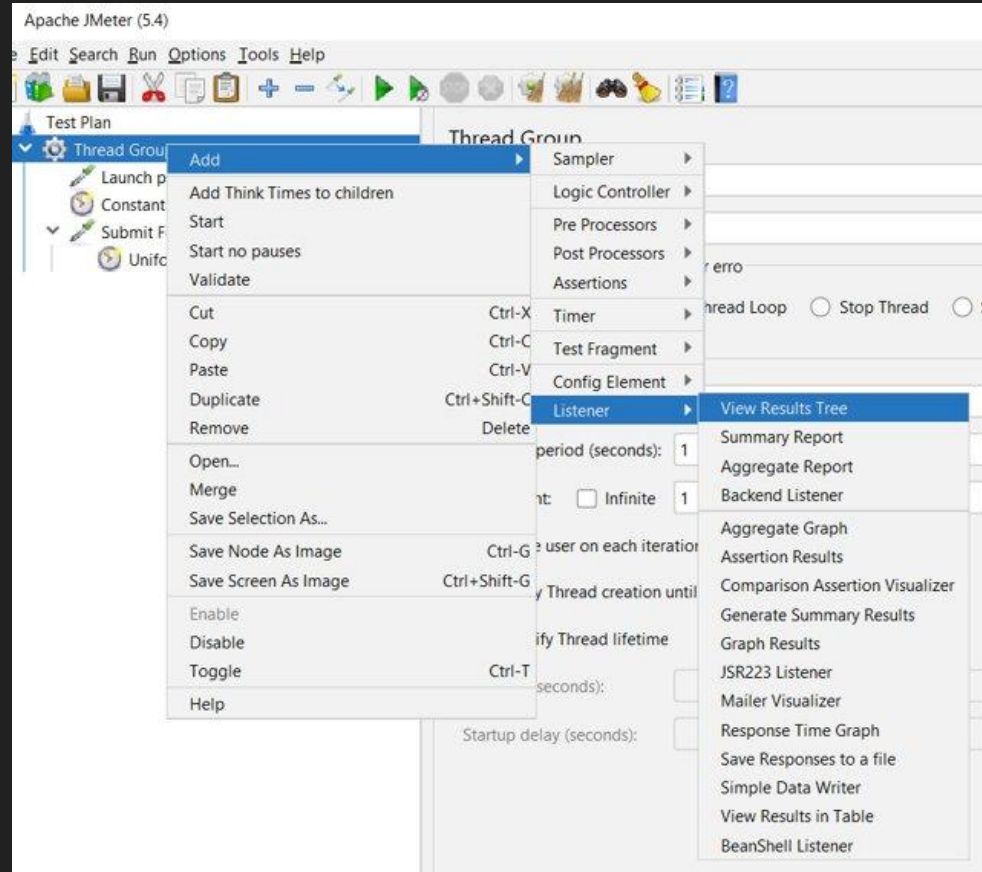
Comments:

Headers Stored in the Header Manager

Name:	Value
Accept-Language	en-US,en;q=0.5
Upgrade-Insecure-Requests	1
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

# Listeners

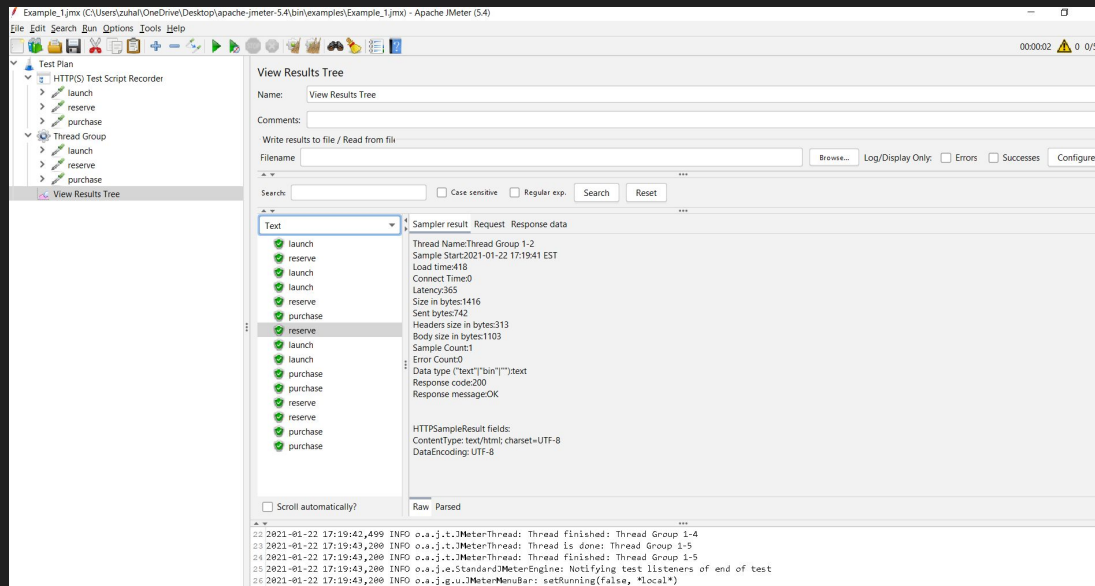
- Listeners are used to display the test results.
- The results can be shown as a tree, tables, graphs or written to a log file.
- Listeners all write the same raw data to the output file - if one is specified.
- Adding more listeners to a test scenario decreases the performance so add listeners as needed.





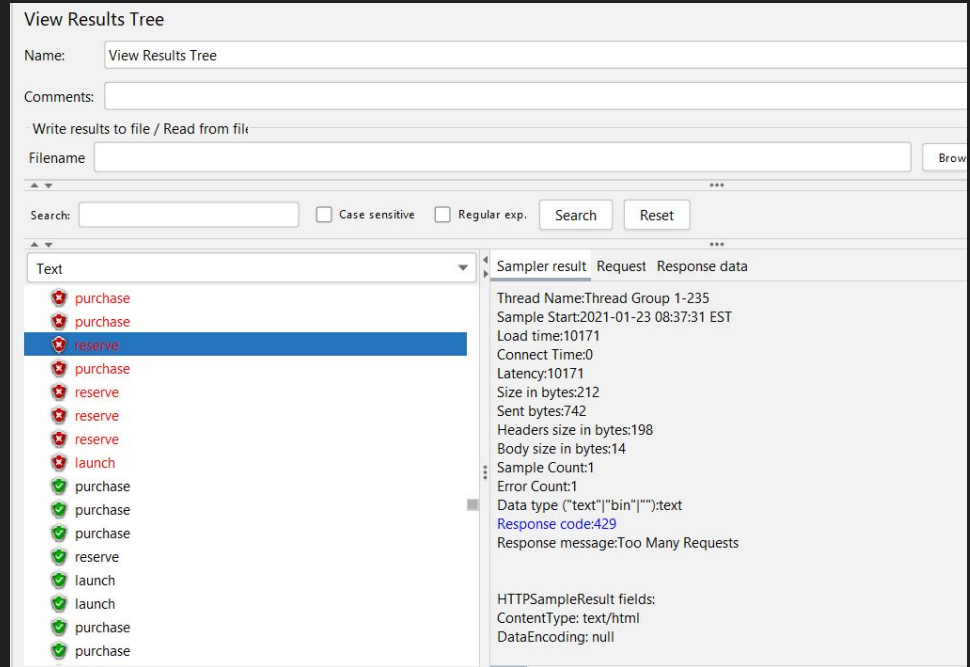
# Sample Scenario – Load Test

1. Add a Thread Group under the Test Plan.
2. Copy and paste the samplers created in the Recording Scenario.
3. Add View Results Tree listener under the Thread Group.
4. Set the number of users in the Thread Group.
5. Run the scenario.



# Analyzing the Test Results – View Results Tree

- **Latency:** Measures the latency from just before sending the request to just after the first chunk of the response has been received,
- **Connect Time:** Measures the time it took to establish the connection.
- **Connect Time / Latency** should be as low as possible, ideally less than 1 second.



# Analyzing the Test Results – Aggregate Report

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/s...	Sent KB/sec
launch	200	18508	8041	37935	38639	70324	458	71809	0.00%	2.8/sec	3.85	1.50
reserve	200	5596	276	33943	36027	38499	159	38729	0.00%	2.7/sec	4.07	1.99
purchase	200	3811	233	9251	27446	35764	159	70086	0.00%	2.7/sec	4.59	2.12
TOTAL	600	9305	653	35635	37807	53475	159	71809	0.00%	8.0/sec	12.17	5.46

- **# Samples:** No of users hit that specific request
- **Average:** Average time taken by all the samples to execute a specific request in milliseconds
- **Min:** The shortest time taken by a sample for a specific request
- **Max:** The longest time taken by a sample for a specific request
- **Median:** Time in the middle of a set of samples result. It tells us that 50% of the samples took no more than this time and the remainder took at least as long.
- **90% Line (90th percentile) :** 90% of the samples took no more than this time. The remaining samples took at least as long as this.

# Analyzing the Test Results – Summary Report

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
launch	200	2777	265	14518	2754.81	0.50%	13.2/sec	18.58	7.17	1440.3
reserve	200	1983	160	13924	2587.02	0.00%	10.4/sec	15.38	7.51	1519.3
purchase	200	1228	157	13565	2314.19	0.00%	9.8/sec	16.61	7.63	1741.4
TOTAL	600	1996	157	14518	2635.46	0.17%	28.4/sec	43.45	19.39	1567.0

- **Error%:** Percentage of failed requests per request
- **Std. Dev.:** Represents the exceptional cases which were deviating from the average value of sample response time. The lesser this value more consistent the data. Standard deviation should be less than or equal to half of the average time for a request.
- **Throughput:** Number of requests that are processed per second by the server. Larger throughput is better.

# Best Practices - Listeners

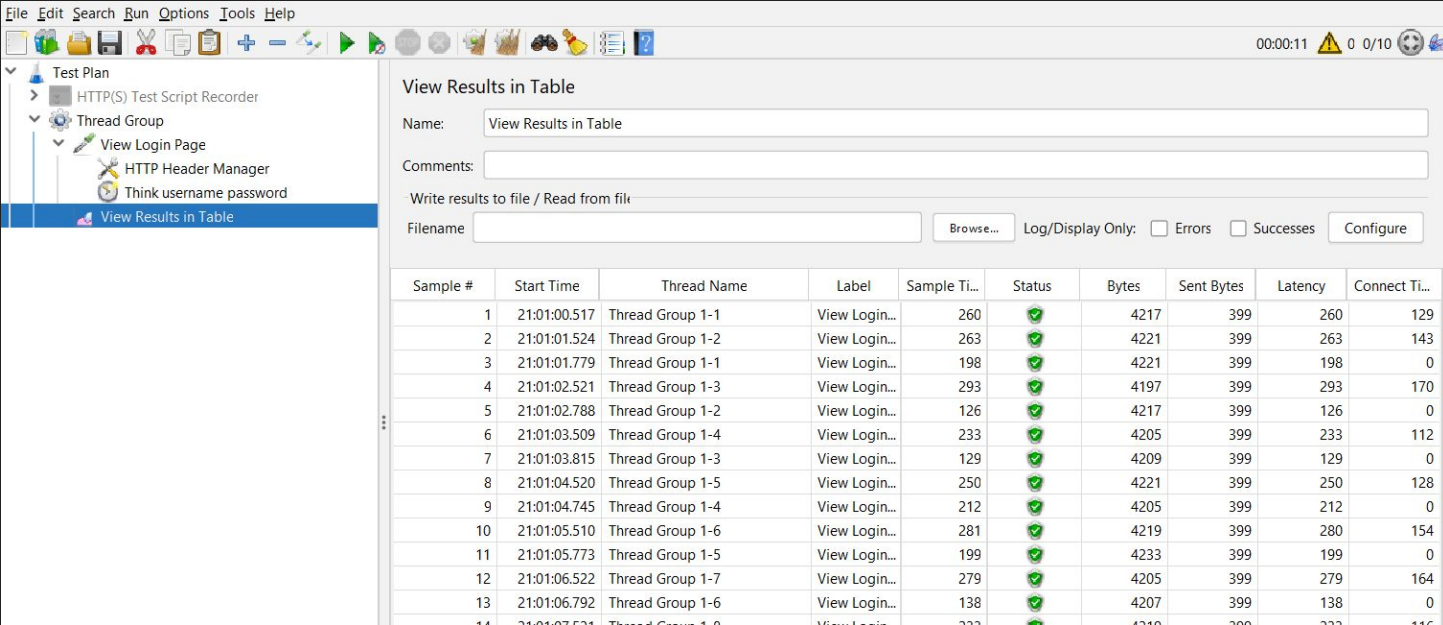
- UI listeners such as View Results Tree, Aggregate Graph and Report are preferred for debugging / testing purpose.
- UI listeners keep the results in JMeter memory which is limited.
- Which listeners to use for real load tests?
  - Headless listeners such as Simple Data Writer and Backend Listener are designed to work when JMeter is run from the command line.
  - These listeners use less memory than UI listeners.

# Think Time and Timers

- In order to realistically simulate the user behavior, we need to add Think Time.
- Timers allow us to add think time between the transactions within an iteration.
- Timers are used to insert delay in execution.
- An average think time for a user is 3-10 seconds.
- We can use timers such as Constant Time or Uniform Random timer to add think time.
- Timer execution time is not added to the sampler time in reports.

# Constant Timer

Adds a fixed delay for a transaction.

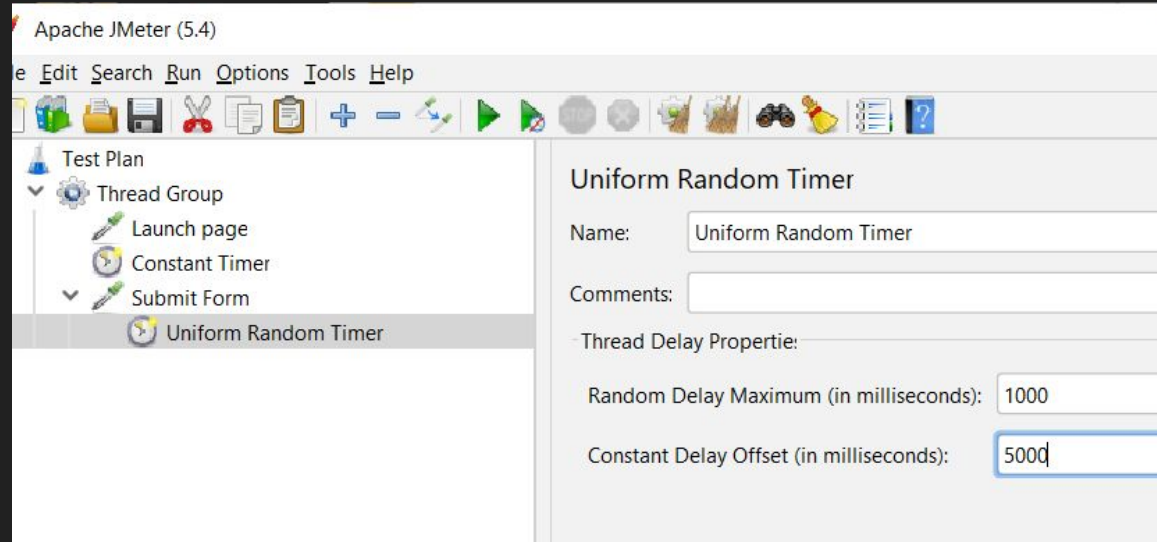


The screenshot shows the JMeter GUI. On the left, the 'Test Plan' tree is expanded, showing 'Thread Group' > 'View Login Page' > 'Think username password' > 'View Results in Table' (selected). The main area is titled 'View Results in Table'. It has a 'Name' field set to 'View Results in Table', an empty 'Comments' field, and a section for 'Write results to file / Read from file' with a 'Filename' field and a 'Browse...' button. There are also checkboxes for 'Log/Display Only', 'Errors', and 'Successes', and a 'Configure' button. Below this is a table of test results.

Sample #	Start Time	Thread Name	Label	Sample Ti...	Status	Bytes	Sent Bytes	Latency	Connect Ti...
1	21:01:00.517	Thread Group 1-1	View Login...	260	✓	4217	399	260	129
2	21:01:01.524	Thread Group 1-2	View Login...	263	✓	4221	399	263	143
3	21:01:01.779	Thread Group 1-1	View Login...	198	✓	4221	399	198	0
4	21:01:02.521	Thread Group 1-3	View Login...	293	✓	4197	399	293	170
5	21:01:02.788	Thread Group 1-2	View Login...	126	✓	4217	399	126	0
6	21:01:03.509	Thread Group 1-4	View Login...	233	✓	4205	399	233	112
7	21:01:03.815	Thread Group 1-3	View Login...	129	✓	4209	399	129	0
8	21:01:04.520	Thread Group 1-5	View Login...	250	✓	4221	399	250	128
9	21:01:04.745	Thread Group 1-4	View Login...	212	✓	4205	399	212	0
10	21:01:05.510	Thread Group 1-6	View Login...	281	✓	4219	399	280	154
11	21:01:05.773	Thread Group 1-5	View Login...	199	✓	4233	399	199	0
12	21:01:06.522	Thread Group 1-7	View Login...	279	✓	4205	399	279	164
13	21:01:06.792	Thread Group 1-6	View Login...	138	✓	4207	399	138	0
14	21:01:07.531	Thread Group 1-8	View Login...	222	✓	4210	399	222	116

# Uniform Random Timer

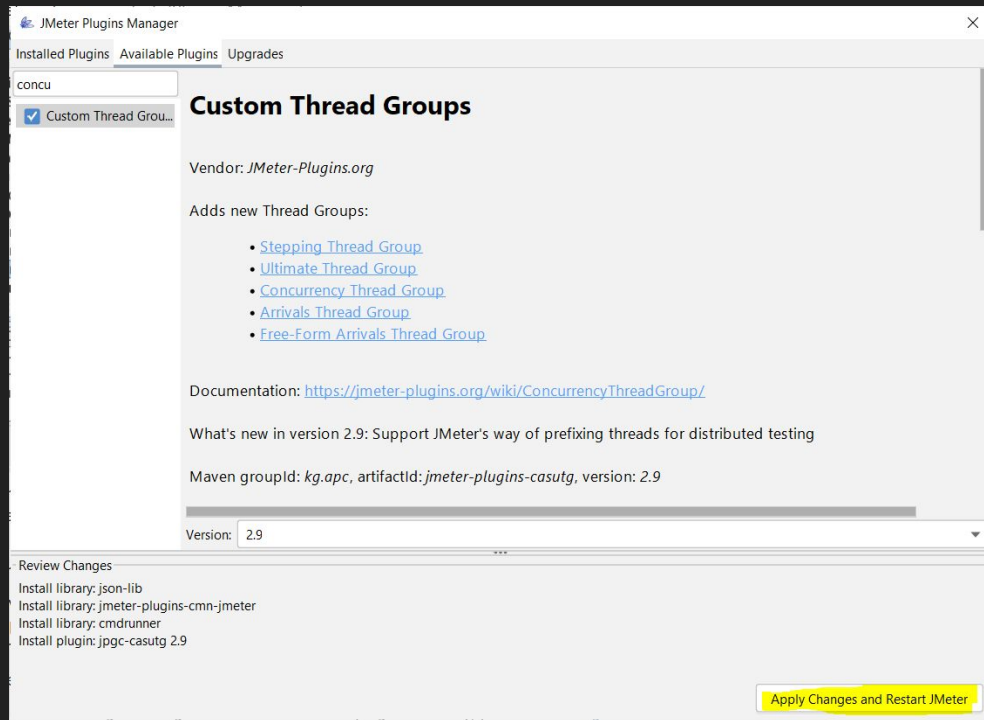
Adds a random time in addition to constant delay time. If delay is 1 sec and constant delay is 5 then the delay will be between 5 and 6 seconds.





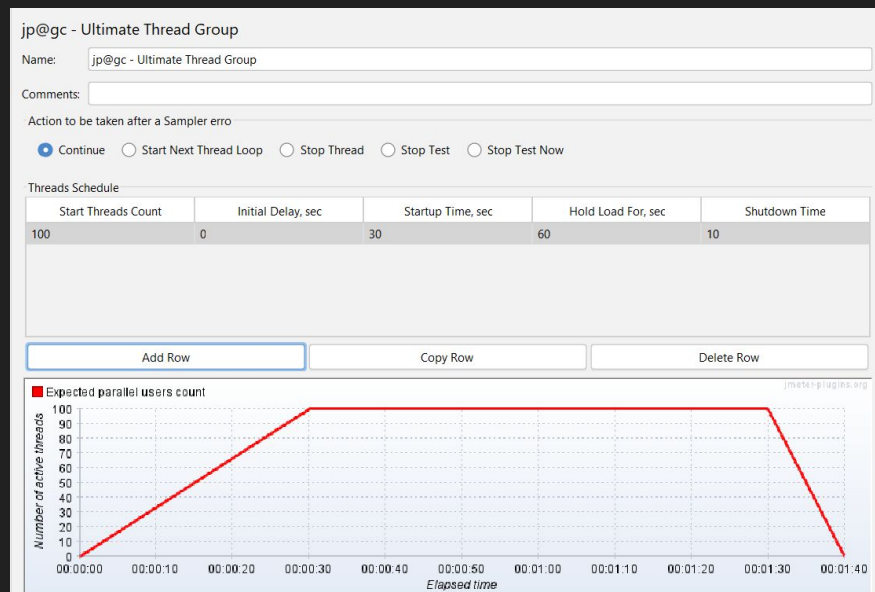
# JMeter Plugins Manager

- Download the Plugins Manager JAR file and put it into JMeter's lib/ext directory.
- Then start JMeter and go to "Options" menu to access the Plugins Manager.



# Specialty Thread Groups

- **Concurrency Thread Group:** We can specify the target concurrency, ramp-up time and ramp up steps so that we can see at which point (number of users) the scenario fails.
- **Ultimate Thread Group:** We can add different groups of users with different profiles (initial delay, startup time, hold load, shutdown time). Ultimate Thread Group can be used to perform Spike Testing.
- **setUp and teardown Thread Group:** To specify the different types of users, we use these thread groups. For example we can set an admin user group to delete all the logins in a teardown Thread Group.



# Load Testing Environment

- A dedicated load testing environment similar to production environment is utilized.

## Example:

- Production Environment: 3 Application servers, 2 Web servers, and 2 Database Servers.
- QA: 1 Application Server, 1 Web server, and 1 Database server.
- When the load testing is conducted on the QA environment which is not equal to the Production, then the tests needs to be adjusted.

# Approach for Load Testing

1. Get the load requirements and the user access patterns from the server logs.
2. Identify the Load test Acceptance Criteria

## For Example:

- The response time of the Login page shouldn't be more than 4 sec even during the max load conditions.
- CPU utilization should not be more than 85%.
- The throughput of the system should be 100 transactions per sec.

# Approach for Load Testing

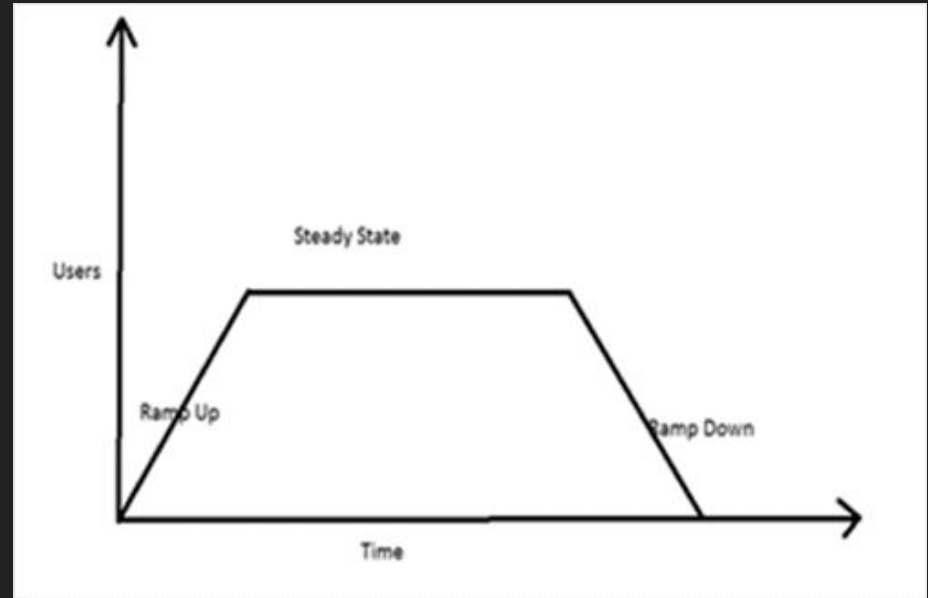
3. Identify the Business scenarios that needs to be tested based on the main business flow. If there is an existing application, user access patterns can be obtained from server logs.

4. Design the workload: Modelling the workload in such a way which mimics the actual user navigation in production or expected.

While designing the workload, consider the **think time** that the user navigates across the application in a realistic pattern.

# Workload Pattern

- The Workload Pattern will usually contains a Ramp up, Ramp down and a steady state.



## Example Workload Model: Online shopping application

Possible business flows:

1. **Browse:** The user launches the application, logs in, browses and logs out without purchasing
2. **Browse, Product View and Add to Cart:** The user logs in, browses, views product details, adds the product to cart and logs out.
3. **Browse, Product View, Add to Cart and Check out:** The user logs in, browses, views product, adds the product to the cart, checks out and logs out.
4. **Browse, Product view, Add to cart Check out and Makes Payment:** The user logs in, browses, views product, adds the product to the cart, checks out, makes payment and logs out.

# Metrics to be collected

Business Flow #	Number of Transactions	Virtual User Load	Response Time (sec)	% Failure rate allowed
1	20	1500	5	Less than 1%
2	20	150	5	Less than 1%
3	16	100	3	Less than 1%
4	18	60	3	Less than 1%



# Approach for Load Testing

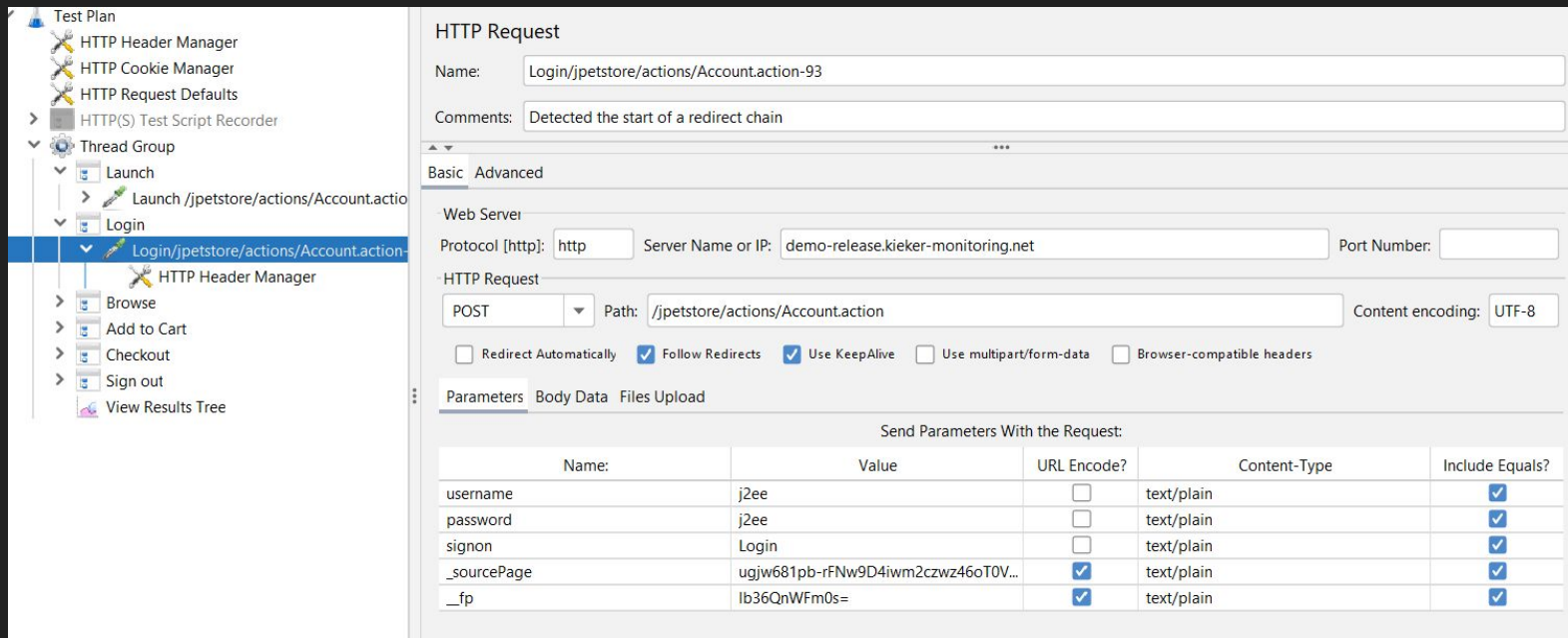
- 4. Execute Load Test:** Make sure that the environment is up and running. The application is functionally tested and is stable. Then start running the tests with the workload pattern.
- 5. Analyze the Test Results:** Have a baseline test to always compare with the other test runs. Gather the metrics and server logs after the test run to find the bottlenecks. Check for memory leaks, high CPU usage, unusual server behavior and any errors. Fix what needs fixing and run the tests again.
- 6. Reporting:** Gather all the metrics after the tests completed and share with the client.

# Sample Scenario – Pet Store Application

1. Record the Test Script Recorder to record the scenario.
2. Select "Put each group in a new transaction controller" option on the HTTP(S) Test Script Recorder.
3. Record the script using the recorder. Name each transaction.
4. Add a Thread Group and copy the samplers from the recorder to the Thread Group.
5. Check the transaction controllers and the POST request for login.
6. Add a View Results Tree.
7. Run the plan and analyze the responses for the requests from the View Results Tree.

URL: <http://demo-release.kieker-monitoring.net/jpetstore/actions/Account.action?signonForm=>

# Sample Scenario – Pet Store Application



The screenshot displays the JMeter GUI. On the left, the Test Plan tree shows a Thread Group with a 'Login' step selected. The main panel shows the 'HTTP Request' configuration for the selected step. The 'Name' is 'Login/jpetstore/actions/Account.action-93' and the 'Comments' are 'Detected the start of a redirect chain'. The 'Basic' tab is active, showing the 'Web Server' section with 'Protocol' set to 'http', 'Server Name or IP' set to 'demo-release.kieker-monitoring.net', and 'Port Number' empty. The 'HTTP Request' section shows 'Method' set to 'POST' and 'Path' set to '/jpetstore/actions/Account.action'. The 'Content encoding' is 'UTF-8'. The 'Redirect Automatically' checkbox is unchecked, while 'Follow Redirects', 'Use KeepAlive', 'Use multipart/form-data', and 'Browser-compatible headers' are checked. The 'Parameters' tab is active, showing a table of parameters to be sent with the request.

Name:	Value	URL Encode?	Content-Type	Include Equals?
username	j2ee	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
password	j2ee	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
signon	Login	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
_sourcePage	ugjw681pb-rFNw9D4iwm2czwz46oT0V...	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
_fp	lb36QnWFm0s=	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>

# Performance Testing with JMeter

Part II

# Agenda

## Part II:

- Data Driven Testing with CSV Data Set Config Element
- Pre/Post Processors
- Assertions
- Pacing
- Database Performance Testing
- JMeter CLI mode
- Monitoring Server Performance

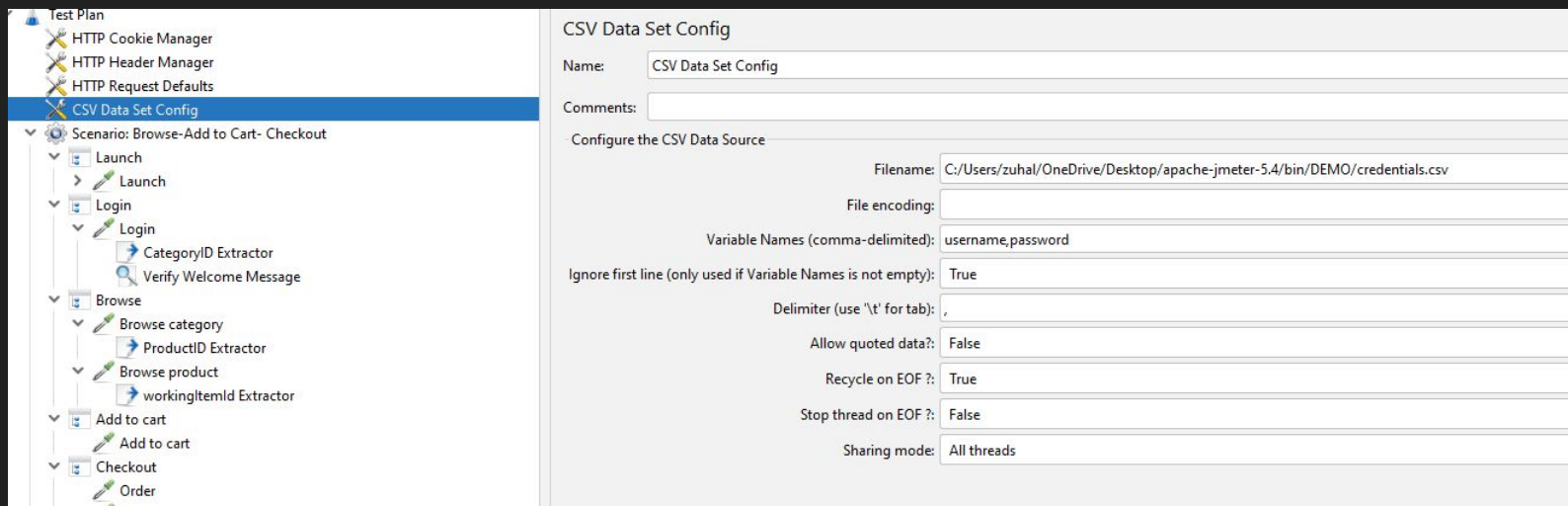
# CSV Dataset Config

- This element is used to supply data to JMeter variables from an external csv file and helps to achieve data driven testing.
- Data driven testing can be used for username/passwords, Product IDs, URL's etc.
- We can use same CSV file to supply data for all threads or we can have separate file for each thread.
- All threads use separate row data from CSV file, starting from line 1.
- CSV file opens only once in the first iteration, then used by all available threads.

# Sample Scenario - Data Driven Performance Testing

1. Create a csv file with the credentials for the users.
2. Add a CSV Data config element to the Pet Store application test script under Test Plan.
3. In the CSV Data config element, browse the csv file for the file name.
4. Enter "\${username}" and "\${password}" for the variable names.
5. Run and view the results in tree.

# CSV Dataset Config



The screenshot displays the JMeter configuration for a CSV Data Set. On the left, the Test Plan tree shows the following structure:

- Test Plan
  - HTTP Cookie Manager
  - HTTP Header Manager
  - HTTP Request Defaults
  - CSV Data Set Config**
  - Scenario: Browse-Add to Cart- Checkout
    - Launch
      - Launch
    - Login
      - Login
        - CategoryID Extractor
        - Verify Welcome Message
    - Browse
      - Browse category
        - ProductID Extractor
      - Browse product
        - workingItemid Extractor
    - Add to cart
      - Add to cart
    - Checkout
      - Order

The right pane, titled 'CSV Data Set Config', contains the following configuration fields:

- Name: CSV Data Set Config
- Comments:
- Configure the CSV Data Source
  - Filename: C:/Users/zuhail/OneDrive/Desktop/apache-jmeter-5.4/bin/DEMO/credentials.csv
  - File encoding:
  - Variable Names (comma-delimited): username,password
  - Ignore first line (only used if Variable Names is not empty): True
  - Delimiter (use '\t' for tab): ,
  - Allow quoted data?: False
  - Recycle on EOF?: True
  - Stop thread on EOF?: False
  - Sharing mode: All threads



# Random CSV Data Set Config

- Random CSV Data Set Config is an additional plugin that you can import using Plugins Manager.
- The element gets records from the csv file in random order.

bzm - Random CSV Data Set Config

Name:

Comments:

[Help on this plugin](#)

Filename:

File encoding:

Delimiter (use '\t' for tab):

Variable names (comma-delimited):

Random order: ☒

Rewind on end of list: ☒

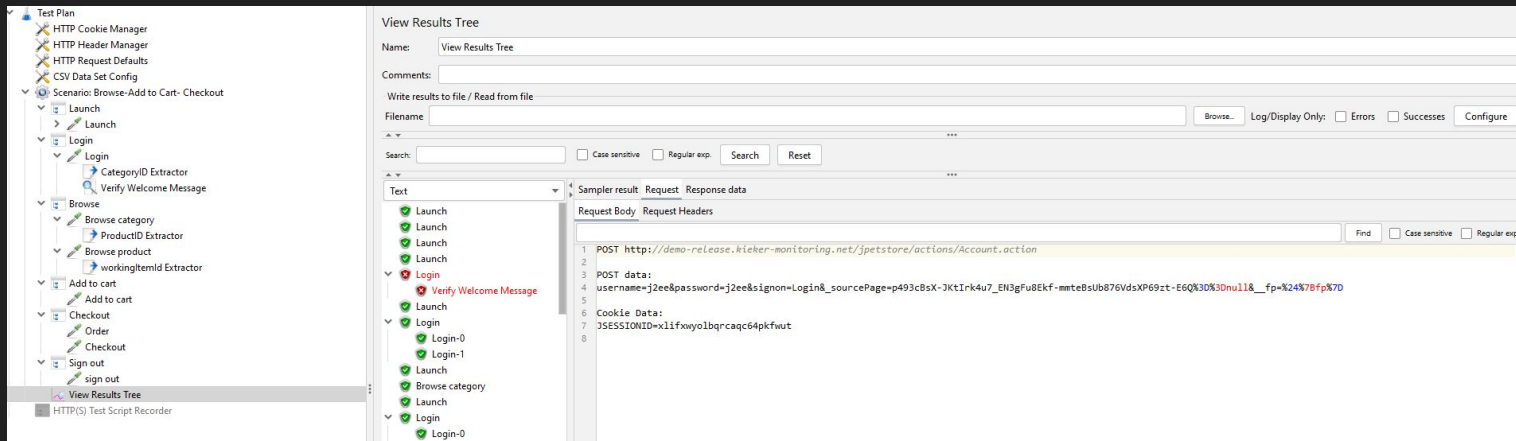
First line is CSV header: ☒

Independent list per thread: ☐

```
$(password) = j2ee
$(username) = j2ee
-----
$(password) = xxxxxxxx
$(username) = xxxxxxxx
-----
$(password) = j2ee
$(username) = j2ee
-----
```

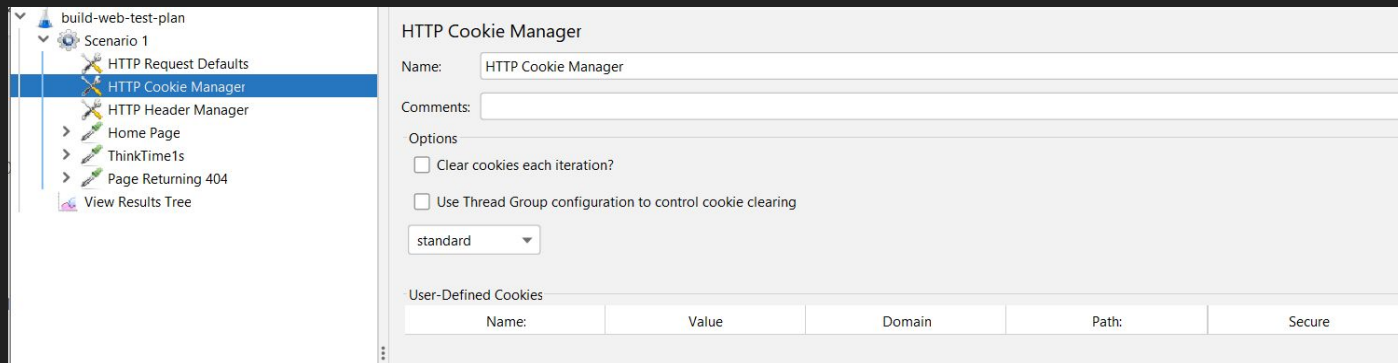
# Assertions

- Assertions are used to apply validations on the response to a request.
- All assertions come with a cost in terms of CPU or memory consumption.
- The most commonly used assertion is the **Response Assertion**, which checks whether a response text/body/code/message/header contains, matches, or equals a specified pattern.



# HTTP Cookie Manager

- The HTTP Cookie Manager stores and sends cookies just like a web browser. If the response of a request contains a cookie, the Cookie Manager automatically stores that cookie and uses for all future requests to that particular web site.
- Each JMeter thread has its own "cookie storage area".



# HTTP Request Defaults

- This element allows to set default values for the HTTP Request controllers. For example, if there are 10 controllers and all of the requests are being sent to the same server, then the controllers will inherit the defaults like server name from the HTTP Request Defaults element.

The screenshot shows the 'HTTP Request Defaults' configuration window. On the left, a tree view shows the project structure: 'build-web-test-plan' > 'Scenario 1' > 'HTTP Request Defaults' (selected). The main panel is titled 'HTTP Request Defaults' and contains the following fields and sections:

- Name:** HTTP Request Defaults
- Comments:** Notice Timeouts:  
Read to 30s  
Connect to 5s
- Tabs:** Basic (selected), Advanced
- Web Server:**
  - Protocol [http]:
  - Server Name or IP:
  - Port Number:
- HTTP Request:**
  - Path:
  - Content encoding:
- Parameters / Body Data:**
  - Send Parameters With the Request:
  - Table with 5 columns: Name, Value, URL Encode?, Content-Type, Include Equals?

# Controllers

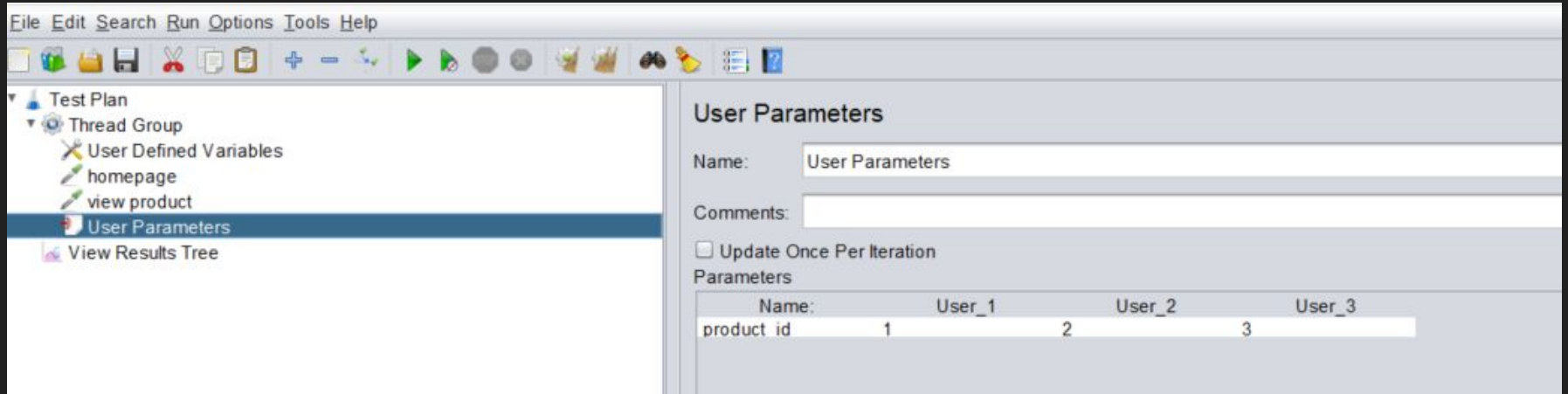
- Logic controllers help determine the order in which samplers are processed.
- Controllers are container elements that group or hold one or more samplers.
- There are various controllers ranging from logic to looping, and recording controllers.
- Common controllers used in Test Plans:
  - Recording Controller
  - Transaction Controller
  - Simple Controller
  - Interleave Controller
  - Random Controller
  - Runtime Controller
  - If and Loop Controllers

# PreProcessors

- PreProcessors are used to execute actions before the sampler requests are executed in the test scenario.
- PreProcessors can be used for different performance testing needs, like fetching data from a database, setting a timeout between sampler execution or before test data generation.
- Some common processors:
  - User Parameters
  - JSR223 Preprocessor
  - JDBC Preprocessor
  - Bean Shell Preprocessor
  - HTML Link Parser

# User Parameters

- We can define a parameter like an id for each user in the User Parameters Preprocessor and use it in the samplers by adding as a parameter.



# PostProcessors

- PostProcessors allow to extract some information from the sampler response to use it for further requests such as session information. Some common

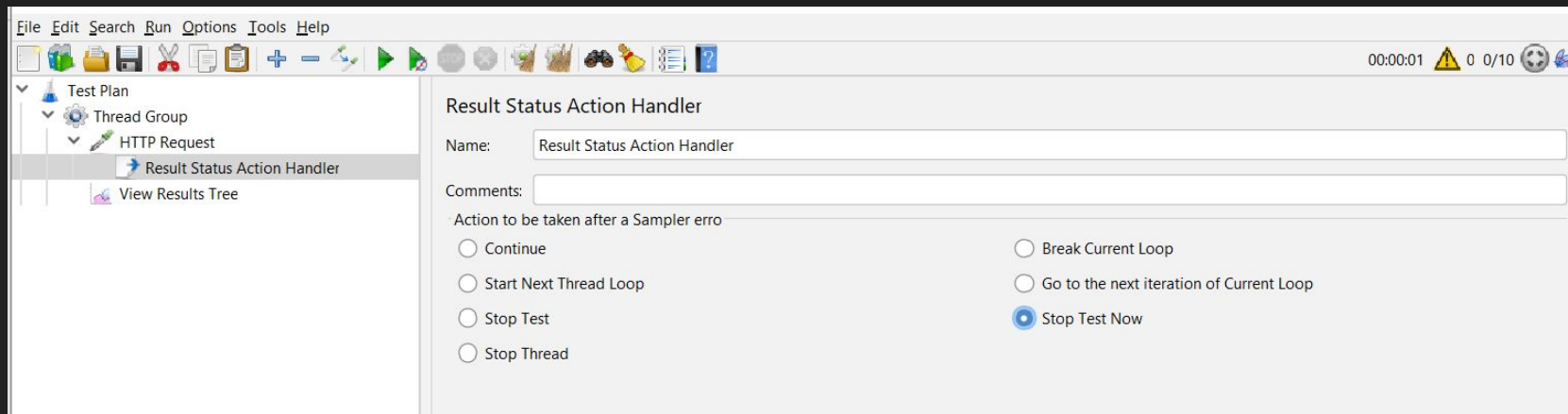
## PostProcessors:

- Regular Expression Extractor
- CSS Selector Extractor
- XPath Extractor
- Result Status Action Handler
- BeanShell PostProcessor
- JSR223 PostProcessor
- JDBC PostProcessor
- JSON Extractor



# PostProcessors

- Result Status Action Handler: This test element allows the user to stop the thread or the whole test if the relevant sampler failed.

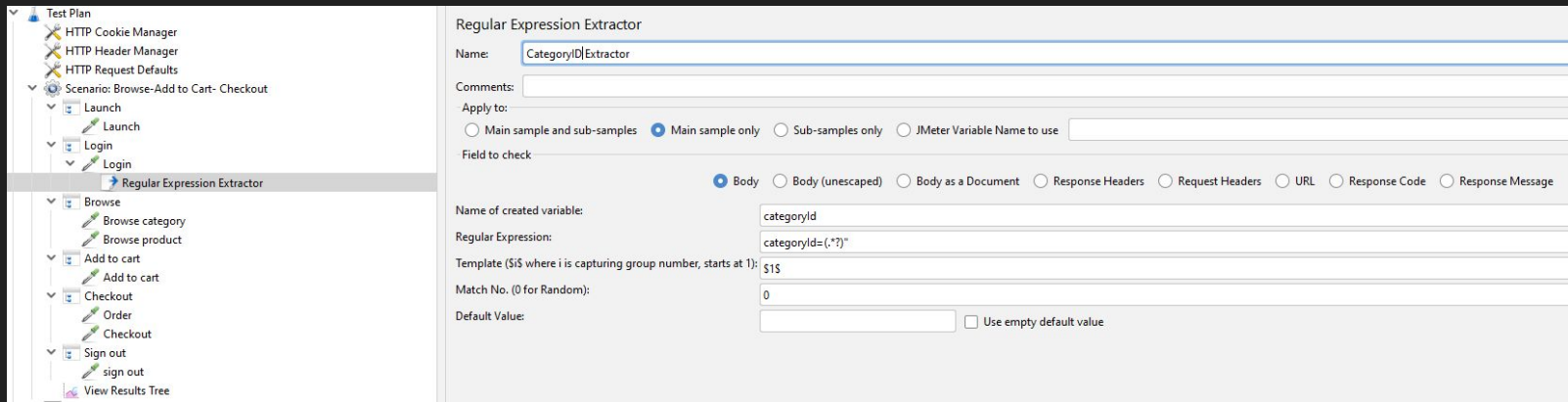


# Regular Expression Extractor

- We can use regex to extract values from the response during test execution, store it in a variable and use further in JMeter. Regular Expression Extractor is a post processor that can be used to apply regex on response data. The matched expression derived on applying the regex can then be used in a different sampler dynamically in the test plan execution.
- It is also possible to extract multiple variables using one extractor.
  1. Create a Test Plan
  2. Add a Regular Expression Extractor to a request to extract a dynamic value.
  3. In the next step, refer to the extracted value.
  4. Add a Debug Sampler to see if the correct values are extracted.
  5. Run the test.

# Regular Expression Extractor

- **Regular Expression:** The pattern against which the text to be extracted will be matched.
- **Template:** Each group of extracted text placed as a member of the variable Person, following the order of each group of pattern enclosed by '(' and ')'. Each group is stored as `refname_g#`, where `refname` is the string you entered as the reference name, and `#` is the group number. `$1$` refers to group 1, `$2$` refers to group 2, etc. `$0$` refers to whatever the entire expression matches.
- **Match No:** The occurrence of the pattern that we want to extract.
- **Default:** Default value if the item is not found (Optional field).



The screenshot shows the JMeter interface with a Test Plan on the left and the Regular Expression Extractor configuration on the right.

**Test Plan Structure:**

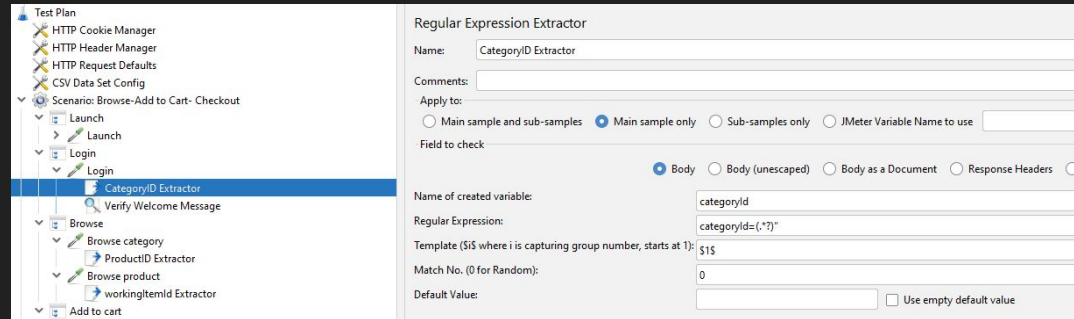
- Test Plan
  - HTTP Cookie Manager
  - HTTP Header Manager
  - HTTP Request Defaults
  - Scenario: Browse-Add to Cart- Checkout
    - Launch
    - Login
    - Regular Expression Extractor (Selected)
    - Browse
      - Browse category
      - Browse product
    - Add to cart
      - Add to cart
    - Checkout
      - Order
      - Checkout
    - Sign out
      - sign out
    - View Results Tree

**Regular Expression Extractor Configuration:**

- Name: `CategoryIDExtractor`
- Comments: (Empty)
- Apply to: ☐ Main sample and sub-samples ☒ Main sample only ☐ Sub-samples only ☐ JMeter Variable Name to use
- Field to check: ☒ Body ☐ Body (unescaped) ☐ Body as a Document ☐ Response Headers ☐ Request Headers ☐ URL ☐ Response Code ☐ Response Message
- Name of created variable: `categoryid`
- Regular Expression: `categoryid=(-?)"`
- Template (\$1\$ where i is capturing group number, starts at 1): `$1$`
- Match No. (0 for Random): `0`
- Default Value: (Empty field) ☐ Use empty default value

# Sample Scenario - Regular Expression Extractor

1. Add a Thread Group in a test plan.
2. Add HTTP request to a thread group.
3. Add Post Processor “Regular Expression Extractor” under Login to extract values for Category ID.
4. Set the Category ID as variable in the next step.



# Command Line Interface

- The GUI mode is fine for recording, building, and running preliminary tests, but it is not recommended to use for large scale testing since it in itself consumes some resources. For large scale testing, the command line should be used.
- Disable all your listeners for running the tests from CLI in order to use the resources effectively.

```
=====
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]
& increase Java Heap to meet your test requirements:
    Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
=====
```

# Command Line Interface

- **Running load test within the CLI** uses less resources. Or to run on a Unix machine or to run distributed test, we can use CLI.

```
jmeter -n -t [jmx file] -l [test results file]
```

-n: run in non-GUI mode,

-t: specifies the path to source .jmx script to run,

-l: specifies the path to the JTL file which will contain the raw results.

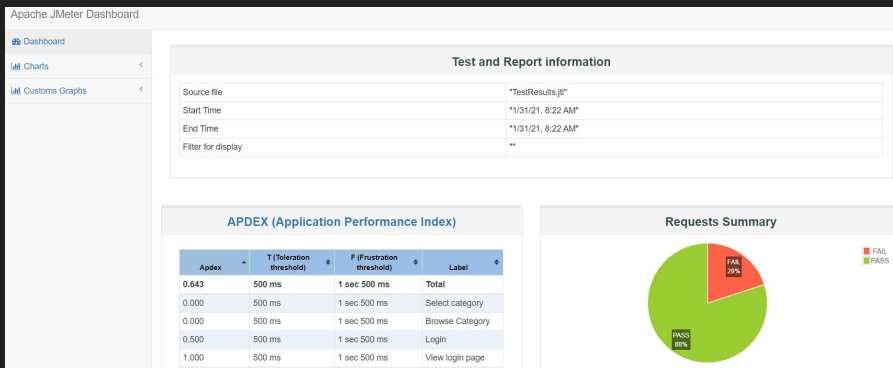
- Create **HTML Dashboard** at CLI runtime to get the results as html statistics.

```
jmeter -n -t [jmx file] -l [test results file] -e -o [Path to the report folder]
```

- Summary Report listener can be used to view the jtl file from the GUI.

# Test Results as HTML

- JMeter supports dashboard report generation to get graphs and statistics from a test plan.
- **Apdex (Application Performance Index)** is an open standard for measuring performance of software applications in computing.
- The Apdex method converts many measurements into one number on a uniform scale of 0 to 1 (0 = no users satisfied, 1 = all users satisfied).



# Throughput

- Throughput is a measure of the number of transactions of a given type that the system processes in a unit of time. For example, the number of orders per hour or the number of HTTP requests per second.

$$\text{Throughput} = [\text{number of requests}] / ([\text{processing time}] + [\text{think time}])$$

Throughput is typically measured according to the number of transactions that can be accomplished within a given time frame.



# Pacing

- As per the performance test requirements, a projected target for the number of transaction might be required to test. For example, at peak hours, the reporting module of an application receives high volumes of traffic, while the product viewing module receives much less traffic.
- By analyzing the server logs, we can come up with the number of requests per second during peak and off-peak hours for various modules within the application under test.
- However, it is difficult to tweak thread groups and timer delay settings within JMeter to simulate a targeted requests per second (RPS) value for the modules. Dynamically adjusting the load and delays per the requested RPS values is called **Pacing**.

# Pacing

- Pacing allows the load test to better simulate the time gap between two sessions. In reality, the same user will not instantaneously go to the next iteration, so this wait time between sessions creates a more realistic load on the application.
- To implement pacing, we can use either Groovy Scripting or following timers:

Constant Throughput Timer: Allows to regulate the amount of requests per **minute**

Throughput Shaping Timer: Allows to regulate the amount of requests per **second** and allows to configure multiple Request Per Second (RPS) scenarios.

# Sample Scenario – Throughput Shaping Timer

The goal is to test a REST endpoint and ensure it could handle 1000 transactions per second under peak load.

## 1. Specify the following values for its attributes:

Number of Threads (users): 300

Ramp-Up Period (in seconds): 60

Loop count: Forever

## 2. Add the HTTP Request component to Thread Group as following

Server Name or IP: api-jcb.herokuapp.com

Method: GET

Path: /holiday-requests

## 3. Add Throughput Shaping Timer to Thread Group

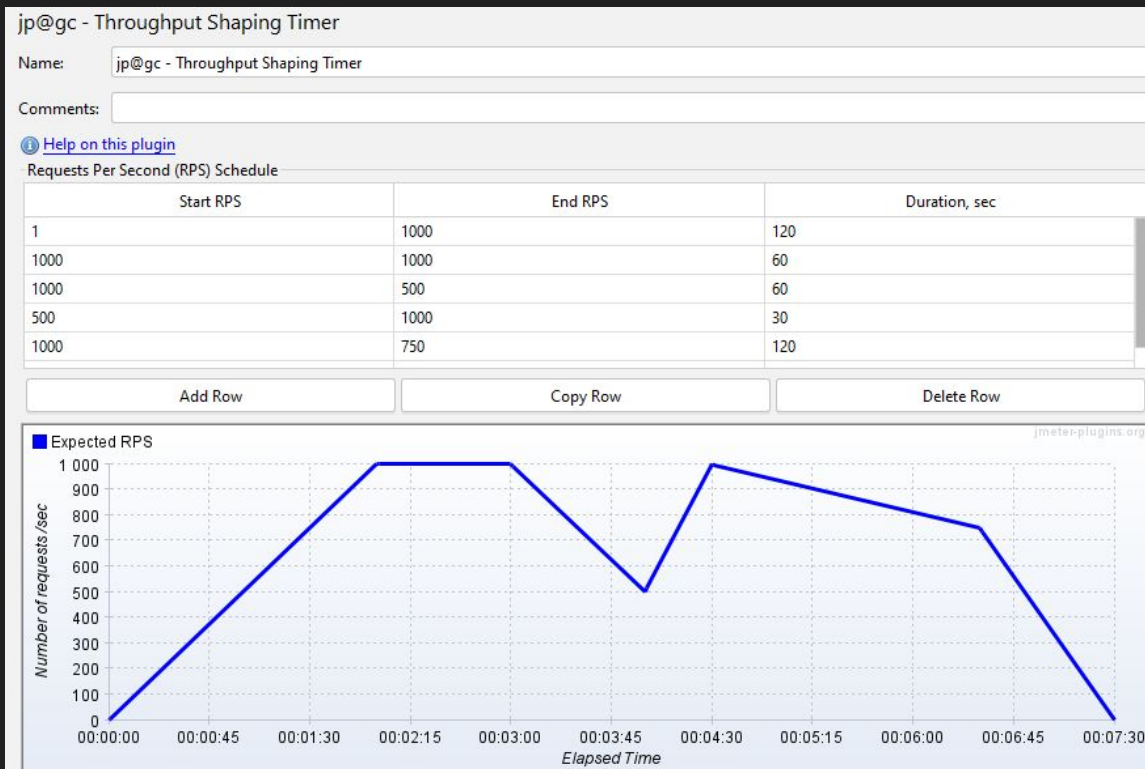
## 4. Set the Start and End RPS values and duration.

## 5. Add the Transactions per Second listener and compare the results with the values set in Throughput Shaping Timer.



# Throughput Shaping Timer – Expected RPS

- Expected Request Per Second profile set in Throughput Shaping Timer



# Transactions per Second

- The results closely mimic the expected RPS.



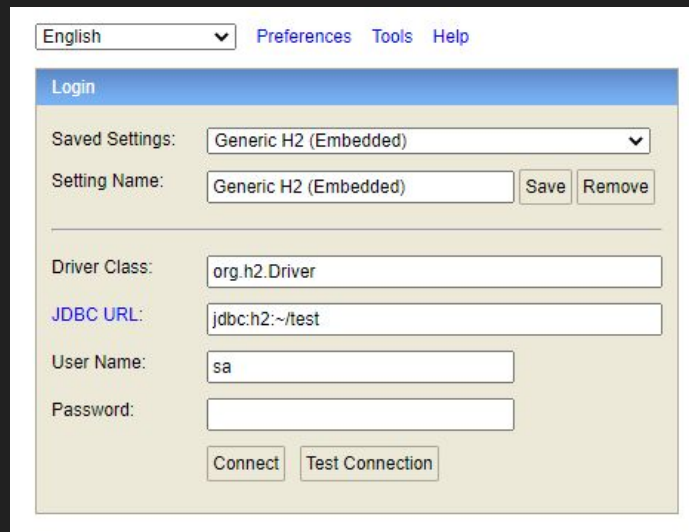
# Database Testing

- Database performance testing may help identify database bottlenecks such as expensive table joins, missing table indexes, slow running queries.
- To test a relational database, JMeter offers two components, **JDBC Connection Configuration and JDBC Request**.
- The JDBC Connection Configuration component is needed to set up the connection to the database. The most important elements in its configuration are the **database URL, JDBC driver class, the username, and the password**. This information is needed to successfully connect to any relational database. In our practice, we will use H2 which is an open source pure Java SQL database.
- For any other database such as Oracle, MySQL, Postgres, DB2, and so on, the way to connect to any of these relational databases is through their JDBC driver classes, which come bundled in the form of a JAR file. The JAR file containing the driver class for your database can be downloaded by searching online or at the vendor's website.

# Sample Scenario – Database Testing

1. For demo purpose, we will set up a database called H2 database. For Mac OSX, use the command **brew install h2**.

- Download the last stable version of H2 database from [www.h2database.com/html/download.html](http://www.h2database.com/html/download.html)
- Extract the archive and save to a location.
- Launch the H2 console and the database will launch your browser.
- Click the Connect button on the Login window.
- Copy the sql script into the space in the console and Run. This will create a table and populate some data for us to test.



The screenshot shows the H2 database Login window. At the top, there is a language dropdown menu set to 'English' and three links: 'Preferences', 'Tools', and 'Help'. The main title of the window is 'Login'. Below the title, there are two sections. The first section is for 'Saved Settings', showing a dropdown menu with 'Generic H2 (Embedded)' selected, and a 'Setting Name' field also containing 'Generic H2 (Embedded)', with 'Save' and 'Remove' buttons. The second section contains four input fields: 'Driver Class' with 'org.h2.Driver', 'JDBC URL' with 'jdbc:h2:~/test', 'User Name' with 'sa', and an empty 'Password' field. At the bottom of this section are two buttons: 'Connect' and 'Test Connection'.

# Sample Scenario – Database Testing

2. Download JDBC driver for h2 database

(<https://dbschema.com/jdbc-driver/H2.html> ) and copy to JMeter/lib folder. This will allow us to connect to the database from JMeter. If you are using an oracle database in your project, you should have ojdbc7.jar file.

3. Launch the JMeter.

4. Add a JDBC Connection Configuration component to the test plan and configure it as follows and leave the rest as is:

- Variable Name: myDB
- Database URL: jdbc:h2:~/test
- JDBC Driver class: org.h2.Driver
- Username: sa

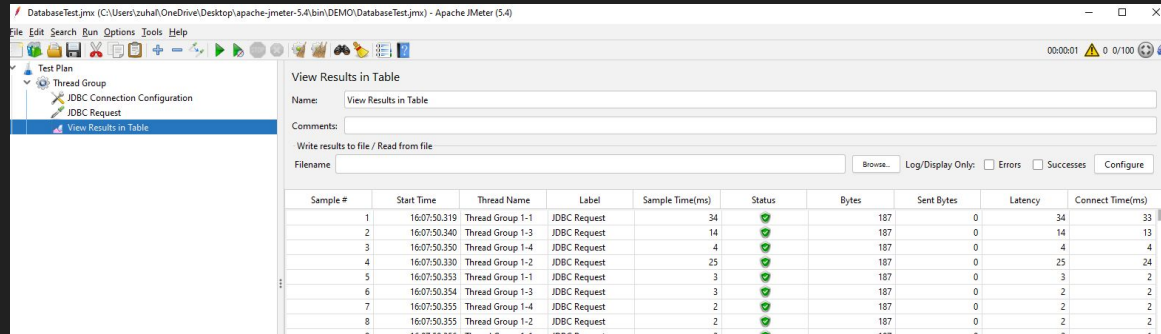


# Sample Scenario – Database Testing

5. Disconnect from the database.
6. Add a Thread Group under Test Plan.
7. Add a JDBC Request sampler to the Thread Group. Fill in the following:

Variable Name: myDB

SQL Query: select \* from employees

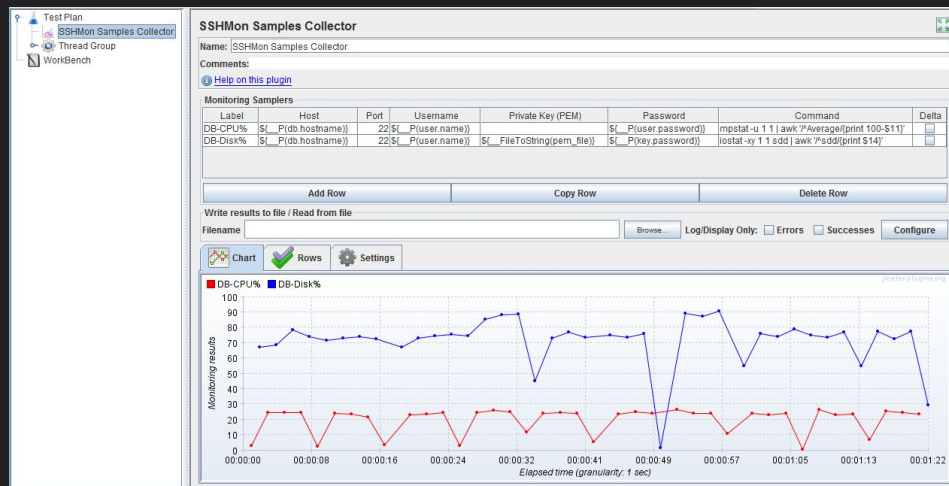


Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	16:07:50.319	Thread Group 1-1	JDBC Request	34	Success	187	0	34	33
2	16:07:50.340	Thread Group 1-3	JDBC Request	14	Success	187	0	14	13
3	16:07:50.350	Thread Group 1-4	JDBC Request	4	Success	187	0	4	4
4	16:07:50.330	Thread Group 1-2	JDBC Request	25	Success	187	0	25	24
5	16:07:50.353	Thread Group 1-1	JDBC Request	3	Success	187	0	3	2
6	16:07:50.354	Thread Group 1-3	JDBC Request	3	Success	187	0	2	2
7	16:07:50.355	Thread Group 1-4	JDBC Request	2	Success	187	0	2	2
8	16:07:50.355	Thread Group 1-2	JDBC Request	2	Success	187	0	2	2

8. Add View Results in Table listener.
9. Save the plan and run it.

# Monitoring Server Performance

- Collecting internal metrics of the application server can allow us to decide the server planning, predict peak load and understand the root cause of possible low throughput or errors.
- PerfMon Metrics Collector** is agent that runs on the server and monitors the performance of the system. But it has some disadvantages like affecting the performance of the system or having to run it after starting the performance tests.
- SSHMon plugin** is an additional plugin of JMeter which allows to connect to the server via SSH and retrieves server data.  
<https://github.com/tilln/jmeter-sshmon>



# What is achieved with **Load Testing**?

- The number of users the system is able to handle is identified.
- The response time of each transaction is analyzed.
- How does each component of the entire system behave under load i.e Application server components, web server components, Database components etc.
- What server configuration is best to handle the load is identified.
- Whether the existing hardware is enough or is there any need for additional hardware.
- Bottlenecks like CPU utilization, Memory Usage, Network delays, reading database, third party apps etc., are identified.