WHAT IS PERFORMANCE TESTING ?

* *It plans to meet the primary testing objective*
* *It comes under BBT*
* *It comes under Non-functional testing*
* *It comes under automation testing*

TYPES OF PERFORMANCE TESTING

* RESPONSE TIME TESTING : *this is the time taken by server to respond to the client request, usually measured in milli-seconds ; here, vUsers=1*
* LOAD TESTING : *here, we simulate volume/bulk of vUsers (say 100) who will bombard/hit the server with various requests concurrently to validate the server response time to the client ; here, we evaluate the server*
* STRESS TESTING : *here, we evaluate the client to know if they are timing out due to stress or having reached the break even point of the system, after waiting for ‘x’ time-out period defined by the developer*
* SOAK/ENDURANCE TESTING : *here, we bombard the server with various client requests for the specified duration of time ‘n’ seconds/minutes/hours*

HTTP METHODS/VERBS : *It describes ‘the type of client request’ being sent to server using http methods such as*

* *GET > is used to fetch/retrieve existing data from the server/database ; similar to SELECT query*
* *POST > is used to create a new resource/object on the server/database ; similar to INSERT query*
* *PUT > is used to update an existing data (full data or partial data) on the server/database ; similar to UPDATE query*
* *PATCH > is used to update an existing data (partial data) on the server/database ; similar to UPDATE query*
* *DELETE > is used to delete existing data on the server/database ; similar to DELETE query*

*REQUEST HEADERS : these are additional meta-data sent by client to the server, along with request*

* *Connection : keep-alive, close*
* *User-Agent : OS, browser-name & version*
* *Authorization : user credentials*
* *Cookies : user preferences*
* *Content-Type : client request payload format*
* *Accept : client preferences of server response*
* *Accept-Lang : server response in international language*

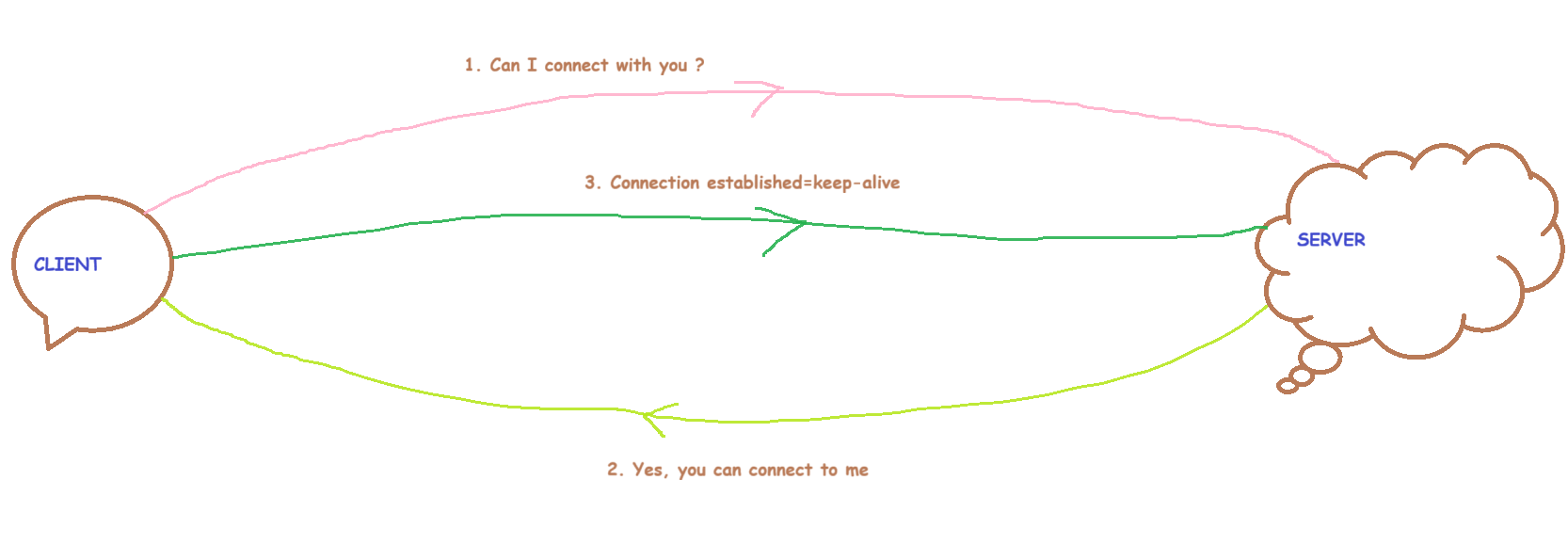
*RESPONSE CODE: It is a 3 digit status code sent by server to the client, to reveal client request status*

*Range:* {100-599}

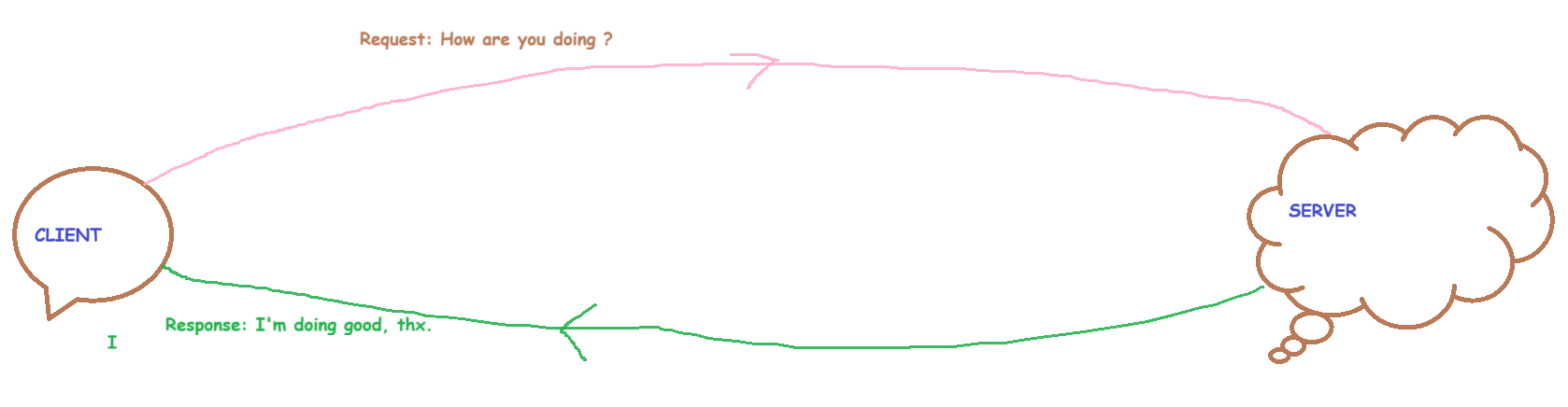
* 100-199 = 1xx = informational responses
* 200-299 = 2xx = successful responses
  + 200 OK // GET, PUT, PATCH, DELETE
  + 201 Created // POST
  + 202 Accepted // PUT, PATCH
  + 204 No Content // DELETE
* 300-399 = 3xx = re-directional responses
* 400-499 = 4xx = client side errors
  + 400 Bad syntax
  + 401 Unauthorized
  + 403 Forbidden
  + 404 Not found
  + 415 Unsupported media type
  + 429 Too many requests
* 500-599 = 5xx = server side errors
  + 500 Internal server error
  + 503 Service unavailable

KPI/PERFORMANCE METRICS

* Connection time : *it is the time taken by client to establish a valid connection with the server*



* Latency : *it is the time taken by server to send the 1st byte/packet to the client*
* Response time : *it is the time taken by server to send the complete response to the client*



* Client side Throughput : *it is the number of client requests submitted to the server per second/minute*
* Server side Throughput : *it is the number of client requests processed by the server per second/minute*
* Error % : *it is the number of errors (4xx or 5xx) encountered during load, stress, soak/endurance testing activities*
* CPU % utilization : *it is the % of CPU utilization measured during load, stress, soak/endurance testing activities to ensure CPU% utilization doesn’t cross 90% & above*
* RAM % utilization : *it is the % of RAM utilization measured during load, stress, soak/endurance testing activities to ensure RAM% utilization doesn’t cross 90% & above*
* 90% response time : *it is the average response time calculated for the top 90% of vUsers who has experienced lowest response times*
* 95% response time : *it is the average response time calculated for the top 95% of vUsers who has experienced lowest response times*
* Size sent : *it is the size of client request sent to server*
* Size received : *it is the size of server response sent to client*

vUsers=10

|  |  |
| --- | --- |
| vUser\_1 | 120 ms [response time] |
| vUser\_2 | 111 ms |
| vUser\_3 | 50 ms |
| vUser\_4 | 75 ms |
| vUser\_5 | 76 ms |
| vUser\_6 | 89 ms |
| vUser\_7 | 39 ms |
| vUser\_8 | 65 ms |
| vUser\_9 | 100 ms |
| vUser\_10 | 45 ms |

* Average response time (for all 10 vUsers) =77 ms
* 90% percentile = 72 ms

|  |  |  |
| --- | --- | --- |
|  | Average response time | 90% metric |
| vUser\_1 | 120 ms [response time] | 39 |
| vUser\_2 | 111 ms | 45 |
| vUser\_3 |  | 50 |
| vUser\_4 |  | 65 |
| vUser\_5 |  | 75 |
| vUser\_6 | 89 ms | 76 |
| vUser\_7 |  | 89 |
| vUser\_8 |  | 100 |
| vUser\_9 | 100 ms | 111 |
| vUser\_10 |  | 120 |

WHY JMETER ?

* Open-source application from Apache community
* Has support of ~120 java based plugins
* Built using java , free from any OS dependencies
* Has support for various protocols
* Simple & intuitive GUI
* Low code platform, more of configurations
* Built in editor to write Beanshell/java scripts
* Supports non-gui/cli mode
* Generates test reports .csv, .html
* Seamless integration with other tools github, Jenkins
* Record & Playback feature
* Supports 3rd party record & playbook tools
* Open community support
* Official support from Apache team
* Regular updates to Jmeter tool

Download > Unzip > Launch JMeter tool from /bin folder by double click on ApacheJMeter.jar or jmeter.bat

OR

Open command prompt > change directory to /bin folder > type ApacheJMeter.jar & press ENTER

OR

Open command prompt > change directory to /bin folder > type jmeter.bat & press ENTER

TEST PLAN ELEMENTS IN JMETER

1. Thread Group : Here, we define # of vUsers, ramp-up time, # of iterations, soak testing definition
   1. Ramp up time: It is the time taken by JMeter to submit the request(s) for all vUsers to the server
   2. Loop count: It is the iteration count per vUser to submit/repeat the same client request to the server
2. Samplers: *it is the request to be sent to server*
3. Listeners: *it is the response coming from server*
4. Config elements : optional element
5. Timers : optional element
6. Assertions : optional element
7. Logic controllers : optional element
8. Pre-Processors : optional element ; we add this as a child to a request/sampler
9. Post-Processors : optional element ; we add this as a child to a request/sampler
10. Test Fragment : optional element ; rough book ; this wont be executed by JMeter
11. HTTPs test script recorder : optional element

EXECUTION ORDER OF JMETER TEST PLAN ELEMENTS

PROBLEM STATEMENTS

1. Conduct a simple performance test on any given webpage

<https://global.rakuten.com/>

Open Jmeter > Thread group > Configure vUsers, ramp-up period, iteration/loop count > Add a sampler > configure the protocol, server, port, http method & path > add a listener > view results tree > save the script > run the script > analyze the test results

1. Conduct a simple performance test on multiple webpages  
     
   <https://global.rakuten.com/corp/>

<https://global.rakuten.com/corp/careers/>   
<https://global.rakuten.com/corp/about/index.html#leadership>

Key KPI metrics: Load time, Error rate, Response code

198 131

1. Conduct a load test on various web pages
   1. vUsers=20
   2. rampUp=3 seconds
   3. View results tree, View results in table
2. Conduct a load test on various web pages
   1. vUsers=10
   2. rampUp=3 seconds
   3. loopCount=3

View results tree, View results in table

1. Conduct a soak/endurance test on various web pages
   1. vUsers=40
   2. Ramp up duration=5 seconds
   3. Duration of test=30 seconds
2. Configuration of HTTP request defaults  
   *it is used to initialize the value of protocol, server name, port, path in 1 location which can be shared across various http samplers*
3. Conduct a load test to simulate a real browser behavior such as Chrome+Android OS, Firefox+Linux OS, Edge+Windows OS, Safari+Mac OS
4. Conduct a load test to simulate multiple browsers at once
   1. Chrome+Android OS
   2. Firefox+Linux OS
   3. Edge+Windows OS
   4. Safari+Mac OS

Solution: Use Data driven load automation testing (DDT) > csv file > Add CSV dataset config to JMeter

Variable Syntax in JMeter : ${variableName}

1. Conduct a load test to simulate browser behavior by accepting incoming cookies from server  
     
   Solution: Use HTTP cookie manager
2. Conduct a load test to simulate browser behavior by accepting incoming cache from server  
     
   Solution: Use HTTP cache manager

To simulate a real browser behavior in JMeter, we have to add the below elements to JMeter test plan:

* HTTP header manager > User-Agent
* HTTP cookie manager > to accept server cookies
* HTTP cache manager > to accept server cache

1. Generate a test report in .csv after executing performance/load/soak testing
   1. The individual results of each vUser in .csv
   2. The average results of all users in .csv
2. How to load the .csv test report in an existing/new listener in JMeter tool ?
3. How to fix the timestamp format in the generated .csv test report?

Navigate to /bin directory > open jmeter.properties file > search for timestamp > remove # symbol > save the file > restart Jmeter > run the .jmx test script > observe the timestamp in the generated test report.csv

1. Generate a test report in .html after executing performance/load/soak testing  
     
   Click on Tools > Generate HTML report > fill all the 3 fields > Click on ‘Generate’ button
2. How do you simulate a real user behavior in performance tests?

Usage of Timers > Constant timer > Uniform random timer >

Timers are executed first & then the samplers

1. How do you validate the server response during performance tests?

We have to conduct assertions/functional checks on the server response received such as

* Response code : Verify if response code=200
  + This can be added to a http request sampler
  + This can be added at a thread group level
* Response message : Verify if response message = OK
  + This can be added to a http request sampler
  + This can be added at a thread group level
* Response time : Verify if response time is <2 seconds
  + This can be added to a http request sampler
  + This can be added at a thread group level
* Response body & content : Verify if the received server response body is correct & valid
  + This can be added to a http request sampler
* Response headers : Verify if the response headers contain valid content-type
  + This can be added to a http request sampler
  + This can be added at a thread group level
* Response body size : Verify if response body size is non-empty
  + This can be added to a http request sampler
  + This can be added at a thread group level
* Response HTML content : Verify if the html response is syntactically correct
  + This can be added to a http request sampler
* Response XML content : Verify if the xml response is syntactically correct
  + This can be added to a http request sampler
* Response JSON content

1. Submission of a form using POST http method
2. Usage of variables in JMeter tool  
     
   A variable acts as a container to store an object’s value

Syntax: ${variableName}

* Test Plan > these variables are global across the test plan thread groups
  + Multiple thread groups
  + Debug sampler
  + Executed thread groups in sequential order
* Config element > User defined variables : these variables are specific to the thread group where they were defined
  + Multiple thread groups
  + Debug sampler
  + Executed thread groups in sequential order
* preProcessor > User Parameters : these variables are created before executing the sampler
  + Multiple users sharing same variable with different values during test execution
  + Debug sampler

1. Usage of SMTP sampler

SMTP = simple email transfer protocol  
This protocol is used to send messages or emails

From: [skilltoupgrade@gmail.com](mailto:skilltoupgrade@gmail.com)   
To: [manager@ibm.com](mailto:manager@ibm.com), [client@rakuten.com](mailto:client@rakuten.com)   
Subject:  
Body:  
Attachment:

1. Usage of FTP sampler
   1. FTP = file transfer protocol
   2. FTP protocol is used to transfer files between FTP client & FTP server
   3. File upload = PUT
   4. File download = GET
   5. FTP server always uses authorization for file upload or file download actions
      1. FTP URL: [ftp.dlptest.com](ftp://ftp.dlptest.com)
      2. FTP User: dlpuser
      3. Password: rNrKYTX9g7z3RgJRmxWuGHbeu
   6. FTP client = Filezilla software
2. Conduct load testing of all the possible URLs in the given web application
   1. Usage of preProcessor > HTML link parser
3. Conduct load testing on a given web application and if the server takes more time than needed, the client has to time-out so that stress testing can be evaluated
   1. Usage of preProcessor > Sample timeout
4. Usage of setUp threadGroup & tearDown threadGroup
   1. setUp threadGroup executes first
   2. threadGroup executes after setUp and before tearDown
   3. tearDown threadGroup executes in the last
   4. setUp threadGroup > threadGroup > tearDown threadGroup
5. Running multiple thread groups
6. Usage of logic controllers
   1. Once only controller
   2. Loop controller
   3. Run-time controller
   4. Simple controller
   5. Random order controller
   6. Random controller
   7. If controller
   8. Throughput controller
   9. Interleave controller
   10. Transaction controller
       1. It will group samplers together
       2. It will calculate response times of all its group samplers & print the overall response time
       3. It will ensure its child group samplers are executed in sequential order while ensuring vUsers are executed in parallel order
   11. Critical section controller
       1. It will ensure its child group samplers are executed in sequential order & also executes the vUsers in sequential order
   12. Module controller > a sampler is reused
   13. Include controller > a .jmx file is reused
       1. Home > Login > Reserve
       2. Home > Login > Vacation
7. Record & Play back tests
   1. Blazemeter chrome extension
      1. Blazedemo.com
      2. Katalon-healthcare application
         1. Login > Book Appt > Logout
            1. DDT-CSV
            2. DDT-JMeter functions

${\_\_RandomFromMultipleVars(hospitalAdmission1|hospitalAdmission2,)}

${\_\_RandomString(6,ABCDefghiJKLMnopqRSTUVwxyz,)}

${\_\_RandomFromMultipleVars(program1|program2|program3,)}

${\_\_RandomFromMultipleVars(facility1|facility2|facility3,)}

${\_\_RandomDate(dd/MM/yyyy,,30/11/2025,,)}

* 1. JMeter built in recording
     1. Set the port on Firefox browser
     2. Add suggested excludes
     3. Install Jmeter certificate file on the Firefox browser

1. eCommerce application load tests  
   URL: <https://petstore.octoperf.com/actions/Catalog.action>
   1. Register as a new user
   2. Login to the application
   3. Click on a pet
   4. Click on the product id
   5. Click on the item id
   6. Click on ‘Add to cart’
   7. Click on ‘Proceed to checkout’
   8. Enter your payment details
   9. Confirm your address details
   10. Verify if order is submitted
   11. Logout the user

Challenge#1: The script should register random users test data

Challenge#1: The script should enter random payment test data

Challenge#3: The script should select a random pet

Challenge#4: The script should handle any product id

Challenge#5: The script should handle any item id

Challenge#6: The script should handle various users payment data simulating various OS, various browsers

Challenge#7: Add assertions as we navigate to subsequent pages

Challenge#8: Verify if order is submitted

<td><a href="/actions/Catalog.action?viewProduct=&amp;productId=*FI-FW-02*">*FI-FW-02*</a></td>

<td><a href="/actions/Catalog.action?viewProduct=&amp;productId=*FI-FW-01*">*FI-FW-01*</a></td>

Let us write a regular expression

. = find atleast 1 matching character

\* = find as many matching characters as possible

? = stop finding, when no search is found

Jmeter regular expression: (.\*?)

<td><a href="/actions/Catalog.action?viewProduct=&amp;productId=**(.\*?)**">**(.\*?)**</a></td>

Let us trim it

productId=**(.\*?)**">**(.\*?)**</a></td>

==================

<td><a href="/actions/Catalog.action?viewItem=&amp;itemId=EST-20">EST-20</a></td>

<td><a href="/actions/Catalog.action?viewItem=&amp;itemId=EST-21">EST-21</a></td>

Let us use regular expression

<td><a href="/actions/Catalog.action?viewItem=&amp;itemId=**(.\*?)**">**(.\*?)**</a></td>

Let us trim it

itemId=**(.\*?)**">**(.\*?)**</a></td>

1. API performance tests

* API = Application Programming Interface
* They facilitate the data transfer between client & server either in XML or JSON format
* Types of APIs:
  + XML-RPC-APIs
    - Remote procedure calls
    - Content-Type=XML
    - 100% outdated
  + SOAP APIs
    - Simple object access protocol
    - Content-Type=XML
    - HTTP method used=POST
    - 90% outdated; 10% usage
  + REST APIs
    - Representational state transfer
    - Google = founder of REST APIs
    - Content-Type=any format (*text, html, xml, json, yaml etc*)
    - HTTP methods used=all (*get, post, put, patch, delete*)
    - 100% usage
  + GRAPH-QL APIs
    - Query Language
    - Facebook=founder of Graph-QL APIs
    - Advanced version of REST APIs
    - 40% usage

Web application URL:

<https://demo.testfire.net/index.jsp>

API Swagger documentation:

<https://demo.testfire.net/swagger/index.html>

Correlation > JSON extractor

JSON assertion

1. JMeter plugins
2. Integration with Github
3. Integration with Jenkins