**Explain how you can execute spike testing in JMeter?**

In JMeter, spike testing can be done by using Synchronizing Timer.  The threads are jammed by synchronizing the timer until a specific number of threads have been blocked and then release at once, creating a large instantaneous load.

* **Benchmark Testing:** It is the method of comparing performance of your system performance against an industry standard that is set by other organization
* **Baseline Testing:** It is the procedure of running a set of tests to capture performance information. When future change is made in the application, this information is used as a reference

**Mention the execution order of Test Elements?**

The test plans elements execution order is

* Configuration elements
* Pre-processors
* Timers
* Samplers
* Post-processors
* Assertions
* Listeners

To run Apache JMeter with minimal resource overhead, follow the following :

1. **Use Simple Data Writer as listener** as it has no display, it just writes results to file. Use csv format instead of xml and store only the response fields required. Once you have the response file, you can load it later using any other listener of choice.
2. Run JMeter in **non-GUI mode**. For this use the command   
     
   **jmeter -n -t myTest.jmx -l myResults.jtl**   
     
   where myTest.jmx is your test plan file and myResults.jtl is the where you want jmeter to write the response logs.

*-n* – non-GUI mode – this specifies JMeter is to run in non-GUI mode

*-t* – JMX file – location of the test plan and the name of JMX file that contains the Test Plan

*-l* – log file name of JTL file to log sample results to

**Read uploading and downloading of files using jmeter**

### Correlation- using regular expression extractor- includes fetching dynamic data from preceding requests/calls and posting it to the subsequent requests.

### Parameterization- using CSV data set config

Dynamically increasing number of threads- there is a plugin Stepping thread group.

Example – we can set Stepping Thread Group to increase load by 10 users each 2 minutes:

When application uses signalR - ??

Running selenium script on Jmeter ??

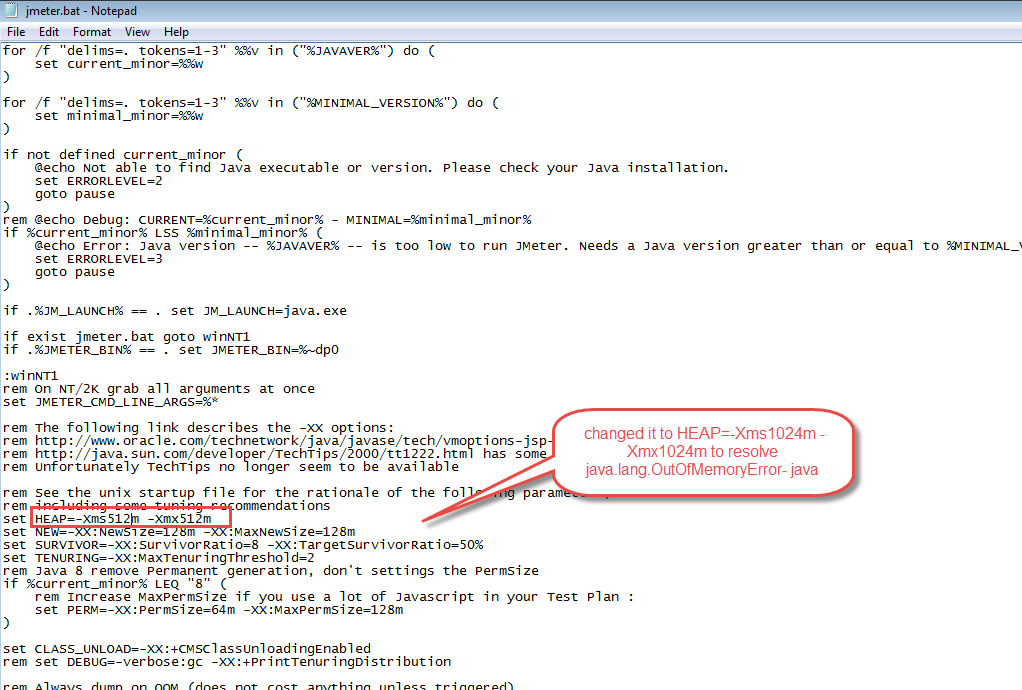
Implementation of REST service??

**Error1-**

For java.lang.OutOfMemoryError – Java memory heap

Resolution-

Change heap size in jmeter.bat and disable unnecessary listeners



**HTTP authorisation manager-** The Authorization Manager lets you specify one or more user logins for web pages that are restricted using server authentication. You see this type of authentication when you use your browser to access a restricted page, and your browser displays a login dialog box. JMeter transmits the login information when it encounters this type of page.

**Error2**-

Response code: “Non HTTP response code: java.io.IOException”  
Response message: “Non HTTP response message: Exceeded maximum number of redirects: 5”

Reason-  [JMeter](http://jmeter.apache.org/) allows maximum 5 redirects by default and your system may be using more than 5 redirects.

Resolution-

Open Jmeter.properties file.

Search for “httpsampler.max\_redirects” property in opened file.

comment it with # and change to 5 from 20.

**Error3-**

For setting the max retry count for the HTTP Request HTTPClient (Number of connection retries performed by HTTP Java sampler before giving up)

Change count of #http.java.sampler.retries in property file

**Error4**-

Connection refused Timed out response while recording

Resolution-

if you are behind a proxy server/Firewall, add proxy settings.

-In command line-

jmeter -H proxy\_host -P proxy\_port

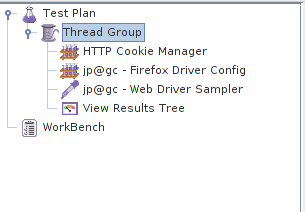
-Via *system.properties*file ,add the next few lines to it

http.proxyHost=proxy\_host  
https.proxyHost=proxy\_host  
http.proxyPort=proxy\_port  
https.proxyPort=proxy\_port

**Use selenium with Jmeter-**

To use Selenium Webdriver with JMeter, simply install "[Webdriver Set](http://jmeter-plugins.org/downloads/all/)" plugins.

From config elements- add jp@gc- Firefox driver config



Write code in webdriver sampler-

WDS.sampleResult.sampleStart();

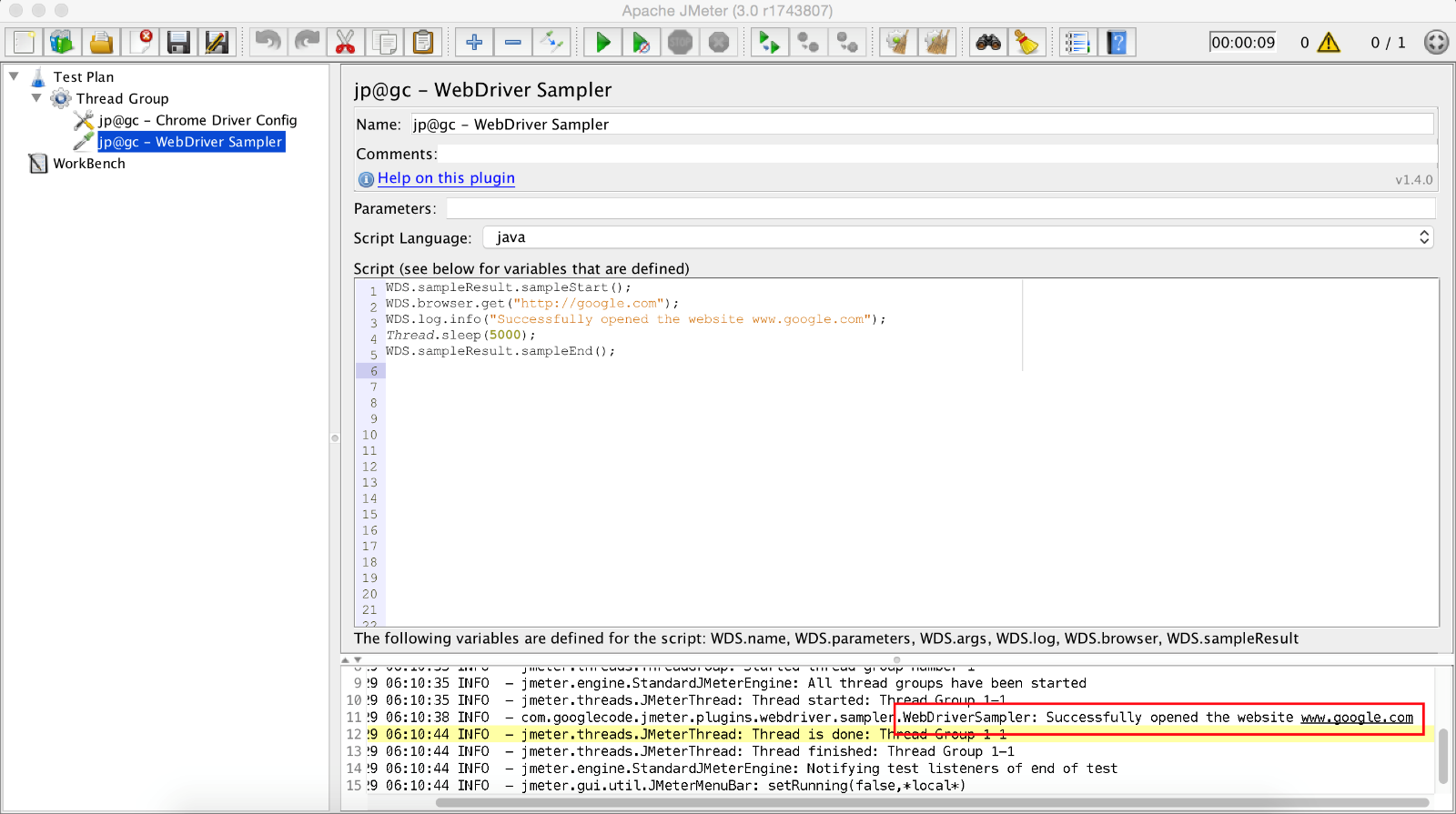
WDS.browser.get("http://google.com");

WDS.log.info("Successfully opened the website www.google.com");

Thread.sleep(5000);

WDS.sampleResult.sampleEnd();

WDS=Webdriver sampler ( as we used driver.get() in selenium)



**Hard page faults** occur when the **page** is not located in physical memory or a memory-mapped file created by the process, making the system go looking for it on the **hard** disk. On the other hand, a **soft page fault** occurs when the **page** is resident elsewhere in memory.

Summary report parameters-

**Label:** It is the name/URL for the specific HTTP(s) Request. If you have selected “Include group name in label?” option then the name of the Thread Group is applied as the prefix to each label.

**#Samples:** This indicates the number of virtual users per request.

**Average:** It is the average time taken by all the samples to execute specific label. In our case, average time for Label 1 is 942 milliseconds & total average time is 584 milliseconds.

**Min:** The shortest time taken by a sample for specific label. If we look at Min value for Label 1 then, out of 20 samples shortest response time one of the sample had was 584 milliseconds.

**Max:** The longest time taken by a sample for specific label. If we look at Max value for Label 1 then, out of 20 samples longest response time one of the sample had was 2867 milliseconds.

**Std. Dev.:** This shows the set of 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 label.

**Error%:** Percentage of Failed requests per Label.

**Throughput:** Throughput is the number of request that are processed per time unit(seconds, minutes, hours) by the server. This time is calculated from the start of first sample to the end of the last sample. Larger throughput is better.

**KB/Sec:** This indicates the amount of data downloaded from server during the performance test execution. In short, it is the Throughput measured in Kilobytes per second.

**Performance testing in mobile--**

**Android proxy configuration**

* We need appropriate JMeter Certificate and save it to our phone. ApacheJMeterTemporaryRootCA.crt can be downloaded from <https://gist.github.com/borisguery/9ef114c53b83e553b635>
* Download zip file for the certificate and send the certificate on mail. And install through mail. Once the certificate is installed phone will ask to apply a lock. And a notification appears showing Network may be monitored.

1. Go to Settings>Wi-Fi option (WiFi network should be same for both Mobile and Laptop/Desktop).

2. Select Wifi and long press. click on Modify network config> Show advanced options.

3. Set ‘Proxy hostname’ as your computer’s IP address and ‘Proxy port’ to 8080 same as tool.

4. Click ‘Save’ option. You can now start running the application on your mobile device and its requests will be recorded in JMeter.

**Handle HTTPS sites**

Add JMeter Certificate Authority to your browser.

For firefox- Go to options> Privacy and security> Certificates> View certificate

Import certificate –‘ ApacheJMeterTemporaryRootCA.crt’ from Jmeter bin folder.

Refer- <http://jmeter.apache.org/usermanual/component_reference.html#HTTP(S)_Test_Script_Recorder>

**Jmeter 4.0 heap size-**

Default-

set HEAP=-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m

New setting-

HEAP=-Xms1024m -Xmx1024m -XX:MaxMetaspaceSize=1024m

Now-

set HEAP=-Xms2048m -Xmx2048m -XX:MaxMetaspaceSize=2048m

### Adding Items to the Script Recorder (Optional)

1. Adding JMeter items to a the HTTP(S) Test Script Recorder will make recorded requests inherit the added item. if we add a Timer item to the Script Recorder, the Timer will be added to each HTTP Request that is recorded.
2. Select *HTTP(S) Test Script Recorder*, then Right-click it
3. Mouse over *Add >*
4. Mouse over *Timers >*
5. Click on *Constant Timer*

Either give fixed delay time or varying by user action.

For varying time to each sampler, change the “Thread Delay” on the timer to “${T}”. This will tell the proxy server to record your time rather than add a constant time for each request.

<http://localhost:9090/node?name=admin&pass=admin> application authentication method

<http://admin:admin@localhost:9090/> server basic authentication method.

**Report dashboard generation-** Jmeter3.0 onwards

Add following lines in user.properties file –

------------------------------------------------add in user.properties---------------------------------------------

jmeter.save.saveservice.bytes=true

jmeter.save.saveservice.label=true

jmeter.save.saveservice.latency=true

jmeter.save.saveservice.response\_code=true

jmeter.save.saveservice.response\_message=true

jmeter.save.saveservice.successful=true

jmeter.save.saveservice.thread\_counts=true

jmeter.save.saveservice.thread\_name=true

jmeter.save.saveservice.time=true

jmeter.save.saveservice.timestamp\_format=ms

jmeter.save.saveservice.timestamp\_format=yyyy/MM/dd HH:mm:ss

--------------------------------------------------------------------end----------------------------------------------

Run script in jmeter and create jtl file.

Write following command to create dashboard from jtl result file

jmeter -g /path/to/jtl/file -o /where/you/want/to/store/dashboard

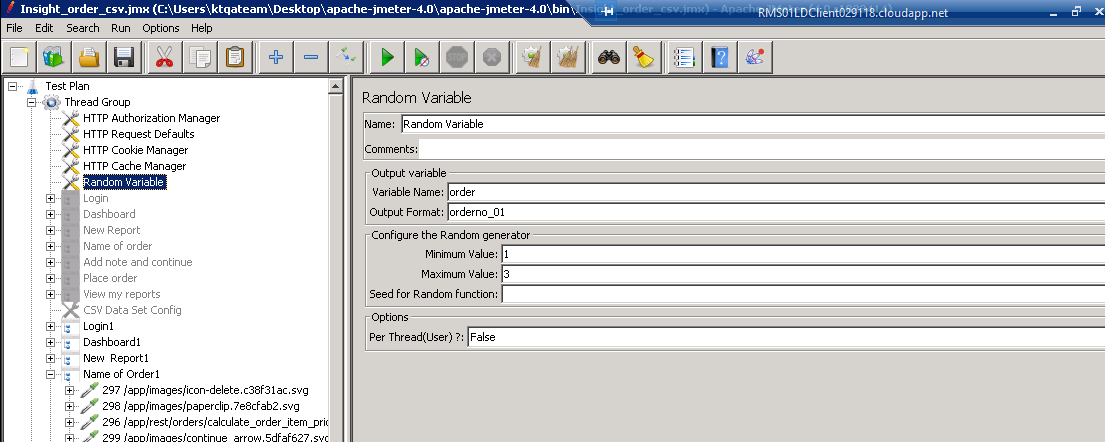
To save dashboard result while running script,

jmeter -n -t <path\_to.jmx> -l <log.jtl> -e -o <dashboard\_folder>

Results will be stored in html pages

**Use of Random variable- config element**

To use random variables in parameterisation, we use random variable



Pass this variable value wherever to apply using ${order}

Output format defines format of variable we want to use, in this case it will be generated as orderno\_21, orderno\_32,..

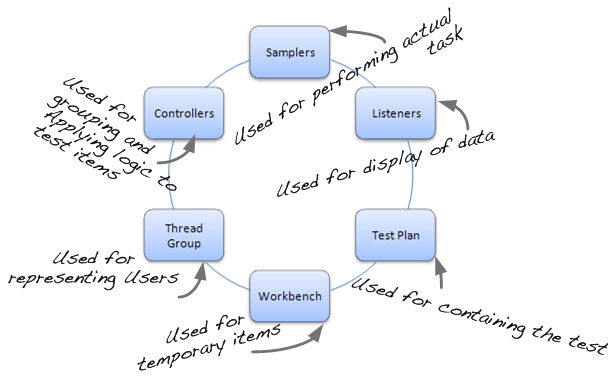
Printout till here---------------------------------------------------------------------------------------------------------------------------

**To pass encoded tokens in URL** while using regular expression extractor-

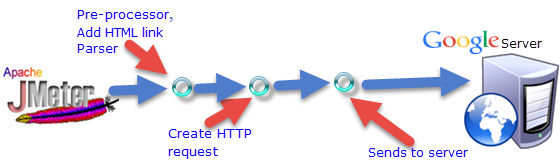
Lets say, For variable ‘contoken’,

Use- ${\_\_javaScript(encodeURIComponent('${contoken}'))}

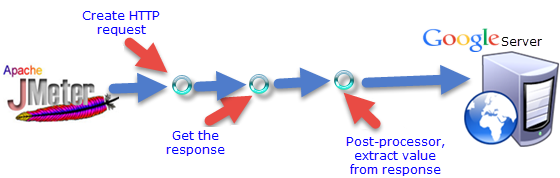
* Run API testing through Load testing
* Run scripts through Jenkins
* CSV + regular expression - <https://dzone.com/articles/advanced-load-testing-scenarios-with-jmeter>



Preprocessor-



Post processor



Parameterization can be done using:-

* Pre-processor elements> User parameters
* Config element> CSV data set config

Correlation can be done using:-

* Post processor elements like- Regular expression extractor, Json extractor, Xpath extractor, CSS/Query extractor

Listener-

Simple Data writer- To specify filename in which you want result to be saved.

Connect time in listener is connection establishment time including SSL handshake

Assertion listener- It list down the samplers if it is successful, if fail, details are also provided.

3 ways to record script-

* Using Badboy tool
* Using blazemeter Chrome plugin
* Using Jmeter HTTP test script recorder element

Octoperf tool- Paid tool for writing scripts, results

Timers-

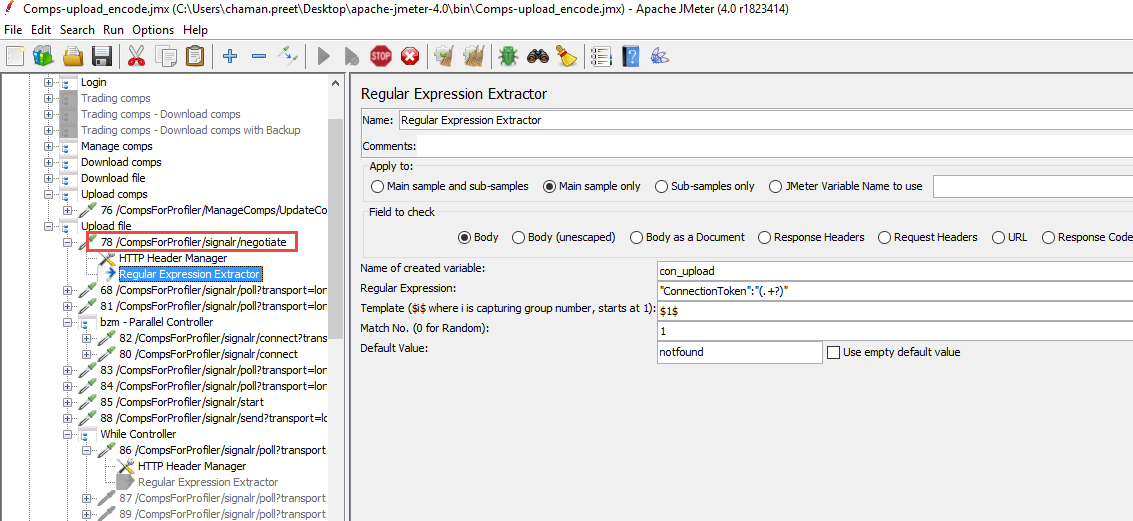
* Think time is pause time between web requests to bring realistic behaviour.
* Timer elements are executed before execution of each sampler in its scope.
* To execute timer before specific sampler, add timer as its child element
* To execute timer after specific sampler, add timer as child of next sampler or child of test action sampler.
* Timer values are decided based on assumption by number of fields on page, time server takes to respond..
* Common timers are constant timer, uniform random timer, Constant throughput timer, synchronizing timer.

Constant throughput timer- To delay the threads in order to meet pre-defined throughput target, to run test to meet specific throughput goal.

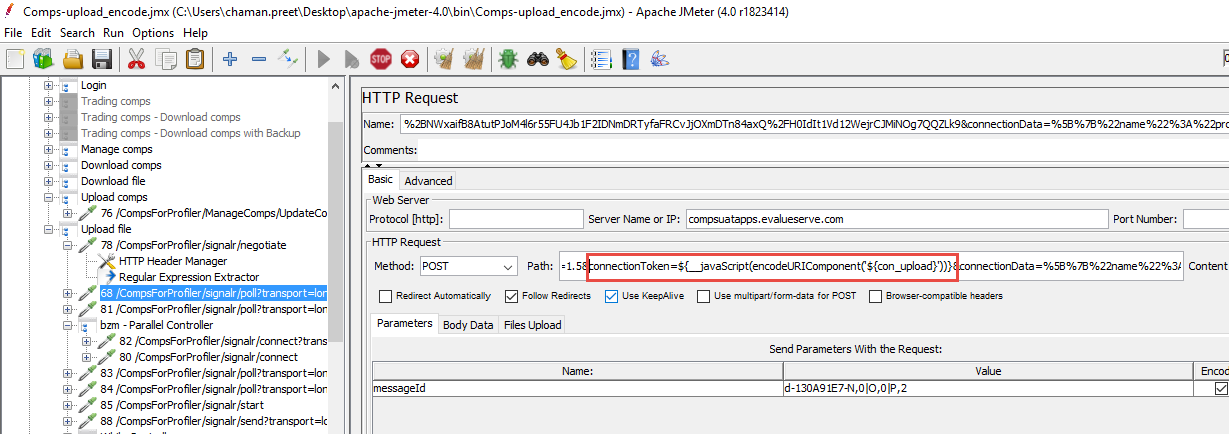
Synchronising timer- used to pause configured number of threads in order to create a rendezvous transaction, to fire the sampler requests exactly at same time. It pauses all the threads until the configured number of threads become active and fire the sampler requests exactly at the same time.

**Handling SignalR application-**

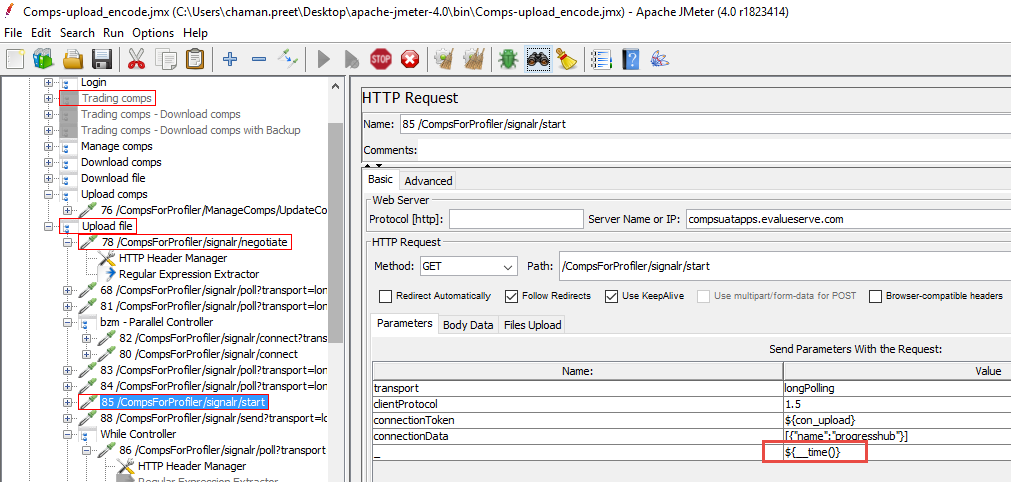
1) Apply regular expression extractor in negotiate step to capture connection token which can be used in subsequent requests.



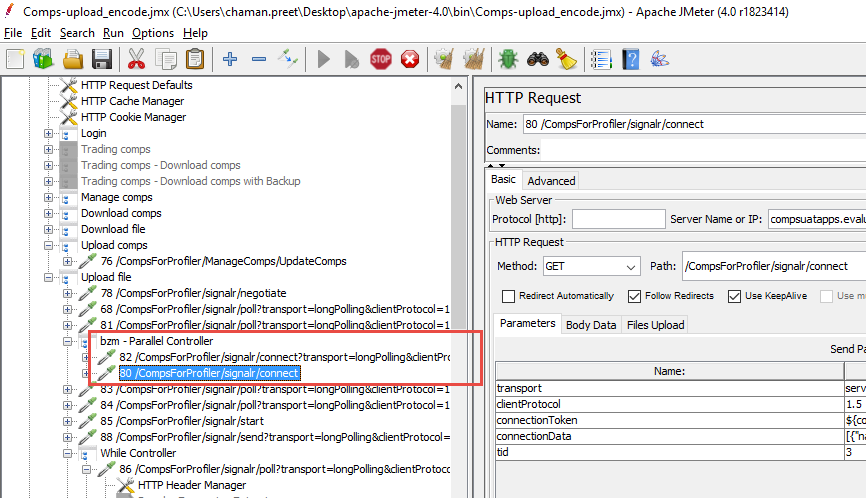
2) Use connection token in encoded form in URLs-



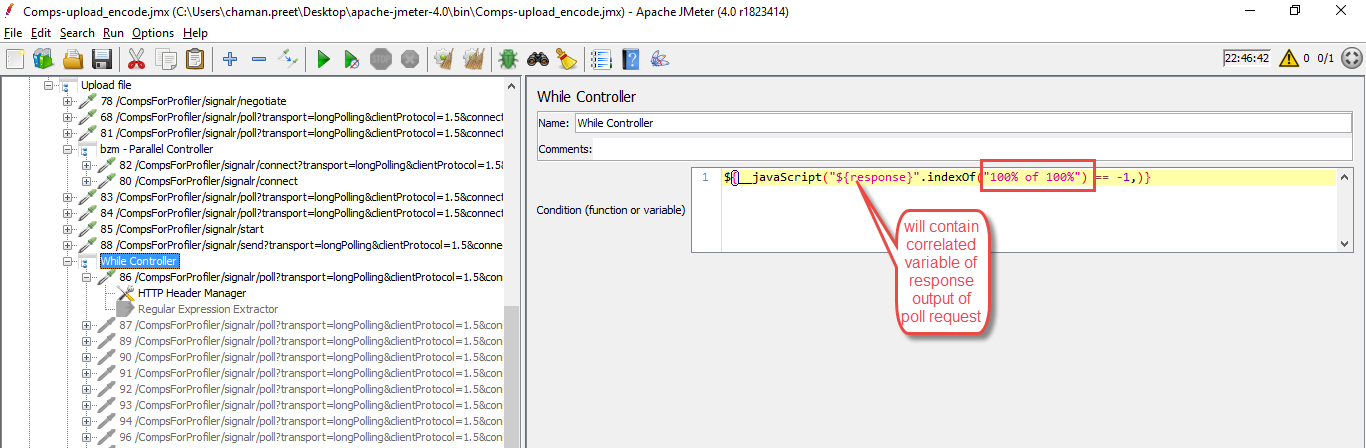
3) Use function ${\_\_time()} for timestamp wherever used- negotiate step and start step-



4) Apply parallel controller (add plugin) to handle asynchronous request of connect which otherwise hangs and give timeout exception.



5) While controller to capture multiple polling requests to server with condition until file is uploaded with status 100% of 100%



**API testing using Jmeter-**

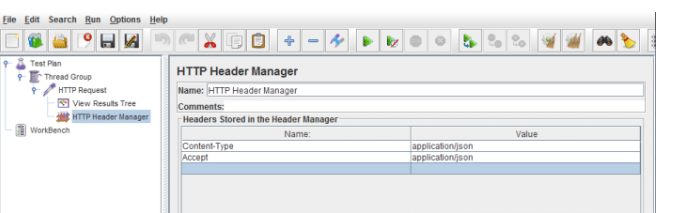
<https://www.testingexcellence.com/jmeter-tutorial-testing-rest-web-services/>

Jmeter to send Json request to a RESTful Web Service and also parse the Json response.

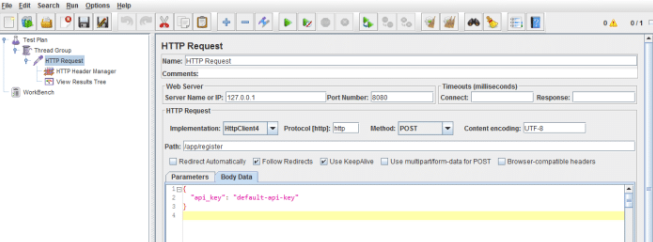
Overall we need to-

1. Handle **Rest API Login** using an [Http POST Request](https://en.wikipedia.org/wiki/POST_(HTTP)),
2. [Extract Variables from a Json Response](https://octoperf.com/blog/2017/03/09/how-to-extract-data-from-json-response-using-jmeter/) and reuse it later in the script,
3. And verify Json responses using [JMeter Json Assertion](https://octoperf.com/blog/2018/04/19/jmeter-assertions/#json-assertion)

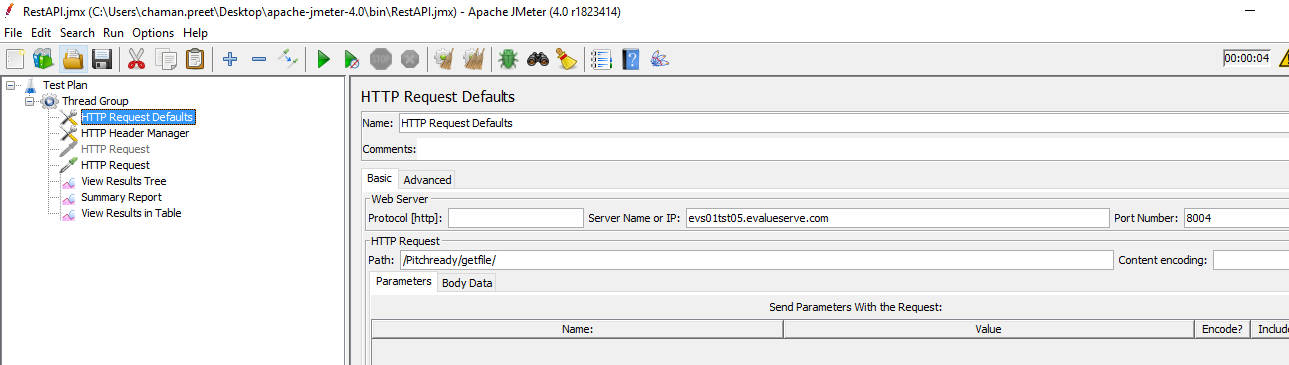
* Add thread group, HTTP request with API path in server name, give error as REST API might expects “Content-Type” and “Access” parameters in the header request.
* So Also add HTTP header manager and send parameters- “Content-Type” and “Access” variables

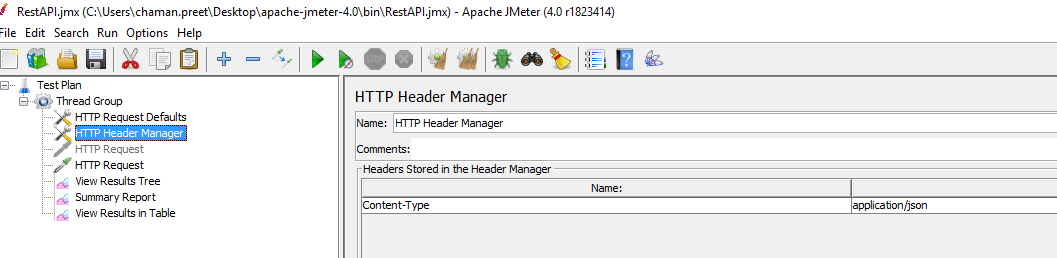


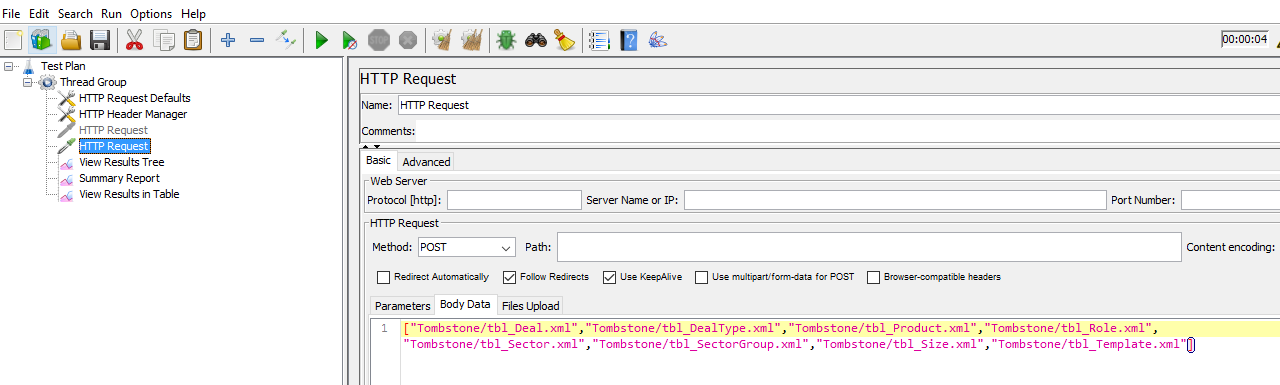
* you need to register your application via an API key. Send it in **body of the request** in POST method



**Simple API testing we did-**

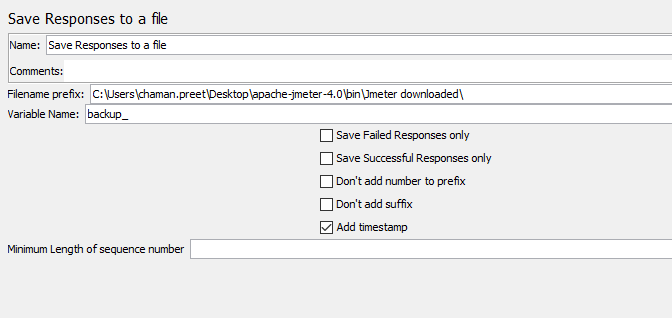


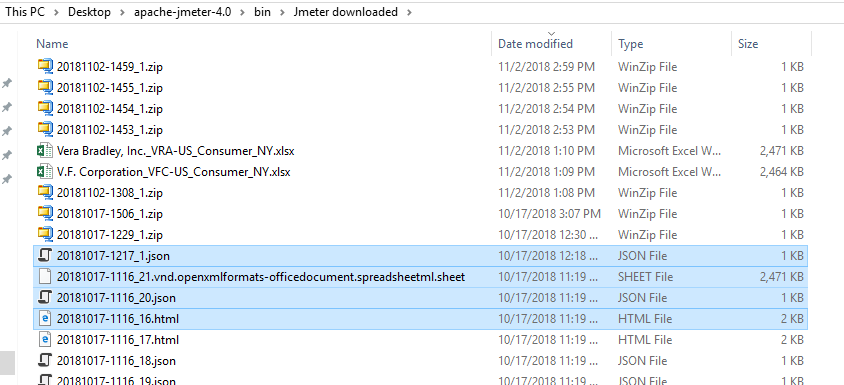




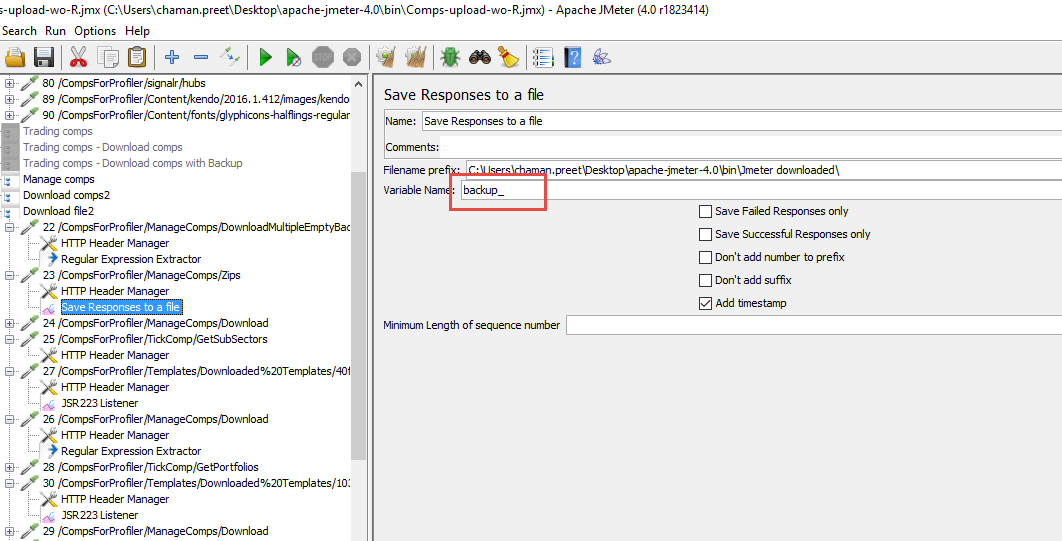
**Save downloaded file-**

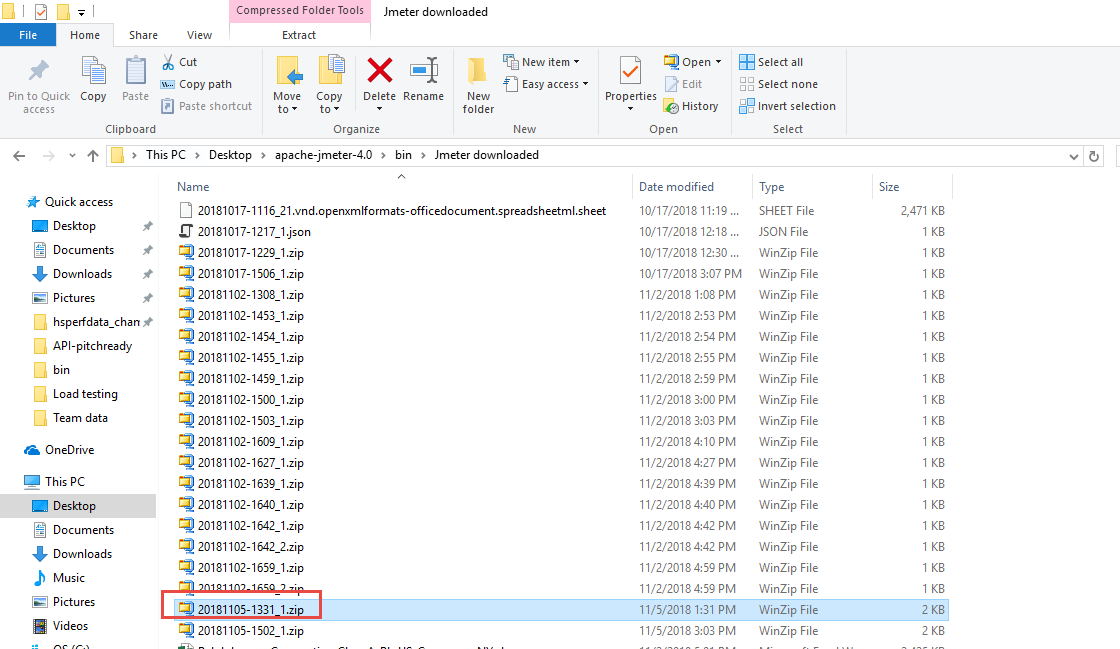
Use listener in individual sampler or Controller- *Save Responses to a file*, it will save results in form of HTML, JSON and sheet format.





**To save zip file, Save responses to a file can be used.**

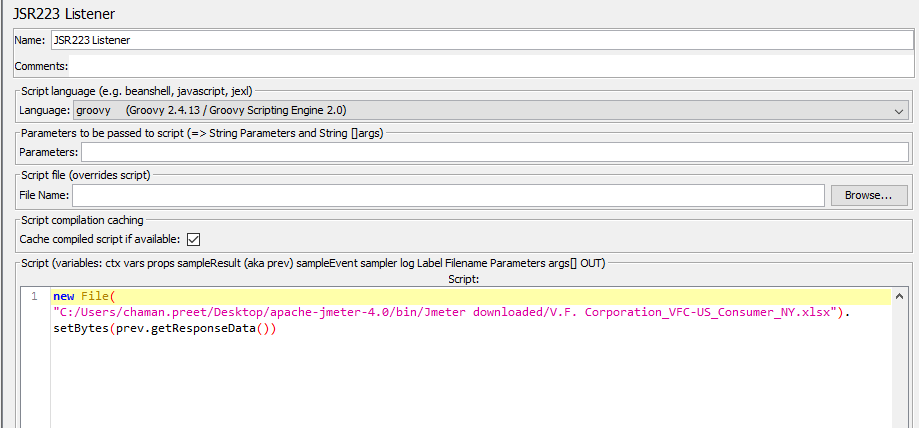




**Save file in same format as downloaded-**

Use listener*- JSR223 Listener*, choose groovy language and add script like (in whatever format you want to store)-

new File("C:/Users/chaman.preet/Desktop/apache-jmeter-4.0/bin/Jmeter downloaded/Vera Bradley, Inc.\_VRA-US\_Consumer\_NY.xlsx").setBytes(prev.getResponseData())



**Distributed testing-**

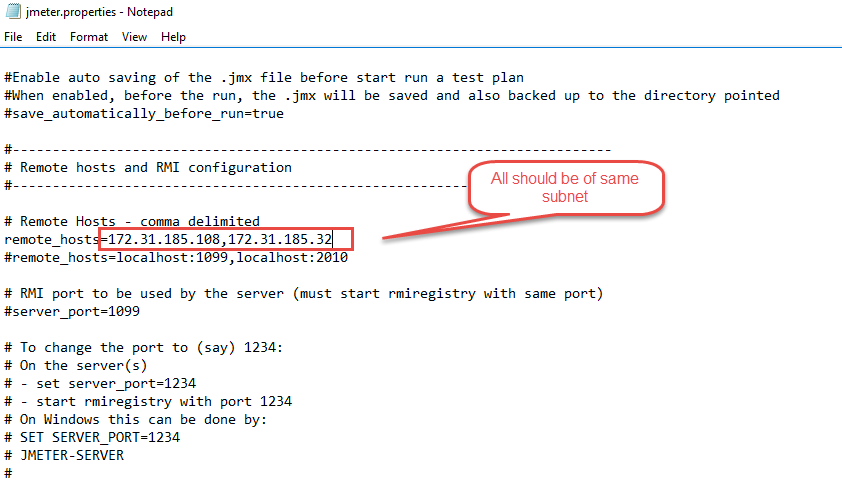
Prerequisite-

* Make sure that all the nodes (client- master and servers- slaves) are running exactly the same version of JMeter
* There should not be any firewall
* Preferred-all nodes are using the same version of Java on all systems
* have a [valid keystore for RMI over SSL](https://jmeter.apache.org/usermanual/remote-test.html#setup_ssl), or you have disabled the use of SSL.

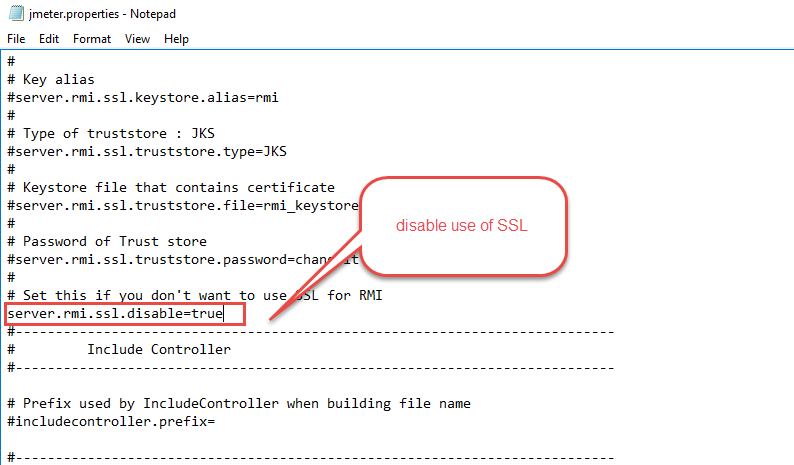
**Steps-**

1) On all machines, increase heap size in jmeter.bat file

2) In Jmeter.properties file of master, add remote hosts



3) Disable SSL if not set valid RMI in all nodes



4) On slave machines, run jmeter-server.bat using command-

jmeter-server -Djava.rmi.server.hostname=10.100.180.29 -Jserver.rmi.ssl.disable=true

Djava.rmi.server.hostname is required if default local IP address is different from what we are using. This can be checked in jmeter-server.log file.

5) Now, run test on master by using command-

jmeter -n -t script.jmx –l result.jtl -r

or

jmeter -n -t script.jmx –l result.jtl -R server1,server2,…

Difference between \_\_CSVRead and \_\_Stringfromfile function is that Stringfromfile should be used when different data are required in a loop for same thread and CSVRead should be used when different data are required for different threads(users).

Client authentication modes-

Kerberos — This is the most secure protocol because it establishes mutual authentication between the client and the server using an encrypted shared key. This protocol requires additional configuration and the appliance will silently downgrade to NTLM if Kerberos is not set up properly or if the client cannot do Kerberos. For more information, see "Preparing for a Kerberos Deployment" on page 1203.

NTLM (window authentication)— Uses an encrypted challenge/response that includes a hash of the password. NTLM requires two trips between the workstation and the appliance, and one trip between the appliance and the Domain Controller (DC). It therefore puts more load on the network than Kerberos, which only requires one trip between the workstation and the appliance, and doesn’t require a trip between the appliance and the DC.

Basic — Prompts the user for a username and password to authenticate the user against the Windows Active Directory.

JMeter doesn’t support Digest Authentication yet. The authorization manager works great for Basic and Kerberos authentication thought.

No one fully supports NTLM fully except Microsoft due to its proprietary nature.

Authentication type can be found in response headers.

http://automationwithselenium.blogspot.com/2012/12/ntlm-authentication-aka-windows.html