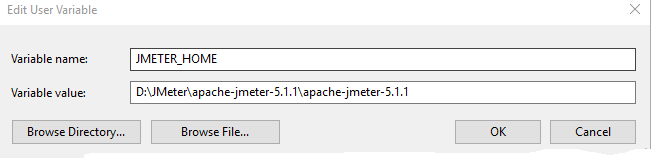
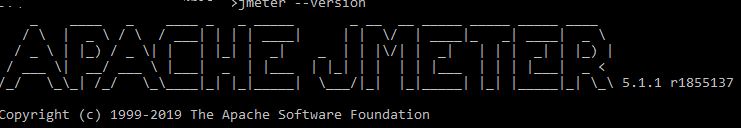
*Note: JMeter Concepts are known - sampler, assertion, Timer, Logic Controller, Config Element, or any JMeter components.*

The idea is, picking an HTTP sampler request for REST API test, and how we can use it by transforming in different variety of execution starting from ***JMeter Tool to Cucumber JVM execution using IDE***.

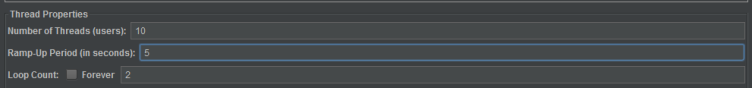
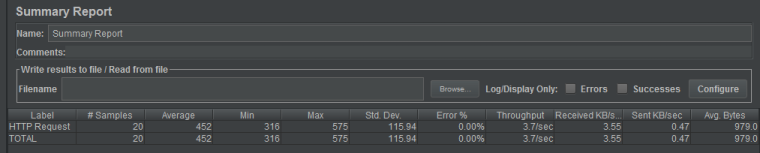
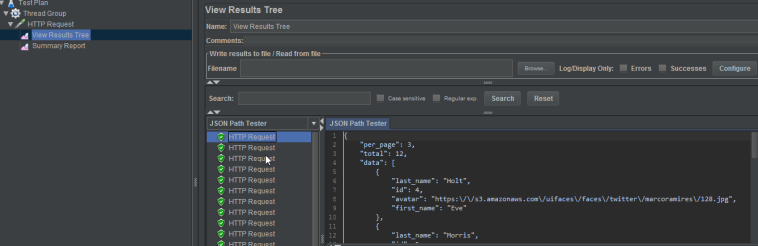
* JMeter Pre-requisites & Setup
* How to run an HTTP sampler performance test using JMeter Tool
* How to run an HTTP sampler performance test using NON-GUI Mode
* How to run an HTTP sampler performance test using Maven
* How to run an HTTP sampler performance test using JMeter API (Java Code)
* How to run an HTTP sampler performance test using Cucumber JVM
  + Run an HTTP sampler Performance test using JMX file by Cucumber JVM
  + Run an HTTP sampler Performance test using JMeter API by Cucumber JVM

JMeter Pre-requisites & Setup

1. Download JMeter from [here](https://jmeter.apache.org/download_jmeter.cgi)
2. 
3. Click on Zip folder for Windows installer
4. Copy the zip folder in a specific folder location and unzip it.
5. Set Environment variable for JMeter Downloaded location and Add Path variable as mentioned below snapshots003_JMeterPath
6. Open command prompt and supply command “jmeter –version” to verify current version gets installed.

How to run an HTTP sampler performance test using JMeter Tool

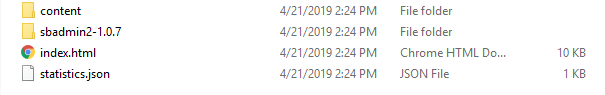
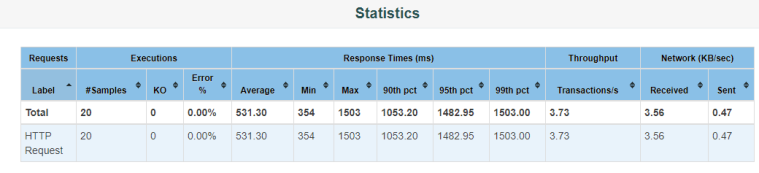
1. Open “jmeter.bat” under bin folder where you extracted JMeter Zip.
2. Right-click on Test Plan -> Add->Thread (Users)->Thread Group
3. Right click on Thread Group->Sampler->HTTP Request
4. Add https in protocol (In case https otherwise leave default)
5. Enter Host or ip address in Server Name or IP field
6. Select Request type, add Rest of resource path in Path field like below snapshot
7. Graphical user interface, application

   Description automatically generated
8. Right click on “HTTP Request”->Add->Listeners->View Results Tree
9. Right click on “HTTP Request”->Add->Listeners->Summary Report
10. Save Test Plan (The extension of JMeter file is JMX
11. Set Through group value as mentioned in the below snapshot
    * For each second 2 users hit the endpoint in parallel that complete executing 10 users in 5 secs which repeat 2 times, so overall 20 users hit this endpoint in 10 seconds.
12. Click on Run 006_Run_Button button
13. After a run, we can see the following results in JMeter

With these steps, we ran and understood how to do simple REST API test using HTTP Sampler in JMeter.

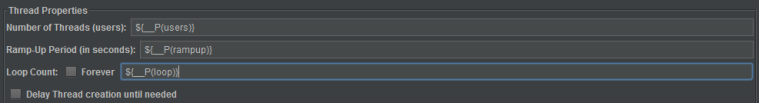
How to run an HTTP sampler performance test using NON-GUI Mode

For this test, I’m going to save as the same JMX file which we used in the previous section. In this section, I will show you how to run the same test in NON-GUI Mode but for this test, I will disable all listeners (View Result Tree and Summary Report) and generate remote using NON-GUI Mode.

1. Go to JMX File stored location in command prompt.010_Generate_Report
2. Supply the following command
   * *jmeter -n -t HTTP\_Non\_GUI\_Mode.jmx -l D:\JMeter\JMeter\_Examples\test\_result\jtlresult.jtl -e -o D:\JMeter\JMeter\_Examples\test\_result\report*
     + -n  : This specifies JMeter is to run in non-gui mode
     + -t   : name of JMX file that contains the Test Plan
     + -l   : name of JTL file to log sample results to
     + -e: generate report dashboard after load test
     + -o: output folder for report dashboard
3. After running the test, go to a specific folder where reports get generated successfully.
4. 011_Report_JTL
5. 
6. 
7. 

How to Run JMeter Test in NON-GUI Mode by assigning  Runtime value

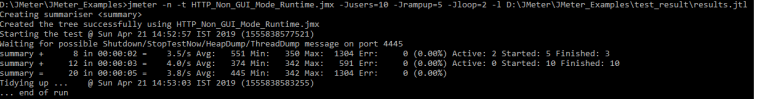
In this example, I use the same JMX file in the previous section by saving as a test plan, modify the following parameters and save it.



\_\_P is for property value which is allowed during runtime for eg., command line execution.

**Supply the following command by assigning runtime value for the variables**

*jmeter -n -t HTTP\_Non\_GUI\_Mode\_Runtime.jmx -Jusers=10 -Jrampup=5 -Jloop=2 -l D:\JMeter\JMeter\_Examples\test\_result\results.jtl*



-J: [prop\_name]=[value]defines a local JMeter property.

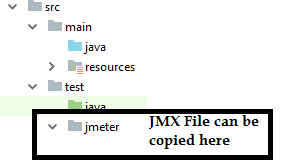
How to run an HTTP sampler performance test using Maven

Pre-requisites:

* Please make sure to download Maven standalone zip file, add environment variable and path.

JMeter can be integrated with [Apache Maven](http://maven.apache.org/) via the [JMeter Maven Plugin](http://wiki.apache.org/jmeter/JMeterMavenPlugin).  In this section, I’m going to explain how to run existing JMX file using Maven Integration.

In order to understand by Maven plugin, the Maven project structure must be created in the following format.



the jmeter folder must exist under test folder, all JMX files can be copied under jmeter folder or its subfolder.

To demonstrate further, I’m going to create Maven project in Intellij and add the following details in pom.xml

<build>

<plugins>

<plugin>

<groupId>com.lazerycode.jmeter</groupId>

<artifactId>jmeter-maven-plugin</artifactId>

<version>2.5.1</version>

<executions>

<execution>

<id>jmeter-tests</id>

<phase>verify</phase>

<goals>

<goal>jmeter</goal>

</goals>

</execution>

</executions>

<configuration>

<jMeterProcessJVMSettings>

<xms>1024</xms>

<xmx>1024</xmx>

<arguments>

<argument>-Xprof</argument>

<argument>-Xfuture</argument>

</arguments>

</jMeterProcessJVMSettings>

<propertiesUser>

<users>${users}</users>

<rampup>${rampup}</rampup>

<loop>${loop}</loop>

</propertiesUser>

<testFilesIncluded>

<jMeterTestFile>${jMeterTestFile}</jMeterTestFile>

</testFilesIncluded>

<testResultsTimestamp>false</testResultsTimestamp>

</configuration>

</plugin>

<plugin>

<groupId>de.codecentric</groupId>

<artifactId>jmeter-graph-maven-plugin</artifactId>

<version>0.1.0</version>

<configuration>

<inputFile>${project.build.directory}/jmeter/results/jmetereg\_HTTP\_Non\_GUI\_Mode\_Runtime.jtl</inputFile>

<graphs>

<graph>

<pluginType>ResponseTimesOverTime</pluginType>

<width>800</width>

<height>600</height>

<outputFile>${project.build.directory}/jmeter/results/jmetereg\_HTTP\_Non\_GUI\_Mode\_Runtime.png</outputFile>

</graph>

</graphs>

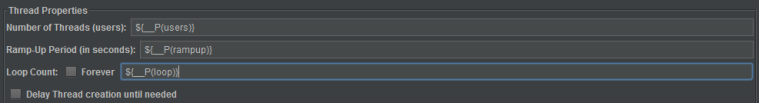
</configuration>

</plugin>

</plugins>

</build>

In our previous example, we kept the following parameters as a runtime variable. Hence I use the same format to show you how to run this in maven command



*mvn clean verify -Dusers=10 -Drampup=5 -Dloop=2 -DjMeterTestFile=jmetereg/HTTP\_Non\_GUI\_Mode\_Runtime.jmx*

Performance test gets succeeded and you can see the results under the following location

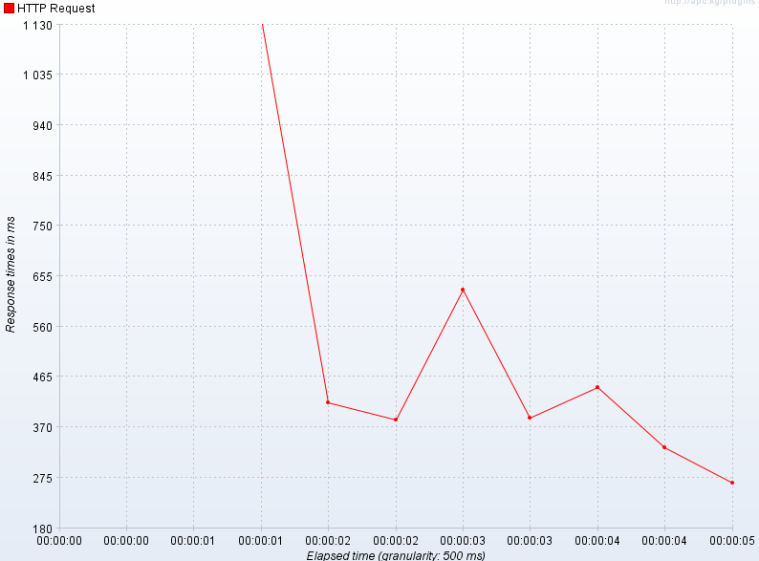
* ***[Maven\_Project]/target/jmeter/results***

After a successful result, supply the following command to generate a graph

mvn jmeter-graph:create-graph

The JPG file gets generated in the same results folder

* ***[Maven\_Project]/target/jmeter/results***



How to run an HTTP sampler performance test using JMeter Java API

This section is mainly focused on how to run a performance test using pure java code and execute existing JMX file using Java code.

I’m going to explain how to create a JMeter test using Java code in an easy way by comparing the JMX File. In below example, I opened one of the existing JMX files which were used in the previous section in this blog.



Take a close look at the above snapshot taken from JMX file. <hashTree>  is associated with Test Plan and Test Plan elements. So, the same logic used to implement in Java API.

To run JMeter using Java API, here are the following or minimum test plan elements/controller required.

1. [StandardJMeterEngine](http://jmeter.apache.org/api/org/apache/jmeter/engine/StandardJMeterEngine.html) – This the main class which actually helps you to configure your test plan and execute.
2. [HashTree](https://jmeter.apache.org/api/org/apache/jorphan/collections/HashTree.html) – Holds Test Plan element for eg., Test Plan, Thread Group, Http Sampler, Loop Controller

**Test Plan Elements**

* [TestPlan](http://jmeter.apache.org/api/org/apache/jmeter/testelement/TestPlan.html" \t "_blank) – This must be mandatory which exists top in JMeter, in fact by default Test Plan exists by default if you open in JMeter

**Example Code:**

public static void initializeTestPlan(String testPlanName) {

testPlan = new TestPlan(testPlanName);

testPlan.setProperty(TestElement.TEST\_CLASS, TestPlan.class.getName());

testPlan.setProperty(TestElement.GUI\_CLASS, TestPlanGui.class.getName());

testPlan.setUserDefinedVariables((Arguments) new ArgumentsPanel().createTestElement());

}

[LoopController](https://jmeter.apache.org/api/org/apache/jmeter/control/LoopController.html) – Since you must have at least one loop, it’s essential to have a Loop Controller instance set as the main Sampler controller for a Thread Group.   **Example Code:**

public static void setLoopController(int loopCount) {

loopController = new LoopController();

loopController.setLoops(loopCount);

loopController.setFirst(true);

loopController.setProperty(TestElement.TEST\_CLASS, LoopController.class.getName());

loopController.setProperty(TestElement.GUI\_CLASS, LoopControlPanel.class.getName());

loopController.initialize();

LOG.info("Initalized Loop Controller");

}

* [ThreadGroup](https://jmeter.apache.org/api/org/apache/jmeter/threads/ThreadGroup.html) – At least one thread group must be present in Test Plan which needs to be configured no of  users, Ramp-up period and loop count.

**Example Code:**

public static void setThreadGroup(int noOfThreads, int setRamupNo) {

threadGroup = new ThreadGroup();

threadGroup.setName("Thread Group");

threadGroup.setNumThreads(noOfThreads);

threadGroup.setRampUp(setRamupNo);

threadGroup.setSamplerController(loopController);

LOG.info("Initialized Thread Group");

threadGroup.setProperty(TestElement.TEST\_CLASS, ThreadGroup.class.getName());

threadGroup.setProperty(TestElement.GUI\_CLASS, ThreadGroupGui.class.getName());

}

* A [Sampler](https://jmeter.apache.org/api/org/apache/jmeter/samplers/Sampler.html) do the actual work for eg., http sampler, ftp request, jdbc request

**Example Code:**

public static void setHttpSampler(String protocolType, String setDomainName, int setPort, String setPath, String requestType) {

httpSamplerProxy = new HTTPSamplerProxy();

httpSamplerProxy.setDomain(setDomainName);

httpSamplerProxy.setPort(setPort);

httpSamplerProxy.setPath(setPath);

httpSamplerProxy.setMethod(requestType);

httpSamplerProxy.setProtocol(protocolType);

httpSamplerProxy.setName("Http Sampler");

LOG.info("Initalized Http Sampler");

httpSamplerProxy.setProperty(TestElement.TEST\_CLASS, HTTPSamplerProxy.class.getName());

httpSamplerProxy.setProperty(TestElement.GUI\_CLASS, HttpTestSampleGui.class.getName());

}

**To configure Test Plan and Test Plan Element in HashTree:**

**Example Code:**

public static HashTree configureTestPlan() {

testplanTree.add("TestPlan", testPlan);

testplanTree.add("loopController", loopController);

testplanTree.add("ThreadGroup", threadGroup);

testplanTree.add("HTTPSamplerProxy", httpSamplerProxy);

return testplanTree;

}

After all initialization and configuration, it’s time to run JMeter Performance test using StandardJMeterEngine. The below lines are important code to run a performance test in Java

static {

jMeterEngine = new StandardJMeterEngine();

}

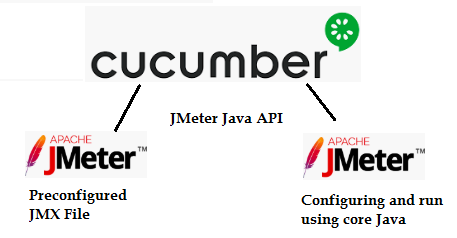
jMeterEngine.configure(testplanTree);

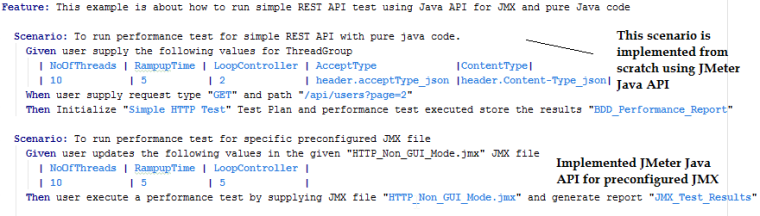
jMeterEngine.run();

So, to look on final code, please go visit bitbucket repository ([JMeterPerformanceFramework](https://github.com/automationcalling/jmeterperformancecode" \t "_blank) ) where JMeterUtil package under src/main/java.

How to run an HTTP sampler performance test using Cucumber JVM

In this section, I’m going to build a simple cucumber framework that helps you to configure all runtime properties in the feature file and run more robust as well as more flexible performance test.





**First scenario:**

* It’s pure Java code (JMeter Util implemented in Step Definition).
* Configuring runtime properties like Threads, Ramp up time and Loop Controller
* Execute test and generate a report in the target folder.

**Second Scenario:**

* The second scenario, I use the same JMeterUtil code but the only difference is, not configured all test plan element in scratch using Java, instead, use pre-configured JMX file.
* Configuring runtime properties like Threads, Rampuptime and Loop Controller – Used XML util to update the value in JMX file.
* Execute test and generate a report in the target folder.

