JMeter

jMeter is an Open Source testing software. It is 100% pure Java application for load and performance testing. jMeter is designed to cover categories of tests like load, functional, performance, regression, etc., and it requires JDK 5 or higher.

Before going into the details of JMeter, let us first understand a few jargons associated with the testing of any application.

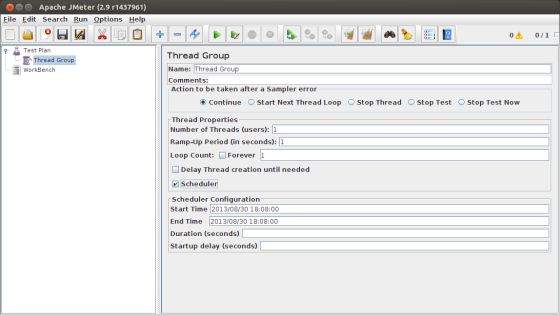
* **Performance Test** − This test sets the best possible performance expectation under a given configuration of infrastructure. It also highlights early in the testing process if any changes need to be made before the application goes into production.
* **Load Test** − This test is basically used for testing the system under the top load it was designed to operate under.
* **Stress Test** − This test is an attempt to break the system by overwhelming its resources.

## Thread Group

Thread Group elements are the beginning points of your test plan. As the name suggests, the thread group elements control the number of threads JMeter will use during the test. We can also control the following via the Thread Group −

* Setting the number of threads
* Setting the ramp-up time
* Setting the number of test iterations

The Thread Group Control Panel looks like this −



The Thread Group Panel holds the following components −

* **Action to be taken after a Sampler error** − In case any error occurs during test execution, you may let the test either −
  + **Continue** to the next element in the test
  + **Stop Thread** to stop the current Thread.
  + **Stop Test** completely, in case you want to inspect the error before it continues running.
* **Number of Threads** − Simulates the number of users or connections to your server application.
* **Ramp-Up Period** Defines how long it will take JMeter to get all threads running.
* **Loop Count** − Defines the number of times to execute the test.
* **Scheduler checkbox** − Once selected, the Scheduler Configuration section appears at the bottom of the control panel.
* **Scheduler Configuration** − You can configure the start and end time of running the test.

## Controllers

JMeter has two types of Controllers − *Samplers* and *Logic Controllers*.

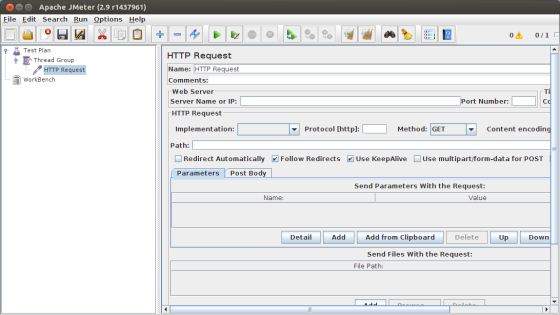
### Samplers

Samplers allow JMeter to send specific types of requests to a server. They simulate a user request for a page from the target server. For example, you can add a HTTP Request sampler if you need to perform a POST, GET, or DELETE on a HTTP service.

Some useful samplers are −

* HTTP Request
* FTP Request
* JDBC Request
* Java Request
* SOAP/XML Request
* RPC Requests

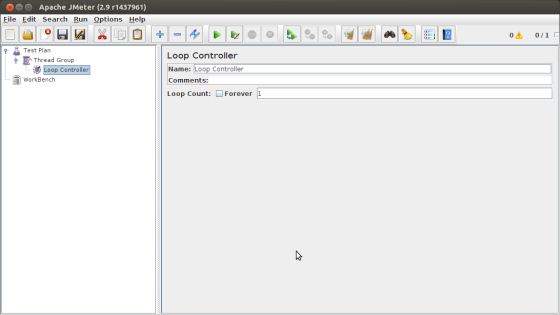
The following screenshot shows an HTTP Request Sampler Control Panel −



### Logic Controllers

Logic Controllers let you control the order of processing of Samplers in a Thread. Logic controllers can change the order of a request coming from any of their child elements. Some examples are − ForEach Controller, While Controller, Loop Controller, IF Controller, Run Time Controller, Interleave Controller, Throughput Controller, and Run Once Controller.

The following screenshot shows a Loop Controller Control Panel −



The following list consists of all the Logic Controllers JMeter provides −

* Simple Controller
* Loop Controller
* Once Only Controller
* Interleave Controller
* Random Controller
* Random Order Controller
* Throughput Controller
* Runtime Controller
* If Controller
* While Controller
* Switch Controller
* ForEach Controller
* Module Controller
* Include Controller
* Transaction Controller
* Recording Controller

### Test Fragments

A Test Fragment is a special type of element placed at the same level as the Thread Group element. It is distinguished from a Thread Group in that it is not executed unless it is referenced by either a Module Controller or an Include\_Controller. This element is purely for code re-use within Test Plans.

## Listeners

Listeners let you view the results of Samplers in the form of tables, graphs, trees, or simple text in some log files. They provide visual access to the data gathered by JMeter about the test cases as a Sampler component of JMeter is executed.

Listeners can be added anywhere in the test, including directly under the test plan. They will collect data only from elements at or below their level. The following list consists of all the Listeners JMeter provides −

* Sample Result Save Configuration
* Graph Full Results
* Graph Results
* Spline Visualizer
* Assertion Results
* View Results Tree
* Aggregate Report
* View Results in Table
* Simple Data Writer
* Monitor Results
* Distribution Graph (alpha)
* Aggregate Graph
* Mailer Visualizer
* BeanShell Listener
* Summary Report

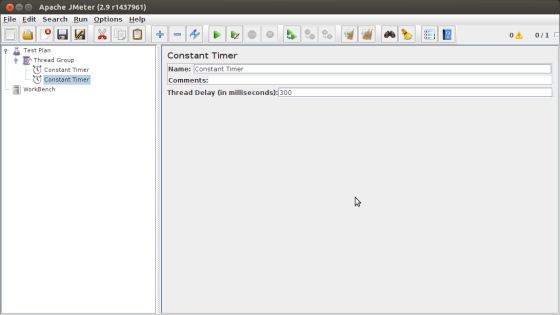
## Timers

By default, a JMeter thread sends requests without pausing between each sampler. This may not be what you want. You can add a timer element which allows you to define a period to wait between each request.

The following list shows all the timers that JMeter provides −

* Constant Timer
* Gaussian Random Timer
* Uniform Random Timer
* Constant Throughput Timer
* Synchronizing Timer
* JSR223 Time
* BeanShell Time
* BSF Time
* Poisson Random Time

The following screenshot shows a Constant Timer Control Panel −



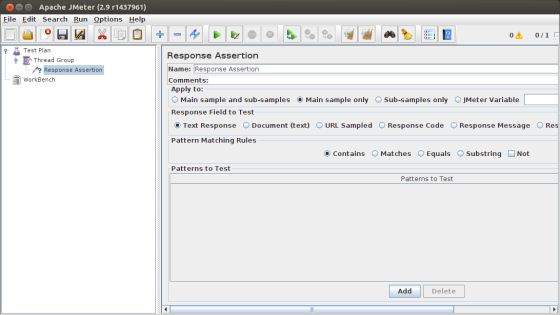
## Assertions

Assertions allow you to include some validation test on the response of your request made using a Sampler. Using assertions you can prove that your application is returning the correct data. JMeter highlights when an assertion fails.

The following list consists of all the assertions JMeter provides −

* Beanshell Assertion
* BSF Assertion
* Compare Assertion
* JSR223 Assertion
* Response Assertion
* Duration Assertion
* Size Assertion
* XML Assertion
* BeanShell Assertion
* MD5Hex Assertion
* HTML Assertion
* XPath Assertion
* XML Schema Assertion

The following screenshot shows a Response Assertion Control Panel −



## Configuration Elements

Configuration Elements allow you to create defaults and variables to be used by Samplers. They are used to add or modify requests made by Samplers.

They are executed at the start of the scope of which they are part, before any Samplers that are located in the same scope. Therefore, a Configuration Element is accessed only from inside the branch where it is placed.

The following list consists of all the Configuration Elements that JMeter provides −

* Counter
* CSV Data Set Config
* FTP Request Defaults
* HTTP Authorization Manager
* HTTP Cache Manager
* HTTP Cookie Manager
* HTTP Proxy Server
* HTTP Request Defaults
* HTTP Header Manager
* Java Request Defaults
* Keystore Configuration
* JDBC Connection Configuration
* Login Config Element
* LDAP Request Defaults
* LDAP Extended Request Defaults
* TCP Sampler Config
* User Defined Variables
* Simple Config Element
* Random Variable

## Pre-processor Elements

A pre-processor element is something that runs just before a sampler executes. They are often used to modify the settings of a Sample Request just before it runs, or to update variables that are not extracted from response text.

The following list consists of all the pre-processor elements that JMeter provides −

* HTML Link Parser
* HTTP URL Re-writing Modifier
* HTTP User Parameter Modifier
* User Parameters
* JDBC PreProcessor
* JSR223 PreProcessor
* RegEx User Parameters
* BeanShell PreProcessor
* BSF PreProcessor

## Post-processor Elements

A post-processor executes after a sampler finishes its execution. This element is most often used to process the response data, for example, to retrieve a particular value for later use.

The following list consists of all the Post-Processor Elements JMeter provides −

* Regular Expression Extractor
* XPath Extractor
* Result Status Action Handler
* JSR223 PostProcessor
* JDBC PostProcessor
* BSF PostProcessor
* CSS/JQuery Extractor
* BeanShell PostProcessor
* Debug PostProcessor

## Execution Order of Test Elements

Following is the execution order of the test plan elements −

* Configuration elements
* Pre-Processors
* Timers
* Sampler
* Post-Processors (unless SampleResult is null)
* Assertions (unless SampleResult is null)
* Listeners (unless SampleResult is null)

# Database Test Plan

we will see how to create a simple test plan to test the database server. For our test purpose we use the MYSQL database server. You can use any other database for testing. For installation and table creation in MYSQL please refer [MYSQL Tutorial](http://www.tutorialspoint.com/mysql/index.htm).

Once MYSQL is installed, follow the steps below to setup the database −

* Create a database with name "tutorial".
* Create a table *tutorials\_tbl*.
* Insert records into *tutorials\_tbl* as shown below −

mysql> use TUTORIALS;

Database changed

mysql> INSERT INTO tutorials\_tbl

->(tutorial\_title, tutorial\_author, submission\_date)

->VALUES

->("Learn PHP", "John Poul", NOW());

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO tutorials\_tbl

->(tutorial\_title, tutorial\_author, submission\_date)

->VALUES

->("Learn MySQL", "Abdul S", NOW());

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO tutorials\_tbl

->(tutorial\_title, tutorial\_author, submission\_date)

->VALUES

->("JAVA Tutorial", "Sanjay", '2007-05-06');

Query OK, 1 row affected (0.01 sec)

mysql>

* Copy the appropriate JDBC driver to **/home/manisha/apache-jmeter-2.9/lib**.

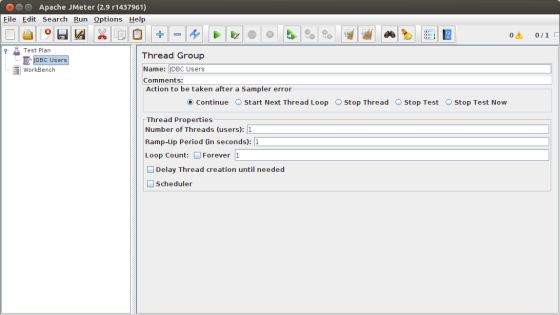
## Create JMeter Test Plan

Let us start the JMeter from **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**.

### Add Users

To create a Thread group,

* Right-click on Test Plan.
* Select Add > Threads (Users) > Thread Group.
* Thus, thread group gets added under the Test Plan node.
* Rename this Thread Group as *JDBC Users*.



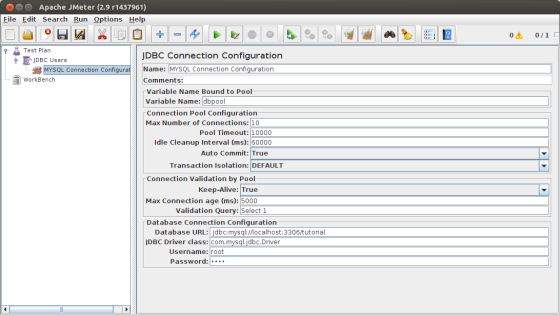
We will not change the default properties of the Thread Group.

## Adding JDBC Requests

Now that we defined our users, it is time to define the tasks that they will be performing. In this section, specify the JDBC requests to perform.

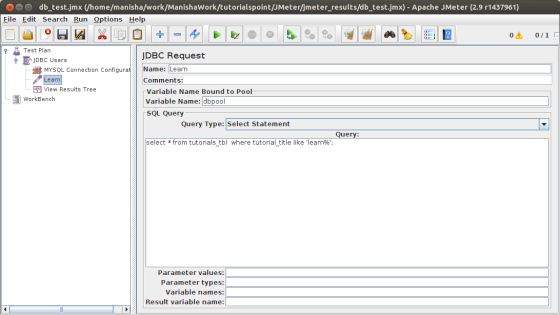
* Right-click on the JDBC Users element.
* Select **Add > Config Element > JDBC Connection Configuration**.
* Set up the following fields (we are using MySQL database called tutorial) −
  + Variable name bound to pool. This needs to identify the configuration uniquely. It is used by the JDBC Sampler to identify the configuration to be used. We have named it as *test*.
  + Database URL − jdbc:mysql://localhost:3306/tutorial.
  + JDBC Driver class: com.mysql.jdbc.Driver.
  + Username: root.
  + Password: password for root.

The other fields on the screen are left as defaults as shown below −



Now add a JDBC Request which refers to the JDBC Configuration pool defined above. Select JDBC Users element.

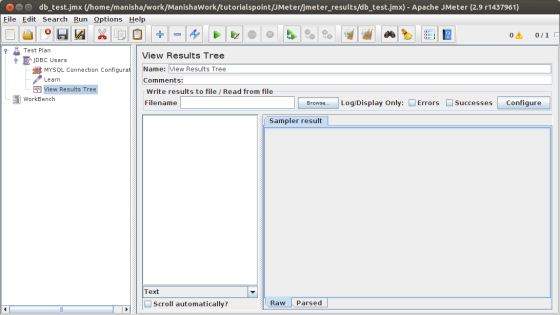
* Click your right mouse button to get the Add menu
* Select **Add > Sampler > JDBC Request.**
* Select this new element to view its Control Panel.
* Edit the properties as shown below −
  + Variable name bound to pool. This needs to uniquely identify the configuration. It is used by the JDBC Sampler to identify the configuration to be used. Named it as *test*.
  + Name − Learn.
  + Enter the Pool Name − test (same as in the configuration element).
  + Query Type − Select statement.
  + Enter the SQL Query String field.



## Create Listener

Now add the Listener element. This element is responsible for storing all of the results of your JDBC requests in a file and presenting a visual model of the data.

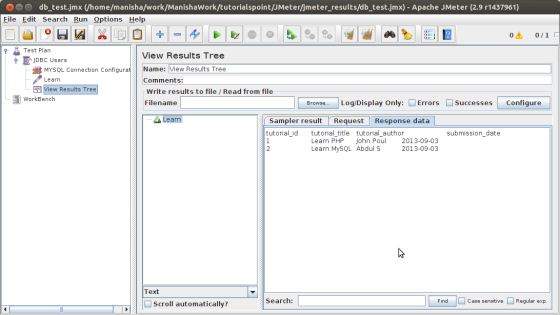
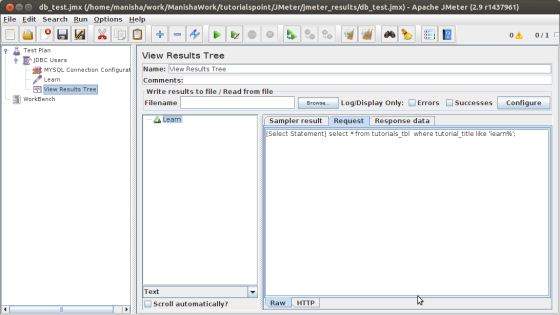
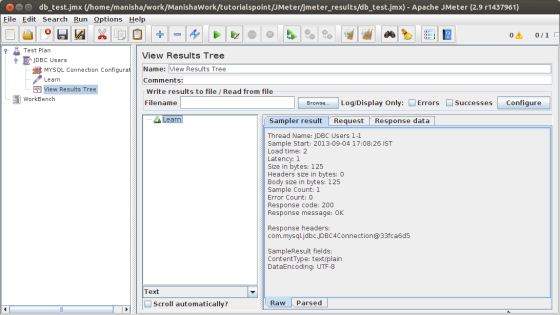
* Select the JDBC Users element
* Add a View Results Tree listener (**Add > Listener > View Results Tree**).



## Save and Execute Test Plan

Now save the above test plan as *db\_test.jmx*. Execute this test plan using **Run > Start** option.

## Verify the Output



In the last image, you can see that two records are selected

How to start Apache JMeter in non-graphical (non-GUI) mode?

If you want to start Apache JMeter in non-GUI mode, use the following command line options:

* -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

### Example

|  |  |
| --- | --- |
| 1 | jmeter -n -t my\_test\_plan.jmx -l log.jtl |

#### Optional

* -j – name of JMeter run log file
* -r – Run the test in the servers specified by the JMeter property “remote\_hosts”
* -R – list of remote servers Run the test in the specified remote servers
* -H – proxy server hostname or ip address
* -P – proxy server port

# How Do I Run JMeter in Non-GUI Mode?

# Do you think the JMeter GUI is a good GUI?

After all, the GUI is always considerate, shows up on time, is willing to carry it's share of the load....

On one hand, the GUI makes the program more intuitive; on the other hand, it consumes more resources.

### 

## Isn’t better to redirect these resources to the generation of useful load tests?

Yes, it is very good idea. Let's review a few typical cases of running JMeter in non-GUI mode.

**Step 1. Prepare the JMeter test**

* Open your \*.jmx file
* Remove all graphic data writers

They will not work in the non-gui mode and there no need to keep them in testing scenario.

**Step 2. Run the JMeter test**

* Go to bin folder of your JMeter installation from command prompt
* Type the following command:

jmeter -n -t D:\TestScripts\script.jmx -l D:\TestScripts\scriptresults.jtl

-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]

-j  [name of JMeter run log file].

Besides these options, JMeter has several other parameters that can be used for running in non-GUI mode.

-R [list of remote servers] Run the test in the specified remote servers

-H [proxy server hostname or ip address]

-P [proxy server port]

These options are used for remote execution of JMeter tests and for using JMeter through a proxy server.

**Step 3. Stop running the test**

When running JMeter in non-GUI mode, there is no Menu, and JMeter does not react to keystrokes such as Control + '.'. So in versions post 2.3.2, JMeter non-GUI mode will listen for commands on a specific port (default 4445, see the JMeter property jmeterengine.nongui.port ). In versions post 2.4, JMeter supports the automatic choice of an alternate port if the default port is being used (i.e. by another JMeter instance). In this case, JMeter will try the next highest port, continuing until it reaches the JMeter property jmeterengine.nongui.maxport, which defaults to 4455. If maxportis less than or equal to port , port scanning will not take place.

Note that JMeter 2.4 and earlier did not set up the listener for non-GUI clients, only non-GUI standalone tests; this has been fixed.

The chosen port is displayed in the console window.

The commands currently supported are:

* Shutdown - graceful shutdown
* StopTestNow - immediate shutdown

These commands can be sent by using the shutdown[.cmd|.sh] or stoptest[.cmd|.sh] script respectively. The scripts are to be found in the JMeter /bin directory. The commands will only be accepted if the script is run from the same host.

Note-before using shutdown.cmd or shutdown.sh CLASSPATH variable should be set to path with ApacheJmeter.jar, because inside these files we have string like

“java -cp %~dp0ApacheJMeter.jar org.apache.jmeter.util.ShutdownClient StopTestNow %\*”. It means that JVM calls class file from ApacheJMeter.jar.

# Five Ways To Launch a JMeter Test without Using the JMeter GUI

## 1. USE THE COMMAND LINE

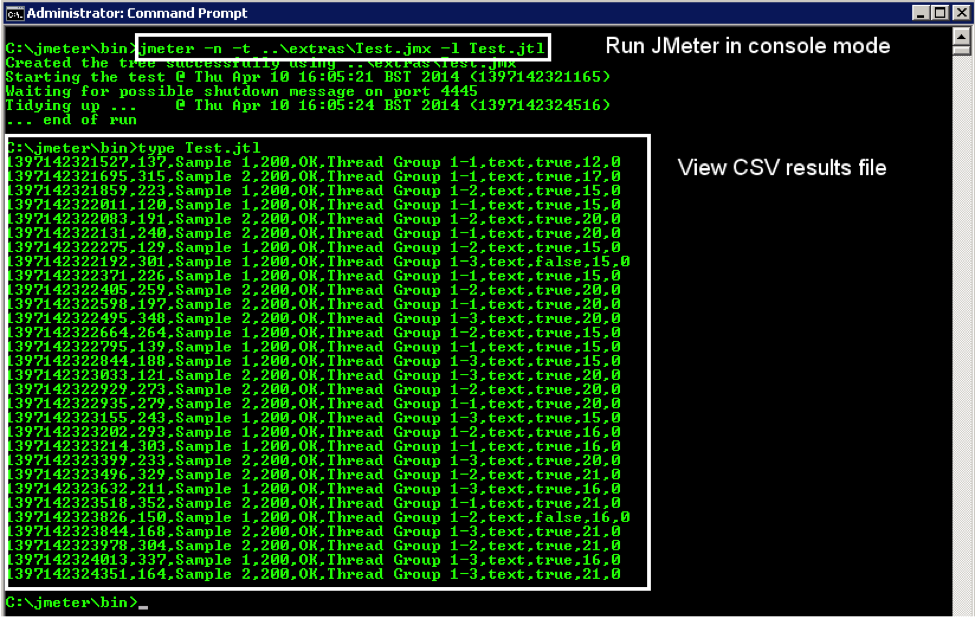
You can execute JMeter test from the command line. It is as simple as

**jmeter -n -t your\_script.jmx**

where

* -n - tells JMeter to run in non-GUI mode
* -t - specifies the path to source .jmx script to run

We often use these minimal options in combination with -l switch, which tells JMeter where to store test results. If a results file already exists, it will be appended. After the test execution you can open the resulting CSV file with any Listener, Excel, or any other analytics software.

Increase image

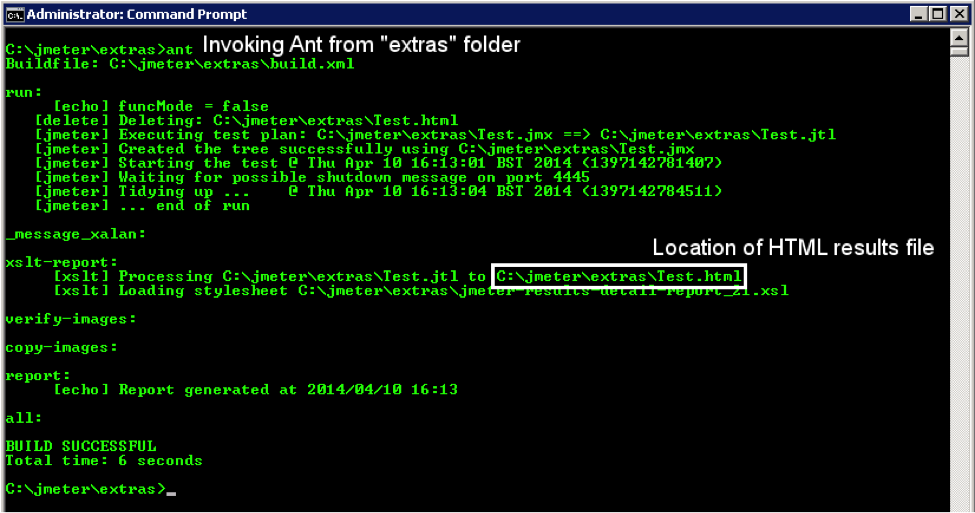
If you want more details, see [“How do I run JMeter in non-GUI mode?”](https://blazemeter.com/blog/dear-abby-blazemeter-how-do-i-run-jmeter-non-gui-mode?utm_source=BM&utm_medium=BM_blog&utm_campaign=5_Ways_to_Launch_blog) or use the -h parameter to quickly view all possible command line options.

## 

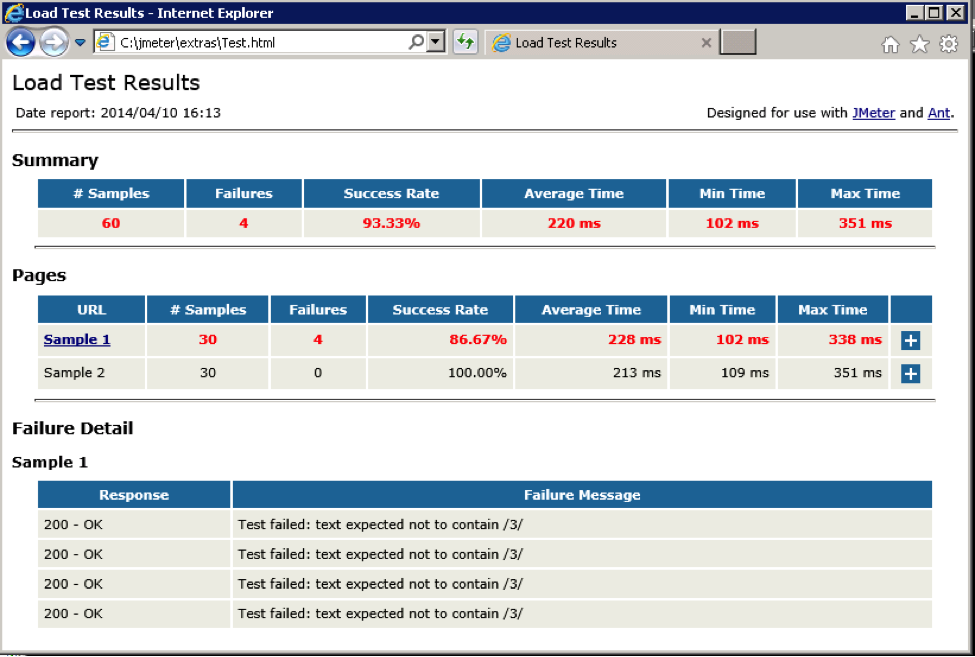
## 2. USE APACHE ANT

You can integrate JMeter with [Apache Ant](http://ant.apache.org/) using the [JMeter Ant Task](http://www.programmerplanet.org/projects/jmeter-ant-task/" \t "_blank). Once you have properly formed an Ant build XML configuration file, you can initiate the JMeter script using Ant. (To see a sample JMeter test plan, look in the **/extras** folder of your JMeter installation. It’s called **Test.jmx**. The example **build.xml** Ant configuration file is in the same location.)

Assuming Apache Ant is properly [installed](http://ant.apache.org/manual/install.html), go to the **/extras** folder and invoke an Ant command there. Ant looks for a **build.xml** file under the current directory and processes it.

Increase image

As shown above, Ant picks up the **Test.jmx** file, executes it, and generates an easily-readable HTML report.

Increase image

## 3. USE APACHE MAVEN

You can integrate JMeter with [Apache Maven](http://maven.apache.org/) using the [JMeter Maven Plugin](http://wiki.apache.org/jmeter/JMeterMavenPlugin" \t "_blank). Maven uses a **pom.xml** file where POM stands for “Project Object Model” as a fundamental unit of work to be done. To enable JMeter integration, you must add a **<build>** event to **pom.xml**. The Maven project structure should look like the following:

JMeter can be integrated with [Apache Maven](http://maven.apache.org/) via the [JMeter Maven Plugin](http://wiki.apache.org/jmeter/JMeterMavenPlugin" \t "_blank). Maven uses a pom.xml file where POM stands for “Project Object Model” as a fundamental unit of work to be done. To enable JMeter integration you need to add a <build> event to pom.xml file. The Maven project structure should look as follows:

Folder structure:

* **root folder**
  + **src**
    - **test**
      * **jmeter**
        + **Test.jmx**
  + **pom.xml**

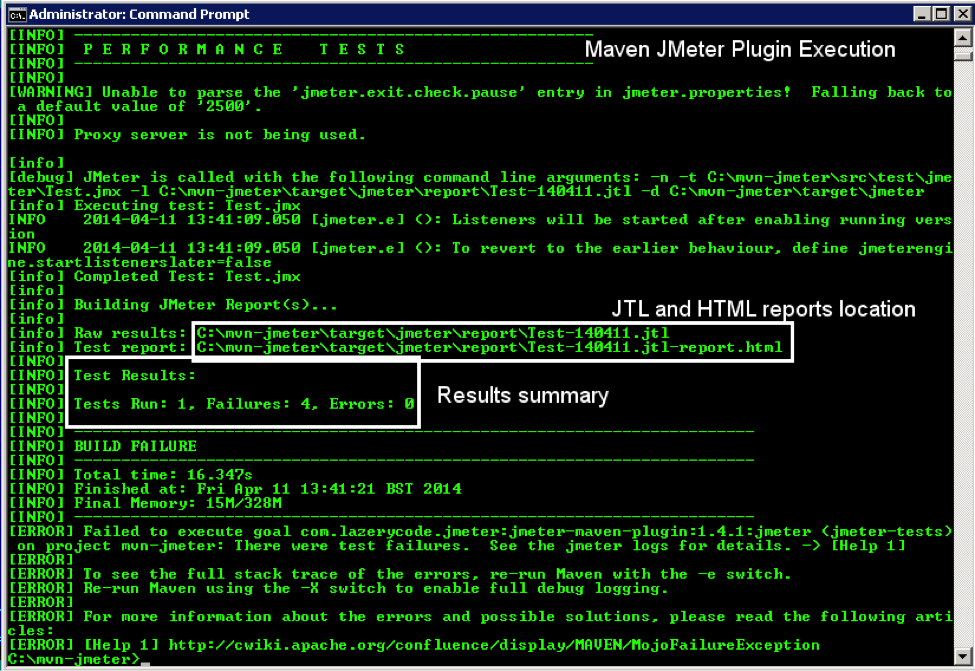
Where:

* **Test.jmx** – can be copied???? from jmeter/extras
* **pom.xml** - see the minimum listing below?

**<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">  
    <modelVersion>4.0.0</modelVersion>  
    <groupId>com.blazemeter</groupId>  
    <artifactId>mvn-jmeter</artifactId>  
    <packaging>jar</packaging>  
    <version>1.0-SNAPSHOT</version>  
    <name>maven-jmeter-demo</name>  
    <url>http://maven.apache.org</url>  
    <build>  
        <plugins>  
            <plugin>  
                <groupId>com.lazerycode.jmeter</groupId>  
                <artifactId>jmeter-maven-plugin</artifactId>  
                <version>1.4.1</version>  
                <executions>  
                    <execution>  
                        <id>jmeter-tests</id>  
                        <phase>verify</phase>  
                        <goals>  
                            <goal>jmeter</goal>  
                        </goals>  
                    </execution>  
                </executions>  
            </plugin>  
        </plugins>  
    </build>  
</project>**

To kick off the test, type **mvn verify** or **mvn install** in the root folder. You should see some Maven output in the command prompt:

Increase image

For basic and advanced configuration options, consult the Maven JMeter Plugin [Project Wiki](http://github.com/Ronnie76er/jmeter-maven-plugin/wiki).

The HTML report that’s created is identical to the one generated by Apache Ant.

## 

## 4. RUN A JMETER TEST THROUGH A PROGRAM (FROM JAVA CODE)

Another option is to run a JMeter script from Java code. To execute an existing JMeter test from Java code or to create one through programming, basic Java knowledge is a must, and the following are mandatory:

1. Have JMeter installed somewhere
2. Have the required JMeter jars from **/lib** and especially **/lib/ext** folders of your JMeter installation in your project or module class path.

### 

### 4.2 Running an existing JMeter Test from Java code

The main class, which is the “heart” of JMeter is [StandardJMeterEngine](http://jmeter.apache.org/api/org/apache/jmeter/engine/StandardJMeterEngine.html" \t "_blank). If you must execute a JMeter test from Java code, this is the best option. Alternatively, you can extend this class or implement your own version of the [JMeterEngine](http://jmeter.apache.org/api/org/apache/jmeter/engine/JMeterEngine.html" \t "_blank) interface.

The absolute minimal code to read the existing **.jmx** file (again we’re referring to **Test.jmx** from the **/extras**folder of the JMeter standard installation) and execute it will be as follows:

**package com.blazemeter.demo;**

**import org.apache.jmeter.engine.StandardJMeterEngine;  
import org.apache.jmeter.save.SaveService;  
import org.apache.jmeter.util.JMeterUtils;  
import org.apache.jorphan.collections.HashTree;  
import java.io.FileInputStream;  
  
public class JMeterFromExistingJMX {  
  
    public static void main(String[] argv) throws Exception {  
        // JMeter Engine  
        StandardJMeterEngine jmeter = new StandardJMeterEngine();**

**// Initialize Properties, logging, locale, etc.**  
**JMeterUtils.loadJMeterProperties("/path/to/your/jmeter/bin/jmeter.properties");  
        JMeterUtils.setJMeterHome("/path/to/your/jmeter");  
        JMeterUtils.initLogging();// you can comment this line out to see extra log messages of i.e. DEBUG level  
        JMeterUtils.initLocale();  
  
        // Initialize JMeter SaveService  
        SaveService.loadProperties();  
  
        // Load existing .jmx Test Plan  
        FileInputStream in = new FileInputStream("/path/to/your/jmeter/extras/Test.jmx");  
        HashTree testPlanTree = SaveService.loadTree(in);  
        in.close();  
  
        // Run JMeter Test  
        jmeter.configure(testPlanTree);  
        jmeter.run();  
    }  
}**

Easy enough, isn’t it?

### 

### 4.3 Creating a New JMeter Test Purely in Java

Here's how to build a JMeter Test Plan from scratch using Java code only. The key classes to look into are:

1. [StandardJMeterEngine](http://jmeter.apache.org/api/org/apache/jmeter/engine/StandardJMeterEngine.html) - The main class that which configures the Test Plan and executes it.
2. [HashTree](https://jmeter.apache.org/api/org/apache/jorphan/collections/HashTree.html) - A special collection that holds Test Plan elements.
3. A minimum of JMeter Controllers necessary to run the test:

○      [TestPlan](http://jmeter.apache.org/api/org/apache/jmeter/testelement/TestPlan.html" \t "_blank) - The root container for all below plus the place where all test properties can be specified

○      [ThreadGroup](https://jmeter.apache.org/api/org/apache/jmeter/threads/ThreadGroup.html" \t "_blank) - A pool of users to execute the test. A test must have at least one Thread Group with at least one thread and one loop.

○      [LoopController](https://jmeter.apache.org/api/org/apache/jmeter/control/LoopController.html" \t "_blank) - Since you must have at least one loop, it’s essential to have a Loop Controller instance set as a main Sampler controller for a Thread Group.

○      A [Sampler](https://jmeter.apache.org/api/org/apache/jmeter/samplers/Sampler.html) to do the actual work.

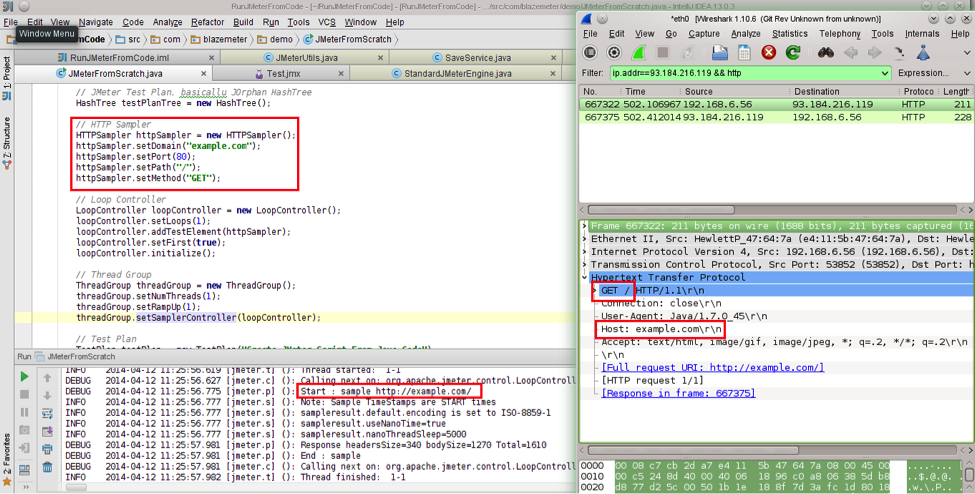
This demo uses all of the above to send one GET request with one user in one loop using the [HTTPSampler](http://jmeter.apache.org/api/org/apache/jmeter/protocol/http/sampler/HTTPSampler.html" \t "_blank) to the [example.com](http://tools.ietf.org/html/rfc2606#section-3) domain.

This example is minimalistic and doesn’t have any extras. This simply gives you an idea of how to start. The code should be self-explanatory.

**package com.blazemeter.demo;**

**import org.apache.jmeter.control.LoopController;  
import org.apache.jmeter.engine.StandardJMeterEngine;  
import org.apache.jmeter.protocol.http.sampler.HTTPSampler;  
import org.apache.jmeter.testelement.TestPlan;  
import org.apache.jmeter.threads.ThreadGroup;  
import org.apache.jmeter.util.JMeterUtils;  
import org.apache.jorphan.collections.HashTree;**  
  
public class JMeterFromScratch {  
  
**public static void main(String[] argv) throws Exception {  
  
        //JMeter Engine  
        StandardJMeterEngine jmeter = new StandardJMeterEngine();  
  
        //JMeter initialization (properties, log levels, locale, etc)  
        JMeterUtils.loadJMeterProperties("/path/to/your/jmeter/bin/jmeter.properties");  
        JMeterUtils.initLogging();// you can comment this line out to see extra log messages of i.e. DEBUG level  
        JMeterUtils.initLocale();  
  
        // JMeter Test Plan, basic all u JOrphan HashTree  
        HashTree testPlanTree = new HashTree();  
  
        // HTTP Sampler  
        HTTPSampler httpSampler = new HTTPSampler();  
        httpSampler.setDomain("example.com");  
        httpSampler.setPort(80);  
        httpSampler.setPath("/");  
        httpSampler.setMethod("GET");  
  
        // Loop Controller  
        LoopController loopController = new LoopController();  
        loopController.setLoops(1);  
        loopController.addTestElement(httpSampler);  
        loopController.setFirst(true);  
        loopController.initialize();**  
  
        **// Thread Group  
        ThreadGroup threadGroup = new ThreadGroup();  
        threadGroup.setNumThreads(1);  
        threadGroup.setRampUp(1);  
        threadGroup.setSamplerController(loopController);  
  
        // Test Plan  
        TestPlan testPlan = new TestPlan("Create JMeter Script From Java Code");  
  
        // Construct Test Plan from previously initialized elements  
        testPlanTree.add("testPlan", testPlan);  
        testPlanTree.add("loopController", loopController);  
        testPlanTree.add("threadGroup", threadGroup);  
        testPlanTree.add("httpSampler", httpSampler);  
  
        // Run Test Plan  
        jmeter.configure(testPlanTree);  
        jmeter.run();  
  
    }  
}**

To verify the code actually sends GET request to example.com, a sniffer tool like [Wireshark](http://www.wireshark.org/) can be used.

Increase image

Similarly, you can add any listener to your test to verify the output with JMeter GUI Listeners or third-party tools.

## 5. RUN A BLAZEMETER TEST USING A PROGRAM

[BlazeMeter provides a REST API](http://community.blazemeter.com/knowledgebase/articles/69998-api-getting-started?utm_source=BM&utm_medium=BM_blog&utm_campaign=5_Ways_to_Launch_blog) that enables you to perform various tasks in an unattended manner using any tool, library, or software capable of sending HTTP requests. That means you can do the following actions by sending a single request to one a BlazeMeter API endpoint:

* Create a test
* Upload new or update existing script file
* Start the test
* Get test status
* Stop the test

All these tasks assume a user\_key mandatory parameter. The user API key is available on the Profile Page. To reach the Profile Page, click your user name in the top right corner of the screen, to the left of the Logout link, and expand the User Key section at the Profile page.

This demo uses the [curl](http://curl.haxx.se/download.html) command-line utility (a free, open-source, cross-platform and multi-protocol utility for transferring data to and from endpoints with a URL syntax).

### 

### Create the Test

Creating a new BlazeMeter test is as easy as sending a GET request to a REST URL. See details below:

Parameters:

|  |  |
| --- | --- |
| **URL** | **https://a.blazemeter.com/api/rest/blazemeter/testCreate** |
| **Method** | **GET** |
| **user\_key** | **API user key available at the Profile page** |
| **test\_name** | **any meaningful string** |

Example:

**:~>curl "https://a.blazemeter.com/api/rest/blazemeter/testCreate?user\_key=blazemeter\_demo\_key&test\_name=MyFirstTest"**

**---**

**response\_code: 200**

**error: null**

**test\_id: 525901**

**test\_name: null**

The request goes before the three dashes (“---”). The response comes  after the three dashes. As you see, the response code is HTTP 200, which is OK. The ID of the test is returned. Mentioning the ID is a must to start or stop the test and to get its status and report link.

### 

### Upload a JMeter .jmx Test Script

Adding a new .jmx file to an existing test or replacing an old one with a newer version can be done as follows:

|  |  |
| --- | --- |
| **URL** | **https://a.blazemeter.com/api/rest/blazemeter/testScriptUpload** |
| **Method** | **POST** |
| **user\_key** | **API user key available at Profile page** |
| **test\_id** | **ID of test available as createTest response or at Test page after Test Name** |
|  | **Path to JMeter .jmx file to be uploaded** |

**:~>curl -X POST "https://a.blazemeter.com/api/rest/blazemeter/testScriptUpload?user\_key=blazemeter\_demo\_key&test\_id=525901" -H "Content-Type: application/json" --data-binary "/path/to/your/jmeter/script/Test.jmx"**

**---**

**response\_code: 200**

**error: null**

### 

### Starting a Test in BlazeMeter

You can initiate a Blazemeter test by issuing a request in the following structure:

|  |  |
| --- | --- |
| **URL** | **http://blazemeter.com/api/rest/blazemeter/testStart/** |
| **user\_key** | **API user key available at Profile page** |
| **test\_id** | **ID of test available as createTest response or at Test page after Test Name** |

**:~>curl "https://a.blazemeter.com/api/rest/blazemeter/testStart?user\_key=blazemeter\_demo\_key&test\_id=525901"**

**---**

**response\_code: 200**

**error: null**

**test\_id: 525901**

**type: null**

**test\_name: MyFirstTest**

**session\_id: r-ec53490fab83a51**

**:~>**

### 

### Getting the Test Status in BlazeMeter

You can obtain the test status by sending a proper GET request to the getTestStatus endpoint.

|  |  |
| --- | --- |
| **URL** | **https://a.blazemeter.com/api/rest/blazemeter/testGetStatus** |
| **user\_key** | **API user key available at Profile page** |
| **test\_id** | **ID of test available as createTest response or at Test page after Test Name** |

**:~>curl "https://a.blazemeter.com/api/rest/blazemeter/testGetStatus?user\_key=blazemeter\_demo\_key&test\_id=525901"**

**---**

**response\_code: 200**

**error: null**

**test\_id: 525901**

**session\_id: r-ec534910ae2407d**

**status: Running**

**test\_name: MyFirstTest**

**:~>**

As we’ve just kicked off the test it’s currently in “Running” state.

### 

### Stopping a Test in BlazeMeter

Just like starting a test, you can also issue a command to stop test execution.

|  |  |
| --- | --- |
| **URL** | **https://a.blazemeter.com/api/rest/blazemeter/testStop** |
| **user\_key** | **API user key available at Profile page** |
| **test\_id** | **ID of test available as createTest response or at Test page after Test Name** |

**:~>curl "https://a.blazemeter.com/api/rest/blazemeter/testStop?user\_key=blazemeter\_demo\_key&test\_id=525901"**

**---**

**response\_code: 200**

**error: null**

**session\_id: r-ec53490fab83a51**

**:~>**

You can verify that the test has been actually stopped by querying the test status one more time:

**:~>curl "https://a.blazemeter.com/api/rest/blazemeter/testGetStatus?user\_key=blazemeter\_demo\_key&test\_id=525901"**

**---**

**response\_code: 200**

**error: null**

**test\_id: 525901**

**session\_id:**

**status: Not Running**

**test\_name: MyFirstTest**

**:~>**

As this confirms, the test is in a “Not Running” state.

## 

## SUMMARY / RESOURCES

In this article, we’ve highlighted the most common options executing a JMeter test. For additional information and other techniques, consider such as:

* The [Jenkins Performance Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Performance+Plugin) which can trigger JMeter test execution as a goal in the Jenkins Continuous Integration system.
* A [JMeter plugin for IntelliJ Idea](http://plugins.jetbrains.com/plugin/7013?pr=" \t "_blank) which integrates JMeter with the Idea Java IDE.

Regardless of which method you chose to match your environment and infrastructure, be sure to never use JMeter GUI for anything but test development and/or [debugging](http://blazemeter.com/blog/how-debug-your-apache-jmeter-script).

## Want to Learn More About JMeter & Load Testing?

If you are new to JMeter, and you’d like to learn more, please sign up for our [free online JMeter training course](http://info.blazemeter.com/jmeter-training-course?utm_source=BM&utm_medium=BM_blog&utm_campaign=5_Ways_to_Launch_blog).

For more experienced JMeter users, you'll want to view the on-demand webcast, [How to Create Advanced Load Testing Scenarios with JMeter](http://info.blazemeter.com/loadtesting_advancedjmeter?utm_source=BM&utm_medium=BM_blog&utm_campaign=5_Ways_to_Launch_blog).

# How the BlazeMeter Load Testing Cloud Complements and Strengthens JMeter

While JMeter represents a strong and compelling way to perform load testing, of course, we recommend supplementing that tool with [BlazeMeter](http://info.blazemeter.com/testing-landing-page2?utm_source=BM&utm_medium=BM_blog&utm_campaign=5_Ways_to_Launch_blog), which lets you simulate up to millions of users in a single developer-friendly, self-service platform.  With BlazeMeter, you can test the performance of any mobile app, website, or API in under 10 minutes.  Here’s why we think the BlazeMeter/JMeter combination is attractive to developers:

* Simple Scalability – It’s easy to create large-scale JMeter tests. You can run far larger loads far more easily with BlazeMeter than you could with an in-house lab.
* Rapid-Start Deployment – BlazeMeter’s recorder helps you get started with JMeter right away, and BlazeMeter also provides complete tutorials and tips.
* Web-Based Interactive Reports – You can easily share results across distributed teams and overcome the limitations of JMeter’s standalone UI.
* Built-In Intelligence – BlazeMeter provides on-demand geographic distribution of load generation, including built-in CDN-aware testing.

# 3 Easy Ways to Monitor JMeter Non-GUI Test Results

The [Apache JMeter™](http://jmeter.apache.org/) GUI enables you to easily configure your performance scripts, add new test elements, and monitor test execution by using a variety of different listeners. But this “easy-to-use” solution has its drawbacks, especially when you need to use it for more complex cases.

For example, you might feel the disadvantages of the GUI mode when you start to perform load tests with thousands of users since running JMeter in GUI mode takes a lot of your machine resources. You might also be confused about how to use JMeter GUI in your continuous integration server (CI). The answer is that there's no way.

Fortunately, JMeter allows you to save performance tests as a script and run them in the non-GUI mode via the command line. This enables running very large tests, as well as adding the same command into continuous integration and automating the process.

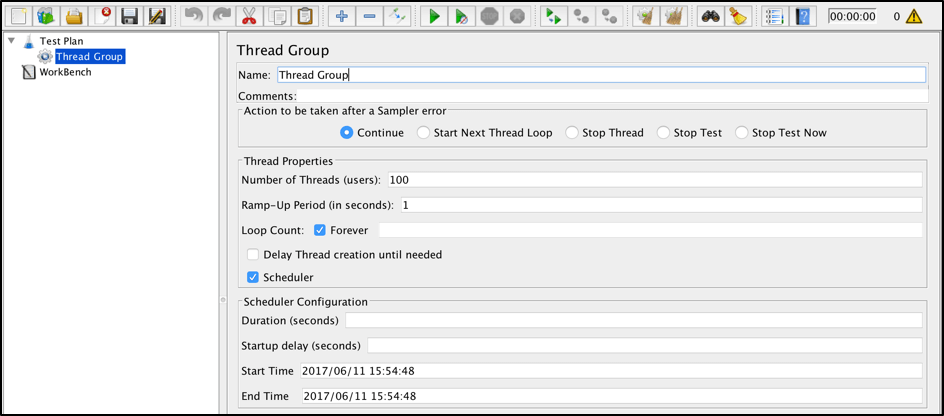
But at this point, you might notice that you are not able to monitor your scripts with GUI JMeter listeners like you were before. In this article, we are going to cover that gap and help you start monitoring your non-GUI tests in a few steps: through out-of-the-box logs, via Taurus, and through the BlazeMeter cloud.

First of all, let’s create a performance script that we can use for the test. We will run a load test for the <http://blazedemo.com/> web app, which is available for everyone who wants to play around with some performance scripts.

We are not going to cover GUI performance script creation from scratch, assuming that you have some hands-on experience on that. Briefly, this is the performance script configuration:

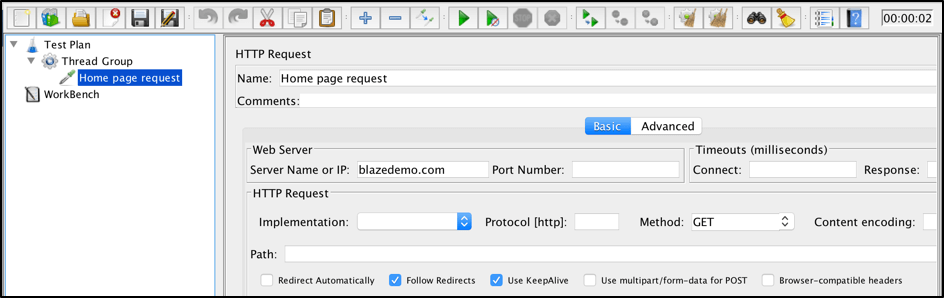
Thread group:

* **Number of threads**: 100
* **Ramp-up period**(seconds): 120
* **Loop count**: Forever
* **Scheduler checked**
* **Duration**(seconds): 300



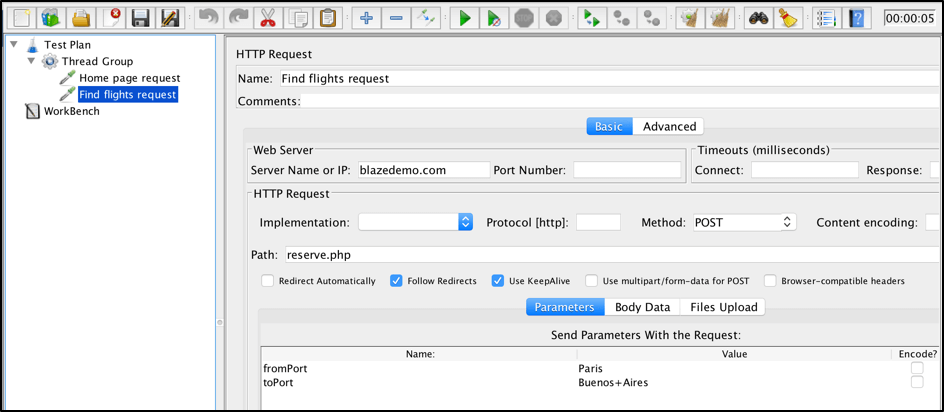
HTTP Sampler (1/2):

* **Name**: Homepage request
* **Server name**: blazedemo.com
* **Method**: GET



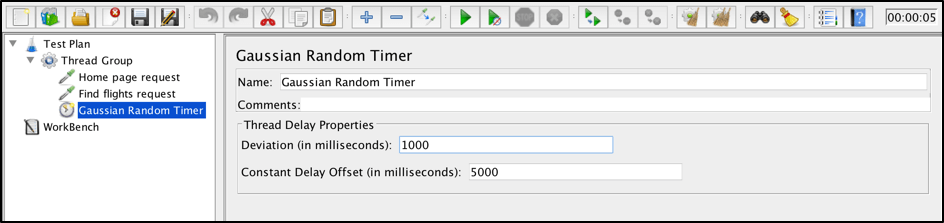
HTTP Sampler (2/2):

* **Name**: Find flights request
* **Server name**: blazedemo.com
* **Method**: POST
* **Path**: reserve.php
* **Parameters**:
  + **fromPort**: Paris
  + **toPort**: Buenos+Aires



Add a random timeout to simulate user behavior:

* **Gaussian Random Timer**:
  + **Deviation** (in milliseconds): 1000
  + **Constant Delay Offset** (in milliseconds): 5000



Save the JMX file to be used to run the performance test via the command line. You can do it this way:

**File** > **Save Test Plan As** > **JMX file**. I named mine NonGuiTestMonitoring.jmx.

At this point, you can forget about the JMeter GUI application and close it until you want to change something in the script.

## 1. Monitoring JMeter Results From Out-of-the-Box Logs

To run the JMeter JMX file, you can use this command line template:

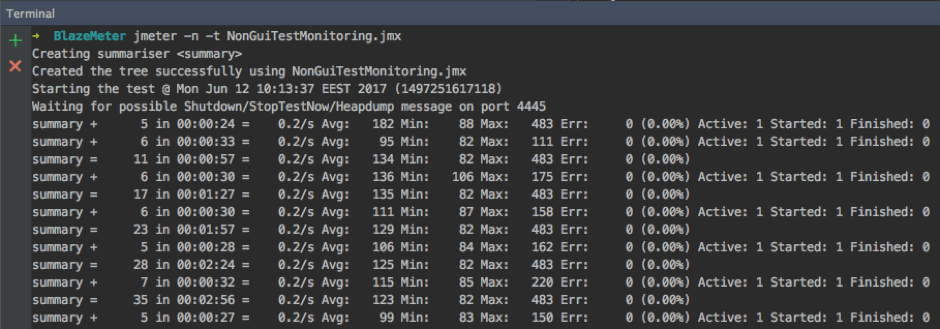
> jmeter -n -t [jmx\_path]

* -n means “run JMeter in Non-GUI mode.”
* -t is used to specify the path to the JMX file.

Assuming that our current location is the folder with the created script, you run it with this command:

> jmeter -n -t NonGuiTestMonitoring.jmx

Let’s run and see which details we can get from JMeter Non-GUI mode.



As you can see, even out of the box JMeter, provides some useful stats that can be used for basic real-time monitoring of your script execution. Let’s go over the stats and verify what we have:

* **5 in 00:00:24 = 0.2/s** means that in 24 seconds we have sent 5 requests to the server with an average throughput of 0.2 requests per second.
* **Avg: 182** means that the average response time at that moment is 182 milliseconds.
* **Min: 88** means that the minimum response time from send requests for that period was 88 milliseconds.
* **Max: 483** means that the maximum response time from send requests for that period was 483 milliseconds.
* **Err: 0 (0.00%)** means that we didn’t have any errors in requests for that period and the percentage of errors from total requests is 0 accordingly.
* **Active:** shows the number of active users who were performing requests for this period.
* **Started:** shows the total number of started threads since the beginning of the tests.
* **Finished:** shows the total number of threads that already finished execution since the beginning of tests.

These metrics show us some basic stats about our performance tests execution. However, they do not contain any specific details and reflect only the general server state. Let’s see how we can improve that.

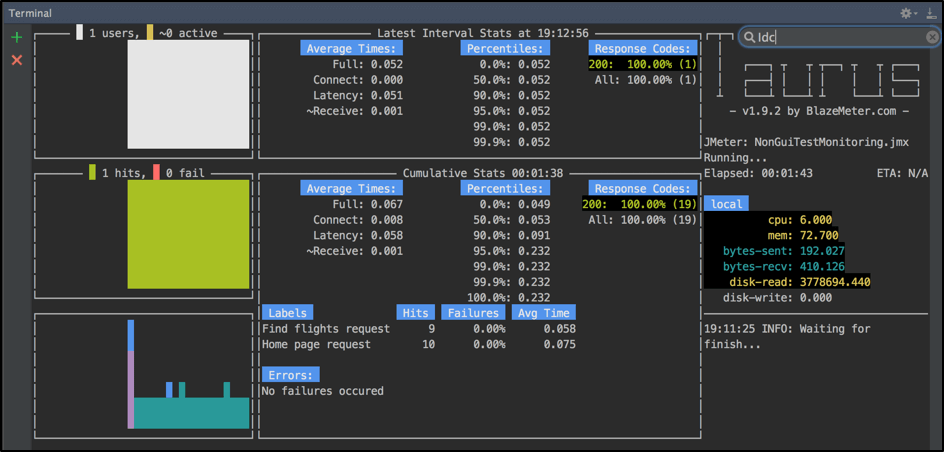
## 2. Monitoring JMeter Results via Taurus

You also can go with another way. By using the Taurus framework, you can get real-time monitors right inside your console. Taurus can be used as a JMX script runner and provides many additional features, including real-time monitoring.

After you finish the Taurus installation with the few steps mentioned on this page, you can go ahead and run the same JMX file from before.

Just run this command:

> bzt NonGuiTestMonitoring.jmx



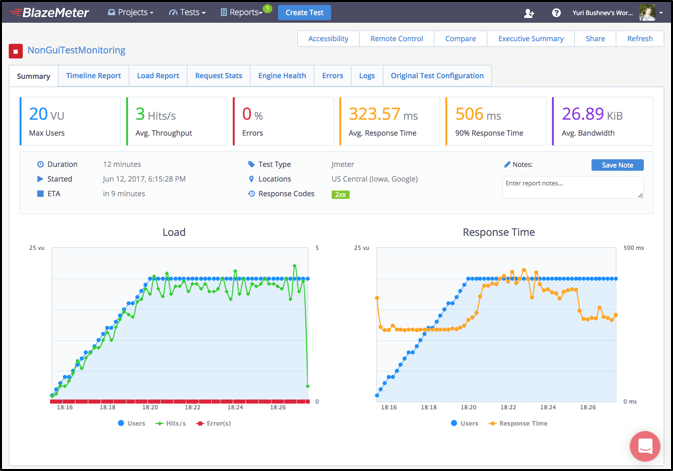
Even without any additional configurations, Taurus allows you to run your performance script with detailed real-time monitoring in the command line. On the Console dashboard, you can find almost all the metrics you might be interested in.

You can track active and all users, different response metrics for each sampler separately, errors, and execution machine metrics. This solution is perfect if you want just to monitor your tests in real time and don’t want to waste time on additional configurations. It’s also helpful that you don’t need to use anything except the console and you can even run these monitors on a remote server that doesn’t have a desktop (via SSH) and still see execution monitors.

## 3. Monitoring JMeter Results via the BlazeMeter Cloud

You can also easily run your JMX script in the cloud by using [*https://a.blazemeter.com*](https://a.blazemeter.com/). Use the free out-of-the-box version to upload your JMX script and run a load of up to 50 users. To run the test, you can either upload your JMX file in the box below or at the top of the page, or open the web application and then choose **JMeter Test** and configure your test.

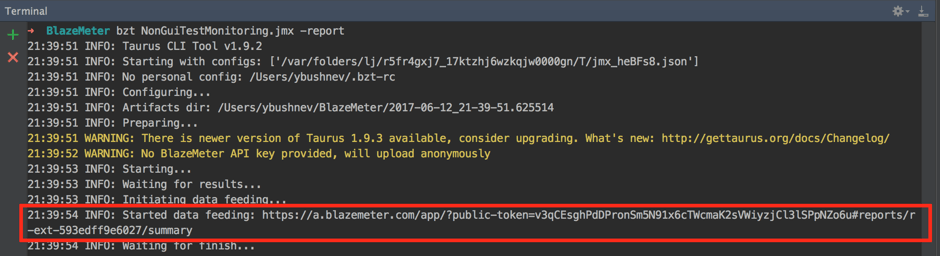
After that, you might need to wait for two to four minutes until cloud server is up and running. As soon as the script is running, you will be redirected to the page of real-time monitoring where you can see all KPIs and analytics, like response time, active users, the percentage of errors and bandwidth, as well as multiple reports.



You can combine Taurus with BlazeMeter. By using just one command line parameter, you can still monitor results through your console, with the BlazeMeter results analysis. This will allow you to save all your test reports and perform additional analyses and metrics’ comparison afterward. All you need to do is to add one argument to the existing command:

> bzt NonGuiTestMonitoring.jmx -report

Then, in the logs execution, you will find the link to real-time monitoring. The new browser tab will also be automatically opened during test execution.



In addition to all these methods, there is one more way to establish detailed monitoring: by integrating JMeter, InfluxDB, and Grafana. This method might suit you if you are able to spend more time on the configuration and maintenance of your solution. We will cover this solution in the next blog post, so stay tuned!