JMETER RESULT ANALYSIS: THE ULTIMATE GUIDE

Pre-requisites

Installation

This tutorial assumes you already have the following software installed:

* [**Java 8**](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html),
* [**JMeter 3.3**](http://jmeter.apache.org/download_jmeter.cgi) or above.

During the whole guide, the following [**Sample JMX**](https://octoperf.com/img/blog/how-to-analyze-jmeter-test-results/jpetstore.jmx) is used. This JMX tests our [**demo application**](https://petstore.octoperf.com/) based on a Java JPetstore bundled in a [**Docker Image**](https://hub.docker.com/r/jloisel/jpetstore6/).

Understanding JMeter Metrics

[**JMeter Metrics**](http://jmeter.apache.org/usermanual/glossary.html) are widely used in the following section, thus it’s better if you’re comfortable with their definition:

* **Elapsed time**: Measures the elapsed time from just before sending the request to just after the last chunk of the response has been received,
* **Latency**: Measures the latency from just before sending the request to just after the first chunk of the response has been received,
* **Connect Time**: Measures the time it took to establish the connection, including SSL handshake,
* **Median**: Number which divides the samples into two equal halves,
* **90% Line (90th Percentile)**: The elapsed time below which 90% of the samples fall,
* **Standard Deviation**: Measure of the variability of a data set. This is a standard statistical measure,
* **Thread Name**: Derived from the Thread Group name and the thread within the group. The name has the format groupName + “ ” + groupIndex + “-” + threadIndex where:
  + **groupName**: name of the Thread Group element,
  + **groupIndex**: number of the Thread Group in the Test Plan, starting from 1,
  + **threadIndex**: number of the thread within the Thread Group, starting from 1.
* **Throughput**: Calculated as requests/unit of time. The time is calculated from the start of the first sample to the end of the last sample. The formula is: Throughput = (number of requests) / (total time).

Interpreting JMeter Metrics

How do you know if a metric is satisfying or awful? Here are some explanations:

* **Elapsed Time / Connect Time / Latency**: should be as low as possible, ideally less than 1 second. Amazon found [**every 100ms costs them 1% in sales**](https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/), which translates to several millions of dollars lost,
* **Median**: should be close to average elapsed response time,
* **XX% line**: should be as low as possible too. When it’s way lower than average elapsed time, it indicates that the last XX% requests have dramatically higher response times than lower ones,
* **Standard Deviation**: should be low. A high deviation indicates discrepancies in responses times, which translates to response time spikes.

See, it’s pretty easy! Most of the figures should be as low as possible. However, depending on the context, your boss may provide you with expected responses times under a given load. Use them to compute the **[Apdex](https://en.wikipedia.org/wiki/Apdex)** of each request:

Apdex (Application Performance Index) is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing.

JMeter HeadLess Tests

To run JMeter in headless (non-GUI) mode, which means without any UI, to run load tests use the following command:

jmeter -n -t scenario.jmx -l jmeter.jtl

The command line has the following parameters:

* **-n**: run in non-GUI mode,
* **-t**: specifies the path to source .jmx script to run,
* **-l**: specifies the path to the JTL file which will contain the raw results.

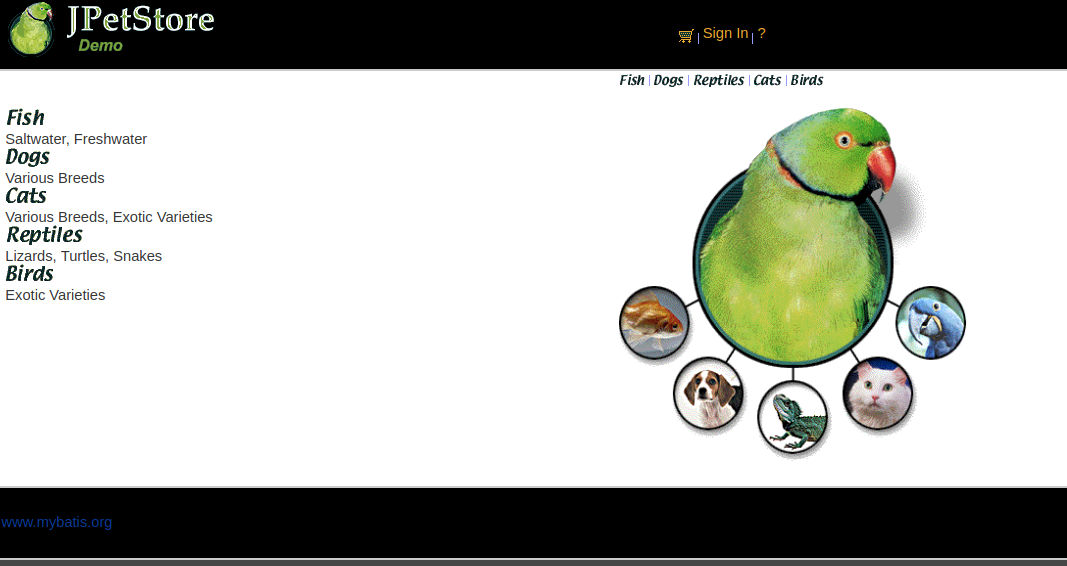
See our blog post [**How To Optimize JMeter for Large Scale Tests**](https://octoperf.com/blog/2017/10/12/optimize-jmeter-for-large-scale-tests) to understand why **running in non-GUI mode is vital**.

Running the Demo App

To run the demo application on your own computer, you will need:

* An operating system compatible with Docker, See [**Docker Installation**](https://docs.docker.com/engine/installation/) for more information,
* [**Docker**](https://docker.com/).

To run the JPetstore demo application, simply execute the command-line docker run -d -p 8080:8080 jloisel/jpetstore6.



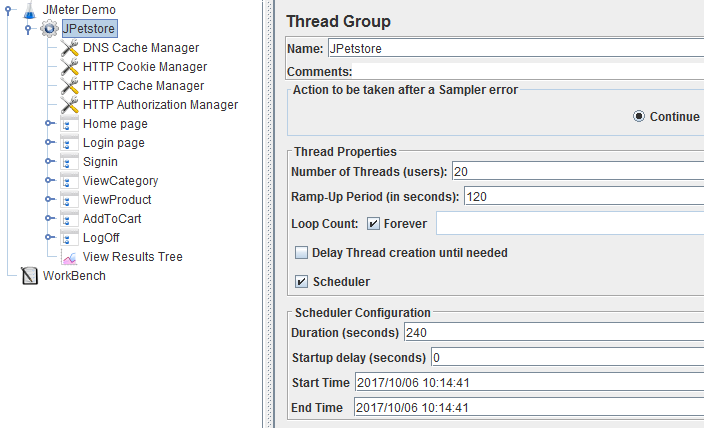
Open your browser, and navigate to http://localhost:8080/actions/Catalog.action. It should show the JPetstore front page.

Thread Group Configuration

The following test will be run:

* 20 concurrent thread groups,
* 120 seconds rampup duration,
* 120 seconds peak test duration.

The test will run for a total of **4 minutes** with **20 concurrent users** peak load.



UI Listeners

JMeter has a number of UI Listeners which can be used to view results directly in JMeter UI:

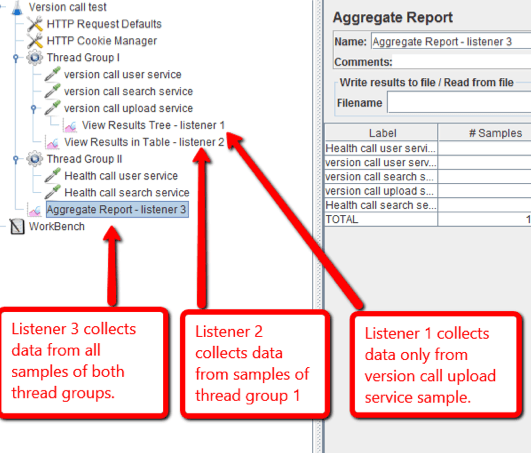
* [**View Results as Tree**](http://jmeter.apache.org/usermanual/component_reference.html#View_Results_Tree): The View Results Tree shows a tree of all sample responses, allowing you to view the response for any sample.,
* [**Graph Results**](http://jmeter.apache.org/usermanual/component_reference.html#Graph_Results): The Graph Results listener generates a simple graph that plots all sample times,
* [**Aggregate Report**](http://jmeter.apache.org/usermanual/component_reference.html#Aggregate_Report): The aggregate report creates a table row for each differently named request in your test,
* [**View Results In Table**](http://jmeter.apache.org/usermanual/component_reference.html#View_Results_in_Table): This visualizer creates a row for every sample result. Like the View Results Tree, this visualizer uses a lot of memory,
* [**Aggregate Graph**](http://jmeter.apache.org/usermanual/component_reference.html#Aggregate_Graph): The aggregate graph is similar to the aggregate report. The primary difference is the aggregate graph provides an easy way to generate bar graphs and save the graph as a PNG file,
* [**Generate Summary Results**](http://jmeter.apache.org/usermanual/component_reference.html#Generate_Summary_Results): This test element can be placed anywhere in the test plan. Generates a summary of the test run so far to the log file and/or standard output. Both running and differential totals are shown.

Some listeners have been omitted: these listeners are for debugging purpose only. These listeners help to diagnose scripting issues but are not intended to provide performance metrics, like the following ones:

* [**Comparison Assertion Visualizer**](http://jmeter.apache.org/usermanual/component_reference.html#Comparison_Assertion_Visualizer): The Comparison Assertion Visualizer shows the results of any Compare Assertion elements,
* or [**Assertion Results**](http://jmeter.apache.org/usermanual/component_reference.html#Assertion_Results): The Assertion Results visualizer shows the Label of each sample taken.

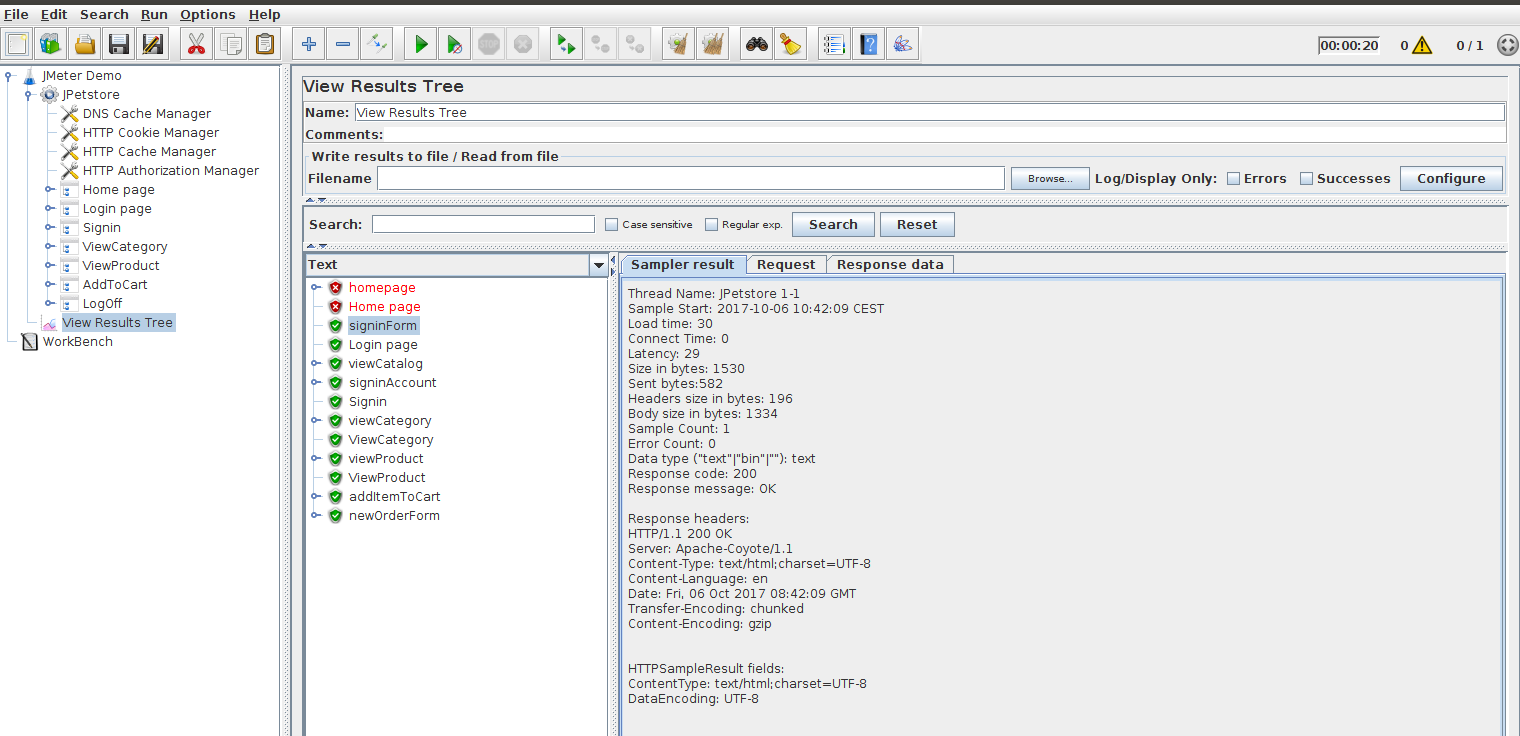
As a general rule of thumb, **avoid using UI Listeners**. They consume a lot of memory. They aren’t suitable for real load tests. Some may even trigger and Out Of Memory error with just a few concurrent threads groups running.

Placing Listeners



Depending on the location where the results listener is placed, it collects different metrics. **A JMeter results listener collects results from all elements at same level or below**. For this reason, it’s advisable to place listeners on *Test Plan* level to collect all thread groups results.

View Results Tree



The View Results Tree is essentially a **tool to debug the requests sent and responses received**. It’s useful to see if the script is running correctly. But, it’s not really suitable to view results when many concurrent users are running. It will quickly run out of memory because it keeps all the results in the main memory.

Some metrics are available when clicking on each request like the following:

Thread Name: JPetstore 1-1

Sample Start: 2017-10-06 10:42:09 CEST

Load time: 30

Connect Time: 0

Latency: 29

Size in bytes: 1530

Sent bytes:582

Headers size in bytes: 196

Body size in bytes: 1334

Sample Count: 1

Error Count: 0

Data type ("text"|"bin"|""): text

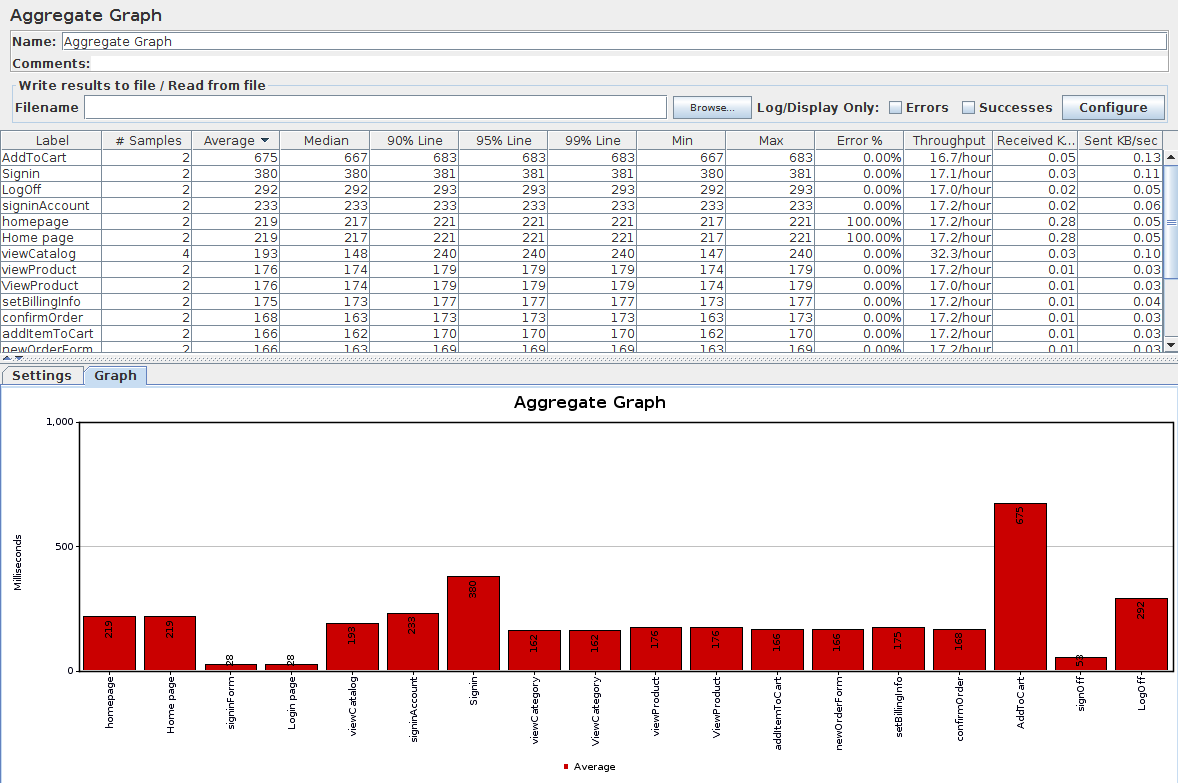
Response code: 200

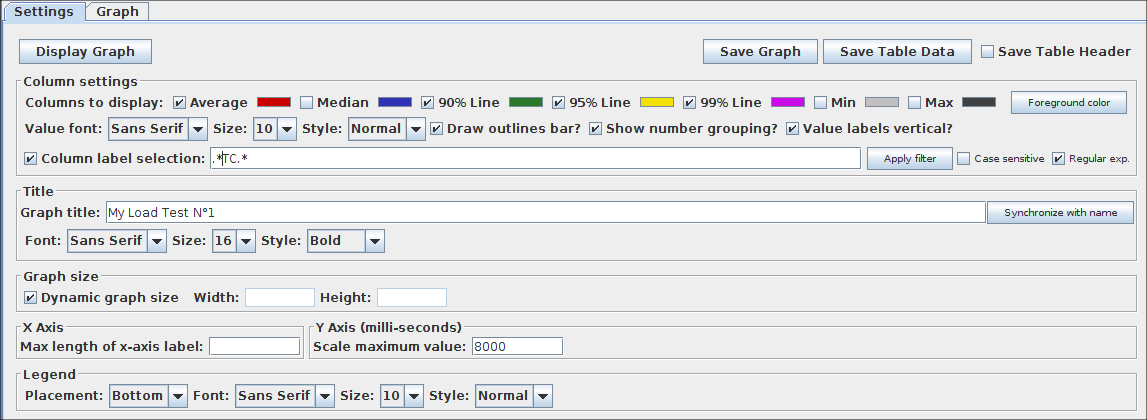
Response message: OK

I would suggest to use this listener to:

* Debug the script before scaling the test to a larger number of concurrent users,
* Define baseline performance metrics by running a single thread group for one iteration,
* and/or Use Received Responses to fix / design post processor to extract dynamic parameters.

Aggregate Graph





The aggregate graph is an UI Listener which brings some useful test-wide metrics about each request and transaction controller. It also includes a Bar Chart which can be tweaked to fit your needs with many different settings. I must say, there are **way too many settings**, and even worse, none of these settings are saved in the JMX. You loose them when you close JMeter.

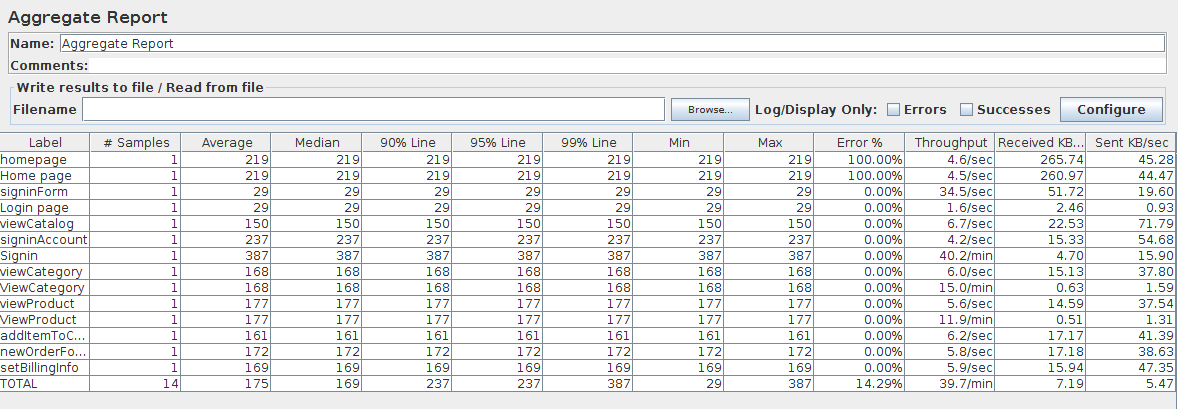
Although, I must admit it’s really nice to be able to **Export the Graph As PNG** and **Export Table as CSV** for future use in a custom designed report.

**The metrics are test-wide**, which means you get for example the average response time of a request for the entire test. the available metrics are:

* **Label**: name of the request,
* **# Samples**: total number of executions,
* **Average**: Average Elapsed Time in milliseconds,
* **Median**: The [**Median**](https://en.wikipedia.org/wiki/Median) is the value separating the higher half of a data sample, a population, or a probability distribution, from the lower half. For a data set, it may be thought of as the “middle” value,
* **90% Line**: 90% [**Percentile**](https://en.wikipedia.org/wiki/Percentile), A percentile (or a centile) is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall,
* **95% Line**: 95% Percentile,
* **99% Line**: 99% Percentile,
* **Min**: Minimum Elapsed Time,
* **Max**: Maximum Elapsed Time,
* **Errors %**: Percentage of errors (errors / (errors + samples) \* 100),
* **Throughput**: Number of samples per second,
* and **KB/sec**: Network Throughput in KiloBytes/sec.

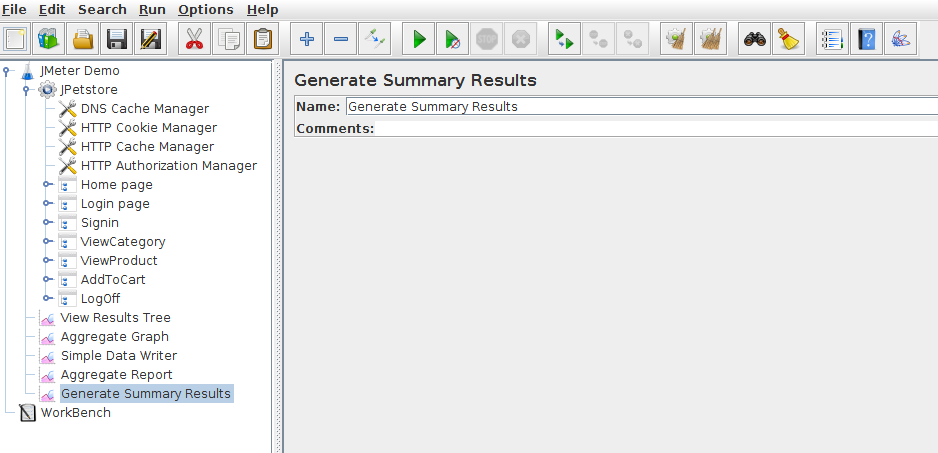
Like any other UI Listener, I wouldn’t recommend using it for real load tests.

Aggregate Report

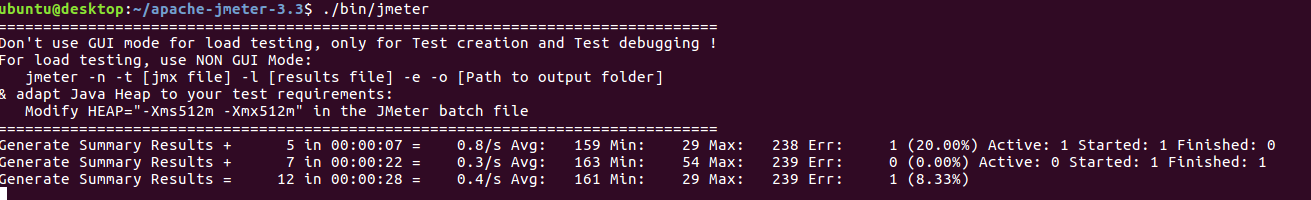


The aggregate report is very similar to the [**Aggregate Graph**](https://octoperf.com/blog/2017/10/19/how-to-analyze-jmeter-results/#aggregate-graph), containing only the metrics table. This listener can be used when running headless load tests (without the UI being launched) because **the statistics can be saved in a CSV file for later use**. It contains exactly the same metrics as the *Aggregate Graph*. These metrics can then be used to write a report using Word for example.

Generate Summary Results



JMeter summary Results listener outputs results during the load test in JMeter’s console as shown below.



It displays just a few general metrics every few seconds:

Generate Summary Results + 5 **in** 00:00:07 = 0.8/s Avg: 159 M**in**: 29 Max: 238 Err: 1 (20.00%) Active: 1 Started: 1 Finished: 0

Generate Summary Results + 7 **in** 00:00:22 = 0.3/s Avg: 163 M**in**: 54 Max: 239 Err: 0 (0.00%) Active: 0 Started: 1 Finished: 1

Generate Summary Results = 12 **in** 00:00:28 = 0.4/s Avg: 161 M**in**: 29 Max: 239 Err: 1 (8.33%)

Generate Summary Results + 17 **in** 00:00:25 = 0.7/s Avg: 185 M**in**: 28 Max: 524 Err: 3 (17.65%) Active: 3 Started: 3 Finished: 0

Generate Summary Results + 32 **in** 00:00:30 = 1.1/s Avg: 160 M**in**: 28 Max: 239 Err: 2 (6.25%) Active: 2 Started: 5 Finished: 3

Generate Summary Results = 49 **in** 00:00:55 = 0.9/s Avg: 169 M**in**: 28 Max: 524 Err: 5 (10.20%)

Generate Summary Results + 29 **in** 00:00:30 = 1.0/s Avg: 164 M**in**: 28 Max: 246 Err: 3 (10.34%) Active: 3 Started: 8 Finished: 5

Generate Summary Results = 78 **in** 00:01:25 = 0.9/s Avg: 167 M**in**: 28 Max: 524 Err: 8 (10.26%)

Generate Summary Results + 31 **in** 00:00:30 = 1.0/s Avg: 165 M**in**: 28 Max: 242 Err: 2 (6.45%) Active: 2 Started: 10 Finished: 8

Generate Summary Results = 109 **in** 00:01:55 = 0.9/s Avg: 166 M**in**: 28 Max: 524 Err: 10 (9.17%)

Generate Summary Results + 4 **in** 00:00:05 = 0.8/s Avg: 168 M**in**: 138 Max: 181 Err: 0 (0.00%) Active: 0 Started: 10 Finished: 10

Generate Summary Results = 113 **in** 00:02:00 = 0.9/s Avg: 166 M**in**: 28 Max: 524 Err: 10 (8.85%)

These log lines are already output by default when running JMeter in headless mode. [**JMeter Jenkins Plugin**](https://wiki.jenkins.io/display/JENKINS/Performance+Plugin) is capable of parsing those lines and output graphs when running JMeter on [**Jenkins**](https://jenkins.io/).

Graph Results



**JMeter Graphs Results** displays line charts for common metrics as well as number figures:

* **No of Samples**: the number of samples being processed,
* **Latest Sample**: Latest Elapsed Time in milliseconds,
* **Average Elapsed Time**: in milliseconds,
* **Standard Deviation**: in milliseconds,
* and **Throughput**: in KB/sec.

This results listener is not worth it. The graphs are barely readable. And, as explained in [**JMeter documentation**](http://jmeter.apache.org/usermanual/component_reference.html#Graph_Results):

**Graph Results MUST NOT BE USED during load test** as it consumes a lot of resources (memory and CPU). Use it only for either functional testing or during Test Plan debugging and Validation.

Summary

To summarize, most UI listeners are great for debugging / testing purpose. Don’t expect to hit high loads ( >= 500 concurrents users), use them sparingly. These listeners have been designed to quickly get metrics while running load tests inside JMeter UI, for very light loads. (<= 50 concurrent users)

It may be possible to use them even for medium load (100 - 500 concurrent users), but **don’t expect to run distributed JMeter tests with JMeter UI**. It’s not the purpose. Remember JMeter is configured with 512MB heap memory by default, which is fairly low. Although you can [**Increase JMeter allocated memory**](https://octoperf.com/blog/2017/10/17/solve-jmeter-out-of-memory-issues/), it feels coping water out of a boat which doesn’t float anymore.

Now that we have tested most of the UI listeners available in JMeter, the question is obviously: **Which Listeners can we use when running real load tests** ?

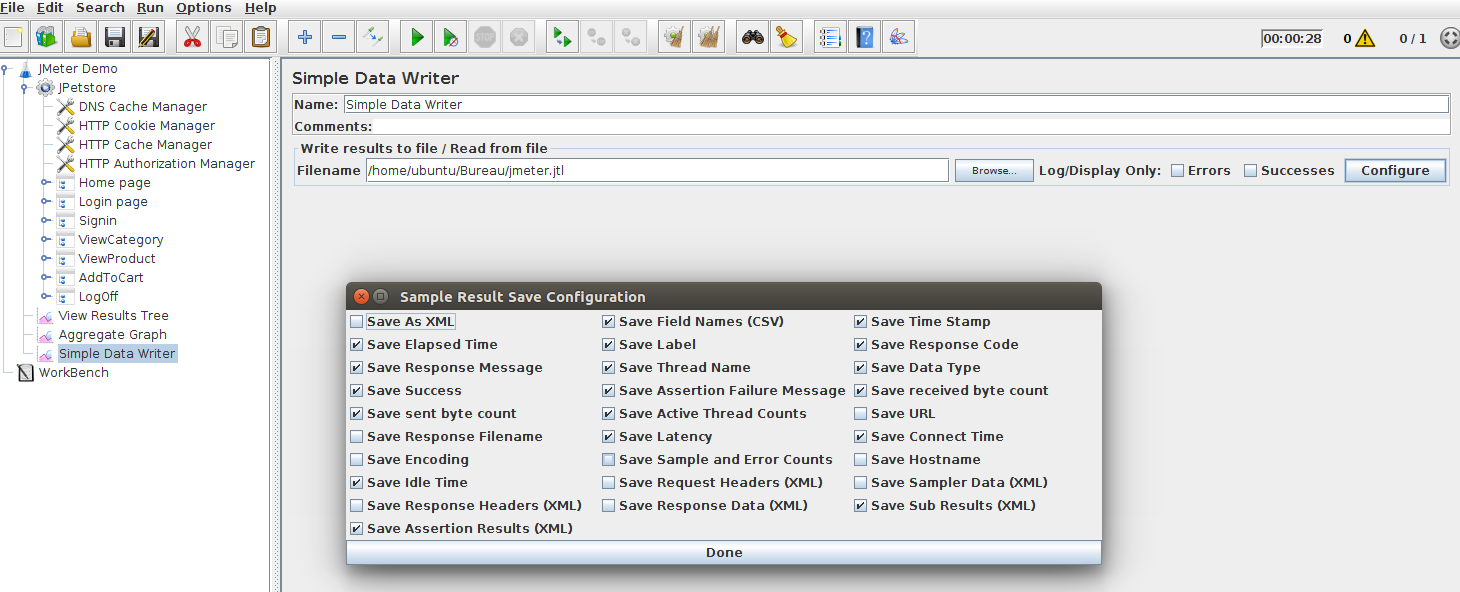
Headless Listeners

**Headless JMeter Listeners** (or non-UI) are specially designed to work when JMeter is run from the command line. Those listeners are those being used when running real load tests, because they use far less memory than UI Listeners. How? These listeners don’t keep results in memory, they are mostly in charge of offloading results to another medium.

The existing non-GUI JMeter Listeners are:

* [**Simple Data Writer**](http://jmeter.apache.org/usermanual/component_reference.html#Simple_Data_Writer): Listeners can be configured to save different items to the **result log files (JTL)**,
* [**Backend Listener**](http://jmeter.apache.org/usermanual/component_reference.html#Backend_Listener): The backend listener is an Asynchronous listener that enables you to plug custom implementations of **[BackendListenerClient](http://jmeter.apache.org/api/org/apache/jmeter/visualizers/backend/BackendListenerClient.html)**. By default, a Graphite implementation is provided.

Simple Data Writer



This is the single **most useful listener** in JMeter. It saves performance metrics according to the configuration inside an external file: the **JTL file**. [**JMeter JTL files**](https://wiki.apache.org/jmeter/JtlFiles) are the best way to analyze results, but come with a down side: **you need another tool to perform data-mining**.

There are currently two types of JTL file:

* **CSV** (default, with or without headers),
* and **XML**.

**The XML files can contain more types of information, but are considerably larger**. Therefore, it’s recommended to stick to the CSV format. The produced **jmeter.jtl** contains datas like these:

timeStamp,elapsed,label,responseCode,responseMessage,threadName,dataType,success,failureMessage,bytes,sentBytes,grpThreads,allThreads,Latency,IdleTime,Connect

1507280285885,221,Home page,,"Number of samples in transaction : 1, number of failing samples : 1",JPetstore 1-1,,false,,59592,10154,1,1,50,1,23

1507280286687,29,signinForm,200,OK,JPetstore 1-1,text,true,,1531,582,1,1,29,0,0

1507280286108,29,Login page,200,"Number of samples in transaction : 1, number of failing samples : 0",JPetstore 1-1,,true,,1531,582,1,1,29,580,0

1507280286819,147,viewCatalog,200,OK,JPetstore 1-1,text,true,,3460,11027,1,1,27,0,0

1507280287967,233,signinAccount,200,OK,JPetstore 1-1,text,true,,3719,13270,1,1,55,0,27

1507280286717,380,Signin,200,"Number of samples in transaction : 2, number of failing samples : 0",JPetstore 1-1,,true,,7179,24297,1,1,82,1104,27

1507280292035,162,viewCategory,200,OK,JPetstore 1-1,text,true,,2600,6502,1,1,56,0,26

1507280288201,162,ViewCategory,200,"Number of samples in transaction : 1, number of failing samples : 0",JPetstore 1-1,,true,,2600,6502,1,1,56,3834,26

1507280297083,174,viewProduct,200,OK,JPetstore 1-1,text,true,,2643,6804,1,1,55,0,26

1507280292198,174,ViewProduct,200,"Number of samples in transaction : 1, number of failing samples : 0",JPetstore 1-1,,true,,2643,6804,1,1,55,4886,26

1507280301651,162,addItemToCart,200,OK,JPetstore 1-1,text,true,,2827,6824,1,1,54,0,25

1507280304617,169,newOrderForm,200,OK,JPetstore 1-1,text,true,,3026,6804,1,1,55,0,27

1507280306851,173,setBillingInfo,200,OK,JPetstore 1-1,text,true,,2759,8194,1,1,63,0,28

1507280310018,163,confirmOrder,200,OK,JPetstore 1-1,text,true,,2980,6475,1,1,56,0,26

We’ll see later in this guide how we can use the results saved into the JTL file for further processing and drill-down. **JTLs are the most powerful way to analyze JMeter results.**

**Pros**:

* JTLs are plain CSV files easy to read,
* Some Web-Based tools are capable of parsing JTL files and render online reports,
* All Raw Results are saved with JTL files.

**Cons**:

* JTLs are written by each load generator on their disk. Distributed testing requires to bring them back to the controller at the end of the test,
* JTLs can grow large (several GB) and clutter the disk,
* JTLs must be data-mined with tools like Excel to get useful metrics out of them.

Let’s see how we can interpret those JTL files.

JTL Analysis with Excel

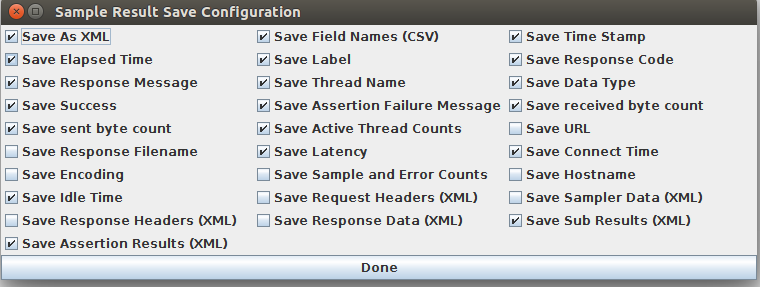
**%APACHE\_JMETER\_HOME%/extras** contains several *xsl* files which are specially designed to **process JTL files in XML format** and output nice reports. Seek for the following files:

* **jmeter-results-detail-report\_21.xsl**: Detailed JMeter Report,
* **jmeter-results-report\_21.xsl**: Basic JMeter Report.

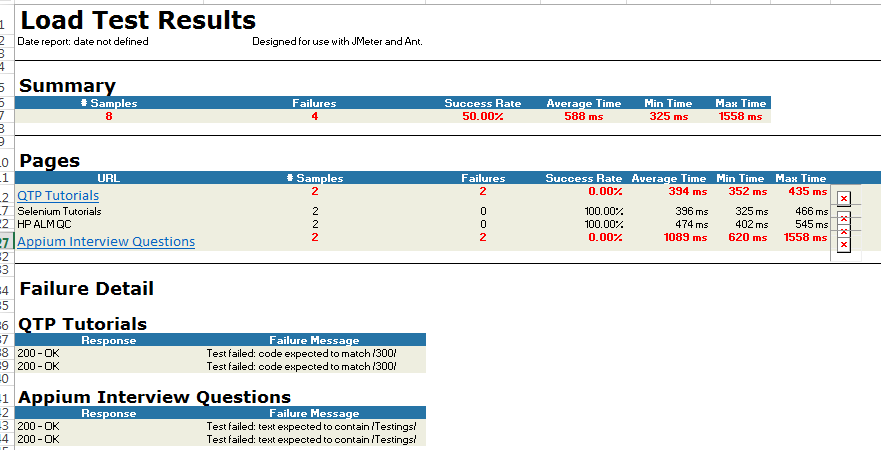
The procedure below explains how to get nice reports using those XSL stylesheets and Microsoft Excel.

**How to Analyze JTL files with Excel**

* **Simple Data Writer Listener**: Add it to your Test Plan configure it to **save the results as XML** in the JTL file,



* **Run the load test**: From APACHE\_JMETER\_HOME, run the command ./bin/jmeter -n -t jpetstore.jmx -l jmeter.jtl,
* Creating summariser <summary>
* Created the tree successfully using jpetstore.jmx
* Starting the test @ Fri Oct 06 15:03:42 CEST 2017 (1507295022425)
* Waiting **for** possible Shutdown/StopTestNow/Heapdump message on port 4445
* summary + 12 **in** 00:00:18 = 0.7/s Avg: 187 M**in**: 30 Max: 418 Err: 2 (16.67%) Active: 2 Started: 2 Finished: 0
* summary + 27 **in** 00:00:29 = 0.9/s Avg: 168 M**in**: 29 Max: 270 Err: 2 (7.41%) Active: 2 Started: 4 Finished: 2
* summary = 39 **in** 00:00:47 = 0.8/s Avg: 173 M**in**: 29 Max: 418 Err: 4 (10.26%)
* summary + 33 **in** 00:00:31 = 1.1/s Avg: 163 M**in**: 28 Max: 259 Err: 3 (9.09%) Active: 2 Started: 7 Finished: 5
* summary = 72 **in** 00:01:18 = 0.9/s Avg: 169 M**in**: 28 Max: 418 Err: 7 (9.72%)
* summary + 27 **in** 00:00:29 = 0.9/s Avg: 165 M**in**: 29 Max: 246 Err: 2 (7.41%) Active: 2 Started: 9 Finished: 7
* summary = 99 **in** 00:01:47 = 0.9/s Avg: 168 M**in**: 28 Max: 418 Err: 9 (9.09%)
* summary + 14 **in** 00:00:13 = 1.1/s Avg: 163 M**in**: 28 Max: 246 Err: 1 (7.14%) Active: 0 Started: 10 Finished: 10
* summary = 113 **in** 00:02:00 = 0.9/s Avg: 167 M**in**: 28 Max: 418 Err: 10 (8.85%)
* Tidying up ... @ Fri Oct 06 15:05:43 CEST 2017 (1507295143106)
* ... end of run
* **Edit the JTL**: add <?xml-stylesheet type="text/xsl" href="PATH\_TO\_jmeter-results-report\_21.xsl"?> after <?xml version="1.0" encoding="UTF-8"?>,
* **Save JTL**,
* **Open Microsoft Excel**: then drag’n drop the JTL file inside it.



Please note that **it doesn’t work with Open Office**. Only Microsoft Office is supported.

With the newly available [**JMeter Report Dashboard**](http://jmeter.apache.org/usermanual/generating-dashboard.html), this legacy solution is not so appealing anymore. The report looks old-fashioned compared to the new JMeter HTML report available since JMeter 3.0.

HTML Report DashBoard

The HTML Report Dashboard can be generated at the end of the test using a separate command line. This report is pretty rich and displays many different metrics. For a complete list of all customisable settings, please see [**Generating Dashboard**](http://jmeter.apache.org/usermanual/generating-dashboard.html) on JMeter’s website.

Once you have a JTL containing all the results, run:

./bin/jmeter -g JTL\_FILE -o OUTPUT\_FOLDER

Where:

* **-g JTL\_FILE**: relative or full path to the JTL file. Example: jmeter.jtl,
* **-o OUTPUT\_FOLDER**: the folder in which the HTML report should be written.

The command-line execution may take a while depending on the JTL file size. Once finished, no error should be displayed within the terminal. The report is ready in the given output folder.

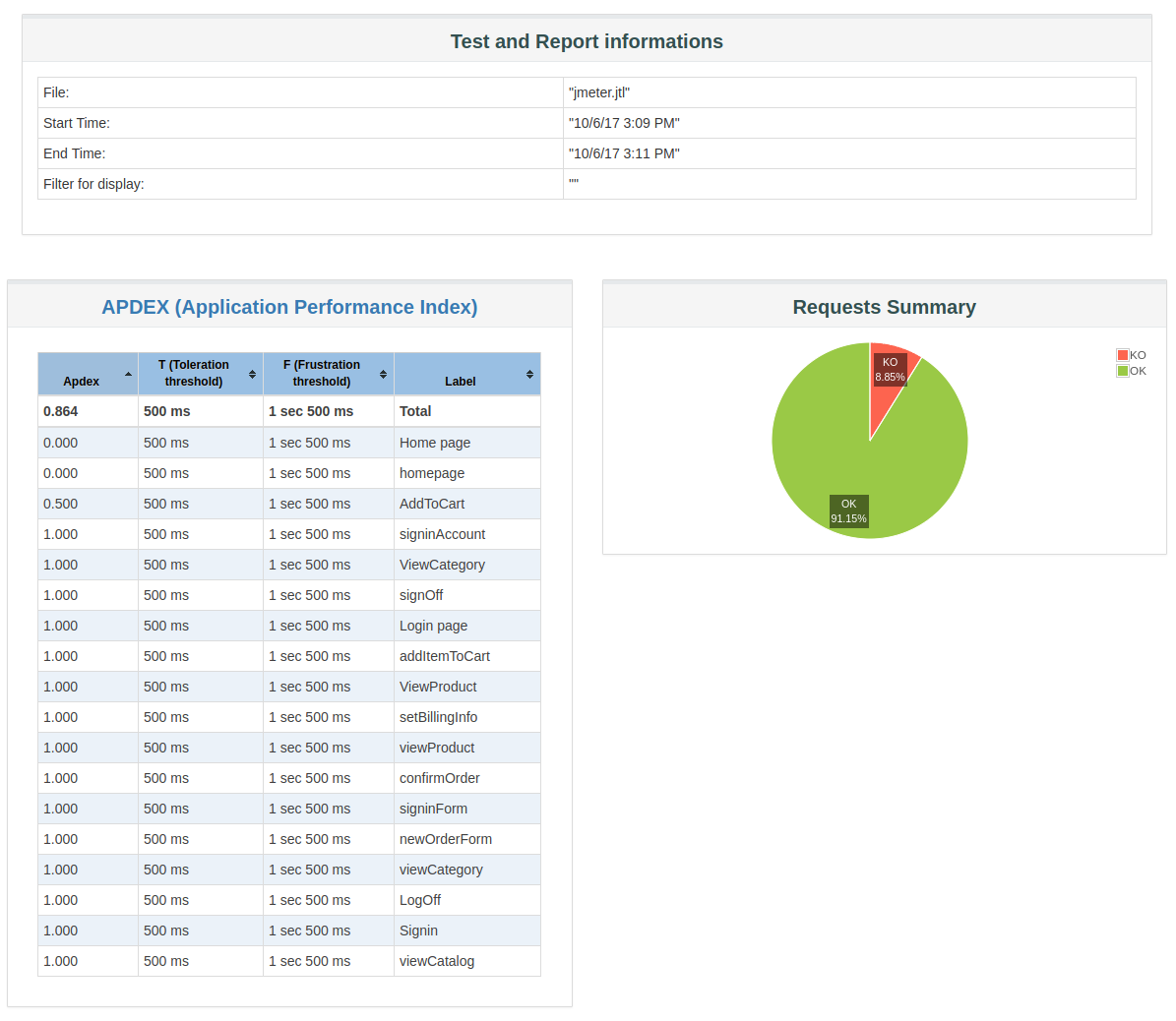
**Pros**:

* HTML Report is easy to generate,
* Graphs and Tables are well designed,
* You can select / deselect requests and/or transactions on each graph.

**Cons**:

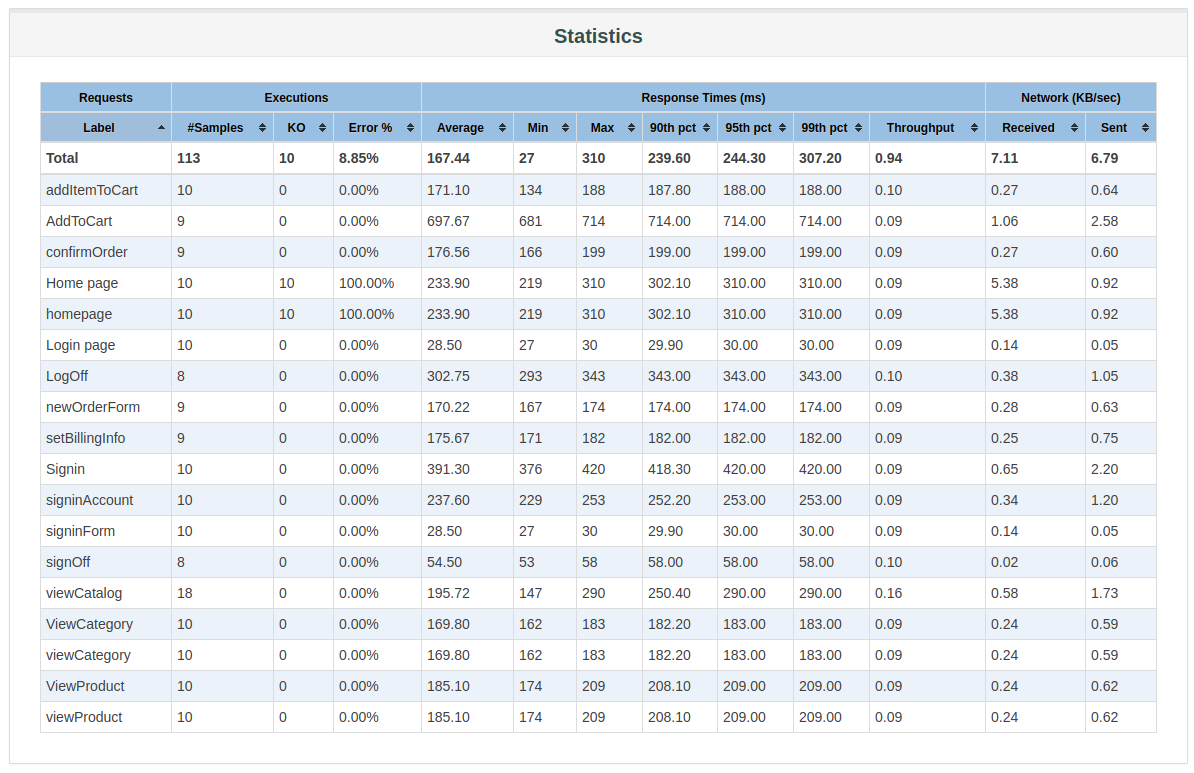
* Too Many customisation settings, where to start?
* The report cannot be fully customized by adding text, images and more. It’s a static report.

Since JMeter 3.0, HTML Report Dashboard is a huge step forward simplifying JMeter test result analysis.



The Report Summary contains the following information:

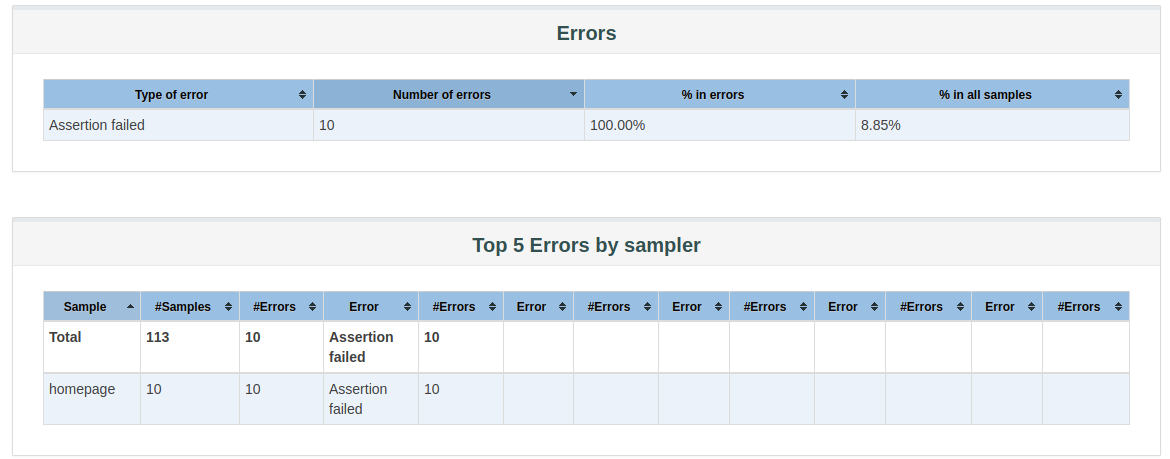
* Test start time and end time,
* APDEX scores for every single request and container,
* A pie chart named *Requests Summary* which gives the proportion of Successful / Failed samples.



The Statistics table provides global test statistics for every single request which has been executed:

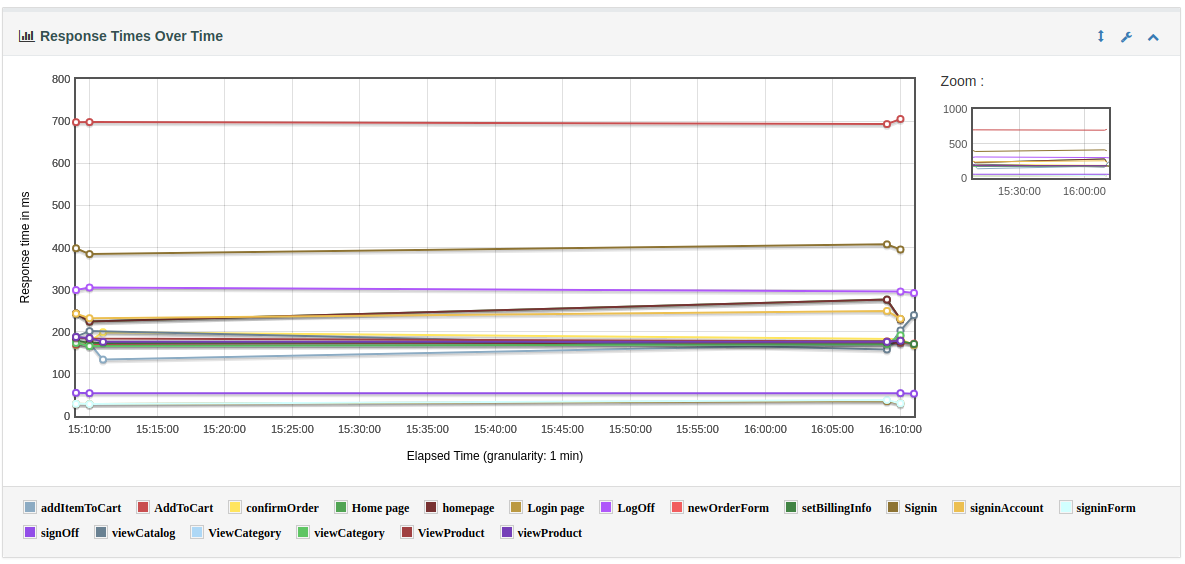
* **Executions**: Number of hits and errors,
  + **# Samples**: Total Number of samples executed,
  + **KO**: Total Number of samples failed to execute,
  + **Errors %**: Percent of errors,
* **Response Times (ms)**: Response times in milliseconds,
  + **Average**: Average Elapsed Time,
  + **Min**: Minimum Elapsed Time,
  + **Max**: Maximum Elapsed Time,
  + **90th pct**: 90th Percentile,
  + **95th pct**: 95th Percentile,
  + **99th pct**: 99th Percentile,
  + **Throughtput**: Number of hits per second,
* **Network**: throughput in KB/sec
  + **Received**: KB received per second,
  + **Sent**: KB sent per second.

The lines can be ordered by any of the statistic above, making it easy to find requests that cause bottlenecks. **Order Requests by decreasing Average**, you should see the slowest request being first in the statistics table.

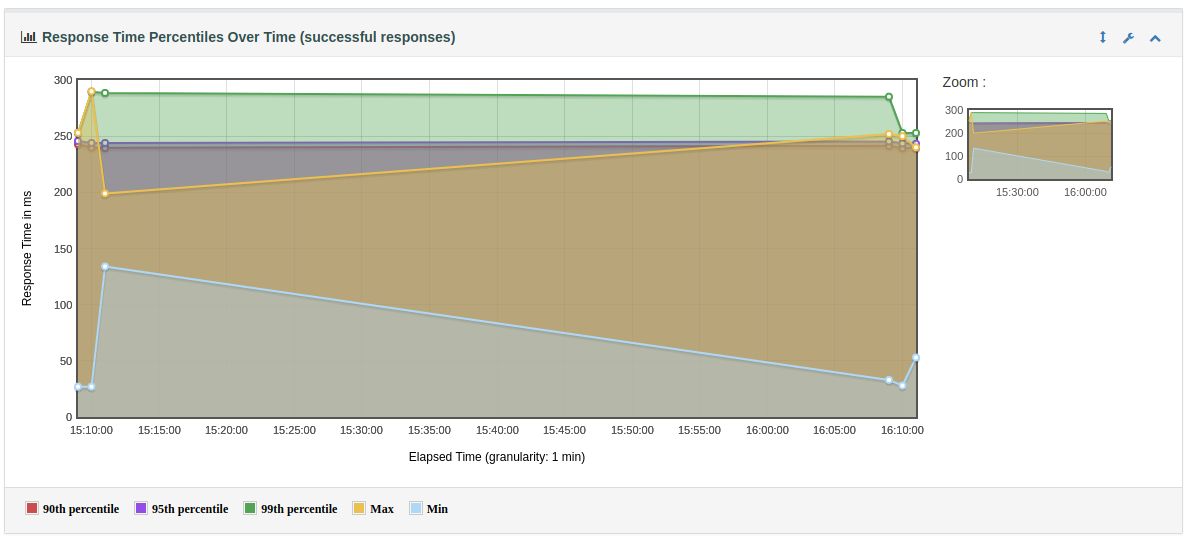


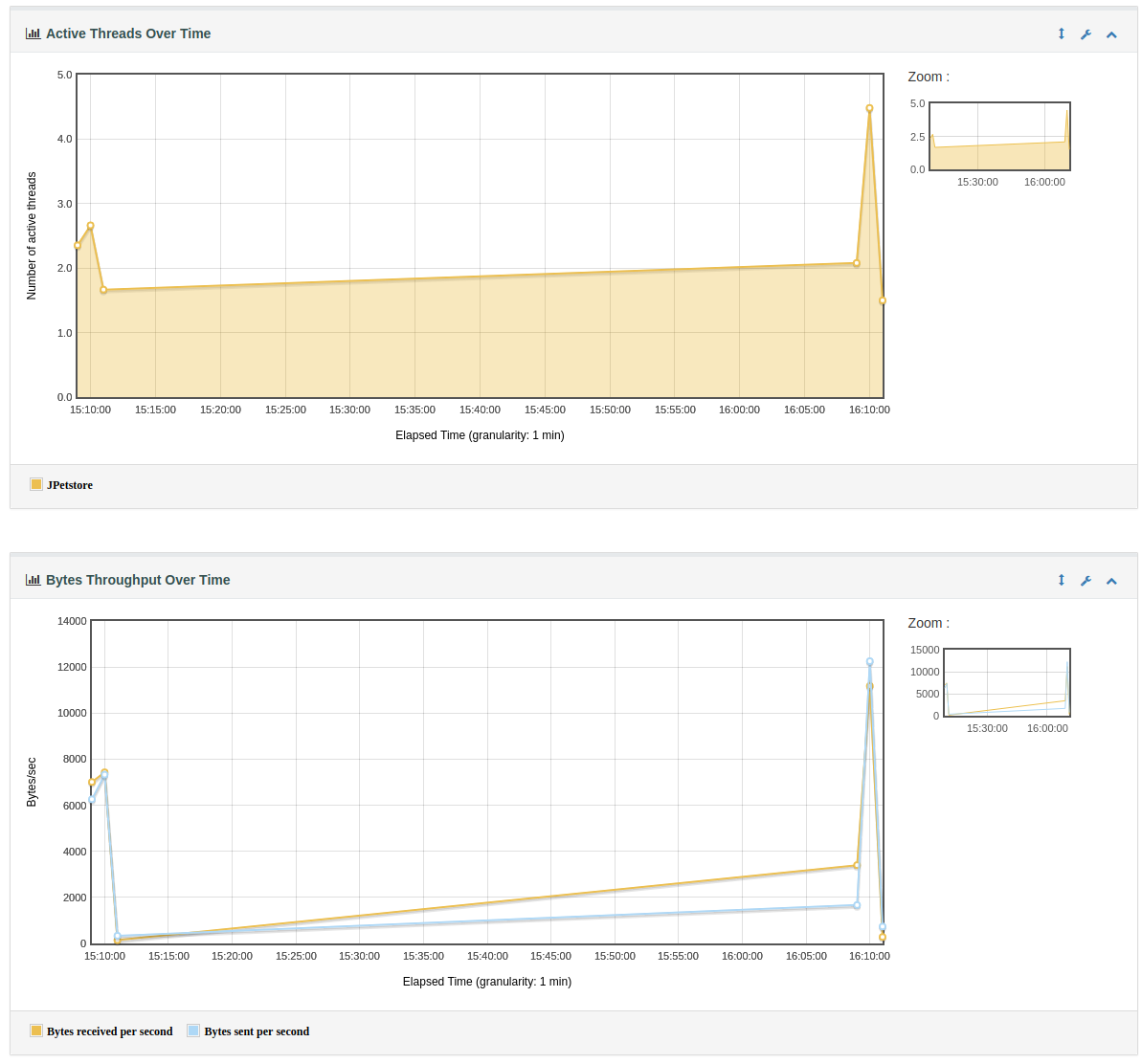
The errors table give more details about the errors encountered during the load test. For each type of error, you will see:

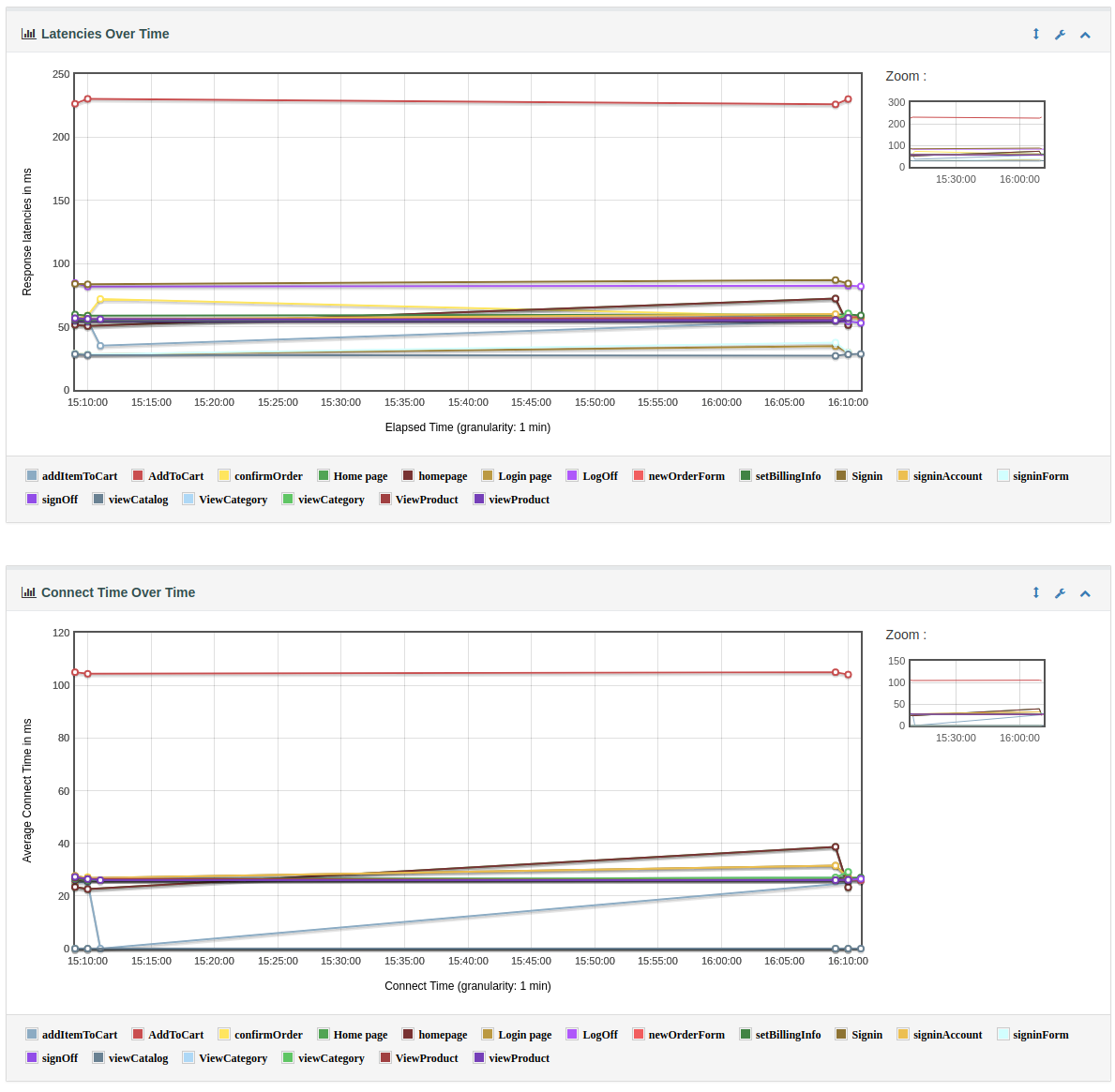
* **Number of errors**: how many errors occured,
* **% in errors**: percentage of requests in error,
* **% in all samples**: Percentage of errors compared to total number of samples.

*Response Time Over Time Chart*

This chart displays the average response time of each transaction over the course of the entire test. Sadly, if you have a lot of transactions, the graph may look cluttered because all the transactions are displayed on it.

*Response Time Percentiles*

*Active Threads, Throughput over Time*

*Active Threads, Throughput over Time*

There are many other graphs available:

* **Throughput**:
  + **Hits Per Second (excluding embedded resources)**: number of hits per second over time,
  + **Codes Per Second (excluding embedded resources)**: HTTP Codes per second over time (200 OK, 500 Internal Error etc.)
  + **Transactions Per Second**: transactions (related to Transaction Controller) per second over time,
  + **Response Time Vs Request**: The response time compared to requests per second,
  + **Latency Vs Request**: Latency compared to requests per second,
* **Response Times**:
  + **Response Time Percentiles**: Elapsed Time per percentile in 10% increments,
  + **Response Time Overview**: Gives the percent of requests per Apdex Range (Satisfying, Tolerating and Frustrating),
  + **Time Vs Threads**: Elapsed Time per Active Threads, to see how the elapsed time degrades when load increases,
  + **Response Time Distribution**: how Elapsed Time is spread between Min and Max elapsed time.

The HTML report is clearly **a good step to catch up with some expensive tools like LoadRunner** or NeoLoad. Sure, it could have been way more customisable to taylor a report which fits your needs. Anyway, it’s a huge leap forward in improving JMeter test results analysis compared to the integrated UI listeners.

Considering JMeter is an open-source load testing tool, available for free, I’m impressed to see how many tools there are to analyze test results. And we’re not even finished yet!

Backend Listener

JMeter’s [**Backend Listener**](http://jmeter.apache.org/usermanual/realtime-results.html) allows to plug an external database to store test results and performance metrics.

In this section, we’re going to combine several open-source tools to collect and visualize JMeter results in real-time:

* [**InfluxData**](https://www.influxdata.com/): database used as a temporary metrics storage to store performance metrics,
* [**Grafana**](https://grafana.com/): Grafana is an open-source platform for time series analytics, which allows you to create real-time graphs based on time series data,
* [**JMeter’s Backend Listener**](http://jmeter.apache.org/usermanual/realtime-results.html): the backend listeners collect JMeter metrics and sends them to the temporary metrics storage.

Exposed Metrics

JMeter sends metrics the the time-series database. The list below describes the metrics being available.

* **Thread Metrics**:
  + **ROOT\_METRICS*PREFIX* test.minAT**: Minimum active threads,
  + **ROOT\_METRICS*PREFIX* test.maxAT**: Maximum active threads,
  + **ROOT\_METRICS*PREFIX* test.meanAT**: Mean active threads,
  + **ROOT\_METRICS*PREFIX* test.startedT**: Started threads,
  + **ROOT\_METRICS*PREFIX* test.endedT**: Finished threads.
* **Response Time Metrics**:
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ok.count**: Number of successful responses for sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .h.count**: Server hits per seconds, this metric cumulates Sample Result and Sub results (if using Transaction Controller, “Generate parent sampler” should be unchecked),
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ok.min**: Min response time for successful responses of sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ok.max**: Max response time for successful responses of sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ok.avg**: Average response time for successful responses of sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ok.pct\_PERCENTILE\_VALUE**: Percentile computed for successful responses of sampler name. There will be one metric for each calculated value,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ko.count**: Number of failed responses for sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ko.min**: Min response time for failed responses of sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ko.max**: Max response time for failed responses of sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ko.avg**: Average response time for failed responses of sampler name,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .ko.pct\_PERCENTILE\_VALUE**: Percentile computed for failed responses of sampler name. There will be one metric for each calculated value,
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .a.count**: Number of responses for sampler name (sum of ok.count and ko.count),
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .a.min**: Min response time for responses of sampler name (min of ok.count and ko.count),
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .a.max**: Max response time for responses of sampler name (max of ok.count and ko.count),
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .a.avg**: Average response time for responses of sampler name (avg of ok.count and ko.count),
  + **ROOT\_METRICS\_PREFIX\_\_ SAMPLER\_NAME .a.pct\_PERCENTILE\_VALUE**: Percentile computed for responses of sampler name. There will be one metric for each calculated value. (calculated on the totals for OK and failed samples).

The following constants are:

* **ROOT\_METRICS\_PREFIX\_**: root metrics prefix. There is none when using **InfluxBackendListenerClient**,
* **SAMPLER\_NAME**: name of the sample within the JMX script,
* **PERCENTILE\_VALUE**: 90, 95 or 99 by default. Depends on the backend listener configuration.

InfluxDB Setup

We’re going to download and install influxDB:

* [**Download InfluxDB**](https://portal.influxdata.com/downloads),
* **Install InfluxDB**, here is the setup for Ubuntu using A Debian package:
* ubuntu@desktop:~$ wget https://dl.influxdata.com/influxdb/releases/influxdb\_1.3.6\_amd64.deb
* ubuntu@desktop:~$ sudo dpkg -i influxdb\_1.3.6\_amd64.deb
* Selecting previously unselected package influxdb.
* (Reading database ... 264577 files and directories currently installed.)
* Preparing to unpack influxdb\_1.3.6\_amd64.deb ...
* Unpacking influxdb (1.3.6-1) ...
* Setting up influxdb (1.3.6-1) ...
* Created symlink from /etc/systemd/system/influxd.service to /lib/systemd/system/influxdb.service.
* Created symlink from /etc/systemd/system/multi-user.target.wants/influxdb.service to /lib/systemd/system/influxdb.service.
* ubuntu@desktop:~$

InfluxDB setup can vary depending on your operating system. Please see **[InfluxDB Installation](https://docs.influxdata.com/influxdb/v1.3/introduction/installation/)** for more information.

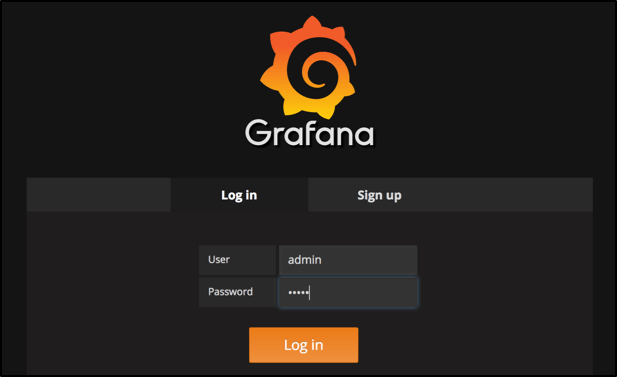
* Start influxdb service by running ubuntu@desktop:~$ sudo service influxdb start,
* Run the command influx in terminal to connect to the database,
* Create JMeter’s database:
* ubuntu@desktop:~$ influx
* Connected to http://localhost:8086 version 1.3.6
* InfluxDB shell version: 1.3.6
* > show databases
* name: databases
* name
* ----
* \_internal
* > CREATE DATABASE jmeter
* >

Great, **InfluxDB is up and running!**

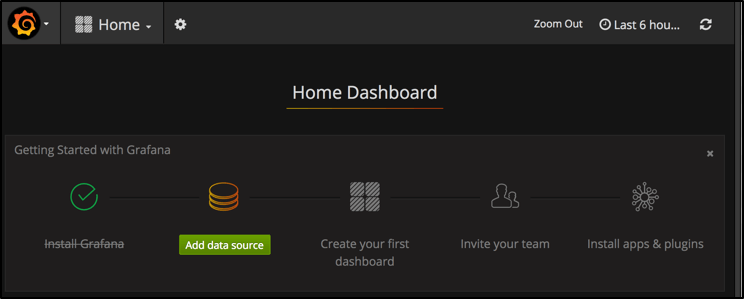
Grafana Setup

Grafana is the dashboard which will allow use to visualize the metrics sent by JMeter to the InfluxDB Database.

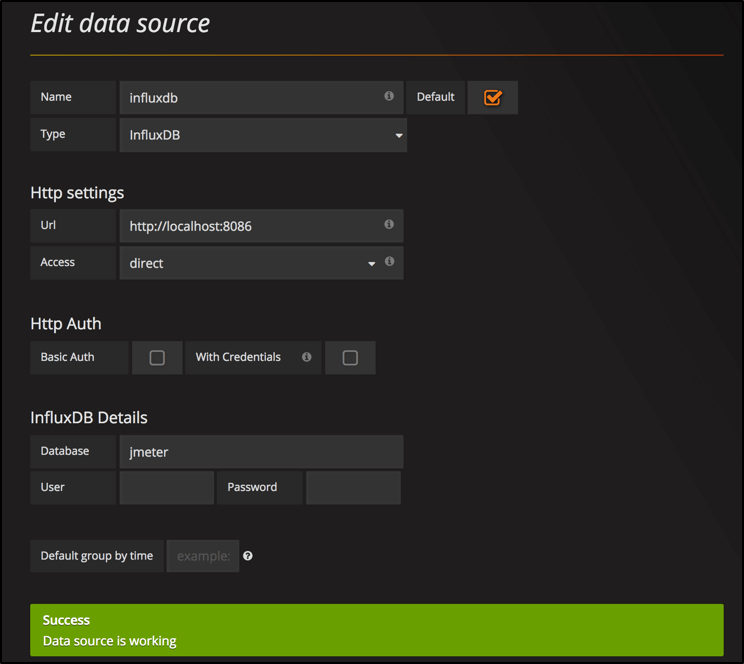
* [**Download Grafana**](https://grafana.com/grafana/download) and install it (Ubuntu setup here):
* wget https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana\_4.5.2\_amd64.deb
* sudo dpkg -i grafana\_4.5.2\_amd64.deb
* Browse to http://localhost:3000 to open grafana dashboard. Use admin as login and password.



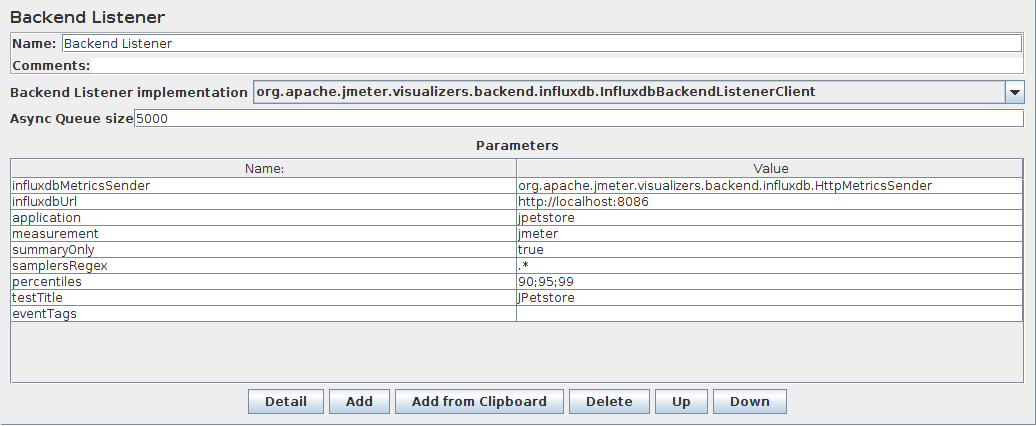
* Select *Add DataSource* option,



* Then configure DataSource with the following settings:
  + **Name**: influxdb, any name should work,
  + **Type**: InfluxDB, as we connect to an InfluxDB Database,
  + **Url**: http://localhost:8086/,
  + **Access**: Direct, because it’s direct connection to the database,
  + **Database**: jmeter, the previously created database.



BackendListener Setup



Now, let’s add a backend listener to our *Test Plan*:

* Open JMeter, then open the sample JMX Script,
* Right-click on the test plan, and select **Add > Listener > Backend Listener**,
* Configure the backend listener with the following settings:
  + **influxdbMetricsSender**: implementation class for sending metrics to InfluxDB. As of JMeter 3.2, InfluxDB is available without the need to add any additional plugin,
  + **influxDbUrl**: InfluxDB database url, the url of InfluxDB in format: **http://[influxdb\_host]:[influxdb\_port]/write?db=[database\_name]**. As we have created the jmeter database and we are running it on local machine with default port then in our case url will be: http://127.0.0.1:8086/write?db=jmeter
  + **application**: name of the application. This parameter allows to group metrics by name, thus allowing to use the same database for multiple different tests,
  + **measurement**: name of measurement that will be stored in InfluxDB (text-based line InfluxDB internal protocol to store metrics). Use default ‘jmeter’ for that property,
  + **summaryOnly**: put false if you want to keep detailed metrics in the database,
  + **samplesRegex**: allows to filter results being stored by sampler name,
  + **percentiles**: defines the percentiles being processed and sent to InfluxDB, 90;95;99 by default,
  + **testTitle**: we use *JPetstore* here,
  + **eventTags**: a list of tags which will be stored in the ‘events’ measurement of InfluxDB.

Running a test

Now, it’s time to run the test within JMeter. Either launch the test in GUI or non-GUI mode.

To check that the results are properly sent to InfluxDB, run the following command:

curl 'http://localhost:8086/query?pretty=true' --data-urlencode "db=jmeter" --data-urlencode "q=SHOW SERIES"

{

"results": [

{

"statement\_id": 0,

"series": [

{

"columns": [

"key"

],

"values": [

...

[

"events,application=jpetstore,title=ApacheJMeter"

],

[

"jmeter,application=jpetstore,responseCode=0,responseMessage=Number\\ of\\ samples\\ in\\ transaction\\ :\\ 1\\,\\ number\\ of\\ failing\\ samples\\ :\\ 1,transaction=Home\\ page"

],

...

]

}

]

}

]

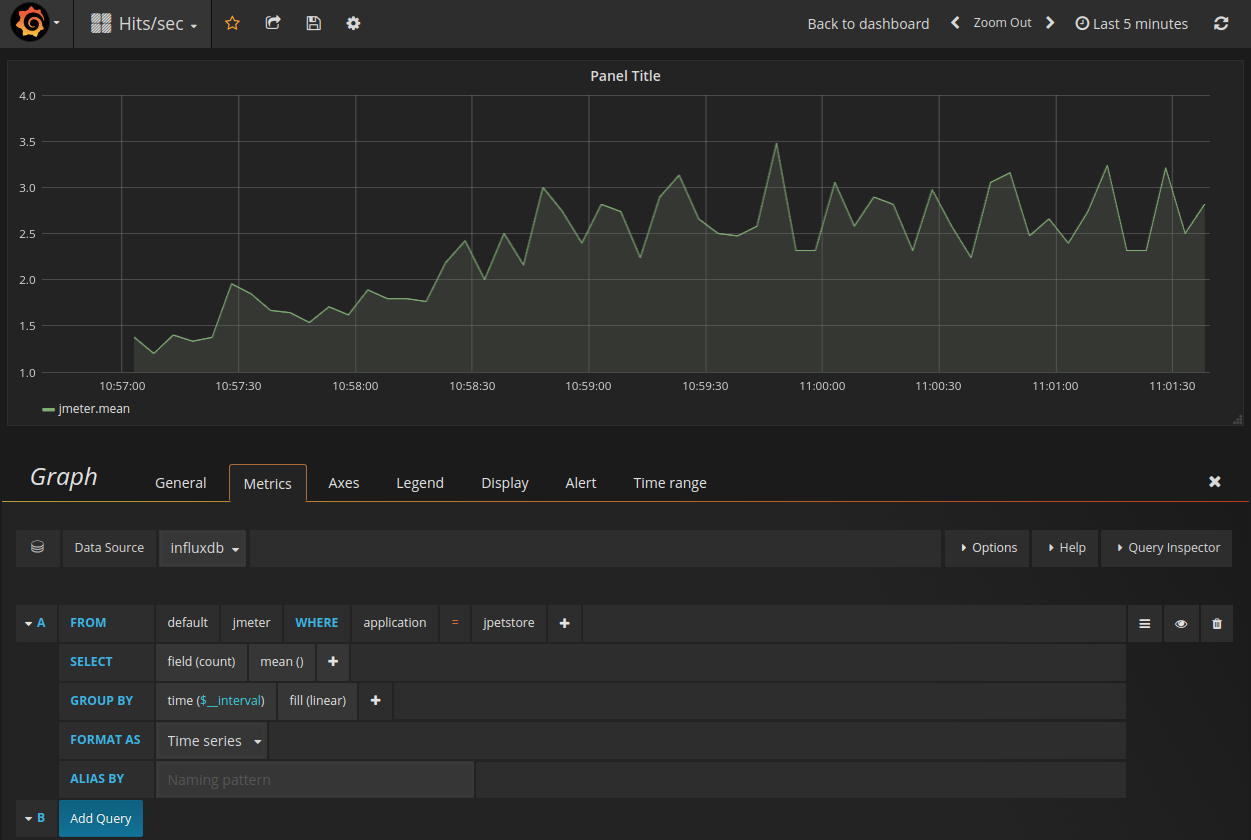
}

The returned Json document should contain several values. Let’s configure a Grafana dashboard to visualize **Hits/sec**.

**Creating a JMeter Dashboard**

* Select *Create your first dashboard*,
* Select *Graph*,
* Click on *Panel Title* then *Edit*,
* Now, let’s configure the metrics:
  + **Data Source**: select the InfluxDB data-source previously configured,
  + **FROM**: default *jmeter*, WHERE *application* = *jpetstore*,
  + **SELECT**: field *count* *mean()*, which is the average number of samples,
  + **GROUP BY**: *time($\_interval)* *fill(linear)*, to get a beautiful line chart,
  + **FORMAT AS**: *Time Series*.

It should produce the graph shown on the screenshot below.

*Hits/sec graph in Grafana using JMeter BackendListener*

NovaTec APM Dashboard

Configuring a grafana dashboard yourself is a tedious and difficult task, especially if you have no extended knowledge in querying metrics. They published a pre-configured [**JMeter Load Test Dashboard**](https://grafana.com/dashboards/1152).

**This dashboard only works with the following backend listener plugin**: [**JMeter InfluxDB Writer**](https://github.com/NovaTecConsulting/JMeter-InfluxDB-Writer/releases)

**Install JMeter InfluxDB Writer**

* Download the plugin from [**JMeter InfluxDB Writer Download Page**](https://github.com/NovaTecConsulting/JMeter-InfluxDB-Writer/releases),
* Copy the plugin JMETER\_HOME/lib/ext,
* Restart JMeter.

**Create Dedicated Database**

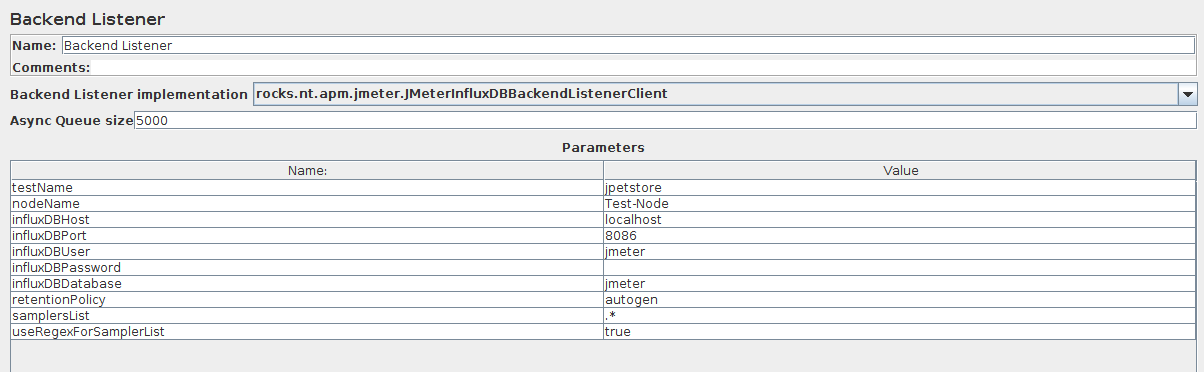
This setup requires a separate database:

* Create new database in InfluxDB named novatec using the following command:
* ubuntu@desktop:~$ curl -i -XPOST http://localhost:8086/query --data-urlencode "q=CREATE DATABASE novatec"
* HTTP/1.1 200 OK
* Connection: close
* Content-Type: application/json
* Request-Id: b04edfe5-acd4-11e7-8647-000000000000
* X-Influxdb-Version: 1.3.6
* Date: Mon, 09 Oct 2017 09:31:54 GMT
* Transfer-Encoding: chunked
* {"results":[{"statement\_id":0}]}

**Configure JMeter InfluxDB Writer**

* Open JMeter, then open the sample JMX Script,
* Right-click on the test plan, and select **Add > Listener > Backend Listener**,
* Configure the backend listener with the following settings:
  + **testName**: jpetstore,
  + **nodeName**: Test-Node,
  + **influxDBPort**: 8086,
  + **influxDBUser**: jmeter,
  + **influxDBPassword**: none,
  + **influxDBDatabase**: novatec.

Let other settings with default settings.

*JMeter BackendListener using NovaTec InfluxDB Writer Plugin*

**Create new Data-Source Novatec in Grafana**

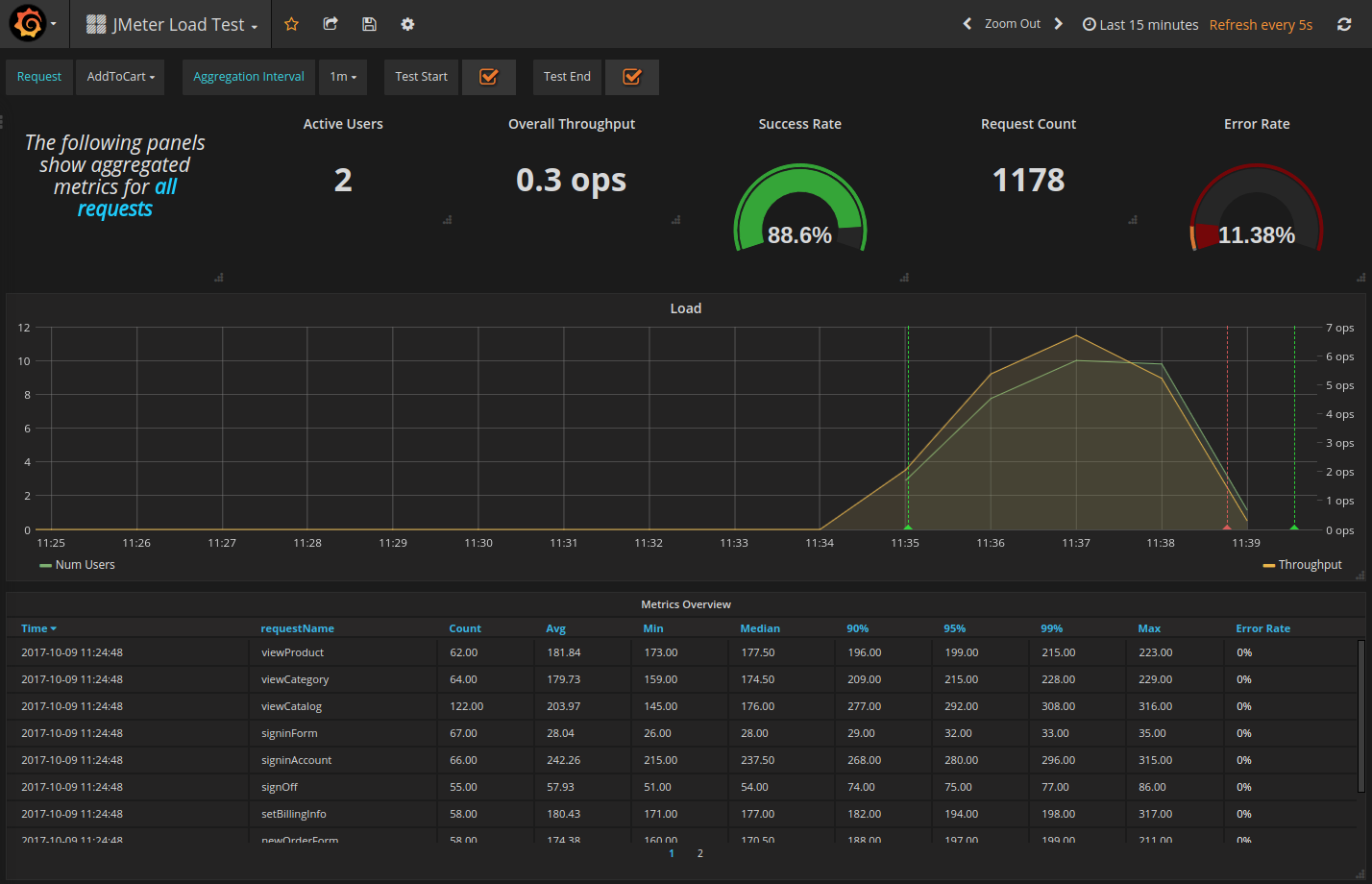
* Create a new datasource mapped on the database novatec.

**Import Novatec Dashboard**

Please follow the documentation explaining [**how to import a Grafana Dashboard**](http://docs.grafana.org/reference/export_import/) in details.

* Open Grafana,
* Select *Import new Dashboard*,
* Enter ID 1152 which is the ID of the Novatec Dashboard,
* Select the data-source pointing the the novatec database.

You should be able to see animated graphs in the dashboard.

*JMeter Novatec Dashboard in Grafana*

This dashboard offers many interesting metrics through graphs, pie-charts and more:

* **Active Users**: currently running threads,
* **Overall Throughput**: Operations per second,
* **Success Rate**: percentage of requests which have succeeded,
* **Request Count**: total number of requests executed,
* **Error Rate**: percentage of requests which have failed,
* **Metrics Overview**: a table displaying all the metrics, one line per request,
* and more!

InfluxDB Studio

[**InfluxDB Studio**](https://github.com/CymaticLabs/InfluxDBStudio) is a UI management tool for InfluxDB. It runs on Windows and allow to administer InfluxDB databases using a user-friendly UI.

Recommended Setup

We strongly recommend using the NovaTec plugin combined with the NovaTec JMeter Dashboard. It provides an out-of-the-box dashboard with many interesting metrics, ready to use. Configuring Grafana by yourself can be difficult and requires knowledge about how InfluxDB queries work.

Saas JMeter Solutions

As we have seen, a complete working setup using the BackendListener can take quite a bit of time to setup. And we’re not even talking about maintenance and updates. This is why Cloud solution like **[OctoPerf](https://octoperf.com/)**, **[Blazemeter](https://blazemeter.com/)** or [**Flood.io**](https://flood.io/) are emerging.

These Saas tools provide a facility to run JMeter tests and collect metrics. Each tool has its own reporting system based on proprietary technologies. We’re going to explore each tool here and compare their reporting capabilities. The goal is to get an overview of the reporting capabilities of each **JMeter Cloud Solution**.

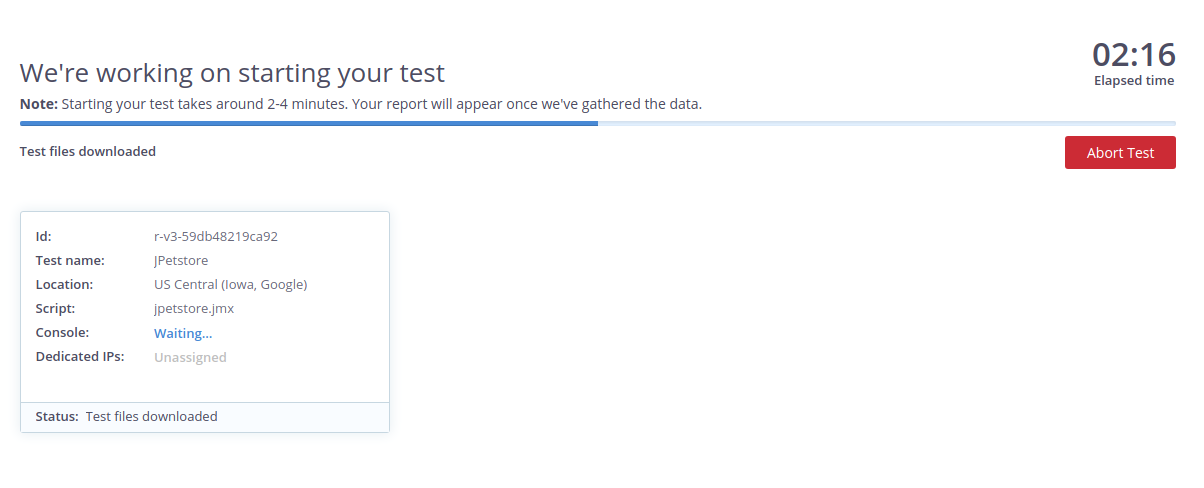
Each tool will be used to run the same test:

* 20 concurrent users,
* 10 min test duration,
* From any location available with a free account.

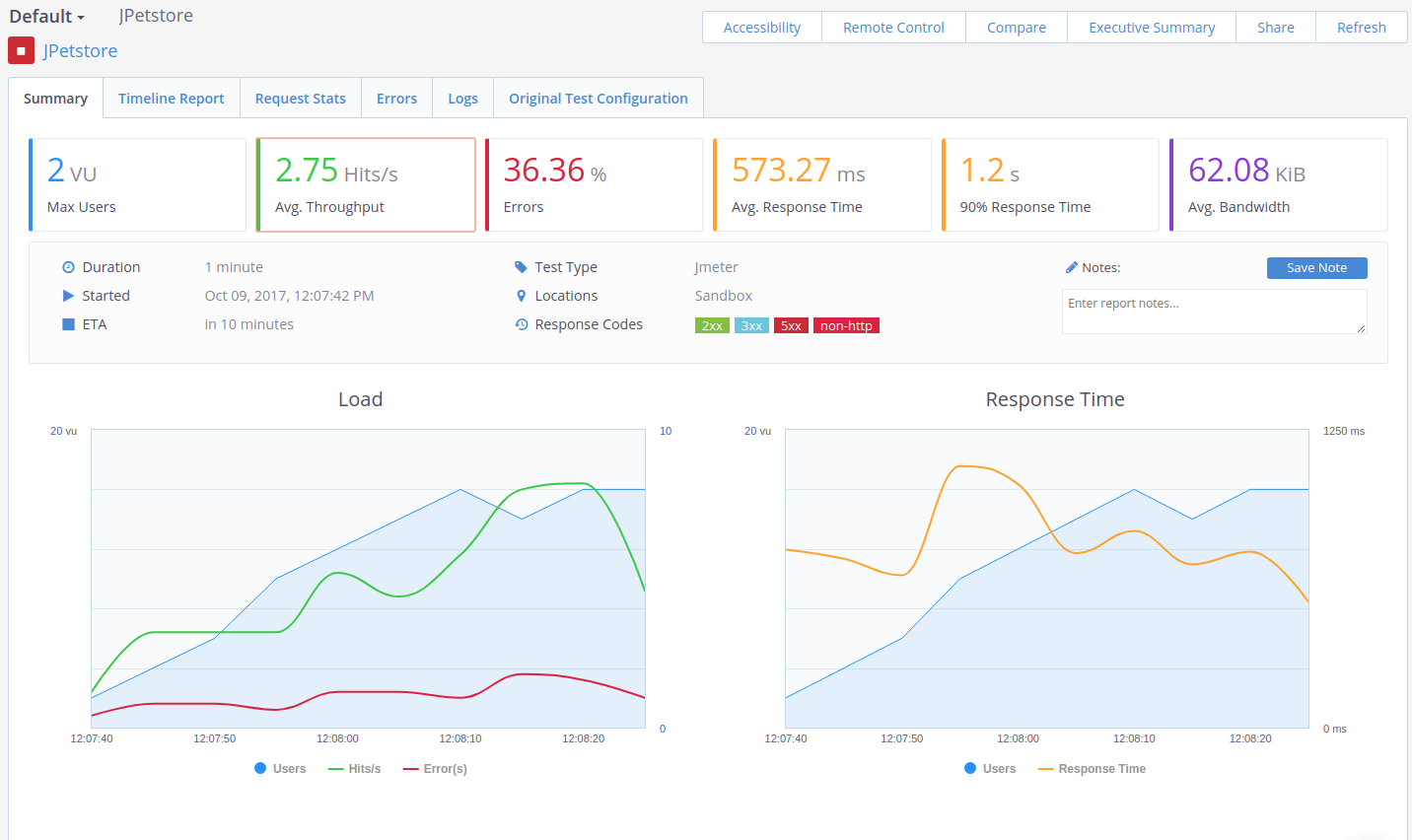
Please keep in mind that we’re trying to be as objective as possible. There are many other tools on the market enabling JMeter results analysis. As a result, we selected only the most popular tools.

Blazemeter

**Blazemeter** is the first tool which came on the market to allow users to scale their load tests in the cloud. Blazemeter is an american company founded by [**Alon Girmonsky**](https://www.crunchbase.com/person/alon-girmonsky) in December 2011.

*Starting a Test on Blazemeter*

Summary Report

*Blazemeter Summary Report*

The summary report provides the following statistics:

* **Max Users**: maximum number of concurrent users,
* **Avg Throughput**: hits per second,
* **Errors %**: percentage of errors,
* **Average Response Time**: average response time in milliseconds,
* **90% Response Time**: 90% Percentile response time,
* **Average Bandwidth**: average KiB per second during the test.

It includes two graphs:

* **Load Graph**: displays hits/sec, errors/sec and concurrent users curves,
* **Response Time Graph**: displays concurrent users and average response time curves.

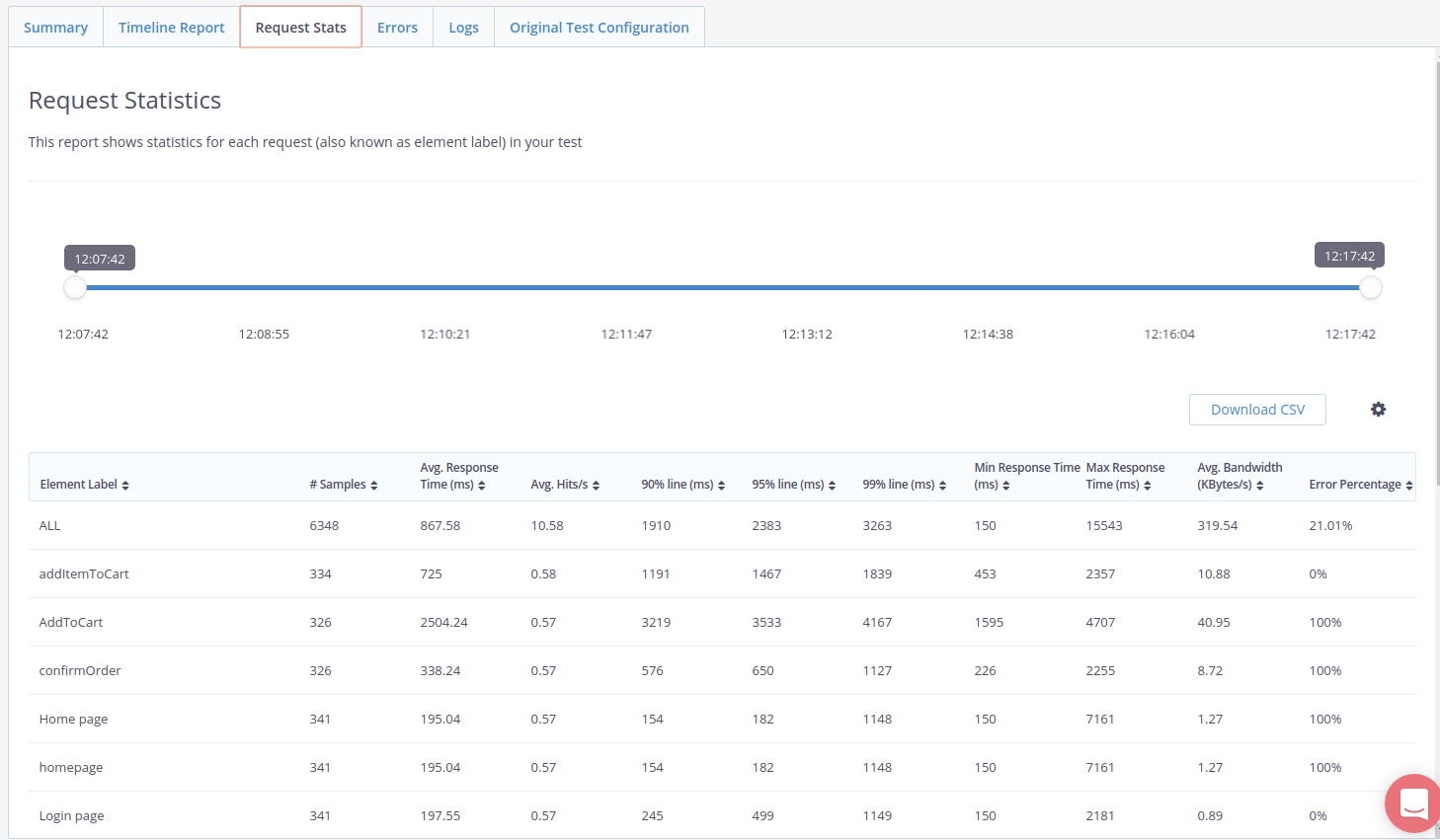
The summary is static: metrics cannot be added or removed.

TimeLine Report

*TimeLine Report*

The timeline report provides a huge graph whose curves can be customised. Transactions can be individually selected and plotted. It’s a little bit sad that the samplers hierarchy isn’t kept: all transactions and requests are within a single list. The timeline can get quite messy if many requests are drawn simultaneously.

Request Stats

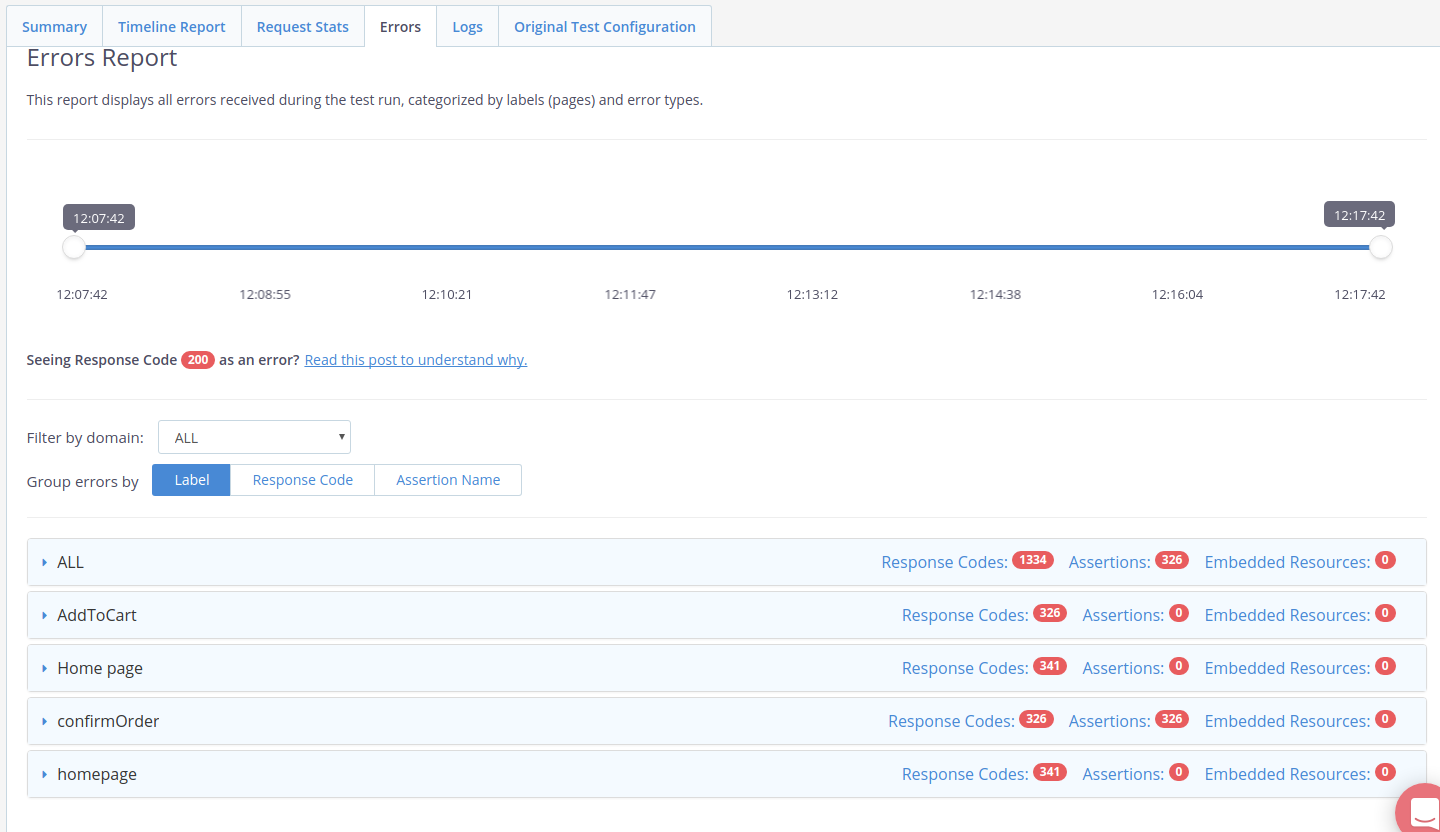
*Request Stats*

The request stats provides a table which contains the global statistics for each transaction or request. The following statistics are available:

* **# Samples**: number of samples,
* **Avg Response Time (ms)**: Average Elapsed Time in milliseconds,
* **90th line (ms)**: 90% centile on Elapsed Time in milliseconds,
* **95th line (ms)**: 95% centile on Elapsed Time in milliseconds,
* **99th line (ms)**: 99% centile on Elapsed Time in milliseconds,
* **Min Response Time (ms)**: Minimum Elapsed Time in milliseconds,
* **Max Response Time (ms)**: Maximum Elapsed Time in milliseconds,
* **Average KiB/sec**: Network Throughput (download) in KiloBytes per second,
* **Error Percentage**: Percent of hits in error.

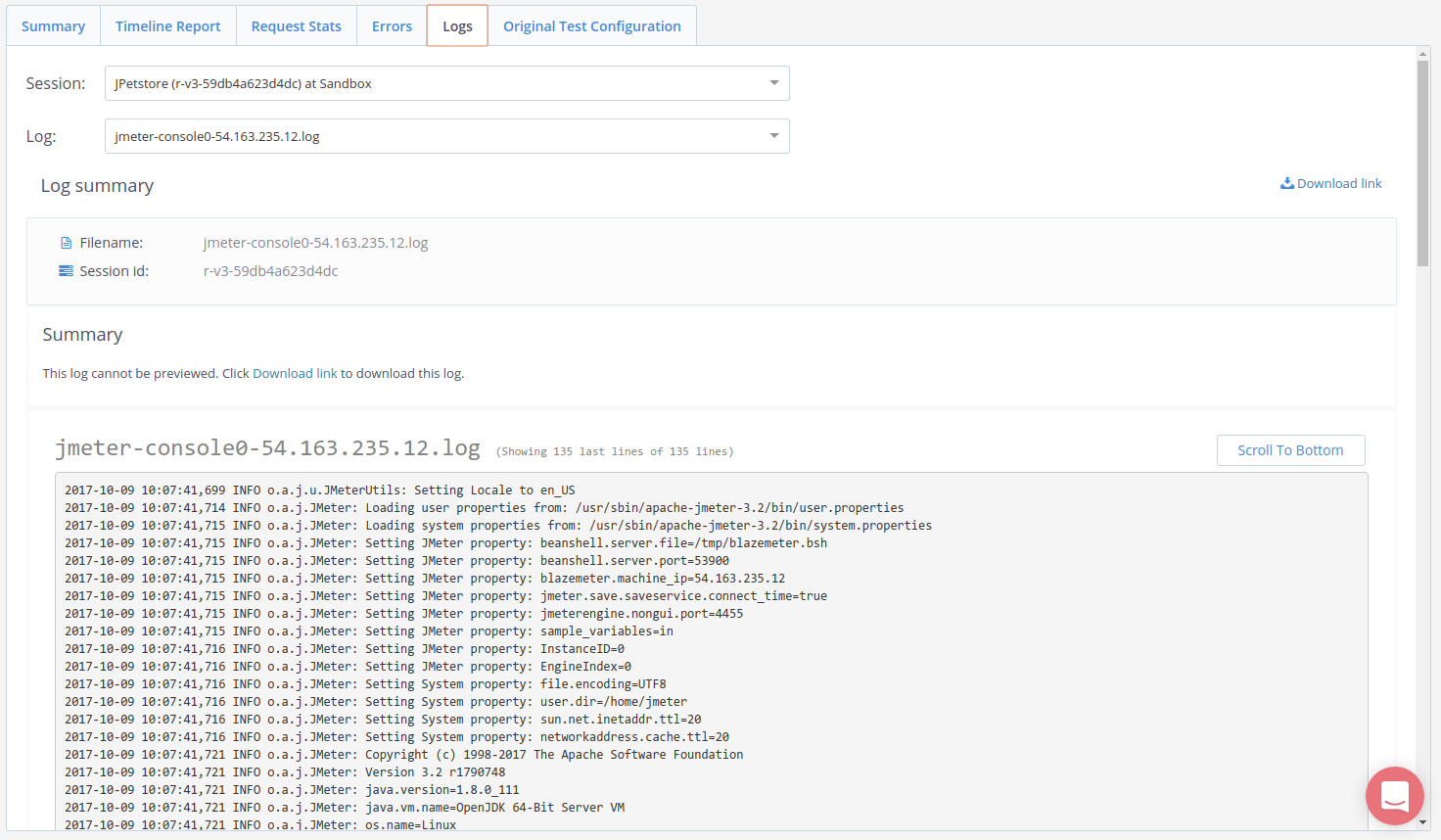
The whole table can be downloaded as a CSV file for external processing. Statistics can be filtered by time.

Errors

*Errors Reporting*

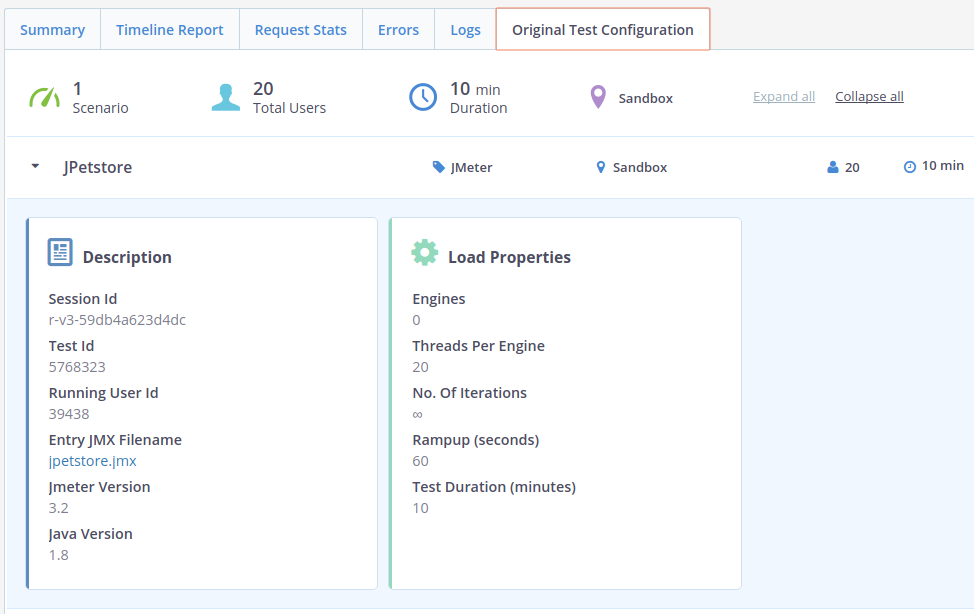
This report displays all errors received during the test run, categorized by labels (pages) and error types.

JMeter Logs

*JMeter Logs*

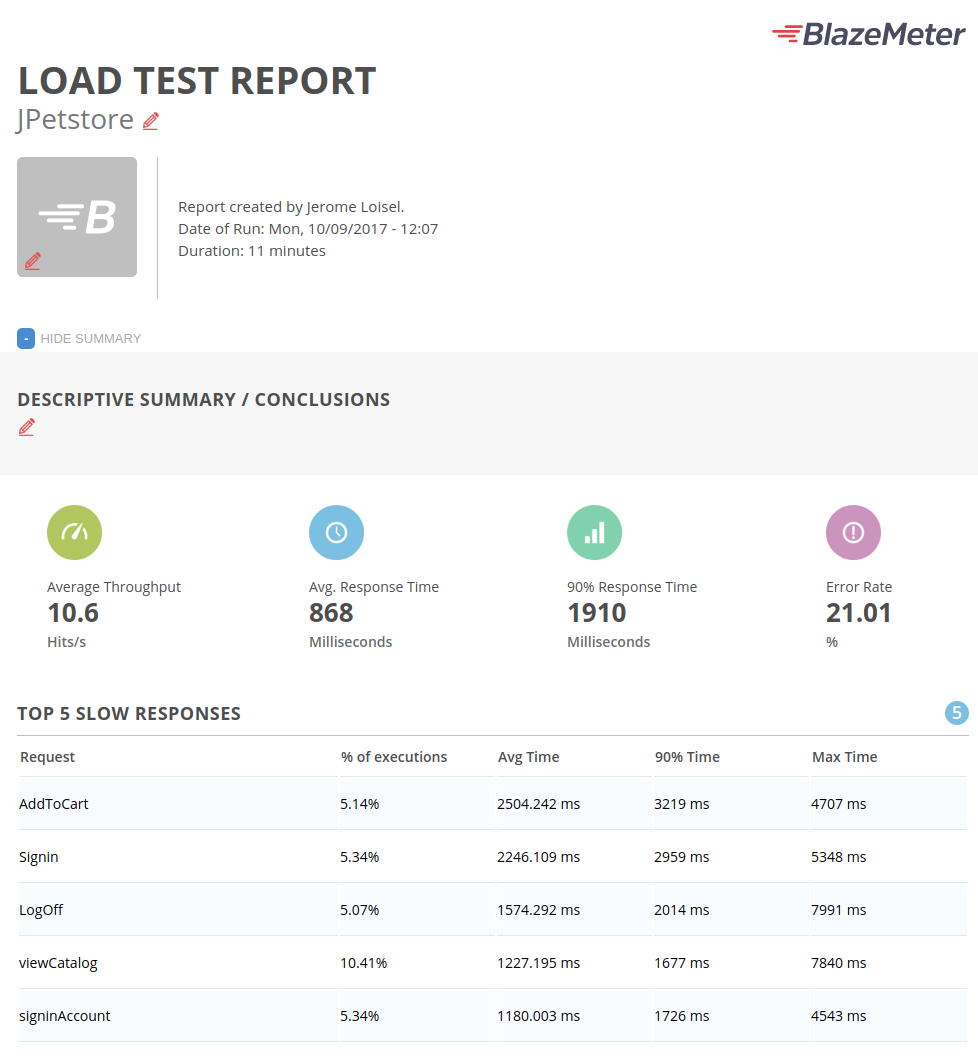
JMeter logs per engine are available. Logs can be downloaded or viewed within the browser directly.

Original Test Config

*Original Test Configuration*

This section is a reminder of the original test configuration.

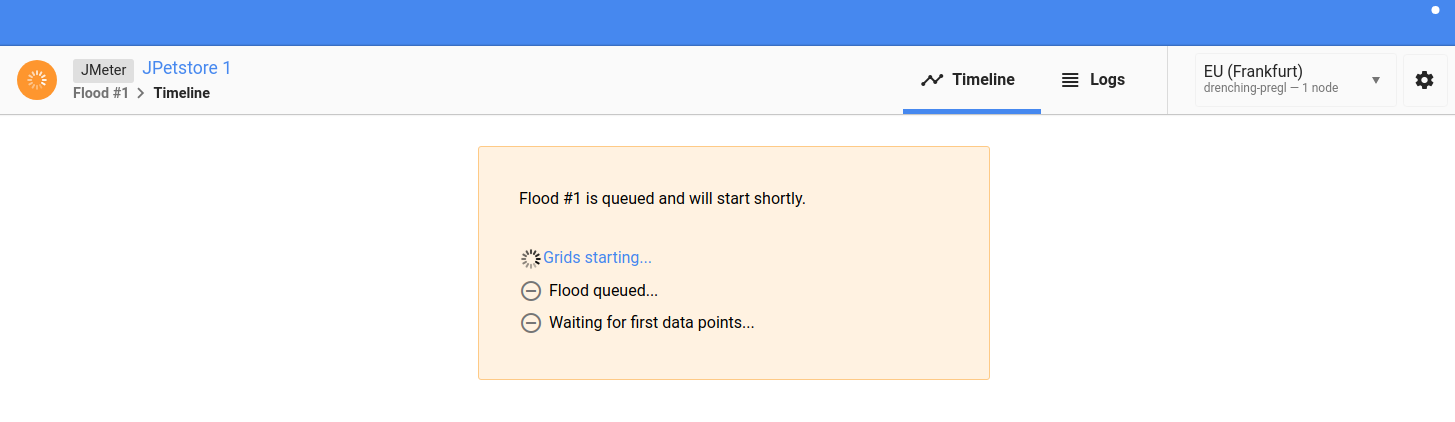
Executive Summary

*Executive summary*

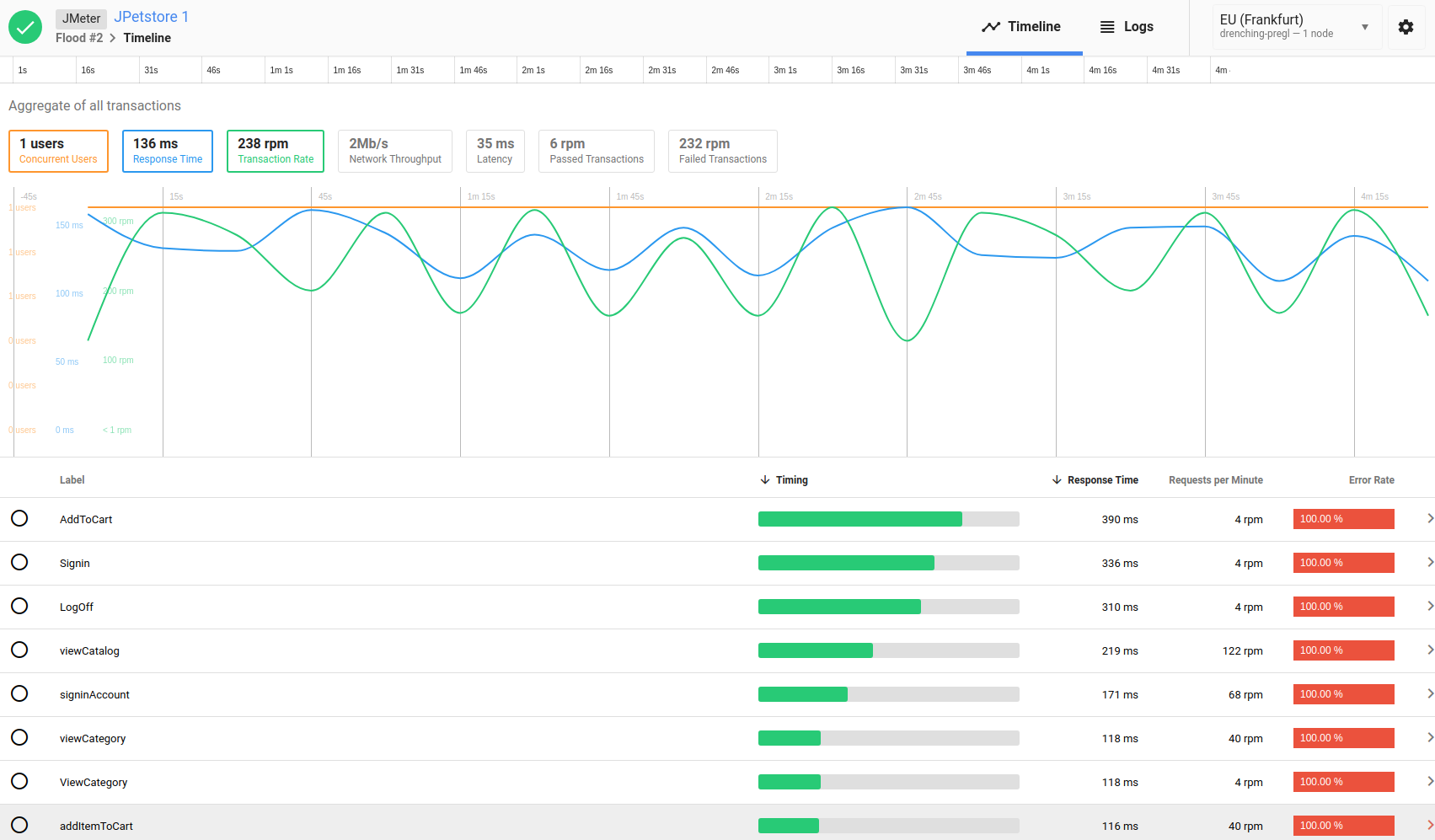
The executive summary is a **printable version of the test report**. It contains everything from the previous sections (Summary, TimeLine and more).

Flood

Flood is a Blazemeter challenger. This Australian company has been founded in September 2013 by [**Ivan Vanderbyl**](https://www.crunchbase.com/person/ivan-vanderbyl) and [**Tim Koopsman**](https://www.crunchbase.com/person/tim-koopmans). They offer pretty much the same features as BlazeMeter: upload your JMX script, run the test and analyze results.

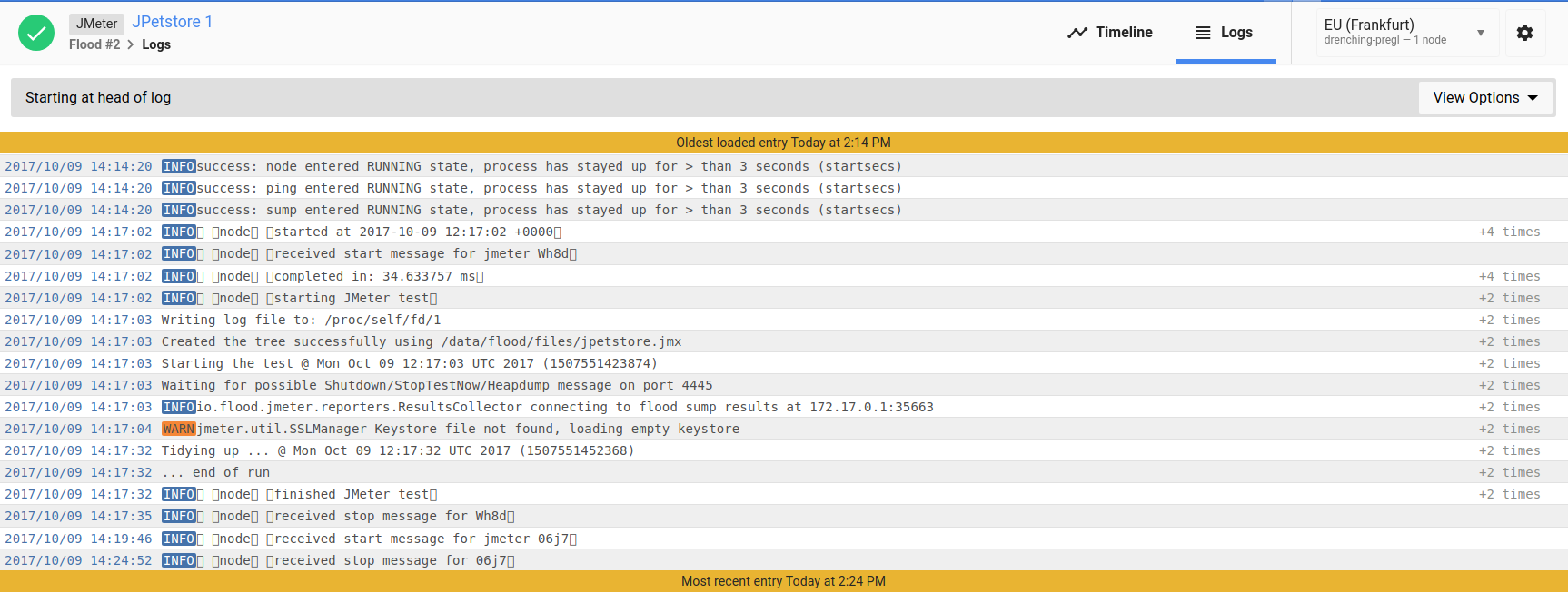
*Starting Test on Flood IO*

TimeLine

*TimeLine includes a main graph with selectable transactions*

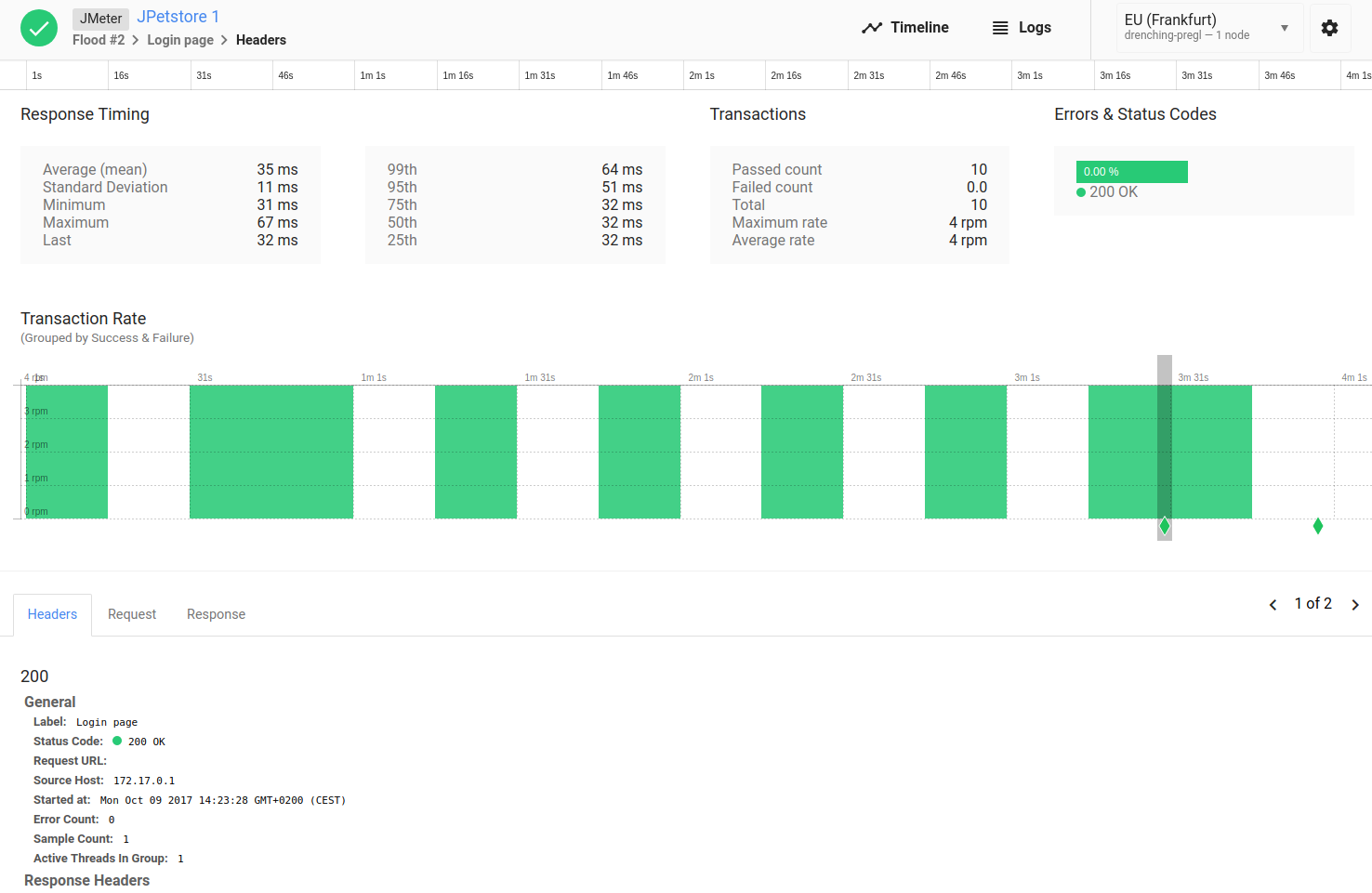
The TimeLine gives an overview of the test result metrics. you can draw a single transaction metrics by selecting it in the table below.

JMeter Logs

*JMeter Logs Live Tail and Download*

JMeter logs can be viewed live while the test is running. The logs can be downloaded at the end of the test.

Request Details

*Transaction / Request details*

By selecting a single request or transaction, you gain access to a sub-report which gives numerous metrics about that transaction. (Average, Minimum, Maximum, Standard Deviation, Centiles, passed vs failed and more) A few requests and responses are also stored at some random points during the load test.

Miscellaneous

The metrics can be downloaded as a CSV file for external processing.

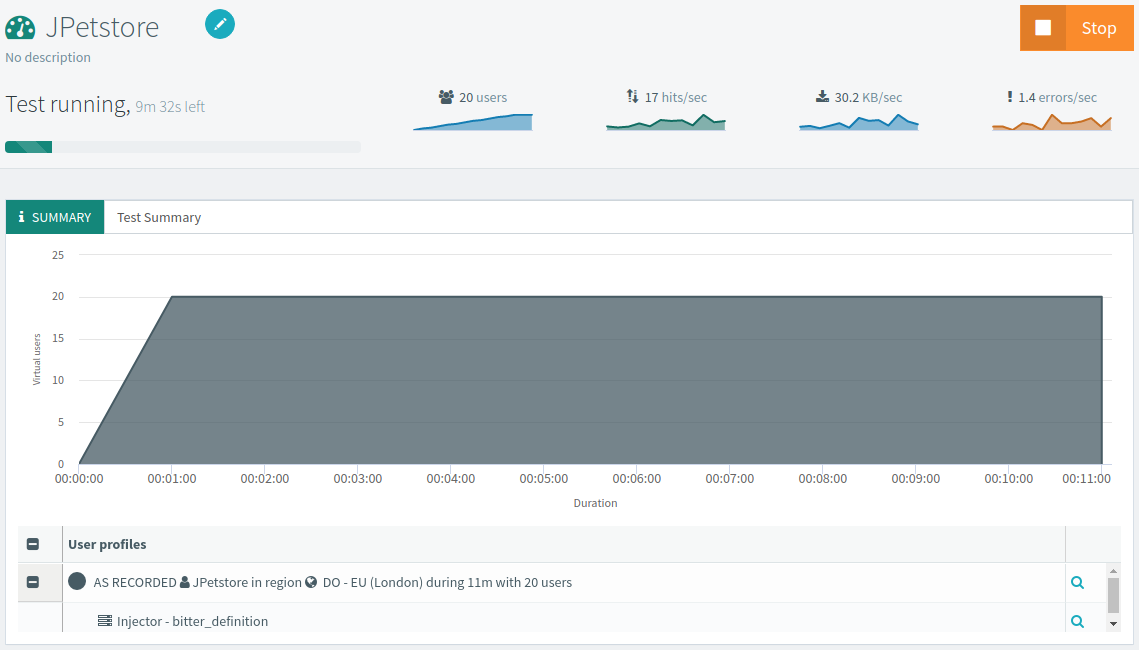
OctoPerf

[**OctoPerf**](https://octoperf.com/) is a French load testing company founded in September 2016. OctoPerf’s reporting system is a modular system designed which can be customized. Any of the report items below can be rearranged, thus making the reporting system dynamic. The report is pre-configured with certain report items. Items can be added or removed as needed.

*Starting a Test on OctoPerf*

For more information, please read the documentation on [**Report Items**](https://doc.octoperf.com/analysis/edit-bench-report/).

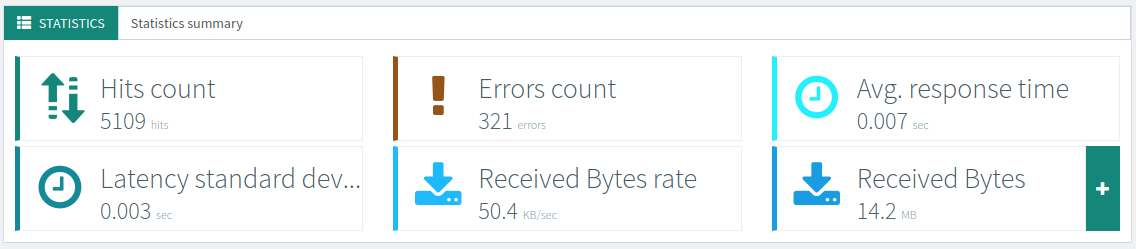
Test Summary

*Test Summary*

The test summary displays details about the test configuration like:

* **Test Duration**,
* **Number of concurrent user**,
* **Geographical location used**,
* and more.

Statistics Summary

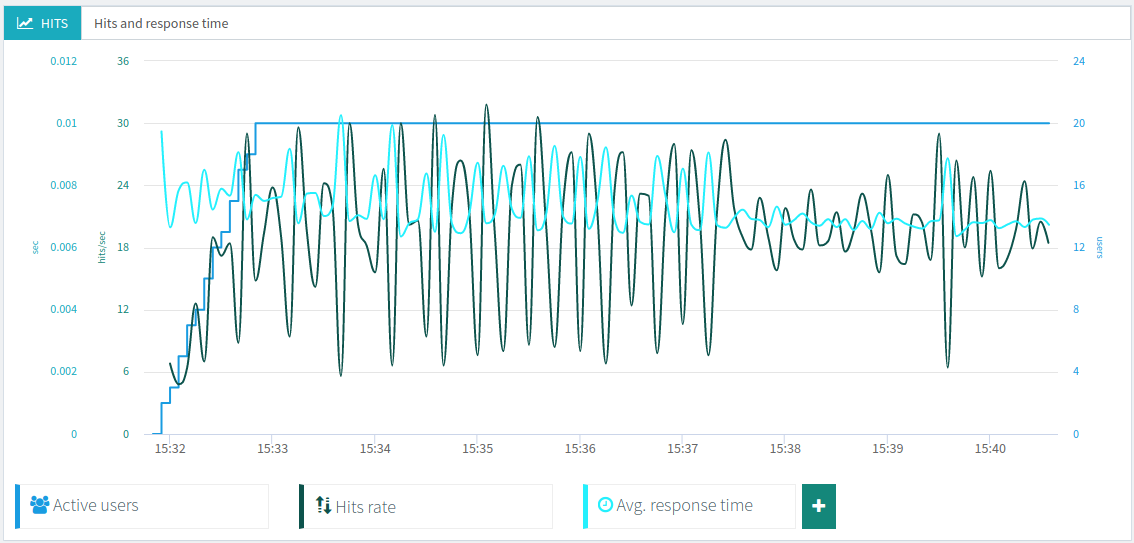
*Statistics Summary*

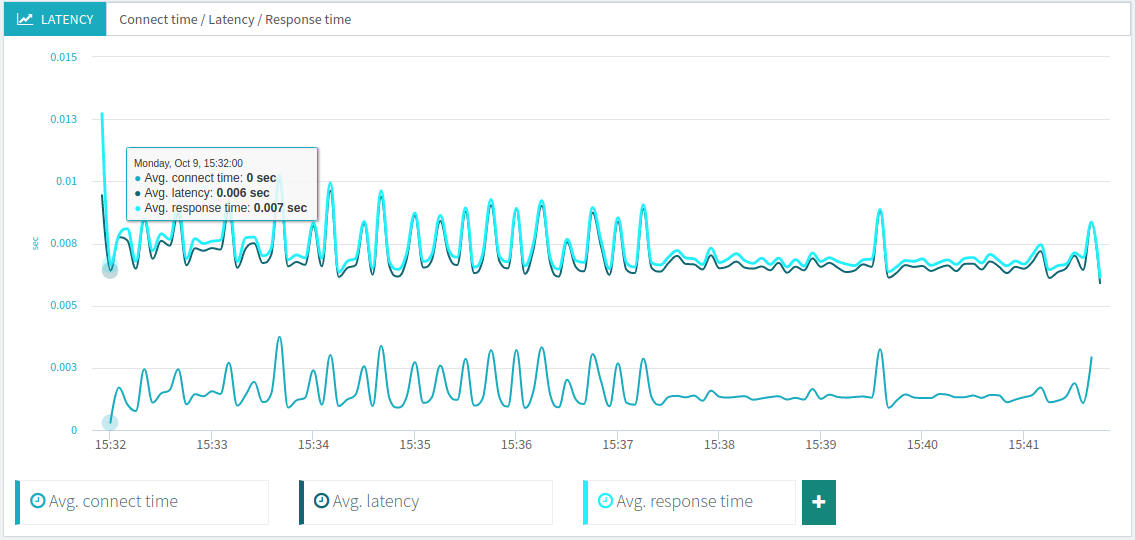
Statistics summary provides test wide statistics. The following settings can be customized:

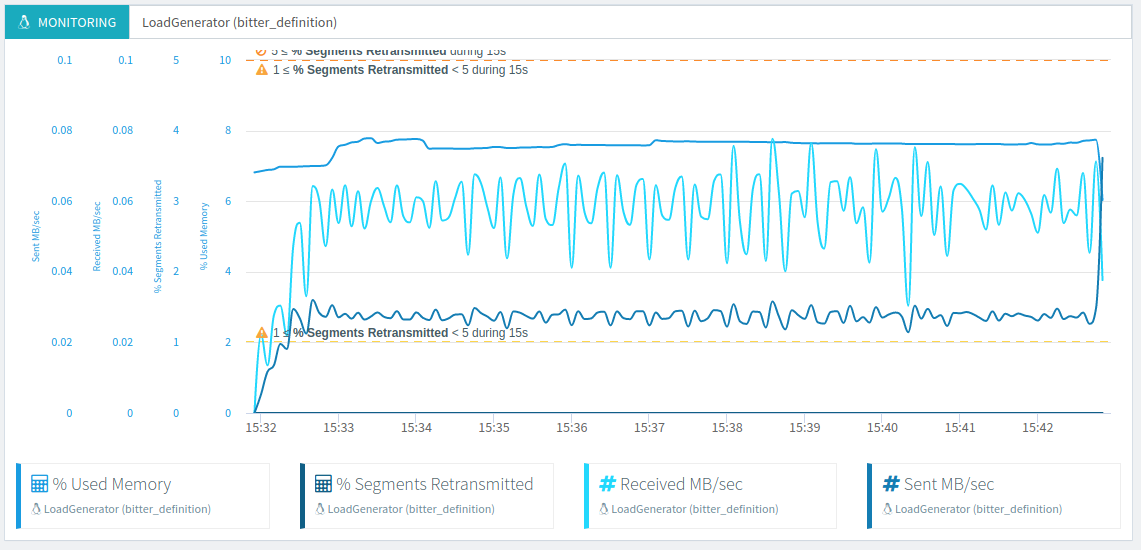
* **The number of statistics being displayed**,
* **The kind of statistics to include**.

There are 30+ metrics available.

Graphs

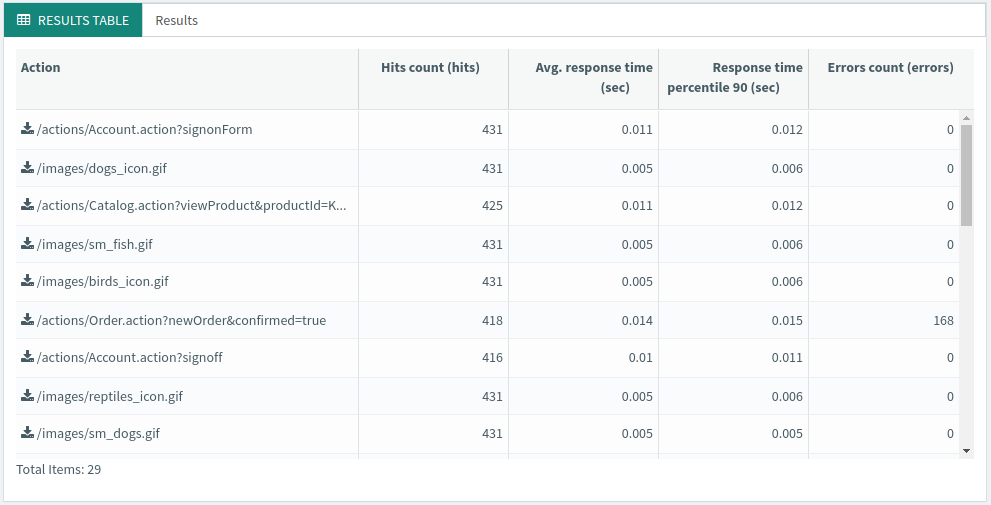






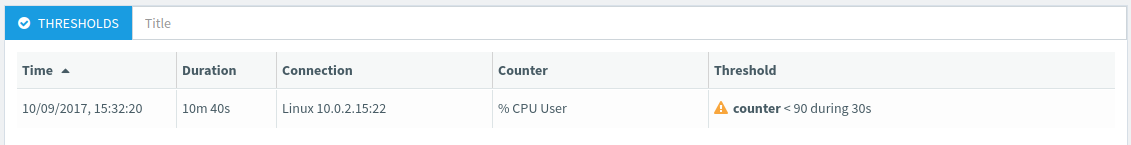
OctoPerf reporting system can feature an unlimited number of graphs, each configured differently. Each graph has customizable curves, from 1 to 4 curves per graph. **You can graph both performance metrics and monitoring metrics**, even on the same graph.

Results Table



The results table provides global statistics per transaction or request.

Thresholds



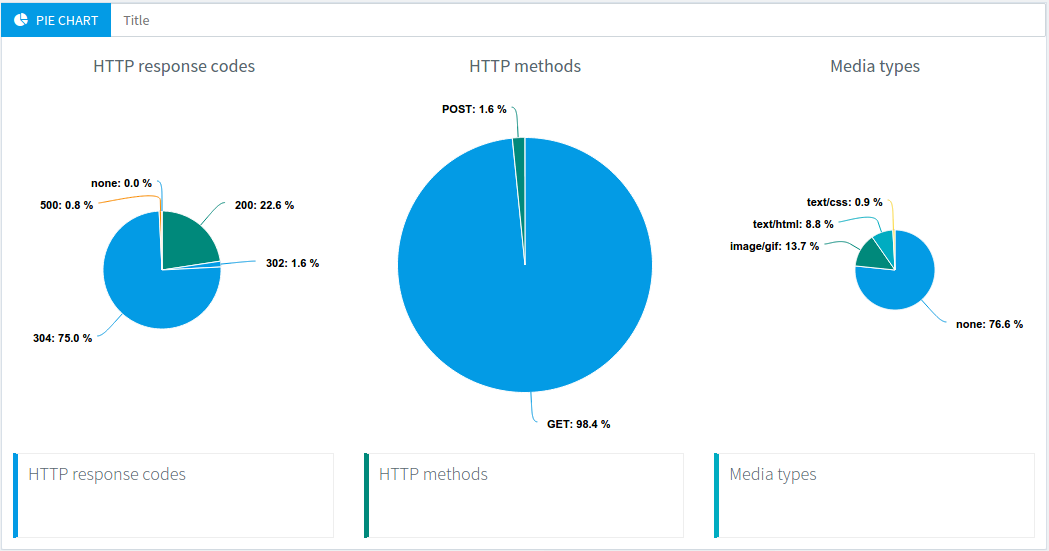
The thresholds table displays the threshold warnings and errors occured during the test. Threshold are associated to the monitoring feature. Monitoring allows you to capture backend server monitoring metrics.

Top Chart



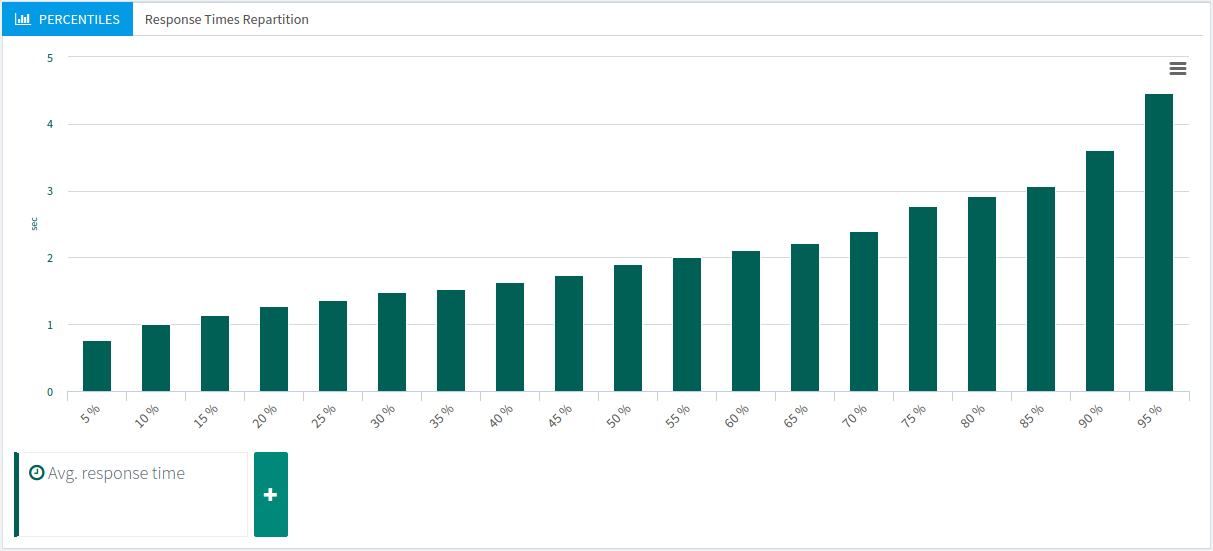
The top chart item provides a top of containers or http requests for a given metric. This chart is great for drill-down to find slow business transactions and/or requests.

Pie Chart



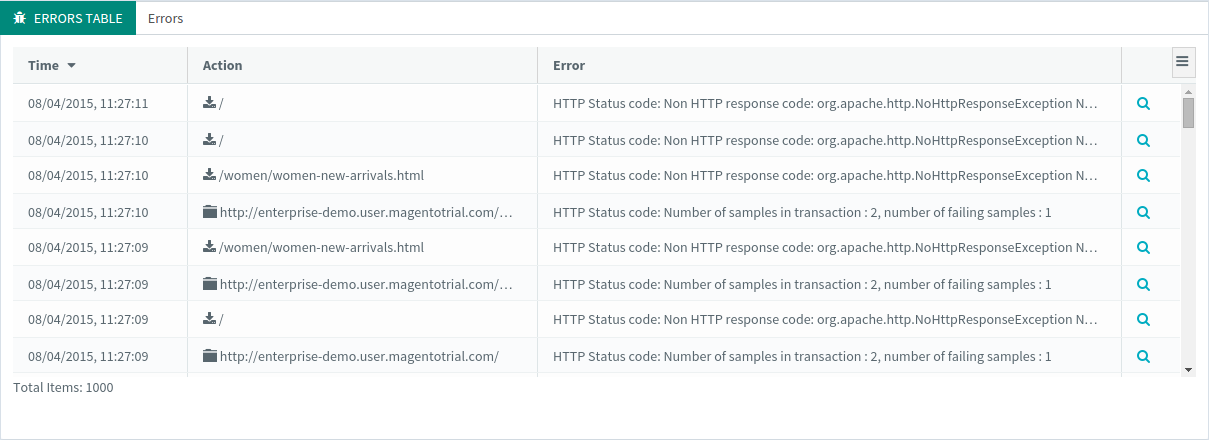
Pie charts are useful to get a quick overview of HTTP Response code, HTTP methods and HTTP response media types repartition. It allows to quickly spot if the web applications is running as expected.

Percentiles

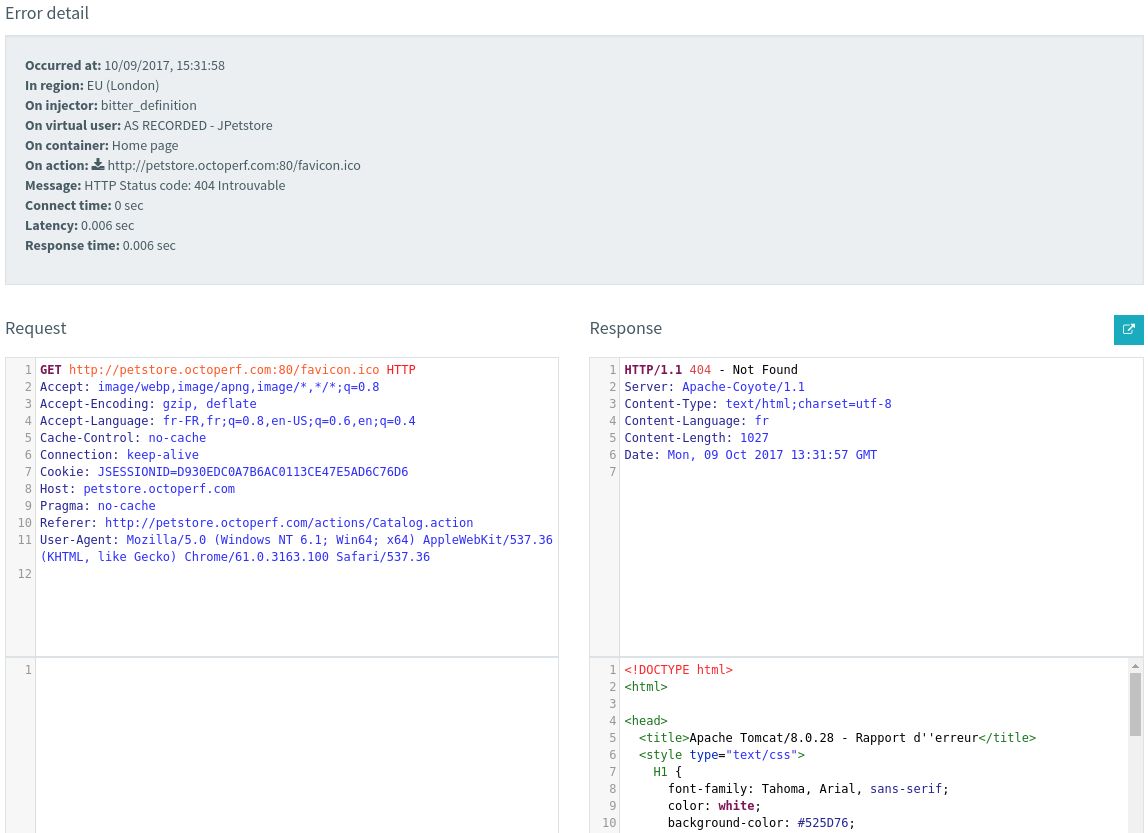


Percentiles charts shows the point at which a certain percentage of observed values occur. For example, the 95th percentile is the value which is greater than 95% of the observed values.

Errors Table

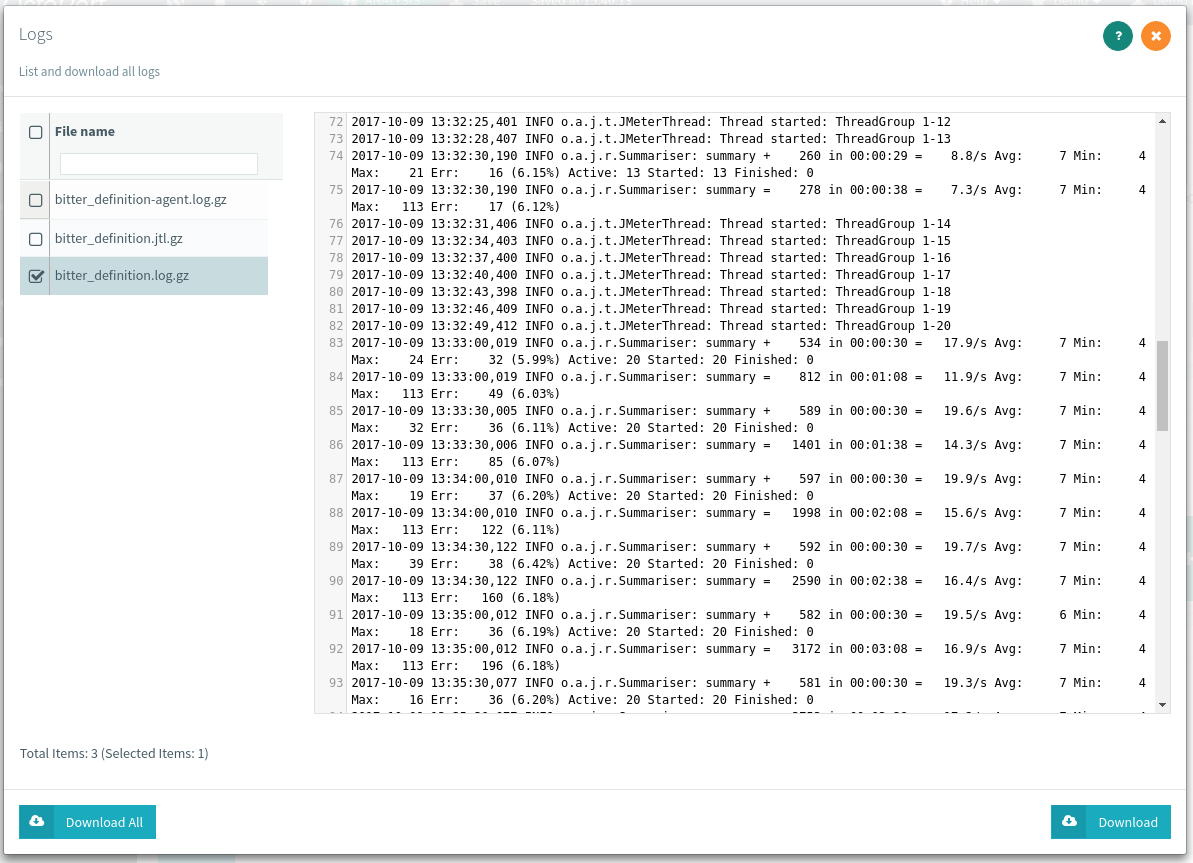


The errors table provide details about each error which occurred during the test. It allows to understand what happened server-side during the load test.

*Details per Error*

For each logged error, you can **view the request sent and response received** from the server.

JMeter JTL and Logs



OctoPerf lets you view JMeter logs after executing a Virtual User validation or a load test. You can also **download the full log file** by clicking on the Download button. A .log.gz is downloaded when you click on it. You need a file compression tool like 7Zip to extract it.

JMeter JTL files are also automatically centralised at the end of the test.

Comparison table

Guess what? We have compiled a **comparison table** which directly compares the **Top 3 JMeter Cloud Solutions** of the market:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **OctoPerf** | **Blazemeter** | **Flood.io** |
| **Recorder** |  |  |  |
| **HAR Import** |  |  |  |
| **Single URL/REST** |  |  |  |
| **Jmeter Import** |  |  |  |
| **Gatling import** |  |  |  |
| **Correlations** |  | In Jmeter | In Jmeter |
| **Variables** |  | In Jmeter | In Jmeter |
| **Validate script** |  | In Jmeter | In Jmeter |
| **Host file override** |  |  |  |
| **SLA** |  |  |  |
| **Sandbox (free unit test)** | 100 tests per month | 10 tests only | 5 hours only |
| **Bandwidth emulation** |  | Global only |  |
| **Latency emulation** |  | Global only |  |
| **Think time** |  | Global only | In Jmeter |
| **Pacing** |  |  |  |
| **Ramp down** |  |  |  |
| **Hits & RPS load policy** |  |  | In Jmeter |
| **Real browser monitoring** |  |  |  |
| **LG startup & config** | Automatic | Manual | Manual |
| **LG monitoring** |  |  |  |
| **Several JMX in one test** |  |  |  |
| **Pre-test checks** |  |  |  |
| **Live filters** |  |  |  |
| **Duration filter** |  |  |  |
| **Automatic SLAs** | APDEX |  | APDEX |
| **Reserve IP** |  |  |  |
| **Default views** | Good | Good | Average |
| **Overall usability** | Good | Average | Average |
| **Collaborative access** |  |  |  |
| **Logs** |  |  |  |
| **Error details** | with details |  |  |
| **Editable graphs** |  | one graph only |  |
| **Export PDF** |  |  |  |
| **Export CSV** | through JTL |  |  |
| **Customize report text** |  |  |  |
| **Comparison** |  |  |  |
| **Trending** |  |  |  |
| **Report public link** |  |  |  |
| **Report availability** | Unlimited | 1 week to unlimited | 1 to 12 months |
| **Jmeter version** | Latest Jmeter version supported | Several Jmeter versions supported | One version of Jmeter only, currently not the latest (3.1 instead of 3.3) |

Feel free to ask for other features to be checked from the comments.

Summary

There are many different ways to collect and display JMeter performance metrics. From DIY open-source tools to proprietary solutions, there is a solution for everyone. Which solution should you use? Here is a brief summary.

* **UI Listeners**: great for debugging purpose, you can use them for very small load tests (under 50 concurrent users),
* **JTL Files + Simple Data Writer**: this solution can be used for distributed tests, although the configuration can be tedious. JTL files can then be analyzed using JMeter XSL sheets or via the HTML report,
* **Backend Listener + InfluxDB + Grafana**: This solution eliminates the tedious work of gathering and merging JTL files in distributed testing. It also provides live metrics during the test. But the setup is difficult, requires advanced knowledge and multiple systems must be maintained,
* **Saas Solutions**: Easiest and most powerful solution, but you have to pay for tests larger than 50 concurrent users on most platforms. You can potentially save a huge amount of time on test setup and results analysis.

The chosen solution highly depends on the following factors:

* **Time**: is the load testing phase tight on time?
* **Budget**: is there a budget being allocated to cover the load testing expenses? How much is the budget?
* **Expertise**: How much expertise in the load testing field do you have?

Open-source and DIY solutions are usually free but cost a lot of time. Proprietary solutions have a cost but are way more time effective. **Whether you have time, budget or even both, there is a solution for everyone**