Running and analyzing stress tests for Oracle APEX apps using Apache JMeter

Author: Niels de Bruijn & Susann Hennemeyer, Date: 19-AUG-2021, Version: 1.1

Introduction

With Apache JMeter, you can simulate users for a web application and see if the server environment is able to handle the load. In this document, we are going to use a configuration with which we can do stress testing of applications built by Oracle Application Express (Oracle APEX).

A special thanks goes out to Kris Rice from Oracle for his valuable input that made this document possible.

Updating the default configuration

The referenced test in this document is based on the file “apexlocal.jmx“. You can download the file at https://apex.mt-ag.com/i/mt_additional_libraries/downloads/apexlocal.jmx.zip.

Open the file after starting JMeter and have a close look at each of the attributes in the of the HTTP requests within the section “Recording Controller”. tree nodes to ensure that the configuration matches your own APEX environment. For each of them, change the server protocol/name/port/path accordingly.

When it comes to the parameters section, let us assume your app id is 108 and the logon page id is 9999. The following changes are then necessary:

With HTTP request "get logon page":
- At "p", the app id and the page id have to be replaced. In this case the app id is 108 and the page id for the logon page 9999.

With HTTP request "post logon credentials to get cookie" change the attribute:
- p_flow_id to app id 108
- p_flow_step_id to page id 9999
- p_request to match the item name for username on the APEX login page
- p_json -> this JSON payload contains the logon page item names including its values. Some necessary references here are automatically set by a script that is already part of the configuration.

With HTTP request "get home page":

- At "p", the app id and the page id have to be replaced. In our example, the app id is 108 and the page id for the logon page is 1.

In the second and third HTTP request you should also set the attribute "referer" in the HTTP Header Manager accordingly.

If your APEX page requires no authentication, the first two HTTP GET requests (“*get logon page*” and “*post logon credentials to get cookie*”) can be deleted.

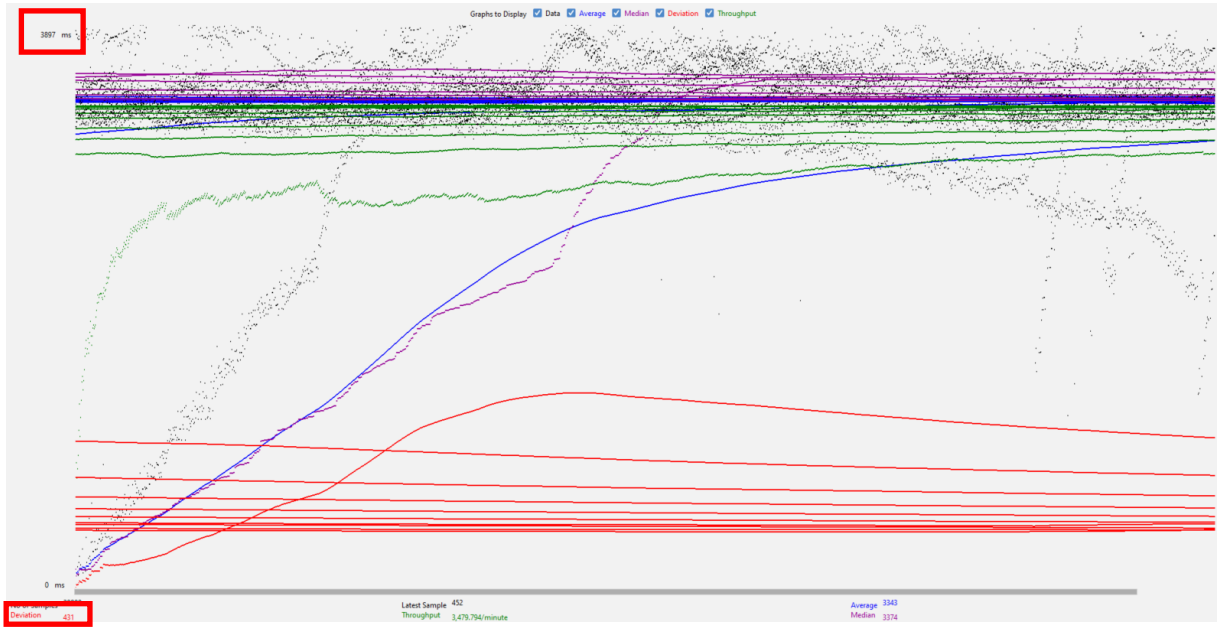
Running and analyzing tests

For the first test we will use the following parameters for the Thread Group:

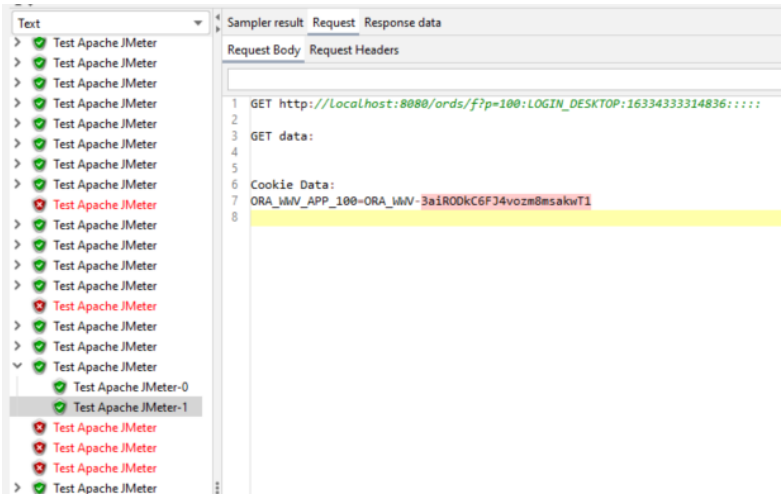
Number of threads (users): 100
Ramp-up period (seconds): 10
Loop count: 100

During the test you can have a look at the "*Graph Results*" graph. Here you can see in real time how much time is needed per request as well as the average over all requests.

Here is an outcome of a simple test that was executed on a laptop to test the local APEX installation:

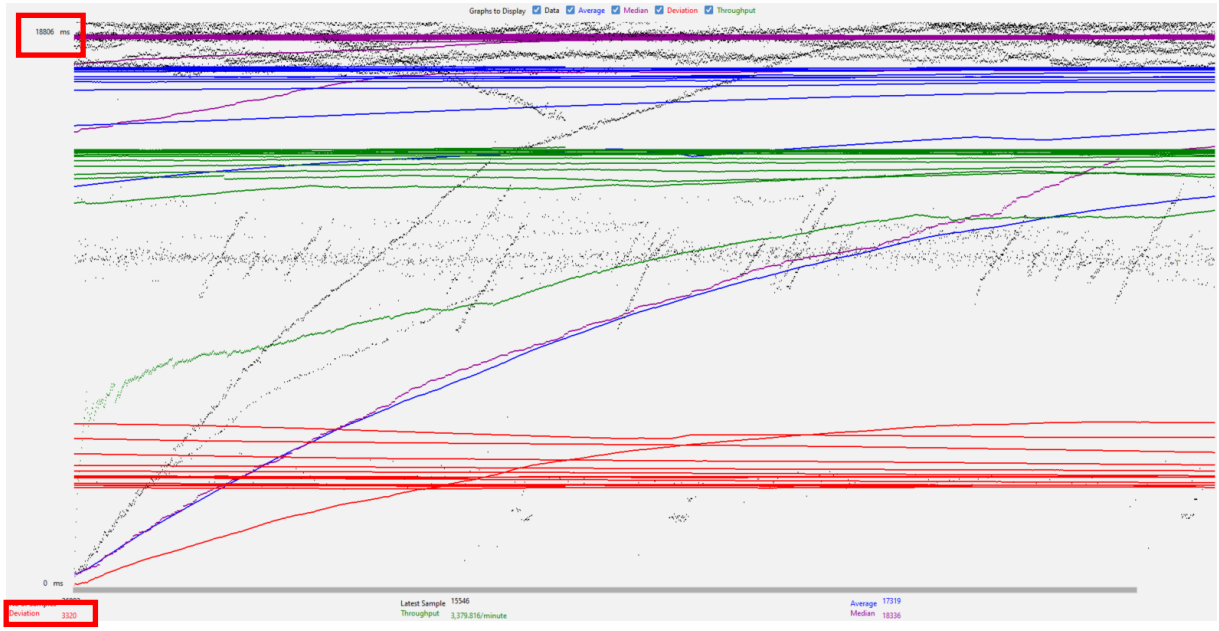


The blue lines are showing the average of the response time of the requests, the purple ones showing the median value, the red ones the deviation and the green ones the throughput. As the test got executed, the lines start to go up, which is normal. The plainer the lines are, the better is the stability of the environment for which the test was executed. In the left down corner, you can see the deviation. The lower this number is, the better. On the left top corner, you see the response time. In this case, each request got a response within 4 seconds. Of course, this doesn't tell you if the response was good or bad. If you look at the report "*View Results Tree*", you can see which HTTP requests got a normal response and which ones got an HTTP error back from server:



Normally, you would want every request listed in green, but with 100 parallel users on a simple laptop, what can you expect? ☺

Now let us see what happens if we run the same test, but now with 1.000 threads running in parallel after the ramp up time:



The major difference now is that at the number of 1.000 threads the average, the median time (purple) and average time (blue) are at around 18 seconds, whereas the first graph had a response time of around 4 seconds. Obviously, our laptop is not able to handle all these requests in an acceptable time. Even worse, most requests in the “View Results Tree” report are now failing as there are no free connections in the ORDS pool available anymore, meaning that a lot of users would see an HTTP error on their screen in a real-world scenario. Of course, you could increase the maximum number of connections in ORDS, but this only makes sense if the environment is capable of handling this.

Final thought

One important thing to remember when sizing an environment for a web application is that only a small percentage of the total active users are requiring system resources. Let me give you an example: a low-budget server these days would be able to serve 50 requests per second. When each user takes on average 20 seconds to ie. request the next web page or to submit a form, this means that up to 1.000 users can work in parallel with the web application.