Apache Bench

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Apache Bench (ab) is a load testing and benchmarking tool for Hypertext Transfer Protocol (HTTP) server. It can be run from command line and it is very simple to use. A quick load testing output can be obtained in just one minute. As it does not need too much familiarity with load and performance testing concepts, therefore it is suitable for beginners and intermediate users.

To use this tool, no complex setup is required. Moreover, it gets installed automatically with Apache web server, or it can be installed separately as Apache utility. It does not have all the features of more popular tools such as jMeter or Grinder, but it is good for a start.

Audience

This tutorial is designed for Application Developers and System Administrators, who are willing to learn Apache Bench in simple and easy steps. This tutorial will give you practical knowledge on Apache Bench, and after completing this tutorial, you will be at an intermediate level of expertise from where you can take yourself to higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of command line interface (CLI), HTTP, text editor and web servers, etc., because you will need these tools to successfully run Apache Bench for load testing. In addition, it will be good if you have knowledge of web development and application testing processes.

Copyright & Disclaimer

Table of Contents

1. Apache Bench – Overview

Performance testing has proved itself to be crucial for the success of a business. Not only does a poor performing site face financial losses, it can also lead to legal repercussions at times.

No one wants to put up with a slow performing, unreliable site in important online interactions such as purchasing, online test taking, bill payment, etc. With the Internet being so widely available, the range of alternatives is immense. It is easier to lose clientele than gain them and performance is a key game changer.

Need for a Load Testing Tool

If we can understand what is the need for a load testing tool, it will give us the reason and motivation to use it. Some famous business sites have suffered serious downtimes when they get large number of visitors. E-commerce websites invest heavily in advertising campaigns, but not in Load Testing. Therefore, they fail to ensure optimal system performance, when that marketing brings in traffic.

Another familiar example of ignoring load testing is that of "error establishing connection" in WordPress websites. Therefore, it is a good idea to load test a website or application before its deployment in production. It is nice to quickly establish a best-case scenario for a project before running more detailed tests down the road.

What is Apache Bench?

Apache Bench (ab) is a tool from the Apache organization for benchmarking a Hypertext Transfer Protocol (HTTP) web server. Although it is designed to measure the performance of Apache web server, yet it can also be used to test any other web server that is equally good. With this tool, you can quickly know how many requests per second your web server is capable of serving.

Features of Apache Bench

Let us see the important features and limitations of Apache Bench. The features and limitations are listed below:

- Being an open source software, it is freely available.
- It is a simple command line computer program.
- It is a platform-independent tool. It means that it can be invoked on Linux/Unix or on Windows server equally well.
- It can conduct load and performance test for only web server - HTTP or HTTPS.
- It is not extensible.

Apache Bench uses only one operating system thread irrespective of the concurrency level (specified by the -c flag). Therefore, when benchmarking high-capacity servers, a single instance of Apache Bench can itself be a bottleneck. To completely saturate the target URL, it is better to use additional instances of Apache Bench in parallel, if your server has multiple processor cores.

Precaution

You need to be aware that there is no directive in the Apache Bench to increase concurrency in particular intervals while running tests. Therefore, running load tests using ab is equivalent to a denial-of-service (DOS) attack. It is recommended that you inform and take prior permission from your VPS service provider if you are going to do heavy load testing for a long period of time. They will allot you an appropriate time interval or shift your node for the load testing task.

Second, if you are testing a third person's website continuously and for a long time just for learning Apache Bench from your VPS (which becomes the testing node), there is a remote possibility that your VPS public IP can be blocked by the third person's website permanently. In that case, you will not be able to connect to that website with the same IP. But if you really want to connect to the website in future, the only solution will be to talk to the system administrator of the target website, or create a new instance of the server with a different IP with the help of your VPS service provider.

Having warned you, let me assure you that all tests in this tutorial are safe enough and out of what system administrators generally call "system abuse" practices.

2. Apache Bench – Environment Setup

In this chapter, we will guide you how to set up your environment for Apache Bench on your VPS.

System Requirement

- **Memory**: 128 MB
- **Disk Space**: No minimum requirement
- **Operating System**: No minimum requirement

Installing Apache Bench

Apache Bench is a stand-alone application, and has no dependencies on the Apache web server installation. The following is a two-step process to install Apache Bench.

Step 1: Update package database.

```
# apt-get update
```

Please note that symbol # before a terminal command means that root user is issuing that command.

Step 2: Install apache2 utils package to get access to Apache Bench.

```
# apt-get install apache2-utils
```

Apache Bench is now installed. If you want to test a web application hosted on the same VPS, then it is enough to install the Apache web server only:

```
# apt-get install apache2
```

Being an Apache utility, Apache Bench is automatically installed on installation of the Apache web server.

Verifying Apache Bench Installation

Let us now see how to verify Apache Bench Installation. The following code will help verify the installation:

```
# ab -V
```

Output

```
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

When you see the above terminal output, it means you have successfully installed Apache Bench.

Creating a Privileged Sudo User

From the safety point of view, it is considered a good practice for system administrator to create a sudo user instead of working as root. We will create a test user, named test, for the purpose:

```
# useradd -m -d /home/test -g sudo test
```

Let us set the password for the new user:

```
# passwd test
```

System will prompt for a new password for the user test. You can enter a simple password as we are just testing, and not deploying to the production server. Usually the sudo command will prompt you to provide the sudo user password; it is recommended not to use complicated password as the process becomes cumbersome.

Output

```
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Testing Apache.org Website

In this section, we will test the Apache.org Website. Let us first switch to the sudo user test:

```
# su test
```

To begin with, we will test the website of Apache organization, https://www.apache.org/. We will first run the command, and then understand the output:

```
$ ab -n 100 -c 10 https://www.apache.org/
```

Here **-n** is the number of requests to perform for the benchmarking session. The default is to just perform a single request which usually leads to non-representative benchmarking results.

And **-c** is the concurrency and denotes the number of multiple requests to perform at a time. Default is one request at a time.

So in this test, Apache Bench will make 100 requests with concurrency 10 to the Apache organization server.

Output

```
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.apache.org (be patient).....done

Server Software:        Apache/2.4.7
Server Hostname:        www.apache.org
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,2048,256

Document Path:          /
Document Length:        58769 bytes

Concurrency Level:      10
Time taken for tests:   1.004 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      5911100 bytes
HTML transferred:       5876900 bytes
Requests per second:    99.56 [#/sec] (mean)
Time per request:       100.444 [ms] (mean)
Time per request:       10.044 [ms] (mean, across all concurrent requests)
Transfer rate:          5747.06 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       39   46  30.9     41     263
Processing:    37   40  21.7     38     255
Waiting:       12   15  21.7     13     230
```

```
Total:          77   86  37.5     79     301

Percentage of the requests served within a certain time (ms)
  50%     79
  66%     79
  75%     80
  80%     80
  90%     82
  95%     84
  98%    296
  99%    301
 100%    301 (longest request)
```

Having run our first test, it will be easy to recognize the pattern of use for this command which is as follows:

```
# ab [options .....]  URL
```

where,

- **ab:** Apache Bench command
- **options:** flags for particular task we want to perform
- **URL:** path url we want to test

Understanding the Output Values

We need to understand the different metrics to understand the various output values returned by ab. Here goes the list:

- **Server Software** – It is the name of the web server returned in the HTTP header of the first successful return.

- **Server Hostname** – It is the DNS or IP address given on the command line.

- **Server Port** – It is the port to which ab is connecting. If no port is given on the command line, this will default to 80 for http and 443 for https.

- **SSL/TLS Protocol** – This is the protocol parameter negotiated between the client and server. This will only be printed if SSL is used.

- **Document Path** – This is the request URI parsed from the command line string.

- **Document Length** – It is the size in bytes of the first successfully returned document. If the document length changes during testing, the response is considered an error.

- **Concurrency Level** – This is the number of concurrent clients (equivalent to web browsers) used during the test.

- **Time Taken for Tests** – This is the time taken from the moment the first socket connection is created to the moment the last response is received.

- **Complete Requests** – The number of successful responses received.

- **Failed Requests** – The number of requests that were considered a failure. If the number is greater than zero, another line will be printed showing the number of requests that failed due to connecting, reading, incorrect content length, or exceptions.

- **Total Transferred** – The total number of bytes received from the server. This number is essentially the number of bytes sent over the wire.

- **HTML Transferred** – The total number of document bytes received from the server. This number excludes bytes received in HTTP headers

- **Requests per second** – This is the number of requests per second. This value is the result of dividing the number of requests by the total time taken.

- **Time per request** – The average time spent per request. The first value is calculated with the formula concurrency * timetaken * 1000 / done while the second value is calculated with the formula timetaken * 1000 / done

- **Transfer rate** – The rate of transfer as calculated by the formula totalread / 1024 / timetaken.

Quick Analysis of the Load Testing Output

Having learned about the headings of the output values from the ab command, let us try to analyze and understand the output values for our initial test:

- Apache organisation is using their own web Server Software: Apache (version 2.4.7)

- Server is listening on Port 443 because of https. Had it been http, it would have been 80 (default).

- Total data transferred is 58769 bytes for 100 requests.

- Test completed in 1.004 seconds. There are no failed requests.

- Requests per seconds: 99.56. This is considered a pretty good number.

- Time per request: 100.444 ms (for 10 concurrent requests). So across all requests, it is 100.444 ms/10 = 10.044 ms.

- Transfer rate: 1338.39 [Kbytes/sec] received.

- In connection time statistics, you can observe that many requests had to wait for few seconds. This may be due to apache web server putting requests in wait queue.

In our first test, we had tested an application (i.e., www.apache.org) hosted on a different server. In the later part of the tutorial, we will be testing our sample web-applications hosted on the same server from which we will be running the ab tests. This is for the ease of learning and demonstration purpose. Ideally, the host node and testing node should be different for accurate measurement.

To better learn ab, you should compare and observe how the output values vary for different cases as we move forward in this tutorial.

Plotting the Output of Apache Bench

Here we will plot the relevant outcome to see how much time the server takes as the number of requests increases. For that, we will add the **-g** option in the previous command followed by the file name (here out.data) in which the ab output data will be saved:

```
$ ab -n 100 -c 10 -g out.data https://www.apache.org/
```

Let us now see the **out.data** before we create a plot:

```
$ less out.data
```

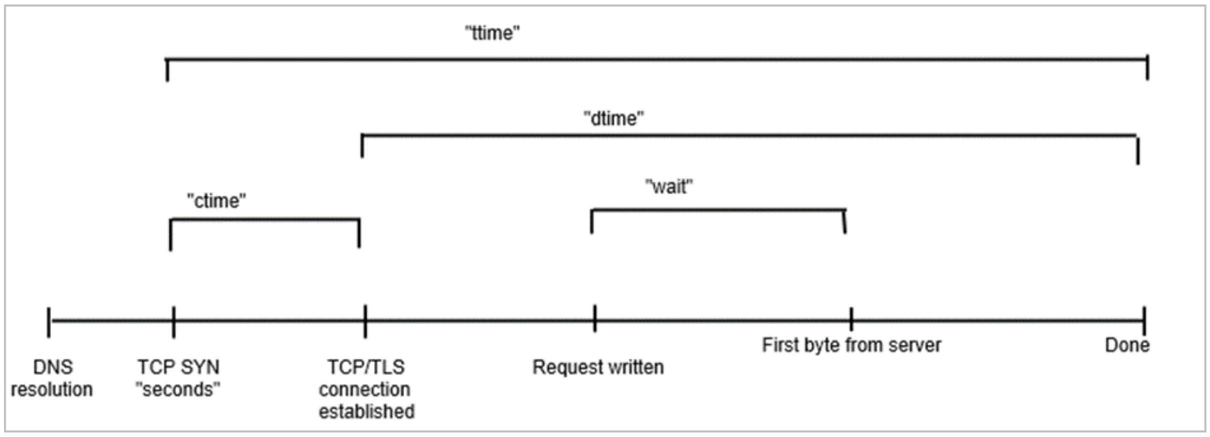
Output

```
starttime       seconds ctime   dtime   ttime   wait
Tue May 30 12:11:37 2017        1496160697      40      38      77      13
Tue May 30 12:11:37 2017        1496160697      42      38      79      13
Tue May 30 12:11:37 2017        1496160697      41      38      80      13
...
```

Let us now understand the column headers in the **out.data** file:

- **starttime:** This is the date and time at which the call started.
- **seconds:** Same as starttime but in the Unix timestamp format (date -d @1496160697 returns starttime output).
- **ctime:** This is the Connection Time.
- **dtime:** This is the Processing Time.
- **ttime:** This is the Total Time (it is the sum of ctime and dtime, mathematically ttime = ctime + dtime).
- **wait:** This is the Waiting Time.

For a pictorial visualization of how these multiple items are related to each other, take a look at the following image:



If we are working over terminal or where graphics are not available, **gnuplot** is a great option. We will quickly understand it by going through the following steps.

Let us install and launch gnuplot:

```
$ sudo apt-get install gnuplot
$ gnuplot
```

Output

```
        G N U P L O T
        Version 4.6 patchlevel 6    last modified September 2014
        Build System: Linux x86_64

        Copyright (C) 1986-1993, 1998, 2004, 2007-2014
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type set to 'qt'
gnuplot>
```

As we are working over terminal and supposing that graphics are not available, we can choose the dumb terminal which will give output in ASCII over the terminal itself. This helps us get an idea what our plot looks like with this quick tool. Let us now prepare the terminal for ASCII plot

```
gnuplot> set terminal dumb
```

Output

```
Terminal type set to 'dumb'
Options are 'feed  size 79, 24'
```

As, our gnuplot terminal is now ready for ASCII plot, let us plot the data from the **out.data** file:

```
gnuplot> plot "out.data" using 9  w l
```

Output

```
  1400 ++-----+------+-----+------+------+------+------+-----+------+-----++
       +      +      +     +      +      +      +"out.data" using 9 ****** +
       |                                                                   |
  1200 ++                      ********************************************
       |     *******************                                           |
  1000 ++    *                                                            ++
       |     *                                                             |
       |     *                                                             |
   800 ++   *                                                             ++
       |    *                                                              |
       |    *                                                              |
   600 ++   *                                                             ++
       |    *                                                              |
       |    *                                                              |
   400 ++   *                                                             ++
       |    *                                                              |
   200 ++   *                                                             ++
       |    *                                                              |
       +****  +      +     +      +      +      +      +     +      +      +
     0 ++-----+------+-----+------+------+------+------+-----+------+-----++
       0      10     20    30     40     50     60     70    80     90    100
```

We have plotted the ttime, total time (in ms) from column 9, with respect to the number of requests. We can notice that for the initial ten requests, the total time was in the nearly 100 ms, for next 30 requests (from 10th to 40th), it increased to 1100 ms, and so on. Your plot must be different depending on your **out.data**.

3. Apache Bench – Testing Our Sample Application

In the previous chapter, we understood the basic use of the Apache Bench to test a third party website. In this section, we will use this tool to test a web application on our own server. To keep the tutorial self-contained to the extent possible, we have chosen to install a python application for the demonstration purpose; you can choose any other language like PHP or Ruby depending on your expertise level.

Installing Python

Generally, Python is installed by default on Linux servers.

Installing Bottle Framework and Creating a Simple Application

Bottle is a micro-framework written in python for creating web applications, and pip is a python package manager. Type the following command in terminal to install Bottle:

```
$ sudo apt-get install python-pip
$ sudo pip install bottle
```

Let us now create a small Bottle application. For that, create a directory and move inside it:

```
$ mkdir webapp
$ cd webapp
```

We will create a new python script, **app.py**, inside the webapp directory:

```
$ vim app.py
```

Now, write the following code in the app.py file:

```
from bottle import Bottle, run


app = Bottle()


@app.route('/')
@app.route('/hello')
def hello():
    return "Hello World!"
```

```
run(app, host='localhost', port=8080)
```

When you have added the above lines, save and close the file. Having saved the file, we can run the python script to launch the application:

```
$ python app.py
```

Output

```
Bottle v0.12.7 server starting up (using WSGIRefServer())...

Listening on http://localhost:8080/

Hit Ctrl-C to quit.
```

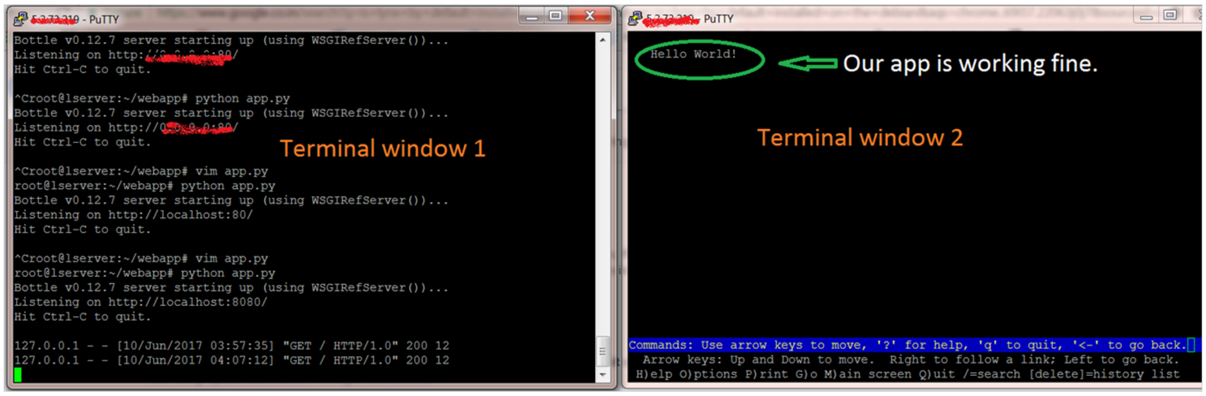
This output shows that our application is running on the local machine at the host **http://localhost** and listening on the port **8080**.

Let us check if our app is responding properly to the HTTP requests. As this terminal cannot take any input without quitting serving the Bottle application, we need to login to our VPS with another terminal. After logging into the VPS with another terminal, you can navigate to your application by typing the following code in the new terminal.

```
$ lynx http://localhost:8080/
```

Lynx is a command line browser and is usually installed by default in various Linux distributions like Debian and Ubuntu. If you see the following output, it means your app is working fine.

Output



```
Bottle v0.12.7 server starting up (using WSGIRefServer())...
Listening on http:[redacted]/
Hit Ctrl-C to quit.

^Croot@lserver:~/webapp# python app.py
Bottle v0.12.7 server starting up (using WSGIRefServer())...
Listening on http://0[redacted]/
Hit Ctrl-C to quit.

^Croot@lserver:~/webapp# vim app.py
root@lserver:~/webapp# python app.py
Bottle v0.12.7 server starting up (using WSGIRefServer())...
Listening on http://localhost:80/
Hit Ctrl-C to quit.

^Croot@lserver:~/webapp# vim app.py
root@lserver:~/webapp# python app.py
Bottle v0.12.7 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

127.0.0.1 - - [10/Jun/2017 03:57:35] "GET / HTTP/1.0" 200 12
127.0.0.1 - - [10/Jun/2017 04:07:12] "GET / HTTP/1.0" 200 12
```

```
Hello World!

Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

If you see the above output, that means our application is live and ready for testing.

Testing the Application with Developmental Web Server

Please note that there is a bug in ab, and it is not able to test the application on the localhost. So we will change the host from localhost to 127.0.0.1 in the app.py file. So the file will change to the following:

```
from bottle import Bottle, run

app = Bottle()

@app.route('/')
@app.route('/hello')
def hello():
    return "Hello World!"

run(app, host='127.0.0.1', port=8080)
```

Let us now test our app by typing the following command on the same terminal on which ran the lynx command:

```
$ ab -n 100 -c 10  http://127.0.0.1:8080/hello
```

Output

```
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient).....done


Server Software:        WSGIServer/0.1
Server Hostname:        127.0.0.1
Server Port:            8080

Document Path:          /hello
Document Length:        12 bytes
```

```
Concurrency Level:      10
Time taken for tests:   0.203 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      16500 bytes
HTML transferred:       1200 bytes
Requests per second:    493.78 [#/sec] (mean)
Time per request:       20.252 [ms] (mean)
Time per request:       2.025 [ms] (mean, across all concurrent requests)
Transfer rate:          79.56 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:     1    6  28.2      2     202
Waiting:        1    6  28.2      2     202
Total:          1    6  28.2      2     202

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      2
  75%      2
  80%      2
  90%      2
  95%      2
  98%    202
  99%    202
 100%    202 (longest request)
```

While the output on the first terminal will be (100 times) as follows:

```
...
127.0.0.1 - - [10/Jun/2017 04:30:26] "GET /hello HTTP/1.0" 200 12
127.0.0.1 - - [10/Jun/2017 04:30:26] "GET /hello HTTP/1.0" 200 12
127.0.0.1 - - [10/Jun/2017 04:30:26] "GET /hello HTTP/1.0" 200 12
...
```