**Approach**

We decided to use an AWS (Amazon Web Services) EC2 instance running Ubuntu 14.04 as our machine/hardware for the Web Server. Also, we used Python 2.7 as the programming language for two reasons: 1. Simple and easy to develop applications; and 2 Available by default in our selected machine.

**Part A: Basic Multithreaded Web Server**

We started by implementing a simple Web Server which accepts HTTP GET requests, and print the request headers in our console.

* To begin with, we used BaseHTTPServer module for implementing our HTTP web server. In this module, we used HTTPServer class to create and listens at the HTTP socket, dispatching the requests to a handler.
* For handling the requests, we used SimpleHTTPRequestHandler class from SimpleHTTPServer module.
* The do\_GET() method in the Handler maps the request to a local file.

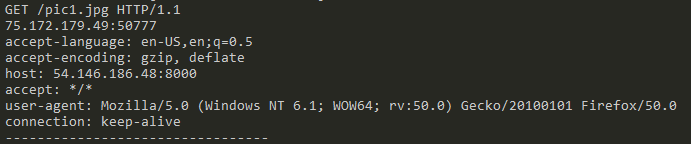
Multithreading

* Python provides SocketServer module to allow network servers to process requests asynchronously. In particular, we used ThreadingMixIn class to create a separate process or thread to handle each request.
* The serve\_forever() method of the SocketServer class keeps handling requests until we explicitly shutdown the server with a KeyboardInterrupt.

The connection basically happens over a 3-step process:

1. A RequestHandler class is created, subclassing SimpleHTTPRequestHandler class.
2. The server class ThreadingHTTPServer is instantiated (as the variable httpd), passing it the server’s address (‘’ or ‘localhost’), port (8000) and the request handler.
3. The serve\_forever() method of the server class is called, to process one or multiple client requests.

In the do\_GET() method of the request handler, the headers dictionary contains a mapping of the header keys and values. This was printed to the console output, along with command (GET), requested path (starting with /) and request\_version, like below:



Also, multiple simultaneous connections from different computers were set up and the multithreading was successfully verified.

**Part B: Request Analysis and Response**

Once we were able to successfully decode a client request; we began with parsing the request.

The path variable of the request handler contains the requested path from the client.

We tested different MIME types, as below:

* text (/, /index.html)
* image (/pic1.jpg, /arnold.gif)

The program tests the ending part of the path string, and associates the corresponding MIME type. If the file is present in the server’s directory, a 200 OK Response along with the file is sent back to the client.

Exceptions (IOError) are handled while opening the requested file. If an unknown MIME type is found or a known MIME type but file not present, 404 Error Code with a custom error page (/error.html) is sent back to the client.

Since the server keeps running indefinitely, a keyboard Interrupt (Ctrl + C) is required to manually terminate the server.