NGINX as a file server

NGINX is good at serviing static files such as images and html files. It means it's already a great server for downloading. In this article I expand it by adding features of authentication, uploading and deleting files using lua. I will also talk about the community's favorite nginx-upload-module at the end of this article.

CONTENTS

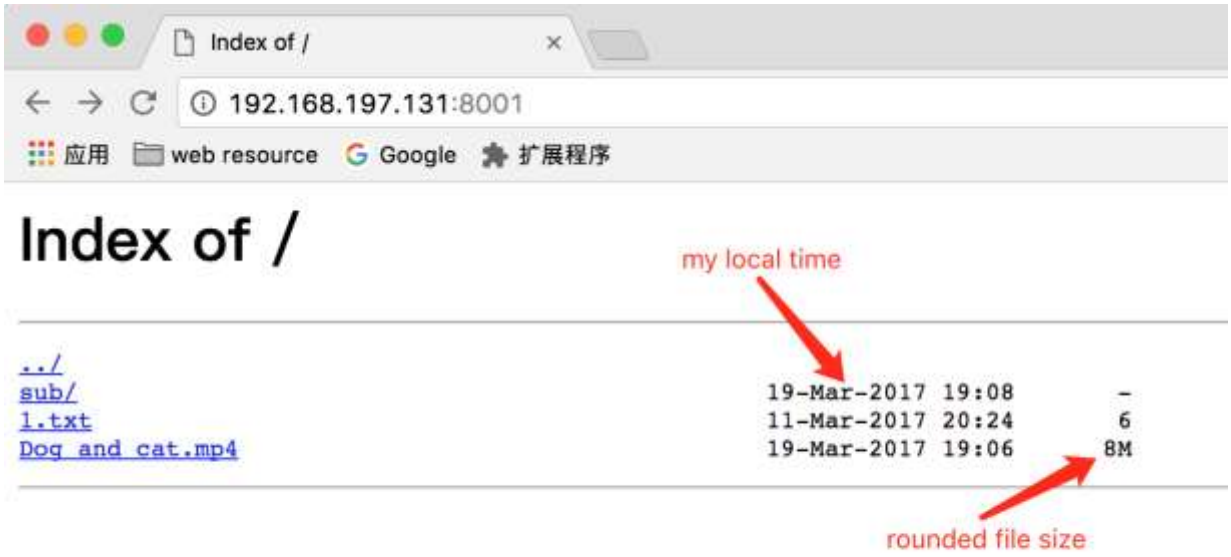
Download

When we send a get request, NGINX searches for a file by appending URI to the path specified by root. If the URI ends with a slash, NGINX treats it as a directory and tries to find an index file which is index.html by default in it. If such a file can not be found, NGINX returns HTTP code **403(Forbidden)**. The ngx_http_autoindex_module can return an automatically generated directory listing in this case.

```
server {
    listen       8001;          # a customed port


    # download
    autoindex on;               # enable directory listing output
    autoindex_exact_size off;   # output file sizes rounded to kilobytes, megabytes, and gigabytes
    autoindex_localtime on;     # output local times in the directory


    location / {
        root upload;
    }
}
```

upload is a folder under NGINX's `prefix` which is `/opt/nginx` in my case:

```
upload/
├── 1.txt
├── Dog and cat.mp4
└── sub
    └── 2.txt
```



If you use a modern browser, you can directly preview many file types such as videos, pdfs, etc. Like the built-in autoindex module, aperezdc/ngx-fancyindex is a fancier alternative of autoindexing by adding customized theme.

Notice: Executable permissions of the listing directory are required besides read permissions.

```
location / {
    root upload;
}
```

is equivalent to:

```
location /download {
    alias upload;
}
```

See Nginx — static file serving confusion with root & alias

Authentication

The ngx_http_auth_basic_module allows limiting access to resources by validating the user name and password using "HTTP Basic Authentication" protocol.

```
server {
    # auth
    auth_basic "Restricted site";
    auth_basic_user_file /opt/nginx/.htpasswd;
}
```

Only two directives are provided by this module.

auth_basic

enables authentication and the string paramter is used as realm. Some browers can display this message in the prompt.

auth_basic_user_file

is the file that keeps user names and **hashed passwords**. Most directives which need a path, `access_log` for example, can take a relative path(relative to nginx's prefix) as argument . But `auth_basic_user_file` must be an absolute path otherwith you will see "403 Forbidden" error page.

Let's generate this file by OpenSSL utilities which may already be available on most servers:

```
echo -n 'foo:' >> .htpasswd
openssl passwd >> .htpasswd
# type your password twice


cat .htpasswd
foo:xOVvNJCt4.P76
```

add another user bar:

```
echo -n 'bar:' >> .htpasswd
openssl passwd -apr1 >> .htpasswd
# type your password twice
cat .htpasswd
foo:xOVvNJCt4.P76
bar:$apr1$/hbFh44e$D5RZ91WBHCQlBymeuMCIv.
```

`-apr1` means the password is hashed with the Apache variant of the MD5-based password algorithm which is more secure than the default **crypt** algorithm.
Notice that the encrypted password will be different each time even if you use an identical password. It's magical, isn't it? Because a unique salt is chosen each time. The first 2 characters of crypt's output is salt and in the second case the output format is `$apr1$salt$hash`. If you specify salt by using `-salt option` you will always get the same result.

Now refresh the page, user will be asked for username and password. If you enter the correct credentials, you will be allowed to access these locations. Otherwith, you will see "401 Authorization Required" error page.

upload

I know it's not the best way of handling file upload, but NGINX is versatile enough to achieve this without any backend support. Until now, all the features above are provided by nginx itself. Making NGINX support uploading needs help from NGINX 3rd Party Modules](https://www.nginx.com/resources/wiki/modules/).

lua-resty-upload

Here I use openresty/lua-resty-upload which is a lua module based on ngx_lua cosocket. It **requires lua-nginx-module** to be compiled into NGINX.

lua-resty-upload contains only one file upload.lua. Place this file to `/usr/local/lib/lua/5.1/resty/upload.lua` and then add it to ngx_lua's LUA_PATH search path by directive `lua_package_path`.

```
lua_package_path '/usr/local/lib/lua/5.1/?.lua;;';
```

This module is so simple that user just needs to call the read method chunk by chunk. Please refer it's readme documentation for details about this api.

nginx_upload.conf

Bellow is the full configuration:

```
pid        logs/nginx_upload.pid;

events {
    worker_connections  1024;
}


http {
    lua_package_path '/usr/local/lib/lua/5.1/?.lua;;';


    server {
        listen       8001;


        # download
        autoindex on;
        autoindex_exact_size off;
        autoindex_localtime on;
```

```
        # auth
        auth_basic "Restricted site";
        auth_basic_user_file /opt/nginx/.htpasswd;

        location /download {
            alias upload;
        }


        location ~ ^/upload_lua(/.*)?$ {
            set $store_path upload$1/;
            content_by_lua_file conf/lua/my_upload.lua;
        }

        location ~ ^/delete/(.*)$ {
            set $file_path upload/$1;
            content_by_lua_file conf/lua/my_delete.lua;
        }
    }
}
```

my_upload.lua

```
local upload = require "resty.upload"

local function my_get_file_name(header)
    local file_name
    for i, ele in ipairs(header) do
        file_name = string.match(ele, 'filename="(.*)"')
        if file_name and file_name ~= '' then
            return file_name
        end
    end
    return nil
end

local chunk_size = 4096
```

```
local form = upload:new(chunk_size)
local file
local file_path
while true do
    local typ, res, err = form:read()

    if not typ then
         ngx.say("failed to read: ", err)
         return
    end

    if typ == "header" then
        local file_name = my_get_file_name(res)
        if file_name then
            file_path = ngx.var.store_path..file_name
            file = io.open(file_path, "w+")
            if not file then
                ngx.say("failed to open file ", file_path)
                return
            end
        end

    elseif typ == "body" then
        if file then
            file:write(res)
        end

    elseif typ == "part_end" then
        if file then
            file:close()
            file = nil
            ngx.say("upload to "..file_path.." successfully!")
        end
    elseif typ == "eof" then
        break

    else
```

```
        -- do nothing
    end
end
```

Every time a multipart/form-data form field is encouterred a file is created using the original file name if the header of that field contains filename attribute.

my_delete.lua

I also add a location which enables user to delete a file or an empty folder.

```
local function file_exists(path)
  local file = io.open(path, "rb")
  if file then file:close() end
  return file ~= nil
end
if not file_exists(ngx.var.file_path) then
    ngx.say("file not found: "..ngx.var.file_path)
end
r, err = os.remove(ngx.var.file_path)
if not r then
    ngx.say("failed to delete: "..err)
else
    ngx.say("delete successfully!")
end
```

usage

```
❯ curl -H "Authorization: Basic Zm9vOjEyMzQ1Ng==" -F filea=@a.txt -F fileb=@b.txt http://192.168.197.131:8001/upload_lua
upload to upload/a.txt successfully!
upload to upload/b.txt successfully!
```

I also set a variable `store_path` to denote the path where this file should be saved. Including the part after `/upload_lua/` of URI enables user to control in which sub folder to save files in this path. But such sub folders should already exist or error will occur.

```
❯ curl -H "Authorization: Basic Zm9vOjEyMzQ1Ng==" -F filea=@a.txt -F fileb=@b.txt http://192.168.197.131:8001/upload_lua/sub
upload to upload/sub/a.txt successfully!
upload to upload/sub/b.txt successfully!
```

I prefer cURL to Postman for debugging. Postman can't display the full request body of Content-Type multipart/form-data for now. cURL with `--trace-ascii` option can dump all incoming and outgoing data flow although it's a little hard to read. Here is an example:

```
❯ curl -H "Authorization: Basic Zm9vOjEyMzQ1Ng==" -F filea=@a.txt --trace-ascii - http://192.168.197.131:8001/upload_lua
== Info:   Trying 192.168.197.131...
== Info: Connected to 192.168.197.131 (192.168.197.131) port 8001 (#0)
=> Send header, 263 bytes (0x107)
0000: POST /upload_lua HTTP/1.1
001b: Host: 192.168.197.131:8001
0037: User-Agent: curl/7.49.1
0050: Accept: */*
005d: Authorization: Basic Zm9vOjEyMzQ1Ng==
0084: Content-Length: 191
0099: Expect: 100-continue
00af: Content-Type: multipart/form-data; boundary=--------------------
00ef: ----2ccbb2d137903ef2
0105:
<= Recv header, 23 bytes (0x17)
0000: HTTP/1.1 100 Continue
=> Send data, 136 bytes (0x88)
0000: --------------------------2ccbb2d137903ef2
002c: Content-Disposition: form-data; name="filea"; filename="a.txt"
006c: Content-Type: text/plain
0086:
=> Send data, 7 bytes (0x7)
0000: hi yxr!
=> Send data, 48 bytes (0x30)
0000:
0002: --------------------------2ccbb2d137903ef2--
<= Recv header, 17 bytes (0x11)
0000: HTTP/1.1 200 OK
<= Recv header, 22 bytes (0x16)
0000: Server: nginx/1.11.8
<= Recv header, 37 bytes (0x25)
0000: Date: Tue, 21 Mar 2017 03:15:05 GMT
<= Recv header, 40 bytes (0x28)
0000: Content-Type: application/octet-stream
```

```
<= Recv header, 28 bytes (0x1c)
0000: Transfer-Encoding: chunked
<= Recv header, 24 bytes (0x18)
0000: Connection: keep-alive
<= Recv header, 2 bytes (0x2)
0000:
<= Recv data, 48 bytes (0x30)
0000: 25
0004: upload to upload/a.txt successfully!.
002b: 0
002e:
upload to upload/a.txt successfully!
== Info: Connection #0 to host 192.168.197.131 left intact
```

As you can see the content of file a.txt is `hi yxr!`.
Deleting a file is as simple as:

```
❯ curl --user foo:123456 http://192.168.197.131:8001/delete/1.txt
delete successfully!
```

Now we have set up a simple file server using NINGX.

nginx-upload-module

Actually I seek help from nginx-upload-module at first. It was an awesome module. However, the owner is no longer maintaining this module.

Below is what I tried with this module which may be helpful in case you want to play with it yourself.

You should download branch 2.2 as it contains the newest commits.

There is a compile error: md5.h: No such file or directory because NIGNX uses internal MD5 and SHA implementations and further the internal ngx_md5.h is different from openssl's md5.h. Here is a simple patch which fixes this problem: fix-md5.h-No-such-file-or-directory

If you see error simimlar to `undefined reference to 'MD5_Upddate'` when compiling NGINX you may need to add `--with-ld-opt='-lssl -lcrypto'`:

```
./configure --prefix=/opt/nginx \
--with-ld-opt='-lssl -lcrypto' \
--add-module=/home/yanxurui/nginx/nginx-upload-module
--with-debug
```

common usage

This module is typlically used to handle file uploads without passing them to backend server. Here is a simple example from this module with slight modification.

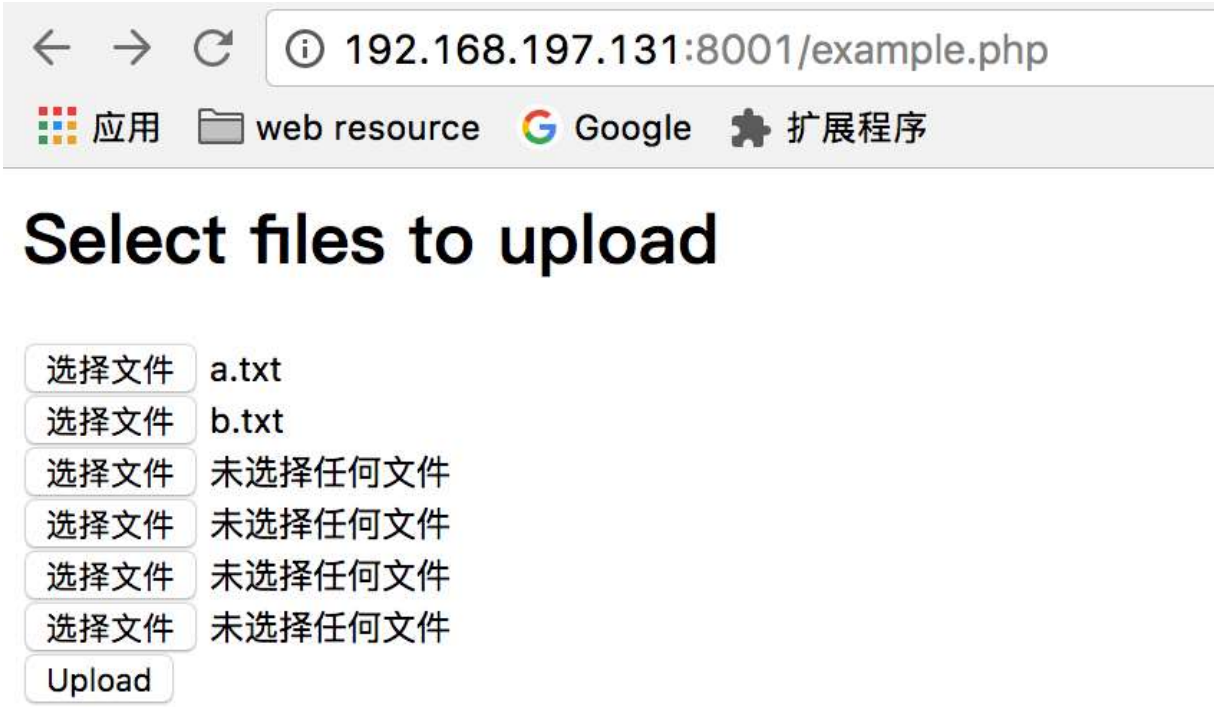
```
# upload
client_max_body_size 100m;
# Upload form should be submitted to this location
location /upload {
    # Pass altered request body to this location
    upload_pass /example.php;
    # 开启resumable
    upload_resumable on;
    # Store files to this directory
    # The directory is hashed, subdirectories 0 1 2 3 4 5 6 7 8 9 should exist
    upload_store /tmp/upload 1;
    upload_state_store /tmp/state;

    # Allow uploaded files to be read only by user
    upload_store_access user:r;

    # Set specified fields in request body
    upload_set_form_field "${upload_field_name}_name" $upload_file_name;
    upload_set_form_field "${upload_field_name}_content_type" $upload_content_type;
    upload_set_form_field "${upload_field_name}_path" $upload_tmp_path;

    # Inform backend about hash and size of a file
    upload_aggregate_form_field "${upload_field_name}_md5" $upload_file_md5;
    upload_aggregate_form_field "${upload_field_name}_size" $upload_file_size;

    upload_pass_form_field "^submit$|^description$";
}
location ~ \.php$ {
   # fastcgi_pass   unix:/run/php-fpm/php-fpm.sock;
   fastcgi_pass   127.0.0.1:9000;
   fastcgi_index  index.php;
   # fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
   fastcgi_param   SCRIPT_FILENAME    $document_root$fastcgi_script_name;
```

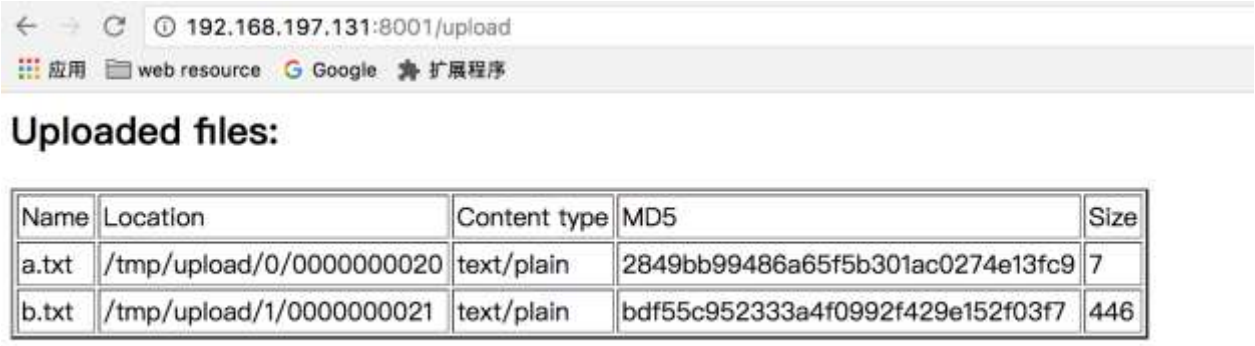
```
   include        fastcgi_params;
}
```

example.php should be placed in html folder:

```
<?php
$header_prefix = 'file';
$slots = 6;
?>
<html>
<head>
<title>Test upload</title>
</head>
<body>
<?php
if ($_POST){
    echo "<h2>Uploaded files:</h2>";
    echo "<table border=\"2\" cellpadding=\"2\">";

    echo                                "<tr><td>Name</td><td>Location</td><td>Content
type</td><td>MD5</td><td>Size</tr>";

    for ($i=1;$i<=$slots;$i++){
        $key = $header_prefix.$i;
        if            (array_key_exists($key."_name",            $_POST)            &&
array_key_exists($key."_path",$_POST)) {
            $tmp_name = $_POST[$key."_path"];
            $name = $_POST[$key."_name"];
            $content_type = $_POST[$key."_content_type"];
            $md5 = $_POST[$key."_md5"];
            $size = $_POST[$key."_size"];

            echo
"<tr><td>$name</td><td>$tmp_name</td><td>$content_type</td><td>$md5</td><td>$size</td
>";
        }
    }

    echo "</table>";
```

```
}else{?>
<h2>Select files to upload</h2>
<form name="upload" method="POST" enctype="multipart/form-data" action="/upload">
<input type="file" name="file1"><br>
<input type="file" name="file2"><br>
<input type="file" name="file3"><br>
<input type="file" name="file4"><br>
<input type="file" name="file5"><br>
<input type="file" name="file6"><br>
<input type="submit" name="submit" value="Upload">
<input type="hidden" name="test" value="value">
</form>
<?php
}
?>
</body>
</html>
```





Name	Location	Content type	MD5	Size
a.txt	/tmp/upload/0/0000000020	text/plain	2849bb99486a65f5b301ac0274e13fc9	7
b.txt	/tmp/upload/1/0000000021	text/plain	bdf55c952333a4f0992f429e152f03f7	446

Files are stored under directory specified by upload_store directive using a simple hash algorithm. For example `upload_store upload 1 2` may result the file saved as `/opt/nginx/upload/1/05/0000000018`. Subdirectories 1 and 05 are both randomly selected and should exist before uploading. Hashed subdirectory is optional. File is renamed to a 10 digits to avoid conflict in case a file with the same name exists.

Fortunately, upload_store can also accept variables.

Hack it as a normal file server

What if I want to save the file as its original file name? Let's hack it.

This module implements a mechanism of resumable file upload aka partial upload. It means a big file can be splitted into severl segments and then uploaded one by one in seperate Post requests. Client is responsible for choosing an unique Session-ID which is an identifier of a file being uploaded as well as the name of file saved in server.

Yes, I can achieve my goal by using the file name as Session-ID. Apply this patch to allow Session-ID to contain dot. hack-allow-session-id-to-contain-dot

configuration now looks like this:

```
# Upload form should be submitted to this location
```

```
location ~ ^/upload_mod(/.*)?$ {
    # Pass altered request body to this location
    upload_pass /upload_return;
    # 开启resumable
    upload_resumable on;

    upload_store upload$1;
    upload_state_store /tmp/state;
}
location /upload_return {
    return 200 ok;
}
```

test file upload, `Content-Type` and `X-Content-Range` are omitted:

```
❯ curl -X POST --user foo:123456 -H "Content-Disposition: attachment, filename=\"a.txt\"" -H "X-Session-ID: a.txt" --data-binary @a.txt "http://192.168.197.131:8001/upload_mod"
ok%
```

File a.txt is uploaded successfully:

```
❯ cat a.txt
hi yxr!
❯ curl --user foo:123456 "http://192.168.197.131:8001/download/a.txt"
hi yxr!
```

resumable file uploads

```
❯ curl -X POST --user foo:123456 -H "Content-Type: text/plain" -H "Content-Disposition: attachment, filename=\"a.txt\"" -H "X-Session-ID: a.txt" -H "X-Content-Range: bytes 0-4/10" --data-binary hello "http://192.168.197.131:8001/upload_mod"

0-4/10
❯ curl -X POST --user foo:123456 -H "Content-Type: text/plain" -H "Content-Disposition: attachment, filename=\"a.txt\"" -H "X-Session-ID: a.txt" -H "X-Content-Range: bytes 5-9/10" --data-binary world "http://192.168.197.131:8001/upload_mod"

ok
❯ curl --user foo:123456 "http://192.168.197.131:8001/download/a.txt"
helloworld
```

reference

- how to set up password authentication with nginx on ubuntu-14-04
- Resumable uploads over HTTP. Protocol specification

« python里的数值类型使用robotframework测试nginx的location指令 »