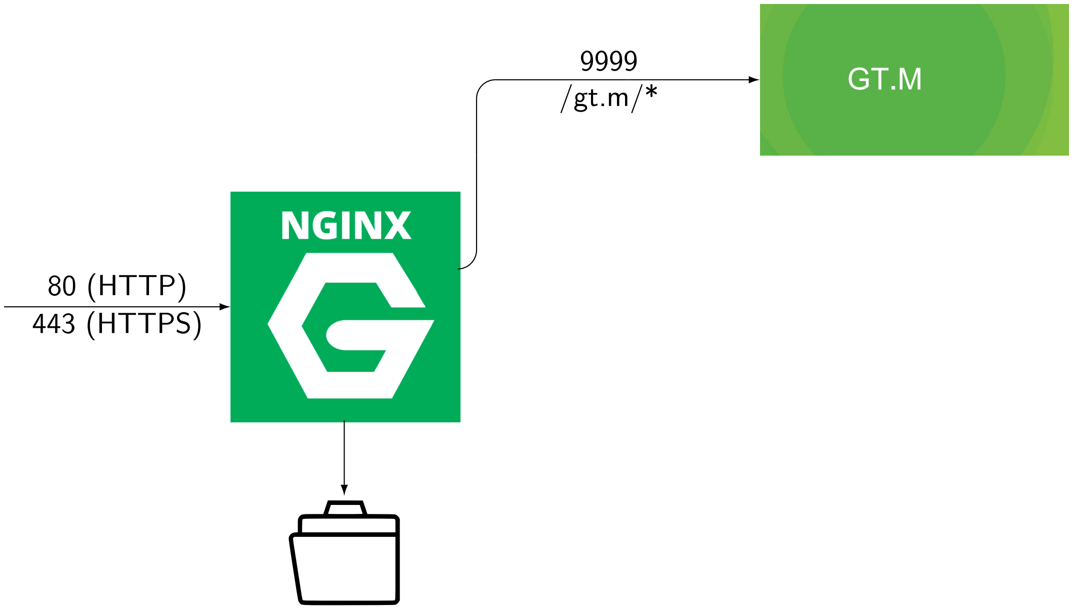
FastCGI for GT.M - Installation and Quick-Start

Winfried Bantel

Aalen University

1. April 2019

The nginx-system



Advantages NOT to write a HTTP-Server

- Very very fast FastCGI-backend written in native GT.M
- nginx is able to cache - less work for GT.M
- HTTPS supported by nginx
- HTTP/2 supported by nginx
- HTTP/2 with dynamic server push for even faster applications
- Filebased Webserver is done by nginx
- With JSON-Parser ideal backend for Single-Page-Applications (i.e. with AngularJS)
- Supports massive parallel HTTP-requests
- Sensible data can be stored physically on another machine
- Other backends like php, couchdb on the same webserver

Test via apache benchmark

```
>>> ab -n 1000 -c 10 -q "localhost/gt.m/dollarh"
...
Concurrency Level:        10
Time taken for tests:     4.568 seconds
Complete requests:        1000
Failed requests:          0
Total transferred:        178920 bytes
HTML transferred:         13000 bytes
Requests per second:      218.90 [#/sec] (mean)
Time per request:         45.683 [ms] (mean)
Transfer rate:            38.25 [Kbytes/sec] received
...
Percentage of the requests served within a certain time (ms)
  50%     40
  66%     40
  75%     40
  80%     40
  90%     40
  95%     40
  98%     42
  99%    558
 100%    620 (longest request)
```

Installation-Steps

I hope You are firm in GT.M!

1. Install nginx
2. Edit nginx-Config
3. Install fis-gtm
4. Install xinetd
5. Edit xinetd-Config-Script
6. Copy FCGI.m
7. Set a global
8. Be happy

User and so on

- In these slides the user is wbantel.
- His home-directory is /home/wbantel/
- If You want another user: adapt!

Step 1: Install nginx

```
>>> sudo apt install nginx
>>> curl localhost
```

Or test from any Computer in WWW / LAN with IP-Address oder DNS

Step 2: Edit nginx-Config

Edit `/etc/nginx/sites-enabled/default`:

- In the global section:

```
upstream gtm_fcgi_backend {
        server 127.0.0.1:9999;
        keepalive 32;
}
```

- In the server-section:

```
            location /gt.m/ {
                   fastcgi_pass gtm_fcgi_backend;
                   fastcgi_keep_conn on ;
                   fastcgi_param   QUERY_STRING        $query_string;
                   fastcgi_param   SID                 $cookie_sid;
                   fastcgi_param   DOCUMENT_URI        $document_uri;
                   fastcgi_param   REQUEST_METHOD      $request_method;
                   fastcgi_param   REMOTE_ADDR         $remote_addr;
           }
```

Restart nginx:

```
>>> sudo service nginx restart
```

Step 3: Install fis-gtm

```
>>> sudo apt install fis-gtm
>>>
```

Step 4: Install xinetd

```
>>> sudo apt install xinetd
>>>
```

Step 5: Edit xinetd-Config-Script

```
==> sudo find /usr/lib/ -name "gtm"
/usr/lib/x86_64-linux-gnu/fis-gtm/V6.3-003A_x86_64/utf8/gtm
...
==>  cat /etc/xinetd.d/gtm-fastcgi
service gtm-fastcgi
{
        protocol        = tcp
        port            = 9999
        type            = UNLISTED
        socket_type     = stream
        wait            = no
        user            = wbantel
        group           = wbantel
        server          = <put here the correct gtm-path from above>
        server_args     = -run FCGI
        env             = gtmdir=/home/wbantel/.fis-gtm
        disable         = no
}
==> sudo service xinetd restart
```

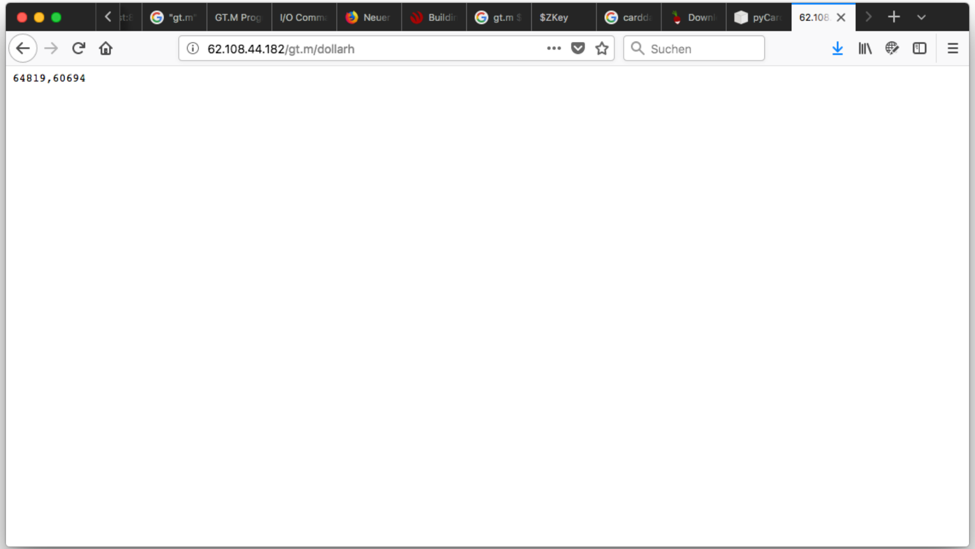
Step 6: Copy FCGI.m

```
>>> cp /from/somewhere/FCGI.m /home/wbantel/.fis-gtm/V.../r/
>>>
```

Step 7: Set a global

```
>>>  /home/wbantel/mumps.sh
GTM> SET ^FCGI("DOCUMENT_URI","/gt.m/dollarh")="DOLLARH"
GTM>
>>>
```

Step 8: Be happy



```
64819,60694
```

More than one System

You need another GTM-System, perhaps for development and production, totally different?

- Create another user
- Create another xinetd-Config with another TCP/IP-Port and another name
- Create another Upstream-Part with the correct Port in nginx-Config
- Create another Location-Part with another URI an the correct Upstream in nginx-Config

Some configuration-parameters

```
^FCGI("PRM","ZLINK")
^FCGI("PRM","LOG")
^FCGI("PRM","GZ")
^FCGI("PRM","TO")
```

ZLINK Use this parameter for developing (set to 1) so when you edit a routine and save it the changes will have an effect (suitable for developing). Otherwise kill the global and it will run a little bit faster (suitable for production).

- 0 (or killed): The called routine will be called without ZLINK
- 1: The called routine will be ZLINKed before called

LOG Some logging in /tmp/fastcgi.log

- 0 (or killed): Logging off
- 1: Logging on

GZ Output written to %fcgi will be compressed before sent. Needs some time, but transmission will be faster. (Not needed for HTTP/2!)

- 0 (or killed): ZIPping off
- 1: ZIPping on

TO Timeout a job will wait for a second request. Default is 60 seconds.

How to call an M-routine

- FastCGI examines `$PIECE(uri,"/",1,3)`
  Attention, first piece is alway empty! I.e. `/gt.m/dollarh` third piece is dollarh
- Second piece has to be the location from nginx-config-file (usually gt.m)
- Third piece is variable and used for distribute to application-routine
- Set an Indirection-Global for Your app (see step 7)
- Forth / fifth / ... piece can be used in application, i.e. a REST-Interface:
  `/gt.m/rest/customer/1` points to rest-interface for file (global) „customer“ and database-index 1

How to generate Output

Several ways for backend-routine to generate output

1. Write to device %fcgi
2. Set a global-name
3. Set a filename
4. Set a single variable
5. Set an array variable
6. Callback-Functions (direct output)

Don't mix it up, use only exactly one way!

Generate output - device %fcgi

- Easiest way to generate Output

```
EXOUTPUT1     ; Generates output using %fcgi
        ; On start %fcgi is open and used!!!
        w "<html><head></head><body>",$H,"</body></html>"
```

Generate output - globalbased

- Ideal in case of the global already exists

```
EXOUTPUT2     ; Generates output using global
        s ^dummy="<html><head></head><body>"_$H_"</body></html>"
        s %fcgi("o","global")="^dummy"
```

Generate output - filebased

- Ideal in case of the file already exists

```
EXOUTPUT3    ; Generates output using file / filename
        s f="/tmp/"_$j_".html"
        u f w "<html><head></head><body>"_$H_"</body></html>" c f
        s %fcgi("o","file")=f
```

Generate output using local variable

```
EXOUTPUT4    ; using local variable
      s %fcgi("o","stdout")"<html><head></head><body>"_$H_"</body><
```

Generate output using local array

```
EXOUTPUT5     ; Generate output using array
        s %fcgi("o","stdout",1)="<html>"
        s %fcgi("o","stdout",2)="<head></head>"
        s %fcgi("o","stdout",3)="<body>"_$H_"</body>"
        s %fcgi("o","stdout",4)="</html>"
```

Generate output using callback-functions

- Fastest of all
- No buffer!

How-to:

- Set Header (optional)
- Call `HEADEROUT^FCGI`
- Call (repatedly) `DATAOUT^FCGI(...)` (optional)
- `SET %fcgi("o","noout")=1`

Generate output using callback-functions

```
EXOUTPUT6; Direct Output
        s %fcgi("o","header","Content-Type")="application/json"  ; opt
        d HEADEROUT^FCGI ; mandatory
        d DATAOUT^FCGI("{""$H"":"""_$H_"""}") ; optional
        s %fcgi("o","noout")=1 ; mandatory
        q
```

Output-Header

- For Content-Type, Redirect and so on

```
EXSETHEADER     ; Generates output using %fcgi
        s %fcgi("o","header","Content-Type")="application/json"
        w "{""$H"":""" _$H_ """,""$J"":""" _$J_ """}"
```

```
>>> curl -i "localhost:8080/gt.m/EXSETHEADER"
HTTP/1.1 200 OK
Server: nginx/1.14.0
Date: Wed, 09 Jan 2019 14:07:03 GMT
Content-Type: application/json
Content-Length: 32
Connection: keep-alive
X-job: 2483
X-nr: 1

{"$H":"65022,54423","$J":"2483"}
```

Session-ID is the new $JOB

- Session-tracking ist forced calling `SID^FCGI`
- Stored in `%fcgi("i","header","SID")`
- Two Comma-separated integers:
  1. 64-bit random-int which ist constant for your session
  2. Counter auto-incrementing with each HTTP-request
- Is done by a temporary (non-persistant) cookie
- Ideal for storing session-specific data

```
EXSID    ; Generates output using %fcgi
      q:'$$SID^FCGI()   s sid=%fcgi("i","header","SID")
      w "<html><head></head><body>"
      w "Your Session-ID is ",+sid,"<br>",!
      w "Your Session-count is ",$P(sid,",",2),"<br>",!
      w "Your last visit ($H) was: ",$G(^dummy(+sid)),"<br>",!
      s h=$H w "Now $H is: ",h,"<br>",!
      s ^dummy(+sid)=h
      w "<br>Feel free to reload!"
      w "<br><a href=""javascript:location.reload()"">Reload</a>"
      w "</body></html>"
```

HTTP-GET-vars

- Easiest way to get data from Webclient

```
EXGETVAR    ;
        w "<html><head></head><body>"
        i $G(%fcgi("i","_GET","name"))="" d
        . w "You did't enter a name"
        e  w "Hello ",%fcgi("i","_GET","name"),"!"
        w "<form method=""GET"">",!
        w "<input type=""text"" name=""name"">",!
        w "<input type=""submit"" value=""Submit"">",!
        w "</form></body></html>"
```

HTTP-POST-vars

```
EXPOSTVAR     ;
        w "<html><head></head><body>"
        i $G(%fcgi("i","_POST","name"))="" d
        . w "You did't enter a name"
        e  w "Hello ",%fcgi("i","_POST","name"),"!"
        w "<form method=""POST"">",!
        w "<input type=""text"" name=""name"">",!
        w "<input type=""submit"" value=""Submit"">",!
        w "</form></body></html>"
```

HTTP-BODY

- Suitable for JSON-data, File-Uploads and so on

```
EXSTDIN     ;
        ; > curl ip-address:port/gt.m/EXPOSTVAR -d "Hallo Welt!"
        ; > curl ip-address:port/gt.m/EXPOSTVAR -d @file.txt
        ; Or a Browser-form with method post:
        ; <form action="/gt.m/EXPOSTVAR" method="POST">...</form>
        w "<html><head></head><body>Your Post-Data is<pre>"
        w $G(%fcgi("i","stdin"))
        w "</pre></body></html>",!
```

```
>>> curl -i "localhost:8080/gt.m/EXSTDIN" -d '{"NN":"Bantel"}'
HTTP/1.1 200 OK
Server: nginx/1.14.0
Date: Wed, 09 Jan 2019 14:13:28 GMT
Content-Length: 83
Connection: keep-alive
X-job: 2699
X-nr: 2

<html><head></head><body>Your Post-Data is<pre>{"NN":"Bantel"}</pre></body
```

Complete HTTP-Info

Hochschule Aalen

- The complete info is stored in %fcgi

```
EXHTTPINFO    ;;
        s %fcgi("o","header","Content-Type")="text/plain"
        zwr %fcgi
```

```
>>>  curl "localhost:8080/gt.m/EXHTTPINFO?test=1"  -d '{"NN":"Bantel"}'
%fcgi="/tmp/fcgi-fifo-4011" ;*
%fcgi("i","FCGI_KEEP_CONN")=1
%fcgi("i","_GET","test")=1
%fcgi("i","_POST","{""NN"":""Bantel""}")=""
%fcgi("i","header","DOCUMENT_URI")="/gt.m/EXHTTPINFO"
%fcgi("i","header","HTTP_ACCEPT")="*/*"
%fcgi("i","header","HTTP_CONTENT_LENGTH")=15
%fcgi("i","header","HTTP_CONTENT_TYPE")="application/x-www-form-urlencoded
%fcgi("i","header","HTTP_HOST")="localhost:8080"
%fcgi("i","header","HTTP_USER_AGENT")="curl/7.51.0"
%fcgi("i","header","QUERY_STRING")="test=1"
%fcgi("i","header","REMOTE_ADDR")="10.0.2.2"
%fcgi("i","header","REQUEST_METHOD")="POST"
%fcgi("i","header","SID")=""
%fcgi("i","stdin")="{""NN"":""Bantel""}"
%fcgi("internal","entryRef")="^EXHTTPINFO"
```

Embedding GT.M-output in static HTML

- Use Server-Side includes
- in nginx-config

```
location /some/where {
        ssi on;
}
```

- in HTML

```
<html>
<head></head>
<body>
<h1>GT.M</h1>
<pre><!--# include virtual="/gt.m/EXHTTPINFO?$args" --></pre>
</body>
</html>
```

- Works only for HTTP-GET

M on another server

A WWW-Server is always vulnerable!
Secure Your important data , don't store them in the WWW-Server!

- Change the address of the FastCGI-backend in nginx-config (see Step 2 above)!
  Example:

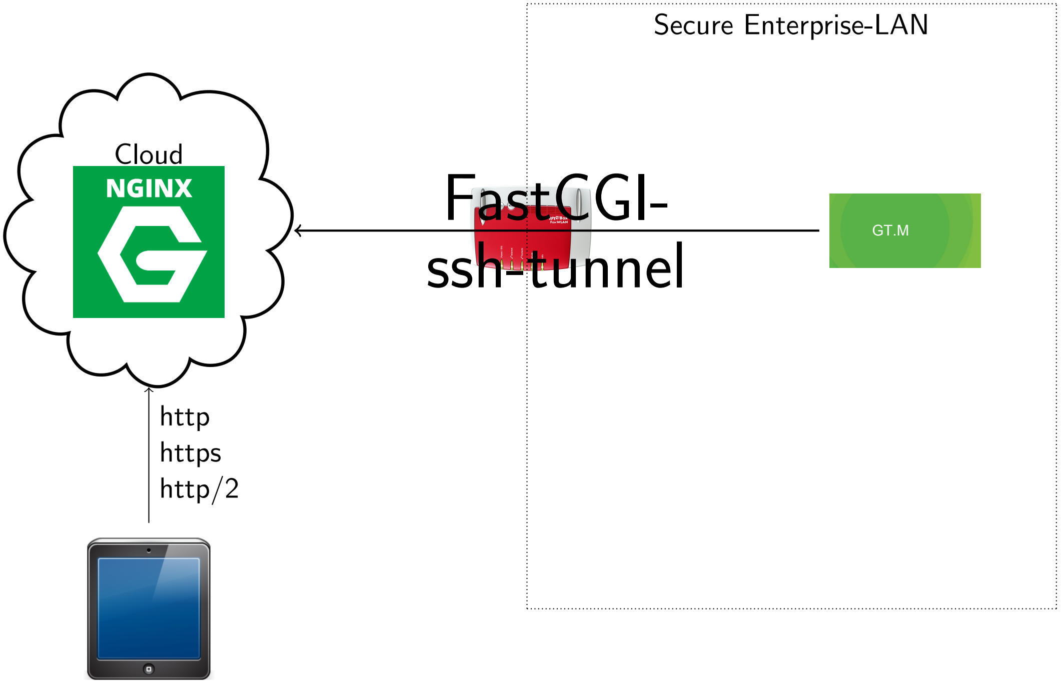
```
upstream gtm_fcgi_backend {
        server 192.168.10.12:9999;
        keepalive 32;
}
```

Load balancing: M on multiple servers

- More than one FastCGI-backend in nginx-config (see Step 2 above)!
  Example:

```
upstream gtm_fcgi_backend {
        server 192.168.10.12:9999;
        server 192.168.10.13:9999;
        server 192.168.10.14:9999;
        server 192.168.10.15:9999;
        keepalive 32;
}
```

M behind a firewall (1)



M behind a firewall (2)

- M is in an enterprise-LAN
- nginx is somewhere in the WWW
- A Firewall whithout Port-Forwarding
- With ssh-tunnel M-Backend becomes a TCP/IP-client in the LAN

To enable start in the M-Server

```
ssh -Nf -R 9999:localhost:9999 www.my-web-server.de
```

(Can be done better with autossh)

Using HTTP/2

- HTTP/2 is the future
- Uses
  - Compression
  - Encryption
  - Persistant connections
  - Parallel connections
- nginx supports http/2
- nginx supports dynamic server push for FastCGI-backends

Using HTTP/2 server-push

- With HTTP/2 there can be sent more than one document for one request
- In example:
  - A HTML-page with an img-tag
  - The static image for the image-tag
- It is much faster than loading html, parsing, loading image

1. In the nginx-config (minimum version 1.13.9)

```
    location /gt.m/ {
        http2_push_preload on;
       ...
   }
```

2. In the M-backend-program: Set HTTP-Header „Link“:

```
s %fcgi("o","header","Link")="</ibs/http-2/server-push.css>; rel=preloa
w $J_" "_$H_" "_$IO
```

For details visit
`https://www.nginx.com/blog/nginx-1-13-9-http2-server-push/`

Single-Page-Apps with AngularJS

Modern Web-2.0-application

- Download angular-1-7-2.min.js to the /lib/-subdirectory of nginx-root-directory
- `SET ^FCGI("DOCUMENT_URI","/gt.m/EXANGULARJS")="^EXANGULARJS")`
- Store EXANGULARJS.html and EXANGULARJS.js somwhere under the nginx-root-directory

Single-Page-Apps with AngularJS – HTML

```
<!doctype html>
<html ng-app="ajaxApp">
 <head>
  <script src="/lib/angular-1-7-2.min.js"></script>
  <script src="EXANGULARJS.js"></script>
 </head>
 <body ng-controller="Controller as q">
  <table>
   <tr><th>ID:</th><td>
    <input size="3" ng-model="q.id"/>
    <input type="button" value="Load" ng-click="q.load()"/>
   </td></tr>
   <tr><th>Vorname:</th><td>
    <input type="text" ng-model="q.address.VN">
    <tr><th>Nachname:</th><td>
    <input type="text" ng-model="q.address.NN"></th></tr>
   <tr><th></th><td>
    <input type="button" value="Save" ng-click="q.send()"/>
   {{q.savetext}}</td></tr>
  </table>
 </body>
</html>
```

Single-Page-Apps with AngularJS – JS

```
var app = angular.module('ajaxApp', []);
app.controller('Controller', function($scope, $http) {
        var c = this;
        var uri = "/gt.m/EXANGULARJS/";

        c.send = function() {
                $http.put(uri+c.id,c.address).then(function (response) {
                        c.savetext = JSON.stringify(response.data);
                        setTimeout(function(){
                                c.savetext = ""; $scope.$apply();
                        }, 2500);
                });
        };

        c.load = function() {
                $http.get(uri+c.id).then(function (response) {
                        c.address =(response.data);
                });
        };
});
```

Single-Page-Apps with AngularJS – M

```
EXANGULARJS     ; A very very simple REST-Interface
        s %fcgi("o","header","Content-Type")="application/json"
        s id=+$P(%fcgi("i","header","DOCUMENT_URI"),"/",4)
        i id<=0 w "{""ERROR"":1}" q

        i %fcgi("i","header","REQUEST_METHOD")="PUT" d
        . s ^EXANGULARJS(id)=%fcgi("i","stdin")
        . w "{""ERROR"":0,""ERRTXT"":""OK"",""ID-WRITTEN"":"""_id_"""
        e  d
        . w $S($D(^EXANGULARJS(id)):^(id),1:"{}")
```

HTML5-Server-Sent Events (SSE)

Advantages of SSE

- Browser can be informed about Server-Events
- No Polling (AJAX) required

HTML5-SSE - HTML

```
<!DOCTYPE html>
<html><head><title>Chat with SSE</title>
<script>
function init() {
      var source = new EventSource("/gt.m/fcgi-sse");
      source.addEventListener('message',f, false);
}
function f(event) {
      document.getElementById("data").firstChild.nodeValue =
      event.data;
}
</script>
</head>
<body onload="init()">
Data: <div id="data">???</div>
</body>
</html>
```

HTML5-SSE - M

- Only possible with direct-output

```
EXSSE ; SSE-Schnittstelle
        s %fcgi("o","header","Content-Type")="text/event-stream;chars
        d HEADEROUT^FCGI
        f i=1:1:1000000 d
        . s txt2send=i_": "_$H
        . d DATAOUT^FCGI("data: "_txt2send_$C(13,10,13,10))
        . h 1
        s %fcgi("o","noout")=1
        q
```