NGINX从入门到精通进阶系列培训

高级篇：SSL、NJS

李西蒙 System Engineer

TLS/SSL在NGINX中的应用

- NGINX SSL基本配置介绍
- SSL module 功能介绍
- 如何提高NGINX的SSL处理性能
- Let’s encrypt 自动化部署

HTTPS 典型配置实例

```
server {
    listen       80 default_server;
    server_name  www.example.com;
    return 301 https://$server_name$request_uri;
}
server {
    listen 443 ssl default_server;
    server_name  www.example.com;
    ssl_certificate cert.crt;
    ssl_certificate_key cert.key;
    ssl_protocols    TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers aRSA:!ECDHE:!EDH:!kDHE;
    ssl_prefer_server_ciphers on;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
}
```

- 强制HTTP流量重定向到HTTPS，满足安全标准
- 配置证书和秘钥完成最基础的SSL加解密过程
- 使用openssl 进行所有的SSL 处理

SSL 卸载

```
server {
    listen       80 default_server;
    server_name  www.example.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl ;
     ssl_certificate     server.crt;
     ssl_certificate_key server.key;
     ssl_protocols    TLSv1 TLSv1.1 TLSv1.2;
     ssl_ciphers aRSA:!ECDHE:!EDH:!kDHE;
     ssl_prefer_server_ciphers on;


    location / {
        proxy_pass http://backend
    }
}
```

- 在反向代理场景中， nginx 与客户端直接采用加密HTTPS，服务器测采用HTTP传输
- 其他配置参考典型配置即可

SNI的支持

```
server {
    listen 443 ssl ;
    server_name  www.example1.com;
    ssl_certificate cert1.crt;
    ssl_certificate_key cert1.key;
}
server {
    listen 443 ssl ;
    server_name  www.example2.com;
    ssl_certificate cert2.crt;
    ssl_certificate_key cert2.key;
}
```

- 基于SNI实现多个域名共享一个IP
- 不同域名使用不同证书
- 无需单独配置SNI

基于SNI的证书lazy loading

```
server {
    listen 443 ssl;

    ssl_certificate      /etc/ssl/$ssl_server_name.crt;
    ssl_certificate_key  /etc/ssl/$ssl_server_name.key;
    ssl_protocols        TLSv1.3 TLSv1.2 TLSv1.1;
    ssl_prefer_server_ciphers on;

    location / {
        proxy_set_header Host $host;
        proxy_pass http://my_backend;
    }
}
```

- 所有域名复用同一个server配置block，极大减少配置量
- 可以动态更新证书，无需reload
- 即使需要reload，因为配置量小，也会大大提高速度
- 在使用动态证书读取可能会造成初次SSL handshake 多用20%-30%左右的时间，但后续流量加密不受影响

SSL Session Caching

```
server {
    listen 443 ssl default_server;
    server_name  www.example.com;

    ssl_certificate cert.crt;
    ssl_certificate_key cert.key;

    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 10m;
}
```

- 避免每次请求都进行ssl handshake
- 提升 SSL/TLS performance
- 1 MB session cache 可以储存4,000 条 session
- 全部nginx worker 共享

SSL Session Ticket

```
server {
    listen 443 ssl default_server;
    server_name  www.example.com;

    ssl_certificate cert.crt;
    ssl_certificate_key cert.key;

    ssl_session_tickets      on;
    ssl_session_ticket_key   ticket_file;
}
```

- 提升SSL 性能，并减小内存使用
- 集群中多个nginx可共享session ticket

如何升级nginx的openssl版本

```
$ nginx –s stop

$ ./config –with-openssl=/usr/local/src/openssl-1.1.1d

$ make && make install
```

- Nginx 使用Openssl进行SSL处理
- SSL 漏洞的快速修复
- 编译OpenSSL 1.1.1及后续版本，可支持TLS1.3
- 开源版本可通过重新编译进行对openssl版本的升级
- 配置无需修改

SSL module 其他功能

- 完备的SSL功能支持
- 支持 SSL 双向认证
- 支持 CRL/OCSP
- 支持 椭圆曲线/DH Parameter 配置

http://nginx.org/en/docs/http/ngx_http_ssl_module.html#

Module ngx_http_ssl_module

Example Configuration
Directives
- ssl
- ssl_buffer_size
- ssl_certificate
- ssl_certificate_key
- ssl_ciphers
- ssl_client_certificate
- ssl_crl
- ssl_dhparam
- ssl_early_data
- ssl_ecdh_curve
- ssl_password_file
- ssl_prefer_server_ciphers
- ssl_protocols
- ssl_session_cache
- ssl_session_ticket_key
- ssl_session_tickets
- ssl_session_timeout
- ssl_stapling
- ssl_stapling_file
- ssl_stapling_responder
- ssl_stapling_verify
- ssl_trusted_certificate
- ssl_verify_client
- ssl_verify_depth

五分钟上线HTTPS站点---Let’s encrypt

1. 安装 Let’s Encrypt

```
$ add-apt-repository ppa:certbot/certbot
```

```
$ apt-get update
$ apt-get install python-certbot-nginx
```

2. NGINX 准备

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    root /var/www/html;
    server_name example.com www.example.com;
}
```

HTTPS自动化上线工具 ---Let’s encrypt

3. 获取证书并自动更新NGINX 配置

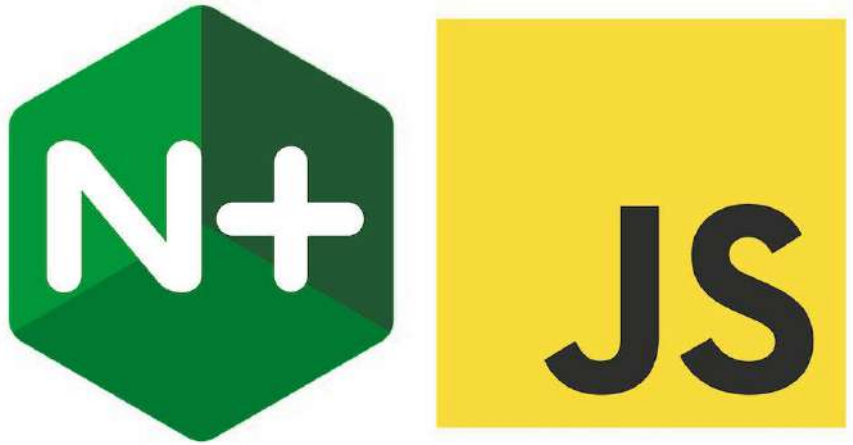
```
$ sudo certbot --nginx -d example.com -d www.example.com
```

4. 通过屏幕交互内容进行HTTPS 配置， 完成自动化证书签发和NGINX配置上线

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    root /var/www/html;
    server_name  example.com www.example.com;

    listen 443 ssl; # managed by Certbot

    # RSA certificate
    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem; # managed by
Certbot

    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot

    # Redirect non-https traffic to https
    if ($scheme != "https") {
        return 301 https://$host$request_uri;
    } # managed by Certbot
}
```

NGINX Javascript （NJS）

- 什么是NJS
- 为什么选择NJS 作为Nginx官方脚本语言
- NJS 功能及案例



什么是NGINX JavaScript （NJS）

NGINX JavaScript是NGINX和NGINX Plus的独特JavaScript实现，专门针对服务器端事件触发，请求处理设计。 它使用JavaScript代码扩展了NGINX配置语法，以实现复杂的配置解决方案。

```
function hello(r) {
    r.return(200, "Hello world!");
}
```

NOTE：
1. NGINX Plus R12/NGINX Open Source 1.11.10 版本后始支持
2. 原名NginScript

理想的脚本语言

- 快速
  - 否则使用Node.js 这种大而全的语言即可
- 能与NGINX的异步无阻塞工作模式完美集成
- 模块化
  - 如果不需要使用，可以disable模块，最大化性能
- 流行的语言
  - 使程序员编写脚本更加迅速，降低学习成本

WHY NJS

- 高度定制化的语言设计
  - 遵从ECMAScript 5.1 国际标准， 使用自研引擎
  - 支持服务器测功能。对其他功能进行裁剪，更轻量，快速
- 事件触发模型 （Event Driven Model）
  - 基于每个请求在触发事件后进行处理， 逻辑清晰，上手迅速
  - 请求之间隔离，更安全，健壮
- Javascript 被广大程序员使用

NGINX Javascript 与Lua

- LUA 是功能丰富的脚本语言，是现阶段使用最多的NGINX 脚本编程扩展工具
- Javascript 相比LUA 拥有更广泛的程序员基础，NJS对于多数程序员来说更易上手
- 两者可以配合共用
- NJS拥有更高的处理性能， 业务延时更小

NJS 处理事件及相关模块

- 针对HTTP和4层Stream处理，分别有如下处理阶段。

Processing Phase	HTTP Module	Stream Module
`Access` – Network connection access control	❌	✅ `js_access`
`Pre-read` – Read/write body	❌	✅ `js_preread`
`Filter` – Read/write body during proxy	❌	✅ `js_filter`
`Content` – Send response to client	✅ `js_content`	❌
`Log / Variables` – Evaluated on demand	✅ `js_set`	✅ `js_set`

Let’s start with Hello World

1. 安装NJS模块

```
$ sudo yum install nginx-module-njs
```

2. 创建一个NJS 脚本文件, hello_world.js.

```
function hello(r) {
    r.return(200, "Hello world!");
}
```

3. 在nginx.conf文件中，启用ngx_http_js_module模块，并使用js_include指令指定hello_world.js运行

```
load_module modules/ngx_http_js_module.so;

http {
    js_include hello_world.js;

    server {
        listen 80;

        location / {
            js_content hello;
        }
    }
}
```

自定义日志

- 实现功能
  - 日志包含客户端HTTP报头
  - 日志包括服务测HTTP报头
  - 使用KV格式记录日志，更利于与日志分析工具集成
- 实现方法
  - kvHeaders 为内部函数，需独立声明，完成格式转换
  - kvAccessLog 调用r （代表当前HTTP request）收集信息并返回最终日志

```
function kvHeaders(headers, parent) {
    var kvpairs = "";
    for (var h in headers) {
        kvpairs += " " + parent + "." + h + "=";
        if ( headers[h].indexOf(" ") == -1 ) {
        kvpairs += headers[h];
        } else {
            kvpairs += "'" + headers[h] + "'";
        }
    }
    return kvpairs;
}

function kvAccessLog(r) {
    var log = r.variables.time_iso8601;    // NGINX JavaScript can access all variables
    log += " client=" + r.remoteAddress;   // Property of request object
    log += " method=" + r.method;          // "
    log += " uri=" + r.uri;                // "
    log += " status=" + r.status;          // Property of response object
    log += kvHeaders(r.headersIn, "in");  // Send request headers object to function
    log += kvHeaders(r.headersOut, "out"); // Send response headers object to function
    return log;
}
```

自定义日志测试结果

```
1  js_include conf.d/header_logging.js;              # Load JavaScript code from here
2  js_set     $access_log_with_headers kvAccessLog; # Fill variable from JS function
3  log_format kvpairs $access_log_with_headers;     # Define special log format
4
5  server {
6      listen 80;
7      access_log /var/log/nginx/access_headers.log kvpairs;
8      location / {
9          proxy_pass http://56.1.1.2;
10     }
11 }
```

```
[root@bogon ~]# tail -n 1 /var/log/nginx/access_headers.log
2020-02-19T10:59:20+08:00 client=10.1.10.1 method=POST uri=/ status=200 in.Content-Type=application/json in.User-Agent=PostmanRuntime/7.22.0 in.Accept=*/* in.
Cache-Control=no-cache in.Host=10.1.10.158 in.Accept-Encoding='gzip, deflate, br' in.Content-Length=16 in.Connection=keep-alive out.Last-Modified='Thu, 19 Sep
 2019 07:12:05 GMT' out.ETag=\x22f-592e2abbeed42\x22 out.Accept-Ranges=bytes
```

NJS功能强大的subrequest功能

- r.subrequest(uri[, options[, callback]])
- 使用给定的uri和options创建一个子请求，并根据callback 函数决定如何处理subrequest
- Options 可以通过如下变量，进一步控制subrequest
  - args
    参数字符串， 默认情况下使用空字符串
  - body
    请求正文， 默认情况下使用父请求对象的请求正文
  - method
    HTTP方法， 默认情况下使用GET方法
- 可以更灵活的控制request和response

Subrequest 请求合并

将几个子请求的结果异步组合到一个JSON响应中

nginx.conf

```
js_include example.js;

server {
    listen 80;

    location /join {
        js_content join;
    }

    location /foo {
        proxy_pass http://localhost:8080;
    }

    location /bar {
        proxy_pass http://localhost:8090;
    }
}
```

example.js

```
function join(r) {
    join_subrequests(r, ['/foo', '/bar']);
}

function join_subrequests(r, subs) {
    var parts = [];

    function done(res) {
        parts.push({ uri:  reply.uri,
                     code: res.status,
                     body: res.responseBody });

        if (parts.length == subs.length) {
            r.return(200, JSON.stringify(parts));
        }
    }

    for (var i in subs) {
        r.subrequest(subs[i], done);
    }
}
```

```
curl http://localhost/join
[{"uri":"/foo","code":200,"body":"FOO"},{"uri":"/bar","code":
200,"body":"BAR"}]
```

Subrequest 返回响应最快的子请求

nginx.conf

```
js_include fastresponse.js;

location /start {
    js_content content;
}

location /foo {
    proxy_pass http://backend1;
}

location /bar {
    proxy_pass http://backend2;
}
```

fastresponse.js

```
function content(r) {
    var n = 0;

    function done(res) {
        if (n++ == 0) {
            r.return(res.status,
res.responseBody);
        }
    }

    r.subrequest('/foo', r.variables.args, done);
    r.subrequest('/bar', r.variables.args, done);
}
```

Subrequest 根据HTTP request payload进行流量分发

nginx.conf

```
server {
        listen 80;

        location / {
           js_content payload;
        }

# 使用此方法移除用于判断的url前缀
        location = /id1/ {
            proxy_pass http://backend1/;
        }
        location = /id2/ {
            proxy_pass http://backend2/;
        }
}
```

Note： requestBody只在js_content event 中可以被调用

fastresponse.js

```
function payload(r) {
    var  payload, json, url;

    payload = r.requestBody;
    json = JSON.parse(payload);

    if (json.id == "1") {
        url = "/id1"+r.uri
        r.subrequest(url, { method: 'POST' },
          function(res) {
            r.return(res.status, res.responseBody);
        });
     }

     if (json.id == "2") {
        url = "/id2"+r.uri
        r.subrequest(url, { method: 'POST' },
          function(res) {
            r.return(res.status, res.responseBody);
        });
     }
}
```

NJS 相关函数和对象支持

- Object, Array, Number, String，Date， Regexp， Function
- JSON， Math
- 异常处理， throw/catch
- 加密，文件处理

https://nginx.org/en/docs/njs/reference.htm

NJS 命令行测试工具

docker run -i -t nginx:latest /usr/bin/njs

```
[root@bogon conf.d]# docker run -i -t nginx:latest /usr/bin/njs
WARNING: IPv4 forwarding is disabled. Networking will not work.
interactive njs 0.3.6

v.<Tab> -> the properties and prototype methods of v.
type console.help() for more information

>> function hi(msg) {console.log(msg)}
undefined
>> hi("Hello world")
Hello world
undefined
>>
```

关注我们

F5 官方微信
（新闻，技术文章）

F5社区
（答疑，吐槽，分享，互动）

加入F5社区：关注“F5社区”微信公众号，注册成为社员，随时参加meet up活动，代码共享，讨论，答疑等。只要你有想法，有创造，那么就快来大展身手吧，让我们在社区里尽情分享，交流，吐槽和互动，在这个自由的国度里，发现闪亮的自己。让我们一起来见证“一群有才能的人在一起做有梦想的事！

NGINX技术群

操作步骤：

1. 扫描二维码并在“入群信息”栏填写姓名
2. 点击下方“我要入群”
3. 长按识别二维码进入群聊


f5
N
Thank You