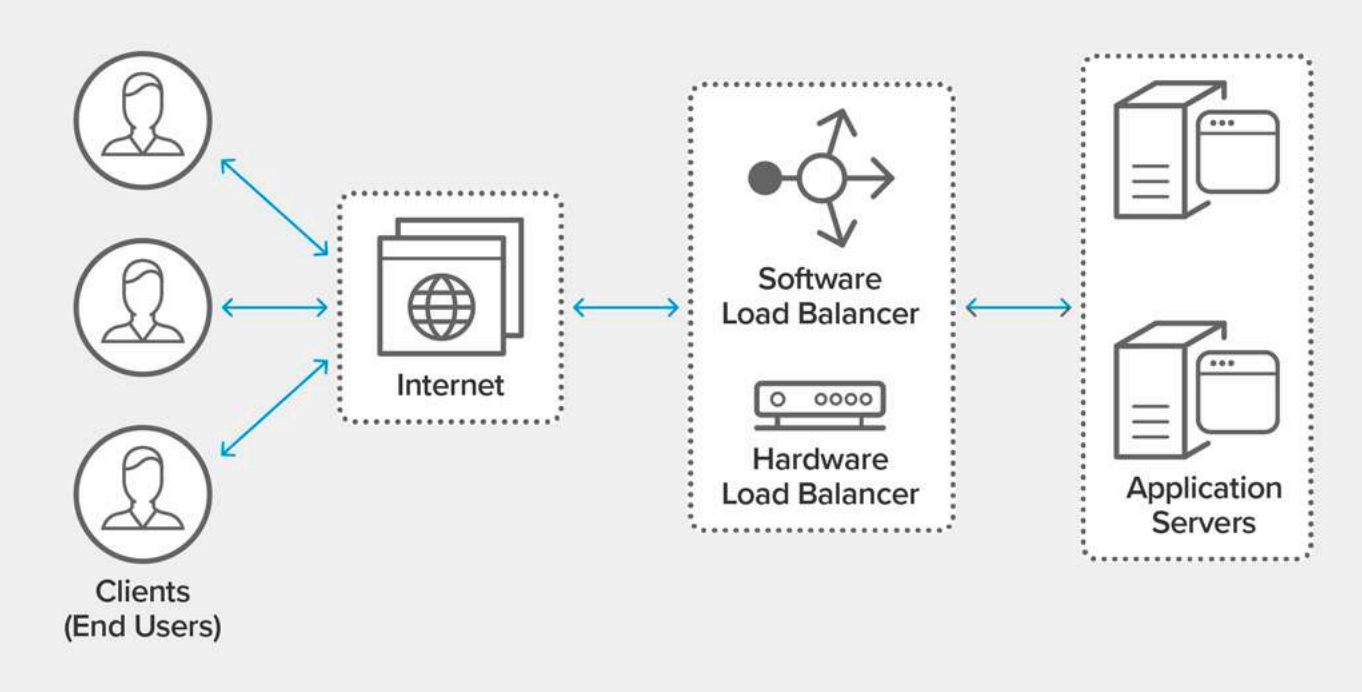
NGINX从入门到精通：
负载均衡以及安全访问控制

presenter :林健 Sr System Engineer

NGINX 负载均衡概述



•在多台服务器之间有效地分配客户端请求或网络负载

•通过仅向正常服务器发送请求来确保高可用性和可靠性

•业务不中断的情况下，按需弹性分配服务器资源

- 减少宕机时间
- 系统高可用
- 动态扩展性能
- 加快访问速度

基本配置

```
http {
    upstream myproject {
        server 10.10.10.1:8080;
        server 10.10.10.2:8080;
        server 10.10.10.3:8080;
        server 10.10.10.4:8080;
    }
    server {
        listen 80;
        server_name www.domain.com;
        location / {
            proxy_pass http://myproject;
        }
}
}
```

后端目标服务器

前端接受流量入口

*setsebool -P httpd_can_network_connect 1

流量转发中的行为

报文头修改行为

•删除所有的空值报文头，空值是没有意义的，不需要转发到后台服务器

•包含下划线"_"的报文头将会被静默丢弃，如需要开启可以使用指令underscores_in_headers on
这样做是为了避免在将header射到CGI变量时混淆，因为在此过程中，破折号和下划线都映射到了下划线

•Host报文头会被重写为$proxy_host变量值，该变量值为proxy_pass指令中所包含的域名或ip地址

•Conntection报文头将会被修改为close，HTTP 1.1将会被修改为HTTP 1.0，该行为会使得收到服务器端的响应后即刻关闭，服务器侧的连接不会被保持，HTTP 1.1变量可以用proxy_http_version指定

代理请求request报文头修改指令：

```
Syntax:
proxy_set_header field value;
Default:proxy_set_header Host $proxy_host;proxy_set_header Connection close;
Context: http, server, location
```

Host报文头对应的变量

Header	Description
$proxy_host	该变量的值是从proxy_paas 指令后的url获取，为替换host 请求报文头的默认值
$http_host	$http_变量群中的一个，$http_后跟小写的报文头名称，所有短划线均由下划线代替。当客户端请求没有有效的“Host”报文头时，这可能会获取失败。
$host	该如果原始请求中包含Host，就是请求中host除端口部分，如果没有，就是server_name

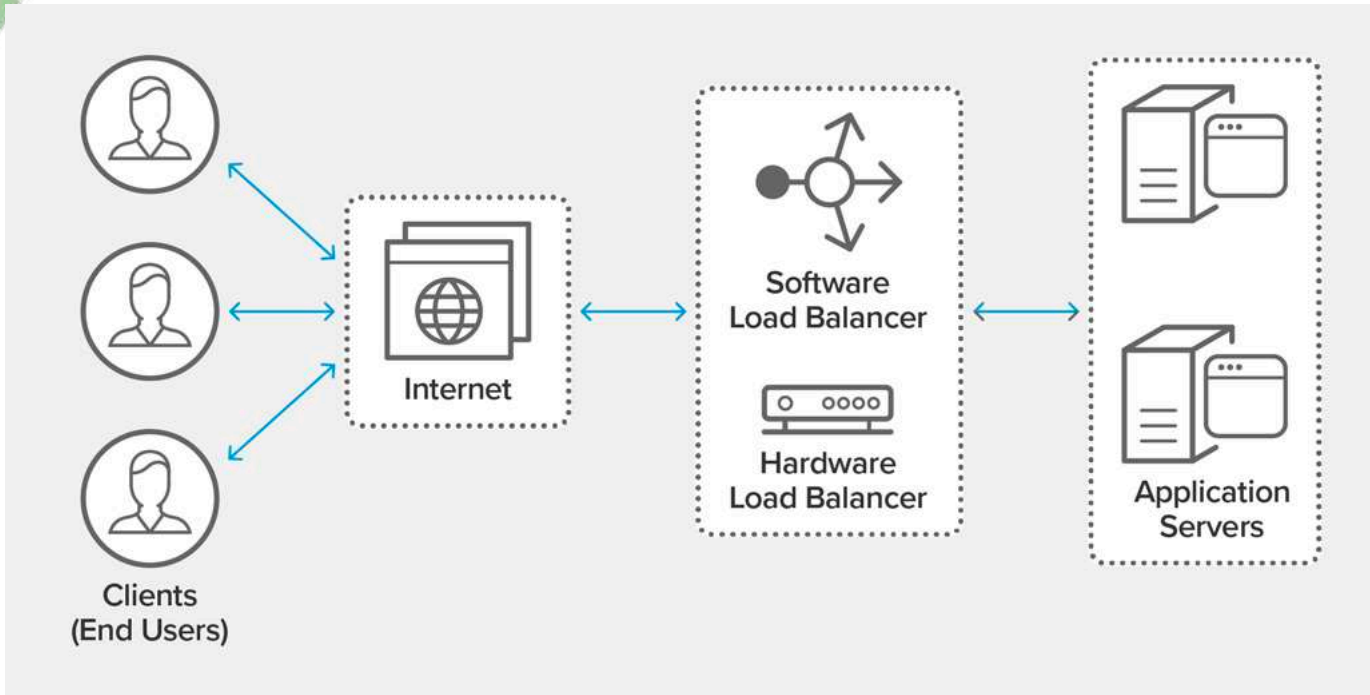
服务器获取真实客户端IP

```
location /match/here {
    proxy_set_header HOST $host;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://example.com;
}
```

Header	Description
X-Forwarded-Proto	所使用的传输协议 http/https
X-Real-IP	使用$remote_addr的值，该值为客户端IP地址
X-Forwarded-For	该报文头是一个list，如该请求被多次代理转发，在每次的转发中，每一级转发服务器都可以将自己的IP地址添加到这个list， $proxy_add_x_forwarded_for 这个变量继承来原来报文头中的X-Forwarded-For 报文内容，并且把nginx自己的ip地址添加在最后

proxy_add_header 可以在http上下文，server下上文，location上下文添加，以统一代理转发行为

Proxy中buffer行为



代理到另一台服务器时，两个不同连接的速度将影响客户端的体验

- 从客户端到Nginx的连接。
- 从Nginx代理到后端服务器的连接。

默认Nginx会开启缓冲机制，启用缓冲后，nginx会尽快从代理服务器收到响应，并将其保存到proxy_buffer_size和proxy_buffers指令设置的缓冲区中。如果整个响应都无法容纳到内存中，则可以将一部分响应保存到磁盘上的临时文件中。写入临时文件由proxy_max_temp_file_size和proxy_temp_file_write_size指令控制。

禁用缓冲后，响应一收到就立即同步传递到客户端。 nginx不会尝试从代理服务器读取整个响应。nginx一次可以从服务器接收的最大数据大小由proxy_buffer_size指令设置。 也可以通过在“ X-Accel-Buffering”响应头字段中传递“ yes”或“ no”来启用或禁用缓冲。可以使用proxy_ignore_headers指令禁用此功能。 。

客户端->Nginx

带宽100Mbps/30ms延迟

Nginx->服务器

带宽万兆/1ms延迟

*请注意在整体buffer行为中，还包含操作系统的TCP协议buffer

Proxy buffer相关参数

Header	Description
proxy_buffering	是否开启buffer，默认为开启，如关闭，服务器发送 的数据会被立即发送给客户端
proxy_buffers	控制一个TCP连接对后端响应的缓冲区的数量（第一个参数）和大小（第二个参数）。默认设置是配置8个缓冲区，每个缓冲区的大小等于一个内存页面（4k或8k），默认来说buffer size等于一个内存分页大小，由操作系统平台指定,
proxy_buffer_size	来自后端服务器的响应的初始部分与其余响应分开缓冲。该指令为响应的这一部分设置缓冲区的大小。默认情况下，它的大小与proxy_buffers相同，但是由于它用于header信息，因此通常可以将其设置为较低的值
proxy_busy_buffers_size	该指令设置可以标记为“客户端就绪”并因此繁忙的缓冲区的最大大小。客户端一次只能从一个缓冲区读取数据，而缓冲区却被放入队列中，以成束的形式发送给客户端。此伪指令控制允许处于此状态的缓冲区空间的大小。
proxy_max_temp_file_size	这是每个请求的磁盘上临时文件的最大大小。它们是在上游响应太大而无法放入缓冲区时创建的 准备就绪”，因此很忙。客户端一次只能从一个缓冲区读取数据，而缓冲区却被放入队列中，以成束的形式发送给客户端。此伪指令控制允许处于此状态的缓冲区空间的大小。

Question

如果nginx收到的完整的response，但是客户端此时异常中断，在服务器上会不会标示这笔response对应的业务是完整的，但是客户端并没有收到。

```
var net = require('net');

var client = new net.Socket();
client.connect(80, '10.128.4.209', function() {
    console.log('Connected');
    client.write('GET /test HTTP/1.1\r\nUser-Agent: curl/7.35.0\r\n \
    Host: 10.128.4.209\r\nAccept: */*\r\n\r\n');
});

client.on('data', function(data) {
    console.log('Received: ' + data);
    client.destroy(); });

client.on('close', function() {
    console.log('Connection closed');
});
```

```
var http = require('http');
var fs = require('fs');

http.createServer(function (req, res) {

                res.writeHead(200, {'Content-Type': 'text/plain'});
                var i=0;
                for(; i < 500; i++)
                {
                 res.write('this is '+i+'\r\n');
                };
                res.end('this is end of response\r\n')

              ;


}).listen(8080, '0.0.0.0');

console.log('Server running at 8080/');
```

proxy_ignore_client_abort

Defaule value: off

With buffer off:

Address A	Port A	Address B	Port B	Packets	Bytes
10.128.4.209	51604	10.128.4.129	8080	16	1558
10.128.4.217	54659	10.128.4.209	80	12	1246

With buffer on:

Address A	Port A	Address B	Port B	Packets	Bytes
10.128.4.217	54662	10.128.4.209	80	10	4906
10.128.4.209	51610	10.128.4.129	8080	44	9590

其他比较重要的参数

Header	Description
client_max_body_size	设置客户端请求正文的最大允许大小，在“ Content-Length”请求字段中指定。如果请求中的大小超过配置的值，则会向客户端返回413（请求实体太大）错误。请注意，浏览器无法正确显示此错误。将size设置为0将禁用对客户端请求主体大小的检查。Default 1M
client_body_buffer_size	设置用于读取客户端请求正文的缓冲区大小。如果请求主体大于缓冲区，则将整个主体或仅将其一部分写入临时文件。默认情况下，缓冲区大小等于两个内存页。在x86，其他32位平台和x86-64上为8K。在其他64位平台上，通常为16K。
proxy_connect_timeout	定义用于与代理服务器建立连接的超时。请注意，此超时通常不能超过75秒
proxy_send_timeout	设置将请求发送到后端服务器的超时。超时仅在两个连续的写操作之间设置，而不用于整个请求的传输。如果后端服务器在此时间内未收到任何信息，则连接将关闭。
proxy_read_timeout	从后端服务器读取响应超时。超时仅在两次连续的读取操作之间设置，而不用于传输整个响应。如果后端服务器在此时间内未传输任何内容，则连接将关闭。

Proxy cache行为

在有proxy cache的配置的情况下，客户端请求不会被转发到后端服务器，而是直接做在nginx上使用本地cache返回：

该行为取决于

1.Nginx是否配置proxy_cache，默认情况下proxy_pass不会做cache
2.默认后端服务器response有cache-control的情况下，会cache

在下列情况下是不会做cache的

1.Cache-Control 为 Private, No-Cache, No-Store
2.响应请求中有Set cookie
3.非GET/HEAD请求

追踪cache: add_header X-Cache-Status $upstream_cache_status;

https://www.nginx.com/blog/nginx-caching-guide/

Proxy_cache指令

```
proxy_cache_path /path/to/cache keys_zone=my_cache:10m levels=1:2  max_size=10g inactive=60m
use_temp_path=off;

server
 { # ...
     location / {
        proxy_cache my_cache;
        proxy_pass http://my_upstream;
        }
}
```

Header	Description
/path/to/cache	Cache对象实际存储路径
keys_zone	存储key的用的share memory,1M可以储存8000个cache条目
levels	所有文件的都在一个目录里会影响性能，使用该参数可以指定使用二级目录，大部分场景都会指定二级目录
max_size	cache的最大容量
inactive	在cache的内容多长时间没有被访问还能存在，nginx不会根据cache control字段定义删除cache，只会根据本参数执行。默认10分钟，超过该时间后nginx会删除cache路径中的缓存对象

Cache 行为Tips

在cache路径下会很多cache对象条目，条目内容包含实际的返回内容，条目名称默认的生成模式为：`Md5($scheme$proxy_host$request_uri)`

```
#!/usr/bin/perl
 use strict;
 use Digest::MD5;
 my $md5 = Digest::MD5->new;
 $md5->add($ARGV[0]);
 my $digest = $md5->hexdigest;
 print "\nDigest is $digest\n";
```

用perl脚本生成nginx一样的cache key，Linux md5sum无法生成，注意$proxy_host Proxy_pass中upstream的名字，不带端口，如：

http://myupstream/js/test.js

使用perl和nginx生成的cache key都是

ea00f1145777995e472141772a339dfa

可以用指令：

```
proxy_cache_key $proxy_host$request_uri$cookie_jessionid;
```

重新定义生成cache key,配合proxy_cache_methods，可以cache动态结果

Request Tracing

经过Nginx负载均衡后，如何把前后请求关联起来，NGINX Plus R10（和NGINX 1.11.0）引入了$ request_id变量，该变量是随机生成的32个十六进制字符的字符串，该字符串会在每个HTTP请求到达时自动分配给它（例如444535f9378a3dfa1b8604bc9e05a303），通过配置NGINX Plus和所有后端服务以传递$ request_id值，可以端对端跟踪每个请求。

```
$request_id

  unique request identifier generated from 16 random bytes, in hexadecimal (1.11.0)
```

用法

proxy_add_header：服务器侧请求跟踪
Add_header：在客户端侧做测试
Log format ：做前后端请求关联，排障

```
log_format trace '$remote_addr:$remote_port -
$remote_user [$time_local]'
                 ' "$request"  $status
$body_bytes_sent "$http_referer" '
                 ' "$http_user_agent"
"$http_x_forwarded_for" $request_id';
```

sample log

```
10.128.4.209 :35706 - - [26/Feb/2020:11:14:59 -0500] "GET / HTTP/1.1" 200 6432 "-" "curl/7.29.0" "-"
bc641677e3f5d5e9a40c270cc1aba261
```

Tracing config sample

```
log_format trace '$remote_addr:$remote_port $upstream_addr- $remote_user [$time_local] "$request" '
                 '$status $body_bytes_sent "$http_referer" "$http_user_agent" '
                 '"$http_x_forwarded_for" $request_id';
```

```
server {
       access_log  /var/log/nginx/access.log  trace;
       listen 80 ;
       proxy_set_header HOST $host;
       proxy_set_header X-Forwarded-Proto $scheme;
       proxy_set_header X-Real-IP $remote_addr;
       add_header X-Request-ID $request_id;
       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
       proxy_set_header X-Request-ID $request_id;
      location / {
        proxy_buffering on;
        proxy_pass http://myupstream;

       }
```

如果后端也是nginx，直接使用$request_id是本机生成id，可以用“$http_x_request_id” 变量获取上级id

负载均衡

一个负载均衡的例子

```
upstream myproject {
    ip_hash;
    server 127.0.0.1:8000 weight=3 max_fail=10 fail_timeout=30s max_conns=300;
    server 127.0.0.1:8001 weight=3 max_fail=10 fail_timeout=30 slow_start;
    server 127.0.0.1:8002 weight=3 max_fail=10 fail_timeout=30 backup;
    server 127.0.0.1:8003 weight=3 max_fail=10 fail_timeout=30 down;
  }

Proxy_pass http://myproject
```

分发参数

Parameter	Description
默认	Round Robin，依照轮询算法分发
weight	默认为1，按照比例分法请求
ip_hash	对客户端IP取Hash，根据hash值分发
least_conn	当前连接最小的服务器优先
least_time（Nginx plus专属）	当前响应最快的优先
Random（Nginx plus专属）	随机选择

获取后端服务器健康信息

Parameter	Description
max_fail	设置在fail_timeout参数设置的持续时间内发生的与服务器通信的不成功尝试次数，如达到该值以认为服务器在fail_timeout参数设置的持续时间内不可用。 默认为1。
fail_timeout	• 判断的服务器是否可用的周期，在该周期内如有max_fail次无法正常通信，则认为该服务器不可用 • 服务器在判断为不可用后，持续标示为不可用的时间，期间不会发送任何请求到该服务器 默认值为10

proxy_next_upstream指定什么什么样的response被认为是不成功：

error

与服务器建立连接，向服务器传递请求或读取响应头时发生错误

timeout

与服务器建立连接，向服务器传递请求或读取响应header时发生超时

一些其他行为控制

Parameter	Description
backup	当非backup服务器出现不可用，则启用由该指令标识的服务器，平时服务器不分配流量
slow_start	如不可用服务器变为可用，在该参数所指定的时间内，起weight值从0增长为正常weight值，该参数无法和hash, ip_hash,random共用
max_conn	服务器的最大连接数，如没有配置zone，为每个worker独立计算，如配置zone，为整体最大连接限制

Tips：

- slow start在只有一个upstream的时候，该参数被忽略，同时该server也不会被mark down
- Max conn需要注意zone的配置，否则整体限制数为worker数量乘该值
- 如果server达到max conn，可以使用queue指令设定队列(nginx plus)

连接限制 limit_req

访问速率限制有助于防范DDOS攻击以及业务秒杀活动中的薅羊毛行为，Nginx中可以应用limit_rate实现，在nginx中使用leak bucket算法实现，你需要配置

- key - 用于区分一个客户与另一个客户的参数，通常是一个变量
- shared memory zone -保持这些key状态的区域的名称和大小（“leak bucket”）
- rate - 以每秒请求数（r / s）或每分钟请求数（r / m）指定的请求速率限制。每分钟请求数用于指定小于每秒一个请求的速率

```
http { #...
  limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
   server { #...
       location /search/ {
                   limit_req zone=one;
                        }
          }
    }
```

超过速率的的客户端请求NGINX将响应503服务不可用错误，错误返回可以通过limit_req_status指定

Active health check（NGINX PLUS）

```
http {
    server {
    ...
        location / {
            proxy_pass http://backend;
            health_check match=welcome;
        }
    }

    match welcome {
        status 200;
        header Content-Type = text/html;
        body ~ "Welcome to nginx!";
    }
}
```

该模块可以设定

- 标示为down的检测次数
- 标示为up的检测次数
- 检查端口
- 正常响应的状态码
- 正常响应的报文头
- 报文头正则
- 报文体正则

http://nginx.org/en/docs/http/ngx_http_upstream_hc_module.html

Stick 会话保持（NGINX PLUS）

```
upstream backend {
     server backend1.example.com;
     server backend2.example.com;
     sticky cookie srv_id expires=1h domain=.example.com path=/;
     }
```

将会话cookie添加到upstream的第一个响应中，并标识发送响应的服务器。客户的下一个请求包含cookie值，NGINX Plus将请求路由到原服务器

```
map $cookie_jsessionid $route_cookie { ~.+\.(?P<route>\w+)$ $route; }
map $request_uri $route_uri { ~jsessionid=.+\.(?P<route>\w+)$ $route; }
upstream backend {
    server backend1.example.com route=a;
    server backend2.example.com route=b;
    sticky route $route_cookie $route_uri;
}
```

当客户端接收到第一个请求时，为其分配“route”。将所有后续请求与server指令的route参数进行比较，以标识将请求代理到的服务器。路由信息来自Cookie或请求URI。

Stick 会话保持（NGINX PLUS）

```
upstream backend {
   server backend1.example.com;
   server backend2.example.com;
   sticky learn
       create=$upstream_cookie_examplecookie
       lookup=$cookie_examplecookie
       zone=client_sessions:1m
       timeout=1h;
}
```

首先通过检查响应找到会话标识符。然后，NGINX Plus“学习”哪个upstream 服务器对应于哪个会话标识符。通常，这些标识符在HTTP cookie中传递。如果一个请求包含一个已经“学习”的会话标识符，则NGINX Plus将请求转发到相应的服务器,该功能需要使用shared memory，1M大约支持4000会话。

根据客户端IP地址分流

```
http {
    geo $remote_addr $geo {
     default 0;
     10.128.4.214/32 1;
     10.128.4.217/32 2;
  }
    ......
   server {
       if ( $geo = 1 ) {
        return 200 "this is for 214";
      }
if ( $geo = 2 ) {
        return 200 "this is for 217";
      }
 }
}
```

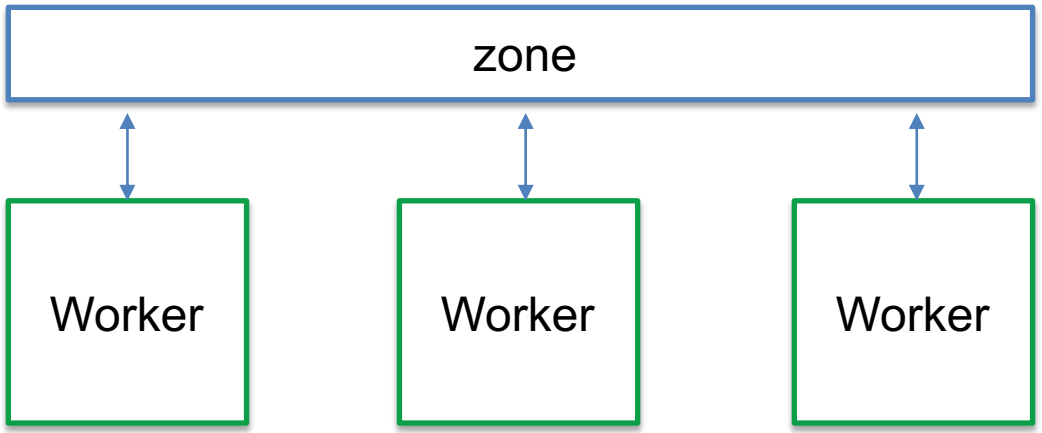
在使用nginx做负载均衡的过程中，可能会出现根据IP地址段进行分流的需求，某些网段客户端去服务器组A，某些网段去服务器组B，此时需要使用Geo指令，该指令包含在：

- ngx_http_geo_module
- ngx_stream_geo_module

上述两个模块，可以在HTTP/TCP/UDP协议负载均衡中根据IP地址进行分流。

该指令用法和MAP类似。

多worker协同：zone

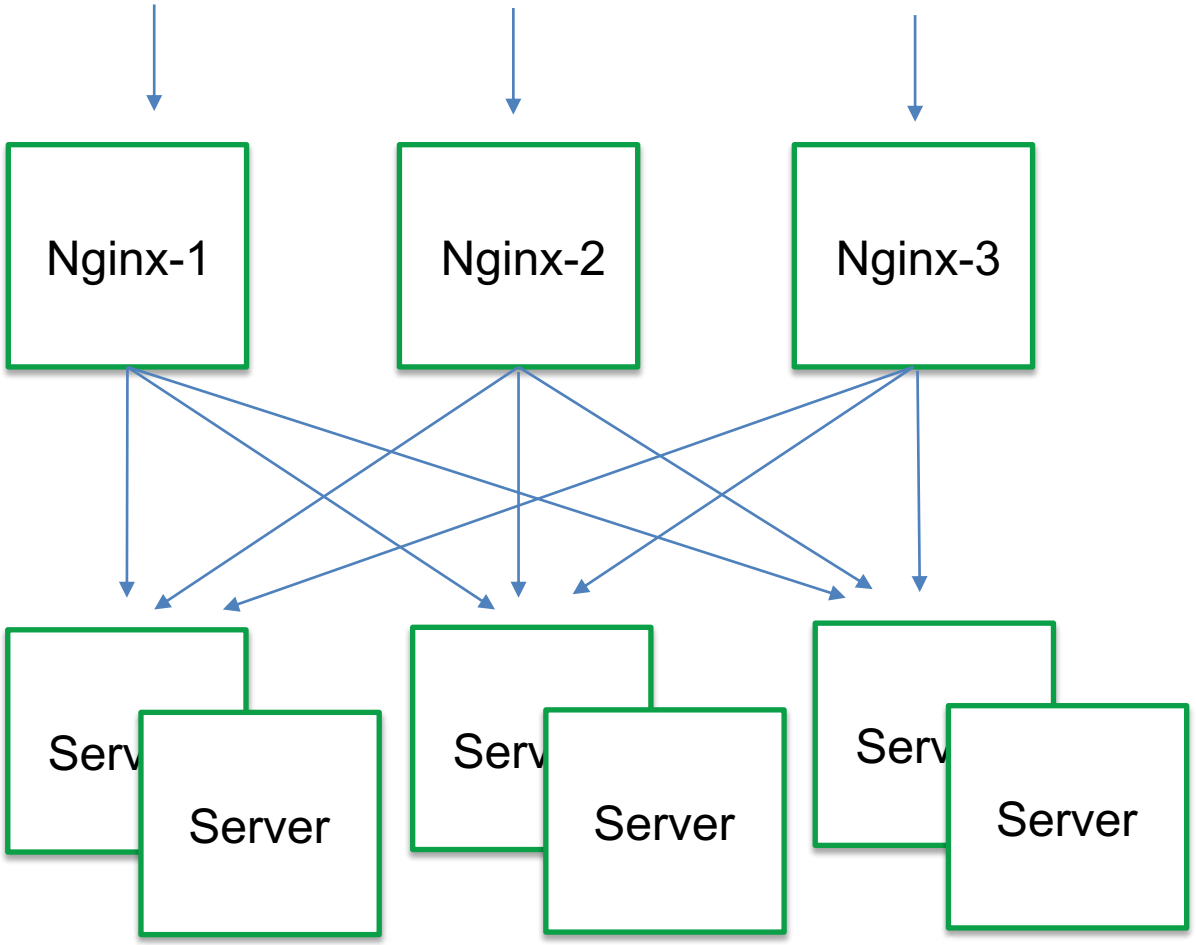


如果upstream不包含zone指令，则每个工作进程将保留其自己的服务器组配置副本，并维护其自己的一组相关计数器。计数器包括与组中每个服务器的当前连接数以及将请求传递给服务器的尝试失败次数，zone对主动监控检查和API动态更新upstream是必须的。另外：

- max_fail
- max_conn
- least_conn

在没有zone的情况下，都是per worker独立计数，在worker-1发现server-1 down，worker-2并不会共享这个信息，而是可能继续发送request，同理，max_conn,least_conn也是per worker独立计数

多实例协同：zone_sync(NGINX PLUS)

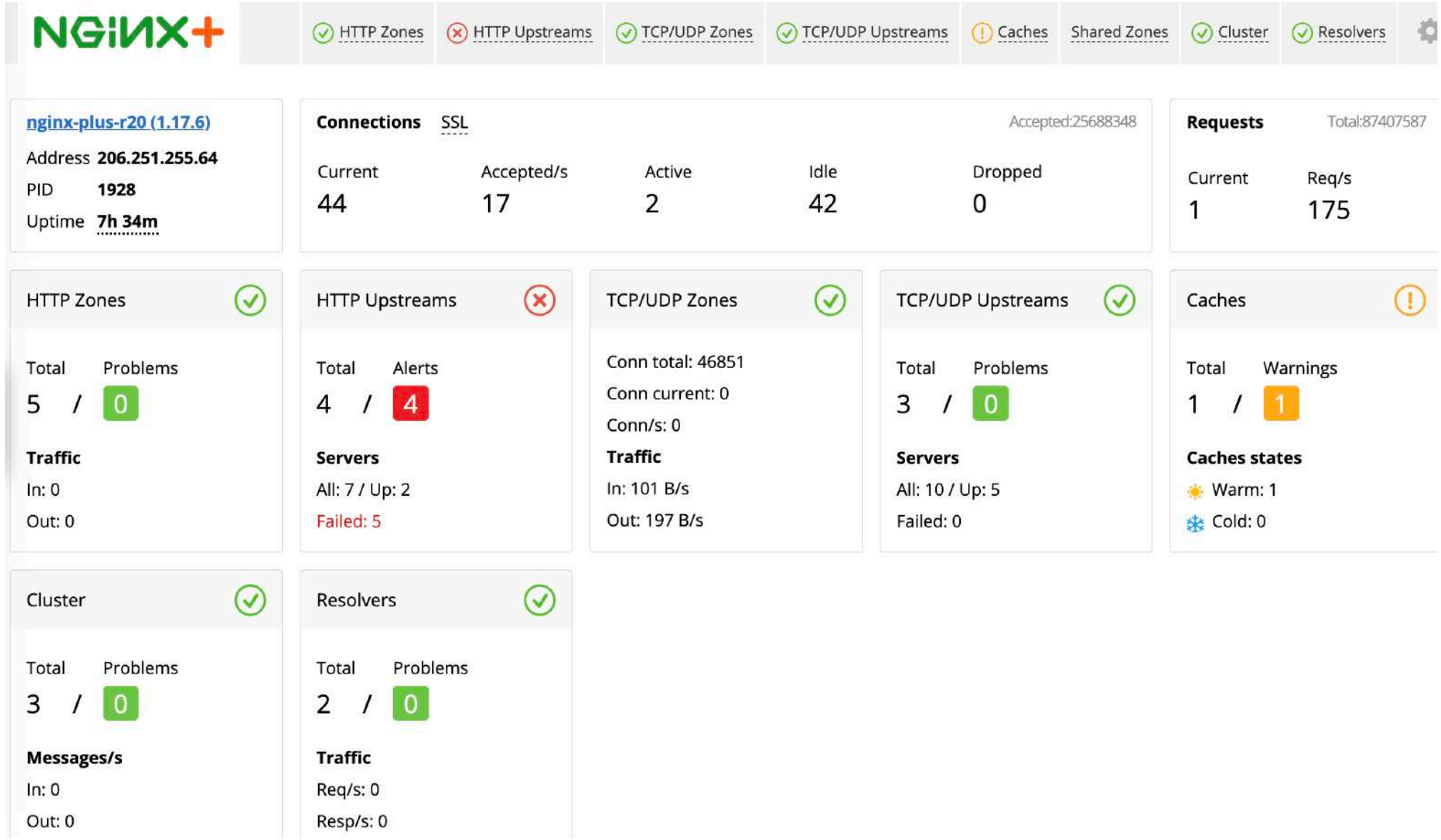


多个nginx实例同时对一组服务器执行负载均衡，需要配置某台服务器的连接上限为1000并发连接，此时可以使用zone_sync模块，实现多个nginx实例共享连接数据，实现后端服务器的一致性行为管理，可以用于：

- sticky learn
- requests limiting
- key-value storage

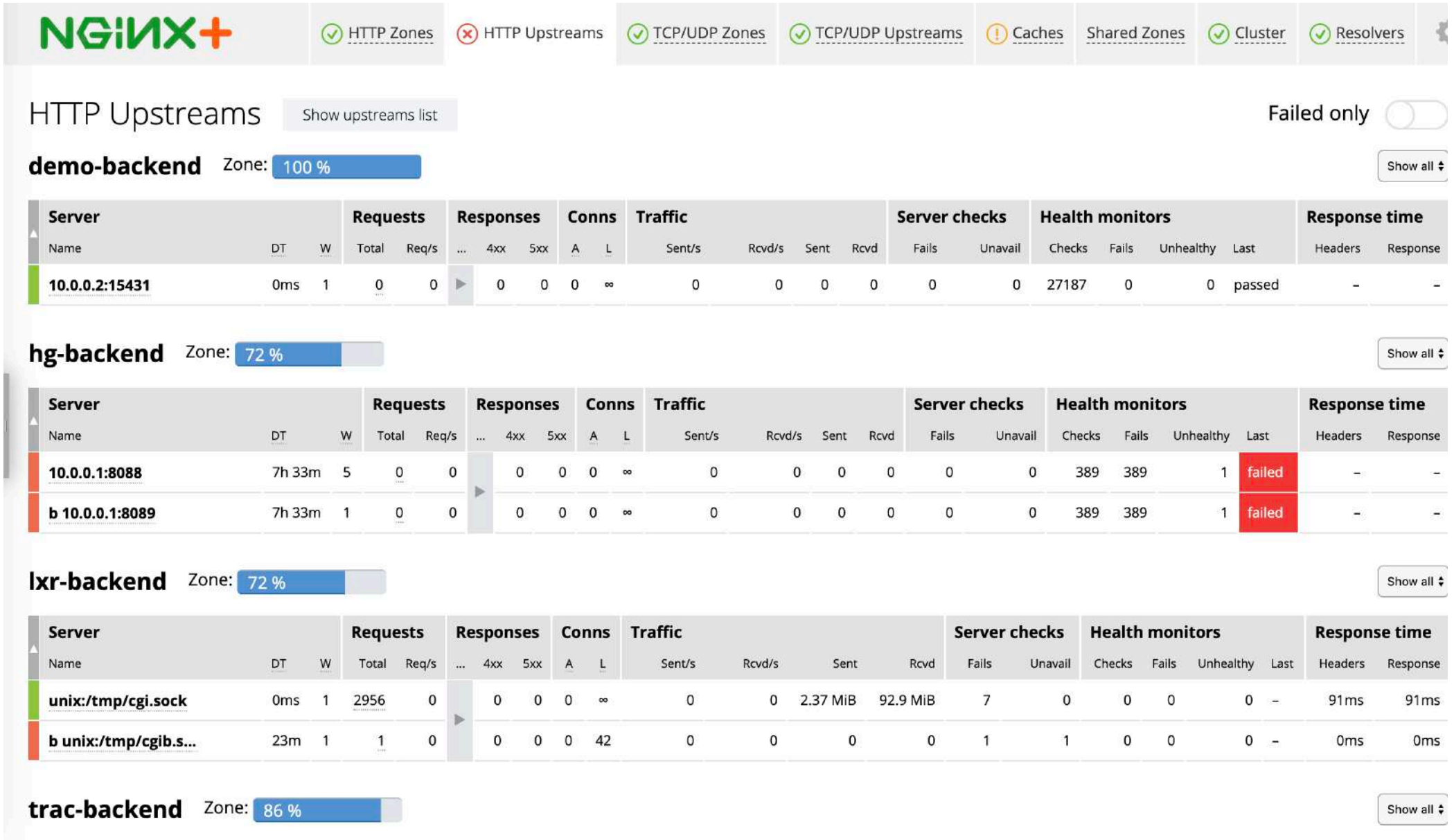
管理开启Zone sync后，nginx之间可使用证书认证彼此，确保通信安全，同步状态可以通过API监控.

Nginx Plus dashboard



NGINX

Nginx Plus dashboard



Nginx Plus dashboard

NGINX+ | HTTP Zones | HTTP Upstreams | TCP/UDP Zones | TCP/UDP Upstreams | Caches | Shared Zones | Cluster | Resolvers

Shared Zones

Zone	Total memory pages	Used memory pages	Memory usage
SSL	12725	1274	1 %
ban	31	2	7 %
demo-backend	7	7	100 %
dns_backends	7	5	72 %
dynamic	7	5	72 %
fo	254	2	1 %
hg-backend	7	5	72 %
http_cache	4071	2	1 %
lxr-backend	7	5	72 %
nginx_plus_versions	31	5	17 %
nginxorg	15	6	40 %
ngx_zone_sync_shared	7	2	29 %
one	254	2	1 %
trac-backend	7	6	86 %

NGINX

安全访问控制

访问控制/IP ACL

基本IP访问控制

```
location / {
    satisfy any;
    allow 192.168.1.0/24;
    deny  all;
}
```

基于报文头用户名密码的访问控制

```
location / {
    auth_basic “Administrator’s Area”;
    auth_basic_user_file conf/htpasswd;
}
```

使用apache2-utils (debian, ubuntu) 或者 httpd-tools（redhat/centos）对用户密码进行加密

```
user1:$apr1$/woC1jnP$KAh0SsVn5qeSMjTtn0E9Q0
user2:$apr1$QdR8fNLT$vbCEEzDj7LyqCMyNpSoBh/
user3:$apr1$Mr5A0e.U$0j39Hp5FfxRkneklXaMrr/
```

JWT认证

JWT（JSON WEB Token）是OpenID Connect标准中用于用户信息的数据格式，该标准是OAuth 2.0协议之上的标准标识层。 API和微服务的部署者也因为其简单性和灵活性而转向JWT标准。通过JWT身份验证，客户端发送JSON Web令牌到Nginx，Nginx根据本地密钥文件或远程服务来验证令牌。

Token包含三部分:报文头,报文体，签名，签名所用的加密key存放在nginx上或者外部，nginx使用签名加密key验证token的合法性，并根据token报文体中的字段判断如何处理该请求。Token明文部分包含本Token使用的签名算法，签发Token，使用本Token的机构，超时时间等相关信息.

JWT认证:本地Key认证

```
server {
    listen 80;

    location /products/ {
        proxy_pass          http://api_server;
        auth_jwt            "API" token=$arg_apijwt;
        auth_jwt_key_file conf/key.jwk;
    }
}
```

- auth_jwt：指定的realm以及获取token的变量头
- auth_jwt_key_file:本地存放key的位置

JWT认证:外部Key/cache

```
proxy_cache_path /var/cache/nginx/jwk levels=1 keys_zone=jwk:1m max_size=10m;

server {
   location / {
        auth_jwt "site";
        auth_jwt_key_request /_jwks_uri;
        proxy_pass http://my_backend;
    }
    location = /_jwks_uri {
        internal;
        proxy_method GET;
        proxy_cache  jwk; # Cache responses
        proxy_pass   https://idp.example.com/oauth2/keys;
    }
}
```

从外部网址获取外部key并cache，此时一般用非对称加密模式key，获取到的是签名token用的密钥对中的公key

如何使用Token信息

```
map $jwt_claim_sub $jwt_status {
    "quotes" "revoked";
    "test"   "revoked";
    default  "";
}
server {
    listen 80;

    location /products/ {
        auth_jwt "Products API";
        auth_jwt_key_file conf/api_secret.jwk;

        if ( $jwt_status = "revoked" ) {
            return 403;
        }
        proxy_pass http://api_server;
    }
}
```

通过使用jwt变量群，
获取token中的相关信息，
比如谁签发本token，谁
使用本token，超期时间为和
等等，根据相关信息，使用map
指令做选择性转发request

JWT的一些思考

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

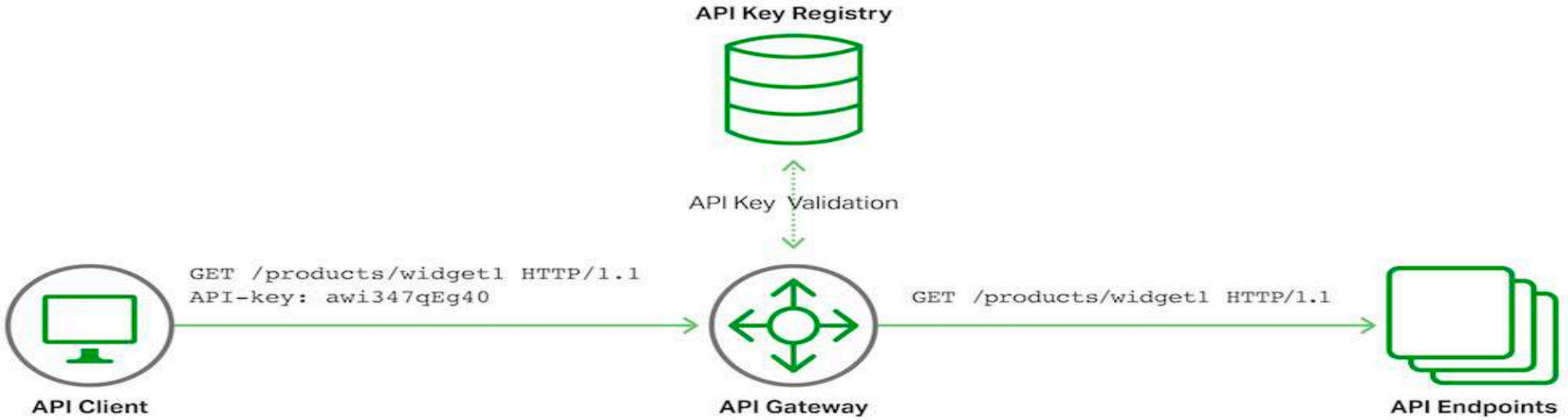
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3
ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.1K
r-vfOtzV6MigmETkt6_O73DQ5zBKYW4LLhRxNuioo
```

SIGNATURE

用户：Token好长，怎么和我在AWS，Google上用的API key差这么多，他们的好短
运维A：每次要撤销Token 要调整nginx 里的map，感觉nginx的配置越来越大了
运维B： Token信息和URI要匹配，感觉是把资源/用户的信赖映射要写在nginx配置里了，怎么运维？

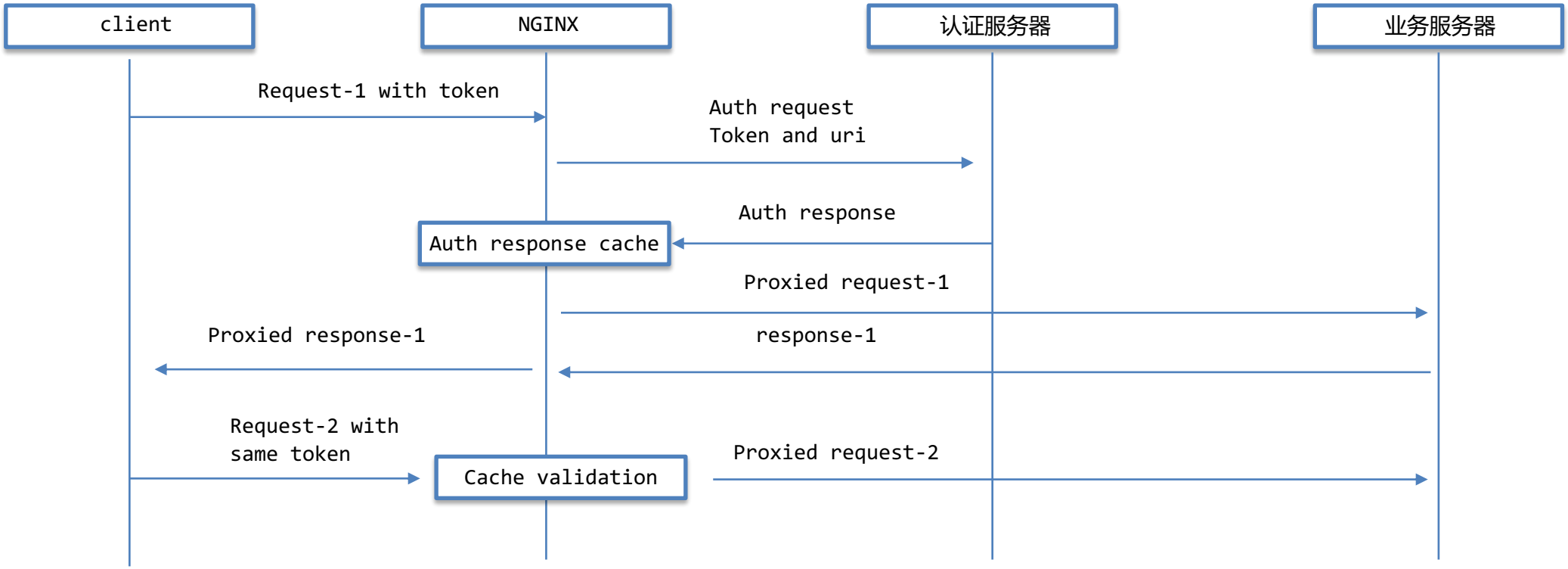
Subrequest纯外部认证



NGINX和NGINX Plus可以使用外部服务器或服务验证对网站的每个请求。为了执行身份验证，NGINX向外部服务器发出HTTP子请求，如果子请求返回2xx响应代码，则允许访问，如果子请求返回401或403，则拒绝访问。这种身份验证类型允许实现各种身份验证方案，例如多因子身份验证，或者允许实现LDAP或OAuth身份验证，在零信任模型下，该模式可以配置为所有应用的统一入口，后端服务器只允许nginx代理请求。不接受其他任何其他客户端直接请求。

Subrequest实现认证例子

- 每个请求包含一个 Authorization报文头，包含预先申请的Token
- Nginx作为Oauth2模型中的资源服务器代理，提取出Token，向认证服务器发出Token验证请求，请求中包含URL和Token信息组合
- Nginx对每个uri和token的认证结果进行cache，cache时间为5分钟，5分钟内同样的url+token组合无需再次发起认证请求



配置细节

```
 location / {
      auth_request /auth;
      auth_request_set $auth_status $upstream_status;
      proxy_pass http://myapp;
}
 location /auth {
       internal;
       proxy_cache_key $http_authorization$request_uri;
       proxy_cache_valid 401 403 2m;
       proxy_cache mycache;
       proxy_pass_request_body off;
       proxy_set_header        Content-Length "";
       proxy_set_header        X-TOKEN $http_authorization;
       proxy_set_header        X-Original-URI $request_uri;
       proxy_pass              http://10.128.4.214/auth;
}
```

- 主请求和子请求变量透传
- 用token+实际访问uri做认证
- 认证结果做cache，cache时间由认证服务器控制，超过10分钟不访问则会删除
- 负面认证结果cache 2分钟

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

Thank you