16

TCP/UDP Load Balancing with NGINX: Overview, Tips, and Tricks

Konstantin Pavlov, NGINX Inc.

TCP load balancing

```
stream {
  upstream mysql_backends {
    server backend1.example.com:3306;
    server backend2.example.com:3306;
  }
  server {
    listen 3306;
    proxy_pass mysql_backends;
  }
}
```

UDP load balancing

```
stream {
  upstream named_backends {
    server named1.example.com:53;
    server named2.example.com:53;
  }
  server {
    listen 53 udp;
    proxy_pass named_backends;
    proxy_responses 1;
    error_log logs/dns.log;
  }
}
```

TCP/UDP load balancer tuning

Supported load-balancing methods:

- weighted round-robin
- hashed key
- least number of connections

```
upstream backends-imaps {
  hash $remote_addr consistent;
  server 192.168.1.1:993;
  server 192.168.1.2:993;
}
```

Additionally available in NGINX Plus:

- least time to connect
- least time to receive first byte
- least time to receive last byte

```
upstream backends-smtp {
  least_time connect;
  server 192.168.1.1:25;
  server 192.168.1.2:25;
}
```

TCP/UDP load balancer tuning #2

```
upstream backends-ldap {
  server 192.168.1.1:389 weight=3 max_fails=2 fail_timeout=30;
  server 192.168.1.2:389 weight=1 max_fails=2 fail_timeout=30;
  server 192.168.1.3:389 down;
  server 192.168.1.4:389 backup;
}
upstream backends-pop3 {
  zone pop3-dynamic 64k;
  server 192.168.2.1:110 max_conns=20 slow_start=60s;
  server 192.168.2.2:110 max_conns=20 slow_start=60s;
}
upstream backends-webservers {
  zone web-dynamic 128k;
  server microservice1.example.com service=http resolve;
}
```

TCP/UDP active health checks

```
stream {
  upstream backend-imaps {
    zone dynamic 64k;
    server 192.168.1.1:993;
  }
  server {
    listen 993;
    proxy_pass backend-imaps;
    health_check interval=10 passes=2 fails=3 port=8080 match=http;
    health_check_timeout 5s;
  }
  match http {
    send "GET /status HTTP/1.0\r\nHost: imaps.example.com\r\n\r\n";
    expect ~* "200 OK";
  }
}
```

TCP/UDP active health checks #2

```
  upstream dns-backends {
    zone dynamic 64k;
    server 192.168.1.100:53;
  }
  server {
    listen 53;
    listen 53 udp;
    proxy_pass dns-backends;
    health_check match=dns_test udp;
  }
  match dns_test {
    send
"\xaa\x67\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x05\x6e\x67\x69\x6e\x78\x03\x6f\x72\x6
7\x00\x00\x01\x00\x01";  # A? nginx.org
    expect ~ "\xce\xfb\xff\x3f"; # 206.251.255.63
  }
```

Access control and limiting

```
stream {
  limit_conn_zone $binary_remote_addr zone=addr:10m;
  server {
    listen 5432;
    deny 192.168.1.2;
    allow 192.168.1.1/24;
    allow 2001:0db8::/32;
    deny all;
    limit_conn addr 1;
    proxy_download_rate 100k;
    proxy_upload_rate   50k;
    proxy_pass postgre.int.example.com:5432;
}
```

Passing client's IP to the backends

```
stream {
  server {
    listen 192.168.1.100:80;
    proxy_protocol on;
    proxy_pass 192.168.2.1:80;
  }
}
```

Passing client's IP to the backends #2

```
user root;

# requires additional OS-level configuration!
# https://www.kernel.org/doc/Documentation/networking/tproxy.txt
stream {
  server {
    listen 192.168.1.1:1234;
    proxy_bind $remote_addr transparent;
    proxy_pass 192.168.2.1:1234;
  }
}
```

TLS: termination

```
stream {
  server {
    listen 192.168.1.100:443 ssl;
    ssl_certificate /var/lib/acme/live/thre.sh/fullchain;
    ssl_certificate_key /var/lib/acme/live/thre.sh/privkey;
    proxy_ssl off;
    proxy_pass 192.168.1.101:80;
  }
}
```

TLS: re-encryption

```
stream {
  server {
    listen 192.168.1.100:443 ssl;
    ssl_certificate /var/lib/acme/live/thre.sh/fullchain;
    ssl_certificate_key /var/lib/acme/live/thre.sh/privkey;
    proxy_ssl on;
    proxy_ssl_verify on;
    proxy_ssl_trusted_certificate /etc/ssl/trusted.pem;
    proxy_pass 192.168.1.101:443;
  }
}
```

TLS: wrapping

```
stream {
  server {
    listen 192.168.1.100:80;
    proxy_ssl on;
    proxy_ssl_verify on;
    proxy_ssl_trusted_certificate /etc/ssl/trusted.pem;
    proxy_pass 192.168.1.101:443;
  }
}
```

TCP/UDP load balancer: logging

```
2016/08/21 18:39:04 [info] 82529#100542: *4507759 client 192.168.55.64:2372 connected to
192.168.55.64:15431
2016/08/21 18:39:04 [info] 82529#100542: *4507759 proxy 192.168.55.64:2373 connected to
192.168.55.64:15434
2016/08/21 18:39:04 [info] 82529#100542: *4507759 upstream disconnected, bytes from/to
client:106/7728, bytes from/to upstream:7728/106

2016/08/21 18:39:04 [info] 82529#100542: *4507752 udp client 192.168.55.64:46255
connected to 192.168.55.64:53
2016/08/21 18:39:04 [info] 82529#100542: *4507752 udp proxy 192.168.55.64:46256 connected
to 192.168.55.64:53
2016/08/21 18:39:04 [info] 82529#100542: *4507754 udp upstream disconnected, bytes
from/to client:27/126, bytes from/to upstream:126/27
```

TCP/UDP load balancer: better logging

```
log_format combined '$remote_addr - - [$time_local] $protocol
$status $bytes_sent $bytes_received $session_time "$upstream_addr"';

access_log /var/log/nginx/stream-access.log combined;

127.0.0.1 - - [26/Aug/2016:18:04:16 +0300] TCP 200 158 90 0.007
"127.0.0.1:9000"
```

TCP/UDP load balancer: variables!

Since NGINX 1.11.2:

- map to build variables using other variables
- geo module to build variables based on client's IP addresses
- geoip module to create variables using MaxMind GeoIP
- split_clients to enable A|B testing
- nginScript via js_set

TCP/UDP load balancer: variables!

```
stream {
  server {
    listen 2007;
    listen [::]:2007 udp;
    return $remote_addr;
  }
}

thresh@fruity ~ $ echo "foo" | nc -w 1 127.0.0.1 2007
127.0.0.1
thresh@fruity ~ $ echo "foo" | nc -6 -w 1 -u ::1 2007
::1
```

TCP/UDP load balancer: variables!

```
stream {
  geoip_country GeoIP.dat;
  geoip_city GeoLiteCity.dat;
  limit_conn_zone $geoip_country_code zone=addr:10m;
  split_clients "${remote_addr}AAA" $upstream {
    0.5%  192.168.1.101:5000; # feature test
    *     192.168.1.100:5000;  # production
  }
  server {
    proxy_pass $upstream;
    limit_conn addr 5;
  }
}
```

Extending TCP/UDP LB with nginScript

nginx.conf:

```
load_module modules/ngx_stream_js_module.so;
stream {
  js_include stream.js;
  server {
    listen 8000;
    js_set $foo foo;
    return $foo;
  }
}
```

stream.js:

```
function foo(s) {
  s.log("hello from foo() handler!");
  return s.remoteAddress;
}
```

Extending TCP/UDP LB with nginScript

Moving the requests gradually between the backends: enjoy the demo!

nginx.conf-transition and transition.js:
https://github.com/thresheek/nginxconf16/

TCP/UDP payload fiddling w/nginScript

Simple WAF to protect against ShellShock and HTTPOXY: enjoy the demo!

nginx.conf and waf.js:
https://github.com/thresheek/nginxconf16/

TCP/UDP w/nginScript: performance?

```
TODO: Some performance metrics
```

TCP/UDP load balancer future

As the stream module is rather new, we welcome the feedback and feature requests you think would be valuable for your use cases.

See also

TCP/UDP Load balancer admin guide:

https://www.nginx.com/resources/admin-guide/tcp-load-balancing/

Load Balancing TCP traffic:

https://www.nginx.com/blog/advanced-mysql-load-balancing-with-nginx-plus/

Load Balancing DNS traffic:

https://www.nginx.com/blog/load-balancing-dns-traffic-nginx-plus/

nginScript documentation:

http://hg.nginx.org/njs/file/tip/README

Thank you!

Questions?

Configuration snippets and slides:

http://github.com/thresheek/somewhere/

Konstantin Pavlov, Nginx Inc.

@cryothresh