



## A Byte of Nginx

Sheng Yuan

June 24, 2018

# Contents

- 1 Introduce
- 2 Configure and Grammar
- 3 Web Server
- 4 Reverse Proxy[2]
- 5 Load Balance
- 6 HTTP Caching
- 7 Security and Access
- 8 Deployment and Operation
- 9 Conclusion

# Introduce

## Introduce

## Configure and Grammar

## Web Server

## Reverse Proxy[2]

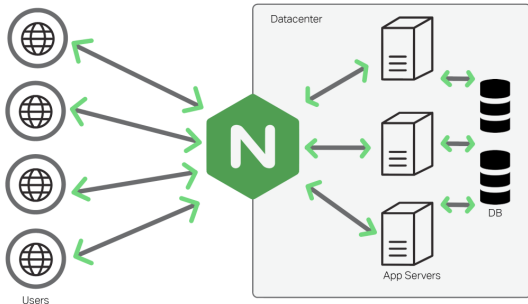
## Load Balance

## HTTP Caching

## Security and Access

## Deployment and Operation

## Conclusion



Nginx is a web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache[3].

## Basic usage

```
$ nginx  
$ nginx -s reload  
$ nginx -s stop  
$ nginx -s quit  
$ nginx -s reopen  
$ nginx -f /conf/nginx.conf
```

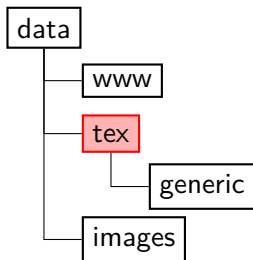
```
$ service nginx start  
$ service nginx stop  
$ service nginx restart  
$ kill -QUIT $( cat /var/run/nginx.pid )
```

- **simple directive** the name and parameters separated by spaces and ends with a semicolon (;)
- **block directive** the same structure as a simple directive, but instead of the semicolon it ends with a set of additional instructions surrounded by braces ({ and })

```
http {  
    server {  
        location / {  
            root /data/www;  
        }  
        location /images/ {  
            root /data;  
        }  
    }  
}
```

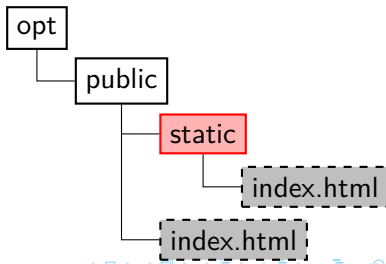
## Simple Web Server

```
http {  
    server {  
        location / {  
            root /data/www;  
        }  
        location /images/ {  
            root /data;  
        }  
    }  
}
```



## Web Server with alias

```
http {  
    server {  
        location /static {  
            alias /opt/public;  
        }  
        location /static {  
            root /opt/public;  
        }  
    }  
}
```



# Proxy Server

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

Deployment  
and Operation

Conclusion

## Proxy Server

```
server {  
    location /debug {  
        proxy_pass http://127.0.0.1:8000/debug;  
    }  
  
    location /static {  
        root /data/public;  
    }  
}
```

## CORS

```
map $request_method $cors_method {  
    OPTIONS 11; GET 1; POST 1; default 0;  
}  
  
server {  
    location /api/v2 {  
        proxy_http_version 1.1;  
        add_header Access-Control-Allow-Origin *.mydomain.com;  
        proxy_pass http://127.0.0.1:8080/api/v2;  
    }  
}
```

# HTTP Load Balance

## Problem

You need to distribute load between two or more HTTP servers.

## Solution

```
upstream backend {  
    server 10.10.12.45:80 weight=1;  
    server app.example.com:80 weight=2;  
}  
server {  
    location / {  
        proxy_pass http://backend;  
    }  
}
```



# TCP Load Balance

## Problem

You need to distribute load between two or more TCP servers.

## Solution

```
stream {  
    upstream mysql_read {  
        server read1.example.com:3306 weight=5;  
        server read1.example.com:3306;  
        server 10.10.12.34:3306 backup;  
    }  
    server {  
        listen 3306;  
        proxy_pass mysql_read;  
    }  
}
```

# Load Balancing methods

## Problem

You need to choose a LB method.

## Solution

- **Round robin** Requests are distributed evenly across the servers, with server weights taken into consideration. This method is used by default (there is no directive for enabling it)
- **Least connections** A request is sent to the server with the least number of active connections, again with server weights taken into consideration
- **Generic hash**
- **IP hash** The server to which a request is sent is determined from the client IP address. In this case, either the first three octets of the IPv4 address or the whole IPv6 address are used to calculate the hash value. The method guarantees that requests from the same address get to the same server unless it is not available
- **Least time** Only supported in Nginx plus.

Sheng Yuan

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

Deployment  
and Operation

Conclusion

- `limit_conn` and `limit_conn_zone` – Limit the number of client connections NGINX accepts, for example from a single IP address. Setting them can help prevent individual clients from opening too many connections and consuming more than their share of resources.
- `limit_rate` – Limits the rate at which responses are transmitted to a client, per connection (so clients that open multiple connections can consume this amount of bandwidth for each connection). Setting a limit can prevent the system from being overloaded by certain clients, ensuring more even quality of service for all clients.
- `limit_req` and `limit_req_zone` – Limit the rate of requests being processed by NGINX, which has the same benefits as setting `limit_rate`. They can also improve security, especially for login pages, by limiting the request rate to a value reasonable for human users but too slow for programs trying to overwhelm your application with requests (such as bots in a DDoS attack).
- `max_conns` parameter to the server directive in an upstream configuration block – Sets the maximum number of simultaneous connections accepted by a server in an upstream group. Imposing a limit can help prevent the upstream servers from being overloaded. Setting the value to 0 (zero, the default) means there is no limit.

# Massively Scalable Content Caching

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

Deployment  
and Operation

Conclusion

## Problem

You need to cache content and need to define where the cache is stored.

## Solution

```
proxy_cache_path /var/nginx/cache
```

```
    keys_zone=CACHE:60m levels=1:2  
    inactive=3h  
    max_size=20g;
```

```
proxy_cache CACHE;
```

Sheng Yuan

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

Deployment  
and Operation

Conclusion

## Solution

```
location /admin/ {  
    deny 10.0.0.1;  
    allow 10.0.0.0/20;  
    allow 2001:0db8::/32;  
    deny all;  
}
```

```
server {  
    listen 80;  
    return 301 https://testai.tclking.com$request_uri;  
}  
  
server {  
    listen 443;  
    ssl on;  
    ssl_certificate /usr/share/certificates/tcl/xxx.crt;  
    ssl_certificate_key /usr/share/certificates/tcl/xxx.key;  
    ssl_session_timeout 5m;  
}  
  
add_header Strict-Transport-Security max-age=31536000;
```

Sheng Yuan

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

Deployment  
and Operation

Conclusion

```
http {  
    log_format geoproxy  
        '[ $time_local ] $remote_addr '  
        '$realip_remote_addr $remote_user '  
        '$request_method $server_protocol '  
        '$scheme $server_name $uri $status '  
        '$request_time $body_bytes_sent '  
        '$geoip_city_country_code3 $geoip_region '  
        '$geoip_city" $http_x_forwarded_for '  
        '$upstream_status $upstream_response_time '  
        '"$http_referer" "$http_user_agent"';  
  
    access_log /var/log/nginx/access.log geoproxy buffer=32k  
    flush=1m;  
  
    error_log /var/log/nginx/error.log main buffer=32k  
    flush=1m;  
}
```

The buffer parameter of the access\_log directive denotes the size of a memory buffer that can be filled with log data before being written to disk. The flush parameter of the access\_log directive sets the longest amount of time a log can remain in a buffer before being written to disk.

# Performance Tuning

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

Deployment  
and Operation

Conclusion

## Keeping Connections Open to Clients

```
http {  
    keepalive_requests 320;  
    keepalive_timeout 300s;  
}
```

**keepalive\_requests:** The number of requests a client can make over a single keepalive connection. The default is 100, but a much higher value can be especially useful for testing with a load-generation tool, which generally sends a large number of requests from a single client.

**keepalive\_timeout:** How long an idle keepalive connection remains open.

## Keeping Connections Open Upstream

```
proxy_http_version 1.1;  
proxy_set_header Connection "";  
upstream backend {  
    server 10.0.0.42;  
    server 10.0.2.56;  
    keepalive 32;  
}
```

**keepalive:** The number of idle keepalive connections to an upstream server that remain open for each worker process. There is no default value. To enable keepalive connections to upstream servers you must also include the following directives in the configuration.



Sheng Yuan

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

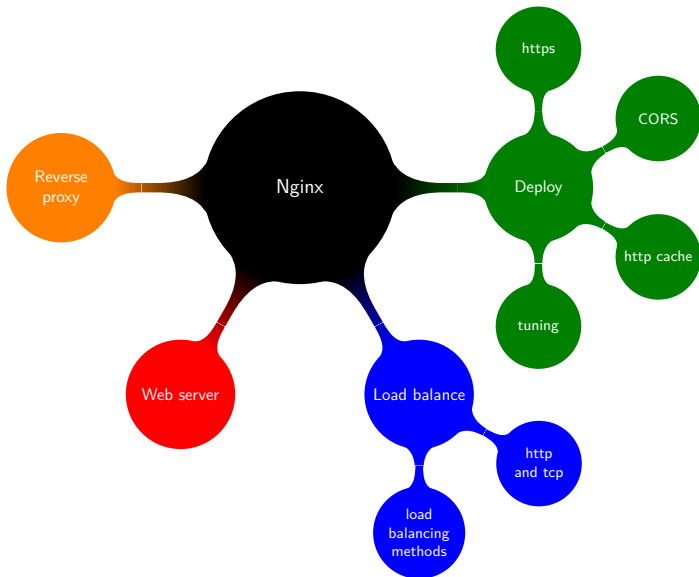
Deployment  
and Operation

Conclusion

- Raising the number of open file descriptors is a more common need.
- Check the kernel setting for `net.core.somaxconn`, which is the maximum number of connections that can be queued by the kernel for NGINX to process.
- Enable more ephemeral ports.

`worker_processes[1]` This is the number of worker processes that will be started. These will handle all connections made by the clients. A good rule of thumb is to set this equal to the number of processor cores for CPU-bound loads and to multiply this number by 1.5 to 2 for I/O bound loads.

`worker_connections` This directive configures the maximum number of simultaneous connections that a worker process may have open. This includes, but is not limited to, client connections and connections to upstream servers. This is especially important on reverse proxy servers. some additional tuning may be required at the operating system level in order to reach this number of simultaneous connections.



Sheng Yuan

Introduce

Configure and  
Grammar

Web Server

Reverse  
Proxy[2]

Load Balance

HTTP  
Caching

Security and  
Access

Deployment  
and Operation

Conclusion



Dimitri Aivaliotis. *Mastering Nginx*. Packt Publishing Ltd, 2016.



DeJonghe Derek. *Complete Nginx Cookbook*. O'Reilly Media, Inc., 2017.



Clement Nedelcu. *Nginx HTTP Server*. Packt Publishing Ltd, 2015.



Rahul Sharma. *Nginx high performance*. Packt Publishing Ltd, 2015.