GitHub Gist

denji / nginx-tuning.md
Last active 13 hours ago

NGINX tuning for best performance

nginx-tuning.md

NGINX Tuning For Best Performance

For this configuration you can use web server you like, i decided, because i work mostly with it to use nginx.

Generally, properly configured nginx can handle up to 400K to 500K requests per second (clustered), most what i saw is 50K to 80K (non-clustered) requests per second and 30% CPU load, course, this was `2 x Intel Xeon` with HyperThreading enabled, but it can work without problem on slower machines.

You must understand that this config is used in testing environment and not in production so you will need to find a way to implement most of those features best possible for your servers.

- Stable version NGINX (deb/rpm)
- Mainline version NGINX (deb/rpm)

First, you will need to install nginx

```
yum install nginx
apt install nginx
```

Backup your original configs and you can start reconfigure your configs. You will need to open your `nginx.conf` at `/etc/nginx/nginx.conf` with your favorite editor.

```
# you must set worker processes based on your CPU cores, nginx does not benefit from setting more than that
worker_processes auto; #some last versions calculate it automatically

# number of file descriptors used for nginx
# the limit for the maximum FDs on the server is usually set by the OS.
# if you don't set FD's then OS settings will be used which is by default 2000
worker_rlimit_nofile 100000;

# only log critical errors
error_log /var/log/nginx/error.log crit;

# provides the configuration file context in which the directives that affect connection processing are specified.
events {
    # determines how much clients will be served per worker
    # max clients = worker_connections * worker_processes
    # max clients is also limited by the number of socket connections available on the system (~64k)
    worker_connections 4000;

    # optmized to serve many clients with each thread, essential for linux -- for testing environment
    use epoll;

    # accept as many connections as possible, may flood worker connections if set too low -- for testing environment
    multi_accept on;
}

# cache informations about FDs, frequently accessed files
# can boost performance, but you need to test those values
open_file_cache max=200000 inactive=20s;
```

```
open_file_cache_valid 30s;
open_file_cache_min_uses 2;
open_file_cache_errors on;

# to boost I/O on HDD we can disable access logs
access_log off;

# copies data between one FD and other from within the kernel
# faster then read() + write()
sendfile on;

# send headers in one peace, its better then sending them one by one
tcp_nopush on;

# don't buffer data sent, good for small data bursts in real time
tcp_nodelay on;

# reduce the data that needs to be sent over network
gzip on;
gzip_min_length 10240;
gzip_proxied expired no-cache no-store private auth;
gzip_types text/plain text/css text/xml text/javascript application/x-javascript application/json application/xml;
gzip_disable msie6;

# allow the server to close connection on non responding client, this will free up memory
reset_timedout_connection on;

# request timed out -- default 60
client_body_timeout 10;

# if client stop responding, free up memory -- default 60
send_timeout 2;

# server will close connection after this time -- default 75
keepalive_timeout 30;

# number of requests client can make over keep-alive -- for testing environment
keepalive_requests 100000;
```

Now you can save config and run bottom command

```
nginx -s reload
/etc/init.d/nginx start|restart
```

If you wish to test config first you can run

```
nginx -t
/etc/init.d/nginx configtest
```

Just For Security Reason

```
server_tokens off;
```

Nginx Simple DDoS Defense

This is far away from secure DDoS defense but can slow down some small DDoS. Those configs are also in test environment and you should do your values.

```
# limit the number of connections per single IP
limit_conn_zone $binary_remote_addr zone=conn_limit_per_ip:10m;
```

```
# limit the number of requests for a given session
limit_req_zone $binary_remote_addr zone=req_limit_per_ip:10m rate=5r/s;

# zone which we want to limit by upper values, we want limit whole server
server {
    limit_conn conn_limit_per_ip 10;
    limit_req zone=req_limit_per_ip burst=10 nodelay;
}

# if the request body size is more than the buffer size, then the entire (or partial)
# request body is written into a temporary file
client_body_buffer_size  128k;

# headerbuffer size for the request header from client -- for testing environment
client_header_buffer_size 3m;

# maximum number and size of buffers for large headers to read from client request
large_client_header_buffers 4 256k;

# read timeout for the request body from client -- for testing environment
client_body_timeout   3m;

# how long to wait for the client to send a request header -- for testing environment
client_header_timeout 3m;
```

Now you can do again test config

```
nginx -t
/etc/init.d/nginx configtest
```

And then reload or restart your nginx

```
nginx -s reload
/etc/init.d/nginx restart|reload
```

You can test this configuration with `tsung` and when you are satisfied with result you can hit `Ctrl+C` because it can run for hours.

DoS HTTP/1.1 and above: Range Requests

By default `max_ranges` is not limited. DoS attacks can many Range-Requests (Impact on stability I/O).

Socket Sharding in NGINX 1.9.1+ (DragonFly BSD and Linux 3.9+)

Socket type	Latency (ms)	Latency stdev (ms)	CPU Load
Default	15.65	26.59	0.3
accept_mutex off	15.59	26.48	10
reuseport	12.35	3.15	0.3

Thread Pools in NGINX Boost Performance 9x! (Linux)

Multi-threaded sending of files is currently supported only Linux. Without `sendfile_max_chunk` limit, one fast connection may seize the worker process entirely.

Happy Hacking!