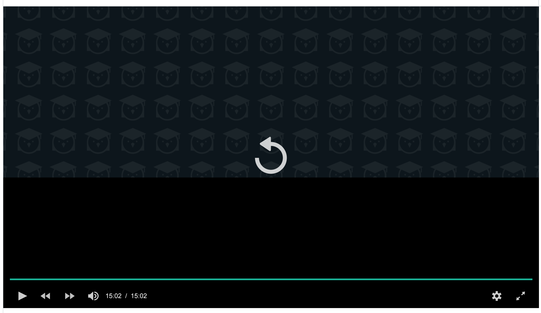
# Lecture: Workers & Connections




Flash Cards Notes Transcript Downloads

Looks like you haven't created any flash cards for this deck yet.

Create My First Flash Card

Popular Decks Popular Cards

Nginx Web Server Deep Dive

NGINX Web Server Deep Dive

NGINX Web Server Deep Dive

When fine-tuning NGINX performance, the two lowest level attributes that we can adjust are the workers and connections. In this lesson, we look into how we can find the proper values for both of these attributes.

*Note:* the commands in this video are run as the `root` user.

## Documentation For This Video

- NGINX `worker_processes` directive
- NGINX `worker_connections` directive

## Adjusting Worker Processes

By default, NGINX comes configured to only use one worker process, but is that good? It really depends on if the server only has 1 CPU core. To optimize our server, we should set the `worker_processes` equal to the number of CPU cores that our server has. Thankfully, we don't need to modify the configuration for every different server that we deploy to because of the `auto` option. Let's tell NGINX to automatically use the right number of worker processes.

*/etc/nginx/nginx.conf* (partial)

```
worker_processes auto;
```

## Adjusting Worker Connections

Sadly, determining the worker connections value that we should use is not as easy as setting something to auto. Adjusting connection information happens in the `events` context of our `nginx.conf` file. The default is `512`, but this number is almost guaranteed to be too low for modern day servers.

Having too low of a limit is going to be bad because if we get a spike in traffic on a 4 core server we could only handle 2k clients even with the `auto` setting for the worker processes. NGINX was designed to handle 10k connections over a decade ago. With that in mind, let's increase our connections pretty substantially:

*/etc/nginx/nginx.conf* (partial)

```
events {
    worker_connections 2048;
}
```

For a four core CPU, this would lead to 8192 simultaneous connections. If NGINX ever hits this limit, it will log an error in `/var/log/nginx/error.log`. Knowing that this will be logged, we should have some log monitoring in place so that we can see if we've set our limit too low. When running a server that is getting a lot of concurrent requests it is possible to run into operating system limits for file connections, and that is something that you'll need to watch for.

If we validate our configuration, we'll see the following:

```
[root] $ nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: [warn] 2048 worker_connections exceed open file resource limit: 1024
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

To view these limits we can use `ulimit` and run it as the NGINX user:

```
[root] $ su -s /bin/sh nginx -c "ulimit -Hn"
4096
[root] $ su -s /bin/sh nginx -c "ulimit -Sn"
1024
```

The `1024` that we're seeing is the "soft" limit. We stayed within the range of the "hard" limit so we can change the value that NGINX uses using the `worker_rlimit_nofile` directive near the top of our configuration:

*/etc/nginx/nginx.conf* (partial)

```
user  nginx;
worker_processes  auto;
worker_rlimit_nofile 4096;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


# Load ModSecurity dynamic module

load_module /etc/nginx/modules/ngx_http_modsecurity_module.so;


events {
    worker_connections  2048;
}
```

In the event that you run into a "Too Many Open Files" error at some point, you can adjust the hard and soft file limits for the nginx user and then change this value to match.

## Utilizing Keepalives

We want to get the most that we can out of the connections that we open up. One way to accomplish this is to configure a `keepalive` value that allows a client to hold an open TCP connection for more than a single request. This same connection can then be used for multiple requests and doesn't need to be closed and reopened. There are two spots in our configuration that we should consider using keepalives.

1. Between the Client and NGINX.
2. Between NGINX and an upstream server/group.

The first type of keepalive that we're going to discuss is between the web client and the NGINX server. These keepalives are enabled by default through the `keepalive_timeout` directive in the `nginx.conf`, and the `keepalive_requests` default value of 100. These values can be set for an entire `http` context or a specific `server`. There's not a set in stone rule or approach to setting the `keepalive_timeout` for NGINX, but the default is used quite often. The `keepalive_requests` directive is interesting because it's not the number of connections to keep open, but rather the number of requests a single keepalive connection can make. If you know that your clients are going to likely make more than 100 requests within your keepalive duration, then you should increase this number to match.

Let's look at how keepalives work for connections between NGINX and proxied servers. To tell an `upstream` to use keepalives, we'll use the `keepalive` directive. Here's what it looks like if we set our `photos.example.com` proxy to use keepalives:

*/etc/nginx/conf.d/photos.example.com.conf*

```
upstream photos {
    server 127.0.0.1:3000 weight=2;
    server 127.0.0.1:3100 max_fails=3 fail_timeout=20s;
    server 127.0.0.1:3101 max_fails=3 fail_timeout=20s;

keepalive 32;

}


server {
    listen 80;
    server_name photos.example.com;


client_max_body_size 5m;

location / {
    proxy_pass http://photos;
    proxy_http_version 1.1;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Real-IP  $remote_addr;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}

location ~* \.(js|css|png|jpe?g|gif) {
    root /var/www/photos.example.com;
}

}
```

The number of connections used with the `keepalive` directive is the number to use with each of the worker processes. It's important to keep this relatively low so that new connections can also be made. Keepalives are one of the areas where we do need to change different proxy settings depending on whether we're using `proxy_pass`, `fastcgi_pass`, or `uwsgi_pass`.

For HTTP and `proxy_pass`, we need to set the HTTP version to 1.1 and make sure that the `Connection` header is set to `"upgrade"` or an empty string `""`. These directives are already used in our photos configuration, but for a different server we would need to make sure that these lines existed:

```
    proxy_http_version 1.1;
    proxy_set_header Connection "";
```

For FastCGI and `fastcgi_pass`, we need to enable the `fastcgi_keep_conn` directive. The directive will look like this:

```
    fastcgi_keep_conn on;
```

For uWSGI and `uwsgi_pass`, there actually isn't the concept of a keepalive connection. uWSGI web services can also work with `proxy_pass`, so that is something to consider if you really want upstream keepalives.

Exceeded my Expectations | Room for Improvement