Beginner's Guide

This guide gives a basic introduction to nginx and describes some simple tasks that can be done with it. It is supposed that nginx is already installed on the reader's machine. If it is not, see the Installing nginx page. This guide describes how to start and stop nginx, and reload its configuration, explains the structure of the configuration file and describes how to set up nginx to serve out static content, how to configure nginx as a proxy server, and how to connect it with a FastCGI application.

nginx has one master process and several worker processes. The main purpose of the master process is to read and evaluate configuration, and maintain worker processes. Worker processes do actual processing of requests. nginx employs event-based model and OS-dependent mechanisms to efficiently distribute requests among worker processes. The number of worker processes is defined in

the configuration file and may be fixed for a given configuration or automatically adjusted to the number of available CPU cores (see worker_processes).

The way nginx and its modules work is determined in the configuration file. By default, the configuration file is named `nginx.conf` and placed in the directory `/usr/local/nginx/conf`, `/etc/nginx`, or `/usr/local/etc/nginx`.

Starting, Stopping, and Reloading Configuration

To start nginx, run the executable file. Once nginx is started, it can be controlled by invoking the executable with the `-s` parameter. Use the following syntax:

```
nginx -s signal
```

Where *signal* may be one of the following:

- `stop` — fast shutdown
- `quit` — graceful shutdown
- `reload` — reloading the configuration file
- `reopen` — reopening the log files

For example, to stop nginx processes with waiting for the worker processes to finish serving current requests, the following command can be executed:

```
nginx -s quit
```

> This command should be executed under the same user that started nginx.

Changes made in the configuration file will not be applied until the command to reload configuration is sent to nginx or it is restarted. To reload configuration, execute:

```
nginx -s reload
```

Once the master process receives the signal to reload configuration, it checks the syntax validity of the new configuration file and tries to apply the configuration provided in it. If this is a success, the master process starts new worker processes and sends messages to old worker processes, requesting them to shut down. Otherwise, the master process rolls back the changes and continues to work with the old configuration. Old worker processes, receiving a command to shut down, stop accepting new connections and continue to service current requests until all such requests are serviced. After that, the old worker processes exit.

A signal may also be sent to nginx processes with the help of Unix tools such as the `kill` utility. In this case a signal is sent directly to a process with a given process ID. The process ID of the nginx master process is written, by default, to the `nginx.pid` in the directory `/usr/local/nginx/logs` or `/var/run`. For

example, if the master process ID is 1628, to send the QUIT signal resulting in nginx' s graceful shutdown, execute:

```
kill -s QUIT 1628
```

For getting the list of all running nginx processes, the `ps` utility may be used, for example, in the following way:

```
ps -ax | grep nginx
```

For more information on sending signals to nginx, see Controlling nginx.

Configuration File' s Structure

nginx consists of modules which are controlled by directives specified in the configuration file. Directives are divided into simple directives and block directives. A simple directive consists of the name and parameters separated by spaces and ends with a semicolon (`;`). A block directive has the same structure as a simple directive, but instead of the semicolon it ends with a set of additional instructions surrounded by braces (`{` and `}`). If a block directive can have other directives inside braces, it is called a context (examples: events, http, server, and location).

Directives placed in the configuration file outside of any contexts are considered to be in the main context. The `events` and `http` directives reside in the `main` context, `server` in `http`, and `location` in `server`.

The rest of a line after the `#` sign is considered a comment.

Serving Static Content

An important web server task is serving out files (such as images or static HTML pages). You will implement an example where, depending on the request, files will be served from different local directories: `/data/www` (which may contain HTML files) and `/data/images` (containing images). This will require editing of the configuration file and setting up of a server block inside the http block with two location blocks.

First, create the `/data/www` directory and put an `index.html` file with any text content into it and create the `/data/images` directory and place some images in it.

Next, open the configuration file. The default configuration file already includes several examples of the `server` block, mostly commented out. For now comment out all such blocks and start a new `server` block:

```
http {
    server {
    }
}
```

Generally, the configuration file may include several `server` blocks distinguished by ports on which they listen to and by server names. Once nginx decides which `server` processes a request, it tests the URI

specified in the request' s header against the parameters of the `location` directives defined inside the `server` block.

Add the following `location` block to the `server` block:

```
location / {
    root /data/www;
}
```

This `location` block specifies the "/" prefix compared with the URI from the request. For matching requests, the URI will be added to the path specified in the root directive, that is, to `/data/www`, to form the path to the requested file on the local file system. If there are several matching `location` blocks nginx selects the one with the longest prefix. The `location` block above provides the shortest prefix, of length one, and so only if all other `location` blocks fail to provide a match, this block will be used.

Next, add the second `location` block:

```
location /images/ {
    root /data;
}
```

It will be a match for requests starting with `/images/` (`location /` also matches such requests, but has shorter prefix).

The resulting configuration of the `server` block should look like this:

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

    location /images/ {
        root /data;
    }
}
```

This is already a working configuration of a server that listens on the standard port 80 and is accessible on the local machine at `http://localhost/`. In response to requests with URIs starting with `/images/`, the server will send files from the `/data/images` directory. For example, in response to the `http://localhost/images/example.png` request nginx will send the `/data/images/example.png` file. If such file does not exist, nginx will send a response indicating the 404 error. Requests with URIs not starting with `/images/` will be mapped onto the `/data/www` directory. For example, in response to the `http://localhost/some/example.html` request nginx will send the `/data/www/some/example.html` file.

To apply the new configuration, start nginx if it is not yet started or send the `reload` signal to the nginx's master process, by executing:

```
nginx -s reload
```

In case something does not work as expected, you may try to find out the reason in `access.log` and `error.log` files

in the directory `/usr/local/nginx/logs` or `/var/log/nginx`.

Setting Up a Simple Proxy Server

One of the frequent uses of nginx is setting it up as a proxy server, which means a server that receives requests, passes them to the proxied servers, retrieves responses from them, and sends them to the clients.

We will configure a basic proxy server, which serves requests of images with files from the local directory and sends all other requests to a proxied server. In this example, both servers will be defined on a single nginx instance.

First, define the proxied server by adding one more `server` block to the nginx' s configuration file with the following contents:

```
server {
    listen 8080;
    root /data/up1;

    location / {
    }
}
```

This will be a simple server that listens on the port 8080 (previously, the `listen` directive has not been specified since the standard port 80 was used) and maps all requests to the `/data/up1` directory on the local file system. Create this directory and put the `index.html` file into it. Note that the

`root` directive is placed in the `server` context. Such `root` directive is used when the `location` block selected for serving a request does not include own `root` directive.

Next, use the server configuration from the previous section and modify it to make it a proxy server configuration. In the first `location` block, put the proxy_pass directive with the protocol, name and port of the proxied server specified in the parameter (in our case, it is `http://localhost:8080`):

```
server {
    location / {
        proxy_pass http://localhost:8080;
    }

    location /images/ {
        root /data;
    }
}
```

We will modify the second `location` block, which currently maps requests with the `/images/` prefix to the files under the `/data/images` directory, to make it match the requests of images with typical file extensions. The modified `location` block looks like this:

```
location ~ \.(gif|jpg|png)$ {
    root /data/images;
}
```

The parameter is a regular expression matching all URIs ending with `.gif`, `.jpg`, or `.png`. A regular expression should be

preceded with `~`. The corresponding requests will be mapped to the `/data/images` directory.

When nginx selects a `location` block to serve a request it first checks location directives that specify prefixes, remembering `location` with the longest prefix, and then checks regular expressions. If there is a match with a regular expression, nginx picks this `location` or, otherwise, it picks the one remembered earlier.

The resulting configuration of a proxy server will look like this:

```
server {
    location / {
        proxy_pass http://localhost:8080/;
    }

    location ~ \.(gif|jpg|png)$ {
        root /data/images;
    }
}
```

This server will filter requests ending with `.gif`, `.jpg`, or `.png` and map them to the `/data/images` directory (by adding URI to the `root` directive' s parameter) and pass all other requests to the proxied server configured above.

To apply new configuration, send the `reload` signal to nginx as described in the previous sections.

There are many more directives that may be used to further configure a proxy

connection.

Setting Up FastCGI Proxying

nginx can be used to route requests to FastCGI servers which run applications built with various frameworks and programming languages such as PHP.

The most basic nginx configuration to work with a FastCGI server includes using the fastcgi_pass directive instead of the `proxy_pass` directive, and fastcgi_param directives to set parameters passed to a FastCGI server. Suppose the FastCGI server is accessible on `localhost:9000`. Taking the proxy configuration from the previous section as a basis, replace the `proxy_pass` directive with the `fastcgi_pass` directive and change the parameter to `localhost:9000`. In PHP, the `SCRIPT_FILENAME` parameter is used for determining the script name, and the `QUERY_STRING` parameter is used to pass request parameters. The resulting configuration would be:

```
server {
    location / {
        fastcgi_pass  localhost:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param QUERY_STRING    $query_string;
    }

    location ~ \.(gif|jpg|png)$ {
        root /data/images;
    }
}
```

This will set up a server that will route all requests except for requests for static

images to the proxied server operating on `localhost:9000` through the FastCGI protocol.