NGINX

NGINX Process

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).

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).

Nginx location directive examples

The location directive within NGINX server block allows to route request to correct location within the file system. The directive is used to tell NGINX where to look for a resource by including files and folders while matching a location block against an URL. In this tutorial, we will look at NGINX location directives in details.

The NGINX location block can be placed inside a server block or inside another location block with some restrictions. The syntax for constructing a location block is:

```
location [modifier] [URI] {
  ...
  ...
}
```

The **modifier** in the location block is optional. Having a modifier in the location

block will allow NGINX to treat a URL differently. Few most common modifiers are:

- **none**: If no modifiers are present in a location block then the requested URI will be matched against the beginning of the requested URI.
- **=**: The equal sign is used to match a location block exactly against a requested URI.
- **~**: The tilde sign is used for case-sensitive regular expression match against a requested URI.
- **~***: The tilde followed by asterisk sign is used for case insensitive regular expression match against a requested URI.
- **^~**: The carat followed by tilde sign is used to perform longest nonregular expression match against the requested URI. If the requested URI hits such a location block, no further matching will takes place.

How NGINX choose a location block

A location can be defined by using a prefix string or by using a regular expression. Case-insensitive regular expressions are specified with preceding "**~***" modifier and for a case-insensitive regular expression, the "**~**" modifier is used. To find a location match for an URI, NGINX first scans the locations that is defined using the prefix strings (without regular expression). Thereafter, the location with regular expressions are checked in order of their declaration in the configuration file. NGINX will run through the following steps to select a location block against a requested URI.

- NGINX starts with looking for an exact match specified with location = /some/path/ and if a match is found then this block is served right away.
- If there are no such exact location blocks then NGINX proceed with matching longest non-exact prefixes and if a match is found where ^~ modifier have been used then NGINX will stop searching further and this location block is selected to serve the request.
- If the matched longest prefix location does not contain ^~ modifier then the match is stored temporarily and proceed with following steps.
    - NGINX now shifts the search to the location block containing ~ and ~* modifier and selects the first location block that matches the request URI and is immediately selected to serve the request.
    - If no locations are found in the above step that can be matched against the requested URI then the previously stored prefix location is used to serve the request.

How nginx processes a request

Name-based virtual servers

nginx first decides **which *server*** should process the request. Let's start with a

simple configuration where all three virtual servers listen on port *:80:

```
server {
    listen      80;
    server_name example.org www.example.org;
    ...
}

server {
    listen      80;
    server_name example.net www.example.net;
    ...
}

server {
    listen      80;
    server_name example.com www.example.com;
    ...
}
```

In this configuration nginx tests only the request's header field "Host" to determine which server the request should be routed to. If its value does not match any server name, or the request does not contain this header field at all, then nginx will route the request to the default server for this port. In the configuration above, the default server is the first one — which is nginx's standard default behaviour. It can also be set explicitly which server should be default, with the default_server parameter in the listen directive:

```
server {
    listen      80 default_server;
    server_name example.net www.example.net;
    ...
}
```

The default_server parameter has been available since version 0.8.21. In earlier versions the default parameter should be used instead.
Note that the default server is a property of the listen port and not of the server name. More about this later.

A simple PHP site configuration

Now let's look at how nginx chooses a *location* to process a request for a typical, simple PHP site:

```
server {
    listen      80;
    server_name example.org www.example.org;
```

```
    root        /data/www;

    location / {
        index   index.html index.php;
    }

    location ~* \.(gif|jpg|png)$ {
        expires 30d;
    }

    location ~ \.php$ {
        fastcgi_pass  localhost:9000;
        fastcgi_param SCRIPT_FILENAME
                      $document_root$fastcgi_script_name;
        include       fastcgi_params;
    }
}
```

nginx first searches for the most specific prefix location given by literal strings regardless of the listed order. In the configuration above the only prefix location is "/" and since it matches any request it will be used as a last resort. Then nginx checks locations given by regular expression in the order listed in the configuration file. The first matching expression stops the search and nginx will use this location. If no regular expression matches a request, then nginx uses the most specific prefix location found earlier.
Note that locations of all types test only a URI part of request line without arguments. This is done because arguments in the query string may be given in several ways, for example:
/index.php?user=john&page=1
/index.php?page=1&user=john
Besides, anyone may request anything in the query string:
/index.php?page=1&something+else&user=john

Now let's look at how requests would be processed in the configuration above:

- A request "/logo.gif" is matched by the prefix location "/" first and then by the regular expression "\.(gif|jpg|png)$", therefore, it is handled by the latter location. Using the directive "root /data/www" the request is mapped to the file /data/www/logo.gif, and the file is sent to the client.
- A request "/index.php" is also matched by the prefix location "/" first and then by the regular expression "\.(php)$". Therefore, it is handled by the latter location and the request is passed to a FastCGI server listening on localhost: 9000. The fastcgi_param directive sets the FastCGI parameter SCRIPT_FILENAME to "/data/www/index.php", and the FastCGI

server executes the file. The variable $document_root is equal to the value of the root directive and the variable $fastcgi_script_name is equal to the request URI, i.e. “/index.php”.

- A request “/about.html” is matched by the prefix location “/” only, therefore, it is handled in this location. Using the directive “root /data/www” the request is mapped to the file /data/www/about.html, and the file is sent to the client.
- Handling a request “/” is more complex. It is matched by the prefix location “/” only, therefore, it is handled by this location. Then the index directive tests for the existence of index files according to its parameters and the “root /data/www” directive. If the file /data/www/index.html does not exist, and the file /data/www/index.php exists, then the directive does an internal redirect to “/index.php”, and nginx searches the locations again as if the request had been sent by a client. As we saw before, the redirected request will eventually be handled by the FastCGI server.
-

Why Use FastCGI Proxying?

FastCGI proxying within Nginx is generally used to t**ranslate client requests for an application server that does not or should not handle client requests directly.** FastCGI is a protocol based on the earlier **CGI, or common gateway interface**, protocol meant to improve performance by **not running each request as a separate process.** It is used to efficiently interface with a server that processes requests for dynamic content.

One of the main use-cases of FastCGI proxying within Nginx is for **PHP processing**. Unlike Apache, which can handle PHP processing directly with the use of the **mod_php** module, **Nginx** must rely on a **separate PHP** processor to **handle PHP requests**. Most often, this processing is handled with **php-fpm,** a PHP processor that has been extensively tested to work with Nginx.

Nginx with FastCGI can be used with applications using other languages so long as there is an accessible component configured to respond to FastCGI requests.