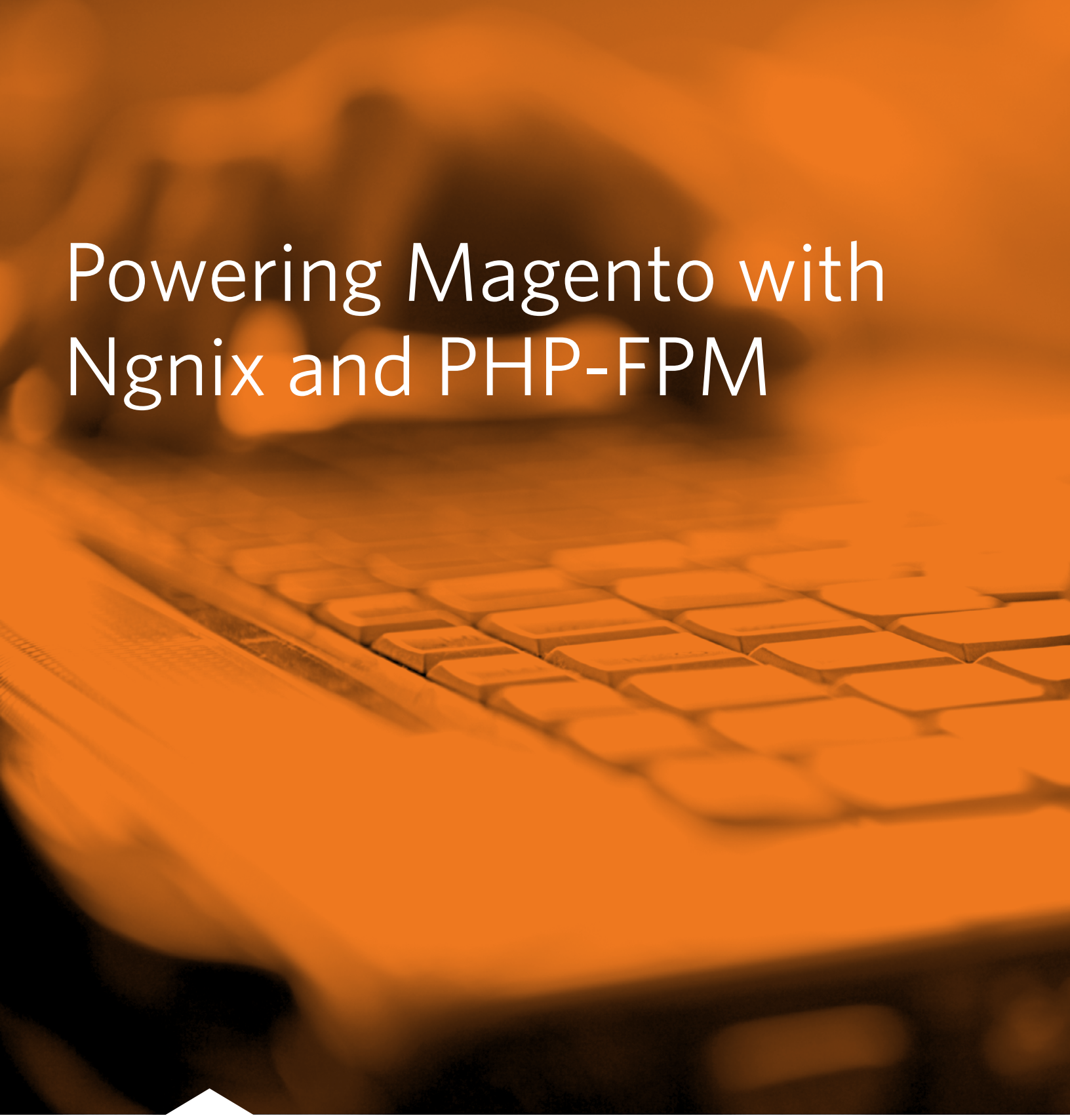

# Powering Magento with Ngnix and PHP-FPM

Magento® EC^g

Written by:
Yuri Golovko
Alexey Samorukov

# Table of Contents

# Introduction

Anyone looking for a simple Magento hosting solution should consider LAMP (Linux/Apache/MySQL/PHP). However, there are cases when Apache with mod_php may not be an optimal solution, particularly if your web sites have dynamic content which is generated by PHP scripts in addition to static files. The purpose of this article is to explore the reasons behind this observation and to demonstrate how using Ngnix with PHP-FPM can help.

Magento ECg

## Why You Should Consider

# Ngnix

# Why You Should Consider Ngnix

When dealing with many simultaneous HTTP/HTTPS connections, Apache uses a high amount of RAM and CPU cycles--especially when talking about a standard Apache configuration with the mod_php prefork. In general, each Apache child process consumes around 100MB of RAM on each request within a typical Magento installation. (To be clear, only resident memory is included in this statistic, minus shared RAM.) On average, a dedicated web server with 16GB of RAM can handle no more than roughly 150 concurrent requests.

The major difference between Nginx and Apache is that Apache is process-based while nginx is event-based. Because Nginx is event-based it doesn't need to spawn new processes or threads to increase its level of concurrency, so its memory footprint is very low. Nginx also exploits non-blocking, asynchronous I/O. Socket() and setsockopt() calls return results without blocking, whereas connect(), send(), recv() and close() may experience some blockage at times. Nginx calls the preceding functions only after confirming there will be no lag. To prevent blockage, for connect() as an example, the socket is changed in advance to a non-blocking socket using ioctl(). As for send() and recv(), epoll is used to prevent blockage. Codes for calling send() or recv() are composed in an event-driven format. Each event is composed of a socket, a socket state, and an operating function.

Nginx is operated by a pre-set number of worker processes. Each process operates as a single isolated process. Ngnix's non-blocking, event-driven architecture allows a single worker process to handle requests by multiple clients.

As stated on the Nginx wiki:

> "Nginx is one of a handful of servers written to address the C10K problem. Unlike traditional servers, Nginx doesn't rely on threads to handle requests. Instead it uses a much more scalable event-driven (asynchronous) architecture. This architecture uses small, but more importantly, predictable amounts of memory under load.
>
> Even if you don't expect to handle thousands of simultaneous requests, you can still benefit from Nginx's high-performance and small memory footprint. Nginx scales in all directions: from the smallest VPS all the way up to clusters of servers."

All of these factors allow Nginx to handle approximately 10,000 HTTP/HTTPS requests per second using just 10 to 20 MB RAM and utilizing about 10% to 15% of average CPU. (When dealing with HTTPS, CPU usage will be higher of course, because HTTPS decryption/encryption routines are highly CPU-intensive.)

Magento EC[g]

Ngnix and

# Static Content

# Ngnix and Static Content

While the advantages of using Ngnix over Apache may be appealing, there is a specific drawback to using Ngnix in that it has nothing like mod_php to execute PHP applications directly its under control.

Nginx is a static content web server and a reverse HTTP or FastCGI proxy. This means that it cannot run Magento directly, but must use another means to do so. Currently, the most beneficial method is to use PHP-FPM (also referred to as the FastCGI Process Manager (http://php-fpm.org/)). PHP-FPM is specifically designed for running high-load web sites with PHP web applications. It consumes a fairly small amount of resident memory (about 30 MB for each child FPM process, roughly three times less than Apache mod_php would consume) and offers a number of unique features. These include:

- Adaptive process spawning
- Ability to start workers with different uid/gid/chroot/environment and different php.ini
- Advanced logging including slowlog for slowly executing PHP scripts
- Emergency restart in case of accidental opcode cache destruction
- Real-time information on server activity and performance (includes real-time memory usage, CPU usage is more detailed, and you can output JSON, XML, HTML, or text). Provides more detail than Apache mod_status

Magento EC[g]

How to

# Use Ngnix

# How to Use Ngnix

An ideal usage of Ngnix to power Magento might include a frontend which provides:

- Reverse FastCGI proxy handling all HTTP/HTTP connections
- All static file delivery
- Caching
- PHP-FPM, FastCGI process manager as a backend for executing Magento

In this setup Nginx acts like a reverse FastCGI proxy for all dynamic content requests: it proxies such requests to a backend PHP-FPM application, waits for a response, and delivers it back to the user. Nginx can also act as a load balancer, which implies that you can have one Nginx frontend and several PHP-FPM backends.  Using this approach provides:

- Backend failover
- High availability and scalability for your Magento installation

Using Ngnix as a load balancer eliminates the need for a hardware load balancer and any associated expenses.

Magento EC[g]

## Ngnix System
# Requirements

# Ngnix System Requirements

Here is the list of necessary components for high performance Magento hosting:

- OS: One of the following x86_64 Linux flavors: Redhat (RHEL 5/6), Debian GNU/Linux stable, Ubuntu LTS. Any 64-bit Linux is fine, although these particular distributions are recommended because they have long-term support and are very stable. One caveat is that the official OS repositories typically contain old versions of packages we are interested in. For Redhat/CentOS, you can optionally use the Remi repository, while for Dotdeb can be used for Unbuntu/Debian distributions.
- Web server: nginx (versions > 1.2.0)
- PHP: php5-fpm (versions > 5.3.8)

This article provides sample configuration files based on Debian setup. For RHEL or other distributions, file system paths or other options may vary slightly.

Magento ECg

Ngnix Setup and

# Load Balancing

# Ngnix Configuration

Configure nginx using its .conf file (/etc/nginx/nginx.conf) that includes other configs as well. Following is a sample configuration of the main file /etc/nginx/nginx.conf:

```
user www-data;
worker_processes 5;
pid /var/run/nginx.pid;
events {
 worker_connections 2048;
 }

http {
 ##
 # Basic Settings
 ##
 sendfile on;
 keepalive_timeout 5;
 types_hash_max_size 2048;
 include /etc/nginx/mime.types;
 default_type application/octet-stream;

map $scheme $fastcgi_https { ## Detect when HTTPS is used
 default off;
 https on;
 }
 ##
 # Logging Settings
 ##
 access_log /var/log/nginx/access.log;
 error_log /var/log/nginx/error.log notice;
 rewrite_log on;
 log_format main ‘$remote_addr - $remote_user [$time_local] $request ‘
 ‘”$status” $body_bytes_sent “$http_referer” ‘
 ‘”$http_user_agent” “$http_x_forwarded_for”’;
 ##
 # Gzip Settings
 ##
 gzip on;
 gzip_disable “msie6”;
```

```
 gzip_vary on;
 gzip_proxied any;
 gzip_comp_level 5;
 gzip_buffers 16 8k;
 gzip_http_version 1.1;
 gzip_types text/plain text/css application/json application/x-javascript
text/xml application/xml application/xml+rss text/javascript;
 ##
 # Virtual Host Configs
 ##
 include /etc/nginx/conf.d/*.conf;
 include /etc/nginx/sites-enabled/*;
}
```

Create the configuration file upstream.conf as follows to define your upstream servers (for a single-host configuration, define PHP-FPM on localhost bound to port 9000) /etc/nginx/conf.d/upstream.conf):

```
upstream fpm_backend {
 server 127.0.0.1:9000; # backend server:port address
 }
```

If both Nginx and PHP-FPM run on localhost, there is no point in using a network; instead, use Unix sockets for performance reasons. In this case the upstream.conf should look like this:

```
upstream fpm_backend {
 server unix:/var/run/php-fpm.sock;
}
```

# Load Balancing on Multiple Hosts

In a multiple-host setup, you may have several upstreams and load-balance among them using persistent connections:

```
upstream fpm_backend {
 server 192.168.100.101:9000;
 server 192.168.100.102:9000;
 server 192.168.100.103:9000;
 server 192.168.100.104:9000;
 keepalive 128; # use fixed number of persistent connections to backend
```

This example distributes load across four backend servers. You have the following options:

- Define the "weight" of each upstream if you have backends of different capacity or otherwise do not want load to be distributed evenly.
- Use "ip_hash" feature to provide "sticky" user sessions (so that logged-in users always connect to the same upstream server).

Nginx provides a basic "health check" feature. For more detailed explanations please refer to official documentation (http://wiki.nginx.org/NginxHttpUpstreamModule).

# Configuring Virtual Hosts

Create an Nginx virtual host file to specify your Magento instance. Below is a sample configuration file /etc/nginx/sites-enabled/magento.conf:

```
server {
 listen 80 default;
 server_name magento.lan www.magento.lan; # like ServerName in Apache
 root /var/www/magento; # document root, path to directory with files
 index index.html index.php;
 autoindex off; # we don't want users to see files in directories

location ~ (^/(app/\|includes/\|lib/\|/pkginfo/\|var/\|report/config.
xml)\|/\.svn/\|/\.git/\|/.hta.+) {
 deny all; #ensure sensitive files are not accessible
 }

location / {
 try_files $uri $uri/ /index.php?$args; # make index.php handle requests for
/
 access_log off; # do not log access to static files
 expires max; # cache static files aggressively
 }

location \~\* \.(jpeg\|jpg\|gif\|png\|css\|js\|ico\|swf)$ {
 try_files $uri $uri/ @proxy; # look for static files in root directory and
ask backend if not successful
 expires max;
 access_log off;
 }
```

```
location @proxy {
 fastcgi_pass fpm_backend; # proxy everything from this location to backend
 }
location \~\.php$ {
 try_files $uri =404; # if reference to php executable is invalid return 404
 expires off; # no need to cache php executable files
 fastcgi_read_timeout 600;
 fastcgi_pass fpm_backend; # proxy all requests for dynamic content to
 # backend configured in upstream.conf
 fastcgi_keep_conn on; # use persistent connects to backend
 include fastcgi_params;
 fastcgi_param SCRIPT_FILENAME $document_root${fastcgi_script_name};
 fastcgi_param MAGE_RUN_CODE default; # Store code is defined in
#administration > Configuration > Manage Stores
 fastcgi_param MAGE_RUN_TYPE store;
 }
}
```

Also under location ~\.php$, an extra parameter should be passed to the backend to use secure checkouts and/or secure access to the Magento Admin Panel:

```
fastcgi_param HTTPS $fastcgi_https;
```

In this setup, Nginx serves static content from its root directory (DocumentRoot in Apache terms) and proxies all requests for dynamic content to upstream/backend servers with PHP-FPM. Nginx might as well cache content and serve it from cache rather than from the PHP-FPM backend. Please refer to official documentation about fastcgi_cache and related directives.

# Using a Dedicated PHP-FPM Server

It is also recommended to use “pm = static” mode (instead of “pm = dynamic”) if you decide to dedicate a server for PHP-FPM exclusively, as there is no need for dynamic allocation of resources to PHP-FPM. The “pm” part of the configuration is more or less the same as if you were to configure Apache.

* parameters directly correspond to Apache configuration directives (pm.max_children equals Apache’s MaxClients, pm.max_requests equals MaxRequestsPerChild, and so on), so a system administrator familiar with Apache performance tuning should have no problem adjusting to a PHP-FPM configuration.

# Got Questions?

Contact ECG at consulting@magento.com