OPTIMIZING Magento® FOR PEAK PERFORMANCE

CONFIGURING ENTERPRISE EDITION FOR GREATER PROCESSING POWER AND IMPROVED RESPONSE TIMES

TECHNICAL WHITE PAPER

TABLE OF CONTENTS

I. Executive Summary

In a world where milliseconds matter, configuring an eCommerce solution for peak performance is essential. Consumers in the online world have become increasingly impatient, and slower response times can affect conversion rates. Online merchants also need an eCommerce solution that can grow and adapt quickly to changing business needs.

Magento Enterprise Edition is designed to meet and exceed these requirements, offering merchants a flexible, powerful eCommerce platform that can easily scale as their businesses grow. Proper configuration of Magento Enterprise Edition together with appropriate hardware and software components will equip merchants with the tools needed to meet their business needs.

This white paper documents the exceptional performance that can be achieved by properly configuring Magento Enterprise Edition. An optimized configuration includes the use of common caching schemes such as Varnish and Nginx along with appropriate hardware. With an optimized configuration, Magento is able to support over 350 million catalog views and 487,000 orders per day with improved response times. The resulting optimization leads to an increase in capacity to accommodate a greater number of visitors and can contribute to higher conversion rates, improved hardware efficiency and overall cost savings.

Comparing Configurations

With the goal of optimizing Magento Enterprise Edition for peak performance, the Magento team set out to identify the optimal hardware and software configuration for a standard 2+1 (two nodes plus one database) setup.

Peer1 Hosting, a respected and experienced hosting provider, provided the servers we used for the test scenarios.

For comparison, we conducted performance tests on three different configurations:

- Standard: Apache + MySQL — The most widely used configuration, combining the Web's most popular HTTP server and open-source database.
- Cache Optimized with Varnish: Varnish + Apache + Percona MySQL — An optimized configuration using Varnish (an open-source reverse proxy server), Apache, and Percona's enhanced version of MySQL.
- Cache Optimized with Nginx: Nginx + PHP-FPM + Percona MySQL — An optimized configuration of Nginx (a combined web server, caching proxy, and load balancing solution), PHP-FPM (the PHP FastCGI process manager), and Percona's enhanced version of MySQL.

The results of our testing—detailed on the following pages—demonstrate that optimizing 2+1 configurations using Varnish or Nginx can lead to significant increases in performance.

II. Results Overview

Through our testing, we found that a combination of data caching optimization and an appropriate hardware and software configuration led to improved overall response times, as well as greater processing power for supporting large numbers of page views and orders per day. With an optimized 2+1 configuration, Magento Enterprise Edition can support more than 350 million catalog views and 487,000 orders per day.

Making Cache Count

Not surprisingly, the greatest performance improvements we saw resulted from optimizing the caching configuration and tools. These changes dramatically improved Magento performance by offloading the application servers.

When it comes to optimization, the main goal is to offload back-end servers by serving as many pages as possible using third-party caching software in addition to native Magento full page caching. If a page does not contain any user-specific information, it can be cached. Any subsequent requests for this page can be satisfied by the third-party cache rather than the application itself. Note, the server performs all caching logic; therefore, no Magento code changes are required to obtain the benefits derived from page caching. Utilizing Memcache, which is natively supported by the Magento application, will cache session data, page layouts, configurations, and product lists. This will significantly improve performance on the servers and offload the application itself. For detailed cache storage configuration information and the location of the slow and fast cache sessions, refer to Appendix A. Caching Backends and Session Save Parameters.

In our performance testing the two optimized configurations—Cached Optimized with Varnish and Cache Optimized with Nginx performed exceptionally well. The external cache provided by Varnish and Nginx with PHP-FPM shows improvements of up to 500% compared to the Standard configuration (Apache + MySQL).

III. Performance Testing Methodology

The various hardware/software configurations were tested by progressively increasing loads while simulating customer page views and product orders. This test scenario models the typical behavior of visitors and buyers—viewing products and categories, adding them to their shopping carts, and then submitting orders.

The following eCommerce parameters were set in each test scenario:

- Number of websites: 1
- Number of stores and store views: 1
- Number of categories and products: 50 categories/100,000 products
- Users per simulation: 2,800
- Conversion rate ranging from 3% – 6%
- Concurrency rate: 100 catalog page views/sec

Two test scenarios were used, each of which simulated different user activities.

The first scenario attempted to serve the maximum number of visitors and buyers performing the following activities:

- Navigating the Magento Enterprise Edition catalog and product pages (visitors).
- Opening category pages, adding products to the shopping cart, and placing orders via one-page checkout as non-logged-in users (buyers).

The second scenario served customer visits only. The goal was to serve as many catalog views from one Web server as possible.

Testing Simulation Tools

To test the performance of Magento Enterprise Edition with different hardware/software configurations, the following testing tools were used:

- Siege v. 2.72 — A load testing and benchmarking utility provided by http://www.joedog.org/siege-home/
- Gatling v. 1.1.0 — An open source stress tool provided by http://gatling-tool.org/[1]

[1] To run the Gatling test tool, a Java Development Kit must be installed on the server. For instructions, consult the Oracle project documentation.

IV. Hardware & Software Specifications

To test Magento Enterprise Edition with different hardware/software configurations, four physical servers (each having a similar hardware configuration) were used. In accordance with the Peer 1 hosting recommendations, each test server had the following characteristics:

Hardware

- CPU: 2 x Intel® Xeon® CPU E5645 @ 2.40GHz
- HDD: RAID1 - LSI MegaRAID SAS 9260-4i; 2 x SAS 164GB 15,000rpm
- RAM: 24GB ECC
- Network Interface Card: Intel 1GB 82576

Software

The following software was installed on the performance testing servers:

- Magento Enterprise Edition 1.11.2.0
- Apache 2.2.15, mpm_prefork, mod_php
- PHP 5.3.10
- MySQL 5.5
- Pecl memcache 3.0.6
- Varnish 2.1.5
- Nginx 1.0.15
- PHP-FPM 5.3.10
- Percona MySQL 5.5
- Memcached 1.4.4
- Operating system: Red Hat Enterprise Linux version 6.2 (Santiago) x86_64

V. Testing the Standard Configuration: Apache with MySQL

The combination of Apache and MySQL is the standard Magento Enterprise Edition configuration. This configuration is documented in the Maximizing Performance and Scalability with Magento Enterprise Edition white paper.

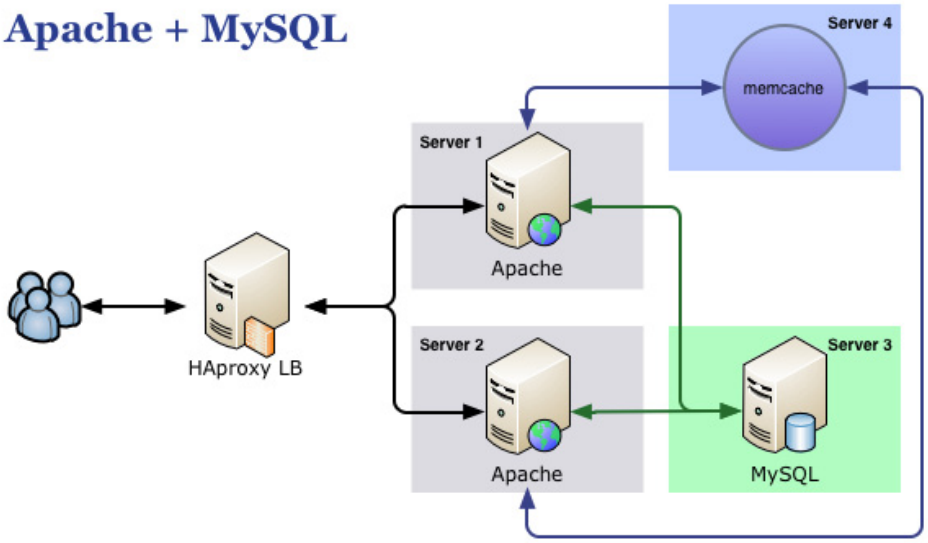

Figure 1: Standard Infrastructure: Apache + MySQL

Figure 1 depicts the following servers and software components:

Servers 1 and 2 — Two application servers; each runs the Apache web server, with mod_php installed; each Apache web server handles both static files and dynamic requests. HAProxy balances requests between each Apache instance

Server 3 — MySQL database server

Server 4 — Memcache key/value storage, which stores session data, temporary data, and rendered blocks

Data Flows and Servicing

Traffic requests are generated using test simulation tools, such as Gatling or Siege. Requests for static content are balanced by HAProxy through the two Apache instances. Requests for dynamic content are handled by the mod_php module built into Apache.

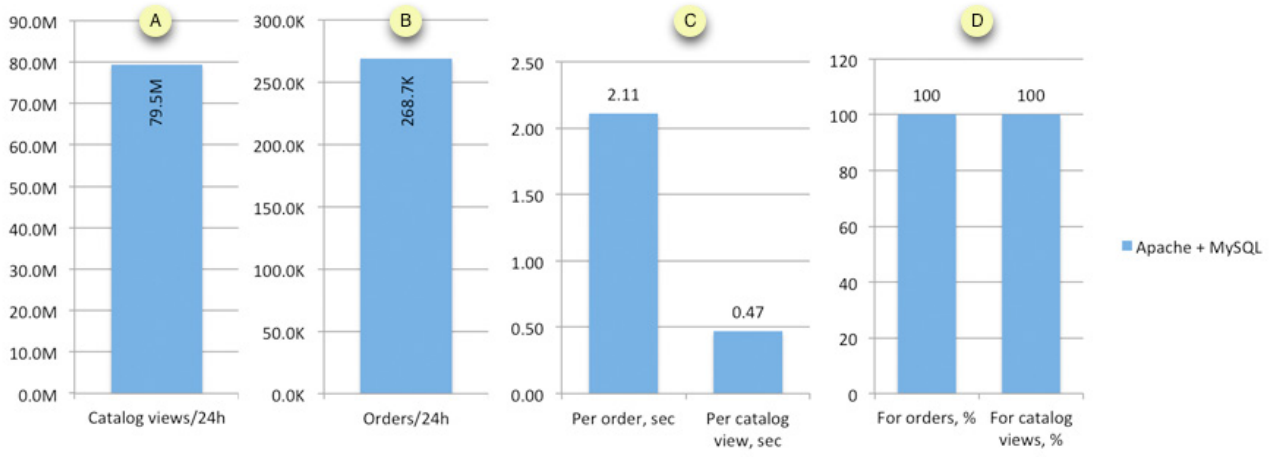

Figure 2: Test Results - Standard Configuration (Apache + MySQL)

Test Results

The tests results for the scenarios described in the Performance Testing Methodology section above are depicted in Figure 2. A and B represent the number of catalog views and orders created during a 24-hour period. C represents the average response time required to serve a single request. D represents the system load of the Apache server.

Summary of Results:

- Number of orders (clients): 2,800
- Time required to process all orders: 900 seconds
- Orders/per sec (average): 3.11
- Response time (average): 2.11 sec
- Web server load: 100%
- Concurrent visitors: 100 page views/sec

With the Standard configuration, the system was able to serve approximately 268,700 orders and 79,500,000 catalog views per day.

Generic Caching Architecture

To optimize hardware/software configuration with an ideal caching scheme, the following components are recommended:

- Cache server — a caching layer; in this case, either Varnish or Nginx
- Application server — one or more servers capable of serving dynamic requests; in this case, the Apache web server or the PHP FastCGI process manager
- Cache storage — storage in RAM or on the disk in which cached documents are stored

Figure 3 shows the basic flow of data in an optimized infrastructure:

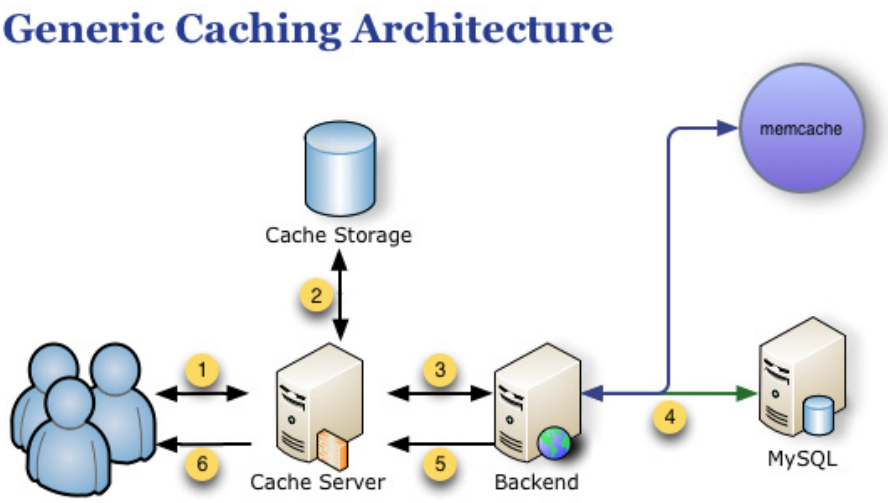

Figure 3: Optimized Infrastructure with Cache Server

In an optimized infrastructure, requests are processed as follows:

1. The cache server evaluates client requests based on the rules defined in its configuration files and chooses how the requests should be served, i.e. whether the requests will bypass the cache 3 or be served from the cache 2 (default case). Some types of exceptions requests require to be served directly by the application server. Those requests include:
    - Specific URLs (adding items to the shopping cart, product reviews, customer login, admin login
    - Presence of the client's "frontend" cookie
    - POST requests

    All other requests should be served from the cache.

2. The requested document (aka information) is looked up in cache storage and is immediately returned to the client 6. If the document is not in cache, the cache server forwards the request to the application server 3.

3. The request is passed to the application server.

4. The application server passes requests to MySQL and Memcached, querying and updating data.

5. The application server sends the response to the cache server. The cache server sends the response to the client in the following cases:
    - If the request-type is not to be cached, it bypasses the cache. All content and cookies (including the "frontend" and "cart" cookies) sent by the application server are preserved and forwarded to the client without modification.
    - The cache server puts the document in the cache. During this process, it unsets cookies, removes headers (such as Pragma and Cache-control - which prevent caching) and increases the time-to-live (TTL) value (by setting the Expires header to 1 hour).

6. The response is sent to the client.

VI. Testing Cache Optimized with Varnish

To improve the software/hardware architecture, the standard MySQL database server was replaced with Percona MySQL and we included the Varnish caching HTTP reverse proxy.

Varnish accelerates web applications by caching most of the static and dynamic content for them. In the caching architecture, Varnish is installed immediately in front of the application server that "speaks" HTTP. The Varnish Configuration Language (VCL) supports writing flexible rules for handling incoming requests and outgoing responses. For detailed information about the Varnish configuration, refer to Appendix C. Varnish Configuration. Additional information about Varnish can be found at: http://www.varnish-cache.org/

For this configuration, Percona enhances MySQL in the following ways:

- Fine-grained mutex locking
- Lock-free algorithms
- Buffer pool pre-load
- Read-ahead improvements
- Shared-memory buffer pool

For more information about Percona MySQL, refer to: http://www.percona.com/

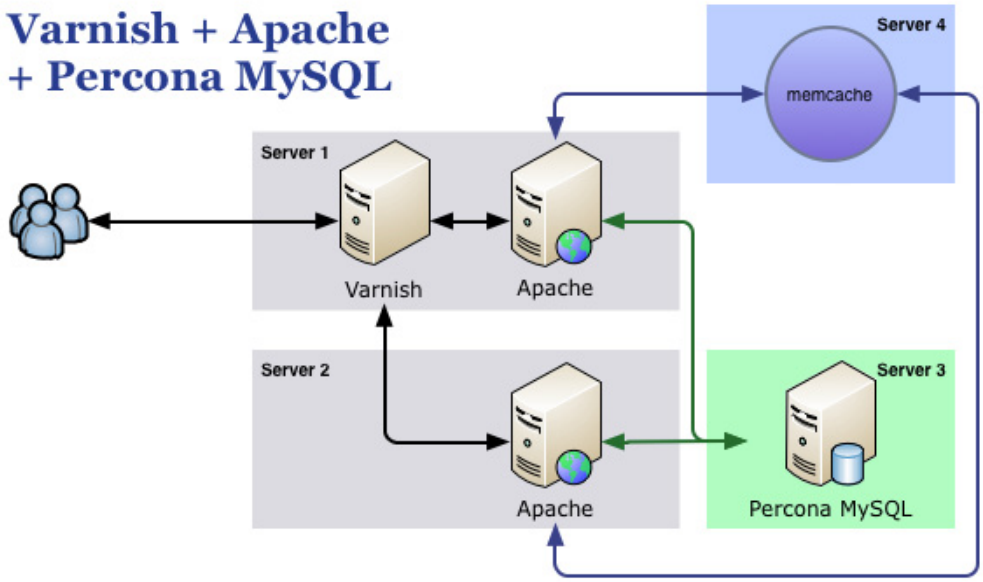

Figure 4: Cache Optimized with Varnish (Varnish + Apache + Percona MySQL)

Disclaimer

Please note that Magento does not claim responsibility for Varnish functional behavior. Magento does not guarantee that all features with Varnish will work as expected. The proposed configurations described in this paper went through extensive load testing, but not full functional testing — they will not fit all scenarios. The suggested configurations are not part of the Magento product, and Magento will not provide assistance, support, maintenance services or customization with the setup of the proposed configurations. Magento has no control of the conditions under which authorized users use the Magento Enterprise software and updates and does not and cannot warrant the results obtained by such use.

The servers shown in Figure 4 have the following components installed:

Server 1 — Reverse proxy server running Varnish. It caches both static and dynamic requests and proxies certain dynamic requests to the application servers (in this case, Apache). The server also has one instance of the Apache web server to handle both static files and dynamic requests. Apache uses mod_php.

Server 2 — Apache web server with mod_php, which also handles both static files and dynamic requests.

Server 3 — MySQL database server.

Server 4 — memcache key/value storage, which stores session and temporary data and rendered blocks.

Data Flows and Servicing

Varnish acts as a cache server, implementing all cache logic and distributing requests among Apache instances. By evaluating client requests based on pre-defined rules in its configuration files, Varnish chooses how these requests should be routed — whether the requests bypass the cache and are proxies to Apache, or served directly from the cache.

As for handling static content — such as images, cascading style sheets, and JavaScript files — Varnish looks up the requested document in its cache and returns it to the client immediately upon a hit. If the document is not in cache, Varnish requests the document from Apache, adds the document to the cache, and replies to the client. All subsequent requests for this specific document are served directly from the cache.

When handling requests to Magento with *.php and *.html dynamic pages, Varnish sends the response to the client by either bypassing the cache (if the content is not to be cached) or by adding the document to the cache and simultaneously responding to the requestor. For additional details on responses to dynamic requests, see the Generic Caching Architecture section above.

Because each reply served from cache is sent without a cookie, the following situation may occur:

- A customer adds a product to the shopping cart.
- Magento checks for the presence of cookies sent as a part of the request.
- If no cookies are found, Magento redirects the customer to the static page, asking to enable browser cookie support.

To prevent this situation, a dummy cookie must be set up on the server side cache.

Test Results

The infrastructure optimized with Varnish full-page caching significantly decreased response time to the client and offloaded the Apache instances. In addition, the Varnish Configuration Language (VCL) provided a way to describe flexible and advanced caching logic. Using Percona, MySQL yields additional benefits, such as better scaling in multiprocessor environments.

Figure 5 shows the results of the two scenarios described in the Performance Testing Methodology section above, as applied to the Standard configuration (Apache + MySQL) and Cached Optimized with Varnish (Varnish + Apache + Percona MySQL). The figure compares the catalog views and orders that were created during a 24-hour period both in the standard and optimized infrastructures. In addition, the figure shows the average response time the system requires in serving a single request. The CPU load in both configurations

reached 100% for order creation. For catalog visits, the optimized configuration only reached 4% of CPU utilization resulting in more bandwidth to handle dynamic operations.

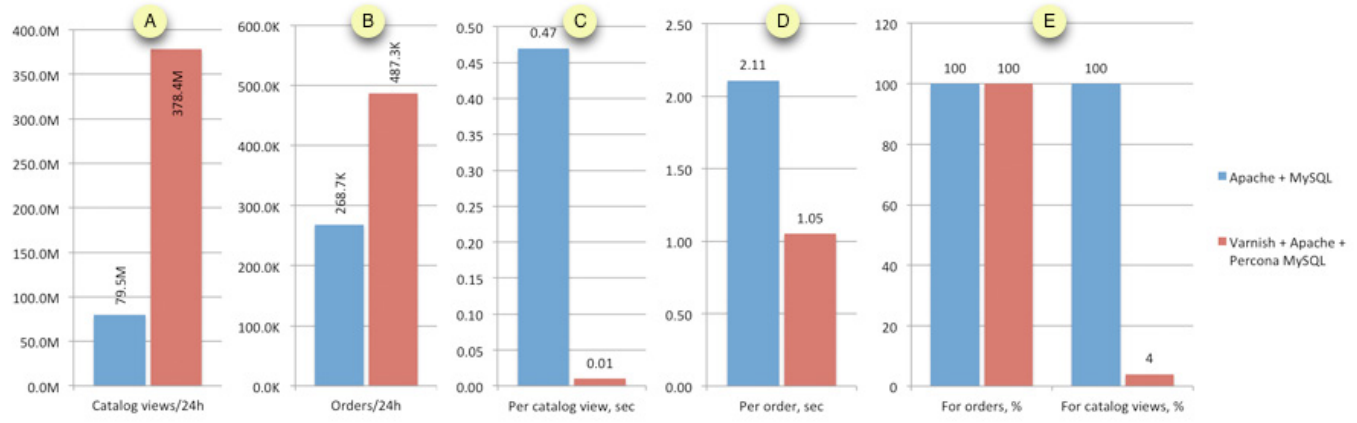

Figure 5: Standard Configuration (Apache + MySQL) vs. Cache Optimized with Varnish (Varnish + Apache + Percona MySQL) Test Results

Summary of Results:

- Number of orders (clients): 2,800
- Time required to process all orders: 496 sec
- Orders/per sec (average): 5.65
- Response time (average): 1.052 sec
- CPU load: 100%
- Concurrent visitors: 100 page views/sec

The system was able to serve approximately 487,300 orders or 79,500,000 catalog views per 24-hour period. The CPU load on web servers during the orders scenario test was as high as it was in the Standard Apache + MySQL architecture (close to 100%). However, due to the aggressive caching of static files as well as catalog pages, the bar was raised to handle 1.8 times additional orders. This architecture can easily serve more than 4,000 catalog page views per second from a single server with Varnish — whereby the only limiting factor is the network interface (1Gbit in these tests). To increase the number of catalog pages served to a client, you can add extra network cards or set up a 10Gbit network. For scaling dynamic content, you must add extra application servers.

VII. Testing Cache Optimized with Nginx

An additional cache-optimized configuration, leveraging the synergy of the Nginx web server, the PHP FastCGI process manager and the Percona MySQL database manager was tested. This configuration was chosen because of the following considerations:

- Nginx provides a unique combination of a web server, a caching proxy with a low memory footprint and a load balancing solution. For serving large amounts of static traffic, Nginx requires fewer system resources than the Apache web server. For more information, refer to this website: http://nginx.com/.
- PHP-FPM (PHP FastCGI Process Manager) handles dynamic requests as an application server. It communicates with Nginx via the FastCGI protocol. Compared to Apache, PHP-FPM has lower memory footprint requirements. For more information about PHP-FPM, refer to this website: http://php-fpm.org/.
- MySQL is enhanced by the Percona support with benefits as outlined above. For more information, refer to this website: http://www.percona.com/.

For detailed information about the configuration of Nginx and PHP-FPM, refer to Appendix D. Nginx and PHP-FPM Configuration.

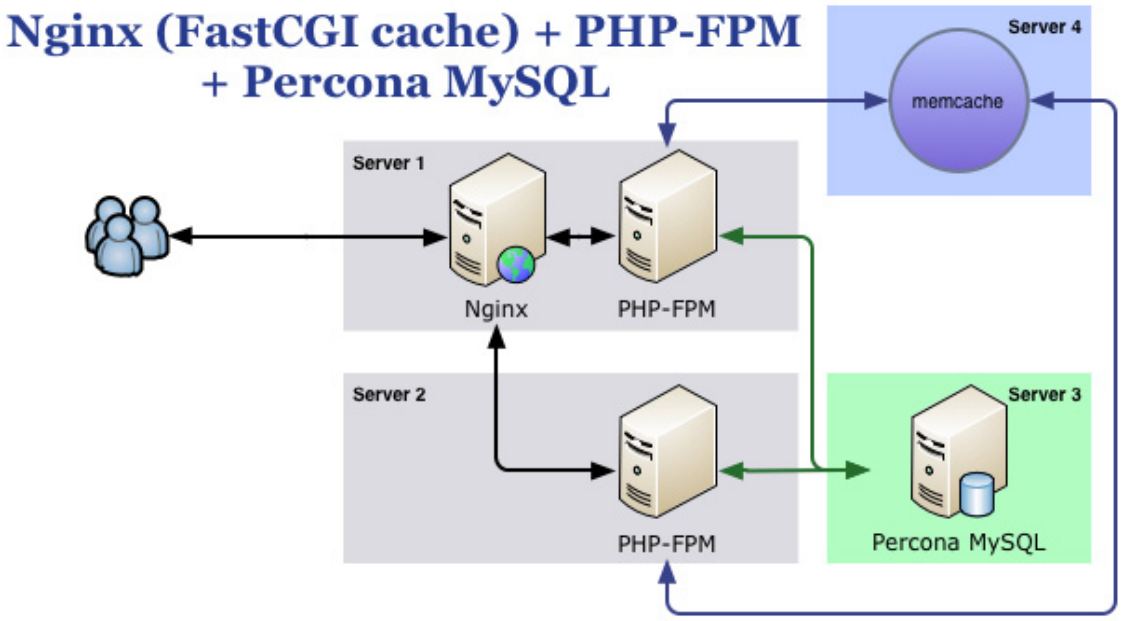

Figure 6: Cache Optimized Infrastructure with Nginx

The Nginx optimized configuration requires the following servers and software components:

Server 1 — This server has Nginx installed. Nginx serves static files directly from the disk and caches dynamic content as much as possible. It proxies certain dynamic requests to the application servers - in this case, PHP-FPM. The server also contains one instance of PHP-FPM which is the FastCGI server for servicing PHP requests.

Server 2 — This server has the second instance of the PHP-FPM application server for servicing PHP requests.

Server 3 — MySQL database server.

Server 4 — This server has the memcache key/value storage, which stores session and temporary data and rendered blocks.

Data Flows and Servicing

Nginx acts as a cache server implementing all cache logic and distributing requests between the PHP-FPM instances. Based on the rules defined in its configuration files, Nginx chooses how to handle requests from the application server by evaluating the client requests. The incoming request either bypasses the cache and goes to the PHP FastCGI process manager or is served directly from the cache.

Because Nginx is a full-fledged web server, it can manage client requests for static content on its own. When a client requests images, cascading style sheets, or JavaScript files, Nginx gets this static content on the disk and returns it to the client immediately.

Nginx handles dynamic *.php and *.html files in one of the following ways:

- If a client provides the "front end" and "client_auth" cookies or requests a specific URL (e.g., the URL for adding a product to the shopping cart, for a product comparison, for customer login, or for store administrator back-end login), Nginx passes the request to the PHP-FPM instance. Nginx then generates the response based on the information received from PHP-FPM and sends it to the client, bypassing the cache
- If no cookie was provided and no predefined URLs were requested, Nginx serves the document to the client directly from the disk

Because each reply is served from cache without a cookie, the following situation may occur:

- A customer adds a product to the shopping cart
- The Magento system checks for the presence of cookies sent as a part of the request
- If no cookies are found, Magento redirects the customer to the static page asking to enable browser cookie support

To prevent this situation, a dummy cookie must be set up on the server-side cache.

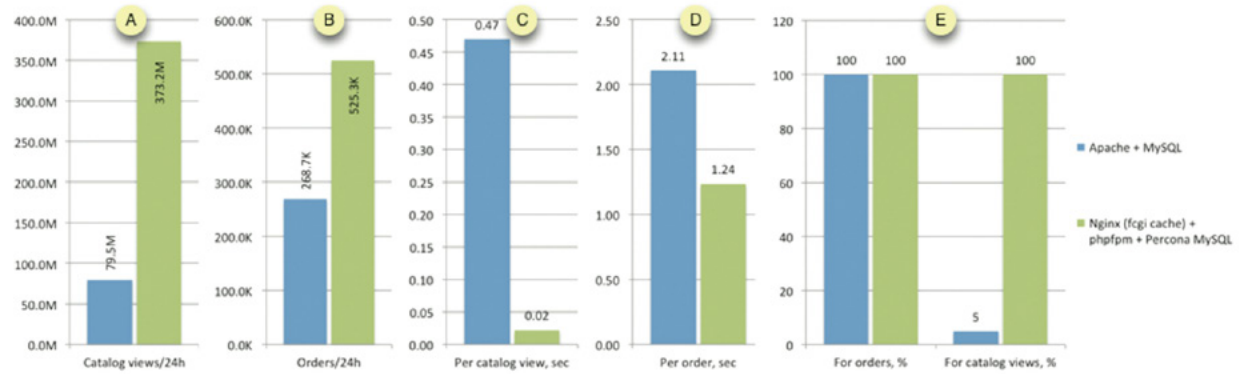

Figure 7: Test Results - Standard Configuration (Apache + MySQL) vs. Cache Optimized with Nginx (Nginx + PHP-FPM + Percona MySQL)

Test Results

The infrastructure optimized with Nginx and FastCGI caching results in a decrease response time and offloads the PHP-FPM instances. This can be achieved with the caching feature built into the Nginx FastCGI module. The use of Percona MySQL yields additional benefits, such as better scaling in multiprocessor environments.

Summary of Results:

- Number of orders (clients): 2,800
- Time required to process all orders: 460 sec
- Orders/per sec (average): 6.09
- Response time (average): 1.237 sec
- CPU load: 100%
- Concurrent visitors: 100 page views/sec

With a configuration optimized by implementing caching inside the FastCGI module of the Nginx server, the system was able to serve approximately 525,300 orders or 373,200,000 catalog views per 24-hour period. During an order scenario test, the CPU load on the web servers was as high as it was on the Standard configuration. However, this configuration managed to offload the system by serving static files to Nginx and aggressively caching catalog pages. This resulted in almost doubling the number of orders created. This architecture can easily serve more than 4,000 catalog page views per second from a single server with Nginx — the only limiting factor is the network connectivity (1Gbit in these tests). To increase the number of catalog pages served to the client, you can add extra network cards or set up a 10Gbit network. For scaling dynamic content, you must add extra application servers.

General Performance Findings

Optimizing the hardware/software configuration with Varnish or Nginx produces impressive results. However, due to differences in their designs, each configuration improves performance in different areas. While Nginx produces better server response times, Varnish does better when it comes to the amount of data being processed.

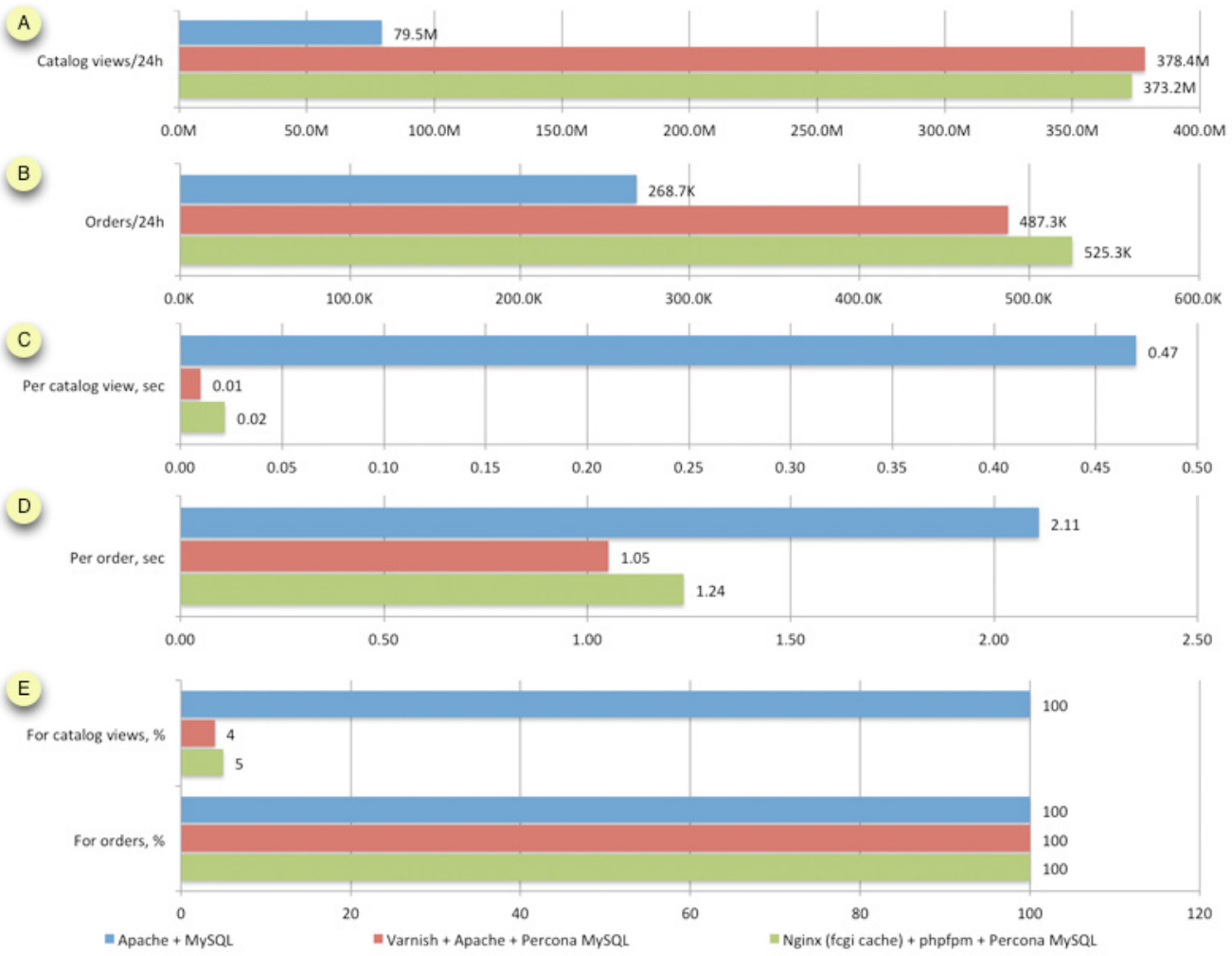

Figure 8: Comparison of Optimized Architectures

VIII. Summary: Results & Recommendations

Proper configuration of Magento Enterprise Edition with the appropriate hardware capabilities and software configuration can lead to a significant improvement in overall system performance. Using Varnish or Nginx as a caching mechanism greatly offloads the application servers from serving dynamic requests. This results in faster response times and allows more requests to be processed by the application server.

The following points summarize the key findings presented for each configuration and provide additional considerations when optimizing a system:

- Using operation code caching provided by the APC accelerator and memcached for storing sessions and temporary data results in significant improvements in overall system performance. For more information on APC accelerators and memcahce, refer to the Maximizing Performance and Scalability with Magento Enterprise Edition white paper.
- Using Nginx with PHP-FPM slightly outperforms Varnish with Apache (by 8%).
- While consuming very little CPU time, both Nginx and Varnish process static files more rapidly compared to Apache. This improvement can be further scaled by adding network cards.
- For the tested scenarios, the database server is not a bottleneck if using one to five application servers. The main bottleneck is the application server CPU bandwidth. With more servers dedicated to handling dynamic requests, more orders can be processed.
- When serving non-cacheable content, the caching layer causes significant delay and decreased response times — by at least 40 times. This results in a net positive outcome.
- By offloading Apache from serving static content, the number of orders handled almost doubles. By using external full-page caching, the number of catalog views can be increased up to 5 times compared to the standard architecture.
- In terms of performance, the caching implementation of Nginx is as efficient as that of Varnish. Thus, it is possible to use Nginx as an alternative to Varnish as a full-page caching solution.
- When deciding between Varnish and Nginx, consider these points: Nginx can be a point of SSL termination; it provides flexible virtual hosting; and preserves cache between restarts. On the other hand, Varnish supports more flexible cache rule definitions due to the Varnish Configuration Language.
- Having a caching layer on two application servers allows the system to survive a single application server failure. If you implement a cache on both servers, use either Active-Passive mode to avoid double caching, or use Active-Active mode behind the load balancer to serve more traffic.

X. APPENDIX

Appendix A. Caching Backends and Session Save Parameters

local.xml

```
<session_save><![CDATA[memcache]]></session_save>
<session_save_path><![CDATA[tcp://localhost:11211?persistent=1&weight=2&timeout=10&retry_
interval=10]]></session_save_path>
<cache>
<backend>database</backend>
<memcached>
<servers><!-- any number of server nodes can be included -->
<server1>
<host><![CDATA[localhost]]></host>
<port><![CDATA[11211]]></port>
<persistent><![CDATA[1]]></persistent>
<weight><![CDATA[2]]></weight>
<timeout><![CDATA[10]]></timeout>
<retry_interval><![CDATA[10]]></retry_interval>
<status><![CDATA[]]></status>
</server1>
<!--
<server2>
...
</server2>
-->
</servers>
<compression><![CDATA[0]]></compression>
<cache_dir><![CDATA[]]></cache_dir>
<hashed_directory_level><![CDATA[]]></hashed_directory_level>
<hashed_directory_umask><![CDATA[]]></hashed_directory_umask>
<file_name_prefix><![CDATA[]]></file_name_prefix>
</memcached>
</cache>
```

Appendix B. Full Page Cache Configuration

FPC is stored on memcache storage

enterprise.xml

```
<config>
  <global>
       <cache>
       <request_processors>
          <ee>Enterprise_PageCache_Model_Processor</ee>
       </request_processors>
    </cache>
    <full_page_cache>
       <prefix>s0meStrInG</prefix>
       <backend>memcached</backend>
```

```
            <memcached>
            <servers>
            <server1>
               <host><![CDATA[host IP address]]></host>
               <port><![CDATA[port]]></port>
              <persistent><![CDATA[1]]></persistent>
               <weight><![CDATA[2]]></weight>
               <timeout><![CDATA[5]]></timeout>
               <retry_interval><![CDATA[5]]></retry_interval>
            </server1>
         </servers>
         <compression><![CDATA[0]]></compression>
         <cache_dir><![CDATA[]]></cache_dir>
         <hashed_directory_level><![CDATA[]]></hashed_directory_level>
         <hashed_directory_umask><![CDATA[]]></hashed_directory_umask>
         <file_name_prefix><![CDATA[]]></file_name_prefix>
         </memcached>
         <slow_backend>database</slow_backend>
      </full_page_cache>
      <skip_process_modules_updates>0</skip_process_modules_updates>
   </global>
</config>
```

Appendix C. Varnish Configuration

Configuration for native Magento (without extension)

/etc/init.d/varnish.vcl

```
backend web1 {
  .host = "Apache IP";
  .port = "8080";
}

backend web2 {
  .host = "Apache IP";
  .port = "8080";
}

director apache round-robin {
   { .backend  = web1; }
   { .backend  = web2; }
}

acl purge {
   "127.0.0.1";
}

sub vcl_recv {
   set req.backend = apache;
```

```
 if (req.request != "GET" && req.request != "HEAD") {
      return (pass);
   }

   # purge request
   if (req.request == "PURGE") {
      if (!client.ip ~ purge) {
         error 405 "Not allowed.";
      }
      purge("obj.http.X-Purge-Host ~ " req.http.X-Purge-Host " && obj.http.X-Purge-URL ~ " req.http.X-Purge-
Regex " && obj.http.Content-Type ~ " req.http.X-Purge-Content-Type);
      error 200 "Purged.";
   }

  if (req.url ~ "^/(media|js|skin)/.*\.(png|jpg|jpeg|gif|css|js|swf|ico)$") {
      unset req.http.Https;
      unset req.http.Cookie;
   }

   if (req.http.Authorization || req.http.Https) {
      return (pass);
   }

   include "/etc/varnish/no_cache.vcl";

   # remove Google gclid parameters
   set req.url = regsuball(req.url,"\?gclid=[^&]+$",""); # strips when QS = "?gclid=AAA"
   set req.url = regsuball(req.url,"\?gclid=[^&]+&","?"); # strips when QS = "?gclid=AAA&foo=bar"
   set req.url = regsuball(req.url,"&gclid=[^&]+",""); # strips when QS = "?foo=bar&gclid=AAA" or QS =
"?foo=bar&gclid=AAA&bar=baz"

   if (req.http.cookie ~ "frontend=") {
         return (pass); }

   return(lookup);

}

sub vcl_hash {

   set req.hash += req.url;

   if (req.http.host) {
       set req.hash += req.http.host;
   } else {
      set req.hash += server.ip; }

   return (hash);
}
```

```
sub vcl_fetch {

   if (beresp.status == 500) {
     set beresp.saintmode = 10s;
     restart; }

   set beresp.grace = 5m;

   unset beresp.http.Server;

   if (beresp.status == 200 || beresp.status == 301 || beresp.status == 302) {

   if (req.url ~ "^/(media|js|skin)/.*\.(png|jpg|jpeg|gif|css|js|swf|ico)$") {
      unset req.http.Https;
      unset req.http.Cookie;
   return (deliver);
   }

      if (beresp.http.Content-Type ~ "text/html" || beresp.http.Content-Type ~ "text/xml") {
      if (req.http.Cookie ~ "frontend=") {
         return (pass); }

      ##URLs we ignore:
      include "/etc/varnish/no_cache.vcl";
            set beresp.http.Set-Cookie = "X-Store=1; path=/";

     if (beresp.ttl < 120s) { set beresp.ttl = 3600s; }
      unset beresp.http.set-cookie;
            set beresp.http.Set-Cookie = "X-Store=1; path=/";

      }
   }

   if (beresp.status == 404) {
      if (beresp.ttl < 120s) { set beresp.ttl = 5s; }
      unset beresp.http.set-cookie;
   }
   return (deliver);
}


sub vcl_deliver {
      remove resp.http.X-Varnish;
      remove resp.http.Via;
      remove resp.http.Age;
      remove resp.http.X-Purge-URL;
      remove resp.http.X-Purge-Host;
```

```
##BEGIN debug section
  if (obj.hits > 0) {
    set resp.http.X-Varnish-Cache = “HIT”;
  }
  else {
    set resp.http.X-Varnish-Cache = “MISS”;
  }
##END debug section

}

sub vcl_error {
    set obj.http.Content-Type = “text/html; charset=utf-8”;
    synthetic {“
<?xml version=”1.0” encoding=”utf-8”?>
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN”
 “http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
<html>
  <head>
    <title>”} obj.status “ “ obj.response {“</title>
  </head>
  <body>
    <h1>Error “} obj.status “ “ obj.response {“</h1>
    <p>”} obj.response {“</p>
    <h3>Guru Meditation:</h3>
    <p>XID: “} req.xid {“</p>
    <hr>
    <p>Varnish cache server</p>
  </body>
</html>
“};
    return (deliver);
}


/etc/varnish/no_cache.vcl
if ((req.url ~ “^/(index.php)/(admin)”) || (req.url ~ “^/(admin)”)) {
    return (pass); }

if (req.url ~ “^/(index.php)/(install)”) {
    return (pass); }

if (req.url ~ “^/(checkout)/(cart)”) {
    return (pass); }

if (req.url ~ “^/(review)/(product)”) {
    return (pass); }

if (req.url ~ “^/(customer)/(account)”) {
        return (pass); }
```

Appendix D. Nginx and PHP-FPM Configuration

/etc/nginx/nginx.conf

```
user            nginx;
worker_processes  12;
error_log       /var/log/nginx/error.log;
pid             /var/run/nginx.pid;

events {
   worker_connections  1024;
}

http {
   include      /etc/nginx/mime.types;
   default_type  application/octet-stream;

   log_format  main  '$remote_addr - $remote_user [$time_local] $request '
                '"$status" $body_bytes_sent "$http_referer" '
                '"$http_user_agent" "$http_x_forwarded_for"';

   log_format cache '***$time_local '
                'Cache Status: "$upstream_cache_status" '
                'Cache-Control: $upstream_http_cache_control '
                'Expires: $upstream_http_expires '
                '"$request" ($status) in $request_time ';

   access_log  /var/log/nginx/access.log  main;

   gzip            off;
      server_tokens         off;
      open_file_cache       max=10000  inactive=30m;
      open_file_cache_valid   10m;
      open_file_cache_min_uses 2;
      open_file_cache_errors  on;

      proxy_read_timeout     60;

      sendfile              on;
      tcp_nopush            on;
      keepalive_timeout      10;
      client_max_body_size    120m;
#       fastcgi_cache_path            /var/cache/nginx levels=1:2 keys_zone=mage:16m max_size=10g
inactive=10h;
      fastcgi_cache_path            /var/cache/nginx levels= keys_zone=mage:16m max_size=10g inactive=10h;

   upstream phpfpm {
   server  127.0.0.1:9000;
   server  192.168.1.4:9000;
   }
```

```
server {
        listen       80 default;
        include /etc/nginx/frontend.conf;
    }
 include /etc/nginx/conf.d/*.conf;

}

/etc/nginx/frontend.conf
server_name     magentoee.lan; #Domain name
root        /var/www/magentoee; #Document root for magento instance

access_log /var/log/nginx/cache.log cache;

location ~* ^.+\.(jpg|jpeg|gif|png|ico|css|zip|tgz|gz|rar|bz2|pdf|txt|tar|wav|bmp|rtf|js|flv|swf)$ {
        root   /var/www/magentoee;
}

location /index { try_files $uri @fcgi_nocache; }

location /checkout { try_files $uri @fcgi_nocache; }

location / {
        try_files $uri @fcgi_cache;
        if ($cookie_frontend) { return 413; }
        if ($cookie_CUSTOMER_AUTH) { return 413; }
        if ($request_method = POST ) { return 413; }
        error_page 413 = @fcgi_nocache;
    }

location @fcgi_cache {
        fastcgi_pass phpfpm;
        include fastcgi_params;
        fastcgi_connect_timeout 60;
        fastcgi_send_timeout 60;
        fastcgi_read_timeout 60;
        fastcgi_buffer_size 4k;
        fastcgi_buffers 512 4k;
        fastcgi_busy_buffers_size 8k;
        fastcgi_temp_file_write_size 256k;
        fastcgi_intercept_errors off;
        fastcgi_param SCRIPT_FILENAME   $document_root/index.php;
        fastcgi_param SCRIPT_NAME       /index.php;
        #fastcgi_keep_conn              on;          # NGINX 1.1.14
              fastcgi_temp_path          /var/tmp 1 2;
        fastcgi_cache              mage;
        fastcgi_cache_key          "$request_method|$http_if_modified_since|$http_if_none_
match|$host|$request_uri";
 #fastcgi_cache_lock         on 5s;   # NGINX 1.1.12
 fastcgi_cache_valid        200 301 302 304 1h;
```

```
        fastcgi_hide_header       "Set-Cookie";
        if ($http_cookie !~ "X-Store=1" ) {
                add_header Set-Cookie "X-Store=1; path=/";
        }
        fastcgi_ignore_headers     "Cache-Control" "Expires" "Set-Cookie";
        fastcgi_cache_min_uses     1;
       fastcgi_cache_valid        30m;
        fastcgi_cache_use_stale    updating error timeout invalid_header http_500;
        fastcgi_cache_bypass       $cookie_EXTERNAL_NO_CACHE $cookie_CUSTOMER_AUTH;
        fastcgi_no_cache           $cookie_EXTERNAL_NO_CACHE $cookie_CUSTOMER_AUTH;
}

location @fcgi_nocache {
        fastcgi_pass phpfpm;
        fastcgi_connect_timeout 60;
        fastcgi_send_timeout 60;
        fastcgi_read_timeout 60;
        fastcgi_buffer_size 4k;
        fastcgi_buffers 512 4k;
        fastcgi_busy_buffers_size 8k;
        fastcgi_temp_file_write_size 256k;
        fastcgi_intercept_errors off;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME   $document_root/index.php;
        fastcgi_param SCRIPT_NAME       /index.php;
        #fastcgi_keep_conn              on;   # NGINX 1.1.14
        fastcgi_temp_path          /var/tmp 1 2;
        if ($http_cookie !~ "X-Store=1" ) {
                add_header Set-Cookie "X-Store=1; path=/";
        }
```

Configuration of PHP-FPM

php-fpm.conf

```
[global]
pid = /var/run/php-fpm/php-fpm.pid
error_log = /var/log/php-fpm/error.log
[www]
listen = 0.0.0.0:9000
listen.backlog = -1
listen.allowed_clients = 127.0.0.1,192.168.1.1
user = apache
group = apache
pm = static
pm.max_children = 800
pm.start_servers = 200
pm.min_spare_servers = 200
pm.max_spare_servers = 200
pm.max_requests = 10000
```

```
slowlog = /var/log/php-fpm/www-slow.log
php_admin_value[error_log] = /var/log/php-fpm/www-error.log
php_admin_flag[log_errors] = on
```

10441 Jefferson Blvd. Suite 200 Culver City, CA 90232
www.magento.com