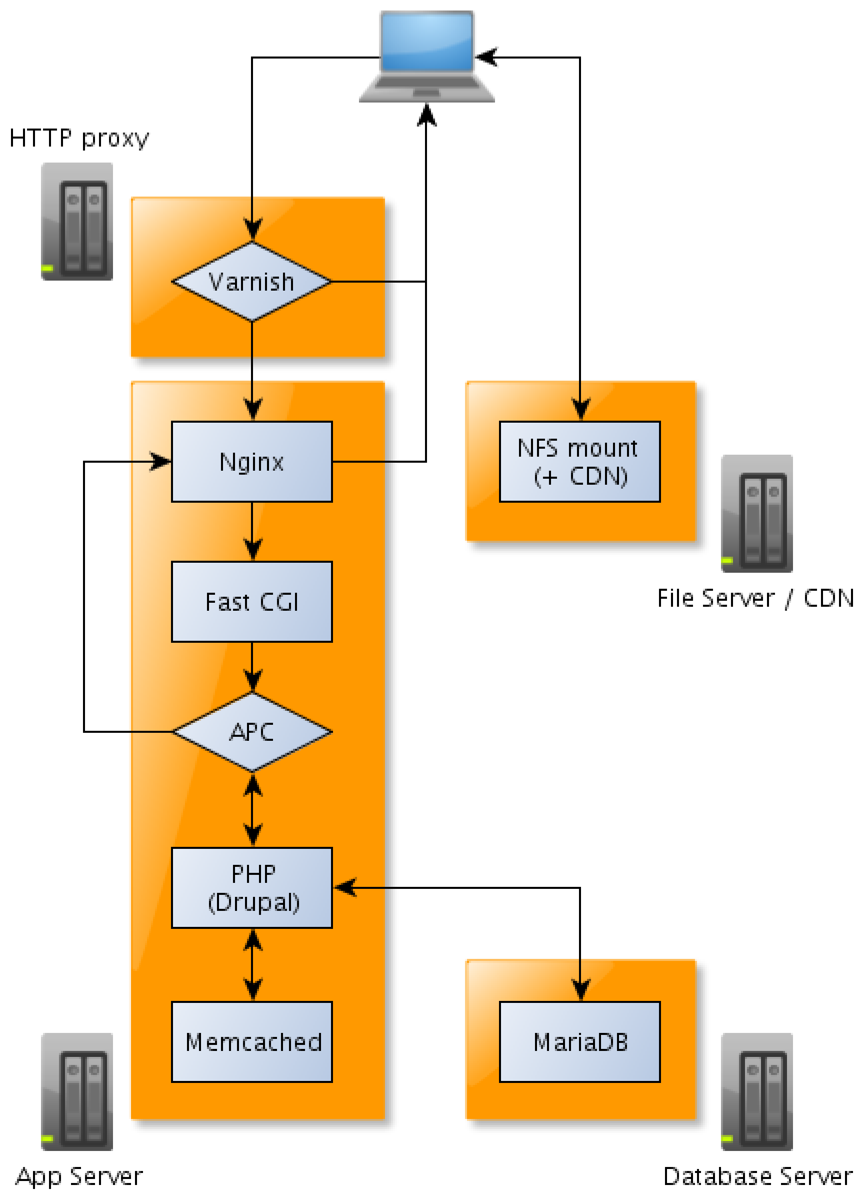
High Performance Stack

This document describes the operation of the Code Enigma High Performance Stack, so you can see how our drop-in LAMP replacement achieves its blistering performance.

If you have any questions about anything in this document, please do not hesitate to contact us on sales@codeenigma.com or telephone +44 (0) 20 3588 1550.

The Perfect Layout

In an ideal world, we would recommend splitting the High Performance Stack over four servers, as shown in the diagram below. Where performance/availability requirements are lower and budget is an issue, all of these systems can (and do) run on a single server, but for resilience reasons we would recommend a degree of fail-over be factored in for any Enterprise client.



Varnish - https://www.varnish-cache.org/

A client request hits the Varnish HTTP proxy first, usually residing on its own server. If the user is logged in to the application then Varnish passes the request straight through to the application server, however if the user is not logged in and Varnish holds a cached copy of the requested page then that is served without calling to the application server at all. This is clearly much, much faster than asking the web server to build a page. Varnish keeps pages in RAM so access to the data is extremely fast. In most circumstances only logged in users actually get sent to the application server, and most traffic is dealt with by Varnish.

Regardless of whether a user is logged in or not, Varnish can also be configured to cache static assets such as images, stylesheets and javascript, further offloading tasks from the application server to the faster Varnish cache and freeing up the application to potentially serve more PHP requests.

Application Server

Essentially, the application server is where the PHP application (usually Drupal) resides. There are several components on the server:

Nginx - http://wiki.nginx.org/

This is a high performance web server used by Code Enigma as a drop-in replacement for Apache in most cases. We can still provide Apache on a High Performance Stack, but the performance will be slightly less high, as Nginx is significantly quicker than Apache in every respect. It is also very stable and growing in popularity, predicted by Netcraft to gain a wider HTTP server market share than Microsoft IIS in the coming months, second only to Apache.

In the case of PHP, Nginx parses client requests that come through from Varnish and pushes them back to PHP itself and the web application code for processing. If the request is for a HTML file or another static asset (e.g. images, JavaScripts, etc.) that has not yet been cached by Varnish, then Nginx will serve that file directly.

Nginx even offers its own internal cache, promising even further improvements in CPU processing time and load.

FastCGI - http://www.fastcgi.com/

When dealing with a PHP request, this is the next step in the chain. There are two common ways PHP is run on a Linux web server. Either as an Apache module (if you are using the Apache web server and the mod_php module) or via good old CGI (the Common Gateway Interface, which you may remember from the days of Perl being the predominant language for writing web applications).

Nginx does not have a PHP module equivalent to mod_php in Apache, so we are stuck with the CGI approach. However, normal CGI is slow and clunky. Enter FastCGI. As a drop-in replacement for CGI but with performance that matches a “proper” HTTP server module for PHP, it is the only sensible choice to run PHP when using Nginx as your HTTP server.

FastCGI is responsible for taking the PHP execution request from Nginx and passing it back to PHP itself for processing.

The specific FastCGI implementation we use is PHP-FPM, which you can read about here: http://php-fpm.org/

APC - http://php.net/manual/en/book.apc.php

APC is a PHP accelerator that uses the opcode caching approach, its full name being “Alternative PHP Cache”. PHP is an “interpreted” programming language, which means in normal operation, every single PHP script request first needs to be compiled into machine code before it can run. Interpreted languages are popular due to their flexibility, but this limitation, the requirement to compile before execution, makes them significantly slower than their pre-compiled counterparts.

Opcode caching actually saves a compiled copy of each PHP script in the server’s memory, so when a request for a PHP execution comes from Fast CGI, APC checks first to see if that script has already been compiled. If it has there is no need to re-compile, so APC allows PHP to effectively behave as though it were a compiled programming language, not an interpreted one, by keeping the PHP code ready as machine code, removing the need to compile every time.

If the requested script is not cached by APC then APC passes the request back to PHP itself, which compiles the script to run. However, at that point APC will keep a copy of the compiled script for future use.

PHP - http://php.net

This is the scripting language our High Performance Stack supports, mainly because this is what Drupal, our own preferred development framework, runs on. It is the last port of call for a client request that has passed through all the caching layers in our stack and is responsible for processing all scripts requested by the client.

Memcached - http://memcached.org/

This is a distributed caching mechanism for storing pretty much anything. We use it in our High Performance Stack to store the Drupal cache for faster retrieval than is possible when the cache resides in a database, as it does by default. We usually run Memcached on the application server because this removes network latency from the equation and the use we put it to means it has a fairly small footprint. However it can be placed on another server (or servers in a cluster) if your application requires - some PHP applications use Memcached as middleware, because it scales easily so stashing data there instead of retrieving it from a database creates a scalable and fast, albeit non-persistent, data source.

File Server

Our file server is usually a separate device when we deploy a high performance and high availability layout for an Enterprise client. This is because when you have many application servers that need the same files there needs to be a single point of storage. (Note, this can easily become a cluster of file servers behind a load balancer, having a single write node and multiple read nodes, just like database replication, if high availability is required.)

A CDN can also be implemented to serve website assets, which has the advantage of staying up if the file server has a problem and also being able to serve files geographically closer to an end user's HTTP request, thus significantly increasing the speed of page load.

Database Server

The final piece in the puzzle is the database itself. This is the "brain" of most Content Management Systems and the most popular open source DBMS is MySQL. We use a drop-in replacement for MySQL called Percona: http://www.percona.com/

Percona is officially supported by Drupal (which we believe means it could be officially supported by any PHP application that supports MySQL).

Percona is preferred over native MySQL because the community-driven Percona project contains various bug fixes and performance enhancements that are either missing from the MySQL product, or have been submitted to MySQL development but stalled due to slow development practices at Sun/Oracle. For more information on the features of Percona as compared to MySQL, please see this page:
http://www.percona.com/software/percona-server/feature-comparison

We can also optionally provide MongoDB as a database server. MongoDB is a NoSQL solution, offering high performance, enhanced scalability and document-oriented storage:
http://www.mongodb.org/

High Performance Tuning

All components of the high-performance stack are deployed using tried-and-tested Puppet manifests that are optimally tuned for performance, out of the box. This cuts down on performance-tuning time (and costs) required, duplication of effort, as well as reducing the margin for error when deploying your Enterprise application.

To gain an appreciation for the performance improvements that can be experienced from this stack, please refer to our Hosting Datasheet.

Disaster Recovery

At the centre of our offering are our back-up and recovery services. If you purchase a server from us, we always have virtual machine snapshot back-ups in place for rapid recovery/cloning of any server. Further, we use an open source application called Duplicity to encrypt the contents of your hard disk and store it with Rackspace on their Cloud Files system. This is done incrementally on a nightly basis and a full back-up is run every fortnight, to be super safe. We use a program written by one of our system administrators, also open source, called Felicity, to stage a recovery and make sure the Duplicity back-ups are sound. This is typically done quarterly, but can be done more frequently if requested. Finally, we take nightly back-ups of your MySQL databases so they can be quickly restored in the event of a catastrophic failure or corruption of data.