UNIVERSITY OF AMSTERDAM

Research Project 1

# Fine-grained control of LAMP component versions for Hosting Companies

Xander Lammertink

University of Amsterdam
MSc System and Network Engineering

2014-2015

## Abstract

Hosting providers offer cheap hosting for websites. Most of these hosting providers provide a LAMP-stack (using Linux, Apache (httpd), MySQL and PHP) updated to the latest stable release. If one or more customer uses out-of-date software that only runs on previous releases of specific components, the hosting company will refrain from upgrading. All users and all applications continue using this previous release which causes security risks. Alternatively the user could switch to another type of hosting which still causes all other applications to run in a previous release.

This research investigates the possibilities to run multiple releases of all or specific components on one server and how to migrate to this situation. Using this situation, every application should be able to use the latest possible release of every component, without the need to switch to other hosting solutions or the need of using multiple servers.

## Acknowledgements

*I would like to thanks Michiel Leenaars from NLnet for providing the research, the help during the research and getting in touch with a hosting provider. I also would like to thanks Martin Heideman from DigiState for participating in the interview about the current situation in their company.*

*Furthermore I would like to thanks the University of Amsterdam (especially the master System and Network Engineering) for providing the workspace, facilities and opportunity to do this research.*

*Xander Lammertink*

# Contents

# 1 Introduction

NLnet has proposed a project to research the possible migration models for hosting providers. In this project they would like to know what migration models hosting provider can use to upgrade their LAMP-stack [1] to a new architecture in which customers can use one authentication structure and can upgrade their (sub-)components independently.

In this project will be researched what software is used by a hosting provider to utilize LAMP-stack hosting, what architecture (new) hosting providers should use to upgrade the different (sub-)components and what migration models can be used to (temporarily) migrate customers to other hosting providers.

## 1.1 Research Question

For this research project a research question with multiple sub-questions have been created. By answering the sub-questions, the research question can be answered. By answering the research question the project will be accomplished.

The research question is: What migration models can be used to (temporarily) migrate customers of LAMP hosting providers to another (new) LAMP hosting provider?

This research question has been split into the following sub-questions:

- What software is used by a LAMP hosting provider to utilize LAMP hosting?
- What architecture should a (new) LAMP hosting provider use to upgrade different (sub-)components of customers?
- What migrations models are available to make hosting providers migrate to the new hosting architecture?
- How can different (sub-) components be up- or downgraded after migrating?

## 1.2 Related Work

Before research began, there was no related academical work found. But during research there appeared to be documentation available that shows how to run multiple instances of Apache (httpd), PHP or MySQL on one machine. This is no academical work.

There has been found official documentation of Apache and MySQL that show running multiple instances are possible.

## 1.3 Methods

This research will be done in three parts. In the first part there will be investigated how the current situation looks like. This will be done by interviewing a hosting company provided by NLnet.

In the second part there will be a look into the "future". How should a new situation look like? And what are the possibilities to set up the new situation?

The last part is the migration. This covers the transition from the old to the new situation. This will cover how to set up your servers to support the new situation.

# 2 Old Situation

To determine what an old situation would look like, Michiel Leenaars arranged a conversation with a hosting provider that is representable for this research. The chosen hosting provider was Digistate. Mr. Heideman founder of DigiState has been interviewed. Mr Heideman founded DigiState in 2008 and is now director in a team with three other colleagues.

Mr. Heideman states that there are 3 types of hosting in the current situation. The first one is shared hosting. Shared hosting means that multiple customers share one server which is kept up to date by the hosting provider. The second type is managed hosting. In managed hosting the customer gets its own dedicated server, but it is being configured and kept up to date by the hoster. The last type of hosting is unmanaged hosting. Here the hosting provider only provides a server (if required even with an operating system). It is up to the customer to install and update the software they would like to use.

## 2.1 Shared Hosting

In most situations a customer will use shared hosting. This is mostly used by customers that do not require a lot of resources. The customers are separated from each other, but use the same server (shared among 300 other customers). The hosting provider makes sure that the operating systems (CentOS), Apache (httpd), PHP and MySQL are up to date, according to the latest stable release (as provided by the repositories). All components are installed on the same server.

## 2.2 Managed Hosting

When a customer requires more resources, but does not want to manage updates and the stable working of the server, it uses managed hosting. In most cases the different components are installed on one server, but based on the needs of the customers it might happen that the MySQL server is installed on another server. In this situation the hosting provider also makes sure that the servers are kept up to date, according to latest stable releases.

In some cases an application cannot use the latest release because the application would stop working correctly. In this case the hosting provider will not update this component and also states this in the Service Level Agreement (SLA).

## 2.3 Unmanaged Hosting

If a customer wants to control their own server, they can use unmanaged hosting. In this case the hosting provider only has provide the hardware and makes sure the server stays available. The software is controlled by the customer.

Customers basically use or share only one server, the authentication is done at the server itself. There is no centralised authentication service that handles the authentication.

## 2.4 Resources

Looking at the used resources, there is just one case in which a cluster has been build to host a customer, normally customers do not use more resources then can handled by one server . In the shared hosting these is just a little overhead in resources per server to make less costs and suppress the prices. Although the capacity is limited, the hosting provider know their capacity and know when to add capacity before it runs out.

## 2.5 Updates and Downtime

In the shared and managed hosting environments the hosting provider applies the updates for customers. Before the updates are done, they first test the updates on a test machine to see if everything works fine. If no problems occur, the updates will be executed on the servers. The updates are executed using the standard repositories during the maintenance windows that are described in the SLA. This way the downtime is reduced to its minimum and most convenient.

## 2.6 Operating System

In managed and unmanaged hosting situations the customer can also choose which operating system they will use. Most customers choose the a Linux operating system (CentOS, Ubuntu, etc.) because of the expertise that is available at the hosting provider. Sometimes customers choose Windows as operating system.

# 3 New Situation

The new situation is a situation in which customers can run their applications on the various infrastructure releases of their choice. Using automated testing, automatic updates should be available. If one application cannot run on the latest release, that application will use a previous release while all other applications on the same server are able to use the latest release.

In addition the authentication to access a server, components or specific applications is centralized. This way authentication is not bounded to the current server and credentials will be consistent everywhere.

## 3.1 Linux

Linux itself will not be covered in the new situation. There are no changes in functionality of the LAMP stack when Linux is being replaced or upgraded since the functionality of the LAMP stack depends on Apache, MySQL and PHP.

Only the authentication will be covered. Since the current situation uses local authentication instead of centralized authentication. Authentication will be covered as part of Linux, but can also be applied in Apache, MySQL and PHP.

The most important requirement about authentication is to use centralized authentication. This means that all authentication is using credentials saved in a centralized system. The second requirement is that all components can use this centralized system and it must be easy to add or remove components that will use the same centralized authentication system.

All four components of the LAMP-stack can use authentication. Linux can use this to allow logging in into the system, Apache can use it to authorize users before they can access certain folders/directories. MySQL uses authentication to set permissions on databases, tables or entries and PHP can use authentication in the application that are build using PHP (for granting access to certain functions)

Looking at authentication one could split up authentication into the authentication itself and the storage of the credentials. Some solutions do both parts themselves and some solutions just do one part and require another solutions to cover the other part.

During research there has been searched for different solutions. These solutions are PAM, Kerberos, LDAP and NIS+. Each of these solutions will be explained and afterwards one solution will be chosen to be used in the migrations models.

**Pluggable Authentication Module**
The first solutions to be discussed is PAM. PAM, which is an abbreviation for Pluggable Authentication Module, is a suite of shared libraries which allows applications to authenticate against a chosen authentication mechanism. If another authentication mechanism is chosen, this can be implemented without recompiling the application. [9]

PAM separates the authentication into four parts. The first is "account" which provides account verification (such as password expiration and access permission). The second part is "authentication" which handles the authentication of the user and sets up the user credentials. This can be done by providing a password, but also using other modules which enables the use of smart-cards or biometrics. The next part is password. The "password" part is responsible for updating the authentication mechanisms. As can be imagined, this part is strongly related to authentication. The last part is called "session" and handles everything that should be done before a service is started and after it is being closed. [10]

A big benefit of PAM is that it allows modules to be plugged in. This gives the possibility to create youown modules or plugging in existing modules without the need to change the applications that are using PAM. Examples of these modules are pam_chroot.so which puts a wrapper around the user by putting them into a "virtual file-system" (e.g. `/` is actually in `/some/directory`), pam_cracklib.so which checks if passwords are strong enough (e.g. by checking for characters in different character classes and not being a palindrome of the old one) or pam_ldap.so to authenticate users against an LDAP directory service. [11]

**Kerberos**
The second option for authentication is Kerberos. This is an authentication protocol which is based on the idea that the internet is an insecure place and therefore no passwords should be sent over the network.

Kerberos is based on three parties, the KDC (Key Distribution Center), the service and the client. The KDC acts as a trusted third party that contains a database with UserID's (e.g. user-name) and the associated symmetric key (e.g. hashed password) and issues tickets to access services.

To log in, the user requests a TGT (Ticket Granting Ticket) that can be used to request tickets. The KDC sends the encrypted TGT using the users hashed password. To access a service, the user sends a request with the TGT as proof. The KDC responds with a ticket to the service which is encrypted using the session key of the user (which was send during the login).

To access the service the user can now send the ticket to the service (encrypted with the key of the service). As an acknowledgement a message is sent back (again encrypted). [12]

**Lightweight Directory Access Protocol**
The next option is LDAP. LDAP, Lightweight Directory Access Protocol, is a directory service which can be compared to a database that is optimized for reading operations. LDAP can be used as a back-end to store the authentication information like user-names and passwords.

LDAP contains a DIT (Directory Information Tree) in which Directory Entries (nodes) can be saved. The Directory Entries contain attributes of different types based on their objectclass. Directory Schemas are used to specify which attributes the different objectclasses should contain.

LDAP can be accessed using the LDIF (LDAP Data Interchange Format) protocol or using a URL. The LDIF protocol is created to search, add, remove or modify data on an LDAP server using plain text instead of binary. A URL (such as `ldap://ldap.example.com/cn=John%20Doe,dc=example, dc=com`) can only be used to search for data in LDAP. More info about LDIF be found in [13]

**Network Information Service Plus**

The last option is NIS+ (Network Information Service). NIS+ provides a lookup service. It distributes information like login names, passwords home directories, groups, hostnames and IP addresses. NIS+ has been introduced by Sun and as a consequence only able to run on Solaris, clients running Linux however can access NIS+

NIS+ also has a less secure variant called NIS. NIS is easier in use, but is also less secure. A downside of NIS and NIS+ is that users need to update passwords on every single system instead of one password update that is being distributed to other systems. [11]

There is a NIS+ version available that runs on Linux. Although all functionality is implemented, the development of NIS+ for linux has stopped "some time ago" because of the lack of time and resources. [14]

**Choosing Authentication Options**

After researching the possibilities for authentication and looking at the requirements one can conclude the following:

- *PAM* is suitable solution when it is combined with a centralized authentication information system. It is very flexible because of the possibility to add or remove modules without recompiling any applications.

- *Kerberos* is a good solution. It does not need to be combined with other solutions because this authenication solution also stores the associated information. The strength of Kerberos is its secure basis by not sending any passwords over the network.

- *LDAP* is also a good solution, but it only handles the storage of authentication information. This means it needs to be combined with an authentication system. The benefits of LDAP are centralization and accessibility from external networks (only if it is configured that way).

- *NIS+* is not a suitable solution because it is not centralized. Every system stores their own credentials and changing them will only lead to changes on that specific system. NIS+ for Linux is also not supported anymore.

Now solutions have been graded, lets take a look at the options we now have and evaluate these:

- *PAM + LDAP*: PAM can be used in combination with LDAP using a module that uses LDAP to get the credentials of the users. This has the benefit of the flexibility of PAM, but lacks the security that Kerberos offers.
- *PAM + Kerberos*: Using a module for PAM it is possible to make PAM handle the Kerberos authentication. This combines the flexibility of PAM and its modules and the safety of Kerberos.
- *Kerberos*: Kerberos can also be used on its own. This way you can benefit the security it offers, but without the benefits of PAM.

Now these options have been evaluated one can conclude that the best option would be a *combination of PAM and Kerberos* to benefit of both the security and the flexibility it offers. In addition to this combination it is also possible to add LDAP for other purposes like storing email addresses associated with users, although this reaches out of the scope of this project.

## 3.2 Virtualization

One way for a customer to use other releases is to set up multiple virtual machines, each using a Linux distribution and another set of instances. This would result in $a$ x $p$ x $m$ servers (where $a$ stands for the number of Apache instances, $p$ or PHP instances and $m$ for the number of MySQL instances).

Assuming 5 different releases of every component will be used, this will result in 125 virtual servers. This does not include that most users are probably able to use the most recent releases of all component, which again will result in one (virtual) server handling all the requests, or the need to use multiple servers.

Another problem arises in how to make sure every request is redirected to the right server. Since this must be done at subdirectory level, this could be achieved using layer 7 switching. A layer 7 switching is able to switch packets based on their content. It could take look into the packets and determine what URL is used in the HTTP-requests. Based on the URL it will then switch the packet to the right server. [15]

Although virtualization is a possible solution, it is not a suitable solution. A lot of virtual machines are needed which all have to be maintained. The virtual machines will also create overhead, since all machines have to run the same Linux distribution next to the unique combination of the LAMP-stack. This means extra server capacity is needed to host the same amount of customers.

## 3.3 Containers

Another way that might be possible is using containers. Containers can be used to package applications. Once a containers is packaged, it can be used, transferred or copied to different servers.

Using containers there are two ways to run multiple versions. One possibility is to create one container that contains the whole LAMP-stack. This would lead to a situation in which every container consists out of a unique LAMP-stack.

Another way is to create a container for every component. By creating a container or every release of a component, one would be able to stack a container of every component to create a unique LAMP-stack.

The advantages of this solution is that it is very easy to set up a unique LAMP-stack. The downside however is every machine would be able to run only one LAMP stack. This would again require virtualization to run multiple virtual machines on one physical server, which will again create overhead.

## 3.4 Multiple instances

The last possibility that has been found is installing multiple instances on one server. This means one server will contain all used releases of every component. Based on the request that comes in at the server it is possible to redirect the request to another instance on directory or file level.

The advantage is that every server can run all instances instead of just one and requests can be redirected on directory or file level. The downside is that all servers need to be configured for all instance.

## 3.5 Details

Looking at the advantages and disadvantages that virtualization, containers and multiple instances offer. I would recommend multiple instances as the best option, based on the facts that it will create less overhead then using virtual machines and gives the possibility to redirect requests on directory or file level.

So lets take a closer look...

### 3.5.1 Apache

The need to run multiple version of Apache on a single machine is a requirement that is getting more common. Apache has now provided some documentation on how to make multiple instances of Apache working on one machine to use multiple instances. [2]

The basic idea is to copy `/etc/apache2` to `/etc/apache2-xxx` so new instances can run independently from each other. Apache provided a script to ease this process.

To send traffic to the right instance a proxy is required. The proxy matches sites or folders and sends the requests to the right instance. The `/etc/apache2` installation will become a reverse proxy server that listens to port 80 (and 443). The other instances use a normal setup, but listen to different ports to avoid communication issues.

The proxy server uses the modules mod_proxy and mod_proxy_http to redirect all incoming traffic on port 80 (and 443) to the new address as specified in the configuration. The configuration will show VirtualHosts using a ServerName to distinguish the different domain names. This VirtualHost contains the ProxyPass and ProxyPassReverse lines to redirect requests to other instances running on different ports.

The other servers use normal configurations. They also contain VirtualHosts using a ServerName to distinguish the different domain names, but instead of the ProxyPass lines they contain a DocumentRoot line to locate files. [3]

Instead of using a proxy on every machine, there is also the possibility to use a proxy that is placed in front of the hosting machines. This server could execute the same functionality but for all servers together.

### 3.5.2 MySQL

To run multiple instances of MySQL, MySQL already proposed a way to run multiple instances on one machine. MySQL can run multiple instances using different settings for every instance.

MySQL proposes to install the different instances in separated folders. Every instance can be connected using a different port number. By knowing which MySQL instance is listening to which port (for example 3306, 3307 and 3308) the customer can connect the right MySQL instance. [4]

Next to the port number, the socket path, pid file and, if used, the paths to the different log files have to be different on each of the instances. These configuration changes can be made by configuring the installation of MySQL (when MySQL is installed from source). More information about running multiple MySQL instances can be found in [5]).

### 3.5.3 PHP

Since there are already products available to install multiple PHP instances, one of these products (ntPHPselector [6]) has been analysed. ntPHPselector is a free script that makes it possible to use a different PHP interpreter per directory.

ntPHPselector works with a single script and is based on the use of SuPHP and cPanel. The concept of the system lies in two things. The first part is the installation of multiple instances of PHP in different folders. The scripts handles all installations and copies the PHP configuration of the main PHP instance.

The second part ntPHPselector is based on is the ".htaccess" file. This file can be placed in every directory and is configured to tell Apache how to handle the files that are in that specific directory. [7]

Seeing how ntPHPselector installs multiple instances of SuPHP, a way to install multiple instances of a normal PHP can be derived from it. PHP is a module that runs on top of Apache or it can be accessed through CGI. [8] Since there are already multiple instances of Apache it will be difficult to use multiple instances of PHP via modules. Therefore CGI will be used to access PHP.

As seen in ntPHPselector it was possible to run multiple versions of PHP when it is installed in separated directories and accessed through CGI. Now only Apache has to know which instance it should pick. In normal situations this can be done by creating MIME-types in the configuration file that are associated to file extension. Based on a MIME-type Apache can decide to use PHP through CGI.

If multiple MIME-types are created, associated with different file extensions, every MIME-type can be configured to use a different PHP instance. This requires file extensions for scripts to be changed if it has to run on a previous PHP release, so in extension to the main configuration .htaccess files can be used.

In every folder it is possible to create a .htaccess file that will overrule the association between file extensions and MIME-types. This way it is possible to let every folder (or even file) use a different instance of PHP.

# 4 Migration

The migration will be covered in three parts, one part per hosting type. The first part is about shared hosting and the second one about a managed hosting environment. The third part, about unmanaged hosting, will be covered shortly.

As described in section 2 the unmanaged hosting will not be managed by the hosting provider, therefore the hosting provider is not responsible for migrating these environments. The migration of the shared hosting environment is split into multiple parts. These parts are Linux, Apache, MySQL and PHP.

Since the shared hosting environment is updated to the latest release of all components, one can assume that websites hosted on current versions are running without problems. Therefore there is no need to set up instances running on previous releases.

This section will only cover the migration from the old to the new situation. Every time updates are available it is up to the hosting provider to add and update instances. There are also a few additional recommendations that would simplify management in the future, but these are not mandatory.

## 4.1 Shared Hosting

### 4.1.1 Linux

The only functionality of the LAMP-stack that is offered by Linux is the authentication. In the old situation the authentication is done locally by storing the credentials in a local file. The new situation offers central authentication by the use of PAM and Kerberos. The credentials that are now stored in Linux have to be transferred to Kerberos.

Currently the credentials of local users are stored in the `/etc/passwd` and `/etc/shadow` files. The `/etc/shadow` file however, makes sure that passwords are practically impossible to retrieve by storing only the hashed (and salted) passwords using the SHA512 algorithm. [16]

Since the passwords are practically impossible to retrieve, there are only a few options left. The fist is reading the `/etc/shadow` file and use a script to detect and add every user (filtered by checking for a password) to Kerberos. This options requires a password reset since the passwords are practically impossible to retrieve.

The second option is using the pam_krb5_migrate module. This module is able to automatically migrate users to Kerberos using the values `PAM_USER` and `PAM_AUTHOK`. This options requires a user to log in before the credentials can be transferred. [17]

The first option is a simple solution that can be executed very quickly using a simple script, the downside is that passwords of users need to be reset. The second option is a solution where users have to interact before they are being migrated. It is a bit more difficult to insert a module, but after a simple user login the user is migrated without experiencing any changes. It is up to the hosting company to choose which options suites the best.

### 4.1.2 Apache

The migration of Apache is split up in two things. The first is setting up multiple versions of Apache using the "setup-instance" script provided by Apache. This script can be found in `/usr/share/doc/apache2/examples/setup-instance`.

Secondly the proxy needs to be set up to redirect traffic to another port on which the right Apache instance is listening. The main instance of Apache can be used to proxy traffic (using the mod_proxy module) while the other instances can be used to actually serve websites. The Apache configuration file should be configured to listen to port 80 (and 443) and contains VirtualHosts for every site (and possibly subdirectories). Every incoming request will then be redirected to the right Apache instance by proxying it to the corresponding port.

The second instance of Apache also needs the configuration file to contain VirtualHosts for every site, but this time to let Apache know the ServerName, DocumentRoot, etc. To make management easier it is recommendable to include a shared configuration file (`include filename.conf`) into the global configuration of all instances (except for the proxy). The shared configuration file should contain all virtual hosts. This way only the proxy configuration has to be changed to make the site run on another instance. [18]

To complete the changes that are made, a restart of the services is enough. This will only take a few seconds. All changes can be executed on the same server without migrating any domains.

### 4.1.3 MySQL

For MySQL there are no configuration changes that have to be made. The system administrator can simply install extra instances of MySQL by downloading the source code and configuring it to use another base directory (`--prefix=[directory]`), unix socket (`--with-unix-socket-path=[file name]`) and port (`--with-tcp-port=[port]`) to install the instance.

It is recommended to let the updated instance listen to the default port (3306) and let other instances listen to other ports (e.g. 3307, 3308, etc.). The customers that want to use the latest version can simply use one port, instead of changing it every time an update is applied.

The database in the current MySQL instance is not automatically migrated or duplicated to other MySQL instances. If the choice is made to use another MySQL instance, the customer also has to migrate the database. There are two ways of doing this, the simplest way is using the MySQL Schema Transfer Wizzard which is included in the MySQL Workbench software. Another option is to do this manually by creating a MySQL dump and import the database. For more information about this topic, see: [19] [20]

### 4.1.4 PHP

Installing different versions of PHP is possible by downloading the source of PHP and configure it to use another install directory (e.g. `--prefix=/usr/local/php5.6`. Every instance of PHP must be installed in a different directory.

Now Apache needs configuration to know which version of PHP it should use. To do this the Apache configuration file (`apache2.conf`) needs to be changed by creating a MIME-type associated to a file extension. This MIME-type can be configured to start a PHP instance through CGI.

Example: (by default files with extension .php and .php56 are interpreted by PHP 5.6, extension .php55 by PHP 5.5, etc.)

```
<Directory "/srv/www/">
  # Create MIME-types
    AddType application/x-httpd-php56 .php56 .php
    AddType application/x-httpd-php55 .php55
    AddType application/x-httpd-php54 .php54
  # Associate MIME-types with associated CGI scripts
    Action application/x-httpd-php56 /cgi-bin/php56.cgi
    Action application/x-httpd-php55 /cgi-bin/php55.cgi
    Action application/x-httpd-php54 /cgi-bin/php54.cgi
</Directory>
```

In this example a customer can set up PHP applications to work with a specific version of PHP on two different ways. They can choose to change the extension to .php** where ** is the version number (e.g. a .php55 file is executed by a PHP 5.5 instance). Another way is to make a directory wide change to make .php files work on a specific PHP instance. This can be done by creating a file named `.htaccess` that contains the line `AddType application/x-httpd-php** .php` where ** is the version number again.

Since PHP needs to be configured in Apache, each instance of Apache needs this configuration to make PHP run. To make the system management easier it is recommended put the MIME-type configuration into a separated file and include this file into the Apache configuration.

## 4.2 Managed Hosting

Managed hosting and shared hosting are practically the same. The biggest difference is that a customer has a system for its own and therefore does not always need multiple instances for every component.

It would be a waste of resources to set up a lot of instances for every component while only two component need multiple instances. Therefore only the components that are necessary for the customer, need to be set up.

Customers running old versions of some components for specific applications can now upgrade to recent versions. The applications that still need the old version can run on one instance, while other applications can simply use another instance that is up to date. This can be different for every customer.

The only part that always needs migration is authentication. The migration of authentication will allow customers to use centralized instead of local authentication.

## 4.3 Unmanaged Hosting

Unmanaged hosting environments are not managed by the hosting provider. Therefore all changes need to be done by the customer. It is up to the customer to decide if they would like to make use of this migration.

Looking at authentication, it is up to the customer to choose if they would like to use the centralized authentication offered by the hosting provider. This could be a benefit if the hosting provider is asked to make changes on the server (for whatever reason).

A benefit for customers to migrate the other components is that every application can use the latest version of all components. As in a managed hosting environment it might not be necessary to use different instances of every component, but the customer should only install multiple instances of a components if it is really needed.

# 5 Conclusions

## 5.1 Old Situation

Hosting providers provide three types of hosting: shared hosting, managed hosting and unmanaged hosting. In a shared and managed hosting environment the hosting provider provides the complete LAMP-stack (Linux, Apache, MySQL and PHP) using the latest stable software available.

In a shared hosting environment multiple customers share one server. Managed hosting customers get their own server managed by the hosting provider. Unmanaged hosting is a type of hosting where the hosting provider provides the server and only manages the hardware.

In a managed hosting environment the customer can choose to use an old version of (multiple) components. The hosting provider will then not update these components. This also means that other applications on the same server have to use the same old components.

## 5.2 New Situation

In a desirable situation applications use the latest stable release of a component. Only when an application needs to use a previous release of a component, that specific application will use the old component. Other applications will still be running on the latest stable release. To accomplish this situation measurements have to be taken by installing and configuring multiple instances of all component. In addition the authentication has to be centralized.

### 5.2.1 Linux

Lets start with Linux, in the current (old) situation Linux uses local authentication. In the new situation it should use centralized authentication. This can be accomplished in various ways using PAM, LDAP and/or Kerberos.

The first options is using PAM combined with LDAP. PAM will handle the authentication and LDAP will store the credentials. This gives the flexibility of using PAM modules, but lacks the security that Kerberos offers.

Then there is the second option: PAM and Kerberos. Using a module PAM can be used to authenticate against Kerberos. This options gives the flexibility of PAM and also the safety that Kerberos offers.

The last option is Kerberos on its own. This of course does not give you the benefits of PAM, but it does offer the safety of Kerberos.

It is recommended to make use of PAM combined with Kerberos. This gives the benefits of both PAM and Kerberos.

### 5.2.2 Apache

Running different versions of Apache on one server requires multiple instances of Apache to be installed into separated directories. These instances also need different ports to listen to.

To redirect every site or folder to the right instance, Apache needs one instance that is configured as a proxy. An incoming request at the proxy (on port 80 or 443) can be redirected to another port (matching the right instance) based on the configuration.

### 5.2.3 MySQL

To run different versions of MySQL on one server it has to run multiple instances installed in separated directories, listening to different ports.

Because every instance of MySQL is running on a separate port, the hosting provider can provide a list with the port and version numbers associated to the different instances. The customer can simply choose the version by connecting to a specified port.

### 5.2.4 PHP

PHP is a bit more difficult since it normally runs as a module on top of Apache. However, it is also possible to send requests to PHP through CGI (Common Gateway Interface).

To let Apache run PHP scripts via CGI, MIME-types can be created. MIME-types can be created in the configuration file and are associated by a file extension. Apache can be configured to execute files via CGI based on the MIME-type.

Next to the global configuration there is also a possibility to change this configuration on specific files or folders. This can be done using a .htaccess file with another configuration. Configuration changes made in the .htaccess files overrule the global configuration

Example: the extension .php is globally configured to run on PHP 5.6. Now a .htaccess file is created which associates the .php extension to another MIME-type that is configured to run PHP 5.5. This configration will overrule the global configuration and will now make the .php file run on PHP 5.5.

## 5.3 Migration

### 5.3.1 Linux

The only Linux part that needs to be migrated is the authentication. Unfortunately migrating the authentication is not as simple as just copying the credentials from the `/etc/passwd` and `/etc/shadow` file to Kerberos. This is practically impossible because the passwords in the `/etc/shadow` are hashed (using SHA512) before they are stored.

There are two options in migrating credentials to the Kerberos server. The fist option is adding the pam_krb5_migrate module to PAM. When a user logs in, the username and passwords can be migrated to Kerberos. The second option is to copy all usernames that have a password from the `/etc/shadow` file. This however requires a password reset, which is inconvenient for customers.

The first option is a very gentle solution where the customer does not experience any changes. The downside is that is requires action from an end user which might take a while. Therefore the second option might be a better solution, it does not require any action from the user, but the password reset can be inconvenient.

### 5.3.2 Apache

To run multiple instances of Apache, the "setup-instance" script provided by Apache can be used to create an instance for every release. The main instance needs to be configured as a proxy server. It should listen to port 80 (and 443) and redirects requests to the other instances based on the configuration.

Every other instance is listening to other ports (e.g. 81, 82, etc.) and serves the actual files. These ports only need to be opened to the local host, the port that is used by the proxy needs to be opened to the internet.

To make management easier it is recommended to split the configuration in multiple files. This way the virtual hosts for serving files can be stored in a separate configuration file and can be included into the configuration files of the specific instances. When a site needs to run on a different release, only the proxy configuration needs to be changed.

### 5.3.3 MySQL

MySQL proposed to install multiple instances by installing MySQL from source. This way the installation can be configured to install into a separated directory listening to a different port (e.g. 3307, 3308, etc.). Every version needs installed in a separate directory.

To distinguish instances, every instance should listen to another port. The customer can choose which version to use by choosing which port to connect to. It is recommended to let the updated instance run on the default MySQL port (port 3306).

Unlike Apache and PHP, MySQL needs databases to be transferred if another instance is chosen. This can be done using tools like the MySQL Schema Wizzard provided by MySQL or by manually creating a dump of the database and import it into the new MySQL instance.

### 5.3.4 PHP

PHP also requires multiple instances to be installed in separated directories. This can also be done by installing it from source. Next, Apache needs to be notified to use PHP for file with the .php (or .php**) extension. This can be done by configuring Apache to assign MIME-types to file extensions. This MIME-type can be configured to activate PHP via CGI.

For every version of PHP a new MIME-type needs to be created. Every MIME-type can be bound to a .php** extension, where ** is the PHP release number (e.g. .php55 for PHP version 5.5).

If a customer wants to use another PHP release then the latest one it can do a few things. The first option is to change the extension to .php**, where ** represents the release it would like to use.

The other option is creating a .htaccess file in the folder where the files are located. This .htaccess file should contain a configuration that binds the .php extension to the MIME-type of a specific release and overrules the global configuration.

It is recommended to bind the .php extension to the latest release of PHP (on global level) and to store the MIME-type configuration in a separated file that is included into the other configuration files.

### 5.3.5 Hosting Environments

Since every hosting environment is different, not every component is relevant to migrate. Looking from a management perspective, a shared hosting environment is completely different from an unmanaged hosting environment.

Therefore it is important for hosting providers and customers to look at the actual needs and requirements. For a shared hosting environment it is recommended to migrate all components to the new situation. A managed hosting environment should at least migrate the authentication, but depending on the needs of the customer and their applications, they might not migrate all components.

An unmanaged hosting customer might not want to implement anything of the new situation. The customer has to decide if they want to apply (parts of) the migration based on their needs. Since an unmanaged hosting environment is not managed by the hosting provider it is up to the customer to apply the migration.

# 6 Suggestions for Future Work

Now research concluded that it is possible to run multiple releases of components by running multiple instances, this also opens new opportunities to research.

The most important topic is how the new situation will affect the performance. For Apache all traffic will first pass through a proxy server and PHP is now loaded via CGI. These configuration changes can affect response time and server load, but will they?

The next topic is about changing modules. LAMP has now been defined as a combination of Linux, Apache, MySQL and PHP. Although this combination is used the most, the MySQL database could also be replaced by MariaDB or PHP by Perl or Python. To take this even further, Linux could be replaced by Windows (Server) which creates a WAMP-stack. Can these and other components also use the same principle as used in this research?

In section 3.3 there has been thought about using containers to separate installations. This was considered to be a bad solutions when it is used as a complete solution. It could however help to package the different releases into containers. This could help system administrators to safe a lot of time for the installation of large environments. In extension these containers might also be able to run isolated to offer more security.

The last topic is management panels. Hosting provides often provide a management panel (like cPanel and DirectAdmin) so customers can easily change some configuration settings. Adding a simple user interface would make it easier for customers to change the releases of components for specific applications. Can this functionality be added to the management panels?

## References

[1] Wikipedia, (2015). LAMP (software bundle). [online] Available at: http://en.wikipedia.org/wiki/LAMP_(software_bundle) [Accessed 7 Jan. 2015].

[2] Anonscm.debian.org, (2015). [pkg-apache] Contents of /trunk/apache2/README.multiple-instances. [online] Available at: http://anonscm.debian.org/viewvc/pkg-apache/trunk/apache2/README.multiple-instances?view=markup [Accessed 14 Jan. 2015].

[3] Internet, C. (2015). New Apache instance with Reverse Proxy — The Art of Web. [online] The-art-of-web.com. Available at: http://www.the-art-of-web.com/system/apache-reverse-proxy/ [Accessed 14 Jan. 2015].

[4] Dev.mysql.com, (2015). MySQL :: MySQL 5.1 Reference Manual :: 2.11.4 MySQL Source-Configuration Options. [online] Available at: http://dev.mysql.com/doc/refman/5.1/en/source-configuration-options.html [Accessed 13 Jan. 2015].

[5] Dev.mysql.com, (2015). MySQL :: MySQL 5.7 Reference Manual :: 5.3 Running Multiple MySQL Instances on One Machine. [online] Available at: http://dev.mysql.com/doc/refman/5.7/en/multiple-servers.html [Accessed 13 Jan. 2015].

[6] Nixtree.com, (2015). Nixtree —— ntPHPselector. [online] Available at: https://www.nixtree.com/ntphp.php [Accessed 13 Jan. 2015].

[7] Simpson, S. (2014). How To Build Multiple Versions of PHP with cPanel - The Wonderful World Of Linux. [online] Thewonderfulworldoflinux.com. Available at: http://thewonderfulworldoflinux.com/blog/2014/03/21/how-to-build-multiple-versions-of-php-with-cpanel/ [Accessed 13 Jan. 2015].

[8] Wikipedia, (2015). Common Gateway Interface. [online] Available at: http://en.wikipedia.org/wiki/Common_Gateway_Interface [Accessed 15 Jan. 2015].

[9] Linux-pam.org, (2015). Chapter 1. Introduction. [online] Available at: http://www.linux-pam.org/Linux-PAM-html/sag-introduction.html [Accessed 22 Jan. 2015].

[10] Linux.die.net, (2015). pam.d(8): Pluggable Authentication Modules for - Linux man page. [online] Available at: http://linux.die.net/man/8/pam.d [Accessed 22 Jan. 2015].

[11] Linuxgeek.net, (2015). Linux Authentication Systems - Linux Geek Net. [online] Available at: http://www.linuxgeek.net/documentation/authentication [Accessed 21 Jan. 2015].

[12] Security of Systems and Networks: Lecture 13 Kerberos SSL TLS. (2014). 1st ed. [ebook] Amsterdam: OS3, p.35. Available at: https://www.os3.nl/_media/2014-2015/courses/ssn/ssn_lecture_13_2012_kerberos_ssl_tls.pdf [Accessed 22 Jan. 2015].

[13] Tools.ietf.org, (2015). RFC 2849 - The LDAP Data Interchange Format (LDIF). [online] Available at: http://tools.ietf.org/html/rfc2849 [Accessed 23 Jan. 2015].

[14] Linux-nis.org, (2015). www.linux-nis.org (Linux NIS+ Support). [online] Available at: http://www.linux-nis.org/nisplus/ [Accessed 23 Jan. 2015].

[15] Foundrynet.com,. 'Layer 7 Switching Overview'. N.p., 2015. Web. 6 Feb. 2015.

[16] Man7.org, (2015). shadow(5) - Linux manual page. [online] Available at: http://man7.org/linux/man-pages/man5/shadow.5.html [Accessed 26 Jan. 2015].

[17] Docs.oracle.com, (2015). Synopsis - man pages section 5: Standards, Environments, and Macros. [online] Available at: https://docs.oracle.com/cd/E26502_01/html/E29043/pam-krb5-migrate-5.html [Accessed 26 Jan. 2015].

[18] Httpd.apache.org, (2015). core - Apache HTTP Server Version 2.2. [online] Available at: http://httpd.apache.org/docs/2.2/mod/core.html [Accessed 26 Jan. 2015].

[19] Dev.mysql.com, (2015). MySQL :: MySQL Workbench :: 9.7 MySQL migration. [online] Available at: http://dev.mysql.com/doc/workbench/en/wb-migration-database-mysql.html [Accessed 26 Jan. 2015].

[20] Digitalocean.com, (2015). How To Migrate a MySQL Database Between Two Servers — DigitalOcean. [online] Available at: https://www.digitalocean.com/community/tutorials/how-to-migrate-a-mysql-database-between-two-servers [Accessed 26 Jan. 2015].

[21] Wiki.debian.org, (2015). LDAP/PAM - Debian Wiki. [online] Available at: https://wiki.debian.org/LDAP/PAM [Accessed 21 Jan. 2015].

[22] Wiki.samba.org, (2015). Samba AD DC HOWTO - SambaWiki. [online] Available at: https://wiki.samba.org/index.php/Samba_AD_DC_HOWTO [Accessed 21 Jan. 2015].

[23] Server, H. (2015). How to Get a Windows Client to Authenticate against a Linux LDAP Server. [online] Askubuntu.com. Available at: http://askubuntu.com/questions/12464/how-to-get-a-windows-client-to-authenticate-against-a-linux-ldap-server [Accessed 21 Jan. 2015].