

Server Migrations - Hints, tips and planning considerations

Table of Contents

Introduction:.....2
 Overview:.....2
Migrating your server.....2
 Audit of the existing server:.....2
 Preparing the new server:.....3
 Syncing data over from the old server:.....4
 Testing:.....4
 Prepare DNS:.....5
 Go-Live:.....5
 Tidy up:.....6
Conclusion:.....7

Introduction:

Migrating your websites from one server to another can seem like a daunting prospect, particularly if you have a large number of sites. You will usually want to keep downtime to a minimum, to avoid impact on any e-commerce sites, and ensure no data is lost during the transfer of sites. This can often seem a little bewildering, with a huge array of things that can potentially go wrong. But, with some careful preparation and planning, it can be a relatively easy to accomplish a stress-free server migration. The aim of this white-paper is to help you plan your server migration in a more structured manner, so as to anticipate and avoid many of the common problems that can blight an otherwise straightforward procedure.

Overview:

The basic steps to migrating from one server to another are:

- 1) Audit of the existing server
- 2) Preparing the new server
- 3) Syncing data over from the old server
- 4) Testing
- 5) Prepare DNS
- 6) Go-live
- 7) Tidy up

The following paper will look at each step in detail.

Migrating your server

Audit of the existing server:

Before we can prepare your new server, we need to know what versions of services and applications are currently running on your old server. If you have a web-based control panel installed, such as cPanel or Plesk, you will be able to find most of this information out within the control panel. However, in these examples, we will assume you are checking from the command line.

Probably the most important information relates to the four core aspects of the traditional LAMP stack –

Linux – what version of Linux is your server running?

This will usually be a Red Hat or Debian based operating system. On a Debian-based system this can be found by running:

```
cat /etc/lsb-release
```

On a Red Hat (or CentOS) system this can be found by running:

```
cat /etc/*release
```

Apache – what version of web server is your server running?

This can be found by running the command: `httpd -v`

MySQL – what version of MySQL is your server running?

This can be found by running the command: `mysql -V` (note the capital 'V')

PHP – what version of PHP is your server running?

This can be found by running the command: `php -v`

If your PHP sites use the **Pear** library, you will need to know what modules are installed. These can be found by running the command: `pear list`

Depending on what other applications you use, you might also want to know the versions of **Perl**, **Ruby**, **Python**, etc.

If your server also handles your email, it's worth checking what mail server software it runs – this will probably be **Exim**, **Sendmail**, **Qmail** or **Postfix**. You can usually find this information running the command `- netstat -tulpn | grep 25` - and check the name of the service listed next to port 25 in the output.

Lastly, if you have any **custom firewall rules**, particularly any IP addresses you have white-listed, you should make a note of these too. These can then be added to the new server's firewall.

Preparing the new server:

If the new server has a web-based control panel (GUI) installed – e.g. cPanel, Plesk, Webmin, etc – the existing sites can easily be re-created and then the data sync'ed across. The actual procedure varies depending on the control panel used, but it is best to create the sites in the following order:

- Create the website.
- Create the FTP users.
- Create the Email addresses.
- Create the databases.

It's best to try and keep the file and directory structure the same – particularly the document root – so that any absolute paths referred to in your code will work on the new server without changes being necessary. If

you are migrating between very similar servers, i.e. the same distribution of Linux and the same GUI, this will be much easier to do.

Some control panels, e.g. cPanel, include automated migration tools, which greatly simplify the procedure, as it will connect to the old server and create the sites, users and databases on the new server as it copies the data across.

Install any **PHP/Pear modules** found during the audit of the existing server.

Any **custom firewall rules** should now be added.

Syncing data over from the old server:

If you are not using an automated method of site migration, such as cPanel's migration tool, the best way of copying the data between servers is to use **rsync**, a popular file-copying tool that works over an SSH connection. It's main advantage is that once the data has been copied across, the rsync can be re-run multiple times to update copied data, as it only transfers the differences between the source files and what has already been copied – the makes updates much faster and uses less bandwidth. To make sure data is kept up-to-date, which will help to minimise the final sync time when the server goes live, it's often a good idea to setup a cron job to re-run the rsync of data at least twice day. Typically you will want to copy over the web folders, which will usually be somewhere like */home/username/public_html*, but always check your server configuration before copying any data across.

Databases can also be copied across using rsync (by sync'ing the */var/lib/mysql* directory), but it is much more effective to take **mysqldumps** of the databases. However, if you are using the **InnoDB** storage engine, it is recommended you always use mysqldumps to migrate your databases.

If the MySQL versions on the old and new servers are very different, it is often advisable to upgrade the version on the old server prior to the migration, to make the setup on the new server easier. Although, you minimise the impact to your clients, you might want to migrate the sites as they are, and then fix whatever coding issues arise during the testing phase.

Testing:

Once data have been copied across to the new server, they will need to be tested to make sure they work correctly before the DNS is re-pointed. If your server has the Apache userdir module installed (which is installed by default on cPanel servers), it is possible to preview sites using the URL format *http://server_IP_address/~username*. However, it is much more efficient to edit your local PC's hosts file. You will also need to do this if any of your sites use any hard-coded URLs.

Your PC will always try and resolve DNS requests locally, using it's hosts files, before looking for external

DNS records. So we can edit it to force domain names to resolve to different IP addresses. On Linux-based PCs, you can do this by opening a terminal and editing the `/etc/hosts` file. On Windows-based PCs it will usually be located in the `\Windows\system32\drivers\etc\` folder. The format is simply `<IP address> <Host name>` so, for example, to force requests for the domain `test.co.uk` to resolve to the IP address `10.0.0.1`, we would add this line: `10.0.0.1 test.co.uk`

In addition to the hard-coded URLs mentioned earlier, you will also need to change any hard-coded IP addresses within the code. You will generally be using `'localhost'` rather than a specific IP address, but in some instances of code the server's IP address might be directly referenced. This will need to be updated to use the new server's IP address before any code relying on it will work – assuming it is not meant to be using an external server for additional services. This most commonly occurs in the configuration files of Content Management Systems (CMS) and database connections.

Assuming most of the web sites appear to be working correctly, it is also a good idea to check the site's log files for errors, as there may be some unseen problems that will need fixing before the new server is made live.

Prepare DNS:

At least 24 hours before the final DNS changes are to be made, it is a good idea to reduce the Time To Live (TTL) of the DNS records for the domains being migrated. TTL's can be set to any value, but are commonly set to 1440 minutes (24 hours), so this will allow enough time for them to expire and the new values to take effect. Change the TTLs to 60 seconds. This is a low enough value to ensure that anyone with a cached DNS record is at worst going to experience 60 seconds connectivity issues, but high enough to ensure that the server is not going to be hit too hard by additional DNS requests.

If you are unsure which nameservers a domain uses, you can run a **whois** lookup from the command line – `whois domainname.com` – or use `www.whois.net`.

Go-Live:

Assuming the website and database data has been kept reasonably up-to-date using a `rsync/cron`, the amount of downtime experienced during the go-live can be kept relatively short.

Prior to switching off services on the old server, port forwarding is configured using **rinetd**. The forwarding rules are added to `/etc/rinetd.conf`, but the service is not made live yet. The standard rule format is: `<source IP> <source port> <destination IP> destination port`

To redirect all traffic on port 80 on the IP address `10.0.0.1` to port 80 on `10.0.0.50`, you would add a rule like this: `10.0.0.1 80 10.0.0.50 80`

Once the port forwarding is setup, we need to start stopping core services on the old server, prior to running a final sync of data. Turn off **ftp**, **email** and **httpd**, but make sure you leave **DNS** and **SSH** running. Until all DNS changes are made, the server will still be resolving DNS requests, although these will all soon be port forwarded to the new server. And if you turn off SSH you will not be able to log back onto the server!

Once services are stopped, we can run a final sync of the data. This just involves one last run of the **rsync**'s and **mysqldumps** – with the services off – so that no new data can be added while they're being copied across, to prevent data being lost.

Once the data has been copied, port forwarding can be started by typing the command: **rinetd**
Any errors reported will generally indicate which service **rinetd** is unable to bind to – list the running services by running **ps fax**, and use the **kill** command to stop any of the named processes that are still running. Then stop **rinetd** (as it will start even if it is unable to bind all ports), and then restart it.

Tidy up:

Once port forwarding is running, it is time to start checking that the sites and other services are working correctly on the new server.

Make sure you remove anything you added to your hosts file while testing, as this will automatically direct your requests to the new server's IP address, and will not allow you to test the migration.

There can be lots of things to think about during a server migration, so prior to the go-live, it is a good idea to think about which are your most important sites (usually the most profitable sites, but you may have other considerations too) and make a list of them. This makes post-migration testing easier, as you can check these key sites first.

Most post-migration problems are caused by coding issues – hard-coded file paths that do not match, references to the old server's IP address, or missing library modules for PHP or other scripting/programming languages. If your sites are checked thoroughly during the testing phase, most issues should already have been fixed, but there is always a chance that something might have been missed.

You should also remember to check the log files for error messages and test email by sending and then trying to access messages to a selection of pre-decided accounts.

Once you have confirmed all of the sites are working, you can start updating the DNS. If your server handles DNS itself, and your own nameserver addresses, this can be a relatively simple matter of just updating the **glue records** for the nameservers to resolve them to the new servers IP address(es). The process for doing this varies depending upon the domain registrant used to register the domain, but generally you will have a option to '*Change nameservers*' in your control panel – when prompted for the changes, keep the existing nameserver addresses, but change the IP addresses to match those used by the new server.

If the domains hosted on your server use a variety of registrant and their nameservers, you will need to log onto each individual control panel and update them all before port forwarding can finally be switched off on the old server.

Check that the DNS is resolving the domain to the correct IP address by doing host lookups from the command line – but not from either the old or new server, as by default these will respond with whatever they have listed in their own records. Remember to check both the www and non-www addresses, as the A records for each might be pointing to different IP addresses, e.g.

```
host -t a example.co.uk 10.0.0.1  
host -t a www.example 10.0.0.50
```

If we have just moved example.co.uk from a server on 10.0.0.1 to a new server on 10.0.0.50, this result would indicate that the www record had been changed, but the non-www record hadn't. If your server also handles the domains email, make sure you check the MX records too.

Once the DNS has been completely updated, the old server can be switched off.

Conclusion:

The process of migrating a server can be made much easier by some simple planning, and having a clear idea of how your existing server is configured. Working through the steps mentioned above, writing them down as a physical check list if necessary, should help remove much of the stress and confusion that can otherwise blight a smooth migration.