COMP150 Lab 9: LAMP

In this lab, you will install, configure, and test a LAMP (Linux, Apache, MySQL, PHP) web server.

There are a lot of steps in this lab. I suggest you check each step as you go along. If something isn't working, chances are you skipped a seemingly minor step. No minor steps here.

A useful command is **service –status-all**, which will show you all the running daemons.

Appache

CentOS comes installed with Apache by default. If you needed to install it (or upgrade it) the package name is `httpd`. It is, however, disabled by default. The first step is therefore to enable it so that it starts when the system boots. The Apache service is also called `httpd`, so to enable it:

1. Enable Apache to start at boot time: `sudo /sbin/chkconfig httpd on`

As always, this does not actually start the `httpd` service immediately. Before doing so, we need to modify the `httpd` configuration file, `/etc/httpd/conf/httpd.conf`, to set a few important options. Do the following:

1. Change directories to: `/etc/httpd/conf`
2. Use your editor of choice to edit the configuration file (remember to use `sudo`): `httpd.conf`
    a. Look through the file to get a feel for how the directives work, and spend some time reading the comments, particularly at the very top of the file.
    b. Scroll through until you find the top of `Section 2`, for the `'Main' server configuration`.
    c. Edit the `ServerAdmin` directive to use your email address.
    d. Edit the `ServerName` directive, setting it to the hostname of your VM and port 80. For example, my VM is named maverick.localdomain, so I set the directive to `maverick.localdomain:80`. Set yours appropriately for your hostname.
    e. Double check that the `DocumentRoot` directive is set to `"/var/www/html"`, which is the base directory where all web files will be hosted.
    f. Also double check that the second `Directory` section starts with `<Directory "/var/www/html">` to match the `DocumentRoot`.
3. Once that's done, save your changes and exit your editor.

Apache relies on finding your `ServerName` in DNS so it knows what IP address is associated with the web server. In a normal environment you would have an actual DNS hostname configured for your server and so this would not be an issue. Our VM hostname, however, is

not in any DNS server, so we need to add an entry to `/etc/hosts` so that Apache will find the mapping anyway. In fact, it's generally a good idea to have your hostname to IP address mapping listed in `/etc/hosts` for a server just in case DNS is unavailable. So:

1. Edit (with `sudo`): `/etc/hosts`
2. Add a line that looks like this: `127.0.0.1 www.hostname.com hostname`
    a. For example, I added this line:
    `127.0.0.1 maverick.localdomain maverick`
3. Save the file and exit your editor.
4. Check that it worked by pinging the hostname: `ping hostname`
    a. For example: `ping maverick.localdomain`
5. You should successful `ping` responses. Stop `ping` with `Ctrl-C` after a few pings have gone by.
6. **TAKE A SCREENSHOT** of the output of the `ping` command showing that you can ping by your hostname.

Now, it's time to start the web server and test it:

1. Whenever you make changes to `httpd.conf`, always check that you have no syntax errors by running: `/usr/sbin/httpd -t`
2. You should get a "`Syntax OK`" message. If it lists any errors, fix them before proceeding.
3. Start the webserver: `sudo /etc/init.d/httpd start`
4. Make sure you get the "`OK`" that indicates it started correctly.
5. Still in your VM, open up the web browser and go to `http://HOSTNAME`, where `HOSTNAME` is, again, your actual hostname.
6. You should get the `Apache 2 Test Page`.
7. **TAKE A SCREENSHOT** of the browser showing the address bar and the top of the Apache test page.

Apache is now up and running, so the next step is add a web page. We saw that `DocumentRoot` was set to `/var/www/html`, which means that is the base directory for the web server to serve files from. Let's add a simple HTML index to our web site:

1. Change directories to: `/var/www/html`
2. Edit (have to use `sudo`) a new file: `index.html`
    a. `index.html` is the default file that Apache will look for in a directory.
    b. Here's a simple HTML file you can use if you've never done HTML before:

```
<html>
<body>
  Hello world!
```

```
</body>
</html>
```

3. Save the file and exit your editor.
4. Go back to the VM web browser and refresh your website.
5. You should see a nearly blank page with "Hello world!" displayed.
6. **TAKE A SCREENSHOT** of the browser showing the address bar and the web page.

PHP

Next up, it's time to add PHP to the web server. In CentOS, the PHP package is named `php`. Note that PHP is not a separate service on your system. It is simply an extension (called a module) for Apache that gives Apache the ability to run PHP scripts on the server. The purpose is to provide a mechanism for creating dynamic web pages, which is what PHP does. In fact, when a PHP-enabled server encounters a `.php` file, it calls the `php` program which runs the PHP script and produces HTML output that is ultimately served to the client. You can also run PHP scripts directly with the `php` program (like running a shell script). In either case, the first step is to install PHP:

1. Install PHP: `sudo yum install php`
2. In order for Apache to see PHP, you have to restart it: `sudo /etc/init.d/httpd restart`

Installing PHP adds configuration files that you should at least be aware of. CentOS does a pretty good job with the default configuration, but most sites have to do at least a little tweaking of some options to get exactly what they want. The PHP configuration file for Apache is `/etc/httpd/conf.d/php.conf`, and it is used to tell Apache how to handle PHP files. The actual PHP configuration file for modifying PHP behavior is `/etc/php.ini`. Take a few minutes to look them over to get an idea of the syntax (different for Apache and PHP). To test PHP, you need a PHP script somewhere inside the `DocumentRoot` directory. It is customary to use the `phpinfo()` function to test a new PHP install as it tells you everything about both PHP and Apache. To do that:

1. Make sure you're still in the `/var/www/html` directory.
2. Edit (again, with `sudo`) a new file: `phpinfo.php`
    a. You can name it whatever you want, so long as it ends in `.php` (that's how Apache knows to treat this as a PHP script rather than an HTML file or something else).
    b. The file should contain just one line, which is a call of the `phpinfo()` function: `<?php phpinfo(); ?>`

c. The `<?` and `?>` syntax denotes a PHP section in the file. Many PHP files are actually combinations of both PHP and plain HTML and the `<?` and `?>` tags are used to mark pieces of PHP code.

3. Save the file and exit your editor.
4. Back in your VM browser, point it to: http://HOSTNAME/phpinfo.php
5. You should see the `phpinfo()` page that contains a series of tables detailing all of the PHP configuration for your VM.
6. **TAKE A SCREENSHOT** of the browser showing the address bar and the top of the `phpinfo()` page.

MySql

At this point, your web server supports PHP scripts for dynamic HTML output. The final piece to the LAMP model is the MySQL database. Of course, some developers prefer other databases (Oracle, PostgreSQL, NoSQL databases, etc), but MySQL is the norm for many websites. The MySQL client programs are installed by default in CentOS, but not the server or other important pieces. The client package is simply called `mysql`, the server package is `mysql-server`, and the PHP-MySQL interface package is called `php-mysql`. The PHP-MySQL interface is an extension to PHP that allows PHP scripts to easily access MySQL databases that are hosted on the same server or other servers. For security and convenience reasons, the SQL server and web server are often on the same machine. However, very large sites typically have more sophisticated infrastructures that allow them to scale up performance-wise. We'll stick with easier approach here. So, we need to install both the server and PHP-MySQL interface packages:

1. Install the MySQL server and PHP-MySQL interface :
   ```
   sudo yum install php-mysql mysql-server
   ```
2. In order for PHP to see the new MySQL interface, Apache needs to be restarted:
   ```
   sudo /etc/init.d/httpd restart
   ```

Like PHP, MySQL comes with a few different configuration files that are important to know about. The PHP-MySQL interface configuration is in `/etc/php.d/mysql.ini`, but this only includes the one line telling PHP where to find the MySQL interface. All options affecting the MySQL interface are actually in the main PHP configuration file (`/etc/php.ini`) in the `[MySQL]` section. The MySQL configuration file itself is `/etc/my.cnf`. There are a number of different fields you want to modify or add to both the PHP and MySQL files, but again CentOS does a good job of configuring a workable default environment. See the PHP and MySQL documentation for specifics.

MySQL is a separate service in Linux, as it can be used for much more than just web hosting. The service is called `mysqld` (the MySQL daemon). As usual when installing a new service, we

need to be sure it's enabled to run when the system boots and actually start the service so we can use it right away:

1. Enable the `mysqld` service to start at boot time: `sudo /sbin/chkconfig mysqld on`
2. Start the `mysqld` service: `sudo /etc/init.d/mysqld start`
3. You also need to do some initial setup for MySQL such as setting the MySQL root password. This is done with the special `mysql_secure_installation` command: `sudo /usr/bin/mysql_secure_installation`
    a. This will prompt your for some information to do the initial configuration.
    b. The current root password is blank, so just hit enter when prompted for it.
    c. You need to choose a new password, so say yes when prompted to change the root password and then supply a new password.
    d. All the other options remove testing databases and restrict root access to the local machine, which are all sensible precautions. So, say yes to each other prompt.

Now that MySQL is installed, you need to verify that PHP sees the MySQL interface:

1. Go back to your VM browser and reload the `phpinfo.php` page.
2. As you scroll down through the page you get to a list of extensions that PHP currently has enabled (in alphabetical order).
3. Scroll down until you see the `mysql` section, which should contain two tables describing the relevant MySQL options in PHP. If you don't see it, double check your configuration files, restart `httpd` again, and reload the page.
4. **TAKE A SCREENSHOT** of the browser that shows the `mysql` section in the `phpinfo()` output.

You now have a functioning LAMP server! There are two additional features of Apache that are used on a regular basis that you also need to know about: virtual hosts and SSL.

Virtual hosts

Virtual hosts are a way to allow one web server to serve out multiple web sites. There are two types of virtual hosting: name-based and IP-based. We'll use name-based here because it's easier, but there are some limitations to the approach. Newer versions of Apache and SSL (i.e., newer than are available through the CentOS repositories) remove most of the limitations, but many sites that use virtual hosts still fall back on the IP-based approach. You can configure that for some extra credit (see the end of the lab). All of the configuration takes place in the final section of the `httpd.conf` file:

1. Edit (with `sudo`): `/etc/httpd/conf/httpd.conf`
2. Go to the end of the file and look for the `Virtual Hosts` section.

3. Add the following line: `NameVirtualHost *:80`
    a. This tells Apache that you are going to be using name-based virtual hosting on port 80.
4. Next, you have to add a `VirtualHost` section for each virtual host. We'll start with just one.
    a. You have to specify at least the `DocumentRoot` and `ServerName` directives, but you generally want to specify a few other things include host-specific logging.
    b. Here is a template you can use:

```
<VirtualHost *:80>
    ServerAdmin EMAIL
    DocumentRoot /var/www/HOSTNAME
    ServerName HOSTNAME
    ErrorLog logs/HOSTNAME-error_log
    CustomLog logs/HOSTNAME-access_log common
</VirtualHost>
```

    c. Clearly, you need to replace the `EMAIL` and `HOSTNAME` values with your actual email address and the hostname of your system.
    d. Note that using this template every virtual host will have a separate base directory for files in `/var/www/HOSTNAME`.
5. Save the file and exit your editor.
6. Now you need to actually add the new directory for your virtual host and add an HTML file to test it.
7. Make the directory: `sudo mkdir /var/www/HOSTNAME`
8. Edit (with `sudo`): `/var/www/HOSTNAME/index.hml`
    a. Add HTML code like the Hello World example above but change the actual Hello World line to something else, like "Welcome to HOSTNAME".
9. Save the file and exit your editor.
10. Double check the syntax of `httpd.conf` before restarting by running: `/usr/sbin/httpd -t`
11. Restart `httpd`: `sudo /etc/init.d/httpd restart`
12. Finally test it out by pointing your VM browser at http://HOSTNAME
13. You should see your new HTML output (not Hello World). If not, double check your settings, restart `httpd`, and try again.
14. **TAKE A SCREENSHOT** of the browser window showing the address bar and the correct output.

Of course, only having one virtual host is not much use (you could just use the main server settings instead). So, let's add a second virtual host. To do that, we need another DNS hostname mapped to the same IP address on your system. Let's use a nice generic name for the new hostname: www.comp150.private. To add it as a virtual host:

1. First, you need to add the new hostname to `/etc/hosts` (and/or your DNS server in a real production environment). So, edit (with `sudo`): `/etc/hosts`
2. Find your `127.0.0.1` line and add another hostname to the end of the line: www.comp150.private
    a. The line should now be: `127.0.0.1 www.comp150.private comp150`
    b. The second value simply says that both comp150 and www.comp150.private map to 127.0.0.1
3. Save the file and exit your editor.
4. Now add the `VirtualHost` entry to `httpd.conf`. Edit (with `sudo`): `/etc/httpd/conf/httpd.conf`
    a. Add a new section below the first `VirtualHost` section from before.
    b. Use the same template as above, but now `HOSTNAME` is `www.comp150.private` rather than your default system hostname.
5. Save the file and exit your editor.
6. Now you need to actually add the new directory for your virtual host and add an HTML file to test it.
7. Make the new virtual host directory: `sudo mkdir /var/www/comp150`
8. Edit (with `sudo`): `/var/www/comp150/index.hml`
    a. Add HTML code like the Hello World example above but change the actual Hello World line to something else, like "Welcome to www.comp315.private".
9. Save the file and exit your editor.
10. Double check the syntax of `httpd.conf`: `/usr/sbin/httpd -t`
11. Restart `httpd`: `sudo /etc/init.d/httpd restart`
12. Finally test it out by pointing your VM browser at http://www.comp315.private
13. You should see your new HTML output (not Hello World). If not, double check your settings, restart `httpd`, and try again.
14. **TAKE A SCREENSHOT** of the browser window showing the address bar and the correct output.

SSL

Finally we will install SSL support for Apache for encrypting web pages. The default SSL libraries in CentOS and most other OSes are the OpenSSL ones, which are installed already (package name `openssl`). All we need to install the glue between OpenSSL and Apache, which is the SSL module for Apache, called `mod_ssl`:

1. Install the SSL module: `sudo yum install mod_ssl`
2. Restart Apache so that it sees the new module: `sudo /etc/init.d/httpd restart`

The Apache SSL configuration file is `/etc/httpd/conf.d/ssl.conf`. It adds all the necessary Apache options to get SSL working, even including a default SSL certificate for the web server. To test it:

1. Point your VM browser at: https://HOSTNAME
2. You should get told that this is an untrusted connection because the identity of the site can't be confirmed. This is expected because the default SSL certificate is not validated by an actual certificate authority.
3. Go through the steps to add an exception and then you should see the same index page for the original "main" server, which was the Hello World page.
4. **TAKE A SCREENSHOT** of the browser showing the address bar with the correct https address and the correct web page.

Note that if you try to browse to https://www.comp315.private and accept the certificate then you will also get the same Hello World page rather than the actual www.comp315.private main page. This is because this version of Apache does not support SSL for name-based virtual hosts.

For a production server, you obviously need to get a valid SSL cert that has been signed by a recognized certificate authority (CA). This involves getting generating a cert and signing request and giving those to a CA (along with some money between $0 and $1000s). The CA then returns to you the signed certificate. All of the steps are detailed by the CAs.