# KIT304 Server Administration and Security Assurance

# Tutorial 9

---

## Goal

This tutorial marks the start of our third module. We will move away from the earlier platform-specific focus, and start to look at applications – specifically web and database servers, and later, container services.

## Introduction

The Internet is one of our most transformative inventions. The availability of information and the speed through which communication can occur over the Internet is unprecedented. One of the key developments that fostered much of this is the World Wide Web, which rides atop the HTTP protocol. In this tutorial you will be enabling an http server process on the CentOS operating system to enable it to receive and respond to http requests. This turns a Unix server into a web server.

On Unix, you'll be learning about **httpd** – the Apache webserver. According to Netcraft, Apache had 34% of the web server market share in December 2018 for the top 1 million busiest sites, followed by Nginx at 26%. Microsoft's web server, IIS, comes in at 9%, although this number jumps to over 40% when considering all sites.

Apache (and Nginx) are popular because they're designed for high performance, and are free and open source projects that are readily available and require very little configuration to get started.

The majority of today's web sites don't operate with just a web server process. Most sites are dynamically generated, and use a scripting language (like **PHP**) to pull content from a database. The database you will be using today is a variant of **MySQL** called **MariaDB**.

MySQL (or MariaDB), paired with Apache, is the most commonly used combination of database and webserver on the web. The combination is often referred to as the LAMP stack – meaning Linux, Apache, MySQL and PHP. (You can probably deduce what the MAMP and WAMP variants run on!)

## Package Management

Most modern Unix based systems use a *package manager* to keep track of what software is currently installed on the system, and to enable you to install new software. Instead of having to browse packages on a website and then manually download them, the package manager lets you download

and install software and its dependencies from the command line or a graphical interface. Most programs depend on other packages to run – the package manager keeps track of these dependencies, and will install or update the other packages as necessary when you install new software.

There are a number of package management systems in common use. Debian-derived systems (including Ubuntu) use **apt** running on top of **dpkg**, while Red Hat-derived systems (including CentOS and Fedora) use **dnf** or **yum** running on top of **rpm**. Other Linux package managers include **homebrew**, **zipper**, and **portage**.

To save you the time of learning how to use the package manager, the additional software that you will use during this tutorial has already been installed on the CentOS VM, but you will still need to configure it before you can run it. If you have a CentOS installation on your own hardware, there's an appendix at the end of this tutorial that details the packages you need to install should you want to set up a web server on that system.

## Activity

1. Launch the CentOS virtual machine, log in as the student user, and assign the machine an IP address in the **`192.168.1.x`** subnet. Next, open a Terminal window, and become the root user.

2. As stated earlier, the additional software you'll need to use web services on CentOS has already been installed. It is not, however, currently running, and it hasn't been fully configured. To check the current status of the http server process, enter the command **`service httpd status`**. This should report that it is currently inactive. To turn the service on, enter the command **`service httpd start`**, and then check the status again. This time, you should see that not only is it running, but that is has started multiple instances of itself to handle incoming requests.

3. Open Firefox and enter **`localhost`** in the address bar. You should see a simple Apache page that shows that the web server is now running.

4. While the **`httpd`** process is currently running, this wouldn't be the case of the VM was shut down and then restarted. To configure CentOS to start a service automatically after boot, you use the **`chkconfig`** command. To have Apache start after reboot, enter this command:

   ```
   chkconfig httpd on
   ```

   You can use the same command to configure other daemons to start automatically, and you can use **`chkconfig httpd`** to see the current auto-startup status of the httpd daemon.

5. Launch **Windows 7** and assign it an IP address in the **`192.168.1.x`** network. Open a web browser and enter the CentOS IP address in the address bar to visit your CentOS web server. You should see that the page doesn't load – this is because the CentOS firewall is turned on, and is blocking

incoming http requests by default. To enable those requests to reach the **httpd** process, enter the following commands in the CentOS terminal window to reconfigure the firewall:

```
firewall-cmd --permanent --zone=public --add-service=http
firewall-cmd --permanent --zone=public --add-service=https
firewall-cmd --reload
```

Now try to reload the webpage in **Windows 7** – this time, you should see the page load as it did locally on the CentOS browser.

6. The Apache web server is configured, by default, to serve web content from the directory **/var/www/html/**. HTML files you create here will be directly viewable in a web browser, and PHP scripts should be executed and their output sent to the web browser. If there is no content in this directory (we have a simple **index.html** in our case), some installations of Apache will serve a default Apache "Testing 123…" page.

   To confirm that you can create new web content files, and that the PHP scripting engine is configured and working, in your terminal session change to the content directory and then create a file called **info.php** with the following script:

```
<?php
   phpinfo();
?>
```

   Next, try to view this page in a web browser – note that you have to specify the filename as part of the URL, so for example on CentOS you would go to **localhost/info.php**. You should see a lot of information about the PHP scripting engine. This simple tool is sometimes used to debug issues affecting PHP sites – for example, letting you see which version of PHP is installed, and how it is configured.

   > ➢ Leaving **phpinfo()** in a publicly visible script on a web site also advertises to attackers where potential weaknesses are in your system.

7. With PHP set up, the next step is to configure the database. You have probably used MySQL before, or at least have heard of it. You will be using the MariaDB fork of the MySQL project, which was started shortly after Oracle bought Sun Microsystems (who had ownership of the MySQL project at the time).

   MariaDB is generally considered to be faster than MySQL, and is largely compatible with it, although in business environments MySQL is more likely to be used since there is strong enterprise support available.

On Linux, the MariaDB executable binary is known as **mariadb**. Using the same basic commands that you used to start the Apache server, start the MariaDB daemon, and configure the system so that the daemon will also start automatically on system boot. Write the two commands you used to do this here:

_____

_____

8. Although the database is now running, it is currently a default installation, and you need to run a configuration step. Both MySQL and MariaDB use the same command to set up their databases and establish some basic security controls. To do this enter the command:

   **mysql_secure_installation**

   > ➢ Note: the setup process refers to a **root** user. This is the root (administrative) user for the ***database***, not for the host (CentOS) system.

   As the script runs, be sure to do the following:

   - when prompted for the initial root password, press enter (since it is not set)
   - apply a password to the root account – make a note of what this is, as you'll need it later
   - remove anonymous users
   - disallow remote root logins
   - remove the test database and access to it, and finally
   - reload the privilege tables

9. Database management through the command line is tedious, and most developers use more advanced tools, typically with a graphical interface. A popular tool for managing MySQL and MariaDB databases is written in PHP and therefore runs via a web browser. You may have heard of it, or even already used it – it is called **phpMyAdmin**.

   To configure phpMyAdmin, open and edit the file **/etc/httpd/conf.d/phpMyAdmin.conf**. Find the first **Directory** section and modify it to match the following. Note that the options are case sensitive, and that lines that you don't need to change are shown in grey:

```
<Directory /usr/share/phpMyAdmin/>
   AddDefaultCharset UTF-8
   <IfModule mod_authz_core.c>
         # Apache 2.4
         <RequireAny>
               Require all granted
```

```
        </RequireAny>
    </IfModule>
    <IfModule !mod_authz_core.c>
        # Apache 2.2
        AllowOverride None
        Options None
        Allow from All
        Require all granted
    </IfModule>
</Directory>
```

When you're done, save your changes, and exit the editor. Since this configuration file is used by Apache, apache needs to be made aware of changes to it. You do that with this command:

```
service httpd reload
```

- If you made any errors while editing the configuration file, there's a good chance that Apache won't start. Go back and check your edits carefully, and ask your tutor for help if you can't resolve the issue.
- If you do see an error while trying to reload Apache, it tells you to enter this command: **`systemctl status httpd.service -l`** – (that last character is a lower-case "L") which will show you more information about the error, but you have to search carefully through the status output to actually find it. It helps to maximise the terminal window.

Now you should be able to access **phpMyAdmin** from both the local browser, and the browser running on the **Windows 7** virtual machine. The URL from the local browser would be **`localhost/phpMyAdmin`**. Once you've loaded the page, try to log in as the root database user with the password you established back in step 8.

## Virtual Hosts

Many web sites on the internet are hosted on shared infrastructure. This allows a single physical server to appear to be multiple distinct web sites. This is possible because the task of serving files over the http protocol generally presents very little load to the CPU, and so it is considerably more efficient to host multiple sites on the one system.

Such sites are called *virtual hosts*, and they are supported by default in Apache. Virtual hosts are able to be distinguished from each other by virtue of the fact that each of them has a different domain name – even though each of these domain names all resolve to the same IP address.

Your next task will be to create a virtual host on CentOS.

Recall that the default web content folder for Apache is **/var/www/html/**. When virtual hosts are implemented, the content area for each of them is usually located inside an associated directory in **/var/www/** that is created by the system manager – thus, if you're developing a virtual host for the domain **networks.com**, you might create the directory **/var/www/networks.com** to store its HTML and associated files.

10. In the terminal, change to **/var/www**, and create a new directory named **networks.com** (you could call it unique, but you need to use that same name later in your configuration files). Then change to this new directory, and create another directory inside called **html** – this will be where your HTML and PHP scripts will be stored.

11. Inside the **html** directory create a file called **index.html** or **index.php** and place some text inside it to state that this is your domain. Don't worry if it is not valid HTML or PHP, it will still appear in a browser.

12. Enter the following command so that your "normal" account (student) will own the domain – this means you can delegate control of the content on the web site to someone other than the root user.

    ```
    chown -R student:student /var/www/networks.com/html
    ```

    Next, ensure that the web server process can read all files in the entire web tree with the command:

    ```
    chmod -R 755 /var/www/
    ```

The next step involves creating an Apache configuration file that tell the server where the content for the new virtual host resides, and where it should store the access log and error log files for the virtual host.

By convention, in modern Apache installations, the configuration files for virtual host configuration files are stored in the directory **/etc/httpd/sites-available**, while another directory named **/etc/httpd/sites-enabled** is used to store **symbolic links** to the configuration files for sites that are meant to be "active" at any given time. In such systems "enabling" a virtual host means creating a symbolic link in the **sites-enabled** directory to that site's configuration file in the **sites-available** directory, while "disabling" a site involves deleting that symbolic link. On Apache installations on Debian-derived Linux systems (such as Ubuntu) there are two simple commands to enable and disable sites (and thus, create or delete the symbolic links). On CentOS, there aren't any equivalent matching commands, and so enabling a web site involves using the **ln -s** command to create the symbolic link.

Creating symbolic links tends to be confusing to many students, so for these tutorials, we'll take a simpler approach. We'll just store the configuration files directly inside **/etc/httpd/sites-enabled** and ignore the use of symbolic links.

Follow these steps to create your first virtual host configuration file:

13. In the terminal, as root, edit the file **/etc/httpd/conf/httpd.conf** (the global Apache configuration file) and add the following two lines at the bottom:

    ```
    NameVirtualHost *:80
    IncludeOptional sites-enabled/*.conf
    ```

    This tells Apache to look in the **sites-enabled** directory for additional configuration files.

14. Change to **/etc/httpd** and create the directory **sites-enabled**.

15. Change into the new **sites-enabled** directory, and create a new configuration called **networks.com.conf** with the following content:

    ```
    <VirtualHost localhost:80>
         DocumentRoot /var/www/html
    </VirtualHost>
    <VirtualHost *:80>
         ServerName www.networks.com
         DocumentRoot /var/www/networks.com/html
         ServerAlias networks.com
         ErrorLog /var/www/networks.com/error_log
         CustomLog /var/www/networks.com/access_log combined
    </VirtualHost>
    ```

    The first **VirtualHost** directive enables the default service to continue to be available. The second configures the virtual host when accessed as **networks.com** or **www.networks.com**.

16. The **VirtualHost** directive above references several log files where the Apache server will store access request details (for analytics) and error details (helpful when debugging, for example, PHP script issues). These files don't yet exist, and so you need to create them and establish an appropriate SELinux security context for them.

> ➢ SELinux is an acronym for Security Enhanced Linux. It is a security feature of the Linux kernel, and lets you limit the files that daemons (like Apache) can access and what they can do to them. This helps reduce the impact of attacks against poorly configured daemons, and daemons that might be the target of attacks. SELinux is a default install in CentOS systems.

Create the log files and set their security context with the following commands:

```
cd /var/www/networks.com
touch error_log
touch access_log
chcon --reference /var/log/httpd/error_log error_log
chcon --reference /var/log/httpd/access_log access_log
```

> ➢ Note the use of the **chcon** command. Rather than explicitly specify what security context you want to use for the new log files, use the **--reference** option to copy the context from two existing files that already have the security context you require.

17. Since you've changed Apache's configuration files, you now need to reload Apache so it picks up the changes. Do that with the command **service httpd reload**. Remember if you get any error messages, try to find and diagnose them as described back in step 9. If you need help, ask your tutor.

18. Your new virtual host should now accessible at **networks.com**, but if you enter this into the browser address bar, it won't work. That's because domain names have to be translated to IP addresses by the DNS service, and it will fail for multiple reasons:

    - you haven't configured a DNS server entry on either of your VMs
    - even if you had, you're VMs are not connected to the public internet, so lookups would fail
    - you don't actually own the **networks.com** domain, so even if you were connected to the internet, the resolved IP address would not be that of your server.

    To work around this, you can use a useful TCP/IP feature, which is the local **hosts** file. This provides a simple hard-coded mapping between IP addresses and host and domain names that is consulted *before* DNS. If you edit the local system's hosts file, and add an entry for your **networks.com** domain, you won't need a DNS server to see if your virtual host is working.

    On Unix systems, the hosts file is at **/etc/hosts**. To edit this file, you need to be the root user.

    On Windows systems, it is at **C:\Windows\System32\drivers\etc\hosts**. To edit it:

    - locate **NotePad** in the **Start** menu
    - right click on it and select **Run as Administrator**
    - in Notepad, go to the **File** menu and use the path above to find and open the file – you will also need to change the file name filter from "Text Documents (*.txt)" to "All files"

Entries in the **hosts** file are single lines with the IP address of the server, a space or tab, and then the host or domain name, for example:

```
192.168.1.1 networks.com
```

You can list multiple host/domain names for the one IP address – just separate each of them with another space.

After editing your hosts files on both CentOS and Windows, enter the domain name into the address bar of both system's browsers, and you should see the web site for your new virtual host.

## Content Management Systems

Your final activity today will be to set up a WordPress website on your server, and give it a domain name as well. WordPress is an incredibly popular content management system. According to **w3techs.com**, it is used by 33% of ***all*** websites and over 60% of all websites that are implemented with a CMS.

19. You need to start by creating your own domain name for your site, and creating a new directory (in **/var/www**) and a new virtual hosts configuration file (in **/etc/httpd/sites-enabled**) for it, just like you did for your **networks.com** domain. That means repeating steps 10-12, and 15-17 of this tutorial for your new domain. If you can't think of a domain name yourself, use **blog.com**.

> ➢ Before going any further, ensure that your new domain actually works by testing it in a browser, **then delete the index.html or index.php file that proves it works**.

20. In the **Downloads** folder of the **student** account you will find a file called **latest-en_AU.tar.gz**. This is a download of WordPress website bundle (it's may not technically be the *latest* version of WordPress, but that doesn't matter for today's tutorial). As the student user, copy it into your new domain's **/var/www/*domainname*/html** folder. If you get a **Permission denied** error, you likely forgot the **chown** command at step 12 for **/var/www/*domainname*/html**.

21. If not already there in the terminal, change to the content directory, and then extract the WordPress bundle using the following command:

```
cd /var/www/domainname/html
tar --strip-components=1 -xvf latest-en_AU.tar.gz
```

22. WordPress needs a database in which to store its content, and you need to set this up before you can configure WordPress. From the command line open MariaDB using **`mysql -u root -p`** and when prompted for the password, enter the one that you created back in step 8.

    Next, you need to create a database user for WordPress (select your own values for ***`username`*** and ***`password`*** – you'll need them soon):

    > ➢ Note: as you enter each of the SQL commands below, you should see the **QUERY OK** response. If you don't, go back and recheck your command(s).

    ```
    CREATE USER userName IDENTIFIED BY 'password';
    ```

    Next, create a database for the content management system. Again, choose any name you want here for the database. If you can't think of one, just use **`wp`** (for WordPress).

    ```
    CREATE DATABASE databaseName;
    ```

    Next, make the user you just created the owner of the database:

    ```
    GRANT ALL ON databaseName.* TO userName@localhost IDENTIFIED BY 'password';
    ```

    Finally, tell the database to clear its cached privileges (so the new ones you've created can be used) and quit the command-line tool:

    ```
    FLUSH PRIVILEGES;
    EXIT;
    ```

23. Next you need to create a configuration file for WordPress that provides the name of the database it can use, as well as the username and password it can use to access that database. There's already a sample configuration file in the WordPress download you can base this on. If you're not already there, in the terminal change to **`/var/www/`*`domainName`*`/html`** and copy the configuration template to what will become the real config file as follows:

    ```
    cp wp-config-sample.php wp-config.php
    ```

    Now edit **`wp-config.php`**. You'll see that the first group of lines are comments, and then you'll see a series of **`define`** function calls. Edit the first three of these ('**`database_name_here`**', '**`username_here`**', and '**`password_here`**'), replacing the quoted values with the database name, user name, and password that you set up in the previous step, then save your changes and quit the editor.

24. Now visit your new domain in a web browser on either CentOS or Windows 7. It should redirect to a page called **`install.php`** asking you for a site title. If instead it asks you about database information you have not completed the previous steps correctly.

    Complete the form as required (noting what username and password you use, as these are required to administer the site afterwards) and then press the **Install WordPress** button. You may see an SELinux security alert as WordPress tries to send a confirmation email to you, but you can dismiss this.

    The next page you see will be the WordPress administration page. This is where you can create new posts and pages, customise the web site theme, and so on. Experiment with the site, and be sure to test the results in a browser that isn't logged in as the WordPress admin user so you can see what the site looks like to regular users.

## Conclusion

In this tutorial you configured a webserver, some simple virtual hosts, a database, and then a CMS – in very few steps. In the next tutorial you will be undertaking many of the same tasks on Windows 2016.

## Skills to Master

To successfully complete this tutorial, you must have mastered the following skills:

- configuring a CentOS system to start specific services on a reboot
- opening the CentOS firewall to allow external hosts to access the local http web server
- creating entries in a Unix and Windows **`hosts`** file to simulate a DNS registration
- configuring Apache's main config file to support named virtual hosts, and to read those host's configurations from the **`sites-enabled`** subdirectory of **`/etc/httpd`**
- creating a web content directory for a new Apache virtual host, including setting up its **`.conf`** file, its content tree under **`/var/www`**, its access and error log files, and its **`/etc/hosts`** entry
- securing a new MySQL/MariaDB database installation
- using a PHP "info" file to confirm PHP is installed and operational, and examine its specific configuration

You could be required to demonstrate any of these skills in this module's practical exam.

## Appendix- Adding the Required Packages to CentOS

If you have your own CentOS installation, you can use the following commands to install PHP, MariaDB, and phpMyAdmin, and download the latest release of WordPress.

Firstly you need to get the Extra Packages for Enterprise Linux (EPEL) versions that CentOS uses. Without this, some of the PHP modules won't be found in a valid state for CentOS to use.

```
dnf install epel-release
```

Next, install PHP:

```
dnf install php php-opcache php-gd php-curl php-pdo php-pecl-zip\
php-json php-mbstring php-mysqlnd
```

You can confirm that PHP is installed by entering the command **`php –v`**

To install MariaDB, use:

```
dnf install mariadb-server mariadb
```

If you want to manage your databases graphically:

```
yum install phpmyadmin
```

And finally download WordPress with:

```
wget https://en-au.wordpress.org/latest-en_AU.tar.gz
```

The state of the machine now matches the CentOS VM you use in the lab, as at the start of this tutorial.