[WSE 380] Technical Foundations of a Startup
Secure Web Application (WordPress) Management

Instructors: Johnny So and Professor Nick Nikiforakis

Contents

1 Course Materials 3

2 Installing an SSH Client 3

3 Connecting to a Remote Machine 3
3.1 SSH Keys 3
3.2 SSH Config 4

4 Linux Command Line Basics 5
4.1 Command Line Anatomy 5
4.2 The Working Directory 5
4.3 Moving Around the File System 6
4.4 Creating, Reading, Moving, Copying, and Deleting Files and Directories 6
4.5 Editing Files 7
4.6 Downloading Programs 7
4.7 Downloading Code Using Git 7
4.8 Running Programs 7
4.9 Sudo 8
4.10 Tab Completion 8

5 Setup and Administration of a Server 8
5.1 SSH 8
5.2 UFW (Uncomplicated Firewall) [4] 9
5.3 Fail2Ban [4] 10

6 Installing Docker and Docker Compose 10

7 Web Application Infrastructure 12

8 Setup: Docker Compose Services 12
8.1 Environment Variables 12

9 Setup: WordPress Installation via the Web Interface 13

10 Setup: DNS 13
10.1 Domain Registration 13
10.2 DNS Name Servers 14
10.3 Updating the Nginx/WordPress Configuration 14

11 Setup: TLS Certificates 14
11.1 Certificate Request 14
11.2 Updating Docker Compose 15

12 Customization: Content 16

13 Customization: Styling and Themes 16

14 Customization: Plugins 16
14.1 Security - Wordfence 16
14.2 Performance - LiteSpeed Cache 16
14.3 Analytics - Google Site Kit 16

14.4 Forms - WPForms 16

15 Monitoring: Resource Usage 17

16 Monitoring: Uptime 17

17 Cloudflare 17

18 Analysis of Server Logs 18

19 Conceptual Review 18

Welcome to WSE 380: Technical Foundations of a Startup! Today, we will be setting up your devices with an SSH client to connect to a virtual machine that we will be running experiments on.

1 Course Materials

All of the course materials are stored at the following GitHub repository: https://github.com/jso123450/wse380-webapp.git.

2 Installing an SSH Client

SSH (Secure SHell) is a network protocol that enables a computer to establish a secure connection to a remote computer that is running an SSH server. To connect over SSH, we need to make sure we have an SSH client on our devices. New versions of Windows (10 and later) and MacOS come with built-in SSH clients, so there is no need to install a separate client; Windows (10 and later) users can use the Windows Terminal application whereas MacOS users can use the `Terminal` application. However, if you are running an older version of Windows or if you would like to install a separate client, you can install MobaXTerm (recommended) or PuTTY.

3 Connecting to a Remote Machine

The default syntax for the command to establish an SSH connection is `ssh -p 22 user@address`, where

- `ssh` is the name of the program,
- `-p 22` is a flag `-p` with a value of 22 (for SSH, this flag indicates the port),
- `user` is the username of the remote account, and
- `address` is an address to the remote machine. This can be a domain name (e.g., `google.com`) or an IP address (e.g., `1.2.3.4`) — in our case, we will be using IP addresses to connect to our remote machines.

As an example, if you are trying to establish an SSH connection to a remote machine with IP address `1.2.3.4` on port `5678` with a user account of `foobar`, you would execute the following command in your terminal:

```
user@local-machine:~$ ssh -p 5678 foobar@1.2.3.4
```

If you are authenticating with a username and password, the SSH client should then prompt you to enter a password:

```
foobar@1.2.3.4's password:
```

Please refer to the email you received with the IP address and login credentials, and make sure that you can establish an SSH connection to it before our next meeting. If you have trouble connecting to your machine, please refer to this guide on how to connect to a machine over SSH or send me an email.

3.1 SSH Keys

We will be connecting over SSH using a public and private key pair to eliminate the use of a password. You can read more about them on this DigitalOcean tutorial or on this GitHub documentation. Let us open a terminal and enter the following:

```
user@laptop:~$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

For the following prompts that ask about the file location and key passphrase, you can hit `Enter` without specifying anything to accept the default values (the file location will be "~/.ssh/id_ed25519" and there will be no passphrase).

After you have done this, please do the following:

```
user@laptop:~$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519 <random_string> <your_email@example.com>
```

Please copy the output and paste it in a reply to the email with your login credentials.

3.2 SSH Config

To facilitate ease of working with remote machines, we will configure a SSH configuration file and assign a name to your own machine. The default SSH configuration file is located at “∼/.ssh/config“, where ∼ is the home directory of your user. Let us open a terminal and enter the following:

```
user@laptop:~$ nano ~/.ssh/config
```

This should open a new (or edit the existing) file at this location using the basic `nano` file editor. Copy and paste the following into the terminal:

```
Host wse380
    HostName <ip address from email>
    Port <port from email>
    User <username from email>
```

Now press *Ctrl+x, y, Enter* (hold down the *Ctrl* key and then press *x*, let go of *Ctrl*, then press *y* and *Enter*). This should prompt `nano` to close and save the file. Now, you should be able to type `ssh wse380` instead of `ssh -p <port> <username>@<ip address>`!

Welcome to the first hands on session of WSE 380: Technical Foundations of a Startup! Today, we are going to start by talking about the Linux command line and finish by setting up our own honeypot. I hope when this session is finished, you will not only know how to setup a honeypot but also have a much better understanding of the Linux command line and how to accomplish tasks using it.

4 Linux Command Line Basics

Before we can setup a honeypot, we will learn some basics of the Linux command line. This will give us the knowledge necessary to install and setup our honeypot, and also give us insight into what attackers are doing once we start to analyze honeypot data.

4.1 Command Line Anatomy

The most basic access that we can have to a computer is through the command line, which is sometimes also called a terminal or a shell. Users are created on machines with certain permissions, and these users can login to the machine via a terminal, and enter commands for the machine one line at a time. Different terminals may present minute differences in the appearance of a command line, but they all share the following "command prompt":

```
<username>@<machine_name>:<working_directory>$
```

This "command prompt" comprises:

- the name of the logged-in user,
- the name of the machine,
- the "working directory", and finally
- the $ symbol which separates the prompt from the command that you enter.

4.2 The Working Directory

When using any command line environment, it is up to the user to keep a mental map of where in the file system they are currently located (the "working directory"). Luckily, there are two easy commands to help us figure out where we are and what files and sub directories are currently around us.

- **pwd** - Lists the absolute path of the current directory

```
user@wse380:~$ pwd
/home/user/
```

- **ls** - Lists the contents of a directory. Specifying no path will display the contents of the current directory.
  - `ls -l`: Displays additional information about each file such as permissions and file size
  - `ls -a`: Displays all files in a directory, including hidden ones

```
user@wse380:~$ ls -l -a
total 24
drwxr-xr-x 2 wse380 wse380 4096 Mar 11 13:00 .
drwxr-xr-x 9 root root   4096 Mar 11 13:00 ..
-rw------- 1 wse380 wse380 5 Mar 11 13:00 .bash_history
-rw-r--r-- 1 wse380 wse380 220 Mar 11 13:00 .bash_logout
-rw-r--r-- 1 wse380 wse380 3771 Mar 11 13:00 .bashrc
-rw-r--r-- 1 wse380 wse380 807 Mar 11 13:00 .profile
```

Above, we used `ls` to list the contents of the current directory while also specifying the `-l` and `-a` flags to show us all files as well as details on each file. Here, from left to right on each row, we can see the permissions on each file and directory, the user that owns it, the user group that has access to it, its size in bytes, the date and time it was last modified, and its name.

4.3 Moving Around the File System

One of the most important symbols in the Linux command line is the ".", or period symbol. In the context of the file system, "./" represents the directory the user is currently in, and "../" represents the parent directory of the current directory. For the most part, when attempting to navigate the file system or interact with files, you will use file paths relative to your current directory. These are called *relative paths*. For example, if a user was trying to list the contents of the current directory's parent, they would use the following command:

```
user@wse380:~$ ls ../
```

The notion of relative file paths becomes vital when attempting to navigate the file system. To accomplish this, we make use of the "change directory" or `cd` command.

- **cd** - Change the current directory to the one specified. Supplying no arguments navigates to the user's home directory (abbreviated as ∼).

```
user@wse380:~$ ls
wse380-webapp/
user@wse380:~$ cd wse380-webapp/
user@wse380:~/wse380-webapp$ pwd
/home/user/wse380-webapp
user@wse380:~/wse380-webapp$ cd ../
```

Above, we issued a series of commands to help visualize our movement around the file system. We first used ls to view the files and sub directories we had available to us from our current directory. We then used the `cd` command to move to the "wse380-webapp" sub directory. A call to the `pwd` command shows that our new current directory was now the directory located at "/home/user/wse380-webapp". Finally, we used the `cd` command with the relative path "../" to move back to the parent directory, the original directory we started at. An example of an *absolute path* would be the output of the earlier `pwd` command: "/home/user/wse380-webapp". We could have, for example, executed `cd /home/user/wse380-webapp` from anywhere in the system instead of needing to execute `cd wse380-webapp` from our home directory. This is powerful because absolute paths enable us to *refer to any part of the file system, regardless of our current working directory*.

4.4 Creating, Reading, Moving, Copying, and Deleting Files and Directories

- **touch** - Create a file with the specified name.

```
user@wse380:~$ touch example.txt
```

- **cat** - Print a file's contents to the terminal window.

```
user@wse380:~$ cat example.txt
```

- **mkdir** - Create a new directory with the specified name.

```
user@wse38:~$ mkdir exampleFiles
```

- **mv** - Move a file from one location in the file system to another. Also the method used to rename files from the command line.

```
user@wse380:~$ mv example.txt exampleFiles/example.txt
```

- **cp** - Copy a file from one location in the file system to another while keeping the original file untouched. When copying a directory, the user must supply the "-R" argument.

```
user@wse380:~$ cp example.txt exampleFiles/example.txt
```

- **rm** - Remove a file from the file system. To remove a directory, the user must include the "-r" argument to *recursively* remove all sub directories and files. **IMPORTANT: Removing files with this command is permanent! This command is the equivalent of using your graphical interface to move a file to the trash and emptying it. Only remove a file or directory you are sure you want permanently deleted!**

```
user@wse380:~$ rm -r exampleFiles/
```

4.5 Editing Files

You may be used to writing code and editing configuration files using an IDE. While many of the most popular IDEs are available on Linux, it is important to know how to edit files from the Linux command line since you will not be able to use a graphical text editor or IDE to edit files on remote servers.

The most popular command line text editors on Linux include: Vim, Nano, and Emacs. We will be using Nano in this course as it has the easiest learning curve. To get started, run the following command to open a file with Nano:

```
user@wse380:∼$ nano example.txt
```

Here, *example.txt* can either be a new file or an existing file. You will then be able to enter any new text or edit any existing text in the file. Once you are ready to save your work, press *Ctrl+x* to exit Nano. You will be asked if you'd like to save the changes that you made. If so, press *Y*, otherwise, press *N*. Lastly, you will be asked to provide the name of the file you'd like to write your changes to. By default, this will be the name of the file you entered when opening Nano. Most of the time we'd just press enter here. However, if you'd like to keep the original file the same as when you opened it and write your changes to a new file, enter the name of the new file and press enter.

4.6 Downloading Programs

When you downloaded and installed applications on your computer in the past, you most likely did so either through an application marketplace or by downloading them from a vendor's website and running an installer program. On Linux, you can download and install programs using the command line program *apt* or "Advanced Package Tool". In many ways, *apt* is easier to use than the process you are familiar with. The only command we have to learn is:

```
user@wse380:∼$ sudo apt install program_name
```

Here, "program_name" is the name of the program you would like to install. In order to install applications using *apt*, we must run it with administrator privileges. We will discuss this in Section 4.9.

4.7 Downloading Code Using Git

As you may be familiar with from your own programming assignments, `git` allows us to upload our code to the cloud for sharing and collaboration. For this course, we are only interested in how to download code from a remote git repository to our machine. To do this we enter the following command:

```
user@wse380:∼$ git clone https://github.com/...
```

Let us clone the repository with the material for this semester by running:

```
user@wse380:∼$ git clone https://github.com/jso123450/wse380-webapp.git
```

This repository will contain most, if not all, of the files and lesson plans for this course.

4.8 Running Programs

After we download a program or write our own, we want to actually run them. Doing so is easy, although the syntax of the command differs depending on what kind of program it is. If a program was downloaded using *apt*, it can be executed simply by typing the name of the program into the terminal, because the program is downloaded to a directory that our command line will check by default. For example, if we downloaded the Internet browser Firefox using *apt* we could run it like so:

```
user@wse380:∼$ firefox
```

To see where such programs are downloaded, we can use `which`:

```
user@wse380:∼$ which firefox
/usr/bin/firefox
user@wse380:∼$ which abc
abc not found
```

If we write our own program or download a program from another source like GitHub, we need to address the executable file directly in the terminal using a relative or absolute path, as we talked about earlier. For example, if we downloaded a GitHub repository containing an executable file named *foo*, we could run it by first navigating into the directory containing *foo*, and executing the command:

```
user@wse380:~$ cd some-random-folder
user@wse380:~/some-random-folder$ ./foo
```

Alternatively, we could have used a different relative path, or an absolute path:

```
user@wse380:~$ ./some-random-folder/foo
user@wse380:~$ /home/user/wse380/some-random-folder/foo
```

When executing any kind of program, there will most likely be data outputted to the terminal window. There are two standard places where a program writes its output: standard out (`stdout`) for regular program output and standard error (`stderr`) for error and log messages. Sometimes we are interested in saving this output to a file for later processing, especially for long-running programs. To do so, we will use the ">" symbol to tell Linux to redirect the `stdout` of the command to a file of our choosing. Take for instance the following command:

```
user@wse380:~$ ./foo > foo_output.log
```

Here, instead of `stdout` of `foo` going to the terminal window, it will now appear in a new file called `foo_output.log` in the current working directory. If we are also interested in the `stderr` of the program, we can run it as follows:

```
user@wse380:~$ ./foo > foo_output.log 2>&1
```

Here, we redirect both `stdout` and `stderr` of `foo` to the file `foo_output.log` in the current working directory.

4.9 Sudo

So far, we learned basic Linux commands we need to setup and manage our servers. However, some commands require administrator, or *root*, privileges. On Linux machines, the root user has permission to do just about anything it wants, even modify or delete critical operating system files. Due to this immense power, we typically don't directly sign in with the root user account, but rather permit regular user to execute commands as root as they see fit by explicitly stating their intent to do so. This prevents users from accidentally performing dangerous commands without thinking.

To state our intent to run a command as root, we prepend the keyword *sudo* ("super user do") before the command we would like to run. For example, if we would like to open a file in Nano with root privileges, we would enter the following command:

```
user@wse380:~$ sudo nano example.txt
```

You will sometimes be asked to enter your password when you attempt to run a sudo command. Just enter the same password you used to log into your account, and you should be good to go, assuming your user account is permitted to execute sudo commands.

4.10 Tab Completion

Most shells offer *tab completion* — if you begin to type part of a file and press the tab key, the shell should autocomplete the name for you (assuming there is only one file that matches the prefix you have typed). If the prefix matches multiple files, you can press tab multiple times, and your shell should attempt to list all files that match the prefix and cycle through them. Tab completion will only work for files that are contained in directories that the shell is configured to check, plus the current working directory.

5 Setup and Administration of a Server

Now that we have learned the basics of the Linux command line, we are going to put everything together to setup our web app server. As previously discussed, setting up a web application in a secure manner requires several different pieces of software. We will start by configuring access control parameters to the server.

5.1 SSH

The Secure Shell Protocol or *SSH* is a networking protocol that allows users to connect to a remote computer over a secure channel. When you connect to a SSH server, you will be presented with a command line to execute the same commands we discussed in part 1. Since compromising a host over SSH allows for access to all programs and files of a particular user, attacks against this protocol are very common. By default, SSH servers listen on port 22, and this is the typical port that attackers choose to probe for vulnerable SSH servers. There are a number of other default ports that attackers commonly scan, and potentially vulnerable services on different ports. We will be configuring a firewall to limit which ports are accessible, and an intrusion prevention software to monitor the SSH port 22, and others, for connection requests and the appropriate enforcement actions. (Another technique to bypass

5.2 UFW (Uncomplicated Firewall) [4]

Firewalls are network security systems that monitor and control incoming and outgoing traffic according to a set of specified rules. They can come in the form of software, or a dedicated physical machine. In this course, we will be using a ubiquitous Linux software firewall called `ufw` (uncomplicated firewall) to setup rules to restrict access to our VMs. In particular, we want to make sure that we are only exposing access to the necessary ports.

Let's install `ufw` with the following:

```
sudo apt update
sudo apt upgrade
sudo apt install ufw
```

Now, let us add some basic firewall rules to our VM. The course repository has a set of predefined basic firewall rules that we can use. Let's see what they are:

```
ubuntu@wse380:~$ cd wse380-webapp
ubuntu@wse380:~/wse380-webapp$ cat ufw/install-rules-basic.sh
#!/bin/bash

# default rules
sudo ufw default deny incoming
sudo ufw default allow outgoing

# http rules
# sudo ufw allow in 80/tcp
# sudo ufw allow in 443/tcp

sudo ufw allow in 22/tcp

sudo ufw enable
sudo ufw status verbose
ubuntu@wse380:~/wse380-webapp$
```

As you may have noticed, this file is a `.sh` file, which indicates that it is a *shell script*. These shell scripts contain commands that we can execute in the command line. In particular, this script is a *bash script* (`bash` is a specific shell), which we can tell from the *shebang* in the first line (`!/bin/bash`). We can execute this shell script (which executes all the commands sequentially) with the following:

```
ubuntu@wse380:~/wse380-webapp$ ./ufw/install-rules-basic.sh
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)
Default outgoing policy changed to 'allow'
(be sure to update your rules accordingly)
Rules updated
Rules updated (v6)
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere
22/tcp (v6)                ALLOW IN    Anywhere (v6)
```

5.3 Fail2Ban [4]

Fail2Ban is a traditional intrusion-prevention software that can run on Linux systems. In this course, we will be using fail2ban to add a layer of protection against attackers who may try to compromise our server. Let's install it by running:

```
sudo apt update
sudo apt upgrade
sudo apt install fail2ban
```

Similarly to `ufw`, let us add some basic configuration rules. The course repository has a set of predefined basic rules that we can use:

```
ubuntu@wse380:~/wse380-webapp$ cat fail2ban/fail2ban.conf
[DEFAULT]
bantime = 1d
findtime = 1d
ignoreip = 127.0.0.1/8 192.168.0.0/16 130.245.0.0/17
maxretry = 3

banaction = ufw
banaction_allports = ufw

[sshd]
enabled = true

[ufw]
enabled = true
filter = ufw
logpath = /var/log/ufw.log
ubuntu@wse380:~/wse380-webapp$
```

We can see that there is a default policy that bans IP addresses for one day if they trigger 3 failed attempts in one day, excluding local IP addresses (`127.0.0.1/8` and `192.168.0.0./16`) and Stony Brook IP addresses (`130.245.0.0./17`). Afterwards, the configuration tells `fail2ban` to check for SSH and UFW logs, among things.

We can install these rules by running the following script:

```
ubuntu@wse380:~/wse380-webapp$ ./fail2ban/copy-config.sh
Waiting 5s for fail2ban to restart...
Status
|- Number of jail:    1
`- Jail list:  sshd
ubuntu@wse380:~/wse380-webapp$
```

Now we have finished the basic access control configuration for our servers, and can begin installing the main prerequisite we will need for the course: Docker.

6 Installing Docker and Docker Compose

Docker is a tool that allows developers to package their programs into portable containers. For the sake of this course, all you need to know is that Docker removes all of the complexity in setting up an application. We will not go into any more detail on Docker (aside from a few basic commands), although I encourage you to continue reading up on Docker as it is an industry standard in the deployment of web applications.

To install Docker, we are going to use the apt tool we discussed earlier. We first want to update apt to make sure it downloads the latest version of each application we request. We do that using the following commands in `docker/install_docker.sh` [2]:

```
ubuntu@wse380:~/wse380-webapp:$ cat docker/install_docker.sh
#!/bin/bash

# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
    https://download.docker.com/linux/ubuntu \
    $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Install
sudo apt-get install docker-ce docker-ce-cli containerd.io \
    docker-buildx-plugin docker-compose-plugin
```

We have now successfully installed Docker! One more thing — we will also let our user account to use Docker without root privileges by running the following:

```
# Use docker without sudo
sudo usermod -aG docker ${USER}
su - ${USER}
```

This will add the user account into the ‘docker‘ user group for the correct permissions, and relogin to refresh the groups. Alternatively, we can also close the current SSH connection and reconnect for the groups to take effect.

Verify your installation by running:

```
docker run hello-world
```

Now let us install Docker Compose to help with running multiple Docker containers [1].

```
sudo apt-get update
sudo apt-get install docker-compose-plugin
```

Verify the Docker Compose installlation by running:

```
ubuntu@wse380:~/wse380-webapp$ docker compose version
Docker Compose version v2.24.6
```

Next Steps

Today, we are going to walk through a hands-on guide for installing the software we need for our servers.

7 Web Application Infrastructure

Servers are machines that are designed to be running software 24/7. However, there are different ways to deploy software on machines: directly on the *bare metal*, or alternatively with *virtualization* (i.e., by using a *virtual machine* (VM) or a *container*). There are differences in these methods that bring certain (dis)advantages, and one method may be better than the others depending on the scenario (usually for performance reasons). In general, virtualization techniques are supposed to be able to provide a common ground by defining standard environments for applications.

In this class, we will not elaborate on these differences; at this stage, it is more important to recognize that there are different methods. You will be given access to a *virtual machine*, and we will be running software directly in the operating system of the VM, and inside Docker containers in the VM.
TODO infrastructure diagram

8 Setup: Docker Compose Services

Let's begin configuring the web application software by using Docker Compose. Compose is a plugin that makes it easy to manage multiple Docker containers in one context. Typically, one Compose project is created for one application, which may comprise multiple smaller pieces of software. In this class, we will be using a provided Compose file to begin configuring our individual projects.

8.1 Environment Variables

Environment variables, as their name suggests, are variables that exist in the environment of an application, and may not actually exist in the application configuration itself. For example, something analogous to an environment variable might be whether there is still sunlight outside. Depending on whether there is still light outside, you may opt to go to a restaurant, or go to a late-night drive-through for a fast food chain.

Applications can read values from their environment in a similar way. Let's change directory into our working directory named "~/wse380-webapp/wordpress", and begin to configure the environment variables used by our software [3].

```
cd ~/wse380-webapp/wordpress
nano .env
```

This should have opened the `.env` file in the `nano` text editor and we should see something like this:

```
MYSQL_DATABASE_NAME=wordpress
MYSQL_ROOT_PASSWORD=root_password
MYSQL_USER=wordpress_database_user
MYSQL_PASSWORD=wordpress_database_password

IP=yourip
DOMAIN=yourdomain
EMAIL=youremail
```

Modify the values for the following variables such that it remains in the format `VARIABLE=value`, with one per line (note that the values are case sensitive):

- `MYSQL_ROOT_PASSWORD` - the root password for your database; please use a secure password,
- `MYSQL_PASSWORD` - the password of the database account used by WordPress; please use a *different* secure password,
- `MYSQL_USER` - the username of the database account used by WordPress; please select any username,

- `IP` - the IP address of your VM; please input the IP address here,
- `DOMAIN` - the domain name you will register; please leave it be for now,
- `EMAIL` - the email you want to associate with TLS certificate expiry reminders; please leave it be for now.

Note that this file now contains *secrets for your application that should not be shared*. In deployment environments, it is important to make sure these files are not accidentally indexed or stored (e.g., by version control software such as git).

Next, please run the following to update the configuration files:

```
ubuntu@wse380:~/wse380-webapp$ ./components/scripts/replace-ip.sh
```

Now, let's start our applications!

```
ubuntu@wse380:~/wse380-webapp$ cd components
ubuntu@wse380:~/wse380-webapp/components$ docker compose up -d
...
[+] Running 3/3
 ✓ Container db        Running            0.0s
 ✓ Container wordpress Running            0.0s
 ✓ Container webserver Running            0.0s
```

Your VM will begin building the application containers and running them. After a short period of time, you should see similar text at the end, saying that all 3 containers `db`, `wordpress`, and `webserver` are running.

9 Setup: WordPress Installation via the Web Interface

One of the typical features of web apps (particularly for large CMS software) is the ability to install the application via the web. Let's see what it looks like: go to your browser and enter the URL `<your_ip>:12345`. You should now see the WordPress installation page!

Follow these steps to complete the installation:

1. Select a language (e.g., English)
2. Enter a site title (e.g., WSE380 - John's Blog)
3. Enter a username (e.g., admin-username)
4. Enter a secure password (or keep the default generated one) and store it in a password manager (e.g., your browser's)
5. Enter your email
6. Press the "Install WordPress" button!

The page should change after a short period of time, informing you of a successful installation and to log in with the username and password that was just provided. Please do so – we should land on the URL `http://<your_ip>/wp-admin/index.php` with a big "Welcome to WordPress!" banner.

10 Setup: DNS

So far we have accessed our web application directly via its IP address, but we want to be able to associate a memorable, human-readable name for our site! Follow along and let's proceed to register our own domain name!

10.1 Domain Registration

We register domain names through *domain registrars*, such as Dynadot, Namecheap, and GoDaddy. These registrars enable us to search for a domain name and tell us whether they are already registered, or available for purchase. Search for the domain you want to register!

10.2 DNS Name Servers

After we purchase (more accurately, lease!) a domain, we can configure its *authoritative name server*. This name server is a server that we identify by its IP address, and is responsible for translating the domain into an IP address. We typically configure the authoritative name server directly via the domain registrar from which we purchased the domain. Using the registrar, configure a DNS A record that points to your IP address.

10.3 Updating the Nginx/WordPress Configuration

Now we need to update our configurations to include the domain.

1. First, update our environment variable for the `DOMAIN` variable by running:

```
ubuntu@wse380:~/wse380-webapp$ nano components/.env
```

   and then modifying the domain value to your newly registered domain.

2. Run the script

```
ubuntu@wse380:~/wse380-webapp$ ./components/scripts/replace-domain.sh
```

3. Next, we need to tell WordPress that it should respond to requests for the domain that we just purchased! Head to `Settings` in the admin panel, which should open the `General Settings` page. Update the following:

   - WordPress Address (URL) to `http://<your_domain>`
   - Site Address (URL) to `http://<your_domain>`

4. Afterwards, scroll down and hit the `Save Changes` button.

5. Verify that you can reach your WordPress site by navigating to `http://<your_domain` in your browser! You will need to re-authenticate to the admin panel, since the origin is now different.

11 Setup: TLS Certificates

Now, we have established a site from which we can access via a domain name! Our next step is to obtain a TLS certificate — the missing piece that will enable our browsers to make secure connections via HTTPS! The entire ecosystem is complicated, but the key pieces are:

- we will request a certificate from a globally trusted Certificate Authority (CA),
- the CA will verify that we own the domain, and
- the CA will issue a certificate that says we own the specified domain.

After we obtain the certificate, we can update our configuration so that our Nginx web server knows to present the certificate for browsers that request HTTPS connections to our website. The browser will see that our valid certificate is from a globally trusted CA, and thus trust that we are the actual owner of the domain.

11.1 Certificate Request

Let's begin the process of obtaining the certificate.

1. Edit the `components/docker-compose.yml` file with `nano`, and remove the leading # symbols in front of the `certbot` service to enable it.

2. Restart the containers by running

```
ubuntu@wse380:~/wse380-webapp$ docker compose up -d
```

3. Monitor the certificate process by running:

```
ubuntu@wse380:~/wse380-webapp$ docker compose logs -f certbot
...
certbot | Requesting a certificate for <your_domain> and www.<your_domain>
certbot |
certbot | Successfully received certificate.
certbot | Certificate is saved at: /etc/letsencrypt/live/<your_domain>/fullchain.pem
certbot | Key is saved at:       /etc/letsencrypt/live/<your_domain>/privkey.pem
certbot | This certificate expires on 2024-06-01.
certbot | These files will be updated when the certificate renews.
certbot | NEXT STEPS:
certbot | - The certificate will need to be renewed before it expires. Certbot can
    automatically renew the certificate in the background, but you may need to take steps to
    enable that functionality. See https://certbot.org/renewal-setup for instructions.
```

4. Verify that you can see the certificates:

```
ubuntu@wse380:~/wse380-webapp$ docker compose exec webserver ls -la /etc/letsencrypt/live
total 16
drwx------   3 root    root         4096 Mar 3 21:59 .
drwxr-xr-x   7 root    root         4096 Mar 3 22:18 ..
-rw-r--r--   1 root    root          740 Mar 3 21:59 README
drwxr-xr-x   2 root    root         4096 Mar 3 22:18 <your_domain>
```

5. Now, re-comment the lines in `components/docker-compose.yml` for the `certbot` service (by adding the leading # symbols) so we do not send unnecessary certificate requests.

11.2 Updating Docker Compose

Now, the final step is to tell Docker Compose to use this new certificate. Let us run the following:

```
ubuntu@wse380:~/wse380-webapp$ ./components/scripts/change-to-https.sh
ubuntu@wse380:~/wse380-webapp$ docker compose up -d
```

Verify that you are now able to use a secure connection to your website by specifying HTTPS in your browser!

Next Steps

12 Customization: Content

TODO

13 Customization: Styling and Themes

TODO

14 Customization: Plugins

TODO

14.1 Security - Wordfence

TODO

14.2 Performance - LiteSpeed Cache

TODO

14.3 Analytics - Google Site Kit

TODO

14.4 Forms - WPForms

TODO

15 Monitoring: Resource Usage

TODO

16 Monitoring: Uptime

TODO

17 Cloudflare

TODO

18 Analysis of Server Logs

TODO

19 Conceptual Review

TODO

References

[1] Install the compose plugin. `https://docs.docker.com/compose/install/linux/`.

[2] Install docker engine on ubuntu. `https://docs.docker.com/engine/install/ubuntu/`.

[3] Kathleen Juell. How to install wordpress with docker compose. `https://www.digitalocean.com/community/tutorials/how-to-install-wordpress-with-docker-compose`, 2024.

[4] Jean-Jerome Levy. Turn your nginx server into a fortress with fail2ban and ufw. `https://scalastic.io/en/ufw-fail2ban-nginx/`, 2023.