Ubuntu VM Web Server

August 22, 2023

---

i want to setup web server on ubuntu on virtualbox

Certainly! Setting up a web server on an Ubuntu virtual machine (VM) running in VirtualBox can be a straightforward process. Here's a step-by-step guide to help you do this:

1. **Setting up VirtualBox:**
   1.1. Download and install VirtualBox from here.
   1.2. Download an Ubuntu ISO image from the official website.
   1.3. Create a new virtual machine in VirtualBox. When asked for the type and version, select "Linux" and "Ubuntu".
   1.4. Assign memory, create a virtual hard drive, and configure any other settings you want.
   1.5. Before starting the VM, go to its settings -> Storage -> Add Optical Drive, then choose the Ubuntu ISO you downloaded.
   1.6. Start the VM and follow the on-screen instructions to install Ubuntu.
2. **Install Updates:**
   Once Ubuntu is up and running, open a terminal and enter the following commands to make sure your system is up-to-date:

```bash
sudo apt update
sudo apt upgrade
```

3. **Install the Web Server (Apache):**
   To install the Apache web server, run:

```bash
sudo apt install apache2
```

4. **Test the Web Server:**
   Once installed, Apache should start automatically. To verify this, open your web browser in Ubuntu VM and navigate to `http://localhost`. You should see the Apache default welcome page.
5. **Access Web Server from Host Machine:**
   If you wish to access the web server from your host machine (the actual machine running VirtualBox):
   5.1. In VirtualBox, select the Ubuntu VM.
   5.2. Go to Settings -> Network.
   5.3. Change the "Attached to" setting to "Bridged Adapter". This will give your VM an IP address on your local network.
   5.4. Save the settings and start the VM.
   5.5. In the Ubuntu terminal, run `ip a` or `ifconfig` to get the IP address assigned to the VM.
   5.6. On your host machine, open a browser and navigate to the IP address you found in the previous step (e.g., `http://192.168.1.100`).
6. **Additional Configuration (Optional):**
   - **Install PHP & MySQL:** If you're planning to run PHP scripts or want a MySQL database, run:

     bash Copy code

     ```
     sudo apt install php libapache2-mod-php php-mysql mysql-server
     ```

   - **Manage Apache:** You can start, stop, or restart the Apache server using:

     bash Copy code

     ```
     sudo systemctl start apache2
     sudo systemctl stop apache2
     sudo systemctl restart apache2
     ```

   - **Apache Config Files:** Main configuration files for Apache are located in `/etc/apache2/`. The main config file is `apache2.conf`.
7. **Secure Your Web Server:**
   Remember to always secure your web server, especially if it's accessible from the internet. This involves steps like:
   - Using strong, unique passwords.
   - Regularly updating software.
   - Configuring a firewall.

- Limiting access to sensitive areas.

8. **VirtualBox Guest Additions (Optional but recommended):**
   Installing the Guest Additions can improve the integration and performance of the VM. To install, from the VirtualBox VM window menu, choose `Devices > Insert Guest Additions CD image`, then run the installer from the virtual CD-ROM that appears in the VM.
   That's it! You now have an Ubuntu VM running in VirtualBox with an Apache web server installed.

how to install phpmyadmin

Certainly, phpMyAdmin is a popular web-based tool for managing MySQL/MariaDB databases. Here's a guide to install phpMyAdmin on your Ubuntu server:

1. **Update the Package Repository:**

bash Copy code

```
sudo apt update
sudo apt upgrade
```

2. **Install phpMyAdmin:**

bash Copy code

```
sudo apt install phpmyadmin
```

During the installation:

- When prompted to choose the web server that should be automatically configured to run phpMyAdmin, select `apache2` by pressing the space bar, then tab to the OK button and press Enter.
- You'll then be asked if you want to use `dbconfig-common` to set up the database. Select "Yes".
- You will be prompted to set a password for the `phpmyadmin` user. Enter your desired password and remember it or keep it secure.

3. **Enable Necessary PHP Extensions:**
   You may need to enable the `mbstring` extension. Do so using the following commands:

bash Copy code

```
sudo phpenmod mbstring
```

4. **Restart Apache:**
   To make all the changes take effect, restart the Apache web server:

   bash Copy code

   ```
   sudo systemctl restart apache2
   ```

5. **Access phpMyAdmin:**
   Open a web browser and navigate to `http://your_server_ip/phpmyadmin` or `http://localhost/phpmyadmin` if you're on the VM. You should see the phpMyAdmin login page.
   Login using:
   - Username: `root` (or another MySQL user you have set up).
   - Password: Your MySQL root password (or the password for the MySQL user you're logging in with).
6. **Secure phpMyAdmin (important):**
   Exposing phpMyAdmin to the public internet can be risky. It's recommended to either:
   - Keep it local and only access it from within your VM or local network.
   - Or, if you need to access it remotely, use security precautions like setting up a VPN, or restricting access via `.htaccess`, or only allowing certain IP addresses.

   Remember, always backup your configurations and databases before making significant changes. This way, you can recover your system if something goes wrong.

mysqli::real_connect(): (HY000/1698): Access denied for user 'root'@'localhost'

The error message `mysqli::real_connect(): (HY000/1698): Access denied for user 'root'@'localhost'` suggests that there's a privilege problem with the `root` user in your MySQL or MariaDB installation on Ubuntu.
By default, the root user for MariaDB and some versions of MySQL on Ubuntu is set to authenticate using the `auth_socket` plugin rather than with a password. This means that only the system root user can log in as the MySQL root user without a password.
Here are steps to address this:

1. **Log in to MySQL/MariaDB with Super User Privileges**:
   If you are on the system where the database is running and you have superuser privileges, you can bypass the password system:

```bash
sudo mysql -u root
```

This will get you into the MySQL/MariaDB shell.

2. **Change the Authentication Method for the Root User**:
   Once logged in, you can switch the authentication method from `auth_socket` to `mysql_native_password` (or the equivalent for your DB version):
   For MySQL:

```sql
ALTER USER 'root'@'localhost' IDENTIFIED WITH 'mysql_native_passwo
```

   For MariaDB:

```sql
ALTER USER 'root'@'localhost' IDENTIFIED BY 'your_password_here';
```

   Replace `'your_password_here'` with a strong password of your choosing.

3. **Flush Privileges**:
   Let the server know to reload the grant tables and apply the changes:

```sql
FLUSH PRIVILEGES;
```

4. **Exit MySQL/MariaDB**:

```sql
exit;
```

5. **Restart the MySQL/MariaDB Service**:

```bash
sudo systemctl restart mysql
```

   Or, if you are using MariaDB:

bash Copy code

```
sudo systemctl restart mariadb
```

6. **Try to Connect Using the Root User and Password**:

bash Copy code

```
mysql -u root -p
```

You should now be prompted for the password you set in step 2.
Remember to use strong passwords and never expose the root user to the public internet. If you're developing applications, it's also a good practice to use dedicated database users with limited privileges instead of using the root user.

i want ftp as well

Sure, if you want to set up an FTP server on your Ubuntu system, one of the most commonly used FTP servers is `vsftpd` (Very Secure FTP Daemon). Here's how you can set it up:

1. **Install vsftpd**:

bash Copy code

```
sudo apt update
sudo apt install vsftpd
```

2. **Backup Original Configuration** (Always a good practice):

bash Copy code

```
sudo cp /etc/vsftpd.conf /etc/vsftpd.conf.orig
```

3. **Edit Configuration**:
Open the vsftpd configuration file in a text editor (like nano):

bash Copy code

```
sudo nano /etc/vsftpd.conf
```

Here are some basic configurations you might consider:

- To allow users to upload files:

makefile Copy code

```
write_enable=YES
```

- To prevent anonymous logins:

makefile Copy code

```
anonymous_enable=NO
```

- To allow local users to log in:

makefile Copy code

```
local_enable=YES
```

- To confine users to their home directories:

makefile Copy code

```
chroot_local_user=YES
```

After making your desired changes, save and close the file (in nano, it's `Ctrl` + X to close, Y to confirm changes, and `Enter` to save).

4. **Restart vsftpd**:

bash Copy code

```
sudo systemctl restart vsftpd
```

5. **Allow FTP Through UFW (If you're using UFW)**:
   If you have UFW (Uncomplicated Firewall) installed and enabled, you'll want to allow FTP traffic:

bash Copy code

```
sudo ufw allow 20/tcp
sudo ufw allow 21/tcp
```

6. **Create an FTP User** (Optional):
   If you want to create a dedicated FTP user:

bash Copy code

```
sudo adduser ftpuser
```

Follow the prompts to set a password and fill in any other info. If you set `chroot_local_user=YES` in the configuration, this user will be confined to their home directory.

7. **Testing**:
   You can test the FTP server locally using:

   bash Copy code

   ```
   ftp localhost
   ```

   Or from another system, replace `localhost` with your server's IP address.

8. **Security Note**:
   FTP by its nature transmits data, including passwords, in plain text. If you're planning to use FTP outside of a local network, consider setting up SFTP (FTP over SSH) or configuring FTPS (FTP with SSL/TLS) to secure data transmission. Both methods encrypt the data transfer, adding a layer of security.

how to give user as root

Giving a user root-like privileges or making a user act like the root user is typically done using the `sudo` command in Ubuntu and other Linux systems. It allows a permitted user to execute a command as the superuser or another user, as specified by the `sudoers` file.

Warning: Giving a user root privileges can be risky. This user will have the power to make system-wide changes, potentially harming the system or compromising security. Ensure you trust the user and understand the implications.

To grant a user sudo (root) privileges, follow these steps:

1. **Login as a Superuser**:
   You should be logged in as a user with sudo privileges to make this change.

2. **Edit the sudoers File**:
   The `sudoers` file controls who can run what commands as which users on what machines and can also control special things such as whether you need a password for particular commands.
   To edit the `sudoers` file safely, use the `visudo` command:

   bash Copy code

```
sudo visudo
```

By default, this will open the `sudoers` file in the `nano` editor on most systems. If it opens in `vi`, and you're unfamiliar with `vi`, you might want to switch to `nano` by running `sudo EDITOR=nano visudo`.

3. **Add User to sudoers File**:
   In the file, you'll find a line that looks something like this:

```sql
root    ALL=(ALL:ALL) ALL
```

   Below this line, add a line for your user, replacing `username` with the actual username:

```sql
username    ALL=(ALL:ALL) ALL
```

   This line means that the user `username` can run any command as any user from any host.
4. **Save and Exit**:
   If you're in `nano`, press `Ctrl` + X to close, then Y to confirm changes, and finally press `Enter` to save. If you're in `vi`, press `Esc`, then type `:wq` and press `Enter`.
5. **Test the Configuration**:
   To test if your user can now use sudo, log in as that user and try running a command with `sudo`. For instance:

```bash
sudo ls /root
```

   It should prompt for the user's password and then display the contents of the `/root` directory.
   Remember, with great power comes great responsibility. It's a good idea to use the `sudo` command judiciously and only when necessary.

how to restart apache and php

To restart Apache and PHP, you generally only need to restart the Apache web server

because PHP runs as a module or as a process managed by the web server in most setups. Here's how you can restart Apache on a system using `systemd` (like Ubuntu):

1. **Restart Apache**:

```bash
sudo systemctl restart apache2
```

If you're using a different system init tool, like `init.d`, you'd use:

```bash
sudo /etc/init.d/apache2 restart
```

2. **Check Apache's Status**:

If you want to verify that Apache restarted successfully:

```bash
sudo systemctl status apache2
```

For PHP:

- If you're running PHP as an Apache module (mod_php), then restarting Apache (as shown above) also restarts PHP.
- If you're using PHP-FPM (FastCGI Process Manager) along with Apache or Nginx, you need to restart the PHP-FPM service separately:

```bash
sudo systemctl restart php7.4-fpm
```

Replace `7.4` with your PHP version if it's different. If you're unsure about your PHP version, you can check it with:

```bash
php -v
```

1. **Check PHP-FPM's Status**:

If you want to verify that PHP-FPM restarted successfully:

```bash
```

```
sudo systemctl status php7.4-fpm
```

Again, the exact commands might differ slightly based on your Linux distribution and the version of PHP you're using, but the provided commands should work for modern versions of Ubuntu.

how to set up ssh

Setting up SSH (Secure Shell) on Ubuntu or any other Linux distribution allows for secure remote access to your server. Here's a step-by-step guide to set up and configure SSH on your Ubuntu system:

1. **Install OpenSSH Server**:
   By default, the SSH client is usually installed on Ubuntu systems. However, to accept SSH connections, you need to install the SSH server. You can do this with:

   bash Copy code

   ```
   sudo apt update
   sudo apt install openssh-server
   ```

2. **Verify SSH is Running**:
   Once installed, the SSH service should start automatically. You can check its status with:

   bash Copy code

   ```
   sudo systemctl status ssh
   ```

   If for some reason it's not running, you can start it with:

   bash Copy code

   ```
   sudo systemctl start ssh
   ```

3. **Configure SSH** (Optional, but recommended):
   The main configuration file for SSH is located at `/etc/ssh/sshd_config`. Before making any changes, it's a good practice to make a backup:

   bash Copy code

   ```
   sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.backup
   ```

Then, edit the configuration:

```bash
sudo nano /etc/ssh/sshd_config
```

Here are some basic configurations you might consider:

- **Change the SSH port**: By default, SSH listens on port 22. For added security, some people prefer to change this to a non-default port. To do this, find the line `#Port 22`, uncomment it (remove the #), and change the port number.
- **Disallow root login**: To prevent the root user from logging in via SSH (a security best practice), find the line `PermitRootLogin` and set it to `no`.
- **Allow specific users**: If you want to allow only certain users to SSH into your server, add or modify a line like this: `AllowUsers username1 username2`.

After making changes, save the file and close the editor. If you're using `nano`, it's `Ctrl` + X to close, Y to confirm changes, and `Enter` to save.

4. **Restart SSH to Apply Changes**:
   Whenever you make changes to the SSH configuration file, ensure you restart the SSH service:

```bash
sudo systemctl restart ssh
```

5. **Allow SSH Through UFW**:
   If you're using UFW (Uncomplicated Firewall) as your firewall, make sure you allow SSH traffic. If you've kept the default port (22):

```bash
sudo ufw allow ssh
```

   If you've changed the SSH port, for example to 2222:

```bash
sudo ufw allow 2222/tcp
```

6. **Connect to Your Server**:
   From another machine, you can now SSH into your server:

bash Copy code

```
ssh username@your_server_ip
```

Replace `username` with the user you want to log in as and `your_server_ip` with your server's IP address. If you changed the port, use the `-p` flag, like `ssh -p 2222 username@your_server_ip`.

Remember, while SSH is secure by default, it's always good to follow best practices like disabling root login and changing the default port. Also, consider setting up key-based authentication instead of (or in addition to) a password for an even more secure setup.