Introduction

What follows is a description of an “Internet of Things” inspired, do-it-yourself, background radiation monitor. The implementation below places a high priority on “open source” software and hardware. Avoiding proprietary, closed source software and cloud services allows for much greater flexibility. This flexibility facilitates ease of deployment and scale-out on a large variety of open source platforms, and a large variety of end user applications.

Prerequisites for building and deploying the DIY radiation monitor include: soldering, ability to follow electrical hookup guides, knowledge of Arduino technology, Linux system administration, and some knowledge of the C and Python computer languages. The open source software provided should be sufficient for most applications with only minor modifications. The server side software has been designed around the file structure and system configuration found in most common Linux distributions. In most cases, the majority of modification will be in the HTML, customizing the look and appearance of the web page presentation to the end user.

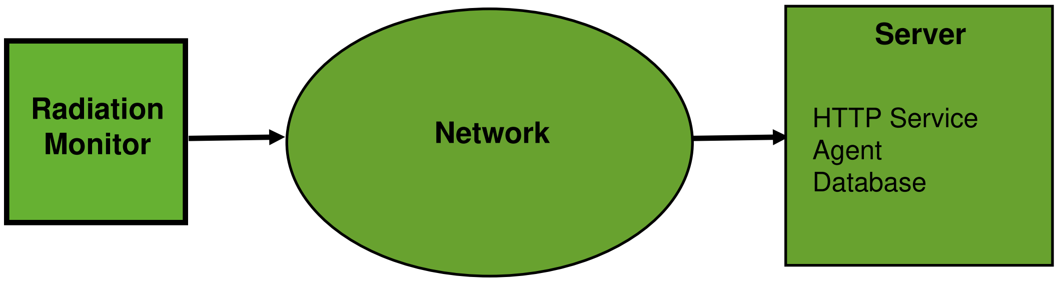

Figure 1. Overall conceptual view showing radiation monitor, network, and server components.

Radiation Monitor System Overview

Referring to figure 1, the DIY radiation monitor consists of three major components. They are the radiation monitor, the network connecting the radiation monitor to the server, and the server. The network connecting the radiation monitor to the server, including wireless access points, switches, routers, etc., will be considered background infrastructure beyond the scope of this project description.

The radiation monitor, itself, consists of several hardware components, as well as a software component running on the Arduino micro-controller. The server hardware can be anything from a full up Linux server to a Raspberry Pi. The server software consists of two components: an agent for processing data from the radiation monitor and HTML documents for displaying the data in a web page.

The server agent periodically sends an HTTP request to the radiation monitor. The monitor replies with a text sting containing the current radiation data. The agent parses this string, converts and reformats the data, updates a database, and generates graphical charts of the data. The agents sends the reformatted data to the HTTP server by writing to an output file, which Java script, embedded in the HTML documents, can read and display in web pages.

Component Descriptions

Radiation Monitor Hardware

Referring to figure 2, the radiation monitor consists essentially of only two components: an Arduino Uno with attached Ethernet shield and a Mighty Ohm Geiger counter. Both components are mounted in the same box. The project repository contains a photograph of the components mounted in a simple, plastic box. (See the file *RadiationMonitor.jpg.*)

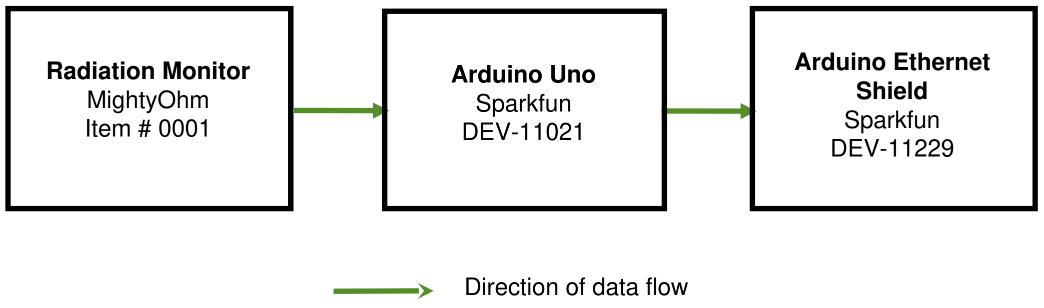

Figure 2. Block diagram of radiation monitor hardware showing individual electronic components.

Radiation Monitor Software

The software for the radiation monitor micro-controller consists of an Arduino “sketch”, along with an ancillary library that must be included at compile time. Referring to figure 3, the main module, or sketch, may be viewed as a “stack” comprising three layers. The top most layer includes the setup routine that runs once at boot up time, and the main loop that contains all the routines that run forever after initial setup. Initial setup runs routines that initialized the Ethernet interface and synchronize the system clock with a network time server.

The main loop runs the second layer routines, referred to in figure 3 as “Listen for Serial Port Command”, “Listen for Internet Client”, “Process Geiger Counter Data”, and “Synchronize System Clock”. In turn Listen for Serial Port Command calls “Process Serial Port Command”. Listen for Internet Client calls “Transmit HTML Document”, or “Transmit JSON String”. Synchronize System clock calls “Get Time from NTP Server”.

Listen for Serial Port Command intercepts keyboard characters received by way of the Uno's USB serial port. In a terminal session connected to the USB serial port, typing a "c" causes the radiation monitor to switch to command mode by calling *Process Serial Port Command*. In command mode, IP address mode, network time server IP address, and verbose mode can be changed by the command line.

Listen for Network Client checks for HTTP client requests and triggers the appropriate

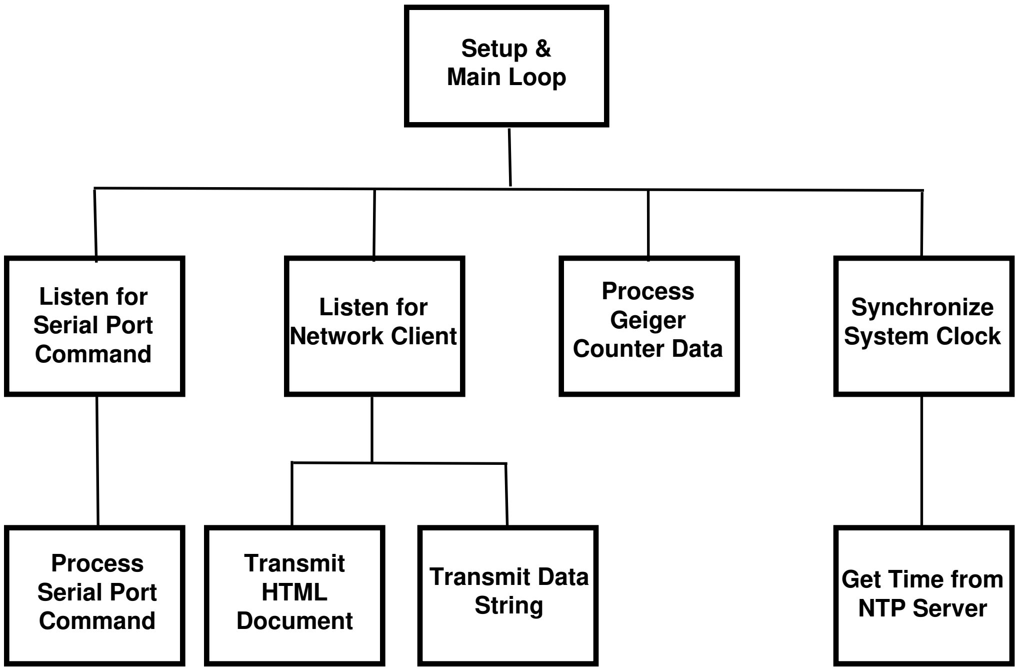
Setup & Main Loop
Listen for Serial Port Command
Listen for Network Client
Process Geiger Counter Data
Synchronize System Clock
Process Serial Port Command
Transmit HTML Document
Transmit Data String
Get Time from NTP Server

Figure 3. Micro-controller software block diagram showing high level components.

response depending on the URL sent by the client. The response may be either an HTML document or a JSON formatted text string.

Process Geiger Counter Data processes the serial data stream from the MightOhm Geiger counter. The MightyOhm sends out data in a CRLF delimited stream of ASCII characters, for example, a typical line has the form

```
CPS, 0, CPM, 22, uSv/hr, 0.12, SLOW
```

Process Geiger Counter Data collects this data, byte by byte, in a character array. When a LF character arrives the contents of the array gets copied to a temporary buffer for use by Listen for Internet Client when Internet clients request the radiation monitor data.

Synchronize System Clock gets the current universal coordinated time (UTC) from *Get Time from NTP Server* and sets the system clock to the UTC. When an Internet client requests data from the radiation monitor, the current UTC gets sent to the client along with the radiation data.

Server Software

Referring to figure 4, the server components consist essentially of two pieces: an HTTP server component and an agent component. The HTTP server component includes HTML documents, with embedded Java script, for displaying the radiation monitor data

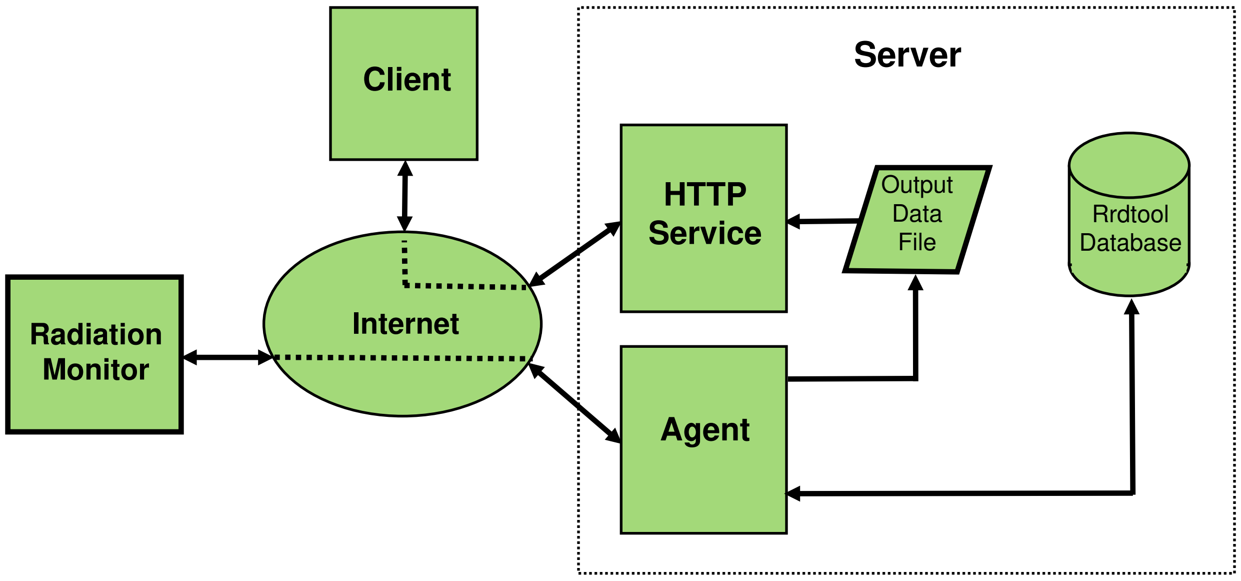

Figure 4. Block diagram of server software components showing data i/o, agent process, HTTP service, and database elements.

in a web browser. The agent, a Python script, manages data conversion and reformatting, updating a database, and generating graphical charts.
Events flow in the following manner. The agent periodically makes an HTTP request of the radiation monitor. The radiation monitor responds with a single string containing the data. Below is a typical example of a data string received from the radiation monitor.

```
$,UTC=0:14:39 9/26/2015,CPS=0,CPM=19,uSv/hr=0.10,Mode=SLOW,#
```

The agent parses this string, writes the data to a rrdtool database, and writes the data to the “Output Data File” for use by HTML documents. When a client requests a radiation monitor HTML document, Java script embedded in the document reads the output data file and displays this data in an HTML document. Although not shown in figure 4, the agent, as mentioned earlier, generates graphic charts for display in HTML documents.

The graphic charts are stored as image files which Java script in the HTML documents can display in the radiation monitor web page.

Building the Radiation Monitor

Parts List

Description	Quantity	Supplier	Part #
MightyOhm Geiger Counter	1	MightyOhm	0001
Arduino Uno micro-controller	1	Sparkfun	DEV-11021
Arduino Ethernet shield	1	Sparkfun	DEV-11229

Supplier websites
• MightyOhm: mightyohm.com/blog/products/geiger-counter/
• Sparkfun: www.sparkfun.com

Some additional hardware may be required, such as, a box to contain the Arduino parts and the MightyOhm Geiger counter. A small power supply (7 to 10 VDC) may be necessary if the Arduino's USB port will not be used for power.

Assembly Notes

1. Build the MightyOhm Geiger counter according to the manufacturer's instructions, substituting out the two resisters as directed in steps 2 and 3 below. The MightyOhm Geiger counter uses a 3 volt battery power supply and must be modified slightly to work with the Arduino Uno's 5 volt power. The following modifications will allow the Mighty Ohm Geiger counter board to operate using 5 volts supplied by the Uno instead of 3 volts supplied by battery. (See *MightyOhmSchematic.jpg* in the project docs folder.)
2. On the MightyOhm Geiger counter printed circuit board, change resistor R6 from a 330 Ohm to a 1K Ohm resistor.
3. On the MightOhm Geiger counter printed circuit board, change resistor R11 from a 100 Ohm to a 330 Ohm resistor.
4. Connect +5v from Arduino Ethernet shield to Geiger counter connector J6 pin 1.
5. Connect GND from Arduino Ethernet shield to Geiger counter connector J6 pin 3.
6. Connect Arduino pin D5 to Geiger counter connector J7 pin 5.
7. Mount the Arduino board and Geiger counter board in a suitable housing, such as a small plastic box. The housing should not be made of any material that shields radiation, such as lead.

This completes assembly of the radiation monitor.

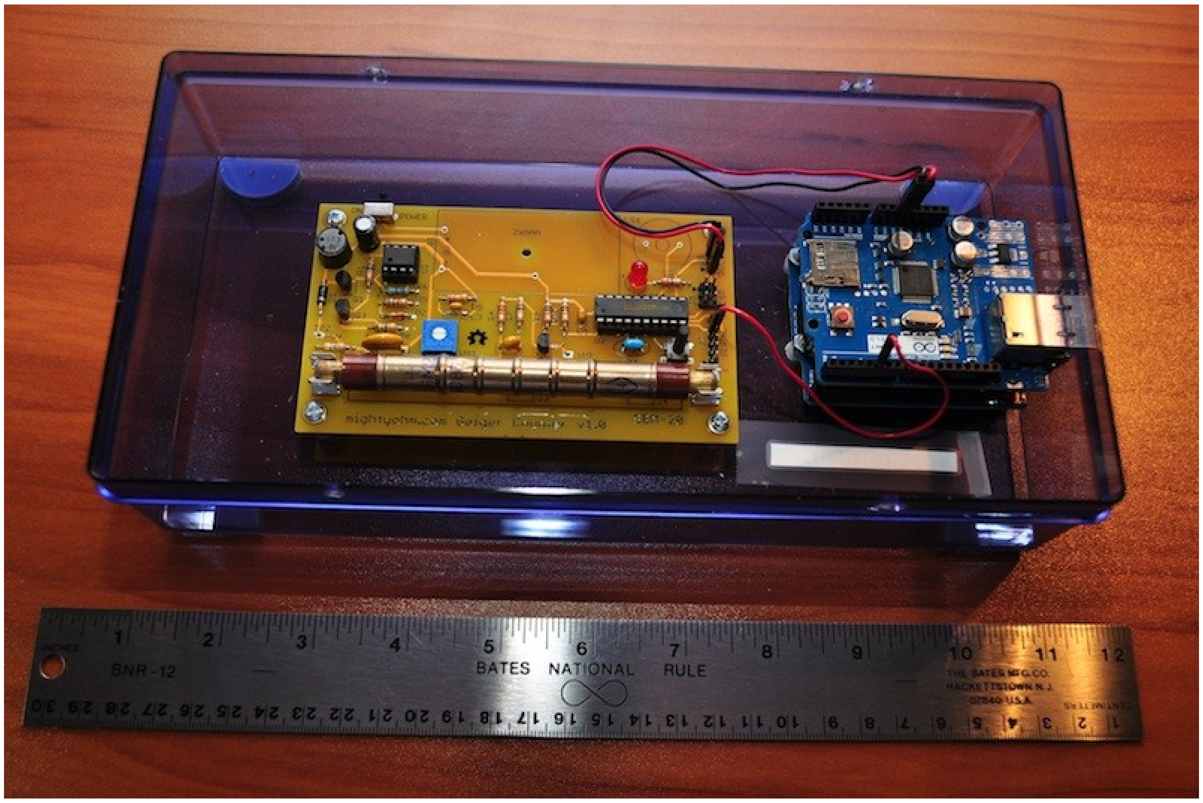

Photo showing assembly of MightyOhm Geiger counter and Arduino with Ethernet shield.

Software Installation

1. Get the latest version of the Arduino IDE from https://www.arduino.cc/en/Main/Software. The IDE is available for Mac, Windows, and Linux operation systems.
2. Before compiling the radiation monitor Arduino sketch, an Arduino time/date library must be installed. Download the time library from http://www.pjrc.com/teensy/td_libs_Time.html or from https://github.com/PaulStoffregen/Time. Follow instructions on the website for installing the library. When you run the Arduino IDE for the first time, the IDE creates a folder Arduino/sketch/libraries. Look for this folder in your home documents folder. The Time library should be located there.
3. Using a USB cable connect the computer running the Arduino IDE to the Arduino Uno's USB port.
4. Follow the Arduino instructions for connecting the Uno to your computer and configuring the IDE for the Uno. Normally this means navigating to the Tools->Board menu item and selecting the Uno, and going to the Tools->Port menu item and selecting USB port to which the Uno is connected.
5. From the Arduino folder in the github project folder, download the Arduino sketch named *Radmon.ino*.
6. Open this sketch in the IDE.

7. Near the top of the sketch look for a line that looks similar to

   ```
   #define ETHERNET_MAC_ADDRESS 0x90, 0xA2, 0xDA, 0x0D, 0x84, 0xF6
   ```

   Change the hardcoded MAC address to the MAC address provided with your Ethernet shield.
8. Compile the sketch. If the Time library was installed properly, and everything else done correctly, the sketch should compile cleanly without errors.
9. Upload the compiled sketch to the Arduino Uno.

This completes the radiation monitor software installation. Before the radiation monitor can be connected to your network, the Ethernet interface must be properly configured. Follow the instructions below to set up the radiation monitor's network interface.

Configuring the Radiation Monitor

1. The radiation monitor should already be connected to your computer from step 4 in the section "Software Installation". If not, then do step 4 above.
2. Close the Arduino IDE.
3. This step depends on what OS your computer is running. If you are running Microsoft Windows, you will probably have to use Putty, or something equivalent, to set up a terminal session with the radiation monitor. If you are running Linux or OS X, you can use a screen session. However you will need to tell screen to which device the radiation monitor is connected, and this will vary with the OS. Normally, look in the /dev folder to find the usb device name. On Ubuntu systems type the command '**ls /dev/tty* | grep ACM**'. Then open a screen session by typing '**screen -S rad {dev path}**', where '{dev path}' is the device path determined above.
4. If you have successfully connected to the Uno, you will see a few lines of text showing the software version number and copyright notice. You may see a synch failed message. This is normal when the Radmon sketch is downloaded to the Arduino for the first time.
5. Press the lower case 'c' character to enter command mode. A menu of command mode options will appear. Please note that the radiation monitor will not respond to HTTP requests while in command mode. You must exit command mode, as described below, for the radiation monitor to respond to HTTP requests.
6. Type '2' and press enter. If you want to assign the radiation monitor a static IP address then enter it now. If you want to use a DHCP assigned address, then simply press enter. (See your network administrator for what IP address to use.)
7. Type '3' and press enter. If you want to enter a specific NTP server IP address, then enter it now. If you want to use the default NTP server IP address, then simply press enter.
8. Type '1' and press enter. Verify that the settings are as you want them. If they are not, the repeat steps 6 thru 8.

9. If the proposed settings are correct, type ‘6’ and press enter. At any time all changes can be discarded and the radiation monitor returned to normal mode by typing ‘5’ and pressing enter.

This completes configuring the radiation monitor.

Installing the Server Software

Dependencies

• The server software should be installed on a recent Linux distribution such as Ubuntu. (The author has successfully developed and installed the software on a Raspberry Pi running the Raspbian operating system.)
• Apache 2 should be installed and configured to allow serving HTML documents from the user’s public_html folder.
• Rrdtool should be installed - type **sudo apt-get install rrdtool**
• Python 3 usually comes pre-installed in virtually all Linux distributions. Type “python” at a command line prompt to verify Python has been installed.

Software Inventory

The software items that need to be installed on the server are

HTML folder:
• radmon.html
• index.html
• static/chalk.jpg

Bin folder:
• createRadmonRrd.py
• radmonAgent.py
• radstart
• radstop

Installation

Note that the following installation procedure assumes that radmon HTML documents will reside in the user’s public_html directory. Typically the full path name to this directory will be something like “/home/[user_name]/public_html”.

1. Use **cd** to change directory to the public_html directory. In the public_html directory, use **mkdir** to create a directory **“radmon”** to contain the radmon HTML files.
2. Change directory to the radmon directory created in step 1. In the radmon directory, use **mkdir** to create a directory **“dynamic”** to contain dynamic content used by html documents.
3. From the html folder found in the github radmon project folder, download all files and folders to the directory created in step 1.

4. The input and output data files (see figure 4) get overwritten frequently; similarly the graphic image files get overwritten frequently. On SD card systems, such as a Raspberry Pi, it is inadvisable to do frequent writes to any file system mounted on the SD card. To use the ram based temporary file system (tmpfs) to store these files, continue with step 5. Otherwise, to write dynamic content to the disk drive, continue with step 7.
5. Assure that the server /tmp directory gets mounted to the temporary file system. This can be done by adding the following line to the **/etc/fstab** file

   ```
   tmpfs /tmp tmpfs nodev,nosuid,size=20M 0 0
   ```

6. The HTTP service cannot freely access files and directories outside of the public_html document root directory. Therefore, the dynamic directory must be bound to the temporary file system by running the following commands

   ```
   mkdir /tmp/radmon
   sudo mount --bind  /tmp/radmon  /home/$USER/public_html/radmon/dynamic
   ```

   The above two commands may be placed in a startup shell script and run at boot up time by launching the script with the **su** command from the **/etc/rc.local** script, e.g.,

   ```
   (su - pi -c "bin/startup.sh")&
   ```

7. If it does not already exist, use **mkdir** to create a directory named **"bin"** in the user home directory. For example, the full path name should look something like "/home/[user_name]/bin'.
8. From the bin folder in the github project directory, download the files into the bin directory created in step 7. In most Linux installations the user's bash profile will automatically add the user's bin directory to the command search path. If such is the case, then the agent can be started up by simply typing **radmonAgent.py** followed by ENTER. Note that you will need to edit **radmonAgent.py**, adding the url for your radiation monitor.
9. In the user home directory use **mkdir** create a directory named "database". Run the python script **createRadmonRrd.py** downloaded in step 8. Running this script creates an empty round robin database file where the agent will store radiation data as it arrives from the radiation monitor. This script should be run once and then kept in a secure place. Running it accidentally at some future date will result in *total loss of all previously stored data*.
10. In the user home directory use **mkdir** to create a directory named "log".
11. For connivence two scripts have been provided to make it easy to turn the agent on and off. The **radstart** script starts up the agent and causes diagnositc output and error messages to be written to a log file in the directory created in step 10. The **radstop** script stops the agent from running.

12. In the **startrad** script look for a line like the following

    ```
    MONITOR_URL=”{your monitor url}”
    ```

    Replace “{your monitor url}” with the url for your radiation monitor. The monitor url may look something like http://192.168.1.155, or look something like http://radiationMonitor.domain.com, depending on whether your local network uses a domain name server.

This completes installation of the server software.

References and Resources

The following sources should be consulted before building the radiation monitor:

- Arduino Uno - www.arduino.cc/en/Main/ArduinoBoardUno
- Mighty Ohm Geiger Counter - mightyohm.com/blog/products/geiger-counter/

The following tutorials are useful for more in depth understanding of programming Arduino:

- C programming - www.gnu.org/software/gnu-c-manual/
- Arduino programming - www.arduino.cc/en/Tutorial/Foundations

The following tutorials are useful for more in depth understanding of the server software:

- Java Script - www.w3schools.com/js/default.asp
- jQuery - www.w3schools.com/jquery/default.asp
- PHP - www.w3schools.com/php/default.asp
- HTML - www.w3schools.com/html/default.asp
- Rrdtool - oss.oetiker.ch/rrdtool/
- Python - greenteapress.com/thinkpython/thinkpython.html

Appendix

The following steps describe how to build a web server on a Raspberry Pi. The steps below configure the Raspberry Pi to serve web pages from the user's document root folder. When user html folders are enabled, Apache looks for the files in the user's **public_html** folder. For example, a browser request **http://raspi.local/~pi/myDocument.html** would cause the host **raspi.local** to look for **myDocument.html** in the folder **/home/pi/public_html**. With the setup in the steps below, documents can also be served virtually, as if from the host's **/var/www** folder. For example, **http://raspi.local/myDocument.html** would also cause Apache to look for **myDocument.html** in the folder **/home/pi/public_html**.

1. If upgrading from a previous Raspbian version, first backup the user home folder by running

   ```
   sudo tar -zcpvf /home/pi_bak.tar.gz /home/pi
   ```

   Copy the **pi_bak.zip** file to a thumb drive or some other external storage media.

2. Copy the new Raspbian OS disk image to the SD card. Follow raspbian instructions for copying the disk image to the SD card.

3. After copying the disk image to the SD card, mount the SD card **boot** partition on the computer you are using to set up the Raspberry Pi. Navigate to the root of the **boot** partition and create an empty file named **ssh**. This will enable setting up the Raspberry Pi without needing to attach a keyboard, mouse, and monitor.

4. If you are connecting to the Pi via wifi, also in the root of the **boot** partition, create a file named **wpa_supplicant.conf** and add the following lines. If you are connecting via wired Ethernet skip this step.

   ```
   ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
   update_config=1
   country=US
   network={
      ssid="your_wifi_network_name"
      psk="your_wifi_network_password"
      key_mgmt=WPA-PSK
   }
   ```

5. Unmount and remove the SD card.

6. Insert the SD card into the Pi and connect the Pi to its power source. The Pi will boot up.

7. Once the Pi has booted, on your computer open a terminal window and run the command

   ```
   ssh pi@raspberrypi.local
   ```

   The default password is **raspberry.**

8. Run the command

   ```
   sudo raspi-config
   ```

   In System Options change

   Hostname to your host name
   Password to your password

   In Localisation Options change

   Locale to en_US.UTF-8 UTF-8
   Timezone to your timezone
   Keyboard US

9. Reboot the Pi by running

   ```
   sudo reboot
   ```

10. Use ssh to login as user pi with the password set in step 8

    ```
    ssh pi@{your host name}.local
    ```

11. Update the package database and install vim

    ```
    sudo apt-get update
    sudo apt-get install vim
    ```

    vim is an editor that makes it easy to make changes in configuration files. Alternatively you can use **nano** or some other terminal based editor of your choice.

12. [Optional] On the client computer, create an ssh key pair. In the Raspberry Pi user **pi** home folder, create a folder named **.ssh**. In the **.ssh** folder create a file named **authorized_keys**, and copy the public key to it. For more information on how to create and use key pairs see https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent .

13. Note that all procedures below that involve editing a system file will have to be done acting as superuser. For example, to edit the file in step 14 run the command

    ```
    sudo vi /boot/config.txt
    ```

14. If they will not be used, disable WiFi and Bluetooth. Backup and then modify **/boot/config.txt** by adding the following two lines to the end of the file:

    ```
    dtoverlay=pi3-disable-wifi
    dtoverlay=pi3-disable-bt
    ```

    and run the once off command:

    ```
    sudo systemctl disable hciuart
    ```

15. [Optional] For added security turn off password authentication. You must have set up ssh keys as discribed in step 12. *Do not do this step unless you are familiar with ssh keys and how they work.* You will not be able to login via ssh to the Pi with a password. The above enhances security when using secure shell (ssh) to access the Raspberry Pi. Backup and then modify **/etc/ssh/sshd_config** as follows

    ```
    #PasswordAuthentication yes
    PasswordAuthentication no
    ```

16. Setup the temporary file system (tmpfs) by backing up and then modifying **/etc/fstab**. Add the following lines to the bottom of the file.

    ```
    # add these lines, if necessary, for web apps to store logs in ram to reduce
    # stress on the SD card due to frequent writes.
    tmpfs /tmp tmpfs nodev,nosuid,size=10m 0 0
    tmpfs /var/tmp tmpfs defaults,noatime,nosuid,size=10m 0 0
    tmpfs /var/log tmpfs defaults,noatime,nosuid,mode=0755,size=10m 0 0
    tmpfs /var/spool/mqueue tmpfs defaults,noatime,
            nosuid,mode=0700,gid=12,size=10m 0 0
    ```

    Highly recommended, this step configures the Raspberry Pi to store all log files and temporary files in RAM. Unless configured to use an external hard drive, the Raspberry Pi mounts the root file system to the SD card. Log files and temporary files are frequently written to the file system, resulting in wear on the SD card. Storing these files in RAM saves the SD card from such wear. Note that all log files and temporary files are lost upon reboot or power down.

17. Reboot the Raspberry Pi.

18. Use ssh to login as user pi with the password set in step 8

    ssh pi@{your host name}.local

19. [Optional] Run all software updates

    sudo apt-get upgrade
    sudo reboot

20. Backup **/etc/rc.local** and then add the user start up script. For example add the following line to **/etc/rc.local**

    (su - pi -c "bin/startup.sh")&

21. Install rrdtool

    sudo apt-get install rrdtool

    rrdtool maintains round-robin databases and generates charts for display in web pages. **rrdtool** can run on the same host as **Maria** (mySql). However they are not interoperable with each other.

22. Install LAMP. This is the standard Linux web server “stack”. For more information about installing LAMP see https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md . Run the following commands

    Apache
    ======
    sudo apt-get install apache2 -y
    sudo a2enmod rewrite
    sudo systemctl restart apache2

    PHP
    ===
    sudo apt-get install php libapache2-mod-php y
    sudo systemctl restart apache2

    [Optional] SQL
    ===========
    sudo apt install mariadb-server mariadb-client php-mysql -y

23. Backup and then modify **/etc/apache2/mods-available/userdir.conf** to allow user **.htaccess** files. For example,

    # changed 2020-01-15 by JLO to allow user .htaccess file

```
#AllowOverride FileInfo AuthConfig Limit Indexes
AllowOverride All
```

This enables apache to use the .htaccess file in the user's document root folder.

24. Enable user directories in Apache by running

    a2enmod userdir

25. Enable php in Apache by running

    a2enmod php7.3

26. Backup and then modify **/etc/apache2/mods-available/php7.3.conf** to allow PHP scripts to run in user directories by commenting the lines at bottom of file. For example,

```
# changed 2020-01-15 by JLO to enable user php scripts
#<IfModule mod_userdir.c>
#	<Directory /home/*/public_html>
#		php_admin_flag engine Off
#	</Directory>
#</IfModule>
```

27. Backup and then modify **/etc/apache2/sites-available/000-default.conf** to make the pi user **public_html** folder the web server document root. For example,

```
# changed 2020-01-15 by JLO to make user
# pi the html document root
#DocumentRoot /var/www/html
DocumentRoot /home/pi/public_html
```

This makes the pi user public_html folder the document root for the Raspberry Pi. For example a client request **http://raspi.local/myDocument.html** would cause Apache to look for the document in the folder **/home/pi/public_html**.

28. Backup and then modify **/etc/apache2/envvars** to create Apache logs in tmpfs. Add the following lines at the top of the file

```
# added 2020-01-15 by JLO to allow apache to use tmpfs
if [ ! -d /var/log/apache2 ]; then
	mkdir /var/log/apache2
fi
if [ ! -d /var/log/mysql ]; then
	mkdir /var/log/mysql
fi
```

Without these lines Apache will fail to start when the system reboots. When the system reboots the folders **/var/log/apache2** and **/var/log/mysql** will not exist if logs are using the temporary file system in RAM. Apache will fail to start if these folders are not present. Adding the above lines causes Apache to create these folders when the system boots up.

29. Enable Apache to access files in the the tmpfs **/tmp** folder by backing up and then modifying **/lib/systemd/system/apache2.service** as follows:

    ```
    # changed {date} by {name} to enable apache to follow
    # symlinks to the /tmp folder in tmpfs
    #PrivateTmp=true
    PrivateTmp=false
    ```

    This change enables Apache to follow symbolic links in the user document root (public_html) folder to folders and files in the temporary file system.

    Reload system deamons

    ```
    sudo systemctl daemon-reload
    ```

    Restart Apache service

    ```
    sudo systemctl restart apache2
    ```

    This change enables Apache to follow symbolic links in the user document root (public_html) folder to folders and files in the temporary file system **/tmp**.

30. Reboot the Raspberry Pi.

31. Copy the backup file pi_bak.zip from the thumb drive media or other external media to the **/home/pi** folder. Restore desired files and folders from backup archive by running

    ```
    tar -zxvpf pi_bak.tar.gz
    ```

    Use mv to move folders and files to their appropriate locations.

32. Test all the above modifications.