Water Board Deployment Guide

MIT CityFARM

This guide focuses on the deployment of a Water Board, from installing the operating system on the Raspberry Pi, to configuring and setting up the board to make measurements.

Items needed:

1. Water Board Working network
2. Internet connection
3. Server running database and web management interface
4. Server URL or IP Address written down
5. Linux computer with SD Card reader
6. 8Gb SD Card and Raspberry Pi

Steps:

1. Files and information needed.

1.1 To quickly get up and running, it is possible to use the MIT CityFARM’s prebuilt Arch Linux, and Image Preparation Utility, however any Linux system may be used.

The prebuilt Arch Linux images can be downloaded on CityFARM’s Google Drive, and the utility and required files can be downloaded from GitHub on the following links:

Image: <https://drive.google.com/file/d/0Bzu5DJ7GsPj5WG1rY1U2VHl6czA/view>

Utilities & Files: <https://github.com/MIT-CityFARM/sensor-board>

By downloading the contents of the *“demo”* folder on GitHub, along with the image file, all the needed files will be present.

1.2 Demo folder structure:

- RPi [1]

- RPi\_Air [2]

- docs

- web\_management

- image\_utility.py

- arch\_linux\_...\_.img (copied after download and unzip)

1.3 Determining the path of the SD Card.

To Determine the SD Card’s path, first disconnect the SD Card. Then on a terminal window, list the files on *‘/dev’*:

*user$ ls /dev*

Which will output something similar to the following:

bus loop1 sda2 tty0

cdrom pts sda5 ttyUSB0

cdrw ram0 stderr

dvd sda stdin

dvdrw sda1 stdout

Now insert the SD Card, and execute the command again:

*user$ ls /dev*

bus loop1 sda2 stderr

cdrom pts sda5 stdin

cdrw ram0 sdb stdout

dvd sda sdb1 🡨 tty0

dvdrw sda1 sdb2 🡨 ttyUSB0

Observe that the disk ‘sdb’, and, some of its partitions (sdb1 and sdb2) have appeared on the *‘/dev’* folder. This means that (in this case) *‘/dev/sdb’* is the SD Card, and the path we are looking for.

Note that *‘sdb’* is an example, which may vary according to each system.

1. Configuring Arch Linux image and burning image on SD Card.

2.1 We can now proceed to executing the *“image\_utility.py”* and start configuring the Arch Linux image.

On a terminal window, go to the demo directory from step 1.2:

*user$ cd /path/to/demo/*

You may check you’re in the right folder by listing the files:

*user$ ls*

arch\_linux\_...\_.img

docs

RPi

RPi\_Air

Web\_management

Image\_utility.py

2.2 Execute *“image\_utility.py”:*

The *“image\_utility.py”* takes 5 optional options, and 1 or 2 required arguments (depending on options):

*image\_utility [--water] [--air] [--no-burn] [--no-unmount] [--only-burn] SOURCE\_IMAGE [TARGET\_DISK]*

Source Image: Arch Linux image on the current folder

Target Disk: The SD Card path, determined in step 1.3

Options (more detailed descriptions inside image\_utility.py file):

--water: Selected by default. Creates image for Water Board.

--air: Creates image for Air Board only. Must specify ‘--water’ if both needed.

--no-burn: Only creates custom image. Does not burn to disk (Do not define TARGET\_DISK).

--no-unmount: Leaves image mounted after creation. Does not burn to disk (Do not define TARGET\_DISK).

--only-burn: Burns source image to disk. No customization (It’s assumed image has already been customized)

On the same terminal window, execute the command­ [3]:

*user$ sudo ./image\_utility.py archlinux\_...\_ARCH.img /dev/sdb*

[sudo] password for user:

The command must be run as superuser, hence the *‘sudo’* command at the beginning, it’ll then ask for the user’s password. Type the password and hit enter.

The utility will ask for the following information:

Enter the database server URL or IP Address:

Enter the database username:

Enter the database password:

Enter the required information, which should be known beforehand.

Breakdown of information:

- Server URL/IP: URL or IP Address to the server running the database used for storing sensor data and settings.

- Username and Password: Login information for authentication on the database, this is not the server’s user login information.

The Login information will be used to store and retrieve the board’s settings, log, and sensor data, so it is required. If the information is incorrect it will not be possible to configure the board through the web interface, nor will the board be able to upload sensor data.

After entering the 3 pieces of information required, the utility will proceed to mount the image, copy the files, set the permissions, add the server information then unmounts the image. This is the typical output during this process:

Target Disk: /dev/sdb

Source Image: archlinux\_cityfarm\_image\_3\_12\_34\_ARCH.img

All files found on: './RPi/'

Press Enter to continue...

Press Enter if the information is correct

Mounting image...

umount: mount\_point\_archlinux\_cityfarm\_image\_3\_12\_34\_ARCH.img: not found

rmdir: failed to remove ‘mount\_point\_...\_.img’: No such file or directory

Mounting succesful

Copying files to: 'mount\_point\_...\_.img/root/RPi’

config.json

rpi\_service.py

rpi\_service.service

rpi\_sh.sh

sensor\_terminal.py

serialsensor.py

Setting permissions on: 'mount\_point\_...\_.img/root/RPi’

Adding connection settings to config file.

Unmounting image.

Press Enter to continue burning image to disk...

The error on line 2 is simply because the utility tries to unmount any images that have not previously been unmounted, the user may simply ignore it.

If all information is satisfactory, the user may press enter to burn the newly created custom image to the SD Card.

Unmounting disk partitions (ignore errors).

Burning image to SD Card at /dev/sdb

Image burning may take as long as 15 minutes, please wait.

Running: dd if=archlinux\_cityfarm\_image\_3\_12\_34\_ARCH.img of=/dev/out bs=1M

The burning process may take as long as 15 minutes (If using a virtual machine, this process may take even longer).

Once the process is complete, the disk will be automatically ejected, and can be removed.

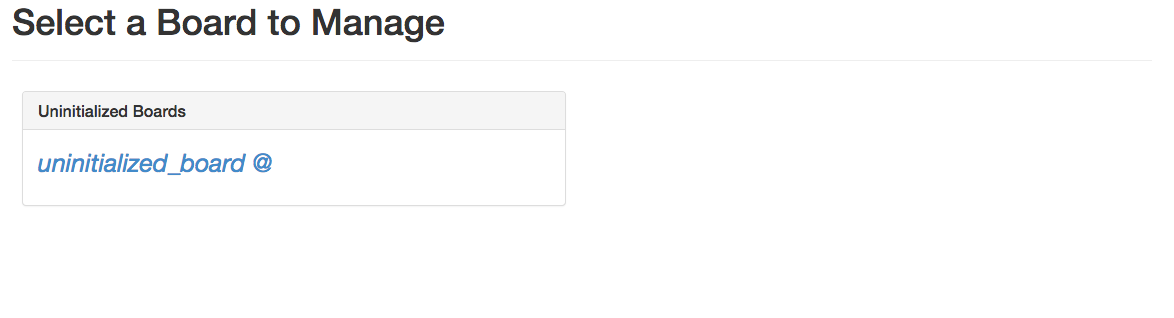
Alternatively the utility may be used only to create the custom image (using the --no-burn flag), after that you may use the utility (with the --only-burn flag) to burn the same image to several boards.

1. Setting up Raspberry Pi, and configuring it through the web interface.

3.1 Plug in the SD Card and connect an Ethernet cable and power to the Raspberry Pi.

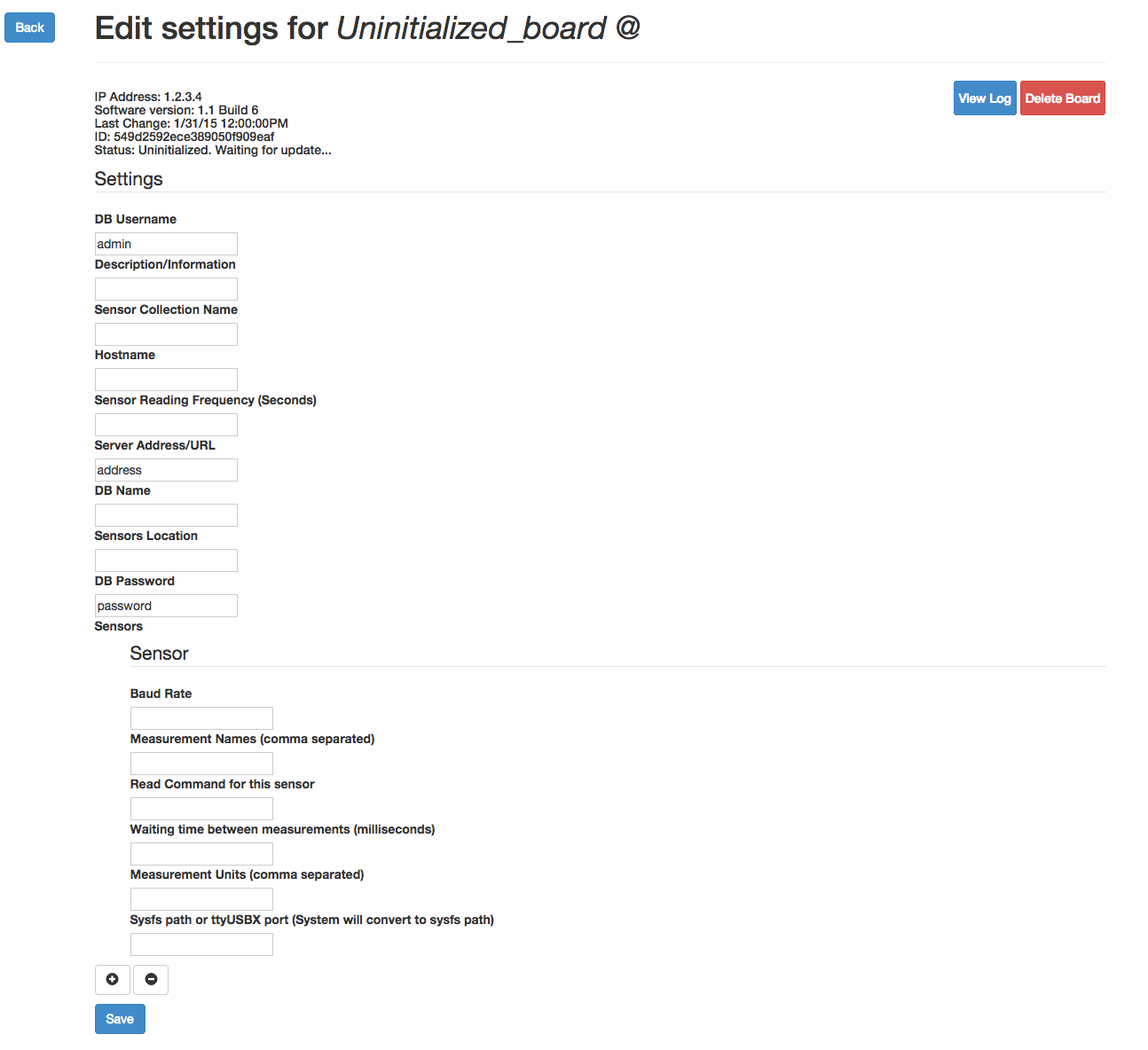
The Raspberry Pi will boot up, and the *“rpi\_service.py”* will be executed automatically, a new empty configuration file will be added to the database after a few minutes (at the *‘admin’* database, *‘boards’* collection) and the configuration may now be edited through the web interface.

3.2 Go to the web interface’s address, and the following page should appear:



It is recommended that one board be added to the system at a time, so as the user may know which board is being deployed. All boards will show up as *‘uninitialized\_board’*

Select the uninitialized board, and the settings page for that board will appear:



In this page you can edit the boards settings and add/remove/edit sensor settings. The settings available for editing are:

* DB Username [4]
* DB Password [4]
* Description
* DB Name [5]
* Collection Name [5]
* Hostname [6]
* Sensors Location
* Server Address [4]
* Sensor Reading Frequency [7]

Sensor settings:

* Measurement Units [8]
* Measurement Names [8]
* Baud Rate
* Waiting Time
* Read Command [9]
* Sysfs path or /dev/ttyUSBx path [10]

The status bar on the top of the page provides information about the board being managed, such as:

* IP Address (which may be used for ssh)
* Software Version
* Last Settings Change
* ID (given by the configuration entry in the database)
* Status [11]

This page also provides access to the current log (*“View Log”*) and a reset button to delete the board from the DB, along with the log.

An example setup is provided at the end of this document [12]

Once saved, it may take a few seconds for the board to reload the changes. Once that happens, if the page is refreshed, *“Initialized”* should appear in the status report. And, provided all required information is correct, the board should start reporting data. The log can always be checked to debug any problems that may arise.

1. Maintenance

4.1 SSH

SSH is a useful tool to log into a remote Linux system. The IP address of the board (or it’s hostname) can be used for SSH, and both are available through the web interface.

Using SSH:

*user$ ssh root@ip\_address*

Where *root* is the username used to log in to the specified address.

A prompt will appear asking if you wish to continue, just type *yes* to continue. Another prompt will ask for the password, type the password and hit enter. You will then have logged in to the board remotely, and can now execute commands to it.

Note:

The default username and password for the prebuilt image are:

* Username : “root”
* Password: “root”

It is strongly recommended that a new password be set, to do so use *passwd:*

*user$ passwd*

Confirm the current password and enter a new one.

Notes:

- This password is not the same as the one set on the web interface. This is the root user password of the operating system running on a certain board. The password set on the image utility, and available on the web interface, correspond to the password needed to connect to the database.

- Changing this password will not affect the behavior of the board.

[1] Description of contents of *‘RPi’* folder:

*config.json* – Base, json formatted, settings file where all information (database, sensors and general settings) are stored locally. Initially only server username and password fields are set (by the image utility), once the board has been initialized on the web management interface, all settings saved will be also saved on the local ‘*config.json’* file. If a board has already been initialized and is introduced in a new server, all information contained in this file will be uploaded to the database, and initial configuration will not be needed.

rpi\_service.py – Main script that handles data acquisition and uploading. This script is automatically initialized after boot.

rpi\_service.service – Service file used on Arch Linux *‘systemctl’* daemon manager. The file itself is not needed, since it has already been copied and linked enabled on the preconfigured image, however it is present for reference.

rpi\_sh.sh – Shell script called by *‘rpi\_service.service’* , since a python script cannot be directly called by a *‘systemctl’* service. This file is needed.

serialsensor.py *–* Serial Sensor library used by *‘rpi\_service.py’* to communicate with the serial sensors. More information about this library can be found on GitHub (https://github.com/MIT-CityFARM/sensor-board)

[2] Description of contents of *‘RPi\_Air’* folder:

Same as [1] except for:

rpi\_service\_air.service – Same as for [1] except that ‘ExecStart’ points to ‘RPi\_Air’ folder

rpi\_sh.sh – Same as for [1] except that it calls rpi\_service.py on the ‘RPi\_Air’ folder.

[3] More detailed description of ‘image\_utility.py’ is present inside the file itself.

[4] DB Username, Password and Server Address correspond to the address and authentication info of the MongoDB server. If username and password are left empty, it is assumed that the server does not require authentication.

[5] DB and Collection name correspond to the database and collection on MongoDB that will receive the data from the sensors.

[6] Hostname is the hostname of the board, it does not affect the behavior of the board, however it is useful for SSH, instead of using IP addresses. Both can be obtained through the web interface. If a new hostname is saved the board will reboot to apply changes.

[7] Sensor reading frequency is the frequency in which all of the sensors are read once and the set of data is uploaded to the database. If the frequency is lower than the minimum running frequency, an alert will be shown in the log.

[8] Measurement Units and Names. Names correspond to one or multiple (comma separated) names of the measurements being produced by the sensor in question. Units correspond to the units of those measurements, which can be, as well, one or several comma separated values.

More information can be found on the documentation: ‘docs/ SerialSensor.rst’

[9] Read command is the ASCII string the sensor needs to receive in order to reply with a measurement. Wait time corresponds to the time it takes to receive such reply after the read command is sent.

[10] Sysfs path is the path to de USB device in Linux systems. The user is not expected to know this value, in this field, the user can also enter the ‘/dev/ttyUSBx’ port the sensor is connected to, in this case the system will find the respective sysfs path and replace and save it to the settings file. The reason for this is because ‘/dev/ttyUSBx’ ports are not associated with a specific device, changing with the order each device is connected, whereas the sysfs path corresponds to the path to a specific USB port.

[11] Example initial configuration of board:

