# Requirements

Raspberry Pi model 2B +

Arduino Mega 2560

*optional* *Insta****push****.im* account

*optional* *elasticsearch* index

## Raspberry Pi Setup

* Rasbian Linux RPi 4.9.35+
* Python v2.7
* NodeJs v4.2.1
* npm v2.14.7

NodeJs modules

* body-parser v1.15.2
* elasticsearch v11.0.1 (es 2.3)
* express v4.14.0
* jade v1.11.0
* request v2.72.0
* serialport v4.0.0
* socket.io v1.4.8

# Installation

## Home Dashboard

1. Copy the files to /home/pi/prod/dashboard directory.
2. Configure the aplication
3. Create the system service to start the dashboard.
4. To test the aplication you can start it by

node /home/pi/prod/dashboard/dashboard.js

## Home Dashboard configuration

Several items can be configured in the application by editing the **dashboard.js** file.

You can configure the

* application port (default 3333)
* the serial interface of Arduino (default /dev/ttyACM0)
* the elasticsearch instance (default <http://elasticlog-fmefi.rhcoud.com/)>

#!/usr/bin/env node

var events = require("events");

var express = require('express');

var bodyParser = require('body-parser');

**var appPort = 3333;**

var app = express();

var http = require('http');

var server = http.createServer(app);

var socket = require('socket.io');

var io = socket.listen(app.listen(appPort));

var SerialInterpreter = require('./SerialInterpreter.js');

var PalisadyModel = require('./PalisadyModel.js');

var IsMessage = require('./IsMessage.js');

var NamesMap = require('./NamesMap.js');

var ElasticsearchLogger = require('./ElasticsearchLogger.js');

var NetDeviceScanner = require('./NetDeviceScanner.js');

var namesMap = new NamesMap();

var stdin = process.openStdin();

//var serial = new SerialInterpreter('/dev/tty.usbserial-AH01GKWI');

//var serial = new SerialInterpreter('/dev/tty.usbmodemfa131');

**var serial = new SerialInterpreter('/dev/ttyACM0');**

var model = new PalisadyModel();

**var esl = new ElasticsearchLogger('http://elasticlog-fmefi.rhcloud.com/');**

var netScanner = new NetDeviceScanner(model, esl);

netScanner.startScanning();

## Creating Raspberry Pi system services

The service for *Home Dashboard* and *Home Monitor* needs to be created.

*Home Dashboard* is the main service providing the REST interface to query and control the lights and heating status.

*Home Monitor* defines rules on events that are monitored and if triggered a Push notification is sent to Instapush.im service.

Both the services are started on Linux boot.

### Home Dashboard Service

1. Create file /usr/lib/systemd/system/home\_dashboard.service

With content

[Unit]

Decription=Home Dashboard Service

After=multi-user.target

[Service]

Type=simple

ExecStart=/usr/local/bin/node /home/pi/prod/dashboard/dashboard.js > /home/pi/prod/dashboard/dashboard.log 2>&1

Restart=on-abort

[Install]

WantedBy=multi-user.target

2. Set the file properties with commands

sudo chmod 644 /lib/systemd/system/home\_dashboard.service

chmod +x /home/pi/prod/dashboard/dashboard.js

sudo systemctl daemon-reload

sudo systemctl enable home\_dashboard.service

sudo systemctl start home\_dashboard.service

### Home Monitor Service

1. Create file /usr/lib/systemd/system/home\_monitor.service

With content

[Unit]

Description=Home Monitor service to send push notifications from Home Dashboard

Alter=multi-user.target

[Service]

Type=simple

ExecStart=/usr/bin/python /home/pi/prod/dashboard/home\_monitor.py > /home/pi/prod/dashboard/home\_monitor.log 2>&1

Restart=on-abort

[Install]

WantedBy=multi-user.target

2. Set the file properties with commands

sudo chmod 644 /lib/systemd/system/home\_monitor.service

chmod +x /home/pi/prod/dashboard/home\_monitor.py

sudo systemctl daemon-reload

sudo systemctl enable home\_monitor.service

sudo systemctl start home\_monitor.service

### How to check the service status

sudo systemctl status home\*.service

### How to check the service logs

sudo journalctl -f -u home\_monitor.service

Background information on how to create a service is from

<http://www.diegoacuna.me/how-to-run-a-script-as-a-service-in-raspberry-pi-raspbian-jessie/>

# APPENDIX A – HOW TO RUN A SCRIPT AS A SERVICE IN RASPBERRY PI - RASPBIAN JESSIE

[**2 YEARS AGO**](http://www.diegoacuna.me/how-to-run-a-script-as-a-service-in-raspberry-pi-raspbian-jessie/)**•**[**IOT**](http://www.diegoacuna.me/category/iot/)**,**[**LINUX**](http://www.diegoacuna.me/category/linux/)**•**[**8**](http://www.diegoacuna.me/how-to-run-a-script-as-a-service-in-raspberry-pi-raspbian-jessie/)

The [Raspberry Pi](https://www.raspberrypi.org/) is an incredible popular credit card size mini computer with awesome capabilities.  Despite of is mini size it behaves pretty much like a regular computer (it has an OS with a graphical interface, an ethernet port so you can surf the web, an HDMI output, etc.) but also it exposes several inputs/outputs (GPIO) to interact with the real world (sensors and cool tech gadgets).

A pretty common task when using this device, is to run some script (for example a python script) as a service in the operating system so it can start on boot, stop and restart using systemctl and more. In this post I'm going to explain how to set a little script as a service using Raspbian Jessie in a Raspberry Pi.

First of all, we are going to write a small python script which print "Hello World" every 60 seconds. This is going to be our service script (hello\_world.py):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #!/usr/bin/python    from time import sleep    try:      while True:          print "Hello World"          sleep(60)  except KeyboardInterrupt, e:      logging.info("Stopping...") |

You can execute it by python hello\_world.py. If you get boring reading so many hello worlds, press Ctrl+C (or Cmd+C on OSX) to stop it. Save this file as hello\_world.py in your home folder (home/pi/). Now we're going to define the service to run this script:

|  |  |
| --- | --- |
| 1  2 | cd /lib/systemd/system/  sudo nano hello.service |

The service definition must be on the /lib/systemd/system folder. Our service is going to be called "hello.service":

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | [Unit]  Description=Hello World  After=multi-user.target    [Service]  Type=simple  ExecStart=/usr/bin/python /home/pi/hello\_world.py  Restart=on-abort    [Install]  WantedBy=multi-user.target |

Here we are creating a very simple service that runs our hello\_world script and if by any means is aborted is going to be restarted automatically. You can check more on service's options in the next wiki: <https://wiki.archlinux.org/index.php/systemd>.

Now that we have our service we need to activate it:

|  |  |
| --- | --- |
| 1  2  3  4  5 | sudo chmod 644 /lib/systemd/system/hello.service  chmod +x /home/pi/hello\_world.py  sudo systemctl daemon-reload  sudo systemctl enable hello.service  sudo systemctl start hello.service |

For every change that we do on the /lib/systemd/system folder we need to execute a daemon-reload (third line of previous code). If we want to check the status of our service, you can execute:

|  |  |
| --- | --- |
| 1 | sudo systemctl status hello.service |

In general:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # Check status  sudo systemctl status hello.service    # Start service  sudo systemctl start hello.service    # Stop service  sudo systemctl stop hello.service    # Check service's log  sudo journalctl -f -u hello.service |

REFERENCES

1. <https://wiki.archlinux.org/index.php/systemd>
2. <https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files>
3. <https://coreos.com/os/docs/latest/getting-started-with-systemd.html>