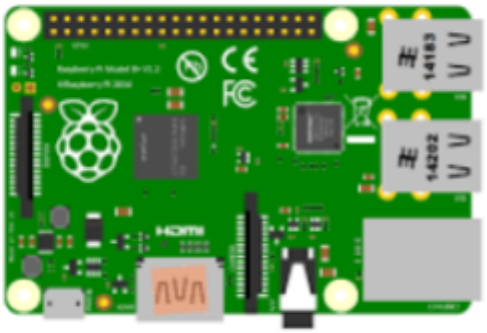


RPI Beacons Scan & CMS Content

por Agustin Bassi 04 Octubre 2019 rpi raspberry beacons ble cms

Tabla de contenido

Objetivos

Los objetivos de este documento son:

- Entender con un ejemplo práctico el funcionamiento de los beacons BLE.
- Crear una aplicación para Raspberry Pi (Raspian) que pueda leer señales cercanas de beacons.
- Personalizar un CMS (Content Media Service) para mostrar información dependiendo el contexto.
- Mostrar en la pantalla de la Raspberry Pi contenido dinámico dependiendo del beacon leído.

> *La práctica está organizada de manera cronológica y para realizarla será necesario contar con una Raspberry Pi 3 (o superior)y una PC/Laptop con Bluetooth 4.0 (o superior) y que también actuará como CMS remoto.*

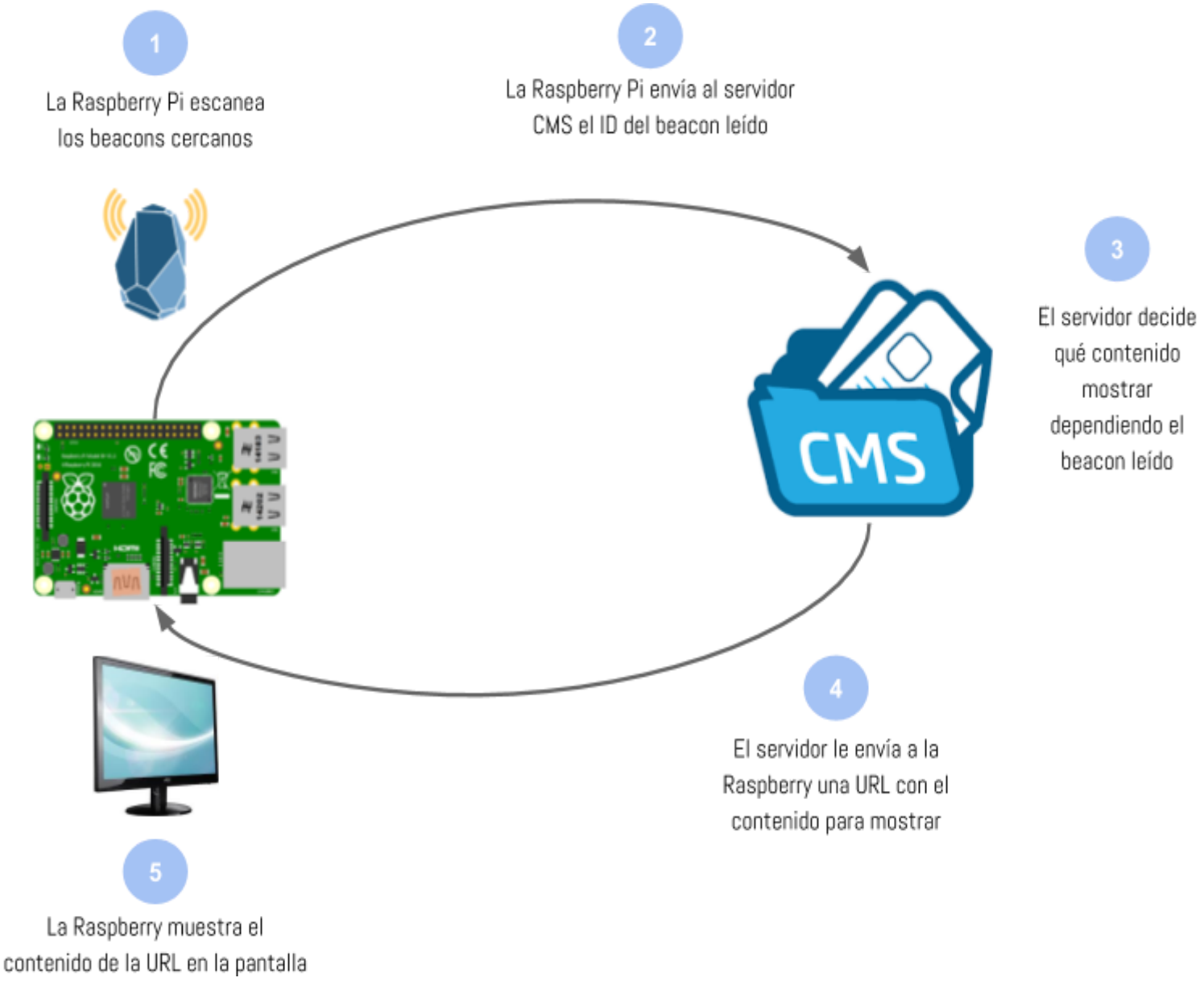
Introducción y propósito de la práctica

De la introducción sobre Bluetooth, BLE & Beacons se mostró que algunos de los posibles usos de los beacons es el marketing de publicidad. Esto significa que se puede mostrar contenido dinámico y relevante según la identificación del beacon más cercano.

La mayoría de las implementaciones están diseñadas para que el usuario instale una aplicación, y ésta en segundo plano, intenta realizar periódicamente la lectura de beacons cercanos para luego disparar una notificación al usuario con contenido relevante. En muchos casos lo más difícil es que el usuario quiera instalar una nueva aplicación en su teléfono.

Para el propósito de esta práctica, se realizará una aplicación sobre una Raspberri Pi conectada a una pantalla mediante HDMI, que estará constantemente leyendo los beacons más cercanos (con el Bluetooth integrado que posee la RPi) y en función de estas lecturas mostrará contenido relevante en la pantalla. En la aplicación de lectura de beacons no estará integrado el contenido, sino que estará administrado por un servicio externo que le brindará a la Raspberry el contenido a mostrar. De esta manera, la aplicación lectora de beacons no manejará la lógica asociada al contenido a mostrar, lo que brinda mayor flexibilidad a la hora de implementar una arquitecura de microservicios.

En la siguiente imagen se puede ver la arquitectura propuesta para esta práctica.



Emular Beacon con una PC con Bluetooth

Para testear que la lectura de los beacons se realice correctamente, en el caso que no se cuente con dispositivos Beacons, la mejor opción para este propósito es emularlos con una PC o smartphone (en ambos casos, los dispositivos deben contar con la versión de Bluetooth superior a 4.0 o Bluetooth Smart Ready).

> *En caso de contar con Hardware que tenga una versión de Bluetooth inferior a 4.0 no instalar ni intentar con ninguna aplicación, ya que no funcionará.*

Instalar paquetes de Bluetooth

El primer paso es instalar los paquetes necesarios para emular beacons desde una PC. Comenzar actualizando la lista de paquetes disponibles con el siguiente comando.

```
sudo apt-get update
```

Una vez actualizada la lista, instalar los paquetes de Bluetooth con los siguientes comandos.

```
sudo apt-get install -y bluetooth
sudo apt-get install -y blueman
sudo apt-get install -y bluez
sudo apt-get install -y bluez-tools
sudo apt-get install -y libbluetooth-dev
sudo apt-get install -y libcap2-bin
```

> *Si bien todos los paquetes no son necesarios, pueden servir a futuro para trabajar con Bluetooth y Beacons.*

Para chequear que se pueda acceder correctamente a la interfaz de Bluetooth de la PC, y las librerías de Bluetooth LE y Beacons estén instaladas correctamente, ejecutar el siguiente comando.

```
sudo hcitool lescan
```

La ejecución del comando debería mostrar la lista de dispositivos Bluetooth cercanos (si los hay). Lo importante es que no muestre un error al ejecutarse el comando, ya que indicaría que el soporte para Bluetooth Low Energy no está disponible. La salida del comando se ve como la siguiente.

```
LE Scan ...
37:A4:5C:10:F6:3F (unknown)
07:60:C7:9C:6D:1B (unknown)
26:5E:C6:92:06:DB (unknown)
26:5D:B9:6F:21:78 (unknown)
37:A4:5C:10:F6:3F (unknown)
07:60:C7:9C:6D:1B (unknown)
26:5E:C6:92:06:DB (unknown)
26:5D:B9:6F:21:78 (unknown)
```

Detener el comando con CTRL + C.

Emular Beacons - Opcion 1: Mediante un binario

Para este caso se utilizará un proyecto de código abierto llamado linux-ibeacon el cual es capaz de emular la emisión de paquetes iBeacons. Para ello, descargarlo con el siguiente comando.

```
git clone https://github.com/dburr/linux-ibeacon.git
```

Una vez descargado, moverse al directorio linux-ibeacon y ejecutar el siguiente comando.

```
sudo ./ibeacon --uuid=ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee --major=111 --minor=222
```

La ejecución del comando debería mostrar una salida como la siguiente.

```
Advertising on hci0 with:
       uuid: 0xFFFFFFFFBBBBCCCCDDDDEEEEEEEEEEEE
major/minor: 111/222 (0x006F/0x00DE)
      power: 200 (0xC8)
```

A partir de este momento, la interfaz Bluetooth de la PC se convertirá en un Beacon que emitirá constantemente paquetes iBeacons. En caso que quiera detenerse la emisión de paquetes iBeacon ejecutar el siguiente comando.

```
sudo ./ibeacon --down
```

Emular Beacons - Opcion 2: Mediante un script de Python

Para este caso se utilizará un proyecto de código abierto llamado beacon-emulator, que está basado en el proyecto de la opción anterior pero es más completo, ya que tiene la capacidad de emular distintos tipos de Beacons. El proyecto cuenta con distintos scripts de Python que emulan paquetes iBeacons, Eddystone y Konkat.

> *Para que los scripts se ejecuten correctamente será necesario contar con la versión de Python 3.6 o superior.*

Comenzar descargando el proyecto con el siguiente comando.

```
git clone https://github.com/wolfhardfehre/beacon-emulator.git
```

Luego, moverse dentro de la carpeta app y ejecutar el siguiente comando.

```
sudo python3 ibeacon.py --uuid=ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee --major=111 --minor=222
```

La ejecución del comando, al igual que en el caso anterior, arrojará la siguiente salida.

```
Advertising on hci0 with:
       uuid: 0xFFFFFFFFBBBBCCCCDDDDEEEEEEEEEEEE
major/minor: 111/222 (0x006F/0x00DE)
      power: 200 (0xC8)
```

Lectura de Beacons desde Raspberrry Pi

En la Raspberry Pi se debe instalar el sistema operativo, relizar las configuraciones para poner el escritorio en modo kiosco, instalar los paquetes relacionados con el Bluetooth y crear una aplicación en Python capaz de leer los beacons más cercanos.

Instalación de Raspian

La primer acción a realizar es descargar el sistema operativo en la Raspberry Pi desde su web oficial (versión de escritorio) y luego descomprimir el archivo ZIP. Una vez descargada la imagen del sistema operativo es necesario grabarla en una tarjeta SD de 16 GB o superior (preferentemente clase 10).

> *La grabación de la imagen en la tarjeta SD se realizará desde una PC con Linux Ubuntu, aunque se puede realizar desde cualquier sistema operativo.*

Insertar la tarjeta SD en la PC y ejecutar el siguiente comando para listar las unidades de almacenamiento.

```
lsblk
```

El comando devuelve las unidades de almacenamiento conectadas. Si la PC cuenta con lector de tarjetas SD integrado, la tarjeta SD aparecerá con el nombre **/dev/mmcblkX**. Si la SD es insertada mediante un lector USB, la misma será listada bajo el nombre **/dev/sdX**. (en ambos casos la letra **X** representa un índice).

Una vez detectada la memoria, desmontarla con el siguiente comando (para fines representativos, en este caso la memoria SD fue ingresada en el lector SD integrado de la PC y la misma fue listada con el índice 0).

```
umount /dev/mmcblk0
```

El siguiente paso es grabar la imagen del sistema operativo de Raspian en la SD. Ejecutar el siguiente comando.

```
sudo dd bs=4M if=name-of-image.img of=/dev/mmcblk0 conv=fsync status=progress
```

Esta acción llevará unos minutos. El flag **status=progress** hará que se muestre el progreso de la acción en la terminal. El argumento **name-of-image.img** es el nombre de la imagen de Raspian descargada. Una vez que el comando finaliza su ejecución, limpiar los buffers de salida con el siguiente comando y luego extraer la tarjeta SD.

```
sync
```

Configuración de Raspian

Insertar la tarjeta SD en la Raspberry Pi. La primera vez que se inicia el sistema operativo se deberán realizar algunas configuraciones iniciales. Configurar el Hostname (por ejemplo beacon_scanner), configurar un usuario y contraseña para ingresar por SSH (no dejar la configuración por defecto por motivos de seguridad) y configurar la red WiFi a la que debe conectarse para iniciar el escritorio. Una vez iniciado actualizar el sistema operativo con las últimas versiones de todos los paquetes con los siguientes comandos (esta acción llevará unos minutos).

```
sudo apt-get update
sudo apt-get upgrade
```

Rotar display (opcional)

En muchos casos, los kioscos son puestos con la pantalla en vertical. En esos casos será necesario modificar una configuración del sistema operativo de Raspian para realizar tal acción. Para ello abrir una terminal y ejecutar el siguiente comando para editar el archivo.

```
sudo nano /boot/config.txt
```

Ir hasta el final del archivo y escribir la configuración **display_rotate** y seleccionar la opción necesaria dependiendo cada caso **(0 = 0°, 1 = 90°, 2 = 180°, 3 = 270°)**. Guardar los cambios (presionando Ctrl+x) y luego presionar 'Y' para aceptar. Por último reiniciar el sistema con el siguiente comando.

```
sudo reboot
```

Configurar el navegador en modo kiosco

Para mostrar el contenido dinámico en pantalla completa (modo kiosco) se puede realizar con la mayoría de los navegadores, incluso con el Chromium (el navegador por defecto de Raspian). Si bien es perfectamente posible realizarlo con este navegador, tiene el inconveniente que cuando aparezca nuevo contenido, este se abrirá en una pestaña nueva. Si se tiene que actualizar el contenido muchas veces, a fin de cuentas, tener muchas pestañas abiertas terminará llenando la memoria de la Raspberry. Para evitar este caso, se puede utilizar el navegador Iceweasel, un navegador liviando basado en Firefox. Comenzar descargandolo con el siguiente comando.

```
sudo apt-get install iceweasel
```

Luego realizar la configuración para que el nuevo contenido se muestre siempre en la misma ventana. Escribir en la barra de navegación **about:config**, aceptar el cartel y modificar las siguientes líneas con los valores adecuados.

```
browser.link.open_newwindow.restriction=0 (default 2)
browser.link.open_newwindow = 1 (default 3)
```

A pesar que la configuración realizada parezca trivial, la mayoría de los navegadores no la tiene. Luego, es necesario instalar un plugin para abrir iceweasel en modo kiosco por defecto. Dirigirse al ícono de configuración (esquina superior derecha), clickear en Add-Ons, buscar el plugin "R-Kiosk", aceptar e instalarlo.

Por último, realizar una prueba de la configuración realizada, ejecutando el navegador desde una consola y viendo que se ejecute en pantalla completa. Para ello, invocar una dirección web que contenga una imagen. Ejecutar el siguiente comando.

```
iceweasel www.google.com
```

Habilitar interfaces

Para poder acceder a la Raspberry Pi de manera remota desde otra PC será necesario habilitar el SSH. Para tal fin, desde el menú de Raspian, ir a Preferences -> Raspi Configurations. En la solapa Interfaces seleccionar las opciones correspondientes a SSH y VNC (Virtual Network Communitacion) para visualizar el escritorio remotament.

> *Para esta última opción es necesario tener instalado un cliente VNC en el host donde se quiera visualizar.*

Configurar arranque automático en Raspberry (opcional)

Es posible configurar Raspian para que ejecute de manera automática un comando al iniciar el sistema. Para ello es necesario modificar el archivo /home/pi/.config/lxsession/LXDE-pi/autostart con el siguiente comando.

```
nano /home/pi/.config/lxsession/LXDE-pi/autostart
```

Y agregar el siguiente contenido dentro del archivo (la línea marcada en rojo debe contener el comando a ejecutar).

```
## Original settings
#@lxpanel --profile LXDE-pi
#@pcmanfm --desktop --profile LXDE-pi
#@xscreensaver -no-splash
#@point-rpi
## Put mouse cursor at right inferior corner
xdotool mousemove_relative 10000 10000
## Deactivate screen saver
xset s off
xset -dpms
## Excecute beacon scanner script
@python /home/pi/beacons_scanner/src/beacon_scanner.py
```

Guardar el contenido (CTRL+X y luego Y). Cuando el sistema inicie nuevamente abrirá automáticamente la aplicación. Reiniciar el sistema con el siguiente comando.

```
sudo reboot
```

Instalar paquetes para Bluetooth

Para instalar las dependencias asociadas al módulo Bluetooth ejecutar los siguientes comandos.

```
sudo apt-get install -y libbluetooth-dev libcap2-bin
sudo setcap 'cap_net_raw,cap_net_admin+eip' $(readlink -f $(which python))
```

Ahora es el momento de instalar los paquetes de python que permitirán escanear beacons bluetooth.

> *Debido a que las bibliotecas de Python necesitan acceder al hardware (bluetooth), estas deben ser instaladas globalmente en el sistema y no mediante un entorno virtual.*

```
sudo pip install beacontools[scan]
sudo pip install beacontools
```

Testear lectura de Beacons

Para comprobar que todo se haya instalado correctamente, crear una carpeta llamada beacons_scanner, dentro de ésta una carpeta src y dentro de src un script llamado **test_beacons.py.**

```
mkdir beacons_scanner && cd beacons_scanner/ && mkdir src && cd src/
touch test_beacons.py
```

Agregar al script test_beacons.py el siguiente contenido.

```
import argparse
import time

from beacontools import BeaconScanner
from beacontools import IBeaconFilter

DEFAULT_TIME_TO_SCAN = 10
DEFAULT_BEACON_UUID  = "ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee"

def scan_beacons():
    # callback to show beacons read
    def callback(bt_addr, rssi, packet, additional_info):
        print("[ MAC: {} | RSSI: {} ] - {}".format(bt_addr, rssi, packet))
    # intance the parser
    parser = argparse.ArgumentParser()
    parser.add_argument('-u', '--uuid', action='store', dest='uuid',
                        help='iBeacon UUID. Def: {}'.format(DEFAULT_BEACON_UUID))
    parser.add_argument('-t', '--time', action='store', dest='scan_time', type=float,
                        help='Scan time. Def: {}'.format(DEFAULT_TIME_TO_SCAN))
    # get result of parse arguments in 'r'
    res = parser.parse_args()
    uuid = res.uuid if res.uuid is not None else DEFAULT_BEACON_UUID
    scan_time = res.scan_time if res.scan_time is not None else DEFAULT_TIME_TO_SCAN
    # scan for all iBeacon advertisements from beacons with the specified uuid
    scanner = BeaconScanner(callback, device_filter=IBeaconFilter(uuid=uuid))
    # start scanning
    print ("Starting to scan beacons with UUID={} for {} seconds".format(uuid, scan_time))
    scanner.start()
    time.sleep(scan_time)
    scanner.stop()
    print ("Scan beacons finished!")

if __name__ == "__main__":
    scan_beacons()
```

Con la aplicación de Beacons transmitiendo paquetes iBeacons, testear el script creado previamente. Para realizar las pruebas es necesario que la PC se encuentre cerca (a menos de 10 metros) de la Raspberry para estar dentro del rango de alcance. Ejecutar el script con el siguiente comando.

```
python test_beacons.py uuid=ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee --time=10
```

Si las lecturas de la apliación fueron correctas, en la salida del script debería verse similar a la siguiente.

```
Starting to scan beacons with UUID=ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee for 10 seconds
[MAC: 28:c6:3f:7c:5c:26|RSSI: -58 ] <id: ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee, M: 13, m: 12>
[MAC: 28:c6:3f:7c:5c:26|RSSI: -58 ] <id: ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee, M: 13, m: 12>
[MAC: 28:c6:3f:7c:5c:26|RSSI: -58 ] <id: ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee, M: 13, m: 12>
Scan beacons finished!
```

La información de la lectura debería corresponderse con las configuraciones del iBeacon realizadas en la PC.

Crear la aplicación de CMS en PC/Laptop

El proceso de mostrar contenido dinámico en el navegador de la Raspberry Pi se producirá al invocar una URL del CMS pasando como argumento el/los beacons más cercanos leídos. En función de ello el servidor CMS decidirá qué contenido debe devolver para mostrar. Este tipo de implementación sirve perfectamente para un sistema de kioscos (pantallas colocadas estratégicamente en distintos lugares) que actualicen su contenido de manera centralizada y sin configuraciones en los kioscos. Las configuraciones estarán determinadas por el servidor y el kiosco actuará comoes una “terminal boba”.

> *A esta misma configuración se puede agregar una aplicación para smartphone realizada ad-hoc que en función del beacon más cercano muestre distintas notificaciones al usuario, relevantes al contexto.*

Para esta parte se creará un servidor web, el cual será capaz de interpretar el beacon recibido como argumento en una petición HTTP GET, y dependiendo el beacon leído, devolverá contenido dinámico al cliente. Si bien gestionar contenido para una elevada cantidad de beacons, locaciones o usuarios, requiere de sistemas especializados (Drupal por ejemplo), la arquitectura de sistema planteada en la introducción sirve perfectamente como está, y en caso de ser necesario, se podrá cambiar el CMS por alguno más complejo sin alterar el resto de las partes.

> *Los comandos y pasos a realizar son para PC/Laptops con Linux, aunque se podrían realizar los mismos pasos para otros sistemas operativos utilizando los comandos adecuados.*

Aplicación CMS - Opción 1: PHP con Docker

La página web con el contenido PHP será servida mediante un contenedor de Docker, demostrado en el documento de instalación de herramientas web mediante Docker. Si aún no se tiene instalado Docker, será necesario también leer el material de Introducción a Docker donde se explica detalladamente qué es y cómo instalarlo.

Comenzar descargando la imagen en la PC/Laptop donde correrá el CMS con el siguiente comando.

```
docker pull abassi/php-server:latest
```

Luego crear una carpeta llamada cms-app y dentro de ésta un archivo llamado index.html (que será invocado por la Raspberry) con los siguientes comandos.

```
mkdir cms-app && cd cms-app/ && touch index.html
```

Agregar dentro del archivo index.html el siguiente contenido.

```
<!DOCTYPE html>
 <html>
   <head>
     <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
   </head>
   <body>
       <h1>Bienvenido a la aplicacion de CMS</h1>
       <h2>Implementada mediante imagen de docker abassi/php-server:latest</h2>
       <br>
       <h4>Esta aplicacion interpreta los beacons recibidos como argumentos</h4>
       <h4>En funcion del beacon recibido devuelve diferente contenido HTMLs</h4>
```

```
      <?php
          $major = $_GET["major"];
          $minor = $_GET["minor"];

          if ($major == "111" && $minor == "111"){
              print_r('<body style="background-color:#663366;"></body>');
              print_r("<h1>Mostrando el contenido del beacon major/minor: 111/111</h1>");
          } elseif ($major == "111" && $minor == "222"){
              print_r('<body style="background-color:#006633;"></body>');
              print_r("<h1>Mostrando el contenido del beacon major/minor: 111/222</h1>");
          } elseif ($major == "222" && $minor == "111"){
            print_r('<body style="background-color:#886633;"></body>');
            print_r("<h1>Mostrando el contenido del beacon major/minor: 222/111</h1>");
          } elseif ($major == "222" && $minor == "222"){
            print_r('<body style="background-color:#5566AA;"></body>');
            print_r("<h1>Mostrando el contenido del beacon major/minor: 222/222</h1>");
          } else {
              print_r('<body style="background-color:#990000;"></body>');
              print_r("<h1>Se recibio un beacon desconocido!</h1>");
          }
      ?>
  </body>
  <footer>
          Copyright Agustin Bassi - 2019
  </footer>
</html>
```

Luego crear un script para poder ejecutar el contenedor de PHP de manera sencilla con el siguiente comando.

```
touch serve_php_app.sh && chmod +x serve_php_app.sh
```

Y agregar al script serve_php_app.sh el siguiente contenido:

```
#!/bin/bash

CONTAINER_NAME=php-server
APP_DIR=$1
HOST_PORT=$2

echo "Serving PHP files {container:$CONTAINER_NAME, app-dir:$APP_DIR, port:$HOST_PORT}"

docker run \
--rm \
--volume $APP_DIR:/usr/src/app \
--interactive \
--name $CONTAINER_NAME \
-p $HOST_PORT:8080 \
abassi/php-server:latest
```

A continuación ejecutar el contenedor. Para este ejemplo se servirá el directorio actual (“$PWD”) en el puerto 8080 del localhost. Ejecutar el siguiente comando.

```
./serve_php_app.sh "$PWD" 8080
```

> *Es necesario pasar el full path para que el contenedor lo reconozca, no funcionan los paths relativos.*

Finalmente, chequear que funcione correctamente el servidor al pasar como argumento algunos de los major/minor de los beacons que se esperan recibir desde el código PHP. Probar invocando la URL para el beacon 111/222 http://localhost:8080?major=111&minor=222 y también para el beacon 222/111 http://localhost:8080?major=222&minor=111. Chequear que el contenido cambie en ambos casos.

Para detener la ejecución del contenedor con el servidor PHP ejecutar el siguiente comando o CTRL + C.

```
docker stop php-server
```

Con los pasos realizados se obtiene una implementación sencilla de un servidor PHP mediante un contenedor de Docker que es capaz de mostrar contenido dinámico dependiendo los argumentos recibidos.

Obtener la IP de la PC/Laptop

Para que la aplicación que lee los beacons en la Raspberry Pi se comunique con el CMS, será necesario obtener la dirección de IP del host donde está corriendo. Para ello, ejecutar el siguiente comando.

```
ifconfig
```

En caso de estar conectado a la red local por Ethernet, la IP corresponderá a la interfaz de red **eth0** y en caso de estar conectada por WiFi la IP corresponderá a la interfaz **wlan0** (es posible que el nombre de las interfaces de red no sean exactamente estos nombres). La ejecución del comando anterior arrojará una salida como la siguiente, en la cual se destacan las posibles IP que puede tener la PC.

```
eth0    Link encap:Ethernet  HWaddr a4:4c:c8:50:d5:59
        inet addr:10.22.82.255  Bcast:10.87.95.255  Mask:255.255.224.0
        UP BROADCAST MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
        Interrupt:16 Memory:ef200000-ef220000


wlan0   Link encap:Ethernet  HWaddr 28:c6:3f:7c:5c:22
        inet addr:10.87.82.255  Bcast:10.87.95.255  Mask:255.255.224.0
        inet6 addr: fe80::ff31:15f6:841:c7f4/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:11785426 errors:0 dropped:0 overruns:0 frame:0
        TX packets:7133742 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:10465696617 (10.4 GB)  TX bytes:1640087588 (1.6 GB)
```

Anotar alguna de las direcciones IP de las interfaces de red del host para luego proveerlas a la aplicación en la Raspberry Pi.

Aplicación de lectura de Beacons en Raspberry

La aplicación que correrá en la Raspberry consistirá en un script de Python que scanea periódicamente paquetes iBeacons transmitidos por la PC/Smartphone configurado en la sección Emular Beacon con una PC. Una vez realizada la lectura, le pedirá al servidor CMS (corriendo también en la PC/Laptop) el contenido a mostrar pasando como parámetro el ID del beacon mediante una petición HTTP GET. Finalmente el CMS le devolverá contenido HTML que mostrará en el navegador Iceaweasel configurado en modo kiosco, como se demostró en la sección Configurar el navegador en modo kiosco.

Comenzar creando un archivo llamado beacon_scanner.py en la carpeta src existente, con el siguiente comando.

```
touch beacon_scanner.py
```

Agregar dentro del archivo beacon_scanner.py el siguiente contenido.

```
# Beacon Scanner and Kiosk application for Raspberry Pi
# Author: Agustin Bassi
# Date: August 2019
# Copyright: Agustin Bassi - 2019

# ======= [Imports] ===========================================================

import logging
import time
import subprocess

from beacontools import BeaconScanner
from beacontools import IBeaconFilter

# ======= [APP Settings] ======================================================

APP_TICK       = 3
SCAN_TICK      = 3
BEACONS_FILTER = "ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee"
CMS_IP         = "192.168.0.172"
CMS_PORT       = 8080
CMS_RESOURCE   = "beacons"

# ======= [APP Classes] =======================================================

class Beacon:

    def __init__(self, mac_address, uuid, major, minor, tx_power, rssi):
        """ Class that represents an iBeacon packet """
        self.mac_address = mac_address
        self.uuid = uuid
        self.major = major
        self.minor = minor
        self.tx_power = tx_power
        self.rssi = rssi

    def __str__(self):
        """ Return a Beacon Object as string """
        return "{} - (MAC={}, UUID={}, MAJOR={}, MINOR={}, TXP={}, RSSI={})".format(
            self.__class__.__name__,
            self.mac_address,
            self.uuid[:6],
            self.major,
            self.minor,
            self.tx_power,
            self.rssi
            )
```

```
class BeaconsController:

    def __init__(self, uuid_filter=BEACONS_FILTER, scan_tick=SCAN_TICK):
        """ Init this class that controls the beacons reads """
        self._beacons_list = []
        self._uuid_filter = uuid_filter
        self._scan_tick = scan_tick
        self.__nearest_beacon = Beacon("", "", 0, 0, 0, 0)
        self.__last_nearest_beacon = self.__nearest_beacon

    def _is_beacon_in_list(self, beacon):
        """ Check if a beacons is in the current beacons list """
        for beacon_item in self._beacons_list:
            if beacon_item.major == beacon.major and beacon_item.minor == beacon.minor:
                return True
        return False

    def _order_beacons_list(self, oderType=None):
        """ Order function by some orderType """
        self._beacons_list = sorted(
            self._beacons_list,
            key = lambda beacon : beacon.rssi,
            reverse = True)

    def update(self):
        """ This function scans for iBeacon packets, save them in a list,
        order packets by RSSI and update near beacons attrs accordingly """
        def _scans_callback(bt_addr, rssi, packet, additional_info):
            beacon = Beacon(bt_addr, packet.uuid, packet.major, packet.minor, packet.tx_power, rssi)
            if not self._is_beacon_in_list(beacon):
                self._beacons_list.append(beacon)
        # clear beacon list
        self._beacons_list = []
        # instance the scanner
        scanner = BeaconScanner(
            _scans_callback,
            device_filter = IBeaconFilter(uuid=self._uuid_filter)
        )
        # perform the scan
        scanner.start()
        time.sleep(self._scan_tick)
        scanner.stop()
        # order the beacons list by RSSI (Power received)
        if len(self._beacons_list) >= 1:
            self._order_beacons_list()
            self.__last_nearest_beacon = self.__nearest_beacon
            self.__nearest_beacon = self._beacons_list[0]
            logging.info("Nearest beacon: {}".format(self.__nearest_beacon))
        else:
            self.__last_nearest_beacon = self.__nearest_beacon
            self.__nearest_beacon = None
            logging.warn("No beacons found in this scan")

    def is_nearest_beacon_change(self):
        """ Checks if neares beacon has change recently """
        if  self.__nearest_beacon.major != self.__last_nearest_beacon.major or \
            self.__nearest_beacon.minor != self.__last_nearest_beacon.minor:
            return True
        return False

    @property
    def nearest_beacon(self):
        return self.__nearest_beacon
    @property
    def last_nearest_beacon(self):
        return self.__last_nearest_beacon
    @property
    def beacons_list(self):
        return self._beacons_list

    def __str__(self):
        return "{} ('uuid_filter': '{}', 'scan_tick': '{}')".format(
            self.__class__.__name__, self._uuid_filter, self._scan_tick)
```

```
class WebController:

    def __init__(self, cms_ip=CMS_IP, cms_port=CMS_PORT, cms_resource=CMS_RESOURCE):
        """ Init the web controller. This class controls the content over web browser """
        self.cms_ip = cms_ip
        self.cms_port = cms_port
        self.cms_resource = cms_resource

    def update_content(self, arguments):
        """ Perform a HTTP GET request and send its content to browser """
        url = "http://{}:{}/{}?{}".format(self.cms_ip, self.cms_port,
                                          self.cms_resource, arguments)
        logging.info("Invoking URL: {}".format(url))
        process = subprocess.Popen(["iceweasel", url])

    def update_content_fake(self, arguments):
        """ Logs the content that will be sent to browser """
        url = "http://{}:{}/{}?{}".format(self.cms_ip, str(self.cms_port),
                                          self.cms_resource, arguments)
        logging.info("Invoking URL: {}".format(url))

    def __str__(self):
        return "{} - ('cms_ip':'{}', 'cms_port': '{}', 'cms_resource': '{}')".format(
            self.__class__.__name__, self.cms_ip, self.cms_port, self.cms_resource)



class AppController:

    def __init__(self, beacons_controller=None, web_controller=None, app_tick=APP_TICK):
        """ Init controllers and establish the application tick """
        self.app_tick = app_tick
        self.beacons_controller = beacons_controller
        self.web_controller = web_controller
        logging.info(str(self.beacons_controller))
        logging.info(str(self.web_controller))

    def __format_http_arguments(self, beacon):
        """ Receives a beacon and create string to send to http arguments """
        http_arguments = "major={}&minor={}".format(beacon.major, beacon.minor)
        return http_arguments

    def run(self):
        """ Main loop that runs forever. Read nearest beacon and update web content """
        while (True):
            # read near beacons
            self.beacons_controller.update()
            # check if nearest beacon has changed against last near beacon
            if self.beacons_controller.is_nearest_beacon_change():
                # format argument to invoke URL with parameters
                http_arguments = self.__format_http_arguments(self.beacons_controller.nearest_beacon)
                # update the content in web navegator
                self.web_controller.update_content_fake(http_arguments)
            time.sleep(self.app_tick)

# ======= [Main function] ==========================================================

def main ():
    """ Main function. Instance needed classes and run AppController loop """
    print ("Welcome to Rpi Beacons Kiosk - Powered by Agustin Bassi")
    # configure logging
    logging.basicConfig(
        format='%(asctime)s - %(levelname)s - %(message)s',
        level=logging.DEBUG,
        datefmt='%H:%M:%S'
        )
    # beacons controller instance
    beacons_controller = BeaconsController(
        uuid_filter=BEACONS_FILTER,
        scan_tick=SCAN_TICK
        )
```

```
    # web controller instance
    web_controller = WebController(
        cms_ip=CMS_IP,
        cms_port=CMS_PORT,
        cms_resource=CMS_RESOURCE
        )
    # app controller instance
    app_controller = AppController(
        beacons_controller=beacons_controller,
        web_controller=web_controller,
        app_tick=APP_TICK
        )
    # app controller main loop
    app_controller.run()

if __name__ == '__main__':
   main()

#============================[ End of file ]==================================
```

Modificar los settings de la aplicación dependiendo cada caso (CMS_IP y CMS_PORT de PC/Laptop y BEACONS_FILTER). Finalmente correr el script de python y chequear que todo funcione correctamente con el siguiente comando.

```
python beacon_scanner.py
```

El script debería abrir automáticamente el navegador y mostrar el contenido relacionado al beacon en el navegador, así como también mostrar mensajes en la consola como se muestra a continuación.

```
Welcome to Rpi Beacons Kiosk - Powered by Agustin Bassi
16:11:21 - INFO - BeaconsController ('uuid_filter': 'ffffffff-bbbb-cccc-dddd-eeeeeeeeeeee', 'scan_tick': '3')
16:11:21 - INFO - WebController - ('cms_ip':'192.168.0.172', 'cms_port': '8080', 'cms_resource': 'beacons')
16:11:21 - INFO - Nearest beacon: Beacon - (MAC=33:33:33, UUID=ffffff, MAJOR=11, MINOR=3, TXP=-50, RSSI=-67)
16:11:21 - INFO - Invoking URL: http://192.168.0.172:8080/beacons?major=11&minor=3
16:11:24 - INFO - Nearest beacon: Beacon - (MAC=22:22:22, UUID=ffffff, MAJOR=11, MINOR=2, TXP=-50, RSSI=-31)
```

Conclusiones

En este documento se abarcaron los siguientes temas:

- Se realizó una introducción al problema, se estableció una solución inicial y se diagramó la arquitectura que debe tener el sistema completo.
- Luego se mostraron las acciones necesarias para guardar el sistema operativo Raspian en una tarjeta SD que se debe insertar en una Raspberry Pi.
- A continuación se mostraron las configuraciones iniciales que se deben realizar sobre el sistema operativo Raspian. También se instalaron los paquetes necesarios para escanear beacons Bluetooth, se mostró la configuración opcional para rotar el display, habilitar SSH e instalar el navegador modo kiosco.
- Por otro lado se mostró cómo instalar una aplicación para Linux que emula paquetes de beacons bluetooth (en caso de no contar con beacons) y cómo se debe probar en conjunto con el script de prueba creado para la Raspberry Pi.
- Seguidamente se mostró cómo crear el contenido dinámico asociado a los beacons mediante un sencillo CMS implementado en PHP mediante una imagen de Docker (también se puede instalar PHP en el host).
- Luego se mostró el código de la aplicación en python que escanea los beacons bluetooth, obtiene el contenido de una petición HTTP GET al CMS y muestra el contenido en el navegador.
- Finalmente se mostró la configuración necesaria para arrancar la aplicación de python automáticamente al iniciar el sistema.

Próximos pasos

A partir de este punto hay varios temas que se pueden investigar. Entre ellos pueden ser:

- Pensar en una aplicación real con el uso de beacons.
- Utilizar un CMS más completo (y complejo) que permita establecer diferentes reglas.
- Crear una aplicación para smartphone que tenga la capacidad de leer los beacons cercanos, enviar esas lecturas al mismo CMS que la Raspberry y mostrar al usuario una notificación relevante al contexto, por ejemplo una promoción.
- Crear contenido de calidad y relevante al contexto mediante algun editor especializado para la tarea.
- Robustizar el script que escanea los beacons y muestra el contenido en el navegador en caso de utilizarse en una aplicación real.
- Investigar cómo escanear otro tipo de beacons con script de python corriendo en la Raspberry.

Referencias

- BeaconTools paquete de Python - *[Disponible 02/08/19]*
- Navegador IceWeasel - *[Disponible 02/08/19]*
- Rotar pantalla en Raspian - *[Disponible 02/08/19]*