Installation de serveur sur Raspberry

Cyril Barrelet & Marc Hartley

Janvier 2025

1 Introduction

Dans ce document nous vous apprendrons à installer un serveur sur votre Raspberry Pi. Quel intérêt ? Créer une base de données, créer des pages web, manipuler des composants éléctroniques, etc...

Pour cela, on va utiliser un outil bien souvent utilisé : Node.js.

2 Connexion au Raspberry Pi

Accédez au Raspberry Pi en utilisant la méthode habituelle de d'accès SSH.

- **Identifiant :** pi
- **Mot de passe :** raspberry

Ouvrez un terminal sur votre système Ubuntu et tapez la commande :

```
ssh pi@<adresse_ip_du_raspberry>
```

Une fois connecté, créez un dossier dans le répertoire *home* pour organiser vos fichiers de travail. Utilisez votre nom pour nommer le dossier, ce qui facilitera l'identification et la gestion de vos fichiers. Pour cela, exécutez les commandes suivantes :

```
mkdir ~/<mon_nom>
cd ~/<mon_nom>
```

(Remplacez `<mon_nom>` par votre propre nom).

Ce répertoire contiendra toute la vie de votre nouveau serveur!

3 Installer Node.js

Le package Node.js s'installe comme presque tous les package, avec `apt-get`.

On va d'abord être sûr que le système est à jour :

```
sudo apt update
sudo apt upgrade -y
```

Ajoutez aussi quelques packages Python qui pourraient nous servir pour la suite :

```
pip3 install pyaudio pydub simpleaudio python-vlc --break-system-packages
```

On va maintenant préciser où on peut trouver le package de `nodejs`, puis on l'installe :

```
curl -sL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Bravo, c'est fait, on va pouvoir configurer notre serveur!

4 Créer et lancer le serveur

L'avantage d'utiliser Node.js, c'est qu'il s'occupe de regrouper des modules grâce à son outil `npm` qui peuvent nous être utiles. Par exemple, une base de données.

Commençons par un serveur simple. Depuis le dossier `<votre_nom>`, lancez la commande :

```
npm init -y
```

Après quelques secondes, le serveur est prêt, on va pouvoir le pimper un peu.

Ajoutons les modules de routing (gestion des URL) `express`, celui de gestion de base de données `sqlite3`, et celui de gestion de transfert de fichiers `multer` :

```
npm install express sqlite3 multer
```

Cette fois ça va prendre quelques minutes...

Mais une fois fini, on va créer le fichier `server.js` qui va coder tout ce que l'on veut!

```
nano server.js
```

On va commencer "simple" pour vérifier que tout fonctionne. Dans `server.js`, ajoutez le code suivant :

```
// Le port d'écoute de notre serveur. Si plusieurs personnes utilisent le même Raspberry,
//  ou que vous voulez avoir plusieurs applications qui tournent sur le même Raspberry,
//  modifiez sa valeur.
const PORT = 3000;

// "require" est l'équivalent du "import" de Python
const express = require('express');
const sqlite3 = require('sqlite3').verbose();
const fs = require('fs');
const path = require('path');
const { exec } = require('child_process');
const multer = require('multer');

// Initialize the Express application
const app = express();
app.use(express.json()); // Middleware to parse JSON

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
```

```
});

app.get('/', (req, res) => {
        console.log("Hello world, bienvenue sur le serveur de <votre_nom>! " +
                                "Ce message est visible côté serveur.");
        res.json({
                response: `Voila une réponse JSON envoyée par le serveur vers le client!`,
                information: "Vous pouvez renseiger ce que vous souhaitez ici",
                exemple : {
                        liste_de_nombres : [ 1, 42, 3, -200],
                        autre_liste : ["Test", 1024, {}]
                }
        });
});
```

Lancez le serveur avec la commande :

```
node server.js
```

Sur votre PC local, ouvrez un navigateur et accédez à l'URL `http://<adresse_ip_du_raspberry>:<port_d_ecoute>/`. (Par exemple: `http://192.168.1.12:3000/`).

Si quelque chose s'affiche dans le navigateur, bravo vous pouvez skipper la prochaine section, sinon...

4.1 Rien ne s'affiche...

Supposons que rien ne s'affiche dans votre navigateur. On va chercher le problème :

- La Raspberry elle-est allumée ? Si non, allumez-la et réessayez
- Êtes-vous connectés en SSH ? Si non, reconnectez-vous et relancez le serveur avec :

  ```
  cd <votre\_nom>
  node server.js
  ```

- Y a-t-il des erreurs qui s'affichent dans le terminal lorsque vous lancez `node server.js` ? Si oui, et l'erreur ressemble à `Error: Cannot find module 'expresso'`. Verifiez l'orthographe des `require` (ex: `const express = require('express');`). Arrêtez le serveur (Ctrl+C) et relancez le.
- Même erreur, mais vous êtes sûr de l'orthographe ? Verifiez que les modules soient bien installés avec `npm`. Relancez la commande d'installation, par exemple :

  ```
  npm install express sqlite3 multer
  ```

  Puis relancez le serveur.
- Vous êtes sûr d'avoir installé les packages et vérifié l'orthographe? Connectez-vous en SSH avec un nouveau terminal, et interrogez "à la main" votre serveur :

```
ssh pi@<adresse_ip_du_raspberry>

curl http://127.0.0.1:<port_d_ecoute>
```

Vous recevez l'erreur "Failed to connect to 127.0.0.1 port ¡port_d_ecoute¿: Connection refused" ? Modifiez les paramètre du pare-feu pour utiliser le port ¡port_d_ecoute¿ :

```
sudo ufw allow <port_d_ecoute>/tcp
sudo ufw reload
```

Relancez le serveur.

- Vous avez toujours la même erreur ? Verifiez une seconde, troisième ou millième fois que le port indiqué à la première ligne de `server.js` (`const PORT = ...;`) corresponde à `<port_d_ecoute>` (`http://127.0.0.1:<port_d_ecoute>`). Relancez le serveur.
- Vous pouvez vous connecter du Raspberry vers Raspberry (127.0.0.1) mais pas de votre PC local au Raspberry ? Verifiez encore l'adresse IP de votre Raspberry et celle de l'URL utilisée. Verifiez que vos deux machines sont sur le même sous-réseau (mais si vous avez pu vous connecter en SSH, normalement c'est le cas).
- Pas d'erreur, mais rien ne s'affiche ? Verifiez très fort si votre "fonction" `app.get(...)` finit bien par `res.json(...);`.

4.2 Connecter une base de données

Pour ajouter une base de données, on va utiliser "sqlite3". Dans votre fichier `server.js`, ajoutez n'importe où (en dehors d'une "fonction") :

```
// Connect to the SQLite database
const db = new sqlite3.Database('./my_DB.db', (err) => {
  if (err) {
    return console.error(err.message);
  }
  console.log('Connected to the SQlite database.');
});

// Create a table (if not already existing)
db.run(`CREATE TABLE IF NOT EXISTS dweets (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  thing TEXT,
  content TEXT,
  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
)`);
```

Relancez le serveur.

On voudrait pouvoir la manipuler, cette base de données ! On va alors ajouter un "Endpoint" (une URL accessible pour faire une requête API), et executer un peu de code SQL. Dans le fichier `server.js`, ajoutez les lignes suivantes :

```
// Endpoint to store a dweet
app.post('/dweet/for/:thing', (req, res) => {
  const { thing } = req.params;
  const content = JSON.stringify(req.body); // Convert JSON content to a string to store in DB
  db.run(`INSERT INTO dweets (thing, content) VALUES (?, ?)`, [thing, content], function(err) {
    if (err) {
      return console.error(err.message);
    }
    res.send({ message: `dweets stored for ${thing}`, id: this.lastID, response: "" });
  });
});

// Endpoint to retrieve dweets for a thing
app.get('/get/dweets/for/:thing', (req, res) => {
  const { thing } = req.params;
  db.all(`SELECT * FROM dweets WHERE thing = ? ORDER BY id DESC`, [thing], (err, rows) => {
    if (err) {
      return console.error(err.message);
    }
    res.json({ thing: thing, response: {dweets: rows} });
  });
});
```

Vous reconnaissez le TP de publication de données Dweet.io dans cette BDD ?
Relancez le serveur.
Testez votre serveur depuis un terminal de votre PC (sans se connecter en SSH) :

```
curl -X POST http://<adresse_du_raspberry>:<port_d_ecoute>/dweet/for/un-nom-original \
    -H "Content-Type: application/json" \
    -d '{"temperature": 22, "humidity": 50}'

curl -X GET http://<adresse_du_raspberry>:<port_d_ecoute>/get/dweets/for/un-nom-original
```

Attention : On a définit l'*ajout* d'un dweet avec `app.post`, il faut alors passer par un **POST**, et pour la récupération des dweets, on a définit `app.get` donc on doit passer par un **GET**. Sinon on reçoit la réponse :

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot POST /get/dweets/for/un-nom-original</pre>
</body>
```

```
</html>
```

4.3 Executer du code sur le Raspberry

Vous pouvez continuer à coder tout ce que vous souhaitez dans votre serveur en modifiant le fichier `server.js`.

Vous souhaitez lancer du code Python, ou un executable, ou une commande système ? Vous pouvez vous inspirer de ce endpoint, qui affiche du texte sur l'écran RGB LCD :

```
// Endpoint to display text using Python script
app.post('/display-text', (req, res) => {
    const { R, G, B, text } = req.body;

    // Validate input
    if ([R, G, B].some(color => color < 0 || color > 255) || typeof text !== 'string') {
        return res.status(400).json({
            response: "Invalid input. RGB values must be between 0 and 255" +
                      " and text must be a string."
        });
    }

    // Construct the command to run the Python script
    const command = `python3 write_text.py ${R} ${G} ${B} "${text}"`;

    // Execute the Python script
    exec(command, (error, stdout, stderr) => {
        if (error) {
            console.error("Error executing Python script:", error);
            return res.status(500).json({ response: "Error executing Python script" });
        }
        if (stderr) {
            console.error("Python script stderr:", stderr);
            return res.status(500).json({ response: stderr });
        }

        // Send back whatever the Python script outputs
        res.json({ response: stdout });
    });
});
```

Ici, verifiez que vous avez un script Python `write_text.py` qui peut prendre 4 arguments.

Vous pouvez tester l'API depuis votre machine locale :

```
curl -X POST http://<adresse_ip_du_raspberry>:<port_d_ecoute>/display-text \
 -H "Content-Type: application/json" \
 -d '{"R": 255, "G": 100, "B": 50, "text": "Hello, World!"}'
```

5 Utiliser l'API dans votre jeu

Pour installer directement le serveur sur la Raspberry :

1. Connectez-vous en SSH sur votre Raspberry,
2. Créez un dossier :

```
mkdir <votre_nom> & cd <votre_nom>
```

3. Copiez-y
4. Verifiez que les packages soient installés :

```
curl -sL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt-get install -y nodejs
```

5. Copiez les fichiers disponibles dans ce Drive : `https://drive.google.com/drive/folders/1cu8GII_ZW52mTaCnZSFPyW5zJR-34ccY?usp=sharing`
6. Adaptez la variable "PORT" dans le fichier `server.js` pour qu'il soit unique sur cette Raspberry (entre 3000 et 3100).
7. Autorisez la Raspberry à utiliser ce port :

```
sudo ufw allow 3000/tcp
```

8. Installez les modules :

```
npm install
```

9. Lancez le serveur :

```
node server.js
```

Pour utiliser ce serveur :

1. Copiez le fichier `API_Raspberry.py` depuis le Drive dans le répertoire de votre projet (`https://drive.google.com/drive/folders/1cu8GII_ZW52mTaCnZSFPyW5zJR-34ccY?usp=sharing`)
2. Dans le fichier `API_Raspberry.ps`, modifiez la variable "BASE_URL" pour correspondre à l'IP de votre Raspberry :

```
BASE_URL = "http://<ip_raspberry>:<port>"
```

3. Ajoutez ce fichier à votre projet Python :

```
import API_Raspberry
```

4. Vous pouvez utiliser les fonctions du script pour ajouter et accéder à la base de données et publier des données dans le réseau. Par exemple :

   - `API_Raspberry.dweet_for("my-thing", "temperature": 100, "x": 42, "user": "you")` va ajouter une entrée "my-thing" dans la base de données.
   - `API_Raspberry.get_dweets_for("my-thing")` retourne toutes les entrées dans la base de données.
   - etc... Fouillez un peu le code pour en trouver d'autres et/ou en ajouter vous-même.