

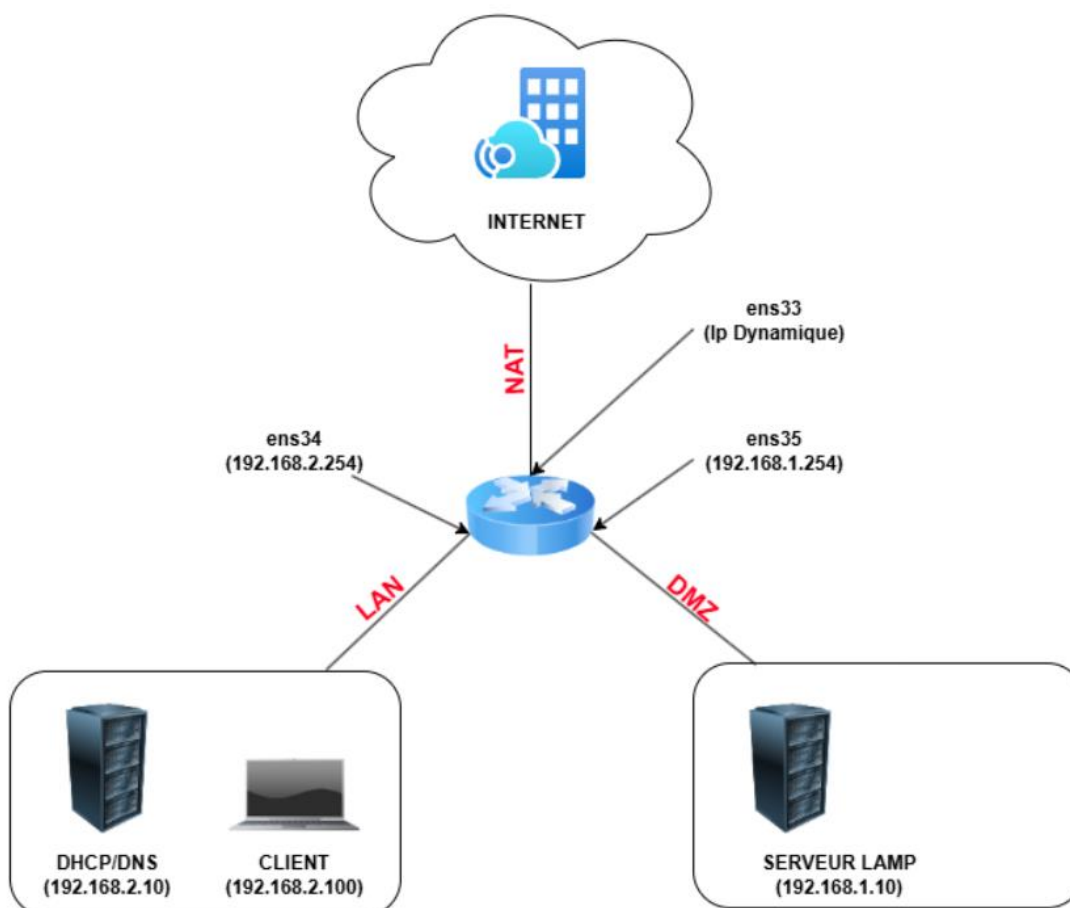
Mise en place d'une infrastructure réseau, d'un site web, d'un serveur web et d'une base de données pour de la collecte de données météorologiques en temps réel pour la Mairie de Bordeaux

Introduction :

Ce document décrit les étapes nécessaires pour mettre en place et exploiter l'infrastructure réseaux pour la collecte de données météorologiques en temps réel pour la mairie de Bordeaux, ainsi que son Site Web et sa Base de données.

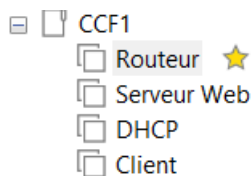
Cette infrastructure est composée d'un réseau de sondes situé à Andernos-les-Bains, d'un serveur web LAMP situé dans une DMZ à Bordeaux, d'un serveur DD (DHCP/DNS) qui assure les services DNS et DHCP, et d'un Client (La Mairie de Bordeaux) qui recevra les données météorologiques en temps réel.

Schéma Réseau :



Installation des Machines Virtualisés :

A l'aide du logiciel "VMWare Workstation Pro" nous allons créer 4 VM comme ceci :



- Pour le routeur
- Pour le Serveur Web
- Pour le DHCP
- Et enfin, pour le client (Mairie de Bordeaux..)

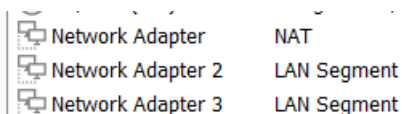
Configuration du Routeur :

Le routeur servira à faire passerelle entre toutes les interfaces réseaux .

Les règles de filtrage et de NAT sont configurées pour contrôler les accès entre les réseaux et assurer la sécurité des ressources internes.

La Mairie voudrait que l'OS soit Debian, alors nous installons celui-ci grâce à un iso, puis nous procéderons à la suite.

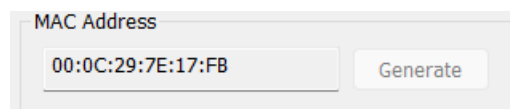
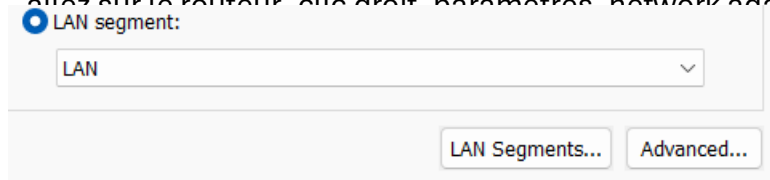
Le routeur aura 3 cartes réseaux :



- Une interface en NAT pour pouvoir se connecter à Internet.
- Une interface en LAN où se trouvent le client et le DD (DHCP/DNS).

- Une interface en DMZ qui sera connectée au Serveur Web LAMP.

Pour savoir quel nom d'interface réseau appartient à quel carte réseau nous allons aller sur le routeur cli droit paramètres network adapter, puis nous allons regarder son adresse MAC :



Donc nous savons que le l'interface réseau LAN du routeur à une adresse MAC se finissant par "FB"

Maintenant pour vérifier quel est le nom de celui-ci, nous allons nous rendre dans la VM, dans le bash puis faire un "ip a"

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:7e:17:f1 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.85.136/24 brd 192.168.85.255 scope global dynamic ens33
        valid_lft 1776sec preferred_lft 1776sec
    inet6 fe80::20c:29ff:fe7e:17f1/64 scope link
        valid_lft forever preferred_lft forever
3: ens34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:7e:17:fb brd ff:ff:ff:ff:ff:ff
    altname enp2s2
4: ens35: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:7e:17:05 brd ff:ff:ff:ff:ff:ff
    altname enp2s3
    inet 192.168.1.254/24 brd 192.168.1.255 scope global ens35
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe7e:1705/64 scope link
        valid_lft forever preferred_lft forever
```

L'ens34 se finit bien par "FB" comme ci-dessous, donc l'interface réseau LAN est bien ens34

L'ens35 est donc l'interface réseau DMZ et l'ens33 est l'interface réseau NAT (cela va nous servir pour plus tard)

```
3: ens34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:7e:17:fb brd ff:ff:ff:ff:ff:ff
    altname enp2s2
    inet 192.168.2.254/24 brd 192.168.2.255 scope global ens34
```

Une fois celà- fait nous allons nous mettre en “root”, donc dans le bash nous allons taper: “su”, puis mettre notre mot de passe.

Ensuite nous allons taper cette ligne de commande :

```
|root@R1:/home/gaudry# nano /etc/network/interfaces
```

Puis rajouter dans celui-ci ces lignes de commande :

```
source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto ens33
iface ens33 inet dhcp

auto ens34
iface ens34 inet static
address 192.168.2.254/24

auto ens35
iface ens35 inet static
address 192.168.1.254/24
```

iface ens33 inet dhcp ---> pour l’ip dynamique

Iface ensX inet static ---> définit une adresse ip fixe

adress 192.168.2.254/24 ----> définit l'adresse IP de l'interface ens34 au DD & Client (LAN)

adress 192.168.1.254/24 ----> définit l'adresse IP de l'interface ens35 au Serveur Web LAMP (DMZ)

Ensuite nous allons faire un “systemctl restart networking” pour pour redémarrer le service en charge avec la configuration réseau, puis taper dans la ligne de commande “ip a” pour afficher la configuration réseau enregistrer :

```
gaudry@R1: ~
inet6 ::1/128 scope host noprefixroute
    valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:7e:17:f1 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.85.136/24 brd 192.168.85.255 scope global dynamic ens33
        valid_lft 1748sec preferred_lft 1748sec
    inet6 fe80::20c:29ff:fe7e:17f1/64 scope link
        valid_lft forever preferred_lft forever
3: ens34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:7e:17:fb brd ff:ff:ff:ff:ff:ff
    altname enp2s2
    inet 192.168.2.254/24 brd 192.168.2.255 scope global ens34
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe7e:17fb/64 scope link
        valid_lft forever preferred_lft forever
```


Les adresses ip ont bien été appliqués.

Pour ce faire nous allons activer l'IP Forwarding, c'est ce qui permet, de transformer une machine en routeur.

Nous allons taper dans la ligne de commande :

```
root@R1:~# nano /etc/sysctl.conf
```

C'est qui nous permet de rentrer dans le fichier de configuration sysctl ensuite nous allons faire apparaitre une ligne de commande présente dans le fichier en enlevant le “#” devant :



```
GNU nano 7.2 /etc/sysctl.conf
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
# Enabling this option disables Stateless Address Autoconfiguration
# based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1

#####

^G Aide      ^O Écrire    ^W Chercher  ^K Couper    ^T Exécuter
^X Quitter   ^L Lire fich ^M Remplacer ^U Coller    ^I Justifier
```

Nous allons maintenant installer le paquet iptables, pour se faire dans la ligne de commande nous allons taper :

```
root@R1:~# apt-get install iptables
```

Puis :

```
root@R1:~# iptables -t nat -A POSTROUTING -o ens33 -j MASQUERADE
```

Et nous allons faire ça pour toutes les interfaces réseaux comme ceci :

LAN <-> Internet

- `sudo iptables -A FORWARD -i ens38 -o ens33 -j ACCEPT`

- `sudo iptables -A FORWARD -i ens33 -o ens38 -m state --state RELATED,ESTABLISHED -j ACCEPT`

DMZ <-> Internet

- `sudo iptables -A FORWARD -i ens37 -o ens33 -j ACCEPT`
- `sudo iptables -A FORWARD -i ens33 -o ens37 -m state --state RELATED,ESTABLISHED -j ACCEPT`

LAN <-> DMZ

- `sudo iptables -A FORWARD -i ens38 -o ens37 -j ACCEPT`
- `sudo iptables -A FORWARD -i ens37 -o ens38 -m state --state RELATED,ESTABLISHED -j ACCEPT`

Nous allons maintenant vérifier si tout est bien appliqué avec un :

```
root@R1:~# iptables -t nat -L
```

```
Chain PREROUTING (policy ACCEPT)
```

```
target      prot opt source                destination
```

```
Chain INPUT (policy ACCEPT)
```

```
target      prot opt source                destination
```

```
Chain OUTPUT (policy ACCEPT)
```

```
target      prot opt source                destination
```

```
Chain POSTROUTING (policy ACCEPT)
```

```
target      prot opt source                destination
```

```
MASQUERADE all -- anywhere             anywhere
```

```
root@R1:~#
```

Tout est bien appliqué, nous allons maintenant sauvegarder les règles de pare-feu avec :

```
root@R1:/home/gaudry# sudo apt install iptables-persistent
```

```
root@R1:/home/gaudry# sudo netfilter-persistent save
```

Pour finir nous allons vérifier si l'ip entre le routeur et internet fonctionne bien :

```
gaudry@R1:~$ ping 8.8.8.8
```

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
```

```
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=14.8 ms
```

```
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=13.5 ms
```

```
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=15.0 ms
```

```
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=13.6 ms
```

```
^C
```

```
--- 8.8.8.8 ping statistics ---
```

```
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
```

```
rtt min/avg/max/mdev = 13.462/14.229/15.029/0.703 ms
```

Le routeur marche bien vers internet.

Configuration du Serveur Web au routeur:

Le serveur Web est comme vu sur le schéma du début, en DMZ du routeur au Serveur Web (LAMP), donc nous allons le paramétrer dans la VM comme ci-dessous avec une seule carte réseau :

Device	Summary
Memory	2 GB
Processors	1
Hard Disk (SCSI)	20 GB
CD/DVD (IDE)	Using file D:\Telechargement...
Network Adapter	LAN Segment
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

LAN segment:

DMZ

LAN Segments... Advanced...

Une fois configuré, nous allons démarrer la VM puis faire un

```
root@R1:/home/gaudry# nano /etc/network/interfaces
```

Comme vu précédemment, et rentrer ces lignes de commande :

```
gaudry@debiangaudry: ~
GNU nano 7.2 /etc/network/interfaces *
# This file describes the network interfaces available on your
# and how to activate them. For more information, see interface

source /etc/network/interfaces.d/*

# The loopback network interface
auto ens33
iface ens33 inet static
    address 192.168.1.10/24    # Une IP libre du LAN
    gateway 192.168.1.254     # IP de ton routeur
    dns-nameservers 8.8.8.8
```

iface ens33 inet static ---> définit une ip fixe

adrees 192.168.1.10/24 ----> définit l'adresse IP de l'interface ens33 à 192.168.1.10 avec un masque de sous-réseau de /24 (vu que 192.168.1.254 est l'ip de l'interface DMZ du routeur, nous allons prendre 192.168.1.X)

gateway 192.168.1.254 -----> adresse ip de l'interface DMZ du routeur.

dns-nameservers 8.8.8.8 ----> adresse IP du serveur DNS public de Google

Nous allons maintenant faire un :

```
root@debiangaudry:/home/gaudry# systemctl restart networking
```

Pour redémarrer le service

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gro
up default qlen 1000
    link/ether 00:0c:29:77:cd:df brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.1.10/24 brd 192.168.1.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe77:cddf/64 scope link
        valid_lft forever preferred_lft forever
```

Tout est bon, nous allons maintenant pinger le routeur pour voir si la connectivité du Serveur Web au routeur est bien établie :

```
root@debiangaudry:~# ping 192.168.1.254
PING 192.168.1.254 (192.168.1.254) 56(84) bytes of data.
64 bytes from 192.168.1.254: icmp_seq=1 ttl=64 time=2.76 ms
64 bytes from 192.168.1.254: icmp_seq=2 ttl=64 time=0.616 ms
64 bytes from 192.168.1.254: icmp_seq=3 ttl=64 time=0.646 ms
64 bytes from 192.168.1.254: icmp_seq=4 ttl=64 time=0.621 ms
^C
```

Et maintenant du routeur au Serveur Web :

```
gaudry@R1:~$ ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=0.578 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=64 time=0.550 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=64 time=0.702 ms
64 bytes from 192.168.1.10: icmp_seq=4 ttl=64 time=0.633 ms
..
```

La connectivité est bien établie entre les deux, maintenant nous allons voir si le routeur ne bloque pas internet vers le serveur Web en piquant les serveurs DNS de Google :


```
root@debiangaudry:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=127 time=13.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=12.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=15.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=127 time=13.1 ms
```

Installation du Serveur LAMP :

Maintenant nous allons installer Apache2, en faisant d'abord un

```
gaudry@debiangaudry:~$ sudo apt update
```

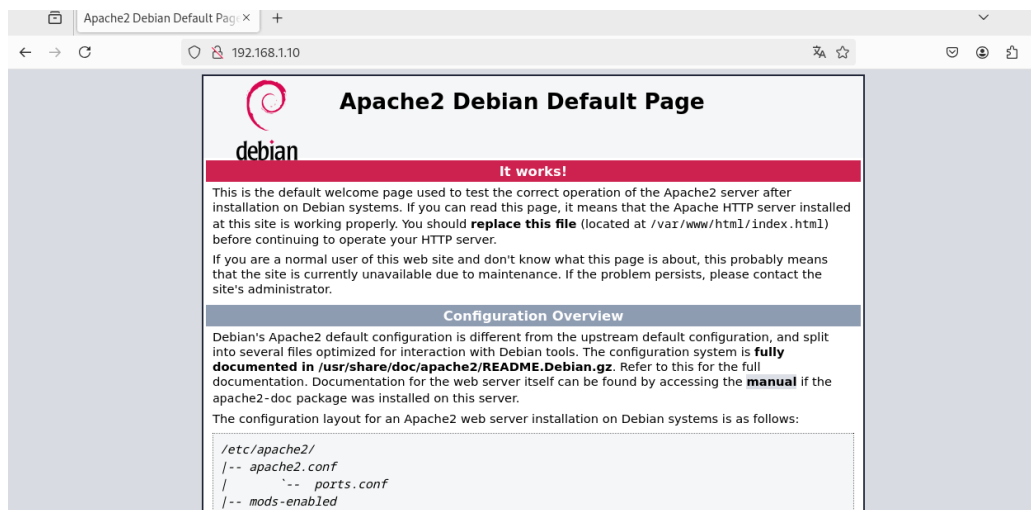
Puis, su pour se mettre en root et

```
root@debiangaudry:/home/gaudry# apt install apache2
```

Et “*apt install ufw*” pour faciliter la configuration et la gestion des règles de pare-feu

```
root@debiangaudry:/etc# apt install ufw
root@debiangaudry:/etc# dpkg -l | grep ufw
ii  ufw                                0.36.2-1
all  program for managing a Netfilter firewall
```

Une fois tout cela installer, nous allons aller sur l'adresse ip du serveur web sur le navigateur pour voir si Apache a bien été installer, donc comme appliquer précédemment nous avons appliquer comme adresse ip statique : “192.168.1.10”, donc nous allons taper cela dans l'url de notre navigateur de la VM :



Apache2 a bien été installer, la page par défaut est bien affichée.

Maintenant nous allons installer MariaDB qui est une dérivée de MySQL, car sur debian12 MySQL n'est plus disponible, mais c'est pareil que MySQL en open source, donc en mieux.

Nous allons l'installer pour pouvoir créer notre base de données, pour que les informations récoltées par l'API Météo à chaque fois que nous rafraichissons notre page web s'affiche aussi bien sur la partie frontend, mais s'enregistrent également dans notre base de données en backend.

Pour installer MariaDB nous allons faire un

```
root@debiangaudry:/home/gaudry# sudo apt update
sudo apt install mariadb-server
```

Puis

```
root@debiangaudry:/home/gaudry# sudo mysql_secure_installation

All done! If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
root@debiangaudry:/home/gaudry#
```

Voilà, l'installation de MariaDB est réussie, nous allons maintenant créer notre base de données, donc d'abord nous allons faire un

```
root@debiangaudry:/home/gaudry# sudo mysql -u root -p
```

Pour rentrer dans MariaDB comme ceci :

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 41
Server version: 10.11.11-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Une fois dedans nous allons pouvoir créer notre base de données, pour ma part je l'ai appelé "Base_Meteo", donc je vais faire un "CREATE DATABASE Base_Meteo" :

```
MariaDB [(none)]> CREATE DATABASE Base_Meteo;
Query OK, 1 row affected (0,000 sec)
```

"Query OK, 1 row affected" nous indique que notre base de données a bien été créée,

donc maintenant nous allons créer une table et des colonnes comme ceci avec la syntaxe suivante :

```
"CREATE TABLE nom_table (  
    colonne1 type_donnees,  
    colonne2 type_donnees,  
    ...  
);"
```

Database changed

```
mariaDB [Base_Meteo]> CREATE TABLE table_meteo (  
    -> id INT AUTO_INCREMENT PRIMARY KEY,  
    -> date DATETIME NOT NULL,  
    -> temperature DECIMAL(5,2),  
    -> humidite DECIMAL(5,2),  
    -> vent_vitesse DECIMAL(5,2),  
    -> vent_direction VARCHAR(10),  
    -> precipitations DECIMAL(5,2)  
    -> );  
Query OK, 0 rows affected (0.007 sec)
```

Notre base de données "Base_Meteo", avec la table "table_meteo" et les colonnes "date, temperature, humidité etc..." ont été créés.

Nous pouvons maintenant installer PHP avec les modules suivants qui vont installer les composants nécessaires pour créer un serveur web PHP avec Apache2 et une base de données MySQL (ou MariaDB) :

```
root@debiangaudry:/home/gaudry# sudo apt install apache2 php libapache2-mod-php php-mysql
```

Nous allons vérifier que ça a bien été installé en regardant sa version :

```
root@debiangaudry:/home/gaudry# php -v  
PHP 8.2.28 (cli) (built: Mar 13 2025 18:21:38) (NTS)  
Copyright (c) The PHP Group  
Zend Engine v4.2.28, Copyright (c) Zend Technologies  
    with Zend OPcache v8.2.28, Copyright (c), by Zend Technologies  
root@debiangaudry:/home/gaudry# █
```

Une fois cela fait nous allons maintenant créer notre répertoire pour le site web dans le serveur Apache2, donc nous allons faire un

```
root@debiangaudry:/home/gaudry# sudo mkdir -p /var/www/SiteMeteo  
sudo chown -R $USER:$USER /var/www/SiteMeteo  
root@debiangaudry:/home/gaudry# █
```

Puis

```
# cd /etc/apache2/sitesavailable# sudo nano /etc/apache2/sites-available/SiteMeteo.conf
```

Une fois dedans nous allons rentrer ces lignes de commande :

```
<VirtualHost *:80>
    ServerName 192.168.1.10
    DocumentRoot /var/www/SiteMeteo
    <Directory /var/www/SiteMeteo>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

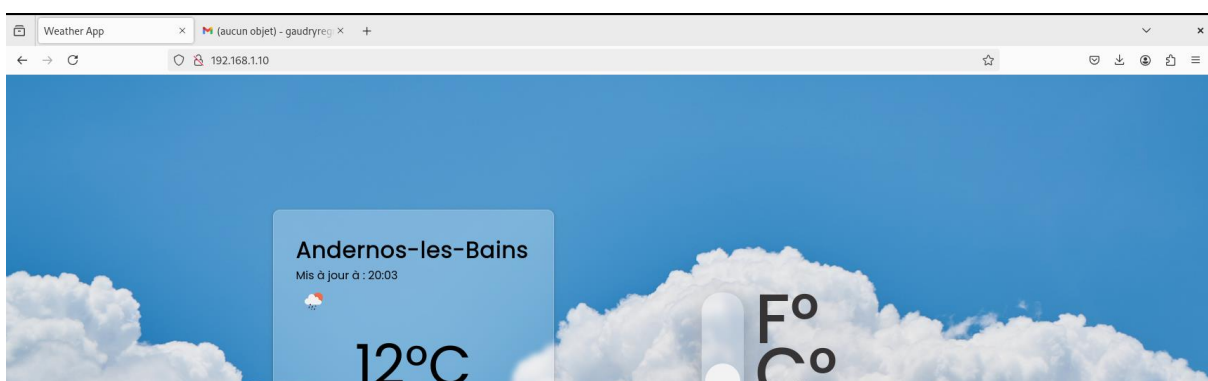
Puis nous allons activer l'hôte virtuel :

```
root@debiangaudry:~# sudo a2ensite SiteMeteo.conf
sudo systemctl reload apache2
Enabling site SiteMeteo.
To activate the new configuration, you need to run:
    systemctl reload apache2
root@debiangaudry:~# █
```

Ensuite je vais rajouter le site web que j'ai codé en HTML/CSS et en Javascript avec l'API OpenWeatherMap dans le dossier /var/www/SiteMeteo que nous avons appliqué au DirectoryX précédemment

```
root@debiangaudry:/var/www/SiteMeteo# sudo nano /var/www/SiteMeteo/style.css
root@debiangaudry:/var/www/SiteMeteo# sudo nano /var/www/SiteMeteo/script.js
root@debiangaudry:/var/www/SiteMeteo# sudo nano /var/www/SiteMeteo/index.html
```

Nous allons retourner sur l'adresse ip de notre serveur (192.168.1.10) dans l'url de notre navigateur voir si cela à pris effet :



Super, le site web et le serveur LAMP sont bien configurés,

Maintenant je voudrais faire en sorte qu'à chaque fois que le client (la Mairie de Bordeaux) rafraichisse la page internet, les données qui s'affichent sur le site web provenant de l'API, se stockent également dans la base de données, pour faciliter la gestion des données météorologiques de l'entreprise.

Pour se faire, je vais passer par un backend comme Node.js.

Il est vrai que j'aurais pu stocker directement les données de l'API directement du site internet à la base de données, mais Node.js va me servir d'intermédiaire pour plusieurs raisons :

- Sécurité : Les clés API, les informations de connexion à la base de données, et autres informations sensibles ne doivent pas être visibles dans le frontend (côté client), car elles peuvent être récupérées par des utilisateurs malveillants.
- Accessibilité à la base de données : Une base de données comme MariaDB ne doit généralement pas être directement accessible depuis le frontend (le navigateur), car elle expose des informations internes et pourrait être vulnérable à des attaques comme l'injection SQL.
- Gestion des requêtes : Parfois, il y a des traitements à effectuer avant d'envoyer les données dans la base de données (par exemple, validation ou nettoyage des données). Cela peut être plus facilement géré sur un backend.

Même si ces données ne sont pas des données sensibles comme des adresse e-mail, adresse, mot de passe, numéro de téléphone etc... Nous privilégions toujours la sécurisation des données, peu importe ce qu'elles contiennent.

Donc tout d'abord je vais installer Node.js en faisant un

```
root@debiangaudry:~# sudo apt install nodejs
```

Une fois installer, nous allons nous rendre dans le dossier de notre site internet, pour ma part “/var/www/SiteMeteo” puis nous allons créer un nouveau fichier Javascript en faisant un “nano server.js”

Dedans nous allons copier notre script , ce fichier sera “le cœur” de Node.js, c’est là où il démarrera le serveur, configurera les routes, se connectera à la base de données, et gèrera les requêtes HTTP.

Donc dedans je vais mettre mon script Javascript comme ceci :

```
const mysql = require('mysql');
const cors = require('cors');
const bodyParser = require('body-parser');

const app = express();

app.use(cors());
app.use(bodyParser.json());

const connection = mysql.createConnection({
  host: '192.168.1.10',
  user: 'gaudry',
  password: '123456',
  database: 'Base_Meteo',
});

connection.connect(err => {
  if (err) {
    console.error('Erreur de connexion à la base de données:', err);
  } else {
    console.log('Connecté à la base de données MariaDB avec succès');
  }
});

app.get('/', (req, res) => {
  res.send('Bienvenue sur le serveur de météo !');
});

app.post('/insert-weather', (req, res) => {
  const { date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations } = req.body;

  const query = 'INSERT INTO table_meteo (date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations) VALUES (?, ?, ?, ?, ?, ?)';

  connection.query(query, [date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations], (err, result) => {
    if (err) {
      console.error('Erreur lors de l\'insertion des données:', err);
    }
  });
});

const connection = mysql
  host: '192.168.1.10',
  user: 'gaudry',
  password: '123456',
  database: 'Base_Meteo'
});
```

Ces lignes de codes vont nous permettre de nous connecter de Node.js à notre base de données à l'utilisateur gaudry, avec le mot de passe "123456" dans la base de données que j'avais précédemment crée "Base_Meteo"

```
    console.error('Erreur de connexion à la base de données:', err);
  } else {
    console.log('Connecté à la base de données MariaDB avec succès');
  }
};

app.get('/', (req, res) => {
  res.send('Bienvenue sur le serveur de météo !');
});

app.post('/insert-weather', (req, res) => {
  const { date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations } = req.body;

  const query = 'INSERT INTO table_meteo (date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations) VALUES (?, ?, ?, ?, ?, ?)';

  connection.query(query, [date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations], (err, result) => {
    if (err) {
      console.error('Erreur lors de l\'insertion des données:', err);
      return res.status(500).send('Erreur lors de l\'insertion des données');
    }
    res.status(200).send('Données insérées avec succès');
  });
});
```

Cette ligne de code est importante

```
app.post('/insert-weather', (req, res) => {  
  const { date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations } = req.body;  
  
  const query = 'INSERT INTO table_meteo (date_meteo, temperature, humidite, vent_vitesse, vent_direction, precipitations) VALUES (?, ?, ?, ?, ?, ?)';
```

Car (/insert-weather) va servir à définir une route , et “INSERT INTO table_meteo...” va nous servir à inclure nos données directement dans notre table de données.

```
const port = process.env.PORT || 8080;  
const host = '192.168.1.10';  
  
app.listen(port, host, () => {  
  console.log(`Le serveur fonctionne sur http://${host}:${port}`);  
});
```

Là, le serveur va écouter uniquement sur l’IP (192.168.1.10) de notre serveur (Serveur Web)

Et il sera disponible que sur le port 80 (port HTTP)

Une fois notre script rentrer, nous allons modifier également notre Javascript de base qui est “script.js”

Nous allons modifier le “updateWeather()”

```

async function updateWeather() {
  try {
    const resp = await fetch(
      `https://api.openweathermap.org/data/2.5/weather?q=${encodeURIComponent(city)}&appid=${apiKey}&units=metric&lang=fr`
    );
    if (!resp.ok) throw new Error('Ville inconnue ou problème réseau...');
    const data = await resp.json();

    const now = new Date();
    const temperature = Math.round(data.main.temp);
    const humidity = data.main.humidity;
    const windSpeed = Math.round(data.wind.speed * 3.6);
    const windDirection = getWindDirection(data.wind.deg);
    const precipitation = data.rain ? data.rain['1h'] : 0;

    cityElem.textContent = data.name;
    updatedAtElem.textContent = 'Mis à jour à : ' + now.getHours().toString().padStart(2, '0') + ':' + now.getMinutes().toString().padStart(2, '0');
    iconElem.src = `https://openweathermap.org/img/wn/${data.weather[0].icon}.png`;
    iconElem.alt = data.weather[0].description;
    tempCElem.textContent = temperature + "°C";
    tempFElem.textContent = ((temperature * 9 / 5) + 32).toFixed(1) + "°F";
    weatherMainElem.textContent = data.weather[0].description.charAt(0).toUpperCase() + data.weather[0].description.slice(1);
    humidityElem.innerHTML = `<i class="fas fa-droplet"></i> ${humidity}%`;
    windElem.innerHTML = `<i class="fas fa-wind"></i> ${windSpeed} km/h ${windDirection}`;

    const weatherData = {
      date_meteo: now.toISOString().split('T')[0],
      temperature: temperature,
      humidite: humidity,
      vent_vitesse: windSpeed,
      vent_direction: windDirection,
      precipitations: precipitation || null
    };

    await sendWeatherDataToServer(weatherData);
  } catch (err) {
    alert("Ville non trouvée ou erreur réseau !\n" + err.message);
  }
}

async function sendWeatherDataToServer(weatherData) {
  const response = await fetch('http://meteogs.fr/insert-weather', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(weatherData)
  });

  if (response.ok) {
    console.log('Données météo envoyées avec succès à la base de données');
  } else {
    console.error('Erreur lors de l\'envoi des données météo');
  }
}

```

Il faut savoir que “*server.js*” et “*script.js*” ne sont pas directement “connectés” comme du HTML et du CSS avec “*link rel="stylesheet href=...*”, mais ils communiquent via le réseau, grâce à HTTP, en gros le “*script.js*” envoie la requête et “*server.js*” répond à cette requête

Ensuite une fois ceci fait je vais me rendre dans le document root “*cd /etc/apache2/sites-available*” et faire un “*nano SiteMeteo.conf*” de mon Site web d’Apache2 et rajouter ces lignes de commande :

```
ProxyRequests Off
ProxyPass /insert-weather http://192.168.1.10:8080/insert-weather
ProxyPassReverse /insert-weather http://192.168.1.10:8080/insert-weather
```

Cela va permettre à Apache de rediriger uniquement les requêtes vers /insert-weather vers le serveur Node.js.

Une fois ceci fait on va activer les modules Apache2 suivant :

```
root@debiangaudry:~# sudo a2enmod proxy
sudo a2enmod proxy_http
```

Puis on va redémarrer les services Apache2 en faisant un

```
root@debiangaudry:~# sudo systemctl restart apache2
```

Maintenant que nos scripts Javascript sont faits, et que node.js et ses requêtes sont bien configuré nous allons lancer Node.js à partir de server.js, qui est la partie centrale de celui-ci.

Donc pour se faire nous allons nous rendre dans le dossier central de notre site web qui est “*/var/www/SiteMeteo*” et une fois dedans nous allons faire un “*node server.js*” comme ceci :

```
root@debiangaudry:/var/www/SiteMeteo# node server.js
Le serveur fonctionne sur http://192.168.1.10:8080
Connecté à la base de données MariaDB avec succès
```

Nous pouvons voir que Node.js fonctionne parfaitement sur “192.168.1.10”, donc de notre site internet sur “80:80: le port 80 (port HTTP)

Pour vérifier si nos requêtes passe bien nous allons nous rendre sur notre site internet, faire “*inspecter l’élément*”, puis sur la partie “*Réseau*” et vérifier que les requêtes du navigateur vers node.js marchent bien.

200	GET	meteoqs.fr	/	document
200	GET	meteoqs.fr	script.js	script
200	GET	cdn.weatherapi.com	116.png	img
200	GET	api.openweathermap.org	weather?q=Andernos-les-Bains&appid=19828690691924002934ae4cb9fd9e5&units=metric&lang=fr	script.js:25 (fetch)
404	GET	meteoqs.fr	favicon.ico	FaviconLoader.sys.mjs:175 (img)
200	POST	meteoqs.fr	insert-weather	script.js:69 (fetch)

Comme nous pouvons le voir tout fonctionne (l’erreur 404 n’est pas importante c’est juste une favicon manquante sur le site web)

200	POST	meteoqs.fr	insert-weather
-----	------	------------	----------------

Le 200 de “*insert-weather*” nous indique que la route Node.js a été traité avec succès, du navigateur à Node.js, mais également de Node.js à la base de données, pour en être sûr nous allons nous rendre dans la base de données pour vérifier que tout à bien fonctionner.

Nous allons donc faire un

```
root@debiangaudry:/var/www/SiteMeteo# mysql -u gaudry -p
```

Puis une fois dedans nous allons sélectionner notre base de données comme ceci

```
root@debiangaudry:/var/www/SiteMeteo# mysql -u gaudry -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 63
Server version: 10.11.11-MariaDB-0+deb12u1 Debian 12
```

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
MariaDB [(none)]> USE Base_Meteo;
```

Puis sélectionner tout ce qu'ils se trouvent dans notre "table_meteo"

Database changed

```
MariaDB [Base_Meteo]> SELECT * FROM table_meteo;
```

```
MariaDB [Base_Meteo]> SELECT * FROM table_meteo;
```

id	date_meteo	temperature	humidite	vent_vitesse	vent_direction	precipitations	heure_meteo
1	2025-04-19	25.50	NULL	NULL	NULL	NULL	22:46:48
2	2025-04-23	14.00	92.00	25.00	0	0.10	22:46:48
3	2025-04-23	14.00	92.00	25.00	0	0.10	22:46:48
4	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
5	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
6	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
7	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
8	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
9	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
10	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
11	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
12	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
13	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48
14	2025-04-23	14.00	93.00	25.00	0	NULL	22:46:48

Voilà, nos données provenant de l'API, passant par le navigateur, puis Node.js qui les renvoie ensuite à MariaDB, sont enfin stocker dans notre base de données (oubli de ma part j'avais oublié l'"heure_meteo" dans mes lignes de code, donc je l'ai rajouté après le screen dans la base de données, le "script.js" et "server.js", donc l'heure est juste un peu faussée par rapport aux autres données saisis, par exemple ci-dessous les données sont bien réels)

268	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:50:07	
269	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:50:44	
270	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:50:51	
271	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:50:51	
272	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:51:51	
273	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:51:51	
274	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:52:51	
275	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:52:51	
276	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:53:51	
277	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:53:52	
278	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:54:51	
279	2025-04-24	13.00	93.00	15.00	N-0	NULL	02:54:51	
280	2025-04-25	19.00	53.00	12.00	E	NULL	12:55:48	
281	2025-04-25	19.00	53.00	12.00	E	NULL	12:55:49	
282	2025-04-25	19.00	53.00	12.00	E	NULL	12:55:50	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								

Voilà, ici l'heure, la date, la température, l'humidité, et le vent sont des données bien réels, par exemple l'heure à laquelle je rafraichis la page :12 heures 55 minutes et 50 secondes précise, montre qu'il fait bien 19° avec 53% d'humidité avec un vent à 12km/h, et que le vent provient de l'EST, à Andernos-Les-Bains.

Maintenant une dernière petite manipulation pour que Node.js fonctionne en continu même après la fermeture du terminal, c'est d'installer PM2 ((Process Manager for Node.js)

Pour se faire on va faire un

```
root@debiangaudry:/var/www/SiteMeteo# sudo npm install -g pm2
```

Puis

```
root@debiangaudry:/var/www/SiteMeteo# pm2 start server.js
```

```
root@debiangaudry:/var/www/SiteMeteo# pm2 start server.js
[PM2] Spawning PM2 daemon with pm2_home=/root/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /var/www/SiteMeteo/server.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	namespace	version	mode	pid	uptime	⌵	status	cpu	mem	user	watching
0	server	default	N/A	fork	4734	0s	0	online	0%	13.1mb	root	disabled

```
root@debiangaudry:/var/www/SiteMeteo# pm2 status
```

id	name	namespace	version	mode	pid	uptime	⌵	status	cpu	mem	user	watching
0	server	default	N/A	fork	4734	7s	0	online	0%	76.5mb	root	disabled

```
root@debiangaudry:/var/www/SiteMeteo# pm2 start server.js
```

Voilà c'est bien installer, si l'on éteins le Serveur Web et qu'on le rallume, il faudra juste faire un "*pm2 start server.js*" dans le dossier "*/var/www/SiteMeteo*", la évidemment on est dans un projet et on utilise des VM, si l'on prenait le cas d'une réelle entreprise, le Serveur Web serait continuellement en train de tourner donc il n'y a pas besoin de faire cela en temps réel.

Maintenant, c'est beaucoup plus pratique pour la Mairie de Bordeaux qui pourra retrouver ses données météorologiques dans leur base de données, et celle-ci est maintenant mieux sécuriser.

Configuration du DNS depuis le Serveur LAMP

Avant de passer à la configuration du DNS dans le côté DD (DHCP/DNS) nous allons devoir faire une petite manipulation pour que nous n'ayons pas à nous connecter via une adresse IP (192.160.1.10) mais via un nom de domaine (www.meteogs.fr) depuis le serveur LAMP, (pour l'instant) , nous verrons dans les parties suivantes comment nous pourrions nous y connecter côté LAN (Client) et plus forcément côté DMZ (Serveur LAMP)

Donc nous allons d'abord nous rendre dans le répertoire "`cd /etc/apache2/sites-available`"

Puis faire un

```
root@debiangaudry:/etc/apache2/sites-available# nano SiteMeteo.conf
```

Ensuite une fois dedans nous allons rentrer ces lignes de commande

```
<VirtualHost *:80>
    ServerName meteogs.fr
    ServerAlias www.meteogs.fr
    DocumentRoot /var/www/SiteMeteo

    ProxyRequests Off
    ProxyPass /insert-weather http://192.168.1.10:8080/insert-weather
    ProxyPassReverse /insert-weather http://192.168.1.10:8080/insert-weather

    <Directory /var/www/SiteMeteo>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Ce qui va nous intéresser ici c'est surtout le ServerName, ServerAlias, et le DocumentRoot :

ServerName meteogs.fr ---> Apache utilisera cette configuration seulement si la requête contient Host: meteogs.fr

ServerAlias www.meteogs.fr ---> le site sera accessible également via <http://meteogs.fr>

DocumentRoot /var/www/SiteMeteo ----> Indique le dossier racine du site web

Voilà, pour l'instant notre site web n'est donc plus accessible via 192.160.1.10, mais pas encore disponible même en tapant dans l'url du navigateur

"`http://www.meteogs.fr`", même depuis le serveur LAMP, il faut faire une autre manipulation pour que cela puisse marcher

Nous allons donc faire un

```
root@debiangaudry:~# sudo nano /etc/hosts
```

Puis rentrer la dernière ligne de commande “192.168.1.10 meteogs.fr
www.meteogs.fr”



```
GNU nano 7.2
127.0.0.1    localhost
127.0.1.1    debiangaudry

# The following lines are desirable for IPv6 capable hosts
::1        localhost ip6-localhost ip6-loopback
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
192.168.1.10  meteogs.fr www.meteogs.fr
```

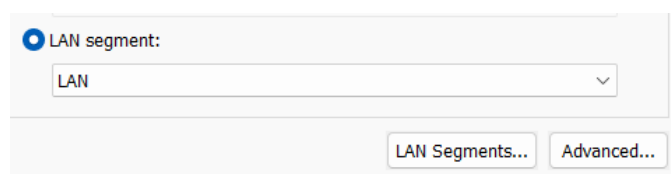
“/etc/hosts” agit comme un mini DNS local pour le Serveur LAMP, le site sera accessible via le nom de domaine mais uniquement via le Serveur LAMP, côté Client et DD (DHCP & DNS) il ne sera pas disponible, nous allons voir comment remédier à cela dans les parties suivantes

Configuration du DHCP/DNS et du Client au routeur :

Le DHCP/DNS et le Client se trouve sur le même réseau LAN (comme sur le schéma du début), donc tout d’abord, nous allons créer deux VM différentes comme ceci :

 DHCP
 Client

Mais dans chacune des VM nous allons leur mettre dans le “Network Adapter” le même réseau nommé “LAN” :



Une fois cela configuré, nous allons également installer Debian12, puis nous allons lancer la VM du “DHCP/DNS”

Dedans nous nous mettrons en root, puis nous allons faire un

```
|root@debiangaudry:/home/gaudry# nano /etc/network/interfaces|
```

Ensuite, une fois dedans nous allons taper ces lignes de commande :

```
# and how to activate them. For more info see the man page
source /etc/network/interfaces.d/*

# The loopback network interface
auto ens33
iface ens33 inet static
    address 192.168.2.10/24    # Une IP fixe
    gateway 192.168.2.254     # IP de
    dns-nameservers 8.8.8.8
```

Iface ens33 inet static ---> adresse ip statique

address 192.168.2.10/24 ----> définit l'adresse IP de l'interface ens33 à 192.168.2.10 avec un masque de sous-réseau de /24 (vu que 192.168.2.254 est l'ip de l'interface LAN du routeur, nous allons prendre 192.168.2.X)

gateway 192.168.2.254 -----> adresse ip de l'interface LAN du routeur.

dns-nameservers 8.8.8.8 -----> adresse IP du serveur DNS public de Google

Une fois ceci fait, nous allons faire un

```
root@debiangaudry:/home/gaudry# systemctl restart networking
```

Pour redémarrer le service, nous allons faire un “ip a” pour voir si ça a bien été appliqué

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:08:79:f0 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.2.10/24 brd 192.168.2.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe08:79f0/64 scope link
        valid_lft forever preferred_lft forever
```

Tout est bon, nous allons maintenant pinger le routeur pour voir si la connectivité du DHCP/DNS au routeur est bien établie :

```
gaudry@debiangaudry:~$ ping 192.168.2.254
PING 192.168.2.254 (192.168.2.254) 56(84) bytes of data.
64 bytes from 192.168.2.254: icmp_seq=1 ttl=64 time=3.20 ms
64 bytes from 192.168.2.254: icmp_seq=2 ttl=64 time=0.722 ms
64 bytes from 192.168.2.254: icmp_seq=3 ttl=64 time=0.497 ms
^C
192.168.2.254: 3% packet loss, 3 ttl=64
```

Et maintenant du routeur au DHCP/DNS :

```
gaudry@R1:~$ ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=64 time=1.38 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=64 time=2.18 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=64 time=0.826 ms
64 bytes from 192.168.2.10: icmp_seq=4 ttl=64 time=1.66 ms
^C
```

La connectivité est bien établie entre les deux, maintenant nous allons voir si le routeur ne bloque pas internet vers le serveur Web en piguant les serveurs DNS de Google :

```
gaudry@debiangaudry:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=127 time=15.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=14.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=13.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=127 time=15.4 ms
```

Le DHCP/DNS est maintenant parfaitement configuré.

Pour la partie Client nous allons suivre les mêmes étapes, nous allons faire un

```
|root@debiangaudry:/home/gaudry# nano /etc/network/interfaces|
```

Puis une fois dedans nous allons taper ces lignes de commande :

```
# The loopback network interface
auto ens33
iface ens33 inet static
    address 192.168.2.100/24 #
    gateway 192.168.2.254
    dns-nameservers 8.8.8.8
```

Iface ens33 inet static ---> adresse ip statique

adresse 192.168.2.100/24 ----> définit l'adresse IP de l'interface ens33 à 192.168.2.100 avec un masque de sous-réseau de /24 (vu que 192.168.2.254 est l'ip de l'interface LAN du routeur, nous allons prendre 192.168.2.X)

(Nous avons cette fois-ci pris 192.168.2.100, car 192.168.1.10 appartient déjà au DHCP/DNS)

gateway 192.168.2.254 -----> adresse ip de l'interface LAN du routeur.

dns-nameservers 8.8.8.8 -----> adresse IP du serveur DNS public de Google

Une fois ceci fait, nous allons faire un

```
root@debiangaudry:/home/gaudry# systemctl restart networking
```

Pour redémarrer le service, puis nous allons faire un “*ip a*” pour voir si ça a bien été appliqué

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grc
up default qlen 1000
    link/ether 00:0c:29:9e:17:bf brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.2.100/24 brd 192.168.2.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe9e:17bf/64 scope link
        valid_lft forever preferred_lft forever
```

Tout est bon, nous allons maintenant pinger le routeur pour voir si la connectivité du Client au routeur est bien établie :

```
gaudry@debiangaudry:~$ ping 192.168.2.254
PING 192.168.2.254 (192.168.2.254) 56(84) bytes of data.
64 bytes from 192.168.2.254: icmp_seq=1 ttl=64 time=2.64 ms
64 bytes from 192.168.2.254: icmp_seq=2 ttl=64 time=0.649 ms
64 bytes from 192.168.2.254: icmp_seq=3 ttl=64 time=0.711 ms
64 bytes from 192.168.2.254: icmp_seq=4 ttl=64 time=0.689 ms
```

Et maintenant du routeur au Client :

```
gaudry@R1:~$ ping 192.168.2.100
PING 192.168.2.100 (192.168.2.100) 56(84) bytes of data.
64 bytes from 192.168.2.100: icmp_seq=1 ttl=64 time=0.552 ms
64 bytes from 192.168.2.100: icmp_seq=2 ttl=64 time=0.647 ms
64 bytes from 192.168.2.100: icmp_seq=3 ttl=64 time=0.617 ms
64 bytes from 192.168.2.100: icmp_seq=4 ttl=64 time=1.62 ms
^C
```

La connectivité est bien établie entre les deux, maintenant nous allons voir si le routeur ne bloque pas internet vers le Client en piguant les serveurs DNS de Google :


```
gaudry@debiangaudry:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=127 time=16.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=13.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=15.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=127 time=15.5 ms
..
```

Le Client est maintenant parfaitement configurer.

Configuration du DHCP

Comme il est dit dans le sujet :

“Le serveur DD fournit un service DHCP permettant d’accueillir des hôtes sur le LAN.”

Cela sert pour que les appareils se configurent automatiquement dès leur connexion.

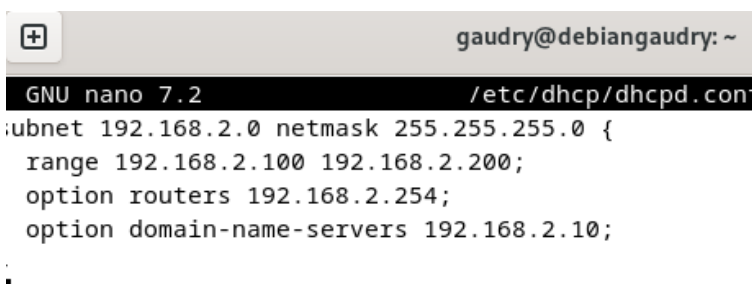
Donc nous allons donc installer ISC-DHCP qui est un serveur DHCP open-source, nous allons donc faire un

```
root@debiangaudry:~# sudo apt update
sudo apt install isc-dhcp-server
```

Puis

```
root@debiangaudry:/etc/dhcp# sudo nano /etc/dhcp/dhcpd.conf
```

Puis nous allons copier ces lignes de commande :



```
gaudry@debiangaudry: ~
GNU nano 7.2 /etc/dhcp/dhcpd.conf
subnet 192.168.2.0 netmask 255.255.255.0 {
    range 192.168.2.100 192.168.2.200;
    option routers 192.168.2.254;
    option domain-name-servers 192.168.2.10;
}
```

Une fois ceci fais, nous allons faire un

```
|root@debiangaudry:/etc# sudo nano /etc/default/isc-dhcp-server
```

Ensuite une fois dedans, nous allons mettre “INTERFACESv4 = ens33”

```
gaudry@debiangaudry: ~  
GNU nano 7.2 /etc/default/isc-dhcp-server  
# Defaults for isc-dhcp-server (sourced by /etc/init.d/isc-dhcp-server)  
  
# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).  
#DHCPDv4_CONF=/etc/dhcp/dhcpd.conf  
#DHCPDv6_CONF=/etc/dhcp/dhcpd6.conf  
  
# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).  
#DHCPDv4_PID=/var/run/dhcpd.pid  
#DHCPDv6_PID=/var/run/dhcpd6.pid  
  
# Additional options to start dhcpd with.  
# Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead  
#OPTIONS=""  
  
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?  
# Separate multiple interfaces with spaces, e.g. "eth0 eth1".  
INTERFACESv4="ens33"  
INTERFACESv6=""
```

Puis nous allons activer et démarrer le service en faisant un

```
root@debiangaudry:/etc# sudo systemctl restart isc-dhcp-server  
sudo systemctl enable isc-dhcp-server
```

Une fois faits-nous allons vérifier qu'il est bien activé

```
root@debiangaudry:/etc# sudo systemctl status isc-dhcp-server # (Debian/Ubuntu)  
● isc-dhcp-server.service - LSB: DHCP server  
   Loaded: loaded (/etc/init.d/isc-dhcp-server; generated)  
   Active: active (running) since Mon 2025-04-21 19:07:19 CEST; 2h 3min ago  
     Docs: man:systemd-sysv-generator(8)  
  Process: 8436 ExecStart=/etc/init.d/isc-dhcp-server start (code=exited, status=0/SUCCESS)  
    Tasks: 1 (limit: 2241)  
   Memory: 5.2M  
      CPU: 56ms  
   CGroup: /system.slice/isc-dhcp-server.service  
           └─8449 /usr/sbin/dhcpd -4 -q -cf /etc/dhcp/dhcpd.conf ens33
```

Voilà, ISC DHCP est maintenant configuré et attribue des IP automatiquement.

Configuration du DNS

Le DNS va nous servir à transformer une adresse ip en un nom de domaine, pour ma part je voudrais le renommer "Meteogs.fr", au lieu de 192.168.1.10

Nous allons donc commencer par installer "BIND9" qui est le serveur DNS le plus utilisé pour gérer la résolution de noms sur les réseaux locaux ou publics.

Nous allons donc l'installer en faisant un

```
root@debiangaudry:~# sudo apt update
sudo apt install bind9 bind9utils bind9-doc dnsutils -y
```

Puis nous allons nous rendre dans “*named.conf.local*” en faisant un

```
root@debiangaudry:/etc/bind# nano named.conf.local
```

Une fois dedans nous allons rentrer ces lignes de commande

```
GNU nano 7.2
zone "meteogs.fr" {
    type master;
    file "/etc/bind/db.meteogs.fr";
};
```

Pour déclarer la zone de DNS locale, maintenant nous allons créer un fichier de zone pour se faire nous allons nous rendre dans “*cd /etc/bind*” puis faire un “*nano db.meteogs.fr*” pour créer un fichier texte dedans

```
root@debiangaudry:/etc/bind# nano db.meteogs.fr
```

Puis mettre ces lignes de commande

```
GNU nano 7.2 db.meteogs.fr
$TTL      604800
@          IN      SOA      ns.meteogs.fr. admin.meteogs.fr. (
                        2      ; Serial - INCRÉMENTE LE !
                        604800 ; Refresh
                        86400  ; Retry
                        2419200; Expire
                        604800 ) ; Negative Cache TTL
;
@          IN      NS       ns.meteogs.fr.
@          IN      A        192.168.1.10
ns         IN      A        192.168.2.10
www        IN      A        192.168.1.10
```

@ IN NS ns.meteogs.fr ---> Nom complet du serveur DNS

@ IN A 192.168.1.10 ---> IP associée au domaine racine (meteogs.fr)

ns IN A 192.168.2.10 ---> IP du serveur DNS lui-même (peut différer de l'IP du site web).

www IN A 192.168.1.10 ---> Même IP que le domaine racine (serveur web partagé).

Puis nous allons configurer le client dans le serveur DHCP pour utiliser le DNS en faisant un

```
root@debiangaudry:/etc/bind# sudo nano /etc/resolv.conf
```

Puis nous allons rajouter "192.168.2.10" après nameserver

```
GNU nano 7.2
# Generated by NetworkManager
nameserver 192.168.2.10
```

Ensuite nous allons faire un

```
root@debiangaudry:/etc# nano hosts
```

Et rentrer ces lignes de commande

```
GNU nano 7.2
127.0.0.1    localhost
127.0.1.1    debiangaudry

# The following lines are desirable for IPv6 capable hosts
::1          localhost ip6-localhost ip6-loopback
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
192.168.1.10 meteogs.fr
```

192.168.1.10 ---> Fait croire à votre PC que meteogs.fr est sur 192.168.1.10

Ensuite dans le serveur DHCP/DNS, nous allons faire la commande

|root@debiangaudry:~# dig meteogs.fr Pour verifier si le DNS fonctionne et quel est l'adresse ip associé au domaine www.meteogs.fr

```
root@debiangaudry:~# dig meteogs.fr

; <<>> DiG 9.18.33-1~deb12u2-Debian <<>> meteogs.fr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24510
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 1232
;; COOKIE: 40e412c248f3b5ec010000006806ad6214e00dff1f047757 (good)
;; QUESTION SECTION:
;meteogs.fr.                IN      A

;; ANSWER SECTION:
meteogs.fr.                 604800  IN      A      192.168.1.10

;; Query time: 0 msec
;; SERVER: 192.168.2.10#53(192.168.2.10) (UDP)
;; WHEN: Mon Apr 21 22:41:06 CEST 2025
;; MSG SIZE rcvd: 83

root@debiangaudry:~#
```

Nous pouvons voir que l'adresse IP 192.168.1.10 (Adresse IP du serveur LAMP) est bien associé au domaine "meteogs.fr"

```
;; ANSWER SECTION:
meteogs.fr.          604800  IN      A       192.168.1.10
```

```
root@debiangaudry:~# sudo nano /etc/resolv.conf
```

Puis rentrer cette ligne de commande

```
GNU nano 7.2
nameserver 192.168.2.10
```

Cela forcera le client à se connecter à un serveur DNS local, en l'occurrence ici au serveur DHCP/DNS

Vérifions en faisant un "dig meteogs.fr" si tout marche bien également côté client

```
root@debiangaudry:~# dig meteogs.fr

<<>> DiG 9.18.33-1~deb12u2-Debian <<>> meteogs.fr
; global options: +cmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41137
; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

; OPT PSEUDOSECTION:
EDNS: version: 0, flags:; udp: 1232
COOKIE: a84b52f57d3f461a010000006806af18d465b0f3f06a2e43 (good)
; QUESTION SECTION:
meteogs.fr.                IN      A

; ANSWER SECTION:
meteogs.fr.                604800  IN      A       192.168.1.10

; Query time: 0 msec
; SERVER: 192.168.2.10#53(192.168.2.10) (UDP)
; WHEN: Mon Apr 21 22:48:24 CEST 2025
; MSG SIZE rcvd: 83
```

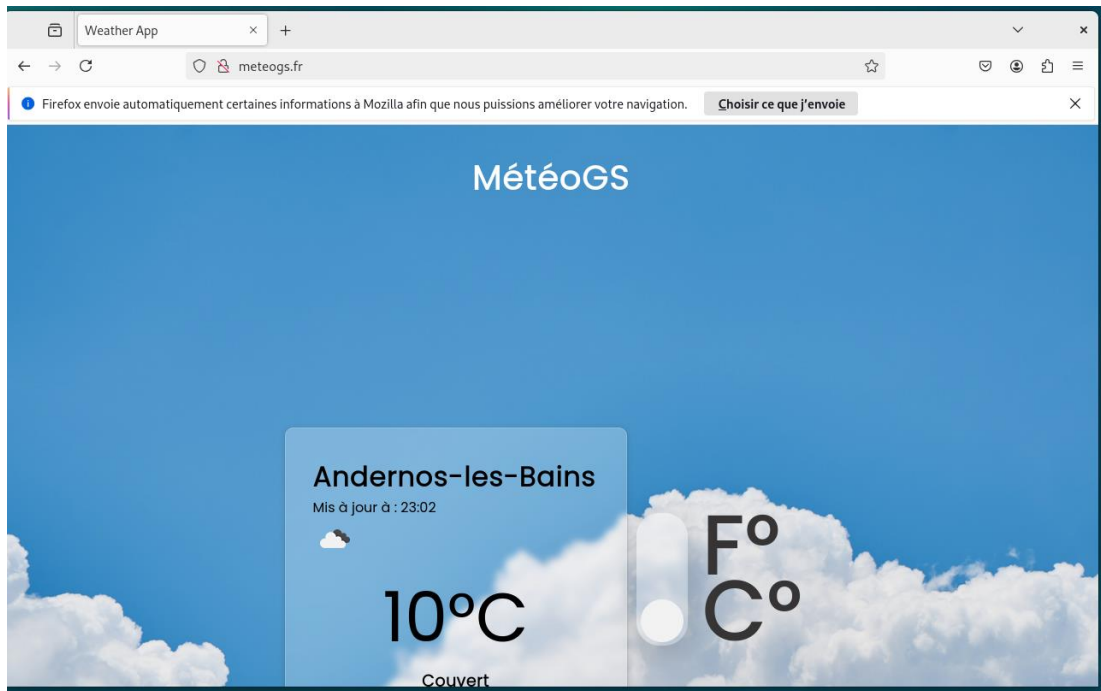
Maintenant, je vais pinger le DNS "meteogs.fr" pour voir si les paquets se transmettent bien et si 192.168.1.10 est bien connecté au nom de domaine, et si le serveur LAN (DD & Client) interfère bien avec le serveur DMZ (Serveur LAMP)

```

root@debiangaudry:~# ping meteogs.fr
PING meteogs.fr (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10 (192.168.1.10): icmp_seq=1 ttl=63 time=4.53 ms
64 bytes from 192.168.1.10 (192.168.1.10): icmp_seq=2 ttl=63 time=1.19 ms
64 bytes from 192.168.1.10 (192.168.1.10): icmp_seq=3 ttl=63 time=1.24 ms
64 bytes from 192.168.1.10 (192.168.1.10): icmp_seq=4 ttl=63 time=1.31 ms
^C

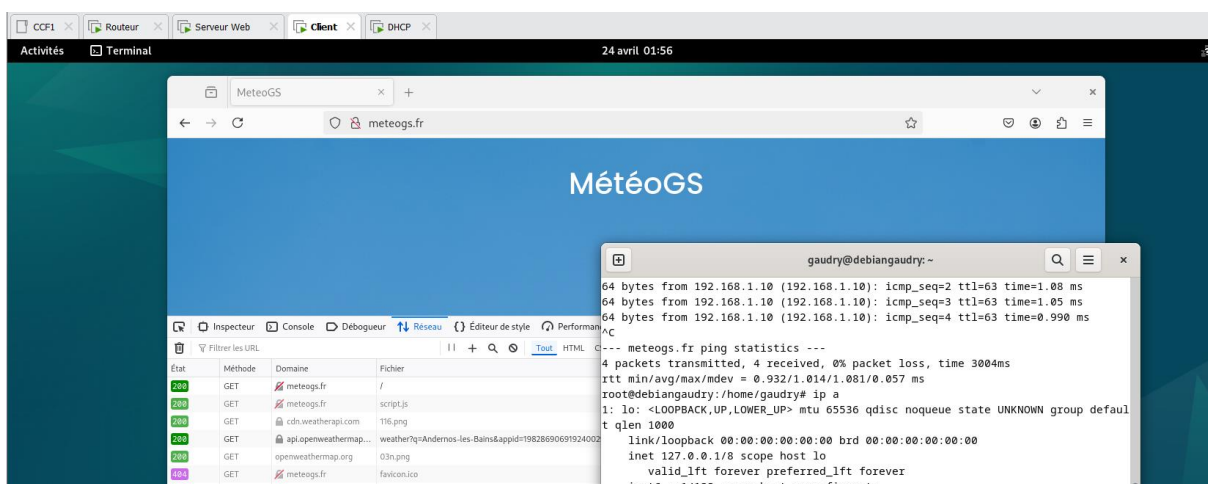
```

Très bien, vérifions maintenant si nous pouvons nous connecter au site web depuis le côté client :



Voilà, le DHCP ainsi que le DNS est bien configuré, et le client a maintenant accès au site web depuis le serveur LAN, côté client

Maintenant une fois que notre Client est bien configuré, normalement nous pouvons nous rendre depuis le Client sur <http://meteogs.fr>, puis rafraîchir la page et celle-ci devrait renvoyer les données provenant de l'API, puis celui-ci vers la base de données MariaDB située sur le Serveur LAMP (DMZ)



200	POST	 meteogs.fr	insert-weather
-----	------	--	----------------

J'ai fait un "ip a" pour vous montrer que je suis bien sur le Client (192.168.2.100) et qu'en actualisant sur le site web, cela renvoie bien la requête 200 vers la Base de données, donc que tout fonctionne parfaitement.

Conclusion

La mise en place de l'infrastructure réseau pour la collecte des données météorologiques en temps réel pour la Mairie de Bordeaux a été réalisée avec succès.

L'ensemble de l'infrastructure comprend plusieurs éléments clés : un réseau de sondes situées à Andernos-les-Bains, un Routeur, un serveur web LAMP hébergé dans une DMZ, un serveur DD (DHCP/DNS) qui gère les services DNS et DHCP, ainsi qu'un Client (la Mairie de Bordeaux) qui reçoit les données météorologiques en temps réel.

Tout au long du projet, les étapes de configuration ont été réalisées de manière détaillée : mise en place du routeur, installation du serveur web LAMP et de sa Base de données, configuration du serveur DD (DHCP/DNS) et du Client.

La configuration réseau, les règles de filtrage et de NAT, ainsi que l'installation des différents services (Apache2, MariaDB, Node.js, ISC-DHCP, BIND9) ont été effectuées avec succès.

Le site web, conçu en HTML/CSS et JavaScript avec l'API OpenWeatherMap, est désormais accessible via le domaine "meteogs.fr" et fournit les données météorologiques en temps réel, ces données sont collectées et stockées dans la base de données MariaDB via Node.js, grâce à cela la Mairie peut conserver un historique des conditions météorologiques, ce qui peut être utile pour diverses analyses à long termes par exemple.

La configuration DNS quant à elle a permis de résoudre correctement le nom de domaine "meteogs.fr" en adresse IP, ce qui garantit un accès fluide au site depuis le Client, les tests réalisés ont confirmé que les données sont bien récupérées et stockées dans la base de données sans problème.

En somme, l'infrastructure mise en place répond parfaitement aux besoins de la Mairie de Bordeaux pour la collecte des données météorologiques en temps réel. Tous les composants de l'infrastructure sont correctement configurés et fonctionnent de manière fiable, garantissant ainsi la disponibilité des données.

Lien GitHub du Projet : <https://github.com/Gaudry33/MeteoGS>