SPRINT 3

Índex

REST API

Què és una REST API?

És un estil d'arquitectura de software que utilitza HTTP per comunicar-se. Les operacions que s'utilitzen en la REST API són: POST (crear), GET (llegir i consultar), PUT (editar) i DELETE (eliminar).

Característiques:

- Protocol client/servidor sense estat: cada petició HTTP conté la informació necessària per realitzar el que es demana, ni el client ni el servidor guarden cap informació.
- Tots els objectes es manipulen amb la URL
- Independència entre el client i el servidor

Què és Node js?

És un entorn de treball del costat del servidor d'una API, de codi obert i multi-plataforma que permet als desenvolupadors crear eines del costat del servidor en el llenguatge de JavaScript i té com a objectiu que un client web l'utilitzi..

Què és express?

És un framework web mas popular de Node. S'utilitza per a l'escriptura de les peticions HTTP en les diferents rutes, ajusta les aplicacions web per ajustar el port on s'escolta o per localitzar les plantilles per donar respostes.

Què és mongoose?

És una biblioteca de javaScript on gràcies a un esquema predefinit, relaciona les col·leccions de MongoDB amb les peticions. Aquest esquema estaran basant en JSON..

Instal·lació d'una REST API amb Node js i express

Instal·lació de Node.js i npm
Instal·larem el framework Node.js amb la comanda 'apt install nodejs'

També instal·larem npm, que ens permetrà gestionar els paquets de Node.js amb la comanda ’apt install npm’

Actualizarem npm a la versió més recent amb ‘npm install npm@latest -g’

Instal·lació de paquets addicionals

Per l’ús de la nostra API necessitarem el paquets: Express, mongoose, method-override, body-parse i http

Instal·larem els paquets amb les següents comandes:

‘npm install express --save’

‘npm install mongoose’

‘npm install method-override’

‘npm install body-parse’

‘npm install’ instal·la totes les dependències en el directori node_modules local

Creació d’un prototip

Crearem una API Rest on podrem usar els mètodes POST, GET, DELETE i PUT per inserir, veure, esborrar i modificar persones en la nostra base de dades de mongo.

Iniciem el projecte

Amb ‘npm init’ crearem el fitxer package.json i ens permetrà iniciar el nostre projecte.

Estructura del projecte

Crearem una estructura de directoris que usarem per diferenciar els models dels objectes (estructura bàsica del què tractarem) i els seus mètodes.

Tindrem un directori /models on es col·locaran les classes personalitzades i un directori /routes on col·locarem un fitxer amb els mètodes de l'API

Fitxers del projecte

Crearem un fitxer anomenat app.js que serà el main de l'API.

Per defecte el fitxer package.json usa un main anomenat index.js i necessitarem modificar el fitxer per poder usar el nostre main.

Obrirem el fitxer package.json amb nano i canviarem la línia "main" : "index.js," per "main" : "app.js, "

Un cop s'ha assignat app.js com a fitxer principal del projecte podem començar a escriure el codi del projecte. El app.js serà el següent:

```
/* API per el projecte de VIA
Autors: grup A2 de PTIN
Any: 2019-2020
Versió: beta, prototip
*/
//------------Carreguem els móduls necesaris
var express = require("express"); //framework web
var bodyParser  = require("body-parser"); // analitza el contingut del cos, per interpretar les dades
var methodOverride = require("method-override"); //paquet que permet realitzar PUT
var mongoose = require('mongoose'); //framework per base de datos MongoDB
var http = require('http'); //framework permet utilizar peticiones http


var app = express(); //Comencem a utilitzar express per utilitzat els métodes de la API a través de les peticions HTTP
var router = express.Router(); //crea un objecte enrutador que habilita l us de GET, POST, PUT, DELETE

//Habilitem les funcions que hem creat anteriormet perquè es puguin utilitzar
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(methodOverride());
app.use(router);

//Indiquem on estaran les routes, métodes bàsics de la API REST
routes=require('./routes/router')(app);

//Intentem fer una connexio la nostre BDD de MongoDB amb l usuari admin
//L estructura de la connexio -> mongodb://user:psswd@addres:port/database?authSource=user
try {
  mongoose.connect('mongodb://admin:admin@192.168.1.6:27017/persona?authSource=admin',{ useNewUrlParser: true, useUnifiedTopology: true });
  const db = mongoose.connection;
  db.on("error", error => console.log(error));
  db.once("open", () => console.log("Connection to the database established"));
}catch(error){
  console.error(error);
}
 // Iniciem el servidor de la API al port 3000
app.listen(3000, function() {
  console.log("Node server running on http://192.168.1.56:3000");
});

```

També tindrem un fitxer person.js al directori /models que contindrà la classe persona. Aquest fitxer tindrà el següent contingut:

```
//Mongoose Schema, per treballar amb mongoose
//Per relacionar cada 'col·lecció' de la base de dades de MongoDB hem de crear un schema per cada.
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;
//Classe persona, conté un nom i un cognom. S'emmagatezma en la col.lecció 'persona' de mongoDB
//Sino s'especifica la col.lecio es creara una col.leccio amb el nom de la classe en plural
var personaSchema = new Schema({
    name: String,
    lastName: String
},{collection:'persona'});

//Exportem el model del schema per poder treballar amb ell en ../routes/router.js
module.exports = mongoose.model('persona', personaSchema);

```

També crearem el fitxer router.js al directori /routes on tindrem els nostres mètodes. Aquest fitxer contindrà el següent:

```
//App routes
module.exports = function(app){

//----------------Variables globals
    //Importem tot el schema de persona que hem creat a ../models/persona per tal de treballar
    // amb mongo db i la seva col·lecció persona
    var persona = require('../models/persona');
//--------------------------------
//---------------Funcions
    //Insertem una persona a la col.leccio
    person = function(req, res){
        var person = new persona({name: req.body.name, lastName: req.body.lastName});
        person.save();
        res.end();
    };

    //Busqueda de totes les persones de la col.leccio
    list = function(req, res){
        persona.find(function(err, people) {
            res.send(people);
        });
    };

    //Busqueda d una persona concreta per el seu _id
    find = (function(req, res) {
        persona.findOne({_id: req.params.id}, function(error, person) {
            res.send(person);
          })
    });

    //Esborra un usuari donant el seu _id
    del = (function(req,res){
        persona.remove({_id: req.params.id}, function(error, person) {
            res.send(person);
          })
    });

    //Modifica els valors especificats d una persona identificada pel seu _id.
    //Els camps no especificats es mantindran igual
    updateALL = (function(req,res){
        persona.updateOne({_id: req.params.id},{$set:req.body},{safe:true}, function(error, upd){
                        res.send(upd);
          })
    });
```

```
    });

//--------------------------------

//Juntem els diferebts métodes de la API i les diferents formes de cridar-los amb les funcions corresponents
//Tots els següents métodes és cridaran quan l'usuari introdueixi en la URL http://adreça:port/person i si fa falta /num_id (al final) segons el que es vulgui fer
//---------------------------------POST---------------
//inserim dades a la BDD
app.post('/person', person);
//---------------------------------GET---------------
//Obtenim dades de la BDD
app.get('/person', list);
app.get('/person/:id', find);
//---------------------------------DELETE---------------
//esborrem dades de la BDD
app.delete('/person/:id', del);
//---------------------------------PUT---------------
//modifiquem dades de la BDD
app.put('/person/:id',updateALL);

};
```

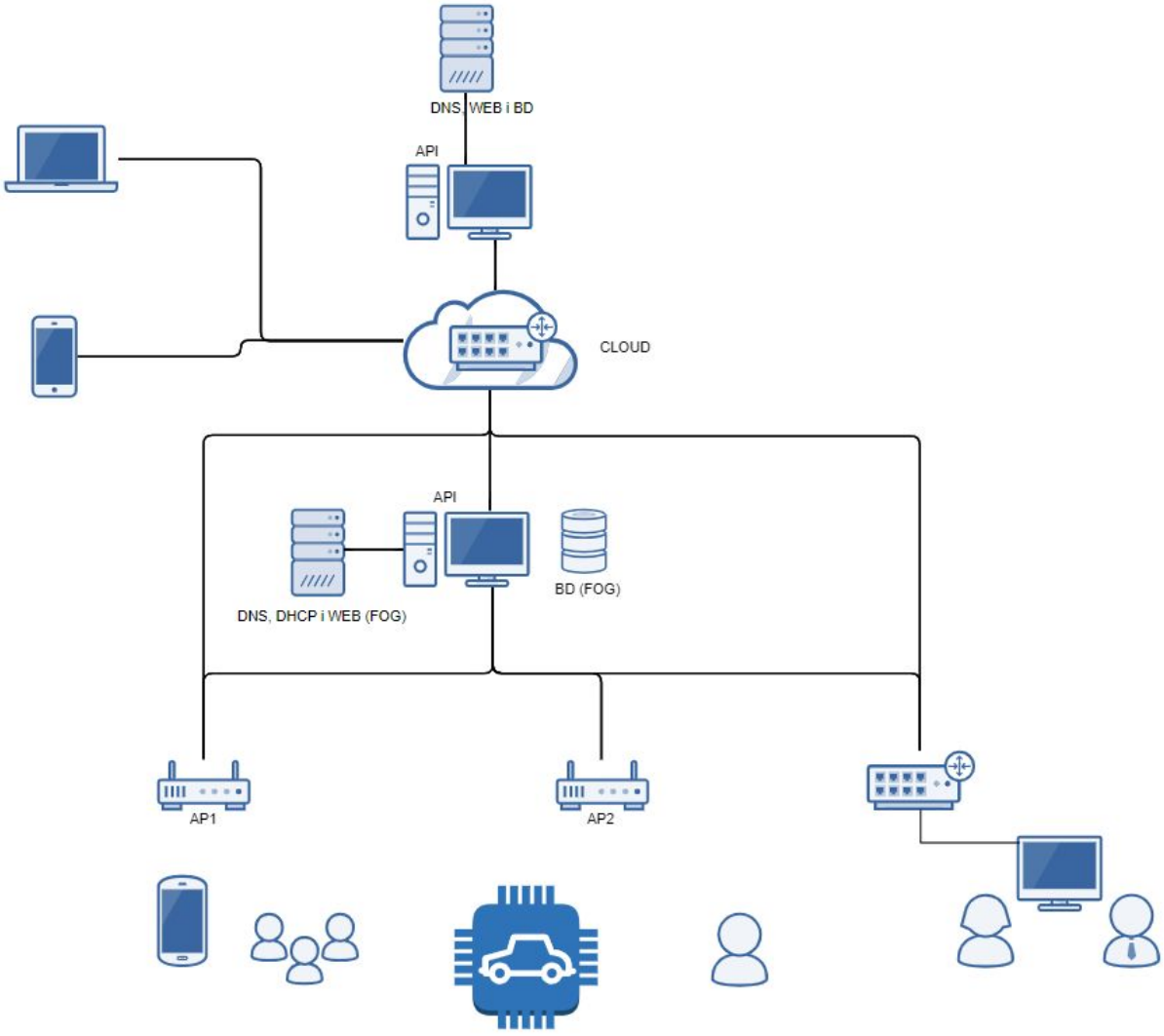
Executar:

node app.js

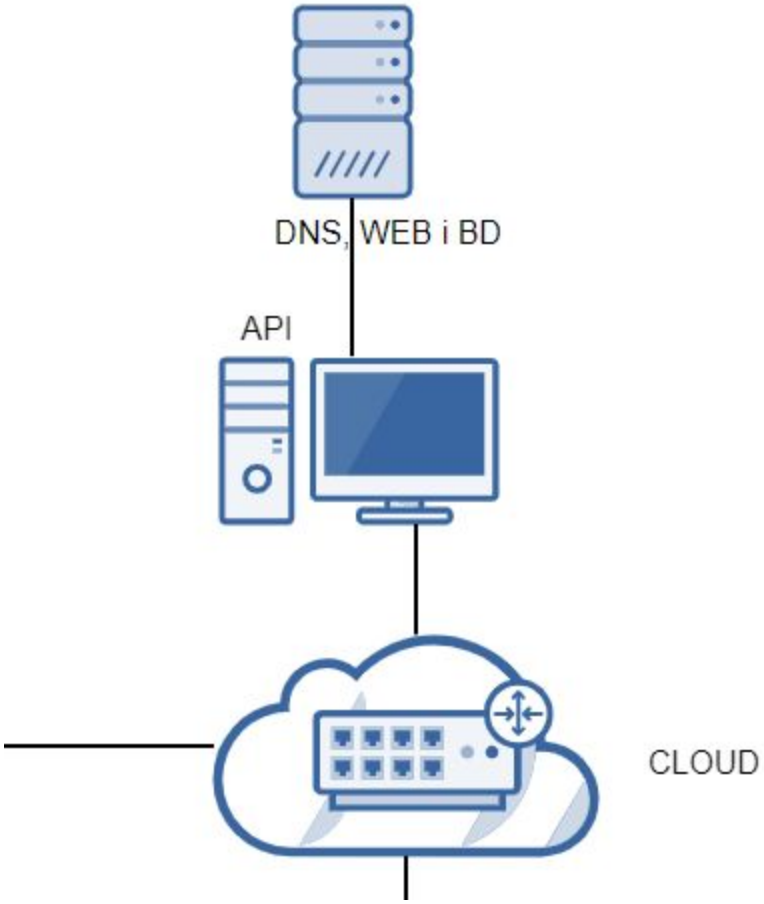
```
root@seax:/myapp# node app.js
Node server running on http://192.168.1.56:3000
Connection to the database established
```

Esquema de comunicació

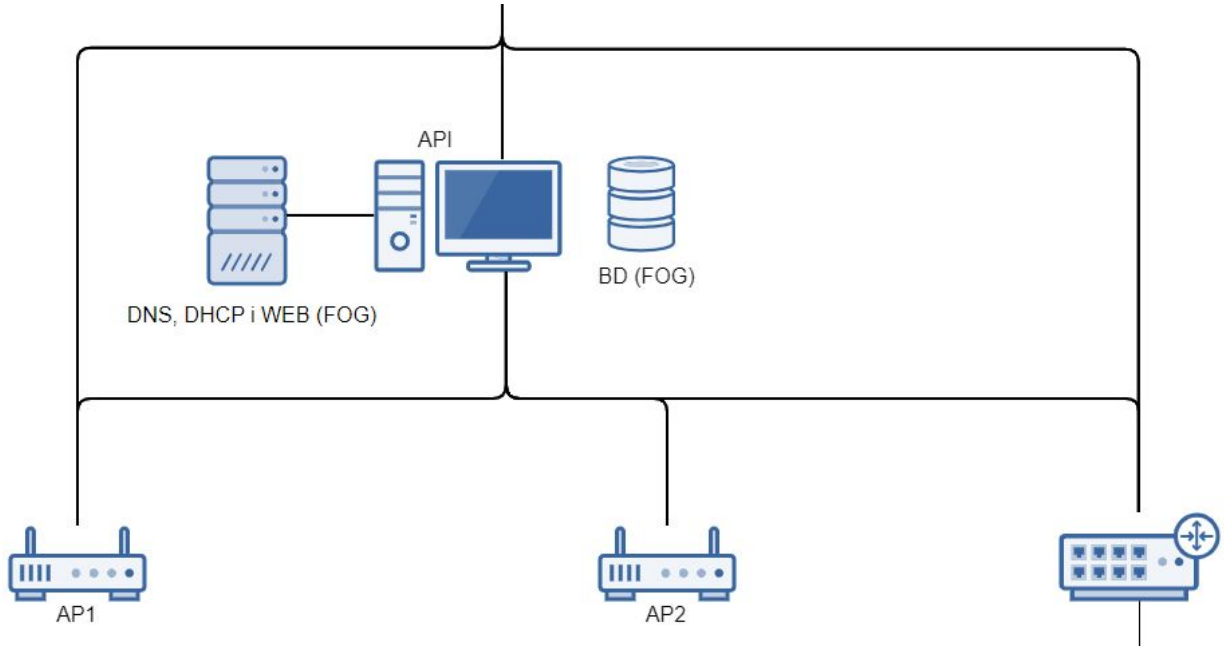
Aquest esquema de comunicació està basat en el F2C que proposem:



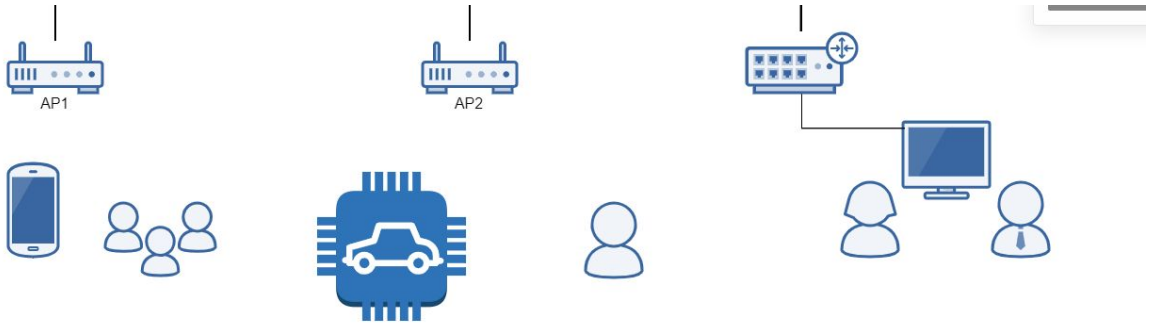
Comunicació al cloud



Comunicació al Fog



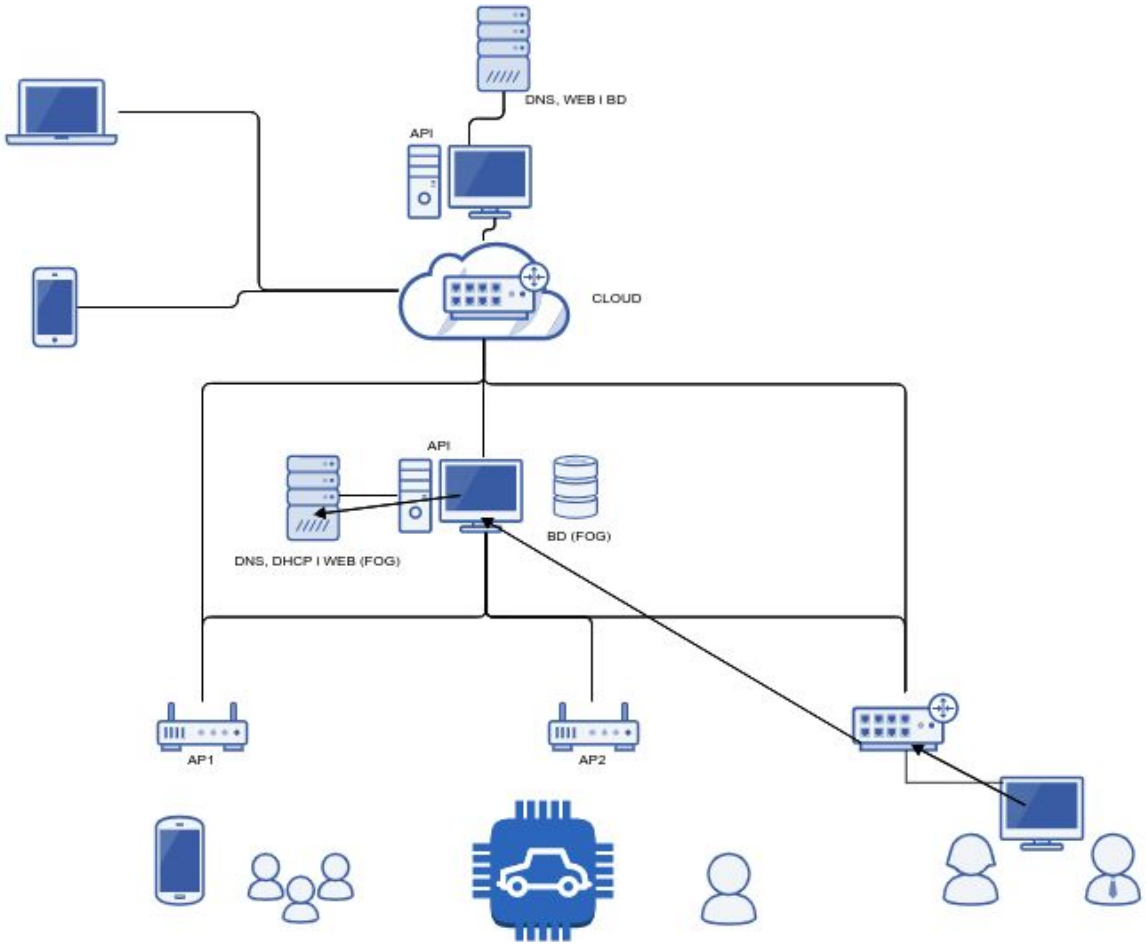
Comunicació del edge



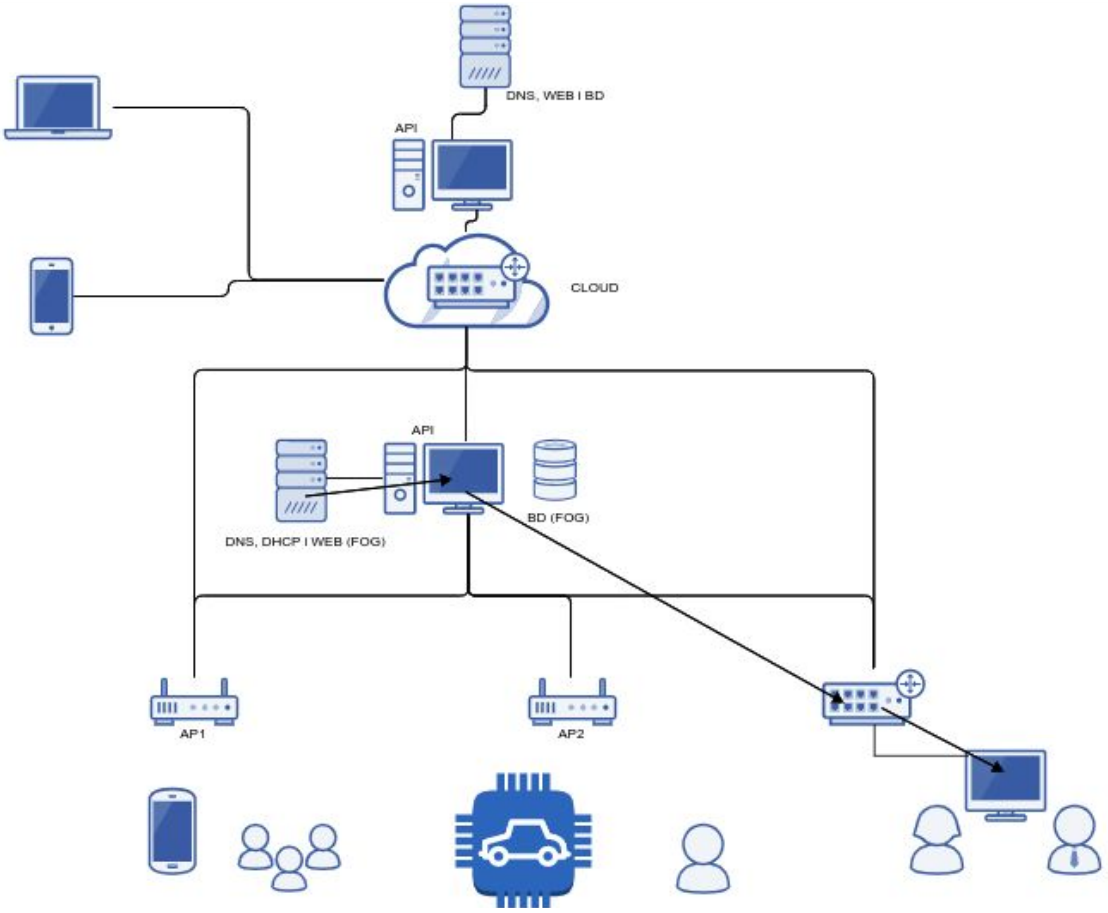
Exemples de comunicació:

Comunicacions del servidor web amb la resta d'elements:

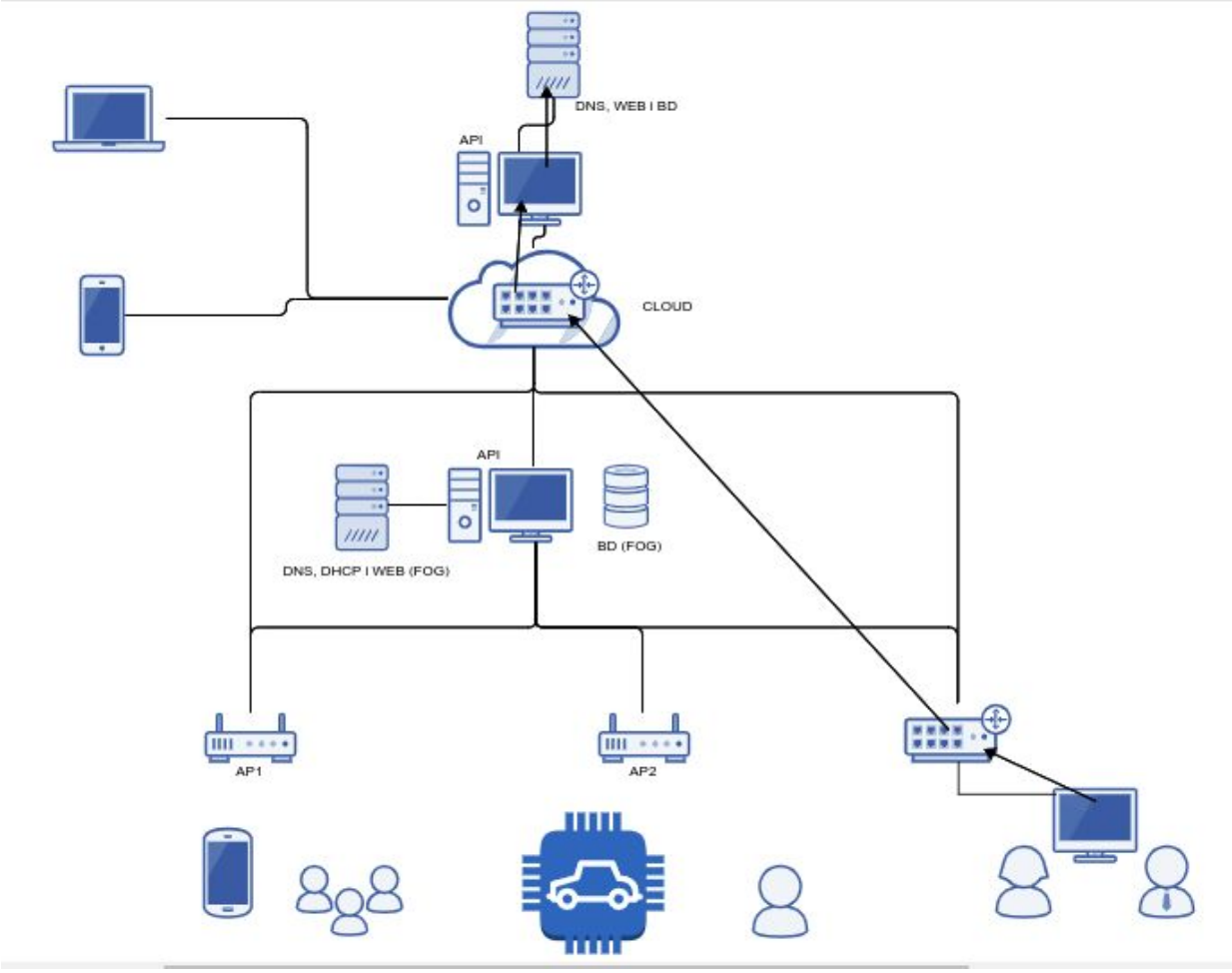
Per parlar de les comunicacions que fa el servidor web amb la resta d'elements dintre de l'arquitectura, comencem explicant que la sèrie de comunicacions que desencadena el fet que els gestors, connectats a la xarxa mitjançant un router, facin una petició mitjançant el servidor del fog, que conté l'API, al servidor web. Els gestors fan la seva petició al servidor del fog, que conté l'API, i aquesta fa la consulta al servidor web per obtenir la resposta d'aquesta petició, el següent esquema mostra aquesta comunicació:



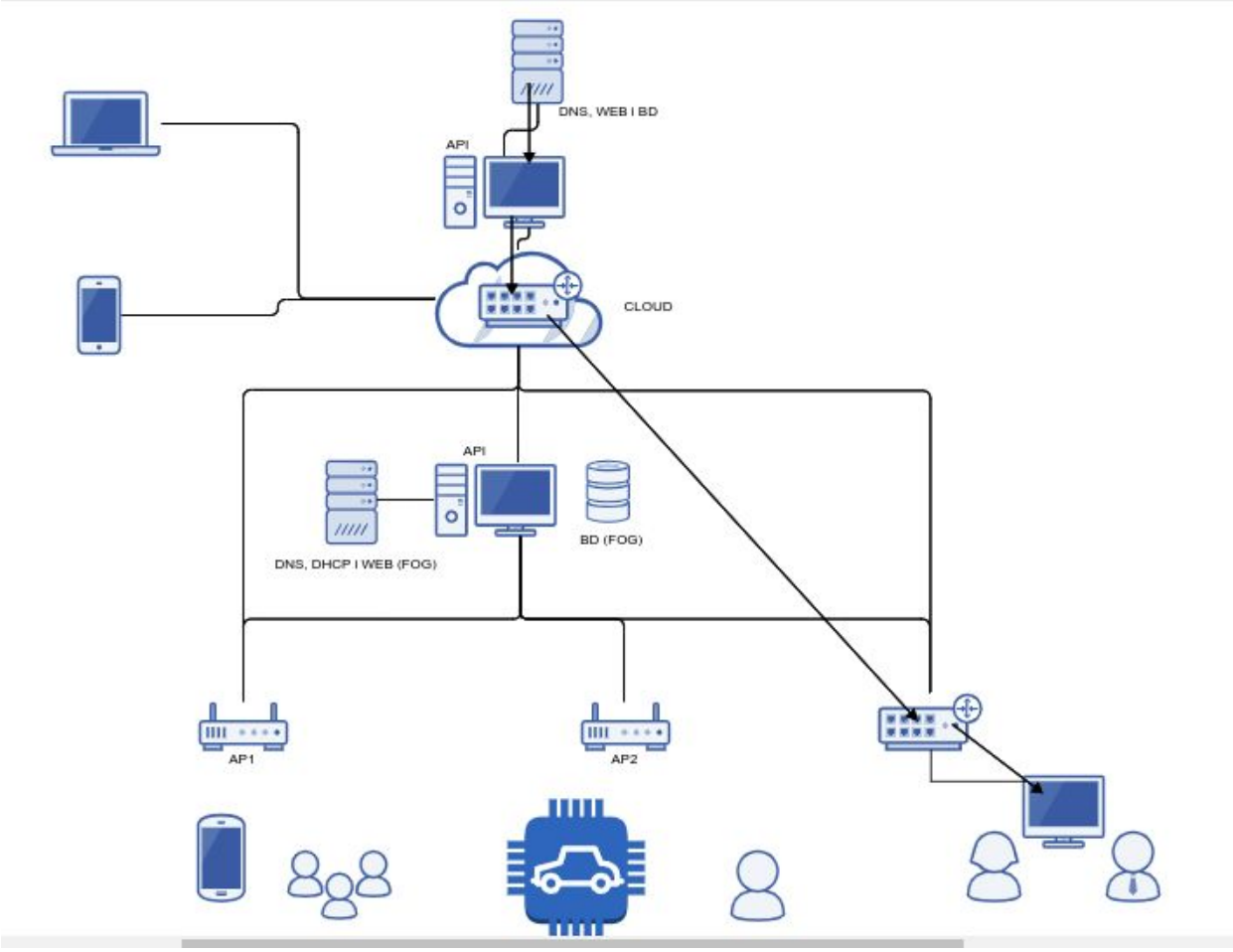
Una vegada l'API del fog obté la resposta a la petició dels gestors del servidor web llavors fa arribar aquesta resposta als gestors, al següent esquema observem aquesta comunicació:



Es pot donar el cas que per satisfer la resposta dels gestors a l'API del fog li faci falta accedir al servidor web del cloud també, llavors l'API de la capa fog faria arribar mitjançant el servidor del cloud, que conté l'API, al servidor web del cloud la petició dels gestors. Aquesta comunicació l'observem en el següent esquema:



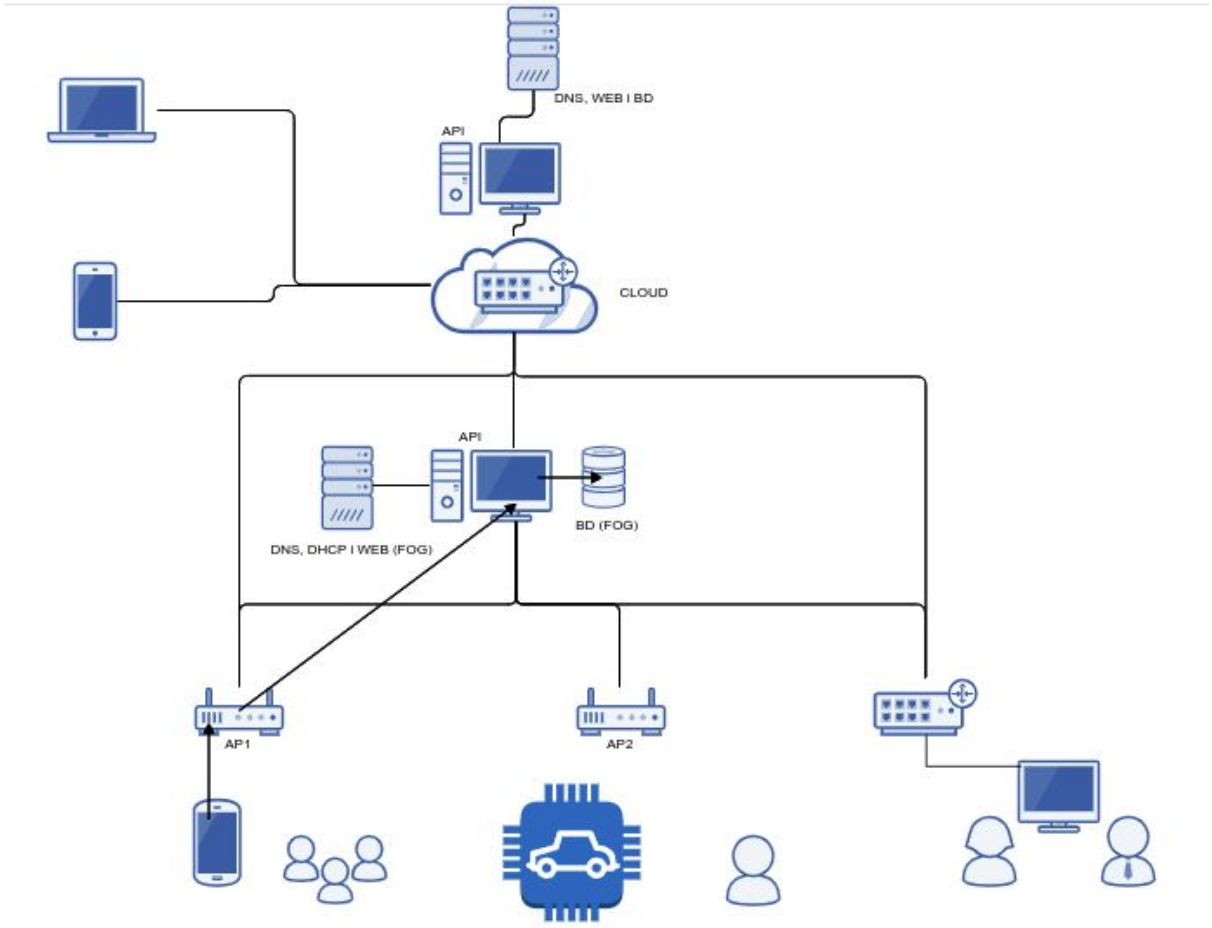
Després el servidor del cloud, que conté l'API, li faria arribar als gestors la resposta a la seva petició, una vegada ha obtingut aquesta resposta del servidor web del cloud. Podem observar aquesta comunicació en el següent esquema:



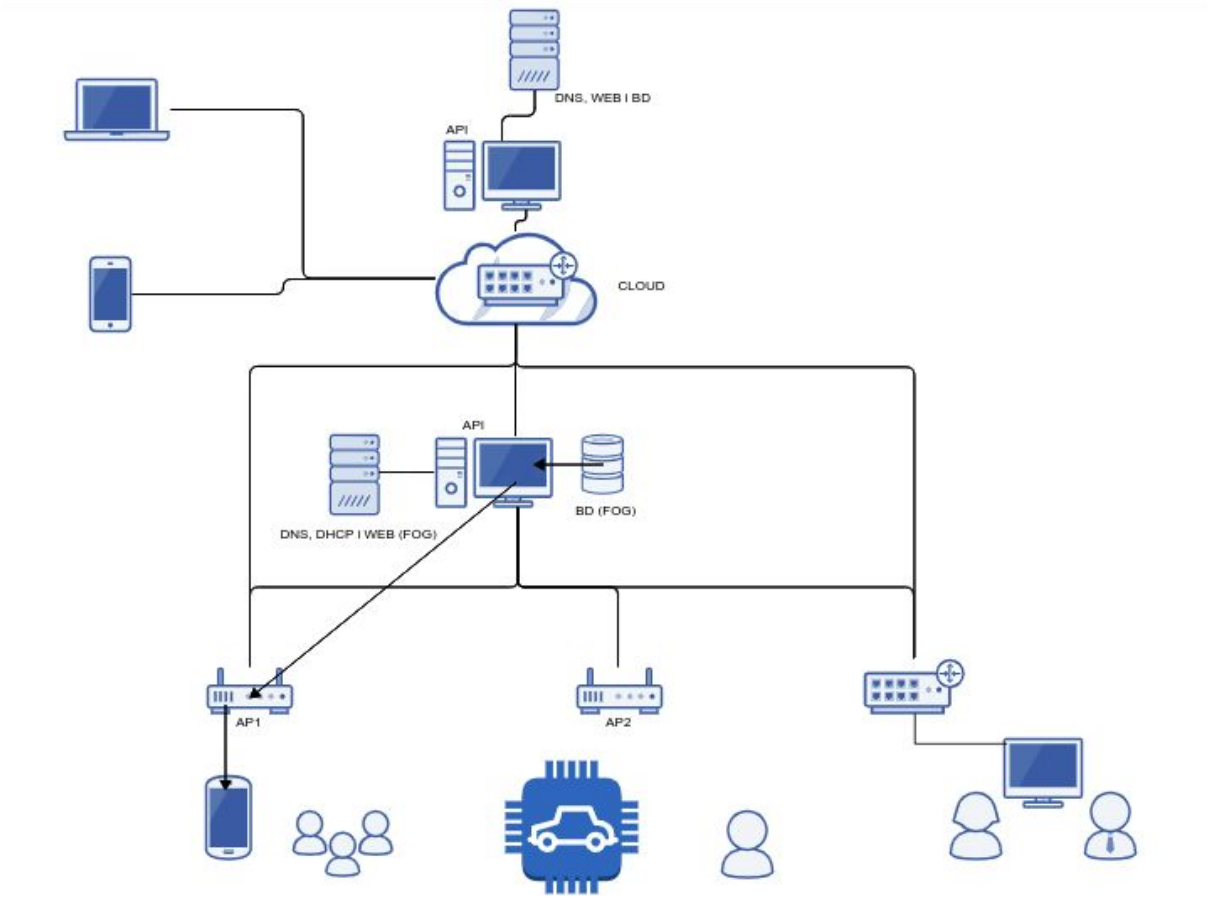
Podem observar que per fer aquesta comunicació la informació passa per més nodes per tant requereix un temps elevat que quan la petició ho fa el servidor del fog, per aquest motiu aquesta comunicació no es farà tan habitualment i el farà el Servidor web del fog.

Comunicacions de l'app mòbil amb la resta d'elements:

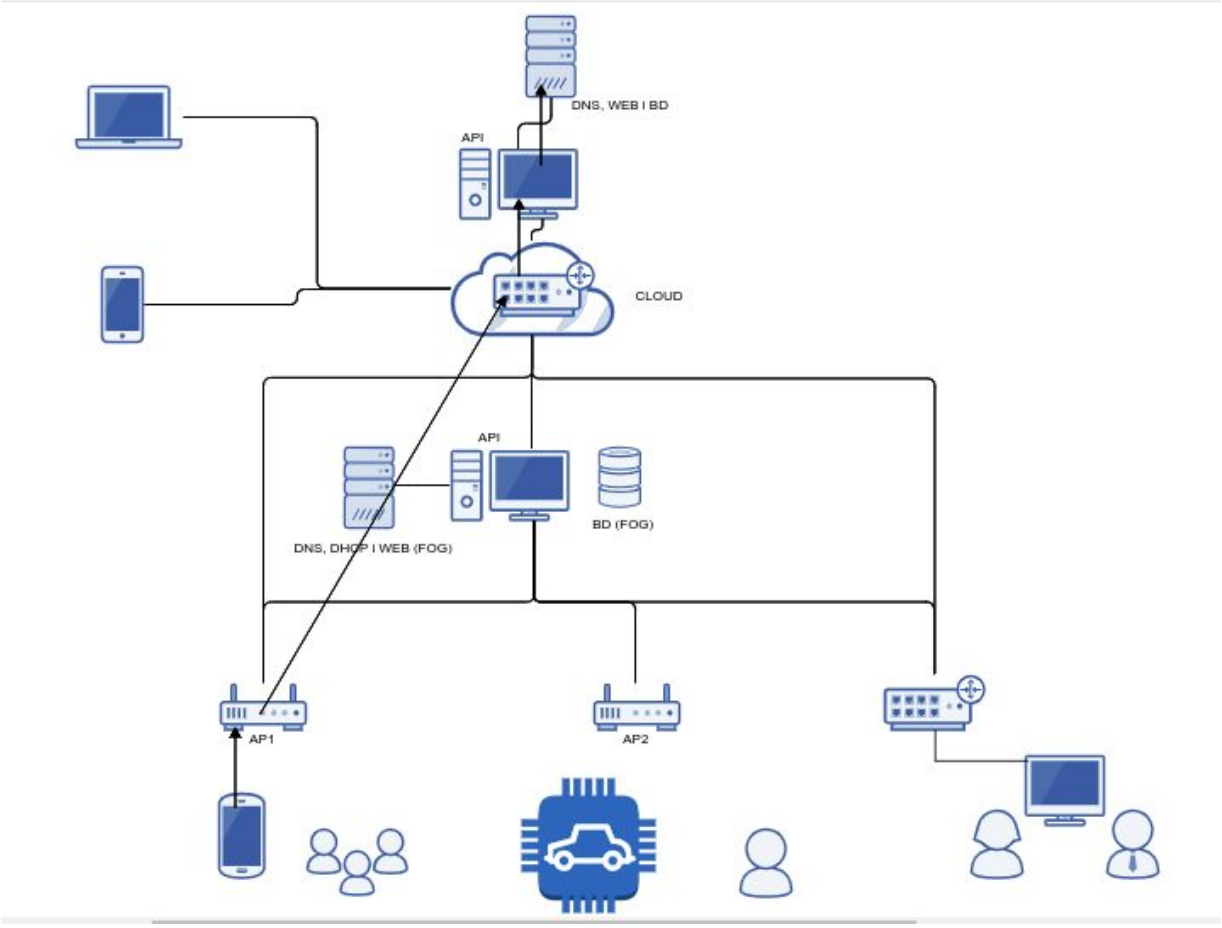
Per parlar de les comunicacions que farà l'app mòbil amb resta d'elements hem de pensar en tots els casos que es poden donar, comencem amb el cas en el que l'app mòbil fa una petició a l'API del fog. El següent esquema mostra aquesta comunicació:



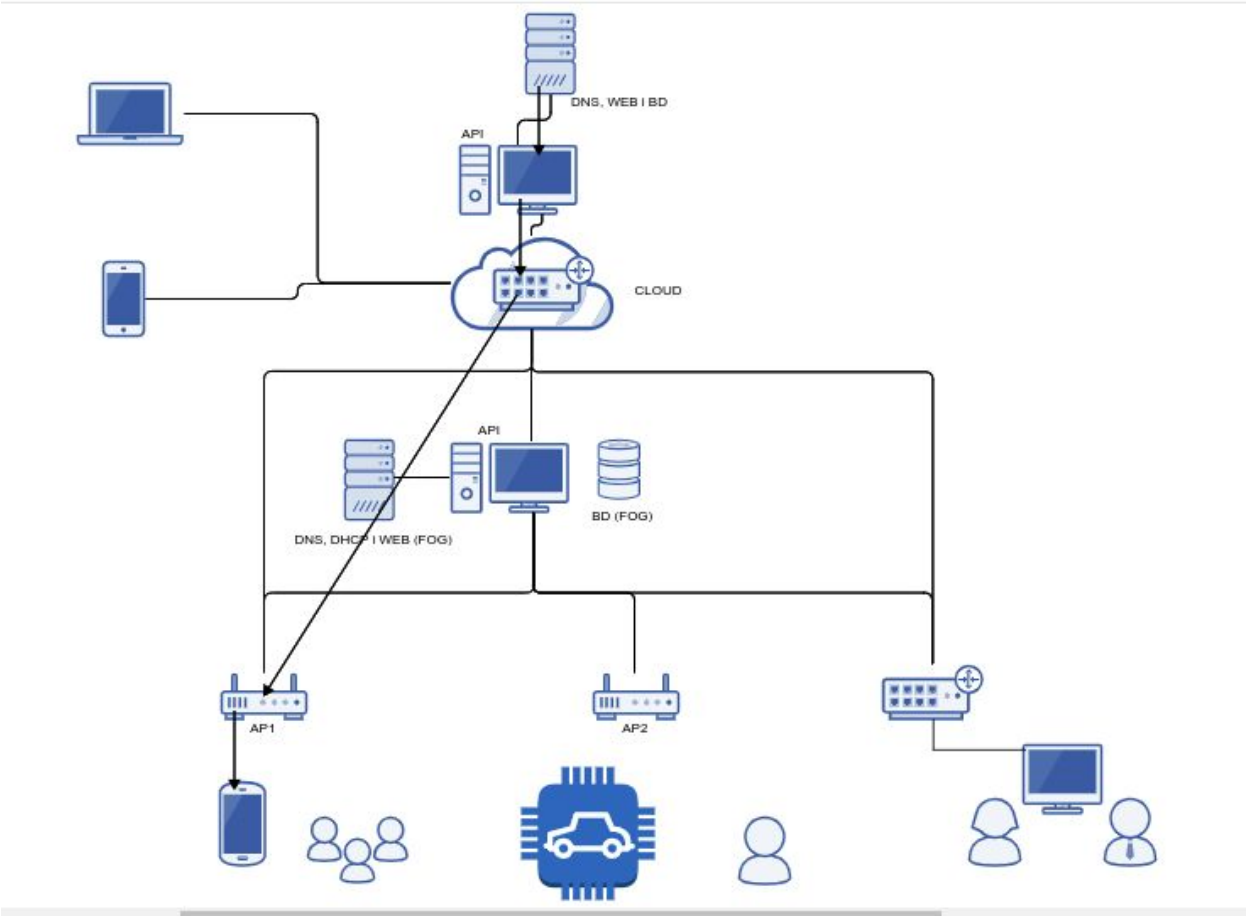
Llavors es pot donar dos casos, el primer cas que l'API pugui resoldre la petició de l'app mòbil accedint a la base de dades del fog, aquesta comunicació ho podem observar en el següent esquema:



El segon cas és que l'API no pugui resoldre la petició de l'app mòbil accedint a la base de dades del fog llavors hauria de fer arribar la petició de l'app mòbil a la base de dades del cloud, aquesta comunicació ho podem observar en el següent esquema:



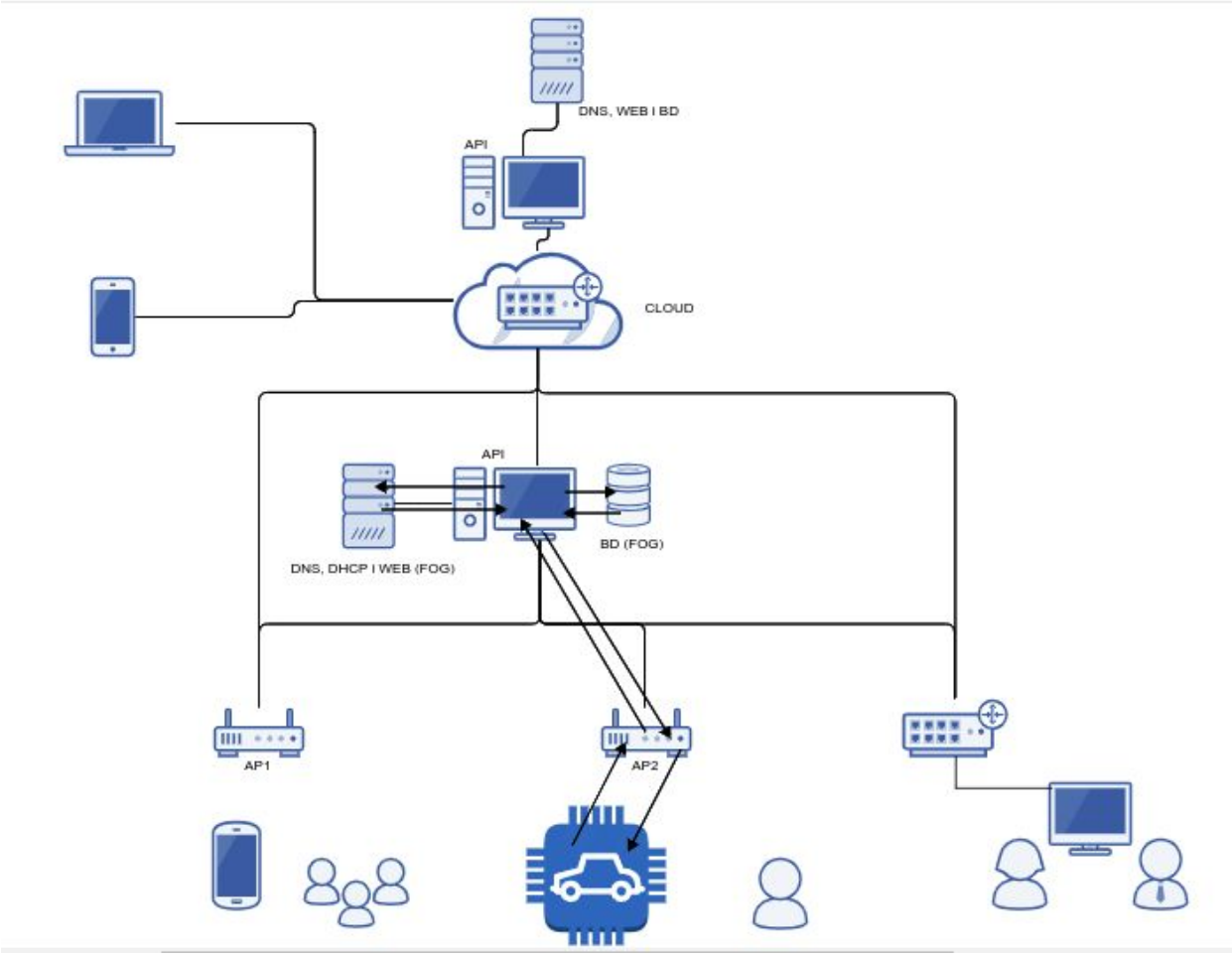
Llavors si la petició ho requereix l'API retornarà la resposta que ha obtingut de la base de dades del cloud, en cas contrari només emmagatzemarà les dades com ho demana la petició a la base de dades del cloud, en el següent esquema observem aquesta comunicació:



Com podem observar en aquest segon cas l'informació passa per més node i això requereix més temps que en el primer cas, però aquest cas no és l'habitual i l'evitarem a la mesura del possible. Ja que sempre mantindrem en la base de dades del fog la informació més sol·licitada i la que respon a les peticions que han de ser resoltes el més ràpid possible.

Comunicacions del cotxe autònom amb la resta d'elements:

Tota la informació rellevant al cotxe autònom com la geolocalització estarà a l base de dades del fog, aquesta informació l'obtindrà l'API del fog hi ho farà arribar a la base de dades del fog. En el següent esquema podem observar aquesta comunicació:



Docker-compose

El docker-compose es una versió de docker que permet configurar i dirigir tots els contenidors de serveis a la vegada, a diferència de Docker, Docker va individualment aixecant serveis mentre que amb el fitxer de configuració principal (docker-compose.yml) podem aixecar tots els serveis, pararlos, borrar els contenidors, etc.

Instal·lació:

- Prerequisits:

S'ha de tenir instal·lat el docker, per això partirem de la màquina virtual que ja el tenia instal·lat i que vam documentar en el sprint anterior

- Procés d'instal·lació:
    1. Ens baixem l'última versió del docker-compose
       ```
       sudo curl -L https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
       ```
    2. Li donem permisos d'execució al directori:
       ```
       sudo chmod +x /usr/local/bin/docker-compose
       ```
    3. Verifiquem la instal·lació:
       ```
       docker-compose --version
       ```
       Veurem → docker-compose version 1.25.5, build 8a1c60f6 (o un build semblant)
    4. Si no funciona l'instal·ació potser hem de fer un link simbólic: (No fa falta)
       ```
       sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
       ```

Configurar Docker-Compose en Cloud:

- Preparació:
    1. Hem de crear un "repositori" (carpeta) a on guardarem tots el arxius. Per exemple:

       ```
       mkdir /Cloud

       cd /Cloud
       ```

    2. També hem de crear una carpeta per cada servei:

       ```
       mkdir apache-php

       mkdir mongobd

       mkdir dns
       ```

    3. I també hem de crear una carpeta pels arxius de la web i per les dades de mongodb

```
mkdir public_html

mkdir mongodb/data
```

- Creació del fitxer docker compose:

  Hem de crear un nou arxiu anomenat “docker-compose.yml” (o amb extensió yaml, ambdues funcionen igual), a on ficarem la següent configuració:

```
version: '3'

services:

 #PHP Service

  app:

    build:

      context: apache-php

      dockerfile: Dockerfile

    image: digitalocean.com/php

    container_name: app

    restart: unless-stopped

    tty: true

    environment:

      SERVICE_NAME: app

      SERVICE_TAGS: dev

    working_dir: /var/www

    volumes:

      - ./apache-php/:/var/www
```

```
-
./apache-php/php/local.ini:/usr/local/etc/php/conf.d/local.
ini
    networks:
      - app-network


  #Nginx Service
  webserver:
    image: nginx:alpine
    container_name: webserver
    restart: unless-stopped
    tty: true
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./apache-php/:/var/www
      - ./apache-php/nginx/conf.d/:/etc/nginx/conf.d/
    networks:
      - app-network


  dns:
    build: './dns/'
```

```
mongodb:
    image: mongo
    container_name: 'mongo-container'
    environment:
      MONGO_INITDB_DATABASE: db
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: admin
    ports:
      - 2717:2717
      - 27017:27017
    volumes:
                                                                    -
./init-mongo.js:/root/Cloud/mongodb/data/init-mongo.js:ro
      - mongodb_data_container:/data/db
    command: mongod --auth
#Docker Networks
networks:
  app-network:
    driver: bridge
#Volumes
volumes:
  dbdata:
    driver: local
```

```
mongodb_data_container:
```

- Pel servei de web necessitarem el Dockerfile i els propis arxius de la web (index.php per exemple) a la carpeta apache-php, els arxius de la web poden tant quedar-se a la carpeta 'web' en la carpeta apache-php, o anar a la carpeta public_html, però si es deixen a la carpeta apache-php s'ha de modificar el 'volumes' del docker-compose.yml per posar el path fins la carpeta web en comptes de la carpeta public_html

  S'ha de tenir en compte que els usuaris que volen accedir a la web han de poder tenir accés a la web, i s'han de garantir accessos en aquests arxius, per començar, només els posarem a 755 (llegir, escriure i executar pel root, llegir i executar pel grup de root i llegir i executar per altres)

- Pel servei dns necessitarem el Dockerfile i l'arxiu dnsmasq.conf a la carpeta dns, aquest últim ha de modificar-se si es canvia la IP
- Pel servei MongoDB necessitem crear a la carpeta de mongo un directori pel data/db especificat al 'volumes' de mongodb, i opcionalment, un script init-mongo que s'executarà quan s'inicii el servei mongo

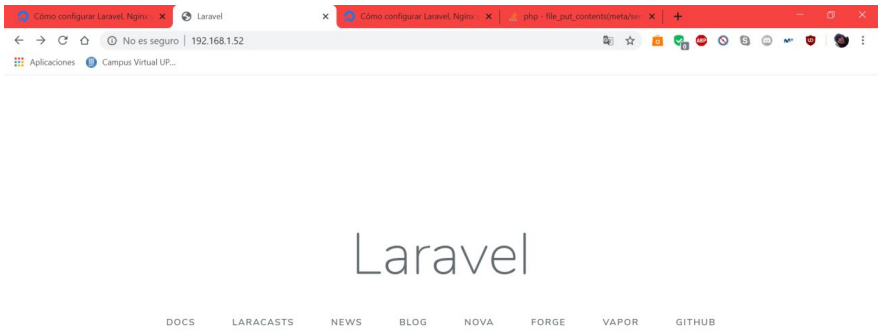
Proves:

Apache+php:

- Només fent: (si no s'han aixecat abans els contenidors)

  ```
  docker-compose up
  ```

  ja hauriem de tenir connexió amb la web:



mongodb:

- Només fent: (si no s'han aixecat abans els contenidors)

  ```
  docker-compose up
  ```

  ja hauriem de poder connectar-nos amb, per exemple, el usuari root especificat a docker-compose.yml: admin. Exemple en una màquina remota:

  ```
  mongo -u admin -p admin --authenticationDatabase admin 192.168.1.121
  ```

```
root@seax~# mongo -u admin -p admin --authenticationDatabase admin 192.168.1.121
MongoDB shell version v4.2.5
connecting to: mongodb://192.168.1.121:27017/test?authSource=admin&compressors=disabled&gssapiServic
eName=mongodb
Implicit session: session { "id" : UUID("2a2cb21c-f450-4d82-a05a-7971c0d06a1e") }
MongoDB server version: 4.2.6
Server has startup warnings:
2020-05-02T01:56:08.506+0000 I  STORAGE  [initandlisten]
2020-05-02T01:56:08.506+0000 I  STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is str
ongly recommended with the WiredTiger storage engine
2020-05-02T01:56:08.506+0000 I  STORAGE  [initandlisten] **          See http://dochub.mongodb.org/c
ore/prodnotes-filesystem
2020-05-02T01:56:09.915+0000 I  CONTROL  [initandlisten]
2020-05-02T01:56:09.916+0000 I  CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_huge
page/enabled is 'always'.
2020-05-02T01:56:09.916+0000 I  CONTROL  [initandlisten] **        We suggest setting it to 'never'
2020-05-02T01:56:09.916+0000 I  CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Configurar Docker-Compose en Fog:

Al fog tindrem un Docker-Compose que agruparà els següents serveis:

- Servidor web (amb laravel)
- Servidor DHCP
- Servidor DNS

- **Preparació:**
  1. Creem un directori per al projecte:

     ```
     # mkdir /Fog

     # cd /Fog
     ```

  2. Creem un directori per a cada servei:

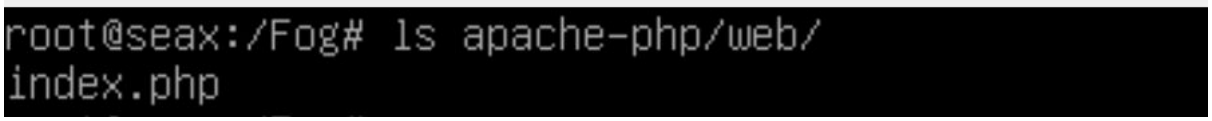
```
# mkdir /Fog/apache-php

# mkdir /Fog/dhcp

# mkdir /Fog/dns
```

3. Creem també un directori per a l'índex de la web:

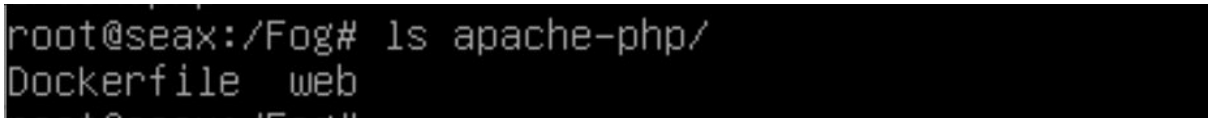
```
# mkdir /Fog/apache-php/web
```

4. Posem a cada directori els arxius corresponents:

A `/Fog/apache-php/web` posem `index.php`.



A `/Fog/apache-php` posem el Dockerfile:



- **Creació del fitxer docker compose yml:**

1. Hem de definir els serveis que tindrem al nostre docker-compose. Creem un arxiu anomenat docker-compose.yml al directori del nostre projecte:

```
# nano /Fog/docker-compose.yml
```

2. Especifiquem a aquest fitxer la següent configuració:

```
version: '3'
```

```
services:
 #PHP Service
  app:
    build:
      context: apache-php
      dockerfile: Dockerfile
    image: digitalocean.com/php
    container_name: app
    restart: unless-stopped
    tty: true
    environment:
      SERVICE_NAME: app
      SERVICE_TAGS: dev
    working_dir: /var/www
    volumes:
      - ./apache-php/:/var/www
                                                                -
./apache-php/php/local.ini:/usr/local/etc/php/conf.d/local.
ini
    networks:
      - app-network


  #Nginx Service
```

```
  webserver:
    image: nginx:alpine
    container_name: webserver
    restart: unless-stopped
    tty: true
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./apache-php/:/var/www
      - ./apache-php/nginx/conf.d/:/etc/nginx/conf.d/
    networks:
      - app-network
  dns:
    build: './dns/'
```

3. Comprovem que l'arxiu no té cap error. Ens hem de situar al directori on tenim el docker-compose.yml i fem:

```
# docker-compose config
```

Si ens surt el contingut de l'arxiu es que tot és correcte:

```
root@seax:/Fog# docker-compose config
services:
  apache-php:
    build:
      context: /Fog/apache-php
    ports:
    - published: 8080
      target: 80
    volumes:
    - /Fog/apache-php/web:/var/www/html:rw
version: '3.8'
```

Hem de tenir en compte que no podem usar tabuladors ja que es queixa, hem d'utilitzar espais. Tampoc podem deixar líneas en blanc, ja que també serà considerat un error.

- **Arrenquem l'aplicació:**

1. Per posar tot en marxa fem al directori on tenim el docker-compose.yml :

```
# docker-compose up
```

```
root@seax:/Fog# docker-compose up
Starting fog_apache-php_1 ... done
Attaching to fog_apache-php_1
apache-php_1  | AH00558: apache2: Could not reliably determine the server's fully qualifi
ame, using 172.18.0.2. Set the 'ServerName' directive globally to suppress this message
apache-php_1  | AH00558: apache2: Could not reliably determine the server's fully qualifi
ame, using 172.18.0.2. Set the 'ServerName' directive globally to suppress this message
apache-php_1  | [Fri May 01 16:40:03.257977 2020] [mpm_prefork:notice] [pid 1] AH00163: A
5 (Debian) PHP/7.0.33 configured -- resuming normal operations
apache-php_1  | [Fri May 01 16:40:03.258923 2020] [core:notice] [pid 1] AH00094: Command
he2 -D FOREGROUND'
-
```

Si volem que corri en background llavors hem de fer:

```
# docker-compose up -d
```

```
root@seax:/Fog# docker-compose up -d
Starting fog_apache-php_1 ... done
```

2. Comprovem quins serveis estan corrent fent al directori on tenim el docker-compose.yml:

```
# docker-compose ps
```

```
root@seax:/Fog# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
fog_apache-php_1   docker-php-entrypoint apac ...   Up      0.0.0.0:8080->80/tcp
```

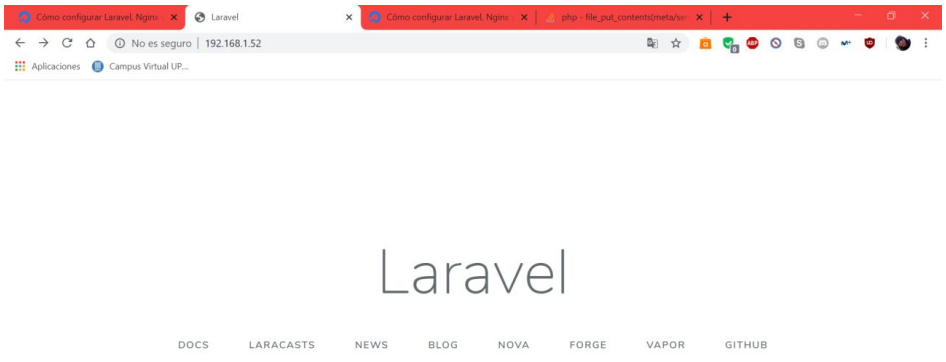
3. Per aturar fem:

```
# docker-compose down
```

```
root@seax:/Fog# docker-compose down
Stopping fog_apache-php_1 ... done
Removing fog_apache-php_1 ... done
Removing network fog_default
```

- **Comprovacions:**

1. Per comprovar que ens podem connectar, simplement ens connectem a la ip pel port corresponent:



Webgrafia Docker-compose

https://docs.docker.com/compose/gettingstarted/

https://www.digitalocean.com/community/tutorials/how-to-set-up-laravel-nginx-and-mysql-with-docker-compose

https://dev.to/aschmelyun/the-beauty-of-docker-for-local-laravel-development-13c0

https://www.digitalocean.com/community/tutorials/how-to-install-docker-compose-on-ubuntu-18-04

https://devhints.io/docker-compose

https://www.linode.com/docs/applications/containers/how-to-use-docker-compose/