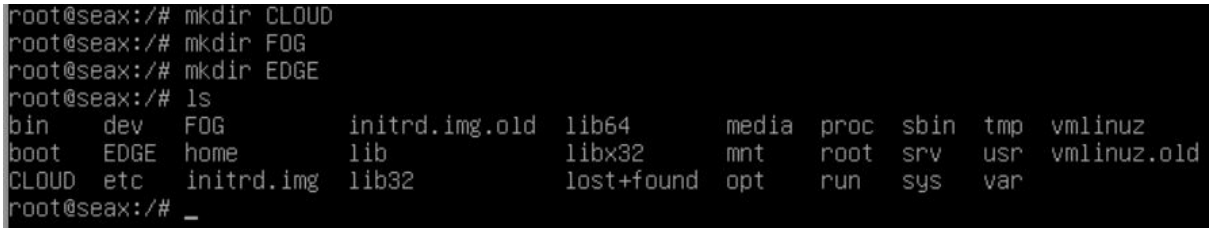
SPRINT 4

Index

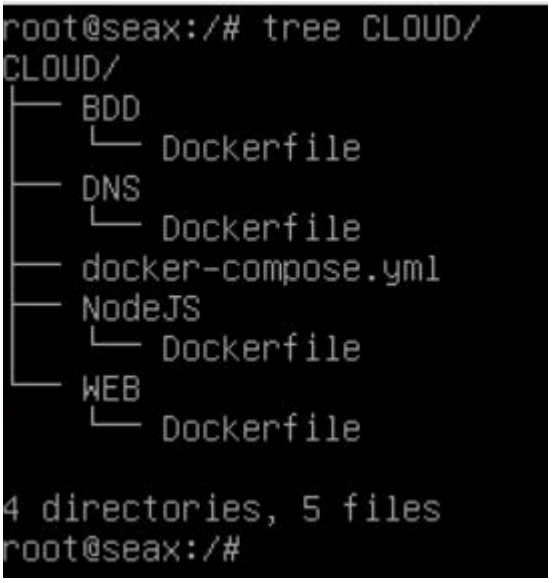
Serveis Docker-Compose

Estructura de directoris

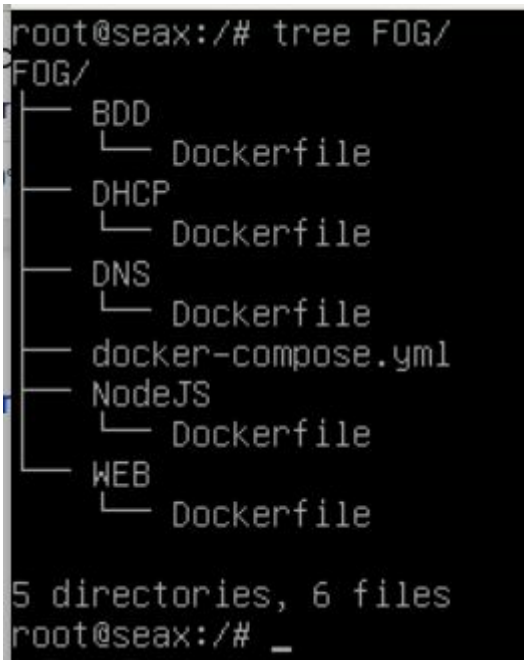
Estructurem cada servei docker creant un directori per cada servei i un directori per cada capa, d'aquesta manera quan aixequem tots els serveis amb docker-compose ens serà més fàcil. Primer de tot crearem els directoris per cada capa a partir de l'arrel (/) llavors dintre de cada directori per capa crearem un directori per cada servei que hi ha allà. Creem els directoris per cada capa de la següent manera:



Ara creem els directoris per cada servei que hi ha al cloud, el fitxer docker-compose.yml i el fitxer Dockerfile per cada servei que es farà servir quan aixequem els serveis amb docker-compose:



Després creem els directoris per cada servei que hi ha al fog, el fitxer docker-compose.yml i el fitxer Dockerfile per cada servei que es farà servir quan aixequem els serveis amb docker-compose:



En principi el directori corresponent a la capa edge està buit però més endavant es posarà el contingut que sigui necessari.

MongoDB

INSTAL·LACIÓ MONGODB AMB DOCKER-COMPOSE

Instal·larem Mongo amb docker-compose seguint una estructura de directoris. En aquest cas, parlaré durant tots els passos del directori **/CLOUD**, però per fer-ho a **/FOG** cal seguir exactament tots els passos.

1. **Preparacions inicials**

1.1 L'estructura de la carpeta **CLOUD** abans de començar és:

```
/CLOUD/
├── BDD
│   └── Dockerfile
├── DNS
│   └── Dockerfile
├── docker-compose.yml
├── NodeJS
│   └── Dockerfile
└── WEB
    └── Dockerfile

4 directories, 5 files
```

Quan acabem tindrà aquest aspecte:

```
/CLOUD
├── BDD
│   ├── data
│   │   ├── db
│   │   └── log
│   ├── Dockerfile
│   └── mongod.conf
├── DNS
│   └── Dockerfile
├── docker-compose.yml
├── NodeJS
│   └── Dockerfile
└── WEB
    └── Dockerfile
```

1.2 Ens baixem la imatge de mongo, en cas de no tenir-la i comprovem que s'ha descarregat correctament:

Primer, podem comprovar si tenim o no la imatge de mongo:

```
# docker images
```

```
root@seax:/CLOUD/BDD# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
   SIZE
```

Si no la tenim, la descarreguem:

```
# docker pull mongo:versió
```

En el nostre cas, volem la última versió:

```
# docker pull mongo:latest
```

```
root@seax:/CLOUD/BDD# docker pull mongo:latest
latest: Pulling from library/mongo
23884877105a: Pull complete
bc38caa0f5b9: Pull complete
2910811b6c42: Pull complete
36505266dcc6: Pull complete
a4d269900d94: Pull complete
5e2526abb80a: Pull complete
d3eece1f39ec: Pull complete
358ed78d3204: Pull complete
1a878b8604ae: Pull complete
dde03a2883d0: Pull complete
4ffe534daa34: Pull complete
f164ba21e17c: Pull complete
6494c387442c: Pull complete
Digest: sha256:50d7b0aef8165b542612a4f57fd7b70703eb7db095588fb76e5a3f01cda396a0
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
```

Comprovem que tot ha anat correctament:

```
# docker images
```

```
root@seax:/CLOUD/BDD# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mongo               latest              3f3daf863757        13 days ago         388MB
```

1.3 Creem els directoris i arxius que necessitarem al directori **/CLOUD/BDD**:

- L'arxiu de configuració de mongo (mongo.conf), que omplirem després
- El directori data/db
- El directori data/log

```
root@seax:/CLOUD/BDD# touch mongod.conf
root@seax:/CLOUD/BDD# mkdir data
root@seax:/CLOUD/BDD# mkdir data/db
root@seax:/CLOUD/BDD# mkdir data/log
```

Ens quedarà de la següent manera:

```
BDD/
├── data
│   ├── db
│   └── log
└── mongod.conf
```

2. **Fitxer mongod.conf**

   Hem d'omplir el fitxer mongod.conf amb el següent:

```
# nano /CLOUD/BDD/mongod.conf
```

mongod.conf:

```
# mongod.conf

# for documentation of all options, see:
#
http://docs.mongodb.org/manual/reference/configuration-option
s/

# Where and how to store data.
storage:
  dbPath: /var/lib/mongodb
  journal:
    enabled: true
#  engine:
#  mmapv1:
#  wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0


# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

#security:
security:
    authorization: enabled
#operationProfiling:

#replication:

#sharding:
```

```
## Enterprise-Only Options:

#auditLog:

#snmp:
```

3. **Fitxer .yml**

3.1 Modificarem el fitxer docker-compose.yml que es troba al directori **/CLOUD,** i afegim el següent:

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

docker.compose.yml:

```
version: '3'
services:
  mongo:
    image: 'mongo:latest'
    container_name: 'my_mongo'
    hostname: 'my_mongo'
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=admin
    volumes:
      - ./BDD/mongod.conf:/etc/mongod.conf
      - ./BDD/data/db:/data/db/
      #- ./BDD/initdb.d/:/docker-entrypoint-initdb.d/
      - ./BDD/data/log/:/var/log/mongodb/
    ports:
      - '27017-27019:27017-27019'
```

- **image** → En aquest cas, com partirem directament d'una imatge oficial, utilitzem aquest camp, on especifiquem que volem la última versió de mongo.
- **container_name** -> El nom que li posem al contenidor. En aquest cas s'anomenarà 'my_mongo', però aquest camp és totalment opcional.

- **hostname** → Posa el hostname que vulguem al contenidor.

- **environment** → Variables que s'utilitzaran al contenidor.

  Indiquem quin username i password volem per a root:

  ```
  MONGO_INITDB_ROOT_USERNAME=admin
  MONGO_INITDB_ROOT_PASSWORD=admin
  ```

- **volumes** → Aquí podem indicar fitxers o directoris que utilitzarem al contenidor. La part de davant dels dos punts és un directori/fitxer local i la part de després és la que correspon a dintre del contenidor.

  `./BDD/mongod.conf:/etc/mongod.conf` → Muntarà el nostre fitxer de configuració situat a ./BDD/mongod.conf com el fitxer de configuració al contenidor. Així permetem que es puguin fer modificaciones fàcilment des de fora.

  `./BDD/data/db:/data/db/` → Muntarà el directori que conté tots els fitxers de les bases de dades a ./BDD/data/db, per així poder tenir tots aquest fitxers locament.

  `./BDD/data/log/:/var/log/mongodb/` → Com l'anterior, podrem tenir els fitxers de log localment, sense necessitat d'accedir al contenidor.

3.2 Comprovem que l'hem escrit tot correctament i el fitxer docker-compose.yml no té cap fallo, executant al directori on es troba l'arxiu, el següent:

```
# docker-compose config
```

```
root@seax:/CLOUD# docker-compose config
services:
  mongo:
    container_name: my_mongo
    environment:
      MONGO_INITDB_ROOT_PASSWORD: admin
      MONGO_INITDB_ROOT_USERNAME: admin
    hostname: my_mongo
    image: mongo:latest
    ports:
    - 27017:27017/tcp
    - 27018:27018/tcp
    - 27019:27019/tcp
    volumes:
    - /CLOUD/BDD/mongod.conf:/etc/mongod.conf:rw
    - /CLOUD/BDD/data/db:/data/db:rw
    - /CLOUD/BDD/data/log:/var/log/mongodb:rw
version: '3.0'
```

4. Accés al servei

4.1 Arrenquem el servei per a que corri en backgrund (opció -d):

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

```
root@seax:/CLOUD# docker-compose up -d
Creating network "cloud_default" with the default driver
Creating my_mongo ... done
```

4.2 Per aturar el servei hauriem d'executar una d'aquestes dues comandes:

```
# docker-compose down
# docker compose stop
```

4.3 Comprovem que estigui corrent:

```
# docker-compose ps
```

```
root@seax:/CLOUD# docker-compose ps
  Name                Command             State                Ports
-------------------------------------------------------------------------------
my_mongo   docker-entrypoint.sh mongod   Up      0.0.0.0:27017->27017/tcp,
                                                 0.0.0.0:27018->27018/tcp,
                                                 0.0.0.0:27019->27019/tcp
```

```
# docker container ls
```

```
root@seax:/CLOUD# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED
        STATUS              PORTS                                      NAMES
b90e0f8b34f6        mongo:latest        "docker-entrypoint.s…"   2 hours ago
        Up 2 hours          0.0.0.0:27017-27019->27017-27019/tcp       my_mongo
```

4.4 Per accedir al servei:

```
# docker exec -it nom_contenidor bash
```

En el nostre cas:

```
# docker exec -it my_mongo bash
```

```
root@seax:/CLOUD# docker exec -it my_mongo bash
root@my_mongo:/#
```

5. Proves en local

5.1 Per accedir com admin hem de fer:

```
# mongo -u admin -p admin
```

```
root@my_mongo:/# mongo -u admin -p admin
MongoDB shell version v4.2.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("84a12e75-d172-4cfd-ab44-d07c29e4a8f5") }
MongoDB server version: 4.2.6
Server has startup warnings:
2020-05-08T09:54:01.531+0000 I  STORAGE  [initandlisten]
2020-05-08T09:54:01.531+0000 I  STORAGE  [initandlisten] ** WARNING: Using the XFS filesyst
2020-05-08T09:54:01.532+0000 I  STORAGE  [initandlisten] **          See http://dochub.mong
2020-05-08T09:54:02.565+0000 I  CONTROL  [initandlisten]
2020-05-08T09:54:02.565+0000 I  CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transpa
2020-05-08T09:54:02.565+0000 I  CONTROL  [initandlisten] **        We suggest setting it to
2020-05-08T09:54:02.565+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
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
```

Com veiem, amb admin podem veure totes les bases de dades.
O bé així si no volem posar la password a la línea de comandes:

```
# mongo -u admin
```

```
root@my_mongo:/# mongo -u admin
MongoDB shell version v4.2.6
Enter password:
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceN
ame=mongodb
Implicit session: session { "id" : UUID("c480ade4-f6bb-47e2-a888-66d92778260b
") }
MongoDB server version: 4.2.6
Server has startup warnings:
2020-05-08T10:45:15.845+0000 I  STORAGE  [initandlisten]
2020-05-08T10:45:15.845+0000 I  STORAGE  [initandlisten] ** WARNING: Using th
e XFS filesystem is strongly recommended with the WiredTiger storage engine
2020-05-08T10:45:15.845+0000 I  STORAGE  [initandlisten] **          See http
://dochub.mongodb.org/core/prodnotes-filesystem
2020-05-08T10:45:17.597+0000 I  CONTROL  [initandlisten]
2020-05-08T10:45:17.598+0000 I  CONTROL  [initandlisten] ** WARNING: /sys/ker
nel/mm/transparent_hugepage/enabled is 'always'.
2020-05-08T10:45:17.598+0000 I  CONTROL  [initandlisten] **        We suggest
 setting it to 'never'
2020-05-08T10:45:17.598+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, e
tc).

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 p
roduct
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.disableFr
eeMonitoring()
---

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

5.2 Fins aquest punt no hem creat cap usuari més. Veurem com accedir en el cas de
terni-ho, i veurem com crear-lo al punt 6.

```
# mongo -u username -p password base_de_dades
```

En aquest cas:

```
# mongo -u user_test -p user_test test
```

```
root@my_mongo:/# mongo -u user_test -p user_test test
MongoDB shell version v4.2.6
connecting to: mongodb://127.0.0.1:27017/test?compressors=disabled&gssapiServ
Implicit session: session { "id" : UUID("aff0f4b7-59bb-4806-bf74-e00dd45a3ead
MongoDB server version: 4.2.6
> show dbs
test  0.000GB
>
```

Amb aquest usuari només podrem veure la seva base de dades, a les demés no té accés.

També podem accedir sense indicar la password a la línea de comandes de la següent manera:

```
# mongo -u username base_de_dades
# mongo -u user_test test
```

```
root@my_mongo:/# mongo -u user_test test
MongoDB shell version v4.2.6
Enter password:
connecting to: mongodb://127.0.0.1:27017/test?compressors=disabled&gssapiService
Name=mongodb
Implicit session: session { "id" : UUID("7c70f692-1d90-4d06-962a-e4393e6da9ab")
}
MongoDB server version: 4.2.6
> show dbs
test  0.000GB
>
```

5.3 Com podem veure les dades es guarden de forma loca, no al contenidor:

```
root@seax:/CLOUD# tree BDD/
BDD/
├── data
│   ├── db
│   │   ├── collection-0-1403795181578330208.wt
│   │   ├── collection-0--4576474797902697218.wt
│   │   ├── collection-2--4576474797902697218.wt
│   │   ├── collection-4--4576474797902697218.wt
│   │   ├── collection-8--4576474797902697218.wt
│   │   ├── diagnostic.data
│   │   │   ├── metrics.2020-05-08T10-45-17Z-00000
│   │   │   └── metrics.interim
│   │   ├── index-10--4576474797902697218.wt
│   │   ├── index-1-1403795181578330208.wt
│   │   ├── index-1--4576474797902697218.wt
│   │   ├── index-3--4576474797902697218.wt
│   │   ├── index-5--4576474797902697218.wt
│   │   ├── index-6--4576474797902697218.wt
│   │   ├── index-9--4576474797902697218.wt
│   │   ├── journal
│   │   │   ├── WiredTigerLog.0000000002
│   │   │   ├── WiredTigerPreplog.0000000001
│   │   │   └── WiredTigerPreplog.0000000002
│   │   ├── _mdb_catalog.wt
│   │   ├── mongod.lock
│   │   ├── sizeStorer.wt
│   │   ├── storage.bson
│   │   ├── WiredTiger
│   │   ├── WiredTigerLAS.wt
│   │   ├── WiredTiger.lock
│   │   ├── WiredTiger.turtle
│   │   └── WiredTiger.wt
│   └── log
├── Dockerfile
└── mongod.conf

5 directories, 28 files
```

6. Creació de BDD i usuaris de prova

6.1 Creem una base de dades anomenada test:

```
> use test
> # db.test.insert({“name”:”test”})
```

```
> use test
switched to db test
> db.test.insert({"name":"test"})
WriteResult({ "nInserted" : 1 })
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
test    0.000GB
```

6.2 Creem un usuari a la base de dades test, amb permís només per a la base de dades test:

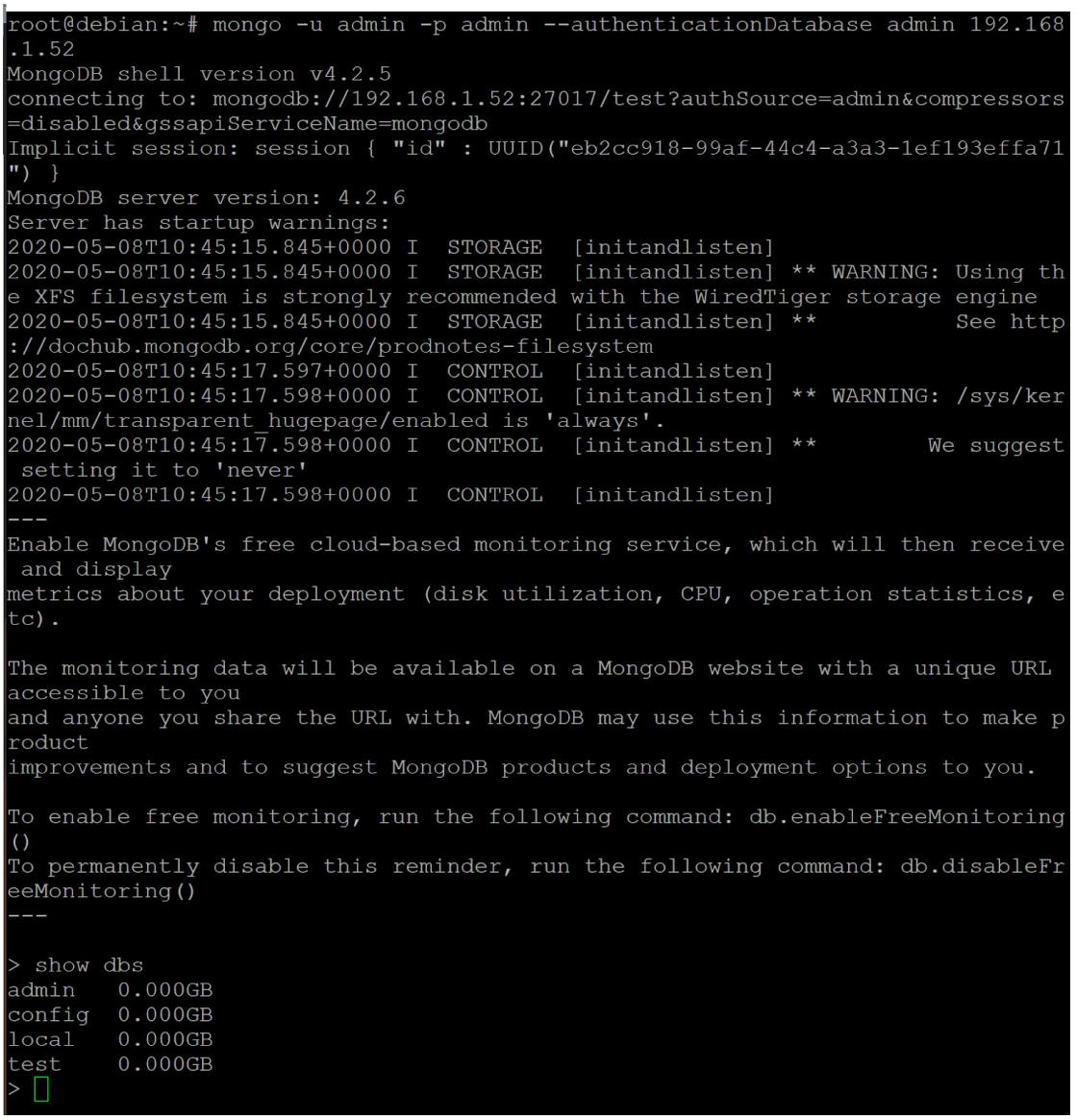
```
> db.createUser({
      user: "user_test",
      pwd: "user_test",
      roles: [
                { role: "userAdmin", db: "test" },
                { role: "dbAdmin",   db: "test" },
                { role: "readWrite", db: "test" }
              ]
  });
```

```
> db.createUser({
...     user: "user_test",
...     pwd: "user_test",
...     roles: [
...               { role: "userAdmin", db: "test" },
...               { role: "dbAdmin",   db: "test" },
...               { role: "readWrite", db: "test" }
...             ]
...  });
Successfully added user: {
        "user" : "user_test",
        "roles" : [
                {
                        "role" : "userAdmin",
                        "db" : "test"
                },
                {
                        "role" : "dbAdmin",
                        "db" : "test"
                },
                {
                        "role" : "readWrite",
                        "db" : "test"
                }
        ]
}
>
```

7. Proves en remot

7.1 Per accedir com admin hem de fer:

```
# mongo -u admin -p admin --authenticationDatabase admin 192.168.1.52
```



7.2 Per accedir amb qualsevol altre usuari:

```
# mongo -u user -p password base_de_dades ip
```

En aquest cas:

```
# mongo -u user_test -p user_test test 192.168.1.52
```

```
root@debian:~# mongo -u user_test -p user_test 192.168.1.52
MongoDB shell version v4.2.5
connecting to: mongodb://192.168.1.52:27017/test?compressors=disabled&gssapiS
erviceName=mongodb
Implicit session: session { "id" : UUID("4a566e80-49ea-45e8-acf7-15e11fe5c3f8
") }
MongoDB server version: 4.2.6
> show dbs
test  0.000GB
>
```

Web

Preparar l'entorn amb laravel

- Descarreguem l'última versió de laravel:

```
git clone https://github.com/laravel/laravel.git laravel-app
```

- Muntem els directoris:

```
docker run --rm -v $(pwd):/app composer install
```

Creem docker-compose.yml

Ho fiquem dintre en la carpeta pare de laravel-php

```
version: '3'
services:

  #PHP Service
  app:
    build:
      context: ./WEB/laravel-app
      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:
      - /FOG/WEB/laravel-app/:/var/www
      - /FOG/WEB/laravel-app/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:
      - /FOG/WEB/laravel-app/:/var/www
      - /FOG/WEB/laravel-app/nginx/conf.d/:/etc/nginx/conf.d/
    networks:
      - app-network

#Docker Networks
networks:
  app-network:
    driver: bridge
```

Creem Dockerfile

Ho fiquem dintre de la carpeta de laravel-php

```
FROM php:7.2-fpm

# Copy composer.lock and composer.json
COPY composer.lock composer.json /var/www/

# Set working directory
WORKDIR /var/www

# Install dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    libpng-dev \
    libjpeg62-turbo-dev \
    libfreetype6-dev \
    locales \
    zip \
    jpegoptim optipng pngquant gifsicle \
    vim \
    unzip \
    git \
    curl

# Clear cache
RUN apt-get clean && rm -rf /var/lib/apt/lists/*
```

```
# Install extensions
RUN docker-php-ext-install pdo_mysql mbstring zip exif pcntl
RUN docker-php-ext-configure gd --with-gd --with-freetype-dir=/usr/include/
--with-jpeg-dir=/usr/include/ --with-png-dir=/usr/include/
RUN docker-php-ext-install gd

# Install composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin
--filename=composer

# Add user for laravel application
RUN groupadd -g 1000 www
RUN useradd -u 1000 -ms /bin/bash -g www www

# Copy existing application directory contents
COPY . /var/www

# Copy existing application directory permissions
COPY --chown=www:www . /var/www

# Change current user to www
USER www

# Expose port 9000 and start php-fpm server
EXPOSE 9000
CMD ["php-fpm"]
```

Configurem el php

Dintre de la carpeta de laravel-app farem el següent:

- Creem directori

```
mkdir laravel-app/php
```

- Creem local.ini

```
upload_max_filesize=40M
post_max_size=40M
```

Configurem el nginx

Dintre de la carpeta de laravel-app farem el següent:

- Creem directori

```
mkdir -p laravel-app/nginx/conf.d
```

- Creem local.ini

```
server {
    listen 80;
    index index.php index.html;
    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    root /var/www/public;
    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
    location / {
        try_files $uri $uri/ /index.php?$query_string;
        gzip_static on;
    }
}
```

Configurar l'entorn

Dintre de la carpeta de laravel-app

```
cp .env.example .env
```

Aixequem els contenidors

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

Generar claus pel laravel

- Generem clau:

```
docker-compose exec app php artisan key:generate
```

- Guardar en la cache:

```
docker-compose exec app php artisan config:cache
```

Donem els permisos

```
sudo chown -R $USER:$USER laravel-app
```

Prova de funcionament

Com queden els directoris

root@seax:/FOG# ls
BDD DHCP DNS **docker-compose.yml** NodeJS **WEB**

root@seax:/FOG# ls WEB/
laravel-app

root@seax:/FOG# ls WEB/laravel-app
app bootstrap composer.json config **Dockerfile** package.json phpunit.xml README.md routes storage vendor
artisan CHANGELOG.md composer.lock database **nginx** **php** public resources server.php tests webpack.mix.js

Exportar les imatges

1- Mirem el id del contenidor a exportar
2- Exportem el contenidor a un .tar

```
root@seax:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
          PORTS               NAMES
799f777e47c2        imagen_web_php      "docker-php-entrypoi…"   39 hours ago        Exited (0) 39 h
ours ago                        nostalgic_solomon
root@seax:~# docker export 799f777e47c2 > apache-php.tar
root@seax:~# ls
apache-php.tar
root@seax:~# _
```

3- Agafarem 2 imatges, la del ngnix i la del php-laravel, agafarem el fitxer de docker-compose.yml i farem un .tar.gz de la web.

4- Un cop tenim el .tar, ara hem de passar-la a una altra màquina utilitzant un USB,ssh o una carpeta compartida o el que ens sigui més còmode.

Importar les imatges

1- Copiem els arxius en les carpetes que tenim

```
root@seax:/FOG# ls
BDD  DHCP  DNS  docker-compose.yml  NodeJS  WEB
root@seax:/FOG# ls WEB/
hola  laravel-app
root@seax:/FOG#
```

2- importem les 2 imatges amb la següent comanda:

```
root@seax:~/webphp# cat /root/apache-php.tar | docker import - imagen_web_php
sha256:96ccce2f4bb6732b2216f365aaf5b0f4fba8d89f93f37f7f919b4780e3f72a92
root@seax:~/webphp# _
```

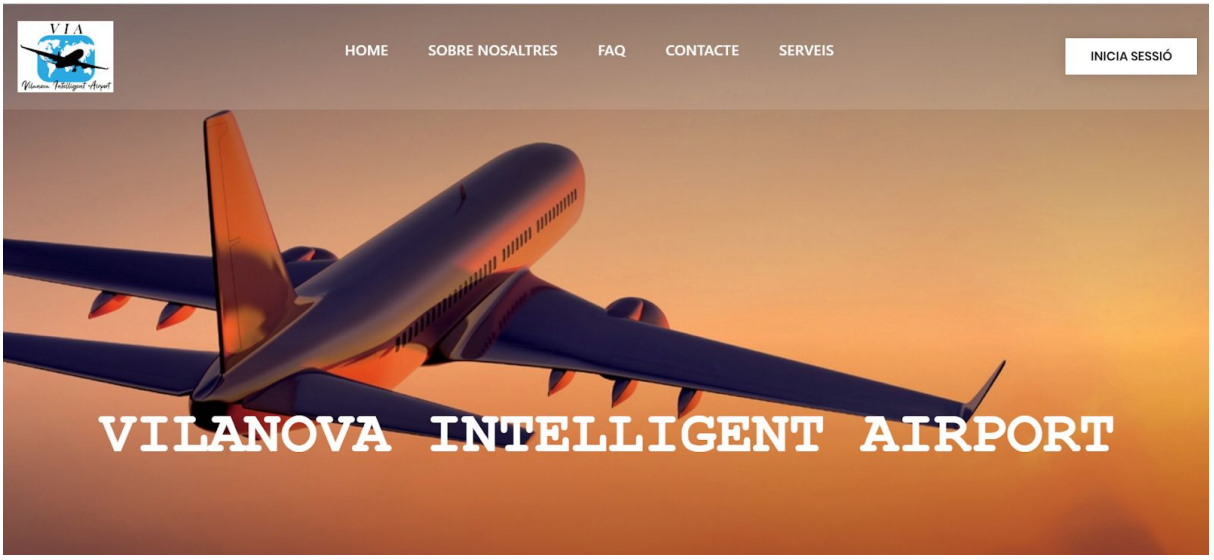
3- Fem el --build de totes les imatges

```
docker-compose up -d --build
```

4- Install compose, executar dintre de /FOG/WEB/laravel-app/

```
docker run --rm -v $(pwd):/app composer install
```

Prova final:



NodeJS

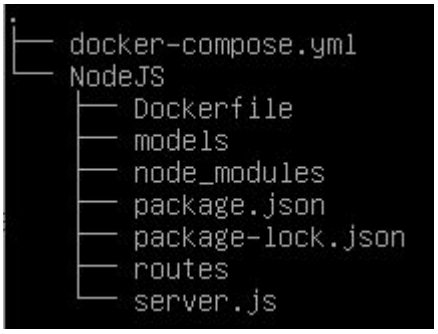
Prerequisits

Tenir instal·lats Docker i Docker-compose (Compatibles):

Nosaltres tenim la versió de

- Docker: 19.03 (versions de yml compatibles: totes)
- Docker-compose: 1.25.5

Muntar el sistema de fitxers inicials per instal·lar-ho tot:



Abans de continuar s'ha de tenir en compte que s'ha partit dels arxius exportats de l'API creada a l'sprint 3 (server.js és app.js, i però la resta de carpetes son importades, l'únic nou es Dockerfile i el docker-compose.yml).

Fitxers necessaris (Docker-compose):

1. Hem de posar al docker-compose.yml:

```
version:  '3'
services:
  nodejs:
    build: ./NodeJS/
    container_name: nodejs
    restart: unless-stopped
    env_file: NodeJS/.env
    ports:
   -  "80:8080"
    volumes:
   -  node_modules:/CLOUD/NodeJS/node_modules
volumes:
  node_modules:
```

S'ha de tenir en compte que el node_modules té el path si la carpeta arrel del projecte s'en diu "CLOUD"

2. Ara al fitxer server.js hem de escriure:

```
-----------------------------------------------------------------
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 dades
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@192.168.1.121:27017/person
a?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://192.168.1.121:3000");
});
```

Atenció!! Tant al “mongoose.connect” com al “console.log” (aquest últim menys important) s’ha de canviar l’ip definida i port al servidor del cloud si aquest canvia. Si el servidor de mongodb está a un contenidor de la mateixa màquina (executat en el mateix docker-compose), podem posar el nom del contenidor en comptes de la ip i la conexió es farà a través de l’interficie de xarxa de docker.

3. Al fitxer package.json posarem tots els atributs del nostre servei, nom, autor, llicencia, dependencies, etc.:

```
{
  "name": "NodeJS",
  "version": "1.0.0",
  "description": "NodeJS amb mongoose, express, bodyParser,
methodOverride i http",
  "main": "app.js",
  "scripts": {
    "start": "node server.js"
  },
  "author": "A2",
  "license": "ISC",
  "dependencies": {
    "body-parse": "^0.1.0",
    "express": "^4.17.1",
    "method-override": "^3.0.0",
    "mongodb": "^3.5.7",
    "mongoose": "^5.9.10"
  }
}
```
---

4. Finalment a Dockerfile posarem el seguent:

```
FROM node:10
WORKDIR /CLOUD/NodeJS
COPY package*.json /CLOUD/NodeJS/
RUN npm install
COPY . /CLOUD/NodeJS

EXPOSE 8080
```

```
CMD [ "npm", "start" ]
```

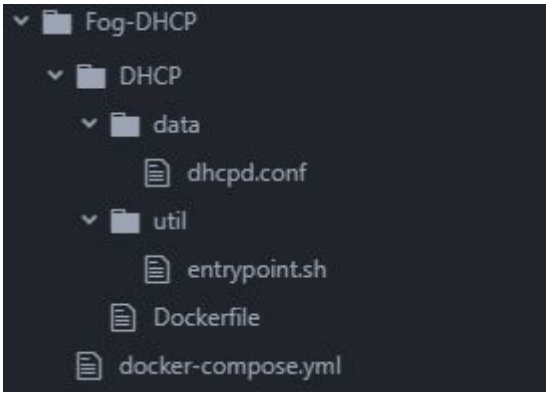
S'ha de tenir en compte que el WORKDIR, i els COPY té el path si la carpeta arrel del projecte s'en diu "CLOUD"
Per ordre significa:
Importarem l'imatge node
Indiquem que treballem al directori /CLOUD/NodeJS
Copiem package.json i package-log.json a /CLOUD/NodeJS (Cosa que haurien d'estar ja)
Instalem totes les dependencies indicades a package.json
Copiem el directori a /CLOUD/NodeJS
Obrim el port 8080
Utilitzem npm i fem "start" (Indicat a l'apartat script de package.json)

DHCP

Estructura del DHCP

Per fer la instal·lació del DHCP partirem de la màquina virtual Debian_Docker_S4.ova que porta instal·lats de sèrie docker i docker-compose. La instal·lació del DHCP es realitzarà en el directori /FOG on aquí es col·locaran els fitxers i directoris necessaris per usar la imatge.

L'estructura del DHCP dins el directori /FOG serà la següent:



Fitxers necessaris:

En el directori /util tindrem el fitxer entrypoint.sh. Aquest fitxer és necessari per el Dockerfile i ve proporcionat per l'autor del contenidor [1]

També crearem un directori anomenat /data on anirà el fitxer dhcpd.conf. Aquest fitxer és el que tindrà els paràmetres del DHCP i l'hem de proporcionar nosaltres segons les nostres preferències
Aquest és el contingut del nostre dhcpd.conf:

```
default-lease-time 600;
max-lease-time 7200;
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.8 192.168.1.250;
  option broadcast-address 192.168.1.255;
  option routers 192.168.1.1;
  option domain-name-servers 192.168.1.3;
}

```

Dockerfile

El Dockerfile que necessitem també en ve donat en [1] i el seu contingut és el següent:

```
FROM ubuntu:18.04

MAINTAINER Robin Smidsrød <robin@smidsrod.no>

ARG DEBIAN_FRONTEND=noninteractive

RUN apt-get -q -y update \
 && apt-get -q -y -o "DPkg::Options::=--force-confold" -o "DPkg::Options::=--force-confdef" install apt-utils \
 && rm /etc/dpkg/dpkg.cfg.d/excludes \
 && apt-get -q -y -o "DPkg::Options::=--force-confold" -o "DPkg::Options::=--force-confdef" install dumb-init isc-dhcp-server man \
 && apt-get -q -y autoremove \
 && apt-get -q -y clean \
 && rm -rf /var/lib/apt/lists/*

COPY util/entrypoint.sh /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]

```

Docker-compose

Ara només ens falta incorporar el servei de DHCP al fitxer docker-compose.yml i afegirem el següent:

```yaml
version: '3'

services:
   #Servei del DHCP
   dhcp:
     build:
       context: ./DHCP
       dockerfile: Dockerfile
     image: networkboot/dhcpd
     container_name: dhcp_ptin
     network_mode: "host"
     restart: always
     ports:
        - "67/udp"
     volumes:
        - ./DHCP/data:/data
```

La línia ‘ **network_mode: “host” ‘ permet al servei usar la direcció que usa el host i obre la porta al món exterior. Es pot usar alhora amb els altres serveis sense problemes.

Arrancar servei

Abans d’arrancar el servei farem un ‘docker-composer build’

```
root@seax:/FOG# docker-compose build
Building dhcp
Step 1/6 : FROM ubuntu:18.04
 ---> c3c304cb4f22
Step 2/6 : MAINTAINER Robin Smidsrød <robin@smidsrod.no>
 ---> Using cache
 ---> 8d3cbad71c96
Step 3/6 : ARG DEBIAN_FRONTEND=noninteractive
 ---> Using cache
 ---> b3271a845783
Step 4/6 : RUN apt-get -q -y update  && apt-get -q -y -o "DPkg::Options::=--force-confold" -o "DPkg:
:Options::=--force-confdef" install apt-utils  && rm /etc/dpkg/dpkg.cfg.d/excludes  && apt-get -q -y
 -o "DPkg::Options::=--force-confold" -o "DPkg::Options::=--force-confdef" install dumb-init isc-dhc
p-server man  && apt-get -q -y autoremove  && apt-get -q -y clean  && rm -rf /var/lib/apt/lists/*
 ---> Using cache
 ---> 565a848e5ef0
Step 5/6 : COPY util/entrypoint.sh /entrypoint.sh
 ---> Using cache
 ---> 5722ce8fa263
Step 6/6 : ENTRYPOINT ["/entrypoint.sh"]
 ---> Using cache
 ---> 80d0cc6cf349
Successfully built 80d0cc6cf349
Successfully tagged networkboot/dhcpd:latest
root@seax:/FOG# _
```

Arranquem el servei amb la comanda ‘docker-compose up [-d] ’ (si afegim -d s’executarà en segon pla)

```
root@seax:~# cd ../FOG/
root@seax:/FOG# ls
BDD  DHCP  DNS  docker-compose.yml  NodeJS  WEB
root@seax:/FOG# docker-compose up
Creating dhcp_ptin ... done
Attaching to dhcp_ptin
dhcp_ptin | Internet Systems Consortium DHCP Server 4.3.5
dhcp_ptin | Copyright 2004-2016 Internet Systems Consortium.
dhcp_ptin | All rights reserved.
dhcp_ptin | For info, please visit https://www.isc.org/software/dhcp/
dhcp_ptin | Config file: /data/dhcpd.conf
dhcp_ptin | Database file: /data/dhcpd.leases
dhcp_ptin | PID file: /var/run/dhcpd.pid
dhcp_ptin | Wrote 3 leases to leases file.
dhcp_ptin | Multiple interfaces match the same subnet: enp0s3 br-f106c18f0980
dhcp_ptin | Multiple interfaces match the same shared network: enp0s3 br-f106c18f0980
dhcp_ptin | Multiple interfaces match the same subnet: enp0s3 br-f106c18f0980
dhcp_ptin | Multiple interfaces match the same shared network: enp0s3 br-f106c18f0980
dhcp_ptin | Listening on LPF/br-f106c18f0980/02:42:6e:cc:db:a7/192.168.1.0/24
dhcp_ptin | Sending on   LPF/br-f106c18f0980/02:42:6e:cc:db:a7/192.168.1.0/24
dhcp_ptin |
dhcp_ptin | No subnet declaration for docker0 (172.17.0.1).
dhcp_ptin | ** Ignoring requests on docker0.  If this is not what
dhcp_ptin |    you want, please write a subnet declaration
dhcp_ptin |    in your dhcpd.conf file for the network segment
dhcp_ptin |    to which interface docker0 is attached. **
dhcp_ptin |
dhcp_ptin | Listening on LPF/enp0s3/08:00:27:49:15:53/192.168.1.0/24
dhcp_ptin | Sending on   LPF/enp0s3/08:00:27:49:15:53/192.168.1.0/24
dhcp_ptin | Sending on   Socket/fallback/fallback-net
dhcp_ptin | Server starting service.
```

Aturar el servei

El servei ja està operatiu i per apagar-lo farem servir la comanda ‘docker-compose down’

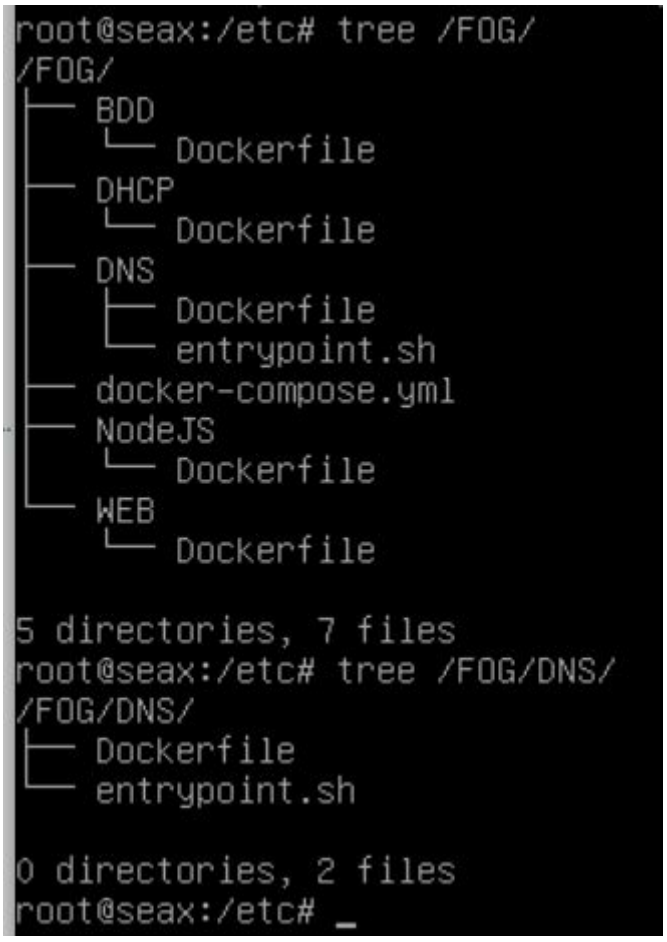
```
root@seax:/FOG# docker-compose down
Stopping dhcp_ptin ... done
Removing dhcp_ptin ... done
root@seax:/FOG#
```

DNS

Estructura del DNS

Per fer la instal·lació del DNS partirem de la màquina virtual Debian_Docker_S4.ova que porta instal·lats de sèrie docker i docker-compose. La instal·lació del DNS es realitzarà en el directori /FOG on aquí es col·locaran els fitxers i directoris necessaris per usar la imatge.

L’estructura del DNS dins el directori /FOG serà la següent:



Modifiquem el fitxer docker-compose.yml amb el següent contingut:

```
version: '3'

services:
  dns:
    build:
      context: ./DNS
      dockerfile: Dockerfile
    image: resystit/bind9
    container_name: dns_ptin
    network_mode: "host"
    restart: always
    ports:
      - "53:53"
      - "53:53/udp"
    volumes:
      - "./DNS/named.conf:/etc/bind/named.conf"
      - "./DNS/VIA.db:/etc/bind/VIA.db"
```

El Dockerfile del DNS, tindrà el següent contingut:

```
FROM alpine:latest

MAINTAINER resyst-it <florian.cauzardjarry@gmail.com>

RUN apk --update add bind

EXPOSE 53

CMD ["named", "-c", "/etc/bind/named.conf", "-g", "-u", "named"]
```

Ara creem el fitxer DNS/named.conf, en aquest fitxer hem de posar les zones de cerca directa del nostre servei DNS, el domini de la nostra zona directa. També hem d'incloure el tipus de servei i en quins fitxers farà la cerca de noms, el contingut del fitxer DNS/named.conf és el següent:

```
options {
        directory "/var/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        forwarders {
                192.168.1.3;
                8.8.8.8;
         };


//========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
```

```
//========================================================================
=========
        dnssec-validation auto;
        auth-nxdomain no;
        listen-on-v6 { any; };
};

//Configuration of the zones
zone "VIA.com"{
        type master;
        file "/etc/bind/VIA.db";

};
```

Després creem el fitxer VIA.db, que hem posat al DNS/named.conf, el fitxer VIA.db tindrà el contingut següent:

```
;
; BIND data file for VIA.db
;
$TTL  86400
@       IN      SOA   VIA.com. root.VIA.com. (
                              2          ; Serial
                         604800              ; Refresh
                          86400              ; Retry
                        2419200              ; Expire
                          86400 )      ; Negative Cache TTL
;
@       IN      NS    VIA.com.
@       IN      A     127.0.0.1
@       IN      AAAA  ::1
;www    IN      A     192.168.1.4
dns     IN      A     192.168.1.2
dhcp    IN      A     192.168.1.1
servidordns IN      CNAME     dns.VIA.com.
```

Muntant la imatge

Abans d'aixecar el servei, executem la comanda # docker-compose build :

Llavors aixequem el servei, executant la comanda # docker-compose up -d :

Per apagar el servei, executem la comanda # docker-compose down

API REST en local

Credencials en la màquina de la API i de mongoDB

Usuari: root
Contrasenya: toor
o
Usuari: entel
Contrasenya: letne

---
comanda: `mongo -u admin`
usuaris mongo: admin
contrasenya mongo: admin

Estructura dels directoris

L'arrel del projecte de la Rest API serà en el directori /myapp partint de l'arrel base. Hi ha dos directoris principals de treball: /routes i /models.

En /routes cada grup té un fitxer anomenat router_AX.js on s'escriuen els mètodes de l'api. El fitxer router_A2.js es pot usar de model per orientar-se al fer els altres fitxers.

Cada un dels grups del terminal tenen un directori personal en /models i s'usa per incloure els seus esquemes de la base de dades. El directori del grup A2 conté un esquema orientatiu de com fer els altres.

Els equips només han d'usar el seu directori de models i el seu fitxer router_AX.js

```
root@seax:/myapp# ls app.js
app.js
root@seax:/myapp# tree routes/
routes/
├── router_A1.js
├── router_A2.js
├── router_A3.js
├── router_A4.js
└── router.old.js

0 directories, 5 files
root@seax:/myapp# tree models/
models/
├── A1
├── A2
│   └── persona.js
├── A3
└── A4

4 directories, 1 file
root@seax:/myapp#
```

Routes

Aquí és on es creen els mètodes de GET/POST/PUT/DELETE, es creen a partir de app.nom_mètode(/ruta_url/:paràmetre, funció), al definir el mètode així, informem que quan es cridi al mètode amb la URL que hem ficat, es farà el que hi hagi a la funció.

```
//Juntem els diferebts métodes de la API i les diferents formes de cridar-los amb les funcions corr
//Tots els següents métodes és cridaran quan l'usuari introdueixi en la URL http://adreça:port/pers
//---------------------------------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);
```

Importem els models que utilitzarem:

```
//----------------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/A2/persona');
//--------------------------------
```

Funcions:

Les funcions que es criden amb els mètodes, han de tenir aquesta estructura:

```
  //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();
  };
```

Aquest és un exemple de post, com veieu les funcions són com les comandes de mongoDB. Es poden veure tots els exemples en /myapp/routes/routes_A2.js

Models

Dintre del directori del teu equip AX, es creen els models, aquests models es creen a partir de cada una de les col·leccions que hi ha a la BDD i que necessitareu.

Important: no us deixeu la part de "collection:'nom_col·lecció' " perquè sinó el mongoose us crea una col·lecció nova amb el nom que indiqueu en el export, pero en plural.
Per cada model, s'ha de posar també un nom nou representatiu al Schema creat.

En l'exemple, veiem una representació dels camps que té una col·lecció a la BDD que es diu persona, que té com a camps: name i lastName.

```
root@seax:/myapp# cat models/A2/persona.js
//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);
root@seax:/myapp#
```

Teniu un exemple /myapp/models/A2/persona.js

Execució del script

Dins el directori /myapp executar la comanda: `node app.js`

La màquina de MongoDB ha d'estar oberta per poder executar correctament l'API

Client visual

Advance REST client
https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo/related?hl=es-419

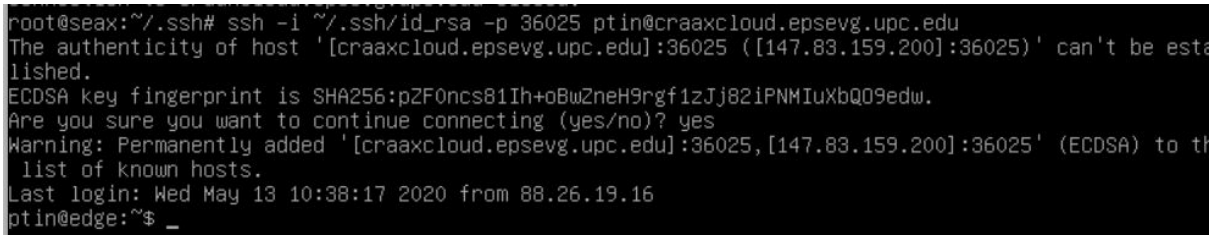
Per més referències visualitzeu el vídeo compartit a l'enllaç dels recursos

Enllaç dels recursos

En el següent enllaç es troben els recursos necessaris:
https://drive.google.com/drive/folders/18D-OhS3VUFGdkSuVmT-BDXq6eTEBFco3?usp=sharing

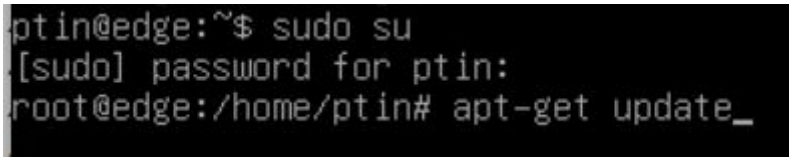
Instal·lació del docker-compose al craax

Ens connectem a l'edge de la màquina del craax amb el parell de la clau privada, de la següent manera:



Com podem observar per fer la connexió hem indicat la clau privada amb l'opció -i i hem escollit el port 36025, que correspon al edge, amb l'opció -p.

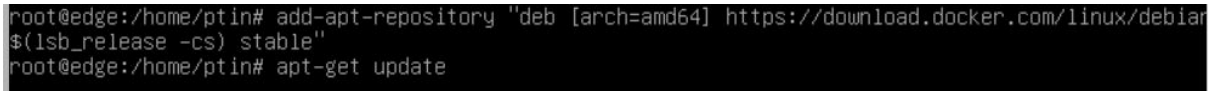
Abans de començar a instal·lar el docker-compose cal tenir instal·lat el docker, per fer-ho canviem d'usuari a root amb la comanda $ sudo su, i actualitzem els paquets del sistema amb la comanda $ update.



La contrasenya que introduïm per passar a l'usuari root és "ptin". Llavors instal·larem una sèrie de paquets que permet treballar amb HTTPS amb la comanda 'apt-get install apt-transport-https ca-certificates curl gnupg2 software- properties-common'

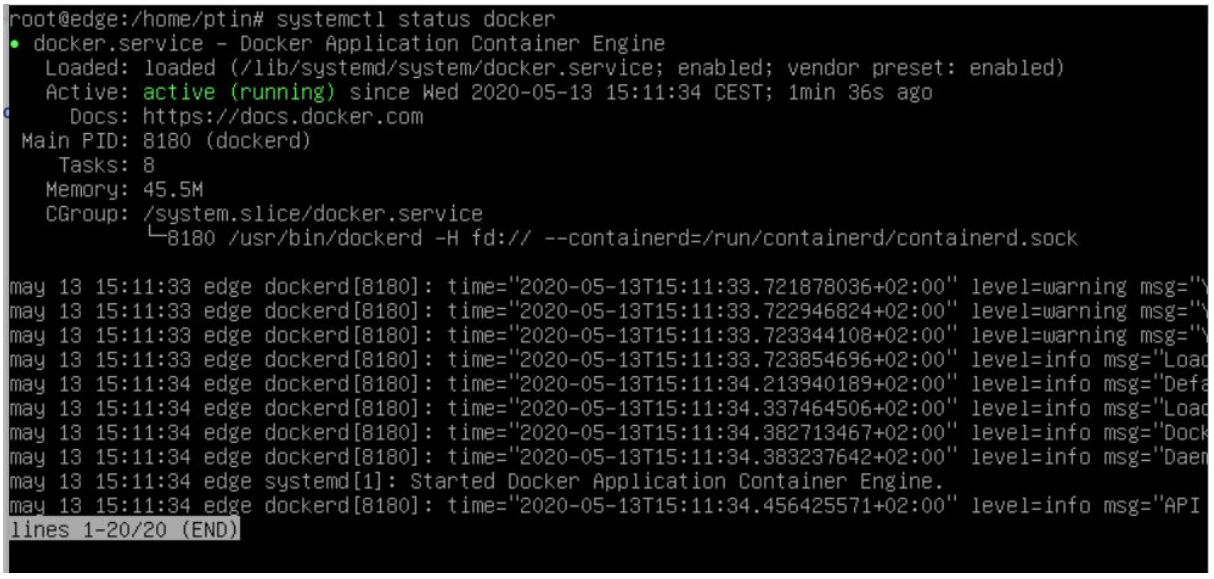
Després obtindrem una clau de Docker per poder accedir al repositori amb la comanda 'curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add –'

A continuació afegim el repositori de Docker amb la comanda 'add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable" '
Llavors tornem a fer un update actualitzar els recursos.



Ara ja podem instal·lar Docker al sistema correctament i ho fem executant la comanda ' apt-get install docker-ce'.

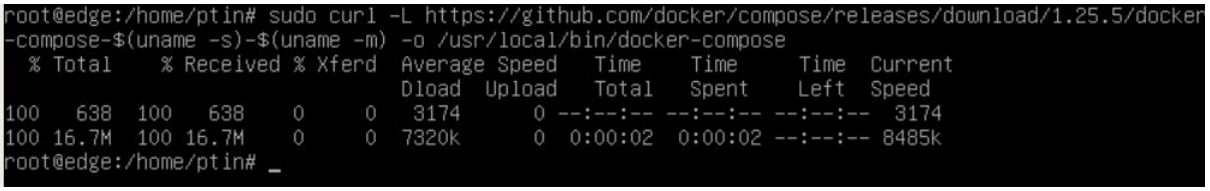
Comprovem que la instal·lació s'ha fet correctament executem la comanda 'systemctl status docker' i veurem que el servei està funcionant.



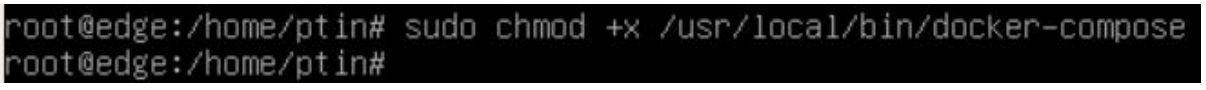
Com podem observar obtenim el missatge de “active (running)” per tant la instal·lació s’ha fet correctament.

Ara que tenim el Docker instal·lat correctament al sistema podem començar amb la instal·lació de docker-compose. Ens baixem l’última versió del docker-compose amb la següent comanda:

```
sudo curl -L
https://github.com/docker/compose/releases/download/1.25.5/doc
ker-compose-$(uname -s)-$(uname -m) -o
/usr/local/bin/docker-compose
```



Li donem permisos d’execució al directori /usr/local/bin/docker-compose



Comprovem que la instal·lació s’ha fet correctament amb la comanda docker-compose --version, veurem un missatge docker-compose version 1.25.5, build 8a1c60f6 (o un build semblant)

```
root@edge:/home/ptin# docker-compose --version
docker-compose version 1.25.5, build 8a1c60f6
root@edge:/home/ptin#
```

Com podem observar la instal·lació del docker-compose s'ha fet correctament.

Transpas al CRAAX

Passar mongoDB i la web a una màquina virtual neta.

WEB

1. Passem lavarel-app al directori /CLOUD
2. Fem un import de les imatges ngnix.tar i php-laravel.tar → imagen

```
# cat /media/usb/SPRINT4/web/php-laravel.tar | docker import -
php_lavarel
```

```
root@seax:/CLOUD/WEB# cat /media/usb/SPRINT4/web/php-laravel.tar | docker imp
ort - php_lavarel
sha256:7fab03fafdeea0c5b3ff23e9e98cbcb5e5affb53f26b95544439e9058d0a9f4a
```

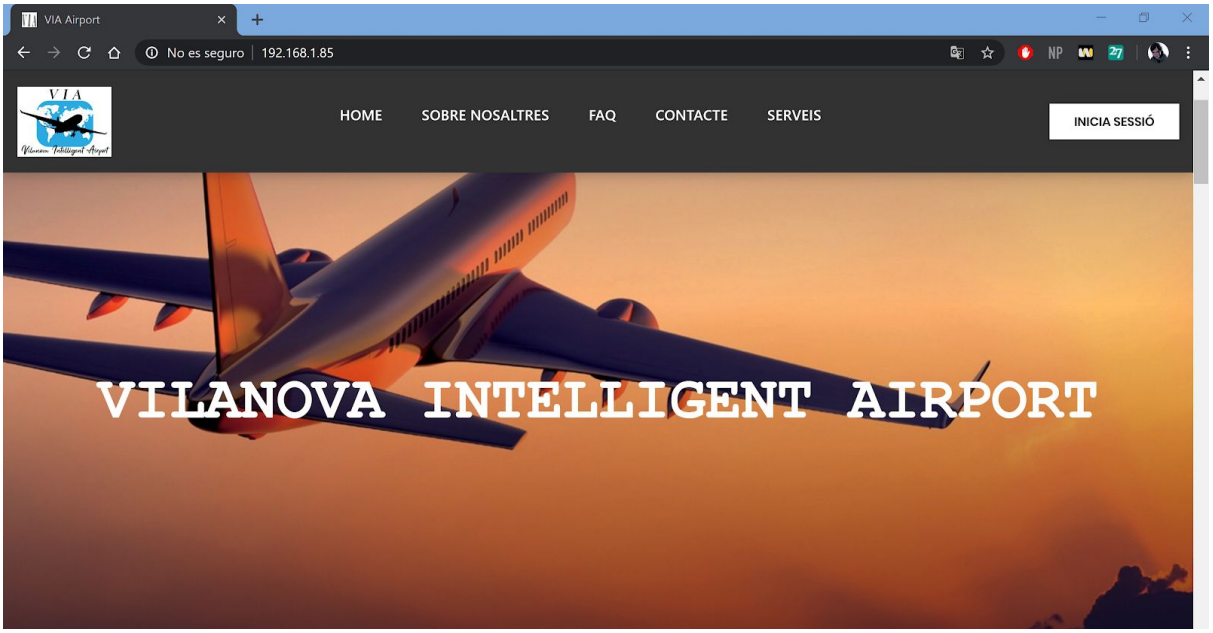
```
# cat /media/usb/SPRINT4/web/nginx.tar | docker import -
nginx.tar
```

```
root@seax:/CLOUD/WEB# cat /media/usb/SPRINT4/web/nginx.tar | docker import -
nginx
sha256:62f8226eb7476ef20602b41e2ef35344c85e99ceb538353df8e0269ec267e4dd
```

3. Comprovem que s'han importat bé:
   ```
   # docker images
   ```

```
root@seax:/CLOUD/WEB# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
    SIZE
nginx               latest              62f8226eb747        32 seconds ago
    19.9MB
php_lavarel         latest              7fab03fafdee        About a minute ag
o   571MB
```

4. Afegim al .yml tot.
5. `# docker-compose up -d --build`
6. Executar dintre de /FOG/WEB/laravel-app/
   ```
   # docker run --rm -v $(pwd):/app composer install
   ```
7. `# chmod 777 CLOUD/WEB/laravel-app/storage`



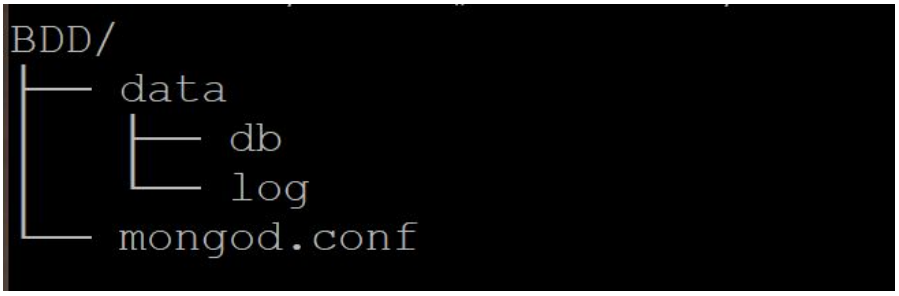
MONGO

1. Pull imatge

   ```
   # docker pull mongo:latest
   ```

2. Copiem l'arxiu mongod.conf a /CLOUD/BDD
3. Creem els directoris de /data/log

   ```
   # mkdir /CLOUD/BDD/data
   # mkdir /CLOUD/BDD/data/db
   # mkdir /CLOUD/BDD/data/log
   ```



4. Copiem data a /CLOUD/data
5. Copiem tot al .yml que quedarà així:

   ```
   # docker-compose config
   ```

```
version: '3'
services:
  # base de dades de mongo
  mongo:
    image: 'mongo:latest'
```

```
    container_name: 'my_mongo'
    hostname: 'my_mongo'
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=admin
    volumes:
      - ./BDD/mongod.conf:/etc/mongod.conf
      - ./BDD/data/db:/data/db/
      #- ./BDD/initdb.d/:/docker-entrypoint-initdb.d/
      - ./BDD/data/log/:/var/log/mongodb/
    ports:
      - '27017-27019:27017-27019'

  #PHP Service
  app:
    build:
      context: ./WEB/laravel-app
      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:
      - ./WEB/laravel-app/:/var/www
      -
./WEB/laravel-app/php/local.ini:/usr/local/etc/php/conf.d/loc
al.ini
    networks:
      - app-network

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

```
#Docker Networks
networks:
  app-network:
    driver: bridge
```

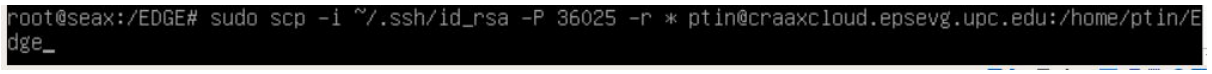
SERVIDOR CLOUD AL CRAAX

web importar:

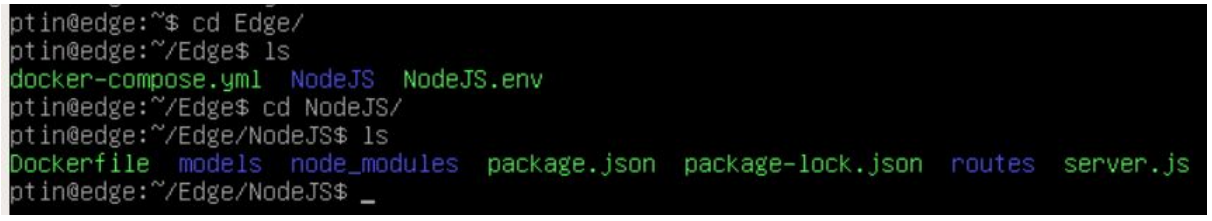
cat docker import 2 sudos

SERVIDOR NodeJs

Per traslladar el servidor de l'edge a la màquina del craax corresponent a l'edge, hem de portar tot el contingut del directori EDGE que tenim localment a la màquina del craax mitjançant la comanda scp. El directori EDGE conté el NodeJS que és el mateix que el del líder del cloud. Comencem amb el traspàs del directori EDGE amb la comanda scp:

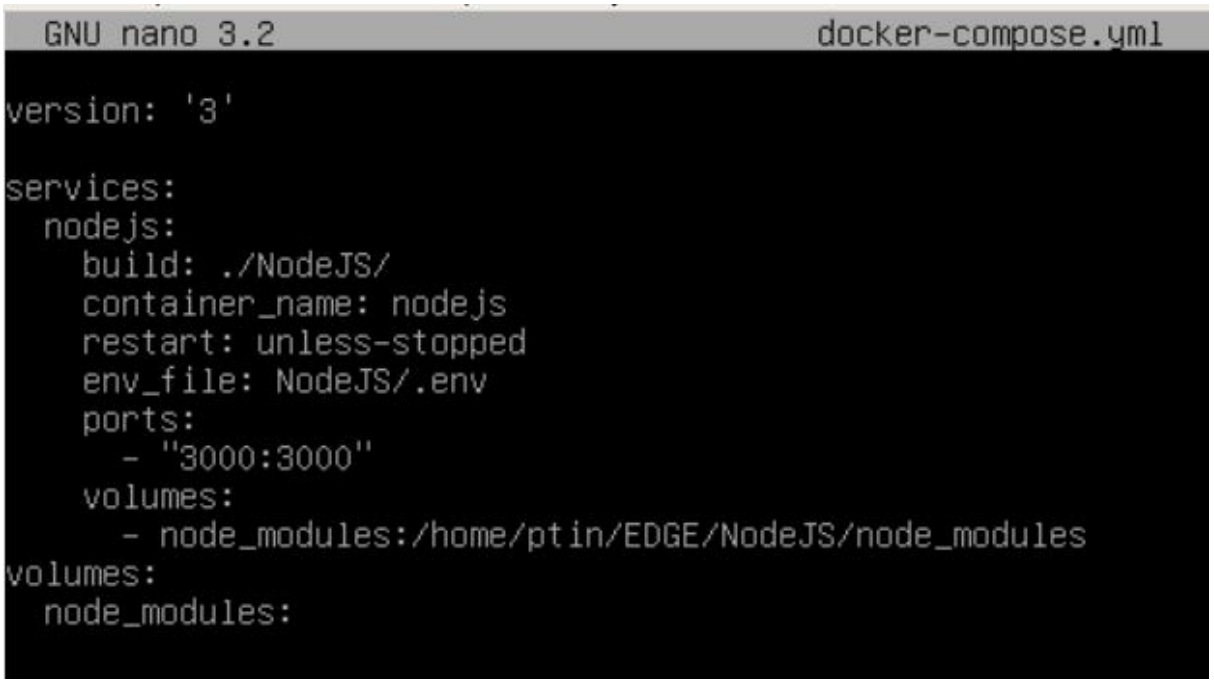


Ara accedim a la màquina edge del craax per comprovar que s'han copiat correctament els fitxers.

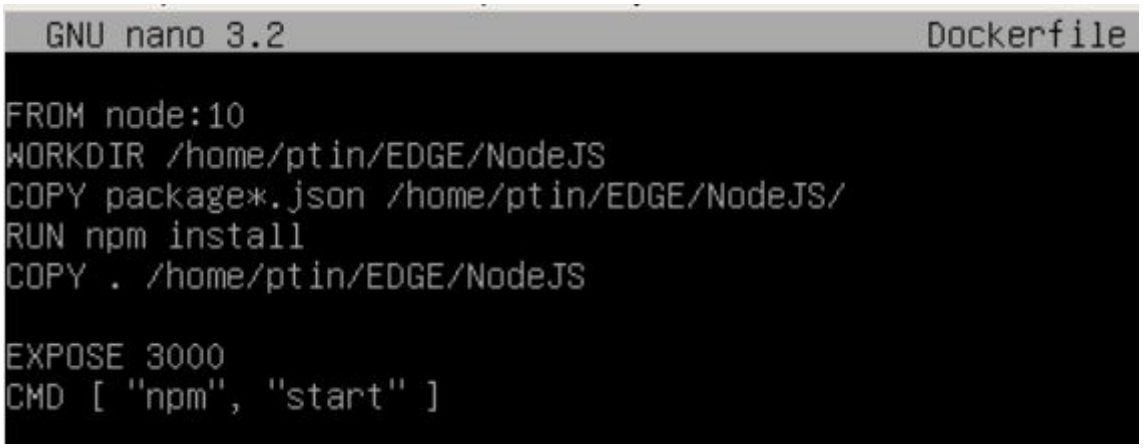


Com podem observar s'han copiat correctament els fitxers. Ara hem de modificar els fitxers docker-compose.yml, Dockerfile i el server.js que hi ha a la màquina de l'edge

del craax amb les dades corresponents. Modifiquem el docker-compose.yml de la següent manera:



Llavors modifiquem el Dockerfile amb el següent contingut:



I el contingut del server.js ha de ser 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
//Comentem el mongoose.connect de moment
//try {

//mongoose.connect('mongodb://admin:admin@192.168.1.52:27017/persona?auth///Source=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://craaxcloud.upc.edu:36302");
});
```

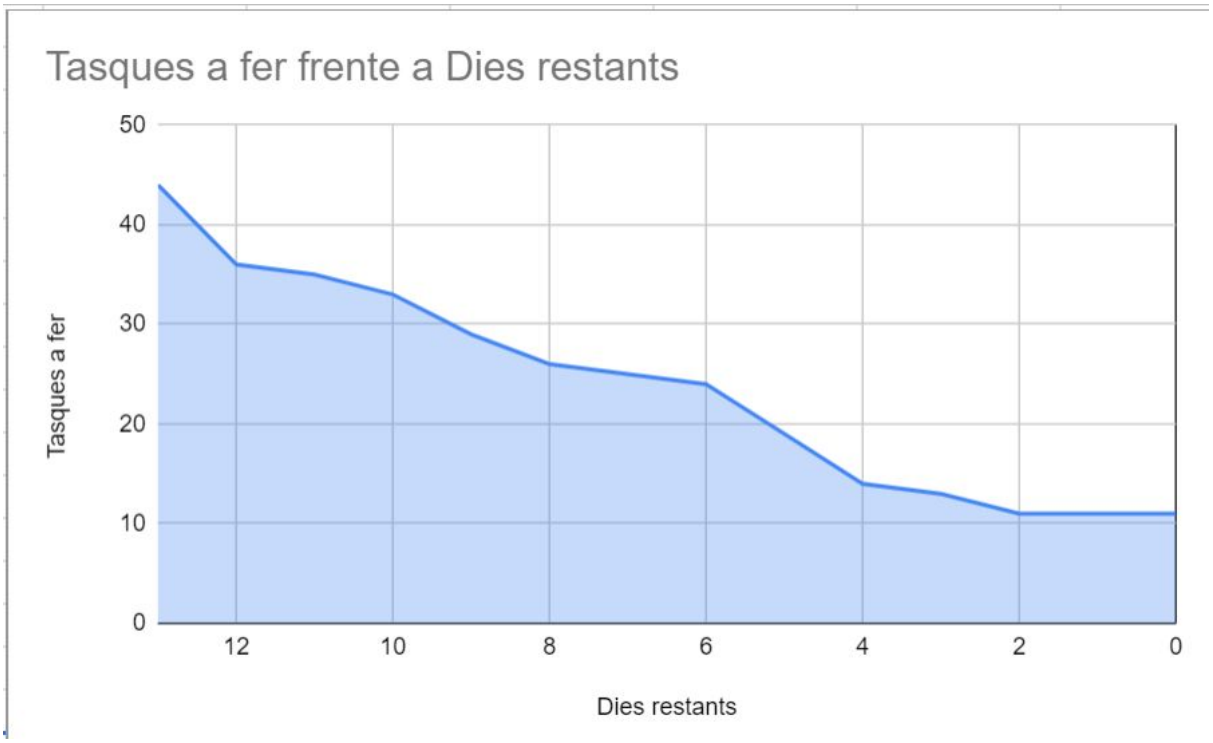
Ara ja podem provar el seu correcte funcionament pero abans tenim hem d'instal·lar els programes node, nodejs, npm i express. Comencem fent un update i després ens instal·lem els programes:

```
ptin@edge:~/Edge$ sudo apt-get update
ptin@edge:~/Edge$ sudo apt-get install nodejs
ptin@edge:~/Edge$ sudo npm install -g express
```

Llavors executem la comanda $ docker-compose build i la comanda $ docker-compose up -d

```
ptin@edge:~/Edge$ sudo docker-compose build
ptin@edge:~/Edge$ sudo docker-compose up -d
```

Burndown



Webgrafia

https://github.com/resyst-it/docker-bind9
https://zgadzaj.com/development/docker/docker-compose/containers/mongodb
https://medium.com/faun/managing-mongodb-on-docker-with-docker-compose-26bf8a0bbae3
https://dev.to/sonyarianto/how-to-spin-mongodb-server-with-docker-and-docker-compose-2lef
https://www.digitalocean.com/community/tutorials/como-configurar-laravel-nginx-y-mysql-con-docker-compose-es