

PTI práctica 3: REST API

Joan Llonch Majó

Luis Jesús Valverde Zavaleta

Entorno y configuración de la práctica

El entorno que hemos utilizado para la realización de la práctica es una imagen proporcionada la UPC que es la Ubuntu22v3r1 y que la hemos corrido en la maquina virtual VirtualBox.

Para configurar la maquina y tenerla preparada para la realización de la práctica, seguimos los comandos especificados en el enunciado de la asignatura. Las herramientas principales con las que tuvimos que trabajar fueron Node.js y cURL, pero no venían por defecto en la maquina virtual, así que tuvimos que instalarlas siguiendo los siguientes comandos:

- `sudo apt-get install -y curl` (Per instalar el cURL)
- `sudo apt-get install -y nodejs` (Per instalar Node.js)

Después preparamos el directorio donde íbamos a realizar la parte más importante de la práctica:

- `mkdir myapp`
- `cd app`
- `npm init` (Comando para crear un fichero package.json en el que registrarás las dependencias de la aplicación)
- `npm install express --save` (Comando para instalar Express en el directorio myapp y guardarlo en la lista de dependencias)
- `npm list` (Comando para revisar la versión de Express)

Familiarización con las herramientas

Después de tener todas las herramientas disponibles y listar para ser utilizadas hicimos unas pruebas para familiarizarnos con ellas donde destacamos:

- Programar un servidor web básico con Node.js, donde creamos un fichero llamado `server.js` y que ejecutamos con el siguiente comando que sería muy importante después: `"node server.js &"`. Después cuando accedíamos a la url <http://localhost:8080> nos salía un mensaje, luego en la realización de la práctica lo haríamos más complejo.
- Después nos mostraron diferentes formas de enrutamiento URL que también las aplicaríamos después como el enrutamiento a la dirección <http://localhost:8080/students>
- También nos mostraron como crear rutas URL con parámetros por encima, pero no profundizamos mucho este concepto en la práctica.
- Otro bloque importante que nos mostraron como ejemplo fue como trabajar con JSON y dos de sus acciones que son:
 - JSON response donde picamos código poniendo diferentes atributos de estudiantes con sus respectivos atributos y mostrar estos a través de una url.
 - JSON request nos enseñó como hace endpoints POST y así poder hacer registro de nuevos estudiantes con sus respectivos atributos.

Realización de la práctica

El enunciado de la práctica nos pedía que implementáramos un sistema de alquiler de coches donde básicamente implantábamos dos funcionalidades, este sistema estaría plasmado y diseñado en el archivo `assignment.js` que es el que después correríamos para llevar a cabo su funcionamiento.

La primera funcionalidad nos pedía que hiciéramos una petición para crear una instancia de un alquiler de coche, en la que teníamos que especificar los siguientes atributos: car maker (entendimos que era la marca del coche como Mercedes, Seat, etc), car model (El modelo del coche dentro de la marca como por ejemplo en Mercedes el AMG o el Brabus), el número de días del alquiler del coche y el número de unidades. Toda esta información de esta nueva instancia la habíamos de guardar en un archivo JSON que haría la funcionalidad de una base de datos, aunque en situaciones reales esto no funciona así. La forma en la que implementamos esta funcionalidad fue de la siguiente manera:

```
//Hace un request de una nueva renta y lo añade a la lista
app.post('/newrental', (req, res, next) => {
  //Leo JSON
  try{
    carrentalFileRawData = fs.readFileSync('carrental.json');
    carrentalJSON = JSON.parse(carrentalFileRawData);
  }
  catch{
    //Si esta vacio lo creo
    carrentalJSON = {"carrental": []};
  }
  console.log(req.body);
  //GUARDAR en JSON
  carrentalJSON["carrental"].push({"car_maker": req.body.car_maker, "car_model": req.body.car_model, "number_of_days": req.body.number_o
f_days, "number_of_units": req.body.number_of_units});
  fs.writeFileSync("carrental.json", JSON.stringify(carrentalJSON));
  res.status(201);
  res.end();
})
```

En el código mostrado en la imagen de encima, el principal obstáculo que encontramos es que si se solicita el primer alquiler de un coche, el archivo json donde almacenamos los alquileres de coches no existe y esto se debe de comprobar, por eso la forma en que lo implementamos fue utilizando la estructura `try{}catch{}`, donde dentro del `try` leeríamos el fichero `carrental.json` y en caso de que no existiera capturaríamos la excepción en el `catch` donde nos encargaríamos de inicializarlo. Después de esto simplemente enchufaríamos la nueva instancia del alquiler en el fichero json. Para entregar una petición JSON que añada una nueva instancia al fichero json ejecutamos el comando `cURL` en el terminal y en la siguiente imagen podemos ver el orden de la ejecución de los comandos y como se escriben estos en nuestro `carrental.json`

```
alunne@nipigon:~/Desktop/carrental/nyapp$ curl -i -H "Content-Type: application/json" -d '{"car_maker":"Lamborghini", "car_model":"Urus", "num
ber_of_days": "3", "number_of_units": "42"}' http://localhost:8080/newrental
{ car_maker: 'Lamborghini',
  car_model: 'Urus',
  number_of_days: '3',
  number_of_units: '42' }
HTTP/1.1 201 Created
X-Powered-By: Express
Date: Wed, 28 Feb 2024 23:35:35 GMT
Connection: keep-alive
Content-Length: 0

alunne@nipigon:~/Desktop/carrental/nyapp$ curl -i -H "Content-Type: application/json" -d '{"car_maker":"Mercedes", "car_model":"Brabus", "numb
er_of_days": "69", "number_of_units": "1"}' http://localhost:8080/newrental
{ car_maker: 'Mercedes',
  car_model: 'Brabus',
  number_of_days: '69',
  number_of_units: '1' }
HTTP/1.1 201 Created
X-Powered-By: Express
Date: Wed, 28 Feb 2024 23:36:21 GMT
Connection: keep-alive
Content-Length: 0

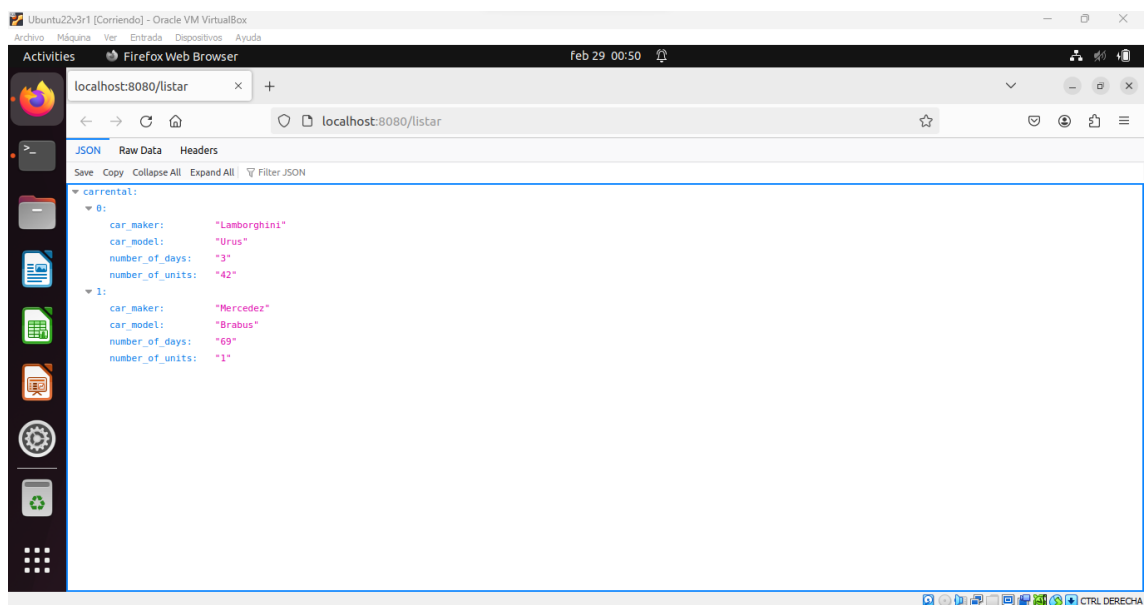
alunne@nipigon:~/Desktop/carrental/nyapp$ ls
assignment.js  carrental.json  Dockerfile  node_modules  package.json  package-lock.json
alunne@nipigon:~/Desktop/carrental/nyapp$ cat carrental.json
{"carrental":[{"car_maker":"Lamborghini","car_model":"Urus","number_of_days":"3","number_of_units":"42"}, {"car_maker":"Mercedes","car_model":"
Brabus","number_of_days":"69","number_of_units":"1"}]}
```

La segunda funcionalidad que teníamos que implementar era la de leer nuestra “teórica BD” y listar todas las instancias de alquiler de coches que había. No tuvimos problemas para la implementación de este método ya que la vimos muy simple, en la siguiente imagen se ve la implementación que hicimos.

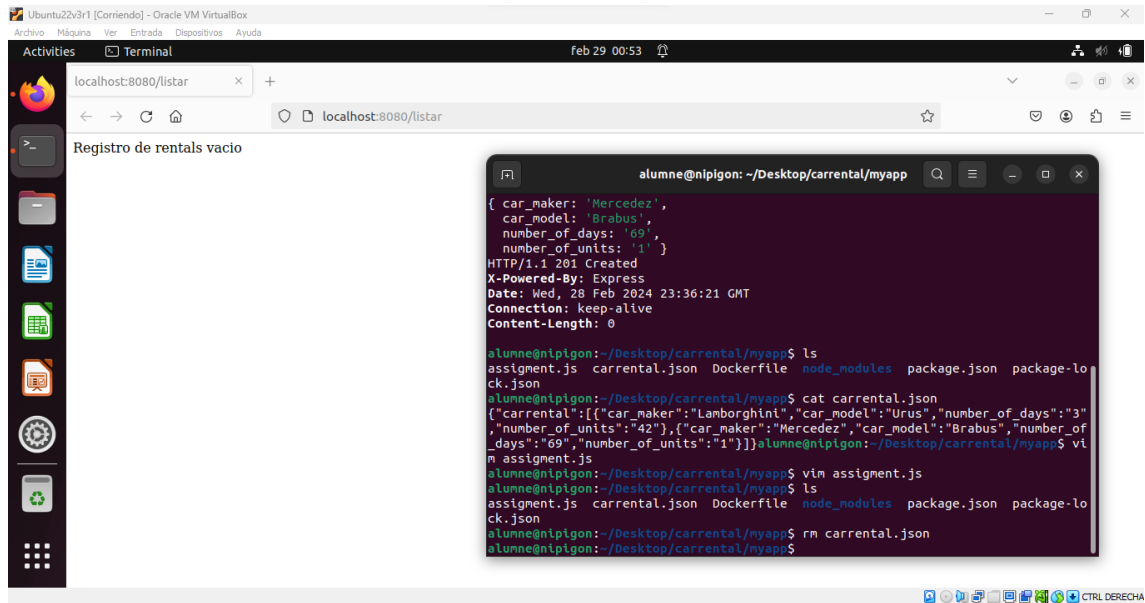
```
//LEE el JSON y lo printa en la url http://localhost:8080/listar
app.get('/listar', (req, res, next) => {
  //LEER DEL JSON
  try{
    carrentalFileRawData = fs.readFileSync('carrental.json');
    carrentalJSON = JSON.parse(carrentalFileRawData);
    res.json(
      carrentalJSON
    );
  }catch{
    res.send("Registro de rentals vacio");
  }

  res.status(200);
  res.end();
})
```

Tratamos del mismo modo que la primera funcionalidad el tema de la existencia previa del fichero carrental.json, utilizando la estructura try{}catch{} y que en el caso que no exista muestre un mensaje de que no hay ningún registro de alquiler, y en el caso contrario mostrar en la url <http://localhost:8080/listar> todas las instancias de alquileres de coches con sus respectivos parámetros.



En la imagen podemos observar las dos instancias de alquiler creadas anteriormente y ahora mostraremos el caso en el que no existe ninguna instancia, borrando el archivo `carrental.json`.



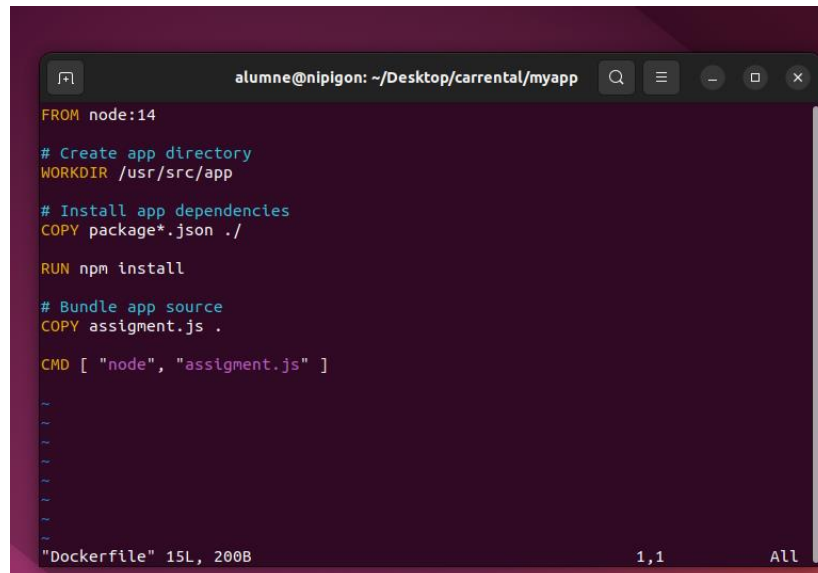
En conclusión, así es la forma en la que implementamos ambos métodos y la forma en el que gestionábamos los posibles casos que pudiera haber en el sistema.

Extension 1: Dockerize your web application

Lo primero que hicimos antes de ponernos manos a la obra con esta parte, fue instalar Docker ya que no estaba presente por defecto en la imagen proporcionada por la universidad, para su instalación seguimos los siguientes comandos:

- `sudo apt-get update`
- `wget -qO- https://get.docker.com/ | sh`
- `sudo usermod -aG docker $(whoami)`
- `newgrp docker`

Una vez realizado esto hicimos unas pruebas para familiarizarnos con Docker. Una vez ya sabíamos como llevar a cabo esta parte del enunciado, nos pusimos manos a la obra. El primer paso que hicimos fue crear un archivo Docker y añadir unas líneas dadas por el enunciado donde simplemente cambiamos server.js por assignment.js, en la siguiente imagen mostramos el fichero y su contenido.



```
FROM node:14

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
COPY package*.json ./

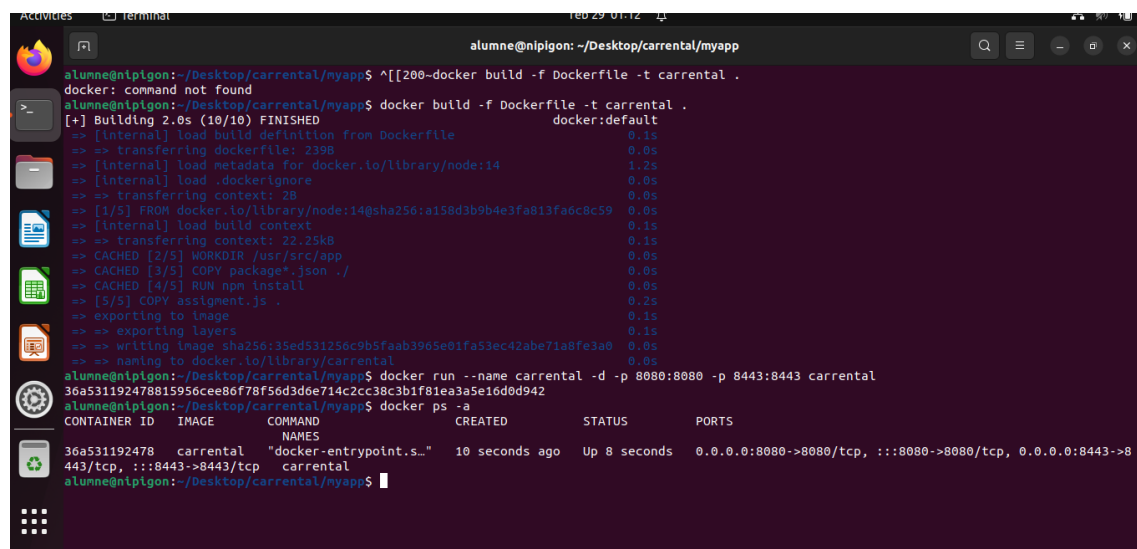
RUN npm install

# Bundle app source
COPY assignment.js .

CMD [ "node", "assignment.js" ]
```

"Dockerfile" 15L, 200B 1,1 All

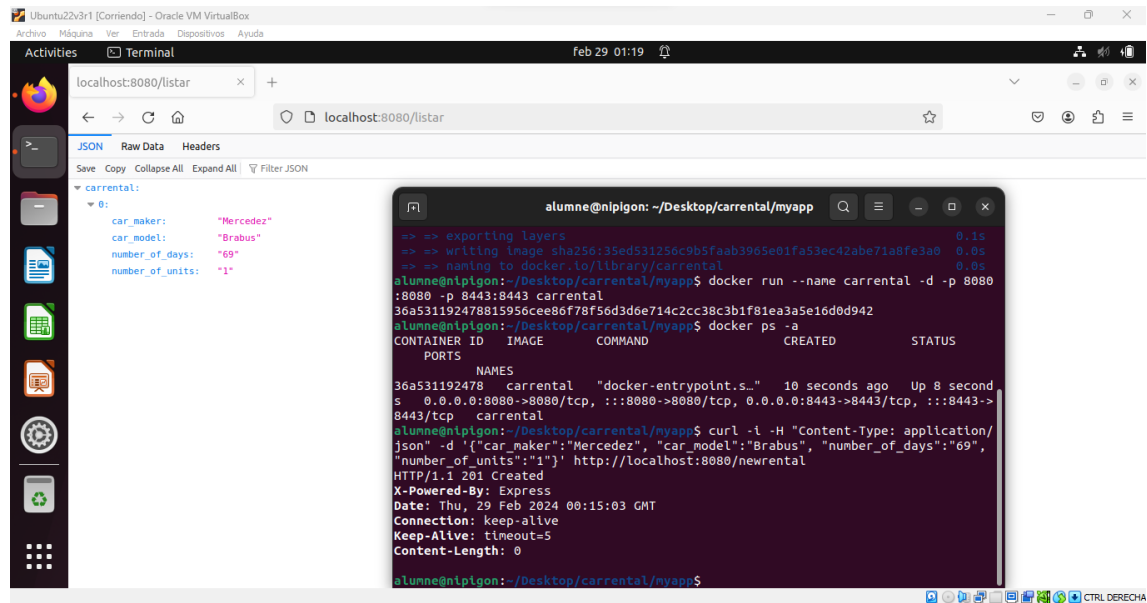
Después de esto, nos encargamos de crear la imagen con el siguiente comando : “docker build -f Dockerfile -t carrental .”, una vez creada la imagen pusimos en marcha el contenedor con este comando: “docker run --name carrental -d -p 8080:8080 -p 8443:8443 carrental”



```
alumno@nipigon: ~/Desktop/carrental/myapp
alumno@nipigon:~/Desktop/carrental/myapp$ docker build -f Dockerfile -t carrental .
[+] Building 2.0s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 239B
=> [internal] load metadata for docker.io/library/node:14
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c59
=> [internal] load build context
=> => transferring context: 22.25kB
=> CACHED [2/5] WORKDIR /usr/src/app
=> CACHED [3/5] COPY package*.json ./
=> CACHED [4/5] RUN npm install
=> [5/5] COPY assignment.js .
=> exporting image
=> => exporting layers
=> => writing image sha256:35ed531256c9b5faab3965e01fa53ec42abe71a8fe3a0
=> => naming to docker.io/library/carrental

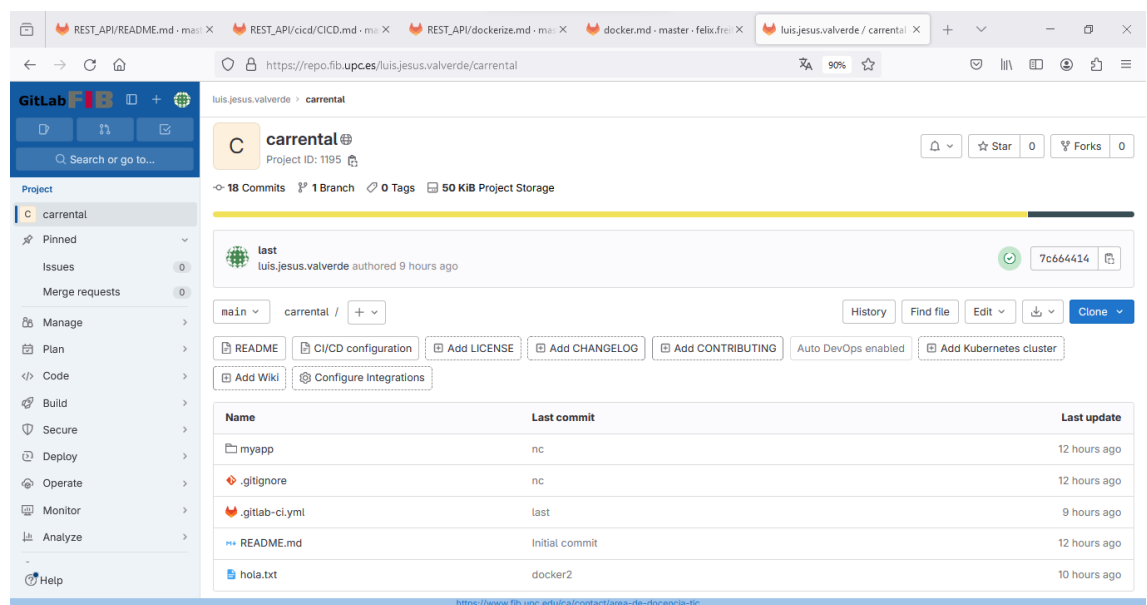
alumno@nipigon:~/Desktop/carrental/myapp$ docker run --name carrental -d -p 8080:8080 -p 8443:8443 carrental
36a531192478815956cee86f78f56d3d6e714c2cc38c3b1f81ea3a5e16d6d942
alumno@nipigon:~/Desktop/carrental/myapp$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
36a531192478   carrental  "docker-entrypoint.s..." 10 seconds ago Up 8 seconds  0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 0.0.0.0:8443->8443/tcp
alumno@nipigon:~/Desktop/carrental/myapp$
```

Una vez hecho todo esto, el sistema funcionaba a la perfección como en el apartado anterior, así que para su comprobación hice una petición para añadir una nueva instancia de alquiler del coche y visité la url donde me listaba todas las instancias y allí apareció.



Extension 2: CI/CD

Lo primero que hicimos fue iniciar sesión en repo.fib.upc.es y aquí creamos un repositorio en blanco, que después clonaríamos con el siguiente comando: “git clone <https://repo.fib.upc.es/luis.jesus.valverde/carrental.git>”, cuando lo teníamos clonado, lo que hicimos fue hacer una copia del directorio myapp y ponerlo en el directorio donde clonamos el repo, después creamos un fichero .gitignore en el que pusimos node_modules/. Una vez realizado todo esto ejecutamos los siguientes comandos, “git add .”, “git commit -m “nc””, y “git push”.



Después de hacer todo lo anterior, nos toca realizar la parte de instalar el GitLab Runner, para realizar esto seguimos una serie de comandos especificados en la pagina web del gitlab, que son los siguientes:

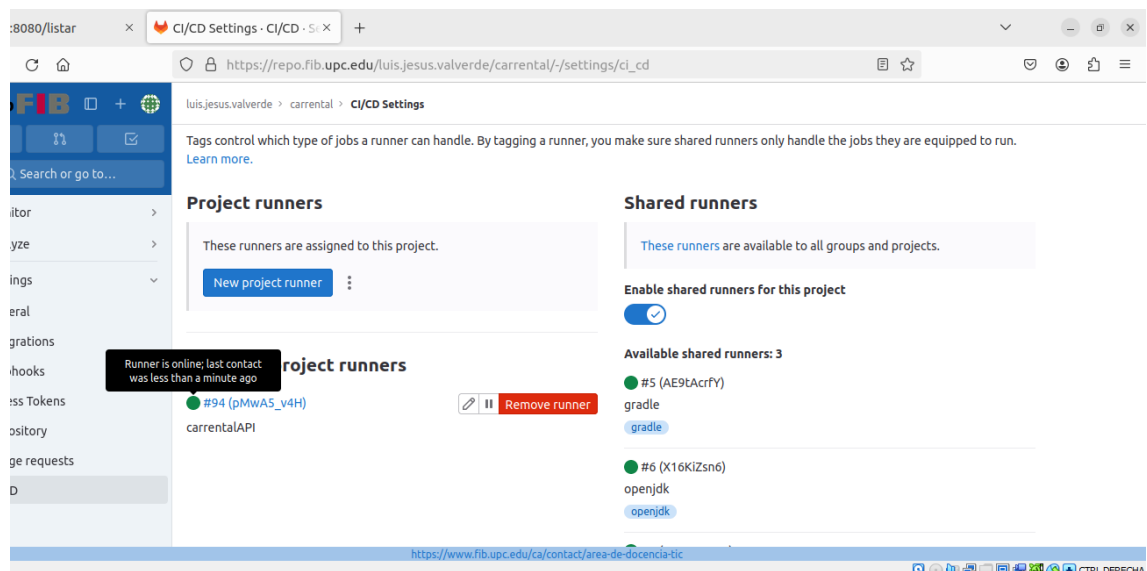
```
# Download the binary for your system
sudo curl -L --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64

# Give it permission to execute
sudo chmod +x /usr/local/bin/gitlab-runner

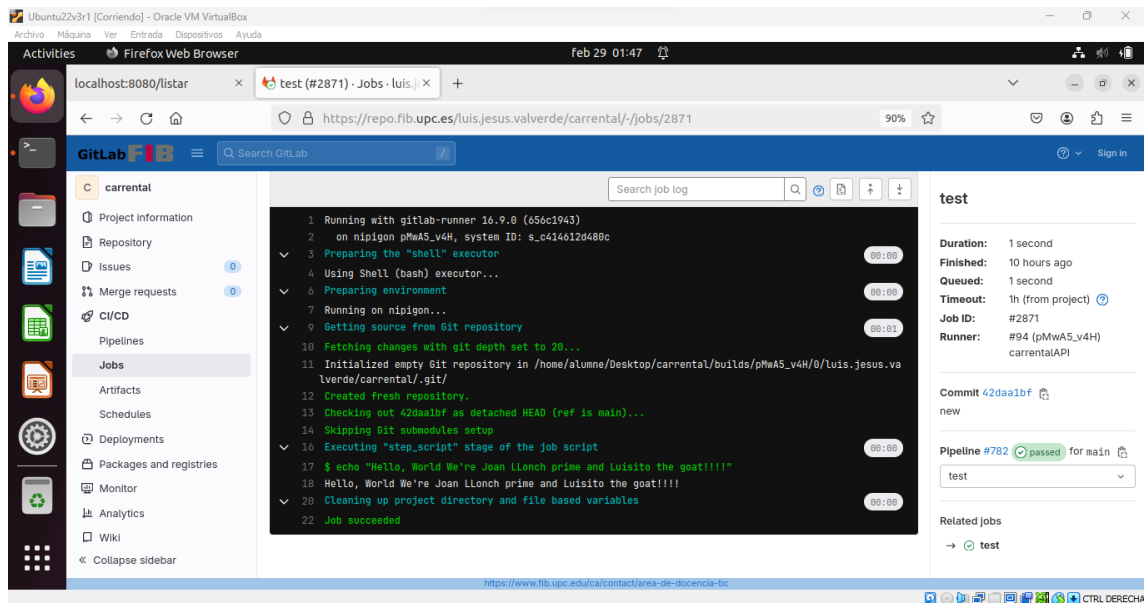
# Create a GitLab Runner user
sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash

# Install and run as a service
sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner
sudo gitlab-runner start
```

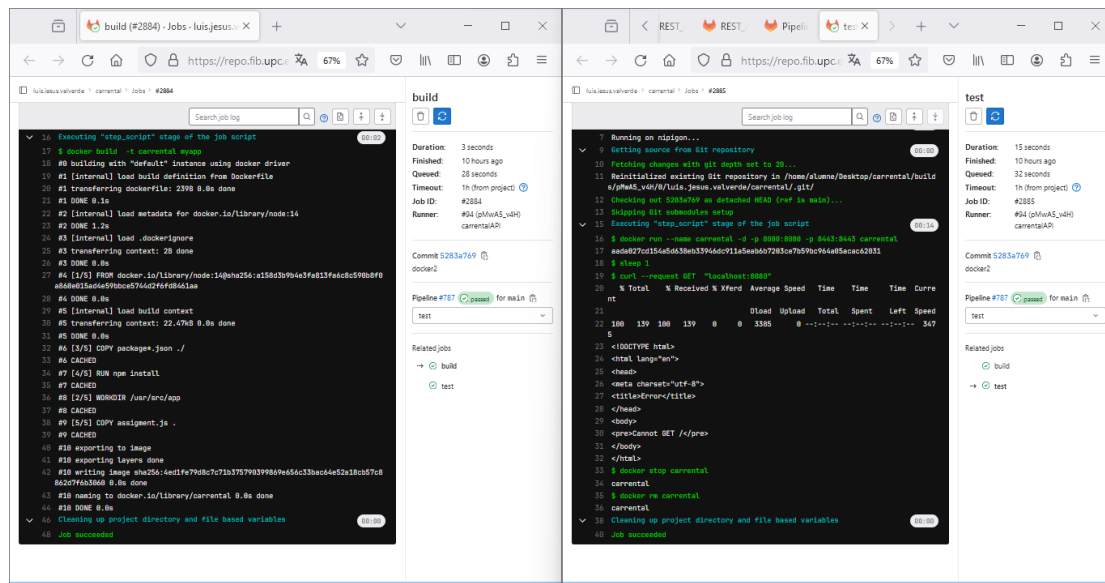
Una vez teníamos ya instalado el GitLabRunner, nos pusimos con la parte de registrar un runner, para realizar esto, lo hicimos todo mediante la pagina web de gitlab siguiendo todos los pasos que nos especificaban. Después de hacer todos los pasos ya teníamos el runner en nuestra posesión como se puede observar en la siguiente imagen.



La última parte era la realización de pipelines, en la que nos costo un poco hacerlo porque estábamos un poco perdidos por la pagina web y no entendíamos como poder visualizar los resultados. Básicamente el primer pipeline que realizamos fue uno muy simple donde mostrábamos un mensaje que fue “Hello, World We're Joan Llonch prime and Luisito the goat!!!!”, para poder hacerlo teníamos que rellenar el fichero .gitlab-ci.yml con unas líneas de comandos muy simples. Una vez hecho esto y mientras corria nuestra runne, ejecutamos los comandos de “git add .”, “git commit -m “nc” ”, y “git push” y así obtuvimos la pipeline.



La segunda pipeline que realizamos fue una un poco más compleja que la anterior, en la que nos encargamos de modificar el fichero .gitlab-ci.yml poniendo el contenido que nos proporcionaba el enunciado, que este consistía en que a partir de la imagen que teníamos creada, construir un contendor y ponerlo en marcha. Hicimos dos versiones, ya que la proporcionada por el enunciado al final ejecutaba dos comandos que eliminaban el contenedor y decidimos quitarlas, y la otra la seguimos al pie de la letra como marcaba la práctica. La siguiente imagen es respecto a la versión especificada en la práctica.



La última parte de la práctica nos pedía modificar otra vez el fichero `.gitlab-ci.yml` según lo que nos proporcionaba el enunciado. Esta nueva modificación consistía en acceder a la imagen que estaba guardada en un registro. Para guardar la imagen en el registro ejecutamos el siguiente comando “`docker run -d -p 5000:5000 --restart=always --name registry registry:2`”, con la imagen en el registro el trozo nuevo añadido a `.gitlab-ci.yml` era para cargar desde el registro la imagen y fue un éxito. En las siguientes imágenes se puede comprobar.

last

passed
carrental created pipeline for commit `7c664414` finished 10 hours ago

For `main`

latest
0 3 Jobs
0 2 minutes 38 seconds, queued for 16 seconds

Pipeline

Needs

Jobs 3

Tests 0

test

passed
build

passed
release-image

passed
test

Search job log

30 #5 transferring context: 22.4/kB done

31 #5 DONE 0.0s

32 #6 [4/5] RUN npm install

33 #6 CACHED

34 #7 [2/5] WORKDIR /usr/src/app

35 #7 CACHED

36 #8 [3/5] COPY package*.json ./

37 #8 CACHED

38 #9 [5/5] COPY assignment.js .

39 #9 CACHED

40 #10 exporting to image

41 #10 exporting layers done

42 #10 writing image sha256:4ed1fe79d8c7c71b375798399869e656c33bac64e52a18cb57c8862d7f6b3060 done

43 #10 naming to docker.io/library/carrental 0.0s done

44 #10 DONE 0.0s

46 Cleaning up project directory and file based variables

48 Job succeeded

00:00

Duration:

2 seconds

Finished:

10 hours ago

Queued:

15 seconds

Timeout:

1h (from project)

Job ID:

#2893

Runner:

#94 (pMwA5_v4H)
carrentalAPI

Commit 7c664414

last

Pipeline #791

passed

for main

test

Related jobs

→ build

release-image

test

Search job log

50 f8a91dd5fc84: Pushed

51 c45660adde37: Pushed

52 b2dba7477754: Pushed

53 e01a454893a9: Pushed

54 latest: digest: sha256:5e699af9edda90004deeb33e1109449b125857bc69e198f7e48c1aa6eb85738a size: 3048

55 \$ docker image remove localhost:5000/carrental

56 Untagged: localhost:5000/carrental:latest

57 Untagged: localhost:5000/carrental@sha256:5e699af9edda90004deeb33e1109449b125857bc69e198f7e48c1aa6eb85738a

58 \$ docker pull localhost:5000/carrental

59 Using default tag: latest

60 latest: Pulling from carrental

61 Digest: sha256:5e699af9edda90004deeb33e1109449b125857bc69e198f7e48c1aa6eb85738a

62 Status: Downloaded newer image for localhost:5000/carrental:latest

63 localhost:5000/carrental:latest

65 Cleaning up project directory and file based variables

67 Job succeeded

00:00

Duration:

2 minutes 19 seconds

Finished:

10 hours ago

Queued:

34 seconds

Timeout:

1h (from project)

Job ID:

#2895

Runner:

#94 (pMwA5_v4H)
carrentalAPI

Commit 7c664414

last

Pipeline #791

passed

for main

test

Related jobs

→ release-image

test

Search job log

22 100 139 100 139 0 0 4792 0 --:--:-- --:--:-- --:--:-- 4964

23 <!DOCTYPE html>

24 <html lang="en">

25 <head>

26 <meta charset="utf-8">

27 <title>Error</title>

28 </head>

29 <body>

30 <pre>Cannot GET /</pre>

31 </body>

32 </html>

33 \$ docker stop carrental

34 carrental

35 \$ docker rm carrental

36 carrental

38 Cleaning up project directory and file based variables

40 Job succeeded

00:00

Duration:

15 seconds

Finished:

10 hours ago

Queued:

18 seconds

Timeout:

1h (from project)

Job ID:

#2894

Runner:

#94 (pMwA5_v4H)
carrentalAPI

Commit 7c664414

last

Pipeline #791

passed

for main

test

Related jobs

→ test