

DATU MASIBOEN PROZESAMENDURAKO AZPIEGITURAK

1.LANA

Andoni Sudupe eta Oier Ijurko

Aurkibidea

1.	Sarrera.....	3
2.	Garatu beharreko fitxategiak.....	4
2.1.	compose.yaml.....	4
2.1.1.	Flask.....	4
2.1.2.	Nginx.....	4
2.1.3.	Adminer.....	4
2.1.4.	Mariadb.....	4
2.2.	proba.sql.....	5
2.3.	nginx.conf.....	5
2.4.	Dockerfile.....	5
2.4.1.	Requirements.txt.....	6
2.5.	app.py.....	6
2.5.1.	/db.....	6
2.5.2.	/.....	7
2.5.2.1.	message (post/get).....	7
2.5.2.2.	test (post/get).....	7
3.	Hobekuntzak.....	8
3.1.	Errore kudeaketa.....	8
3.2.	GET /message eskaeren filtraketa.....	8
4.	Emaizak.....	8
5.	Erreferentziak.....	10

1. Sarrera

Burututako praktika honetan, nodo artean lana banatzen duten klusterrak inplementatuko ditugu. Hau gauzatzeko, honako azpiegitura duen ingurune bat hedatuko dugu: Nginx kontainer bat, Flask aplikazioa inplementatzen duten kontainer multzo batera eta adminer kontainer batera berbidaltzen duena. Adminer eta Flask kontainer hauek, mariadb datubasea duen beste kontainer batera konektatuta egongo dira ere.

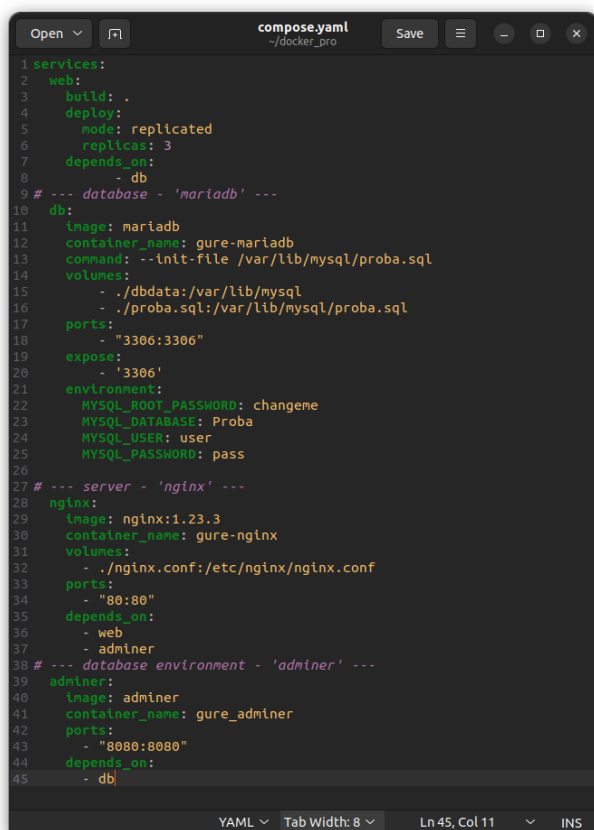
Azpiegitura hau Nginx-eko kontainerrean jasotako eskaerak toki desberdinetara berbidaltzeko gai izango da. Aukeren artean, adminer kontainerrean joan, uneko kontainerrearen identifikatzailea lortu, datu basean informazioa sartu eta datu-basea bistaratzea daude.

Dockerren oinarritutako beste edozein lanetan egin behar den bezala, gure compose.yaml, nginx.conf, Dockerfile eta app.py fitxategiak inplementatu beharko ditugu ataza gauzatu ahal izateko.

2. Garatu beharreko fitxategiak

2.1 compose.yaml

Compose fitxategia YAML fitxategi bat da, eta Docker aplikazio baterako zerbitzuak, sareak eta bolumenak definitzen ditu. Gure fitxategiak ondorengo zerbitzuak izango ditu:



```
1 services:
2   web:
3     build: .
4     deploy:
5       mode: replicated
6       replicas: 3
7     depends_on:
8       - db
9   # --- database - 'mariadb' ---
10  db:
11    image: mariadb
12    container_name: gure-mariadb
13    command: --init-file /var/lib/mysql/proba.sql
14    volumes:
15      - ./dbdata:/var/lib/mysql
16      - ./proba.sql:/var/lib/mysql/proba.sql
17    ports:
18      - "3306:3306"
19    expose:
20      - "3306"
21    environment:
22      MYSQL_ROOT_PASSWORD: changeme
23      MYSQL_DATABASE: Proba
24      MYSQL_USER: user
25      MYSQL_PASSWORD: pass
26
27 # --- server - 'nginx' ---
28 nginx:
29   image: nginx:1.23.3
30   container_name: gure-nginx
31   volumes:
32     - ./nginx.conf:/etc/nginx/nginx.conf
33   ports:
34     - "80:80"
35   depends_on:
36     - web
37     - adminer
38 # --- database environment - 'adminer' ---
39 adminer:
40   image: adminer
41   container_name: gure_adminer
42   ports:
43     - "8080:8080"
44   depends_on:
45     - db
```

2.1.1. Flask

“web” bezala izendatu dugu honako zerbitzua. Build, deploy eta depends_on azpi atalak ere jarri dizkiogu. Zerbitzu honek datu-basearekiko duen dependentzia azaldu eta 3 replika izateko beharrezkoak diren atalak ere zehaztu ditugu.

2.1.2. Nginx

Nginx-eko zerbitzua inplementatzean, bere irudia, kontainerraren izena, .conf-aren kokapena, portua eta dependentziak zehaztu behar izan ditugu. Irudian ikusten den bezala, 80 portua esleitu diogu eta Flask eta Adminer kontainerrekiko dependentzia duela zehaztu dugu.

2.1.3. Adminer

Zerbitzu hau zehazteko, adminer-en irudia, honen izena, bere portua eta dependentzia adierazi ditugu. 8080 portua esleituz eta

datu-basearekiko dependentzia adieraziz inplementatu dugu zerbitzu hau.

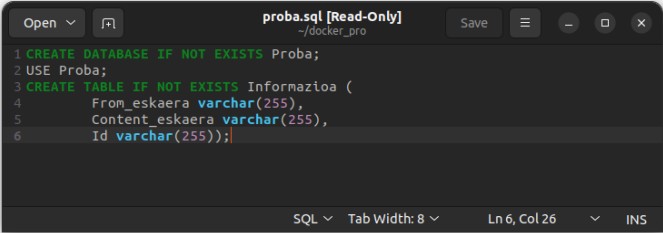
2.1.4. Mariadb

Datu-basearen zerbitzua izan da inplementatzeko zailena. Azpiatalen artean, Mariadb-ren irudia, datu-basearen izena, exekutatu behar den .sql fitxategia, bolumenak, portuak eta ingurumena daude. Mariadb-ren azken bertsioa erabili dugu, eta kontainerrari “gure-mariadb” izena ezarri diogu. Ondoren azalduko den .sql fitxategia “docker compose up” egiten denean exekutatu behar dela ere adierazi dugu. Gure fitxategiak non dauden eta kontainerrean non kokatuko diren “volumes”-en adierazi dugu. Portuei dagokionez, 3306 portu estandarra esleitu dugu eta portu hau irisgarria izateko “expose” atala gehitu behar izan dugu. Azkenengo atala “environment” da. Bertan, erabiltzaile bat, pasahitz bat eta datu-basearen izena adierazi behar izan dugu. Beste zerbitzuetan ez bezala, zerbitzu honek dependentzirik ez duenez, ez dugu behar izan “depends_on” atalik.

2.2. proba.sql

Datu-basearen zerbitzua definitu dugunean compose.yaml fitxategian, “docker compose up” egiterakoan exekutatu den .sql fitxategi bat esleitu dugu “command” atalean. Fitxategi honen bitartez, “Proba” izeneko datu-base bat sortuko dugu oraindik sortua izan ez bada, baita bere barruan “Informazioa” izeneko taula bat ere.

Taula honi hiru atributu esleituko dizkiogu: From_eskaera, Content_eskaera eta Id. Hirurak String-ak izango dira. Lehengoak, eskaera zenek idatzi duen gordeko du, bigarrenak mezuaren edukia izango du eta azkenengoak erabilitako flask kontainerren id-a gordeko du.



```
1 CREATE DATABASE IF NOT EXISTS Proba;
2 USE Proba;
3 CREATE TABLE IF NOT EXISTS Informazioa (
4   From_eskaera varchar(255),
5   Content_eskaera varchar(255),
6   Id varchar(255));
```

2.3. nginx.conf

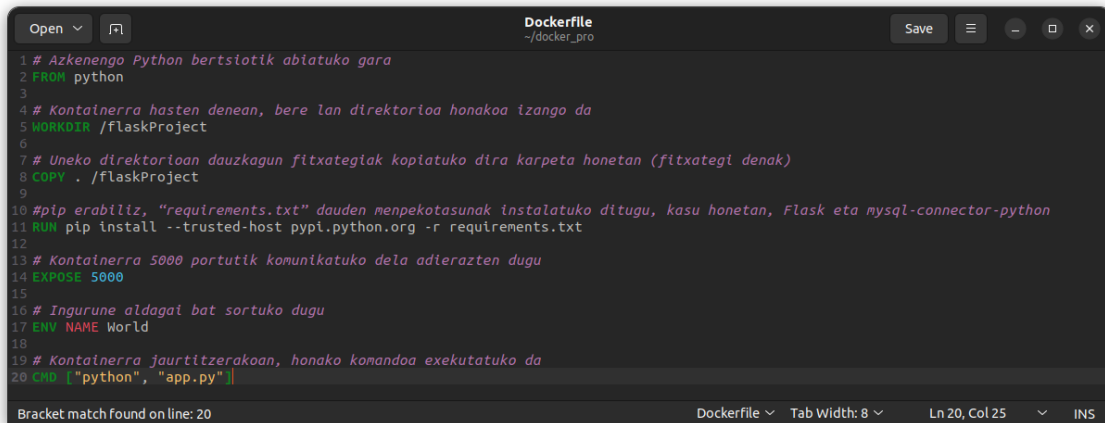


```
1 user nginx;
2 events {
3   worker_connections 1000;
4 }
5 http {
6   server {
7     listen 80;
8     location / {
9       proxy_pass http://web:5000;
10    }
11    location /db {
12      proxy_pass http://adminer:8080;
13    }
14  }
15 }
```

Konfigurazio fitxategi honen bitartez, lan-prozesu kopurua eta eskaeren zatitzea nolakoa izango den adierazi ahalko dugu. HTTP motako eskaerak jasoko ditugu 80 portuan (horregatik listen 80) eta eskaerak Flask edo Adminerrera berbidaliko ditugu kokapenaren arabera (/ edo /db).

2.4. Dockerfile

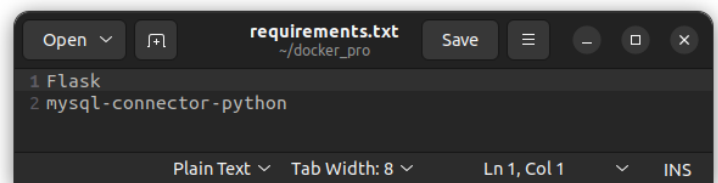
Sekuentzialki exekututzen diren instrukzio multzo bat gordetzen den fitxategia da, eta irudi publiko batetik, kasu honetan Python berrienaren irudi batetik, beste irudi bat sortzen laguntzen du. Hau exekutatzeke Docker compose build egiten da, eta bertako instrukzioak jarraituz, Docker irudi bat sortzen du.



```
1 # Azkenengo Python bertsiotik abiatuko gara
2 FROM python
3
4 # Kontainerra hasten denean, bere lan direktorioa honakoa izango da
5 WORKDIR /flaskProject
6
7 # Uneko direktorioan dauzkagun fitxategiak kopiatuko dira karpeta honetan (fitxategi denak)
8 COPY . /flaskProject
9
10 #pip erabiliz, "requirements.txt" dauden menpekotasunak instalatuko ditugu, kasu honetan, Flask eta mysql-connector-python
11 RUN pip install --trusted-host pypi.python.org -r requirements.txt
12
13 # Kontainerra 5000 portutik komunikatuko dela adierazten dugu
14 EXPOSE 5000
15
16 # Ingurune aldagai bat sortuko dugu
17 ENV NAME World
18
19 # Kontainerra jaurtitzera, honako komandoa exekutatu da
20 CMD ["python", "app.py"]
```

2.4.1. Requirements.txt

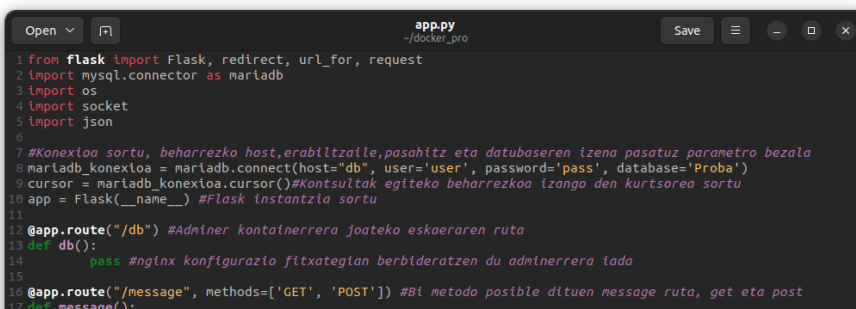
Dockerfile fitxategian instalatzen diren aplikazioak banaka RUN pip install egin beharrean, behin exekutatzen dugu eta testu fitxategi batean adieraziko dugu zein aplikazio diren instalatu behar direnak. Kasu honetan, “Flask” aplikazioa eta “mysql-connector-python” Datubasearen konektorea instalatuko ditugu.



```
1 Flask
2 mysql-connector-python
```

2.5. app.py

Kontainerrean exekutatuko den Flask aplikazioa da. Bertan, HTTP bidez egiten diren eskaerak desberdinu eta kasu bakoitzean egin behar dena exekutatzen da. Lehenik, mariadb datubasearekin konexioa ezarriko dugu behar diren parametroak jarrita, eta kontsultak egiteko beharrezkoa den kurtsorea. Flask aplikazioaren instantzia ere sortuko dugu. Ondoren, eskaerak kudeatuko ditugu eta bi eskaera nagusi desberdintzen dira:



```
1 from flask import Flask, redirect, url_for, request
2 import mysql.connector as mariadb
3 import os
4 import socket
5 import json
6
7 #Konexioa sortu, beharrezko host,erabiltzaile,pasahitz eta datubaseren izena pasatuz parametro bezala
8 mariadb_konexioa = mariadb.connect(host="db", user='user', password='pass', database='Proba')
9 cursor = mariadb_konexioa.cursor()#Kontsultak egiteko beharrezkoa izango den kurtsorea sortu
10 app = Flask(__name__) #Flask instantzia sortu
11
12 @app.route("/db") #Adminer kontainerrean joateko eskaeraren ruta
13 def db():
14     pass #nginx konfigurazio fitxategian berbideratzen du adminerrera iada
15
16 @app.route("/message", methods=['GET', 'POST']) #Bi metodo posible dituen message ruta, get eta post
17 def message():
```

2.5.1. http://ip/db

Eskaera hau egitean adminerrera berbidaliko du. Hala ere, bertan ez dugu ezer egin, izan ere nginx.conf-ean adierazten baitugu.

2.5.2. http://ip/

Eskaera hau egitean berriz, Flask kontainer replikatara bideratuko dugu eta bertan hiru eskaera desberdin egin daitezke:

2.5.2.1. message (post/get)

Bi eskaera messagekin egiten dira, POST eta GET. Lehenengoak datubasean tupla bat gordeko du, curl eskaerarekin batera pasatzen den json formatuko informazioarekin eta erabiltzen den momentuko kontainerren id-arekin.

Bigarrenengoak berriz, datubasean gordetako informazio guztia itzultzen du. Ondoren azalduko ditugun hobekuntzetan eskaera hauek nola filtratu adieraziko dugu.

```
15 @app.route("/message", methods=['GET', 'POST']) #Bi metodo posible dituen message ruta, get eta post
16 def message():
17     if request.method == 'GET': #Datu basearen erregistro guztiak bueltatu
18         try:
19             from_eskaera = request.args.get('From') #eskaeraren edukia gordetzen dugu
20             if str(from_eskaera) == "ALL":
21                 cursor.execute("SELECT * FROM Informazioa") #Beharrezko taularen gainean eskaera
22                 return cursor.fetchall()
23             else:
24                 a = "SELECT * FROM Informazioa WHERE From_eskaera = '" + str(from_eskaera) + "'" #Beharrezko taularen gainean eskaera
25                 cursor.execute(a)
26                 return cursor.fetchall() #Lortutako emaitzak bueltatzeko beharrezko komandoa
27         except Exception as e:
28             cursor.execute("SELECT * FROM Informazioa") #Beharrezko taularen gainean eskaera
29             return cursor.fetchall()
30     elif request.method == 'POST': #Eskaeraren edukia datu basean gordeko du. Json formatuko edukia
31         try:
32             from_eskaera = request.json["From"] #eskaeraren edukia gordetzen dugu
33             content_eskaera = request.json["Content"] #eskaeraren edukia gordetzen dugu
34             id = socket.gethostname() #Uneko flask kontainerren id-a lortu
35             query = ("INSERT INTO Informazioa (From_eskaera, Content_eskaera, Id)"
36                    "VALUES (%s, %s, %s)") #Gordetzeko query-a
37             cursor.execute(query, (from_eskaera, content_eskaera, id)) #Exekutatu query-a
38             mariadb.konexioa.commit() #Aldaketak burutu
39             return "Gordeta"
40         except Exception as e: #Post eskaera desegoki bat tratatzeko mezua
41             return ("Post eskaera desegokia izan da."
42                    "Itxura honetako dei batekin saiatu: curl -d {From:AAB, Content:ttt}"
43                    "-H Content Type: application/json -X POST localhost:80/message")
44
```

2.5.2.2. test (get)

Test eskaerak aukera posible bakarra izango du, GET. Eskaera honek, ALIVE testua bueltatuko du, unean erabiltzen ari garen kontainer konketuaren id-arekin batera.

```
45 @app.route("/test", methods=['GET']) #Test rutako eskaera
46 def test(): #ALIVE testua bueltatuko digu, bere flask kontainerren id-arekin batera
47     try:
48         return "ALIVE " + socket.gethostname() #buelatu alive eta kontainerren id-a
49     except Exception as e:
50         return "Test eskaera desegokia. Itxura honetako dei batekin saiatu: localhost:80/test"
```

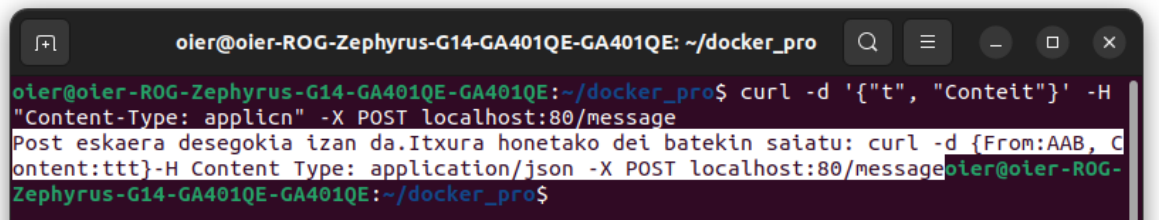
Azaldutako eskaera hauek guztiak errore kudeaketa bat izango dute ere bai. Hobekuntzetan azalduko dugu argiago hau.

3. Hobekuntzak

Hainbat hobekuntza egin ditugu lanean ere bai. Hona hemen hobekuntza desberdinen azalpen sakonagoak:

3.1. Errorreen kudeaketa

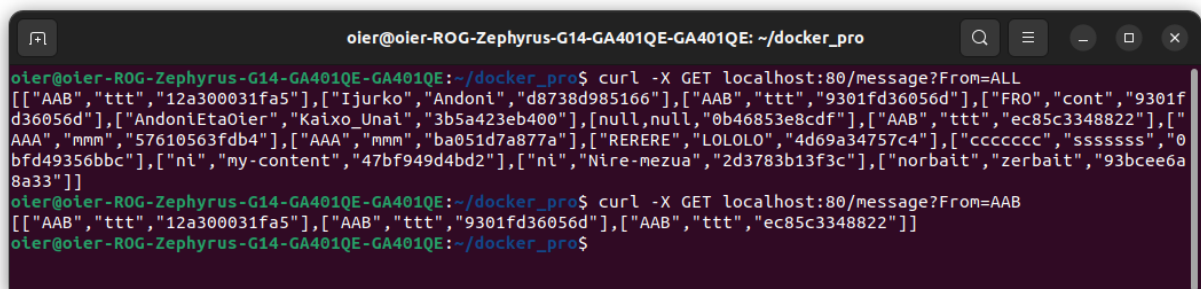
Gure programako erroreak kudeatu ahal izateko, “TRY EXCEPT” blokeak gehitu ditugu. Bloke hauen bitartez, errore bat gertatu bada eskaera bat egitean, errore mezu bat pantailaratzeko gai izango gara. Goiko argazkietan hauen implementazioa ikusgai badago ere, POST message egitean gertatzen den mezu errore bat erakutsiko dugu adibide bezala:



```
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE: ~/docker_pro
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$ curl -d '{"t": "Conteitt"}' -H "Content-Type: applicn" -X POST localhost:80/message
Post eskaera desegokia izan da.Itxura honetako dei batekin saiatu: curl -d {From:AAB, Content:ttt}-H Content Type: application/json -X POST localhost:80/messageoier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$
```

3.2. Emaizen filtraketa

Gure datubaseko informazioa eskuratzerako garaian, gure aplikazioak gorde ditugun tupla guztiak itzultzen ditu, beraz, filtratzeko aukera inplementatu dugu. Curl eskaeran “From=Norbait” jarritz gero, filtraketa bat egiten da, eta “Norbait” horrek bidalitakoak bakarrik itzultzen ditu. “ALL” jartzen bada, datubaseko tupla guztiak itzuliko ditu. Ondoren ikusten den bezalako eskaerak egitea beharrezkoa da emaitzak ondo itzuli ahal izateko:



```
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE: ~/docker_pro
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$ curl -X GET localhost:80/message?From=ALL
[[{"AAB","ttt","12a300031fa5"},{"Ijurko","Andoni","d8738d985166"},{"AAB","ttt","9301fd36056d"},{"FRO","cont","9301fd36056d"},{"AndoniEtaOier","Kaixo_Unai","3b5a423eb400"},[null,null,"0b46853e8cdf"],{"AAB","ttt","ec85c3348822"},{"AAA","mmm","57610563fdb4"},{"AAA","mmm","ba051d7a877a"},{"RERERE","LOLOLO","4d69a34757c4"},{"ccccccc","sssssss","0bfd49356bbc"},{"ni","my-content","47bf949d4bd2"},{"ni","Nire-mezua","2d3783b13f3c"},{"norbait","zerbait","93bcee6a8a33"}]]
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$ curl -X GET localhost:80/message?From=AAB
[[{"AAB","ttt","12a300031fa5"},{"AAB","ttt","9301fd36056d"},{"AAB","ttt","ec85c3348822"}]]
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$
```

4. Emaitzak

Azpiko irudietan, message POST, message GET eta test-en emaitzak ikusi ditzakegu terminalean.

GET /MESSAGE

```
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE: ~/docker_pro
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$ curl -X GET localhost:80/message?From=ALL
[["AAB","ttt","12a300031fa5"],["Ijurko","Andoni","d8738d985166"],["AAB","ttt","9301fd36056d"],["FR0","cont","9301fd36056d"],["AndoniEtaOier","Kaixo_Unai","3b5a423eb400"],[null,null,"0b46853e8cdf"],["AAB","ttt","ec85c3348822"],["AAA","mmm","57610563fdb4"],["AAA","mmm","ba051d7a877a"],["RERERE","LOLOLO","4d69a34757c4"],["cccccc","ssssss","0bfd49356bbc"],["ni","my-content","47bf949d4bd2"],["ni","Nire-mezua","2d3783b13f3c"],["norbait","zerbait","93bcee6a8a33"]]
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$ curl -X GET localhost:80/message?From=AAB
[["AAB","ttt","12a300031fa5"],["AAB","ttt","9301fd36056d"],["AAB","ttt","ec85c3348822"]]
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$
```

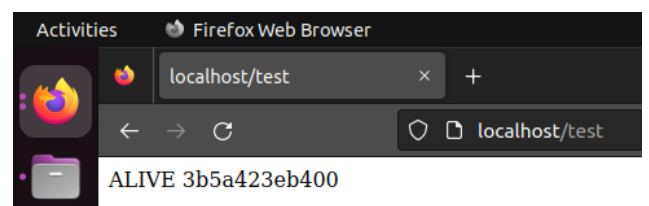
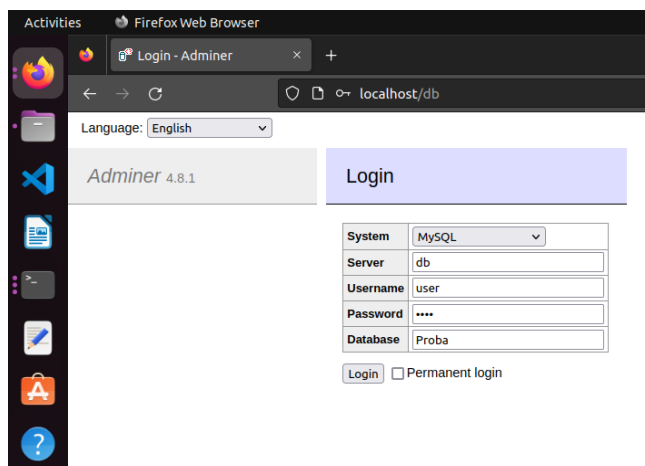
POST /MESSAGE

```
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE: ~/docker_pro
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$ curl -d '{"From":"aurkezpena2", "Content":"aurkezpen2"}' -H "Content-Type: application/json" -X POST localhost:80/message
Gordetaoier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$
```

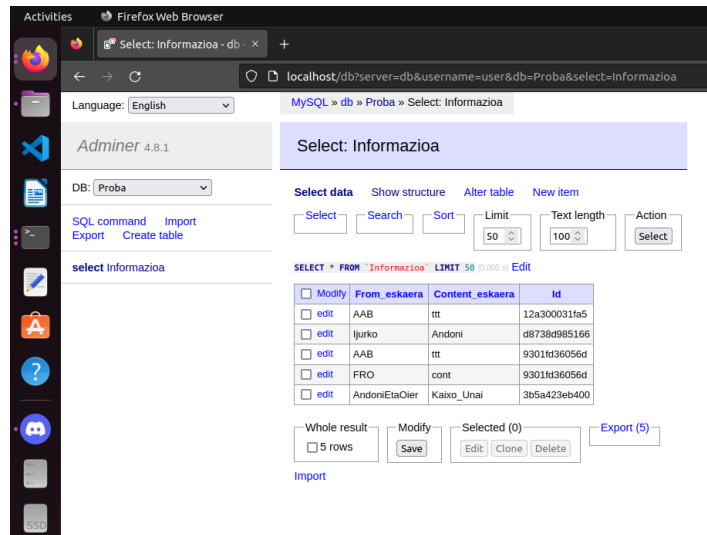
/TEST

```
oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$ curl localhost:80/test
ALIVE 1a232bfcbad3oier@oier-ROG-Zephyrus-G14-GA401QE-GA401QE:~/docker_pro$
```

Ondorengo bi irudietan nabigatzailetik egindako bi irudi ikus ditzakegu. Batean /db jarrita adminerrera bidaltzen diguela ikus daiteke, eta bestean test egin ondoren ALIVE eta kontainerraren id-a ikus daiteke.



Bukatzeko, datu-baseko informazioa taularen irudi bat ere badugu, informazio guztia behar bezala gordetzen dela ziurtatzeko:



5. Erreferentziak

- Gure proiektuaren github biltegia:
https://github.com/oierIM/DMPA_docker.git
- Route desberdinak kudeatzeko:
<https://pythonbasics.org/flask-http-methods/>
- Adminer kontainerraren dokumentazioa:
https://hub.docker.com/_/adminer/
- Mariadb kontainerraren dokumentazioa:
https://hub.docker.com/_/mariadb
- Datubasearen portua erabilgarria izateko:
<https://stackoverflow.com/questions/40801772/what-is-the-difference-between-ports-and-expose-in-docker-compose>
- Datubasearen egitura compose fitxategian:
<https://stackoverflow.com/questions/43322033/create-database-on-docker-compose-startup>