Práctica 2 AOS Grupo 10

LUCAS SANTO DOMINGO AMUTIO
SERGIO CLAVER FEITO
GEAN FRANCO RAMOS LEÓN
GABRIEL ALEJANDRO SILVA LABRADA
LUIS ZAMBRANA RODRIGO

Sobre nuestra práctica

Para esta práctica hemos integrado en nuestra práctica anterior, las partes de la API de los demás grupos, conseguidas desde los repositorios de cada uno de ellos. Hemos optado por mantener el sistema de Proxy - Back - Front que utilizamos en la práctica anterior, tanto para levantar los servicios como para hacer un Mock, con Stoplight/Prism, Caddy, y Swagger.

Cabe destacar que la práctica se ha realizado utilizando Ubuntu mediante WSL2, y todos los comandos se han ejecutado desde su línea de comandos linux.

Los enlaces relevantes son los siguientes:

Repositorio de GitHub (Los archivos relevantes a la práctica 2 estan en la sub carpeta “p2” del repositorio): https://github.com/SergioClaver/AOS-23/tree/main/p2

Siguiendo las recomendaciones de uso de Docker Hub para imágenes de docker, hemos optado por un sistema de 3 Imágenes (Front - Back - Proxy) que están almacenadas en un repositorio individual y público cada una:

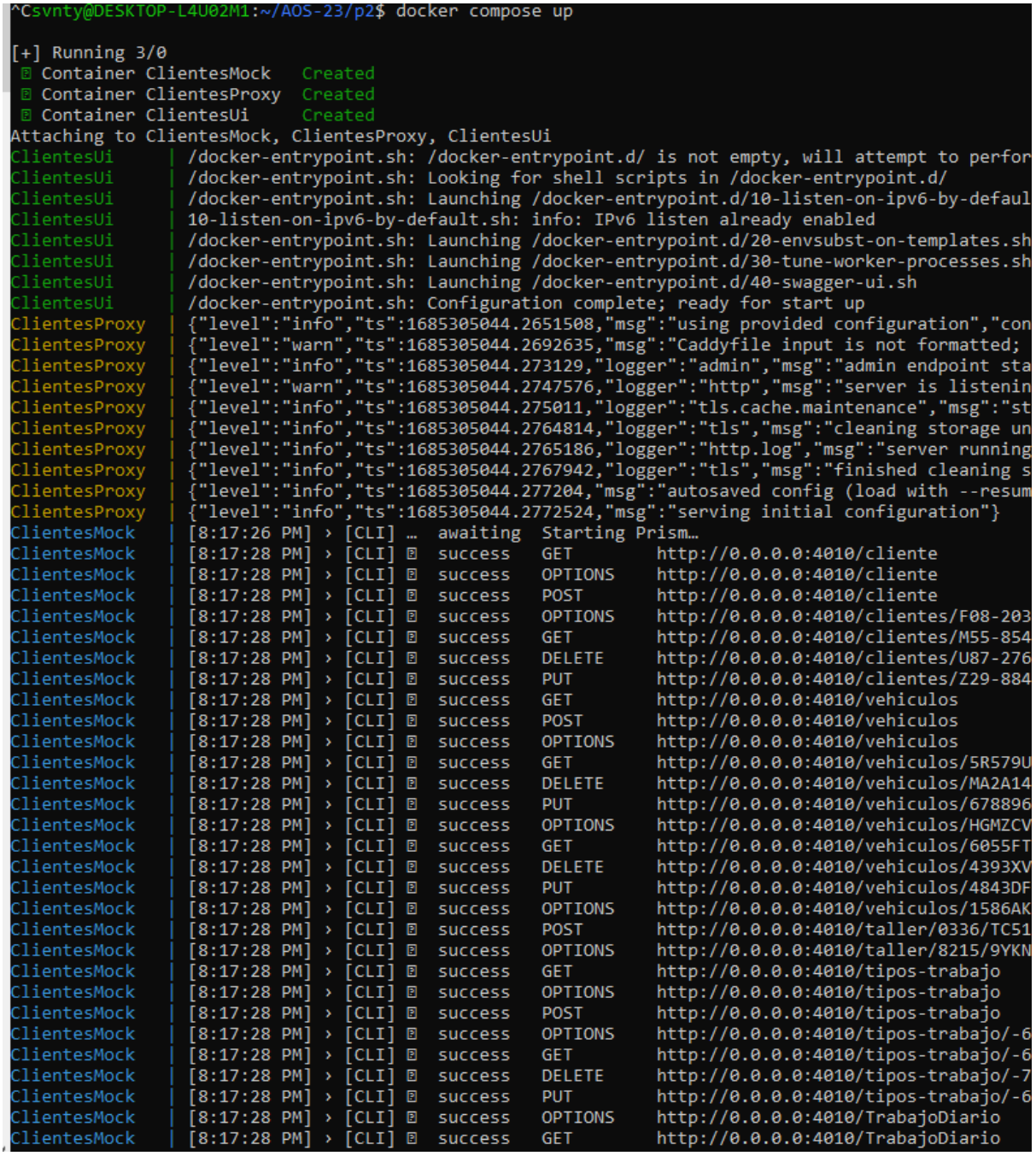
Front: https://hub.docker.com/repository/docker/svntyfour/frontaosgrupo10/general
Back: https://hub.docker.com/repository/docker/svntyfour/backaosgrupo10/general
Proxy: https://hub.docker.com/repository/docker/svntyfour/proxyaosgrupo10/general

Decisiones de diseño y funcionamiento

Hemos comenzado la práctica basándonos en el sistema de la práctica anterior, así que, usando un sistema similar, hemos añadido los archivos .yaml de los otros equipos a nuestra carpeta openapi, completando así la API del sistema de gestión de talleres de mecánica rápida. Una vez hecho esto hemos modificado el archivo openapi general para incluir los archivos y paths de los otros archivos .yaml.

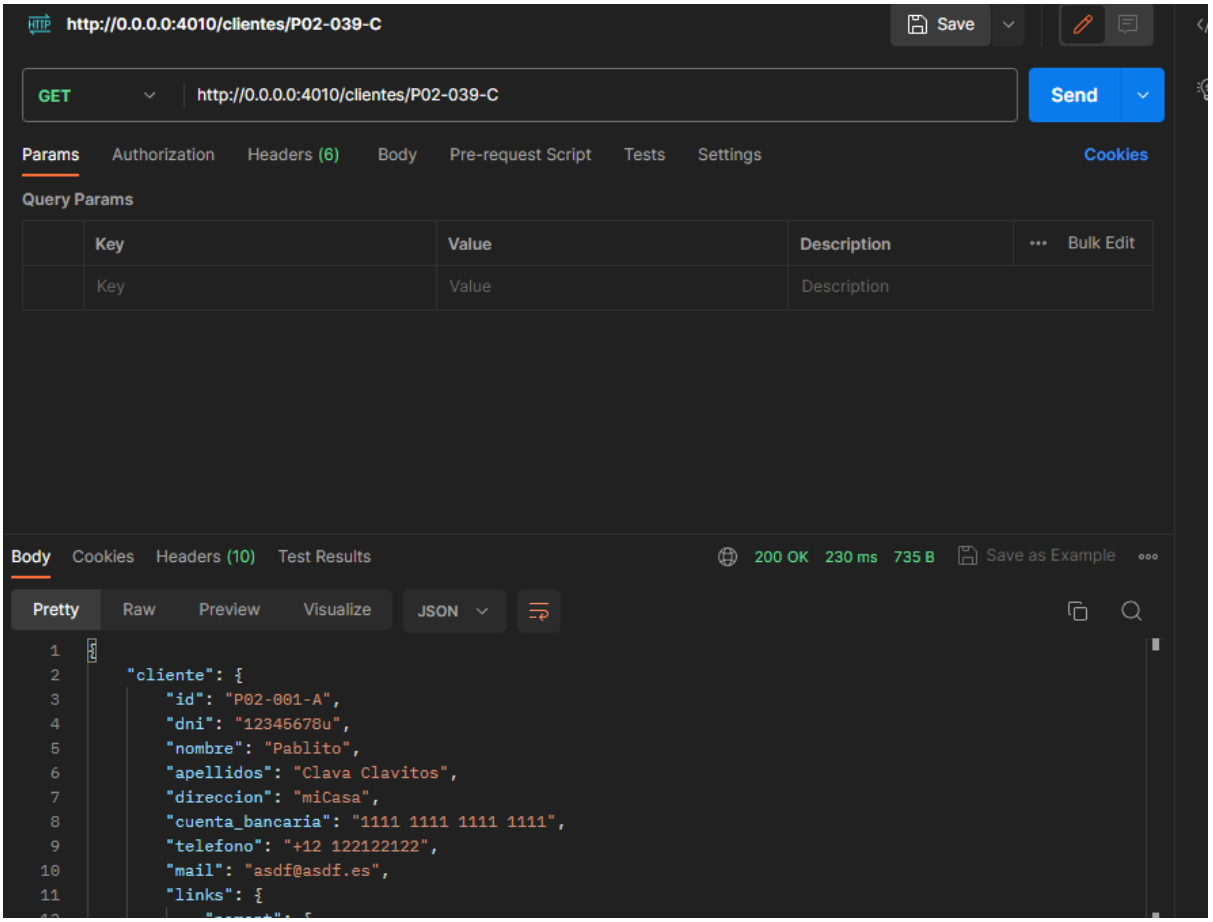
Tras esto, comprobamos que la aplicación funciona correctamente con las llamadas originales de nuestra aplicación de Clientes, y con las nuevas llamadas de los otros archivos.

Al hacer **docker compose up** vemos lo siguiente:



Se levantan los 3 servicios, y tenemos acceso a todas las llamadas http de las distintas partes del sistema de gestión. Podemos verlo de forma gráfica en swagger entrando en un navegador en la dirección http://127.0.0.1:8000.

Además podemos comprobar mediante herramientas como **Postman**, que las llamadas funcionan. Hacemos un get y recibimos una respuesta correcta:



Una vez completada y testeada esta parte, pasamos a la integración con **Kubernetes**.

El primer paso es generar imágenes **Docker** y subirlas a un repositorio **Docker Hub**. Para nuestro sistema, decidimos que la mejor forma es generar 3 imágenes independientes, que contengan de forma individual los archivos y dependencias que necesitan, y almacenarlos en repositorios separados.

Empezamos creando haciendo build y push de las imágenes que necesitamos, que en nuestro caso son 3. (**Front** - **Back** - **Proxy**)

```
svnty@DESKTOP-L4U02M1:~/AOS-23/p2/back$ docker build . -t  svntyfour/backaosgrupo10:v2
[+] Building 0.4s (8/8) FINISHED
 => [internal] load build definition from Dockerfile                                                0.1s
 => => transferring dockerfile: 108B                                                                0.0s
 => [internal] load .dockerignore                                                                   0.1s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/stoplight/prism:4                                        0.0s
 => [1/3] FROM docker.io/stoplight/prism:4                                                          0.0s
 => [internal] load build context                                                                   0.1s
 => => transferring context: 3.77kB                                                                 0.0s
 => CACHED [2/3] RUN mkdir /aos                                                                     0.0s
 => CACHED [3/3] COPY ./openapi /aos                                                                0.0s
 => exporting to image                                                                              0.2s
 => => exporting layers                                                                             0.0s
 => => writing image sha256:2e1bd443d1ab52513bb815bf9a9a5903e6a970995cf99719672a36a618ed063e        0.0s
 => => naming to docker.io/svntyfour/backaosgrupo10:v2                                              0.0s
svnty@DESKTOP-L4U02M1:~/AOS-23/p2/back$ docker push svntyfour/backaosgrupo10:v2
The push refers to repository [docker.io/svntyfour/backaosgrupo10]
d02540cf3187: Pushed
b15ba265fa17: Pushed
82da0127e89e: Mounted from stoplight/prism
5f70bf18a086: Layer already exists
eba68910279e: Mounted from stoplight/prism
10fbcb5e6404: Mounted from stoplight/prism
9ac01852af6f: Mounted from stoplight/prism
c3802b6a14f0: Mounted from stoplight/prism
604c46c08191: Mounted from stoplight/prism
88783fc1b3ab: Mounted from stoplight/prism
b6373f9dd81a: Mounted from stoplight/prism
ae4a422309cc: Mounted from stoplight/prism
b7ea1ad5bb4c: Mounted from stoplight/prism
1c97f3bff7f5: Mounted from stoplight/prism
47651630adf9: Mounted from stoplight/prism
afbbb2b3b366: Mounted from stoplight/prism
a9ef202f231b: Mounted from stoplight/prism
143dc43fef15: Mounted from stoplight/prism
4cbb5a0ee2b3: Layer already exists
3cddc64f59e2: Layer already exists
b951f8a113f5: Layer already exists
1d8bcb7a961e: Layer already exists
f1417ff83b31: Layer already exists
v2: digest: sha256:c6d13e5f0d0a350d378640051b9bc7f8b8a6327ccdd54e3215f3f9930ea4922d size: 5106
```

Una vez hecho esto, modificamos el archivo **docker-compose.yaml**, para que en vez de apuntar a los archivos locales, apunte a las imágenes y de los repositorios, asegurando además que es la versión correcta de cada imagen.

```yaml
services:
    backend:
        image: svntyfour/backaosgrupo10:v2
        container_name: ClientesMock
        command: mock --cors -h 0.0.0.0 "/aos/openapi.yaml"
        ports:
          - "4010:4010"

    frontend:
        image: svntyfour/frontaosgrupo10:v2
        container_name: ClientesUi
        ports:
            - "8000:8080"
        environment:
            SWAGGER_JSON: /aos/openapi.yaml


    proxy:
        image: svntyfour/proxyaosgrupo10:v1
        container_name: ClientesProxy
        ports:
            - "80:80"
        depends_on:
            - backend
```

En cuanto a los **Dockerfile**, se pueden observar 3 carpetas, que hacen referencia a las distintas partes del sistema, front, back y proxy, en las que cada una tiene un Dockerfile, compuesto a partir de lo especificado originalmente en el docker-compose del sistema, haciendo referencia a la imagen a pullear, comandos a ejecutar, variables de entorno, puertos expuestos… etc

Ubuntu > home > svnty > AOS-23 > p2 > front > Dockerfile

```dockerfile
FROM swaggerapi/swagger-ui:latest
ENV SWAGGER_JSON=/aos/openapi.yaml
RUN mkdir /aos
COPY ./openapi /aos
EXPOSE 8080

```

Una vez comprobado que todo el sistema sigue funcionando mediante **docker compose up**, pasamos a integrarlo con Kubernetes.

Hemos tomado la decisión de utilizar **Minikube** para la integración, pues es una herramienta que simplifica los procesos de Kubernetes permitiendo ver su funcionamiento

de forma sencilla. También hemos utilizado **Kompose**, una herramienta que nos ayuda a generar los archivos necesarios para Kubernetes según nuestro docker-compose.

Comenzamos instalando Minikube, y iniciándose mediante **minikube start**, habiendo definido previamente a docker como driver predeterminado usando **minikube config set driver docker**.

Instalamos también **Kompose** usando la guía de su página de instalación

Hacemos **minikube stop**, para parar momentáneamente y en la carpeta **p2** que contiene el docker-compose, hacemos **kompose convert - f docker-compose.yaml** generando así los archivos necesarios para Minikube, los cuales movemos a una carpeta propia llamada **minikube**.

```
svnty@DESKTOP-L4U02M1:~/AOS-23/p2$ kompose convert -f docker-compose.yaml
INFO Network p2-default is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Network p2-default is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Network p2-default is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Kubernetes file "backend-service.yaml" created
INFO Kubernetes file "frontend-service.yaml" created
INFO Kubernetes file "proxy-service.yaml" created
INFO Kubernetes file "backend-deployment.yaml" created
INFO Kubernetes file "p2-default-networkpolicy.yaml" created
INFO Kubernetes file "frontend-deployment.yaml" created
INFO Kubernetes file "proxy-deployment.yaml" created
```

Ahora volvemos a iniciar **Minikube** con **minikube start** y ejecutamos el comando **kubectl apply -f .**

Veremos lo siguiente por línea de comandos

```
svnty@DESKTOP-L4U02M1:~/AOS-23/p2/minikube$ kubectl apply -f .
deployment.apps/backend created
service/backend created
deployment.apps/frontend created
service/frontend created
networkpolicy.networking.k8s.io/p2-default created
deployment.apps/proxy created
service/proxy created
error: error validating "docker-compose.yaml": error validating data: [apiVersion not set, kind not set];
```

Para comprobar que los pods están corriendo sin problemas hacemos **kubectl get pods**

```
svnty@DESKTOP-L4U02M1:~/AOS-23/p2/minikube$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
backend-7c8b96d566-4k5zz    1/1     Running   0          18s
frontend-cc468f587-6twwb    1/1     Running   0          18s
proxy-6dfb4d6cf4-hfbqr      1/1     Running   0          18s
```

Viendo que los pods están corriendo sin problemas, comprobamos los servicios con **kubectl get svc**, viendo así que todo sigue corriendo correctamente

```
svnty@DESKTOP-L4U02M1:~/AOS-23/p2/minikube$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
backend      ClusterIP   10.106.184.24    <none>        4010/TCP   98s
frontend     ClusterIP   10.104.166.4     <none>        8000/TCP   98s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    24m
proxy        ClusterIP   10.106.149.136   <none>        80/TCP     98s
svnty@DESKTOP-L4U02M1:~/AOS-23/p2/minikube$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
backend-7c8b96d566-4k5zz    1/1     Running   0          100s
frontend-cc468f587-6twwb    1/1     Running   0          100s
proxy-6dfb4d6cf4-hfbqr      1/1     Running   0          100s
```

Ahora vamos a intentar acceder a los servicios, que funcionaban previamente con docker compose. Para ver la IP con el puerto correcto que asigna **Minikube**, utilizamos el nombre de los servicios. Por ejemplo para encontrar la dirección del front hacemos **minikube service frontend --url**

```
svnty@DESKTOP-L4U02M1:~/AOS-23/p2$ minikube service frontend --url

    service default/frontend has no node port
http://127.0.0.1:37035
  Because you are using a Docker driver on linux, the terminal needs
```

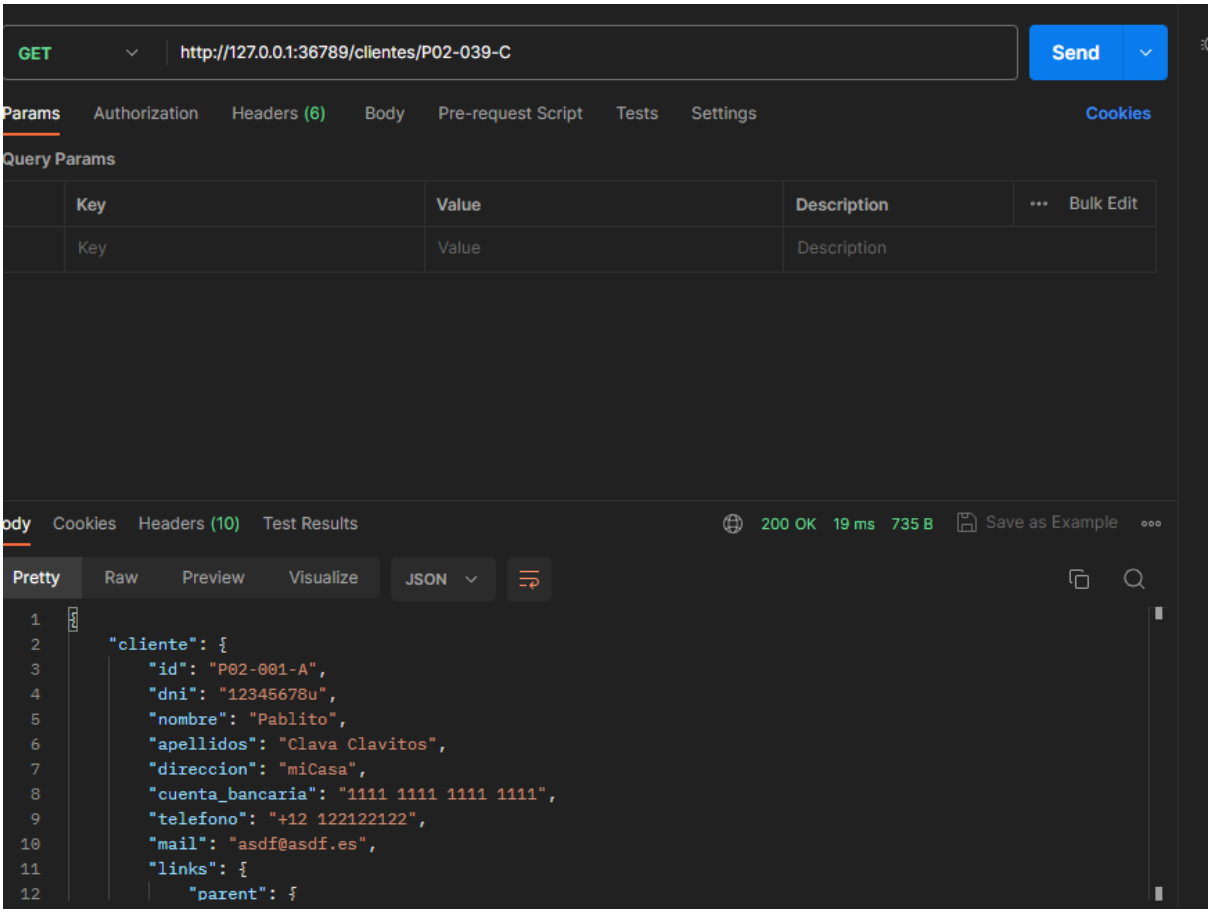
Cogemos la dirección que nos devuelve y la introducimos en un navegador, en el caso del ejemplo es **http://127.0.0.1:37035**

Entramos y podemos ver el frontend de Swagger funcionando. También podemos comprobar que las llamadas funcionan. Hacemos **minikube service backend --url** para encontrar la dirección y puerto asignado al backend.

```
^[[A^[[D^Csvnty@DESKTOP-L4U02M1:~/AOS-23/p2$ minikube service backend --url

   service default/backend has no node port
http://127.0.0.1:36789
  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

En este ejemplo la dirección es 127.0.0.1:36789, introducimos esta en **Postman** y realizamos la misma llamada que anteriormente, recibiendo una respuesta correcta.



Se puede finalmente comprobar que el sistema sigue funcionando mediante **docker compose up**

```
svnty@DESKTOP-L4U02M1:~/AOS-23/p2$ minikube service frontend --url
🏃  service default/frontend has no node port
http://127.0.0.1:46355
❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
^Csvnty@DESKTOP-L4U02M1:~/AOS-23/p2$ docker compose up

[+] Running 3/0
 ⠿ Container ClientesMock   Created                                                                0.0s
 ⠿ Container ClientesProxy  Created                                                                0.0s
 ⠿ Container ClientesUi     Created                                                                0.0s
Attaching to ClientesMock, ClientesProxy, ClientesUi
ClientesUi      | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
ClientesUi      | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
ClientesUi      | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
ClientesUi      | 10-listen-on-ipv6-by-default.sh: info: IPv6 listen already enabled
ClientesUi      | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
ClientesUi      | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
ClientesUi      | /docker-entrypoint.sh: Launching /docker-entrypoint.d/40-swagger-ui.sh
ClientesUi      | /docker-entrypoint.sh: Configuration complete; ready for start up
ClientesProxy   | {"level":"info","ts":1685305044.2651508,"msg":"using provided configuration","config_file":"/etc/caddy/Caddyfile
ClientesProxy   | {"level":"warn","ts":1685305044.2692635,"msg":"Caddyfile input is not formatted; run the 'caddy fmt' command to
ClientesProxy   | {"level":"info","ts":1685305044.273129,"logger":"admin","msg":"admin endpoint started","address":"localhost:2019
ClientesProxy   | {"level":"warn","ts":1685305044.2747576,"logger":"http","msg":"server is listening only on the HTTP port, so no
ClientesProxy   | {"level":"info","ts":1685305044.275011,"logger":"tls.cache.maintenance","msg":"started background certificate ma
ClientesProxy   | {"level":"info","ts":1685305044.2764814,"logger":"tls","msg":"cleaning storage unit","description":"FileStorage
ClientesProxy   | {"level":"info","ts":1685305044.2765186,"logger":"http.log","msg":"server running","name":"srv0","protocols":["
ClientesProxy   | {"level":"info","ts":1685305044.2767942,"logger":"tls","msg":"finished cleaning storage units"}
ClientesProxy   | {"level":"info","ts":1685305044.277204,"msg":"autosaved config (load with --resume flag)","file":"/config/caddy/
ClientesProxy   | {"level":"info","ts":1685305044.2772524,"msg":"serving initial configuration"}
ClientesMock    | [8:17:26 PM] › [CLI] …  awaiting  Starting Prism…
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   GET        http://0.0.0.0:4010/cliente
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   OPTIONS    http://0.0.0.0:4010/cliente
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   POST       http://0.0.0.0:4010/cliente
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   OPTIONS    http://0.0.0.0:4010/clientes/F08-203-R
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   GET        http://0.0.0.0:4010/clientes/M55-854-D
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   DELETE     http://0.0.0.0:4010/clientes/U87-276-Z
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   PUT        http://0.0.0.0:4010/clientes/Z29-884-N
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   GET        http://0.0.0.0:4010/vehiculos
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   POST       http://0.0.0.0:4010/vehiculos
ClientesMock    | [8:17:28 PM] › [CLI] ⠿  success   OPTIONS    http://0.0.0.0:4010/vehiculos
```