

Centro Universitario de Ciencias Exactas e ingenierías



Proyecto Final:

Proyecto Tolerancia a Fallas

Integrantes del equipo:

Guzmán Munguía Omar Alejandro

Lazo Villa Gabriel

López De Rueda Fernando Yair Valentín

Materia: Computación Tolerante a Fallas

Maestro: Lopez Franco Michel Emanuel

NRC: 179961

Sección: D06

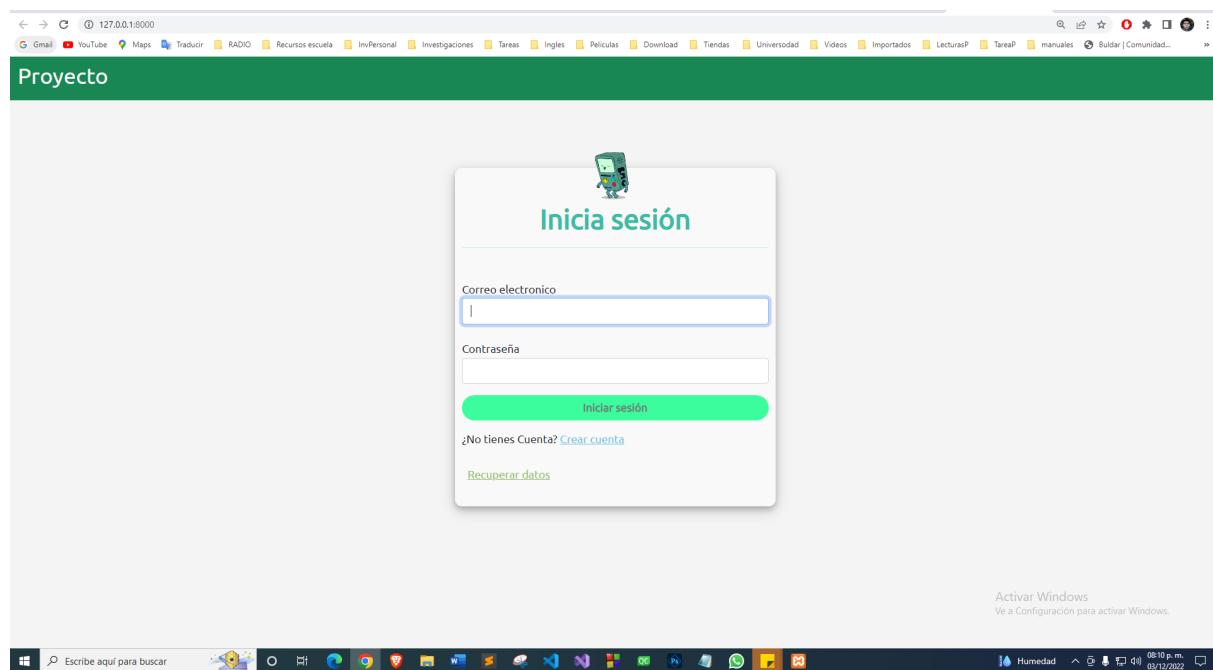
Clave: I7036

Calendario: 2022-B

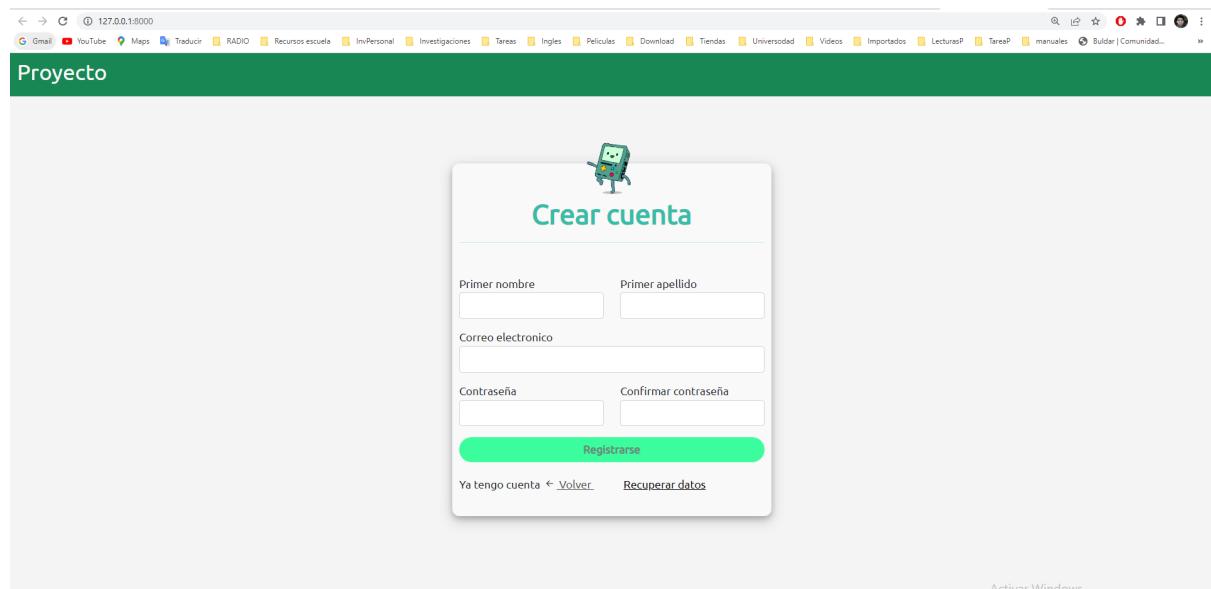
Herramientas y Lenguajes utilizados para el proyecto:

- **VS Code:** Visual Studio Code es un editor de código fuente, lo utilizamos por su facilidad para generar código con sus extensiones y porque es más fácil utilizarlo para hacer páginas web. Incluye soporte para la depuración, control integrado de Git, entre otras características.
- **Python:** Es un lenguaje de programación, fácil de entender y de ejecutarse, lo utilizamos para realizar nuestra aplicación web
- **Flask:** Microframework para el desarrollo web
- **Html:** Es un lenguaje de diseño web
- **JS:** Es un lenguaje de programación, lo usamos para generar las principales funciones que se encargaran de darle funcionalidad a la aplicación web
- **Css:** Es un lenguaje de diseño gráfico, lo usamos para darle un formato o diseño a la aplicación
- **MySQL:** Es un sistema de gestión de BD, lo usamos para guardar los datos de los formularios.
- **Docker:** Es una herramienta que usamos para nuestra aplicación que usamos para generar una imagen y un contenedor y para después usar esta en futuras herramientas
- **Kubernetes:** Es una plataforma de sistema distribuido para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores
- **Istio:** Lo usamos para gestionar el trabajo que le metimos a nuestra aplicación, junto con otras extensiones de este sistema como es el kiali, prometeo, grefana y jaeger, estas nos permiten tener una mejor visualización del tráfico, si todo está correcto o si hay algún error en el tráfico.
- **GitHub:** Es una plataforma para alojar nuestros proyectos, lo utilizamos para controlar las versiones de nuestra aplicación, y crear nuestro repositorio.
- **Minikube:** Es un servidor local, optamos por esta opcion por que es gratis y es fácil de ejecutarse

Iniciamos este proyecto utilizando una base de datos de un login, donde dicho usuario registrado y logueado podría tener acceso al uso de todas las APIs. Asimismo, metimos algo de localStorage para poder recuperar los datos registrados y de inicio de sesión en caso de que haya una falla en el sistema, como lo puede ser una mala conexión conexión a internet o caída del servidor. Sin embargo, para subir nuestra aplicación a docker tuvimos problemas con la base de datos por lo que no nos dejaba conectarla correctamente y nos generaba errores, desafortunadamente no pudimos solucionar el problema por lo que optamos en eliminar el login y utilizar nuestra aplicación de manera directa sin la necesidad de registrarse e iniciar sesión. A continuación mostraremos algunas imágenes de lo que se había hecho:



Página de registro de usuario



Una vez que lo registraba lo guardaba en nuestra base de datos.

	id	nombre	apellido	email	password	create_at
<input type="checkbox"/>	4	gabriel	lazo	gabo12@gmail.com	sha256\$Zbxw20ucYoWFV3T\$88b07fa578b43660fa238f1	2022-12-02

Como se mencionó anteriormente, al no poder unir la base de datos con docker, se optó a solo utilizar nuestra app de manera libre sin la necesidad de un inicio de sesión. A continuación se hablará un poco sobre algunas funciones del código y todo el proceso que se aplicó.

Lo primero que se realizó fue nuestra aplicación, donde utilizamos flask para desplegarlo en un servidor web. A continuación mostraremos algunas de las funciones principales de nuestro programa.

```
#Redireccionando cuando la página no existe
@app.errorhandler(404)
def not_found(error):
    if 'conectado' in session:
        return redirect(url_for('home')), 404
    else:
        return render_template('public/modulo_login/index.html')

#Creando mi Decorador para el Home
@app.route('/')
def inicio():
    return render_template('public/dashboard/home.html')

@app.route('/inicio', methods=['GET', 'POST'])
def home():
    if request.method == 'GET':
        return render_template('public/dashboard/pages/Dashboard.html')
    if request.form['search']:
        url= "https://api.giphy.com/v1/gifs/search?api_key=rGmZSIYd6EvztznZGnv66seXlpwAItb&limit=12&q="
        giphy = requests.get(url)
        dataGiphy = giphy.json()
        # print(dataGiphy)
        return render_template('public/dashboard/pages/Dashboard.html', data=dataGiphy['data'])
    else:
        return render_template('public/dashboard/pages/Dashboard.html')
```

Integración de las apis de DOGS, la referencia a la api de MARVEL y a la de pokémon

```
@app.route('/dog', methods=["GET", "POST"])
def index():
    if request.method == 'GET':
        url = "https://dog.ceo/api/breeds/image/random"

    if request.method == 'POST':
        name = request.form['name']
        url = "https://dog.ceo/api/breed/{}/images/random".format(name)

    resultado = requests.get(url)
    datos = resultado.json()
    imagen = datos["message"]

    return render_template("public/dashboard/pages/dog.html", imagen=imagen)

@app.route('/marvel')
def marvel():
    return render_template('public/dashboard/pages/marvel.html')

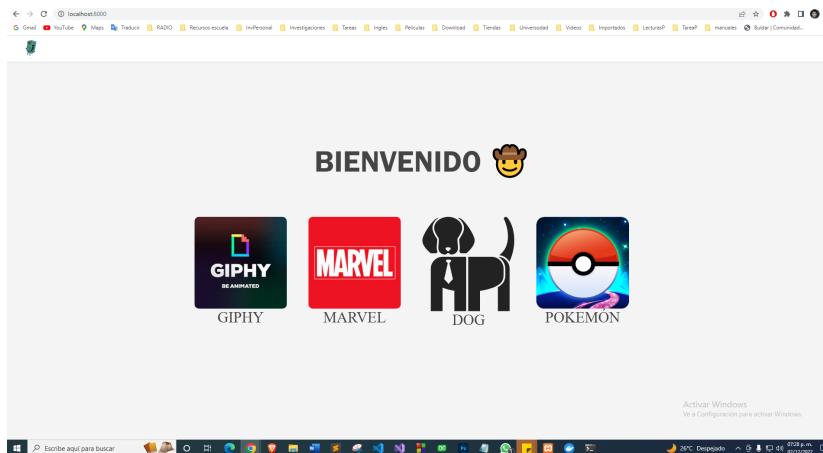
@app.route('/pokemon')
def pokemon():
    return render_template('public/dashboard/pages/error.html')
```

Integración de la api de marvel, la cual fue codificada en javascript integrando HTML dentro de la función.

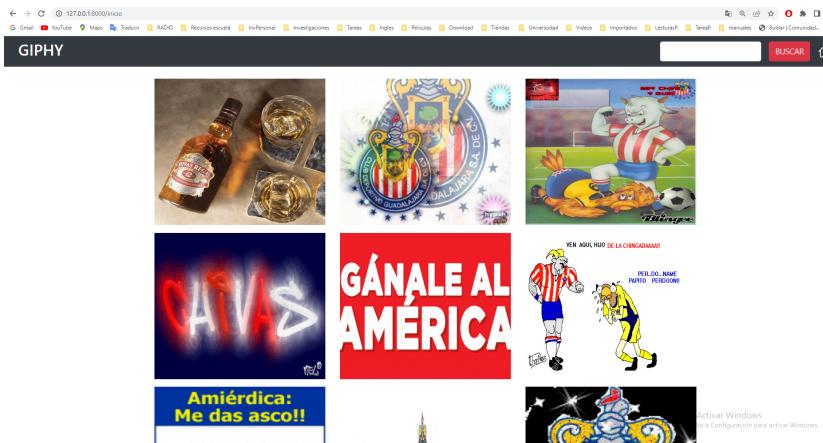
```
OPEN E... D... S... @
  app > static > assets > js > appMarvel.js > ...
  < app.py app
  < requirements.txt
  < routes.py app 1
  < X JS appMarvel.js app.js
  < THEPRO... D... S... U... @
> .vscode
> app
  > static / assets
    > css
    > imgs
    > js
      < JS app.js
      < JS appMarvel.js
      < JS custom_alert.js
      < JS error.js
      < JS jquery-3.6.0.js
      < JS menus.js
      < JS pace.min.js
      < JS scripts.js
  > templates / public
    > dashboard
    > modulo_login

1  const marvel = {
2    render: () => {
3      const urlAPI = 'https://gateway.marvel.com:443/v1/public/characters?ts=1&apikey=04c50da0ef8a4f49a1679b619fb88aeb&hash=8589a
4      const container = document.querySelector("#marvel-row");
5      let contentHTML = '';
6
7      fetch(urlAPI)
8        .then(res => res.json())
9        .then(json => {
10          for (const hero of json.data.results) {
11            let urlHero = hero.urls[0].url;
12            contentHTML += `
13              <div class="col-md-4">
14                <a href="${urlHero}" target="_blank">
15                  
16                </a>
17                <h3 class="title">${hero.name}</h3>
18              </div>;
19            }
20            container.innerHTML = contentHTML;
21          }
22        });
23      marvel.render();
24    }
25  };
26
27  marvel.render();
```

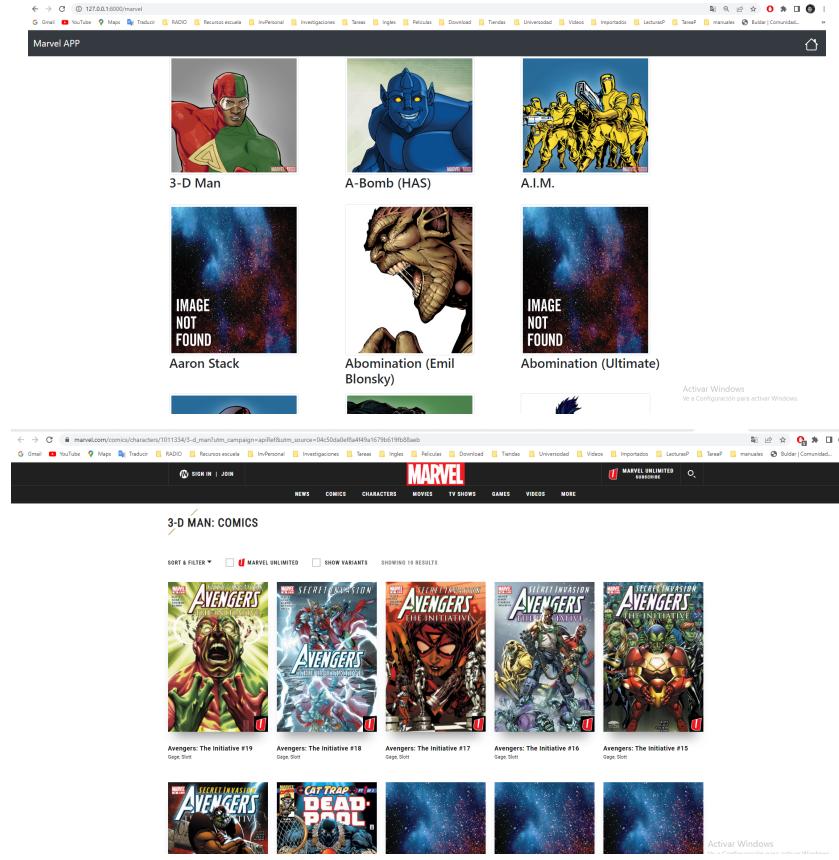
Esta imagen representa el menú principal de nuestra aplicación, el cual consta de un mensaje de bienvenida seguido de unas tarjetas las cuales representan las distintas APIs que el usuario puede consumir (GIPHY, MARVEL, DOGS, etc).



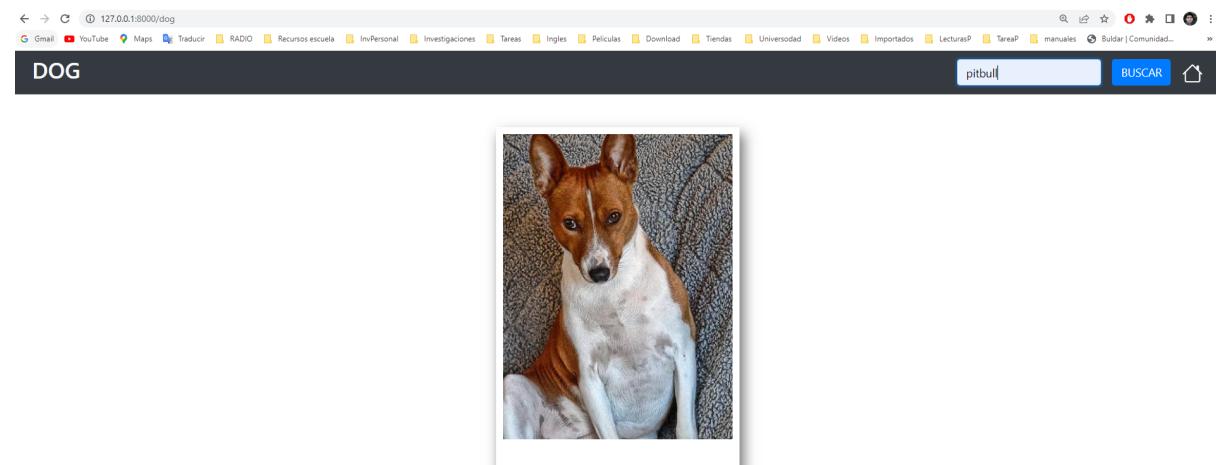
Al dar click en la tarjeta de GIPHY lo dirigirá a una página en la cual el usuario podrá buscar una gran variedad de gifs los cuales podrá guardar en su ordenador.



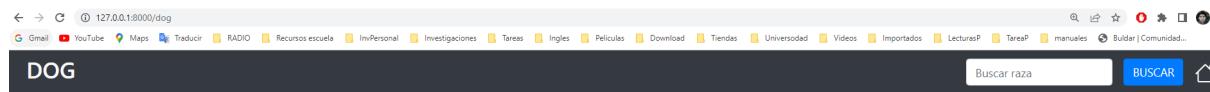
Al seleccionar la opción de MARVEL se dirigirá a una página en la cual aparecerán varios personajes de la franquicia los cuales al dar click en alguna nos dirigirá a una página donde se nos desplegará información sobre dicho personaje, su biografía y las películas y cómics en los que hace aparición.



Al seleccionar la opción del Dog se direccionara a una página en la cual le aparecerá en primera visión un perro de cualquier raza aleatoria, se puede ver que se encuentra un botón de búsqueda, este hace una búsqueda de los tipo de raza y genera una imagen con un perro de la raza buscada, en nuestro caso le pondremos pitbull a la barra de búsqueda



Y como se puede ver nos genero una imagen de un perro pitbull, y esto se debe a que la petición de la busqueda se la hace a una api que genera imágenes de perros aleatoriamente donde tiene como campo principal la raza, por esa razón es fácil buscar un perro por su raza en este apartado



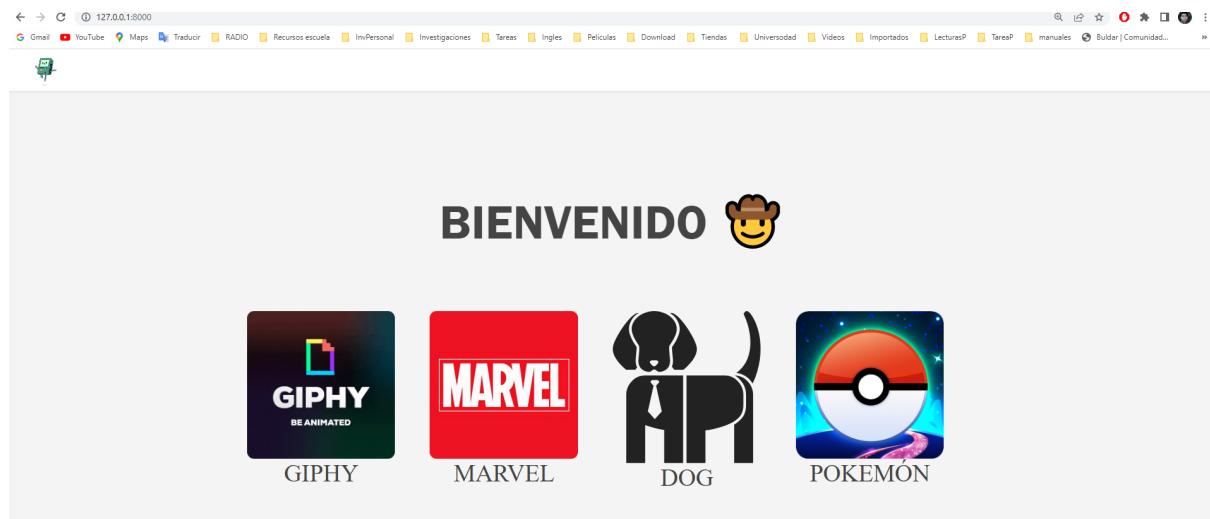
Al dar click en la opción de pokemon direcccionará al usuario a una página en la cual se le informará que dicha API está en fase de pruebas durante un lapso de 4 segundos los cuales al transcurrir lo direcccionará automáticamente a la página de inicio pudiendo aplicar como una función tolerante a fallas dentro de nuestra app.



PROXIMAMENTE

Esta API se encuentra en fase de pruebas. Vuelve pronto!

Regresando...



Una vez que teníamos nuestra aplicación, lo siguiente que hicimos fue construir nuestra imagen en docker. Esto dando click derecho a nuestro Dockerfile o bien, desde la línea de comandos con ‘docker build -t name_image’

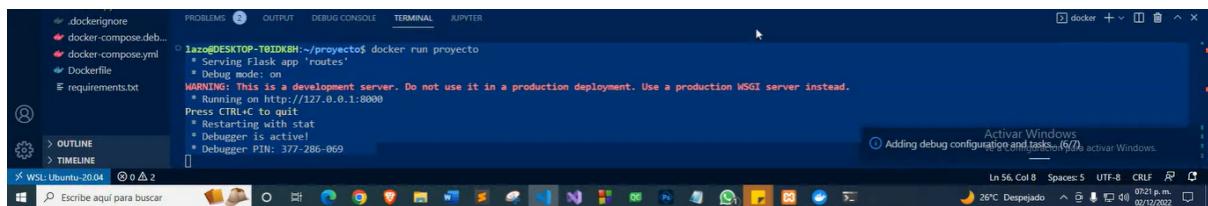
```
* Executing task: docker-build  
> docker build --rm -f "/home/lazo/proyecto/Dockerfile" --label "com.microsoft.created-by=visual-studio-code" -t "proyecto:latest" "/home/lazo/proyecto" <  
#1 [internal] load build definition from Dockerfile  
#1 sha256:1db00382ef364932cfe98539e271a36b1c4cdc8239c8d7e60685478f14ec48  
#1 transferring dockerfile: 946B done  
#1 DONE 0.0s
```

Vemos que se nos generó nuestra imagen correctamente. De la misma forma, podemos confirmar que se creó la imagen con el comando docker images

```
#10 sha256:c1a88a58e693568934f454/c41acd/64720a1/18bb12528e4b0d1/740f9c4ef0  
#10 DONE 0.0s  
  
#11 [6/6] RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app  
#11 sha256:605abde7b0a9163ff4a0ff681fe9ab12c220e83620a786e77f04efbba345291  
#11 0.622 Adding user 'appuser' ...  
#11 0.622 Adding new group 'appuser' (5678) ...  
#11 0.640 Adding new user 'appuser' (5678) with group 'appuser' ...  
#11 0.691 Creating home directory '/home/appuser' ...  
#11 0.691 Copying files from '/etc/skel' ...  
#11 DONE 0.8s  
  
#12 exporting to image  
#12 sha256:e8bc613e07b0b7ff33893b694f7759a10d42e180f2b4dc349fb57dc6b71dcab00  
#12 exporting layers 0.1s done  
#12 writing image sha256:68c271cdab8aa7583814805ba6d7601b7b38716b65fb1275b29ec45f588df8b8 done  
#12 naming to docker.io/library/proyecto:latest done  
#12 DONE 0.1s  
  
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them  
Terminal will be reused by tasks, press any key to close it.
```

Activar Windows
Ver la configuración para más

Una vez que se subió solamente corrimos nuestra aplicación desde docker, primero lo hicimos solo con docker run y luego hicimos un expose desde otro puerto.



Lo siguiente que hicimos fue subirlo a nuestro Docker hub, por lo que primero verificamos que nuestra imagen esté cargada en nuestro docker para ya después solo hacer un push y subirla a nuestro repositorio de Docker hub.

```

● ^C lazo@DESKTOP-T0IDK8H:~/theproject$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
theproject          latest   2fc10894b9bc  11 minutes ago  135MB
<none>              <none>   5eb320fad0e1  12 minutes ago  127MB
gifhub              latest   b26be94d1cf1  41 minutes ago  134MB
<none>              <none>   0c3567e9eacc  41 minutes ago  126MB
<none>              <none>   e8797f5b1947  48 minutes ago  126MB
gabreillazo/go-web-app    latest   56f802efc9d5  3 weeks ago   355MB
gcr.io/k8s-minikube/kicbase v0.0.36  866c1fe4e3f2  5 weeks ago   1.11GB
vscodeexample        latest   0b5987aa43e0   5 weeks ago   126MB
● lazo@DESKTOP-T0IDK8H:~/theproject$ docker tag theproject:v1 gabreillazo/theproject
Error response from daemon: No such image: theproject:v1
● lazo@DESKTOP-T0IDK8H:~/theproject$ docker tag theproject:latest gabreillazo/theproject
● lazo@DESKTOP-T0IDK8H:~/theproject$ docker push gabreillazo/theproject
Using default tag: latest
The push refers to repository [docker.io/gabreillazo/theproject]
b07ae73c26d8: Pushed
9e4d465c1b0c: Pushed
eef533b41f41: Pushed
289058d5fe0c: Pushed
d6f90630b9a0: Pushed
d092e599f754: Mounted from library/python
0cbed68a1289: Pushed
4834dc4f352a: Mounted from library/python
d4ea5264b420: Mounted from library/python
ec4a38999118: Mounted from library/python
latest: digest: sha256:97d7e47495f8c7c4392506aec89f8b722afadce10c02c650ad39df86e76bf97f size: 2415

```

Una vez que lo subimos, solo nos vamos a docker hub y actualizamos la página y podemos apreciar que se subió correctamente nuestra imagen.

The screenshot shows the Docker Hub interface. At the top, there's a search bar and navigation links for Explore, Repositories, Organizations, Help, Upgrade, and a user dropdown for gabreillazo. Below the search bar, there's a filter for Content. A search result for 'gabreillazo' is shown, with a dropdown menu open. The first result, 'gabreillazo / theproject', is highlighted with a red border. It shows details: Contains: Image | Last pushed: a minute ago. To the right, it shows Not Scanned, 0 stars, 0 downloads, and a Public icon. Below this, there's another repository entry for 'gabreillazo / go-web-app' with similar details. At the bottom, there's a tip message: 'Tip: Not finding your repository? Try a different namespace.'

Un vez hecho lo anterior nos pasaremos a lo que es KUBERNETES para lo cual iniciaremos lo que minikube con el comando **minikube start**

```

lazo@DESKTOP-T0IDK8H:~/proyecto$ minikube start
😄 minikube v1.28.0 on Ubuntu 20.04
👍 Using the docker driver based on existing profile
Starting control plane node minikube in cluster minikube
Pulling base image ...
🕒 Restarting existing docker container for "minikube" ...
🔄 Preparing Kubernetes v1.25.3 on Docker 20.10.20 ...
⚡️ Verifying Kubernetes components...
  • Using image docker.io/kubernetesui/dashboard:v2.7.0
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
  • Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡 Some dashboard features require the metrics-server addon. To enable all features please run:
      minikube addons enable metrics-server

⭐️ Enabled addons: storage-provisioner, default-storageclass, dashboard
🎉 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

```

Después procederemos a observar nuestros pods con el comando **kubectl get pods** para ver que todo esté en orden.

NAME	READY	STATUS	RESTARTS	AGE
go-web-app-f4dc668f4-24bm9	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-29r4j	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-bmx8k	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-hqgj5	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-jhrf4	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-sb9t6	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-z5cns	1/1	Running	12 (26h ago)	21d
go-web-service-76b95588c4-bxnsr	1/1	Running	12 (26h ago)	21d
hello-v1-57489c999-dbxqb	1/2	Running	8 (26h ago)	15d
hello-v1-57489c999-t6k89	1/2	Running	8 (26h ago)	15d
hello-v1-57489c999-vb2k4	1/2	Running	8 (26h ago)	15d
hello-v2-557cff9dbc-5htxm	1/2	Running	8 (26h ago)	15d
hello-v2-557cff9dbc-l8fq2	1/2	Running	8 (26h ago)	15d
hello-v2-557cff9dbc-sbr5j	1/2	Running	8 (26h ago)	15d
hello-v2-5dcc64496b-td4vb	0/2	ImagePullBackOff	4 (26h ago)	15d

Una vez mostrados los pods que se encuentran en actualmente procederemos a aplicar los cambios a los archivos deployment y service para que se puedan aplicar nuestros pods que pusimos en los archivos

lazo@DESKTOP-T0IDK8H:~/theproject\$ kubectl apply -f deployment.yml				
deployment.apps/theproject created				
lazo@DESKTOP-T0IDK8H:~/theproject\$ kubectl apply -f service.yml				
service/theproject-service created				
lazo@DESKTOP-T0IDK8H:~/theproject\$ kubectl get pods				
NAME	READY	STATUS	RESTARTS	AGE
go-web-app-f4dc668f4-24bm9	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-29r4j	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-bmx8k	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-hqgj5	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-jhrf4	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-sb9t6	1/1	Running	12 (26h ago)	21d
go-web-app-f4dc668f4-z5cns	1/1	Running	12 (26h ago)	21d
go-web-service-76b95588c4-bxnsr	1/1	Running	12 (26h ago)	21d
hello-v1-57489c999-dbxqb	1/2	Running	8 (26h ago)	15d
hello-v1-57489c999-t6k89	1/2	Running	8 (26h ago)	15d
hello-v1-57489c999-vb2k4	1/2	Running	8 (26h ago)	15d
hello-v2-557cff9dbc-5htxm	1/2	Running	8 (26h ago)	15d
hello-v2-557cff9dbc-l8fq2	1/2	Running	8 (26h ago)	15d
hello-v2-557cff9dbc-sbr5j	1/2	Running	8 (26h ago)	15d
hello-v2-5dcc64496b-td4vb	0/2	ImagePullBackOff	4 (26h ago)	15d
theproject-79854bbcf-dfwjx4	1/1	Running	0	15s
theproject-79854bbcf-nlhnm	1/1	Running	0	15s
theproject-79854bbcf-vzxlv	1/1	Running	0	15s

Para verificar que nuestros pods se crearon correctamente, podemos abrir el dashboard que nos proporciona kubernetes. Lo podemos abrir con el comando de ‘minikube dashboard’.

De esta forma podemos apreciar nuestros pods y nuestro servicio y ver que está corriendo sin errores.

Si nos vamos al apartado de pods podemos apreciar que nuestros pods están en perfecto estado y que estos se generaron correctamente.

Podemos entrar a un pod en específico y ver en qué estado se encuentra y ver la dirección y el puerto que está agarrando para después abrirlo y ver que funciona.

Abrimos nuestro servicio y vemos que también está corriendo correctamente.

The screenshot shows the Kubernetes UI for a service named 'theoproject-service'. The 'Metadatos' section includes the service name, creation date (2 dic. 2022), and last applied configuration hash (92ae9cba-155d-4fb5-b096-a80f01459216). The 'Información del recurso' section shows it's a LoadBalancer type with IP 10.107.207.44 and no session affinity. The 'Endpoints' section lists three entries: 172.17.0.20, 172.17.0.21, and 172.17.0.22, all pointing to port 3000 TCP and node 'minikube'. The 'Endpoints' table has columns: Anfitrión (Host), Puerto (Nombre, Puerto, Protocolo) (Port), Nodo (Node), and Listo (Ready).

Con el siguiente comando **kubectl scale --replicas=5 deployment/<name_image>** escalaremos nuestra aplicación aumentando el número de pods a un total de 5 y al final corroboramos obteniendo de nuevo nuestros pods

```
lazo@DESKTOP-T0IDK8H:~/theoproject$ kubectl scale --replicas=5 deployment/theoproject
deployment.apps/theoproject scaled
lazo@DESKTOP-T0IDK8H:~/theoproject$ kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
go-web-app-f4dc668f4-24bm9       1/1     Running   12 (26h ago)  21d
go-web-app-f4dc668f4-29r4j       1/1     Running   12 (26h ago)  21d
go-web-app-f4dc668f4-bmx8k       1/1     Running   12 (26h ago)  21d
go-web-app-f4dc668f4-hqgj5       1/1     Running   12 (26h ago)  21d
go-web-app-f4dc668f4-jhrf4       1/1     Running   12 (26h ago)  21d
go-web-app-f4dc668f4-sb9t6       1/1     Running   12 (26h ago)  21d
go-web-app-f4dc668f4-z5cns      1/1     Running   12 (26h ago)  21d
go-web-service-76b95588c4-bxnsr  1/1     Running   12 (26h ago)  21d
hello-v1-57489c999-dbxqb        1/2     Running   8 (26h ago)   15d
hello-v1-57489c999-t6k89        1/2     Running   8 (26h ago)   15d
hello-v1-57489c999-vb2k4        1/2     Running   8 (26h ago)   15d
hello-v2-557cff9dbc-5htxm      1/2     Running   8 (26h ago)   15d
hello-v2-557cff9dbc-18fq2       1/2     Running   8 (26h ago)   15d
hello-v2-557cff9dbc-sbr5j       1/2     Running   8 (26h ago)   15d
lalo-2-51-6WU2L-4WJL            0/2     Image Pulling 0 (26h ago)   15d
theoproject-79854bbcf4-f55m4     1/1     Running   0          11s
theoproject-79854bbcf4-fwjx4     1/1     Running   0          6m29s
theoproject-79854bbcf4-nlhnm     1/1     Running   0          6m29s
theoproject-79854bbcf4-vzxlv     1/1     Running   0          6m29s
theoproject-79854bbcf4-x8rkz     1/1     Running   0          11s
```

Después con el primer comando crearemos un puente para desplegar nuestra aplicación y una vez creado con el comando **minikube service <name_image>-service** desplegamos el puente creado

```

error: service theproject-service does not have a service port 3000
lazo@DESKTOP-T0IDK8H:~/theproject$ kubectl create deployment theproject-service --image=k8s.gcr.io/echoservice:v1.4
deployment.apps/theproject-service created
lazo@DESKTOP-T0IDK8H:~/theproject$ minikube service theproject-service
|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|
| default   | theproject-service | http/80 | http://192.168.49.2:31831 |
|-----|
💡 Starting tunnel for service theproject-service.
|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|
| default   | theproject-service | http/80 | http://127.0.0.1:37375 |
|-----|
💡 Opening service default/theproject-service in default browser...
👉 http://127.0.0.1:37375

```

Lo siguiente que utilizaremos será utilizar istio para monitorear el tráfico, las estadísticas y las métricas de nuestros pods y servicios.

```

lazo@DESKTOP-T0IDK8H:~/theproject$ curl -L https://istio.io/downloadIstio | sh -
% Total    % Received % Xferd  Average Speed   Time     Time   Current
          Dload  Upload Total   Spent    Left Speed
100  101  100  101    0      0  483      0 --:--:-- --:--:-- --:--:-- 483
100 4856  100 4856    0      0  9145      0 --:--:-- --:--:-- --:--:-- 48560

Downloading istio-1.16.0 from https://github.com/istio/istio/releases/download/1.16.0/istio-1.16.0-linux-amd64.tar.gz ...
Istio 1.16.0 Download Complete!

Istio has been successfully downloaded into the istio-1.16.0 folder on your system.

Next Steps:
See https://istio.io/latest/docs/setup/install/ to add Istio to your Kubernetes cluster.

To configure the istioctl client tool for your workstation,
add the /home/lazo/theproject/istio-1.16.0/bin directory to your environment path variable with:
export PATH="$PATH:/home/lazo/theproject/istio-1.16.0/bin"

Begin the Istio pre-installation check by running:
  istioctl x precheck

Need more information? Visit https://istio.io/latest/docs/setup/install/
lazo@DESKTOP-T0IDK8H:~/theproject$ export PATH="$PATH:/home/lazo/theproject/istio-1.16.0/bin"
lazo@DESKTOP-T0IDK8H:~/theproject$ istioctl x precheck
✓ No issues found when checking the cluster. Istio is safe to install or upgrade!
  To get started, check out https://istio.io/latest/docs/setup/getting-started/

```

En esta parte debemos de aplicar de nuevo nuestros archivos deployment y service para que nuestros pods que generamos sean inyectados para así generar nuestro tráfico después

```

lazo@DESKTOP-T0IDK8H:~/theproject$ kubectl apply -f deployment.yml
deployment.apps/theproject created
lazo@DESKTOP-T0IDK8H:~/theproject$ kubectl apply -f service.yml
service/theproject-service unchanged
lazo@DESKTOP-T0IDK8H:~/theproject$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
go-web-app-f4dc668f4-24bm9   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-29r4j   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-bmx8k   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-hqgj5   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-jhrf4   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-sb9t6   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-z5cns   1/1     Running   12 (27h ago)  21d
go-web-service-76b95588c4-bxnsr 1/1     Running   12 (27h ago)  21d
hello-v1-57489c999-dbxqb   1/2     Running   8 (27h ago)   15d
hello-v1-57489c999-t6k89   1/2     Running   8 (27h ago)   15d
hello-v1-57489c999-vb2k4   1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-5htxm  1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-l8fq2  1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-sbr5j  1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-twkkk  0/2     ImagePullBackOff 4 (27h ago)  15d
theproject-79854bbcf4-4wbh2  2/2     Running   0           29s
theproject-79854bbcf4-ht6gf  2/2     Running   0           29s
theproject-79854bbcf4-v64mc  2/2     Running   0           29s

```

Después con el siguiente comando aplicaremos la inyección a nuestros pods con el siguiente comando

```

lazo@DESKTOP-T0IDK8H:~/proyecto$ kubectl label namespace default istio-injection=enabled
namespace/default labeled

```

Una vez aplicados los cambios y hecha la inyección volveremos a escalar nuestra aplicación de forma que de tener 3 pods pasaremos a tener 5 además de que algunos ya muestran la inyección metida con anterioridad.

```

lazo@DESKTOP-T0IDK8H:~/theproject$ kubectl scale --replicas=5 deployment/theproject
deployment.apps/theproject scaled
lazo@DESKTOP-T0IDK8H:~/theproject$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
go-web-app-f4dc668f4-24bm9   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-29r4j   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-bmx8k   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-hqgj5   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-jhrf4   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-sb9t6   1/1     Running   12 (27h ago)  21d
go-web-app-f4dc668f4-z5cns   1/1     Running   12 (27h ago)  21d
go-web-service-76b95588c4-bxnsr 1/1     Running   12 (27h ago)  21d
hello-v1-57489c999-dbxqb   1/2     Running   8 (27h ago)   15d
hello-v1-57489c999-t6k89   1/2     Running   8 (27h ago)   15d
hello-v1-57489c999-vb2k4   1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-5htxm  1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-l8fq2  1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-sbr5j  1/2     Running   8 (27h ago)   15d
hello-v2-557cff9dbc-twkkk  2/2     ImagePullBackOff 4 (27h ago)  15d
theproject-79854bbcf4-4wbh2  2/2     Running   0           108s
theproject-79854bbcf4-ht6gf  2/2     Running   0           108s
theproject-79854bbcf4-mpahr  0/2     PodInitializing 0           4s
theproject-79854bbcf4-v64mc  2/2     Running   0           108s
theproject-79854bbcf4-zwkkk  1/2     Running   0           4s

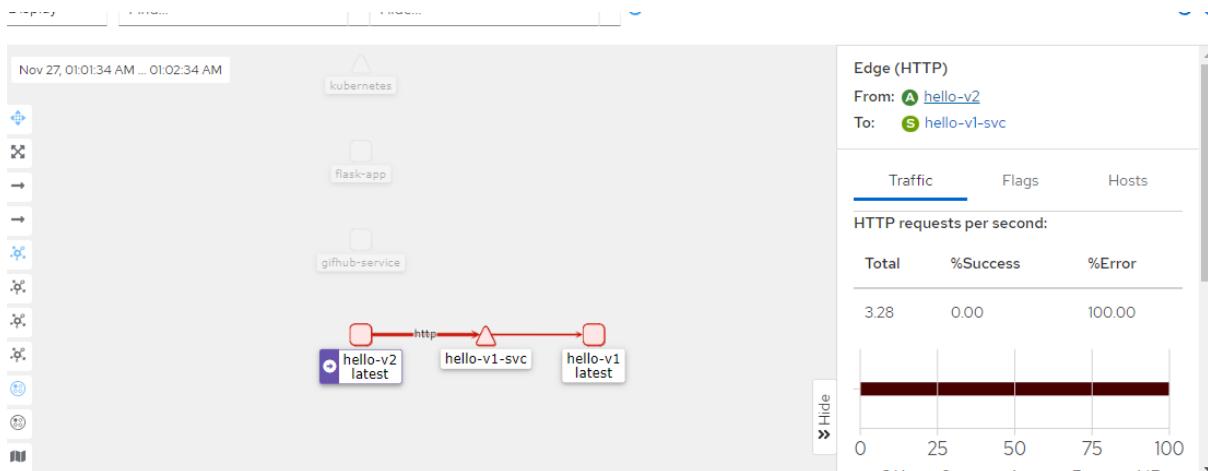
```

Al hacer el describe de cualquier pod podemos apreciar como se le generó un proxy a nuestra aplicación, esto gracias a la inyección que se le aplicó con istio, donde dicha inyección le generó otro contenedor a nuestros pods.

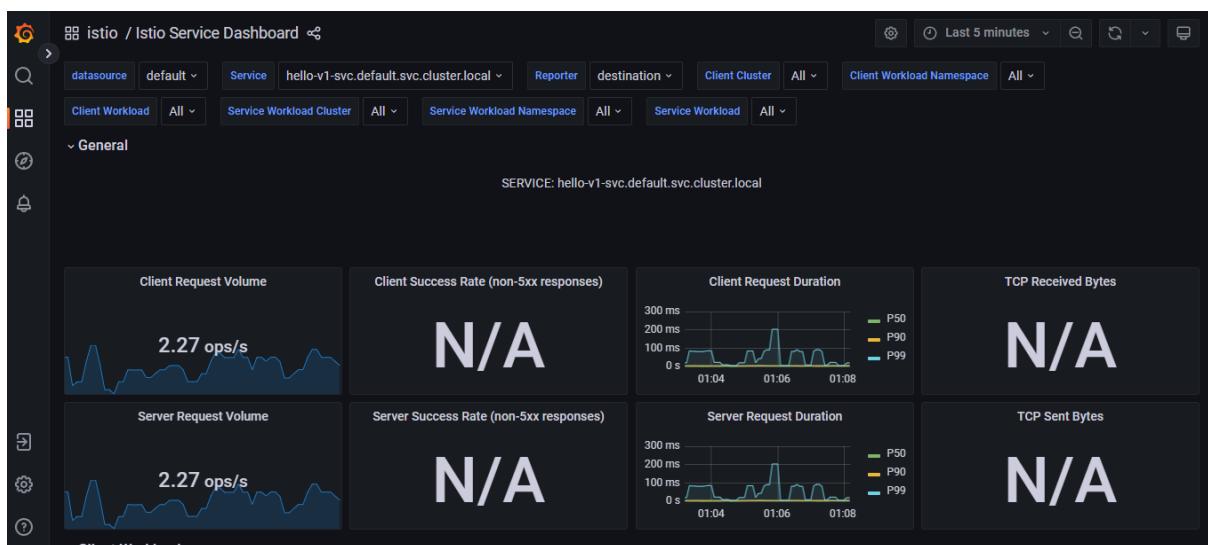
```
error: the server doesn't have a resource type "theproyect-79854bbcf8-ht6gf"
lazo@DESKTOP-T0IDK8H:~/theproyect$ kubectl describe pod theprojetc-79854bbcf8-ht6gf
Name:           theprojetc-79854bbcf8-ht6gf
Namespace:      default
Priority:       0
Service Account: default
Node:          minikube/192.168.49.2
Start Time:    Fri, 02 Dec 2022 23:17:37 -0600
Labels:         app=theproject, component=theproject, pod=theproject-79854bbcf8-ht6gf
Annotations:    
Status:        Running
IP:            192.168.49.2
IPv6:          <none>
Ports:         <none>
Conditions:   1 ready, 0 unhealthy, 0 total
Containers:
  application:
    Container ID:  docker://78602f370d193b8eb3a99f3cede411da6fb4d467426c065a00c893cea0e39d5
    Image:         gabreillazo/theproject
    Image ID:     docker-pullable://gabreillazo/theproject@sha256:97d7e47495f8c7c4392506aec89f8b722afadce1
    0c02c650ad39df86e76bf97f
    Port:          3000/TCP
    Host Port:    0/TCP
    State:        Running
    Started:     Fri, 02 Dec 2022 23:17:40 -0600
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:       <none>
    /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-nscpt (ro)
  istio-proxy:
    Container ID:  docker://19a323a54058f777a4fd71027efda6f64356cc37cd13f0a17fd811aea5c6c90b
    Image:         docker.io/istio/proxyv2:1.16.0
    Image ID:     docker-pullable://istio/proxyv2@sha256:f6f97fa4fb77a3cbe1e3eca0fa46bd462ad5b284c129cf57bf
    1575c4fb50cf9
    Port:          15090/TCP
    Host Port:    0/TCP
```

Una vez que ya teníamos nuestra aplicación quisimos ponerla a prueba, generando un poco de tráfico para ver si nuestra aplicación soportaba. Simplemente hicimos un `while` que hiciera una petición cada segundo.

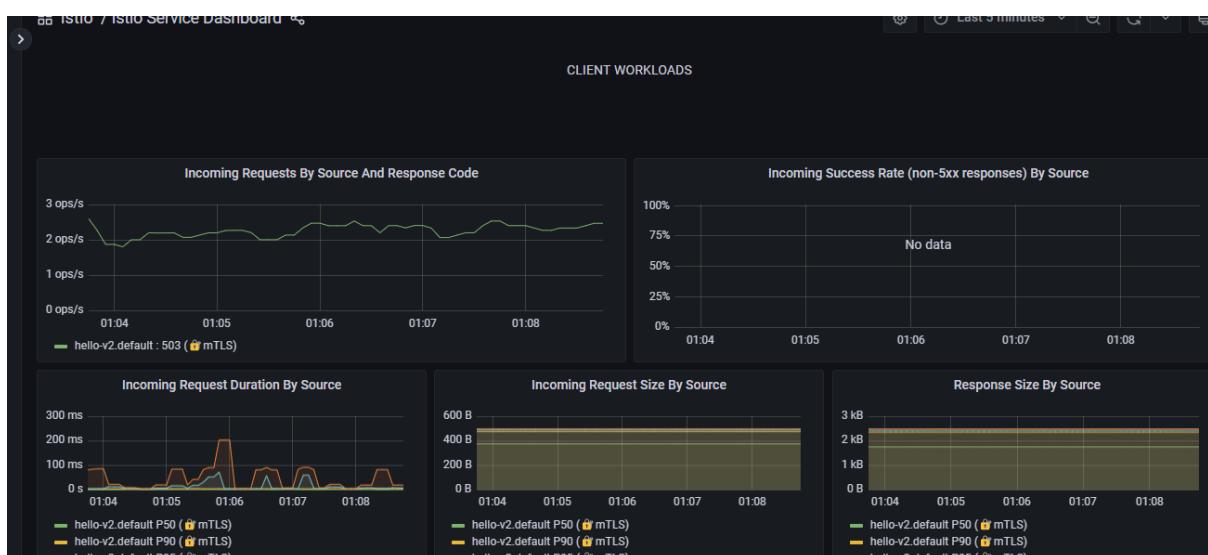
Para ver el tráfico, utilizamos unos módulos de Istio como lo son Kiali, grafana y Jaeger. Esto para monitorear el tráfico de nuestra aplicación. Lo que pudimos ver fue que después de muchas peticiones continuas, nuestra aplicación tronó y nos mandaba el error. Lo más probable por tantas peticiones en simultáneo.



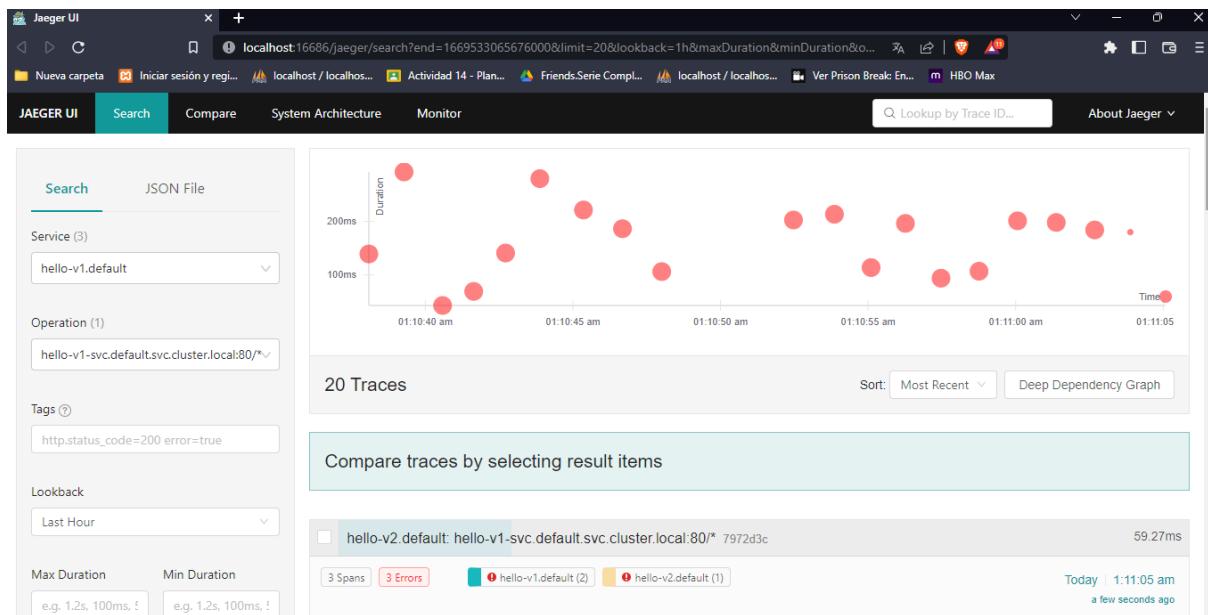
Tráfico de nuestra aplicación observado desde el dashboard de grafana dandonos una muestra de distintas métricas de nuestro servicio.



Algo interesante de grafana es que nos permite ver muchas estadísticas de nuestro servicio, cada una de distinto modo, generando así un informe más profundo de lo que está pasando con nuestra aplicación.



Aquí podemos apreciar Jaeger de acuerdo al servicio que se seleccionó; dándonos algunas características sobre este.



Conclusión:

El juntar nuestra aplicación con docker nos resultó muy complejo debido a que al hacer uso del servidor local para lo que es el registro y el login, docker nos nos quería generar la imagen por lo que tuvimos que quitarle la parte de bases de datos a la aplicación para que se pudiera hacer uso de docker, kubernetes, etc.

También una de las cosas que nos ayudó fue la forma en cómo implementamos nuestra aplicación, al hacerla con flask nos permitía tener en consola los errores que teníamos en tiempo real, ahorrándonos mucho tiempo y nos permite saber cuando estaba fallando nuestra aplicación y en qué parte.

Finalmente, podemos decir que al hacer este proyecto aprendimos sobre nuevas tecnologías como docker, kubernetes e Istio. Dichas herramientas no las conocíamos antes de entrar al curso y son herramientas que lo más probable es que las utilicemos en el mundo laboral. Además, esto para nosotros fue un verdadero reto ya que siempre se nos dificultaba usar estas tecnologías, y con este proyecto pudimos pulir cada una de ellas, entendiendo su funcionamiento y en que nos pueden beneficiar a futuro.