Istio

Computación Tolerante a Fallas

Alumno:

Código:

Profesor(a): Dr. Michel Emanuel López Franco

Sección: D06

Fecha de Entrega: 14 de noviembre de 2022

¿Qué es Istio?

Istio es una malla de servicios que ofrece una manera transparente e independiente de cualquier lenguaje de automatizar las funciones de red de una aplicación de forma flexible y sencilla. Permite el intercambio de datos entre los microservicios y su diseño facilita su ejecución en distintos entornos, como contenedores de Kubernetes.

Utilizar una malla de servicios simplifica las implementaciones y alivia la carga de los equipos de desarrollo. Entre sus principales funciones encontramos la gestión de tráfico, seguridad en las comunicaciones entre pods, y la capacidad de observar la actividad de los servicios y el rendimiento general.

Ejemplo

Lo primero que hice fue construir un contenedor de Docker que almacene la aplicación a implementar. Esta es la misma que utilice en el ejemplo de Kubernetes: un programa en Flask que realiza peticiones a una API de pasos aleatorios de distintas publicaciones de WikiHow.

```python
from flask import Flask
import requests
import json

app = Flask (__name__)

@app.route ("/")
def llamadaAPI ():
    # URL de la API.
    url = "https://hargrimm-wikihow-v1.p.rapidapi.com/steps"

    # Numero de pasos a obtener.
    querystring = {"count":"5"}

    # Headers de la peticion.
    headers = {
        "X-RapidAPI-Key": "8070df0992msh7d97d032e4f12fdp15a65bjsnbe724bdba2ea",
        "X-RapidAPI-Host": "hargrimm-wikihow-v1.p.rapidapi.com"
    }

    # Peticion a la API.
    response = requests.request("GET", url, headers=headers, params=querystring)
    respuestaJSON = response.text

    # Formateamos el JSON a un diccionario de Python.
    datos = json.loads (respuestaJSON)

    return f"<div align='left'><h1>Cinco pasos aleatorios de tutoriales de WikiHow

if __name__ == "__main__":
    app.run ()
```

Es necesario crear un Dockerfile y un requirements.txt para poder construir una imagen de Docker exitosamente.

```
FROM python:3.8-buster

RUN mkdir /app

WORKDIR /app/

ADD . /app/

RUN pip install --no-cache-dir -r requirements.txt

CMD [ "gunicorn", "--bind", "0.0.0.0:8000", "main:app" ]
```

requirements.txt

```
flask
requests
gunicorn
```

Usamos el comando *docker build* para construir la imagen.

```
$ docker build -t istio-app .
```

```
istio-app   IN USE   latest   49ba3bcc2246   1 minute ago   899.46 MB
```

Una vez hecho lo anterior, podemos utilizar *minikube* para crear un clúster de Kubernetes localmente.

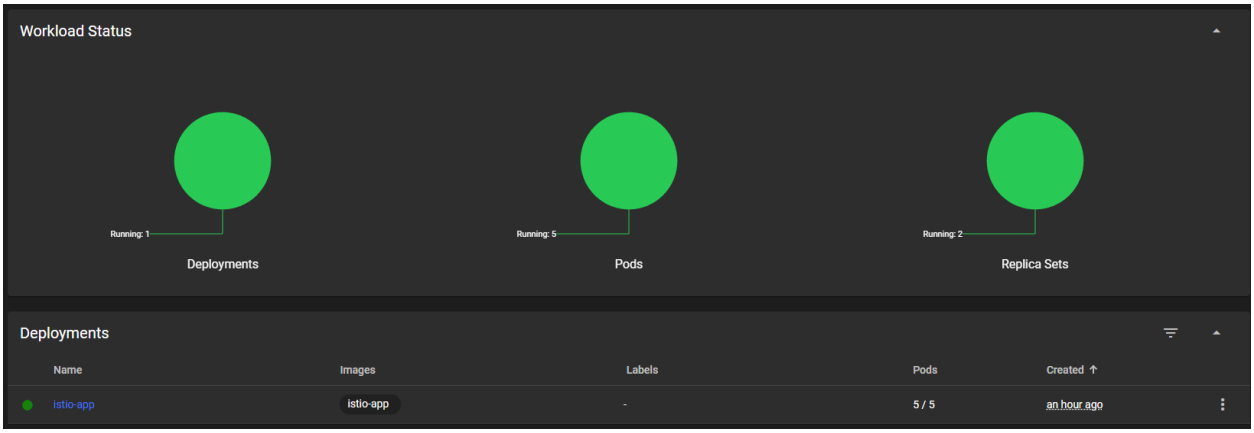
```
$ minikube image load istio-app
```

Luego, aplicamos las configuraciones de despliegue.

```
$ kubectl apply -f ./flask_deployment.yaml
deployment.apps/flask-app created
```

```
$ kubectl get deploy
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
istio-app   5/5     5            5           73m
```

```
$ kubectl get pods
NAME                          READY   STATUS    RESTARTS      AGE
istio-app-557cc89b58-d7dmn    1/1     Running   1 (115s ago)  73m
istio-app-557cc89b58-l42nj    1/1     Running   1 (115s ago)  73m
istio-app-557cc89b58-nfqk7    1/1     Running   1 (115s ago)  73m
istio-app-557cc89b58-t2g77    1/1     Running   1 (115s ago)  73m
istio-app-557cc89b58-w8wqv    1/1     Running   1 (115s ago)  73m
```



Es momento de comenzar a aplicar Istio a la aplicación. Utilizamos el comando *istioctl install* para realizarlo.

```
$ istioctl install
```

```
$ kubectl get pods
NAME                          READY   STATUS    RESTARTS      AGE
istio-app-557cc89b58-d7dmn    1/1     Running   1 (5m22s ago)  77m
istio-app-557cc89b58-l42nj    1/1     Running   1 (5m22s ago)  77m
istio-app-557cc89b58-nfqk7    1/1     Running   1 (5m22s ago)  77m
istio-app-557cc89b58-t2g77    1/1     Running   1 (5m22s ago)  77m
istio-app-557cc89b58-w8wqv    1/1     Running   1 (5m22s ago)  77m

Raúl@LAPTOP-K05TVMDU MINGW64 ~/Desktop/UDG 2022B/Computación Tolerante a Fallas/Programas/Istio
$ kubectl get pods -n istio-system
NAME                                    READY   STATUS    RESTARTS   AGE
istio-ingressgateway-6b989496d5-qpt7w   1/1     Running   0          84s
istiod-67f88486c6-n6sfq                 1/1     Running   0          107s
```

Ahora, hay que configurar los pods del *namespace* para que se les inyecte automáticamente el proxy de *Istio.* Esto se hace etiquetando el *namespace* para que permita la inyección *Istio*.

```
$ kubectl label namespace default istio-injection=enabled
namespace/default labeled
```

Para que los cambios surtan efecto, es necesario reiniciar todos los pods con el siguiente comando:

```
$ kubectl rollout restart deployment -n default
deployment.apps/istio-app restarted
```

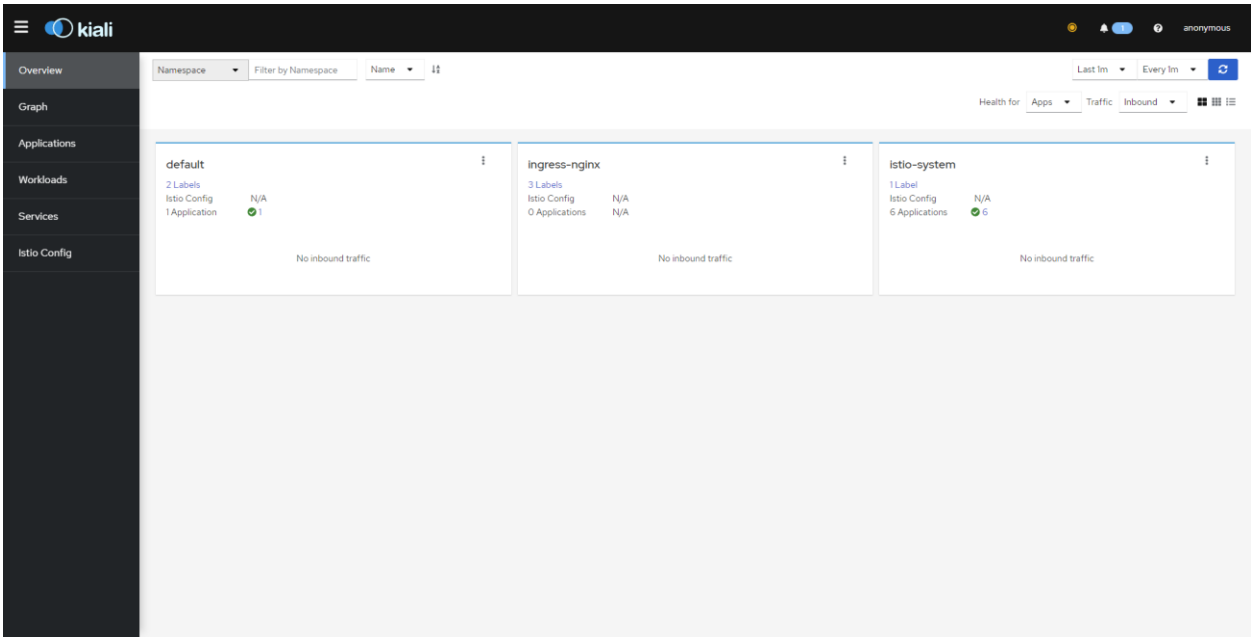
```
$ kubectl get pods -n default
NAME                         READY   STATUS    RESTARTS   AGE
istio-app-7586c5fb89-hq529   2/2     Running   0          15s
istio-app-7586c5fb89-nc9lt   2/2     Running   0          21s
istio-app-7586c5fb89-q82t2   2/2     Running   0          21s
istio-app-7586c5fb89-qxtmq   2/2     Running   0          18s
istio-app-7586c5fb89-txz5n   2/2     Running   0          21s
```

Como vemos, ahora los pods tienen dos contenedores en lugar de uno. Lo único que queda es instalar algunos *add-ons* para monitorizar gráficamente el rendimiento de los microservicios.

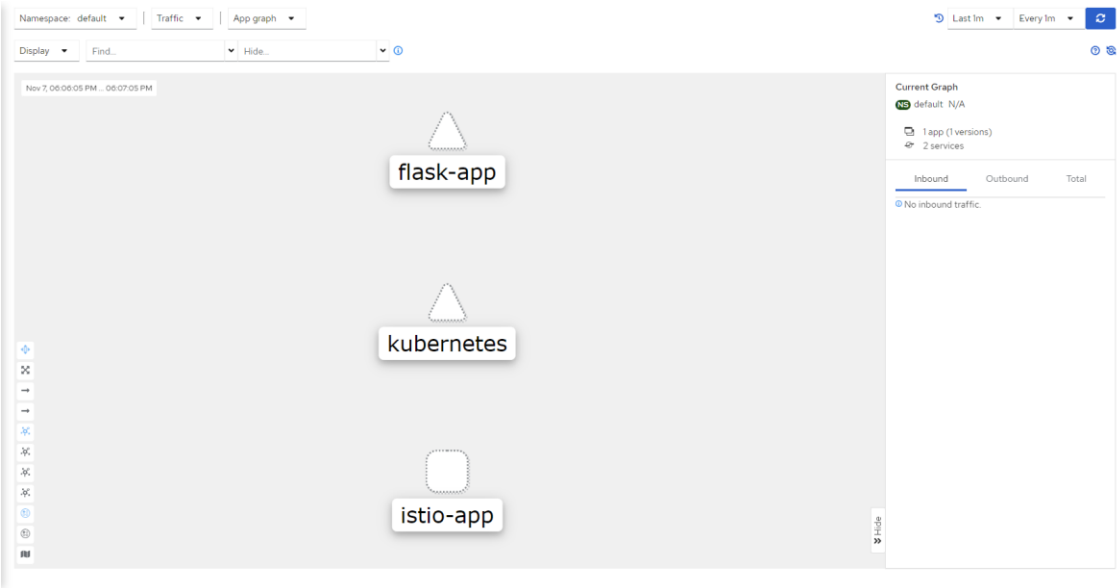
```
$ kubectl get pod -n istio-system
NAME                                    READY   STATUS    RESTARTS   AGE
grafana-56bdf8bf85-j7kjl                1/1     Running   0          2m35s
istio-ingressgateway-6b989496d5-qpt7w   1/1     Running   0          24m
istiod-67f88486c6-n6sfq                 1/1     Running   0          24m
jaeger-c4fdf6674-mfz2h                  1/1     Running   0          2m35s
kiali-5ff49b9f69-xmdfl                  1/1     Running   0          2m34s
prometheus-85949fddb-5jtxr              2/2     Running   0          2m34s
```

Ejecutamos el *dashboard* de *Kiali* con:

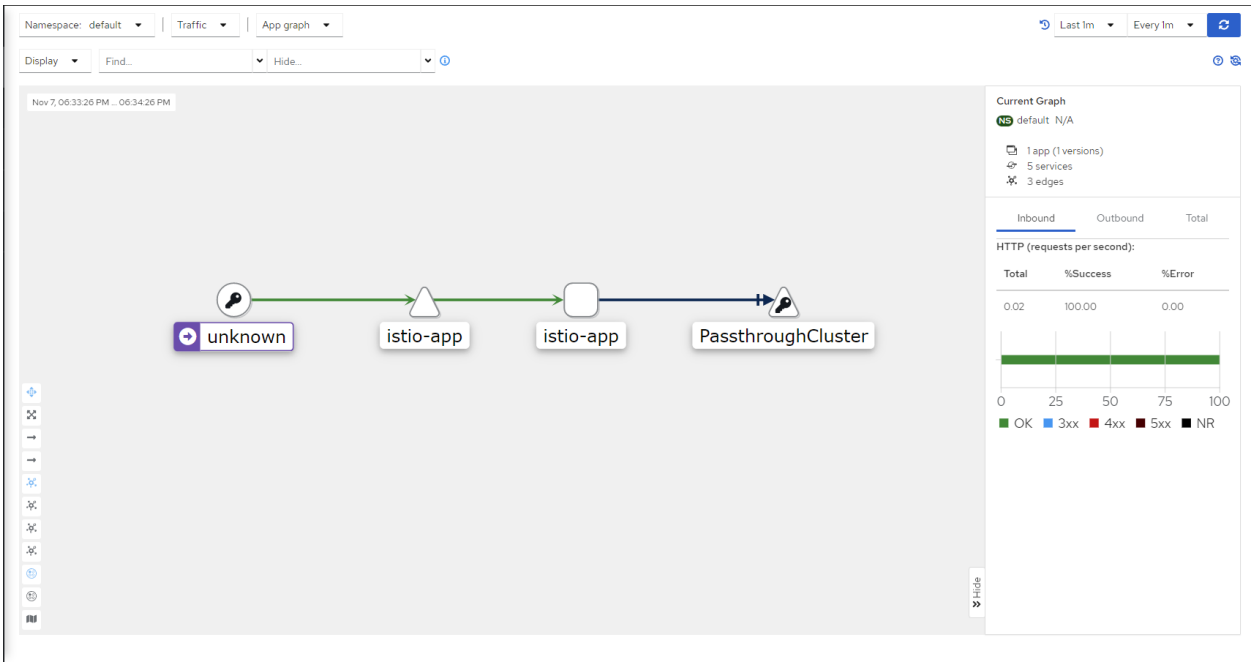
```
$ istioctl dashboard kiali
http://localhost:20001/kiali
```



Accedemos al grafo del programa, que nos muestra los tres elementos principales sin comunicación, ya que no se está realizando ninguna petición.

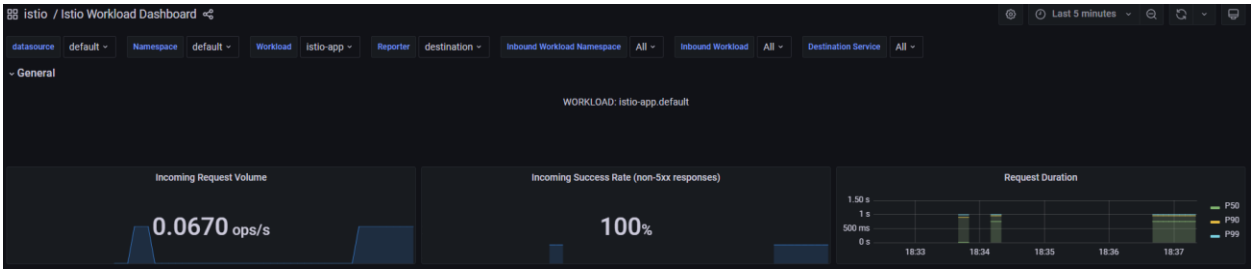


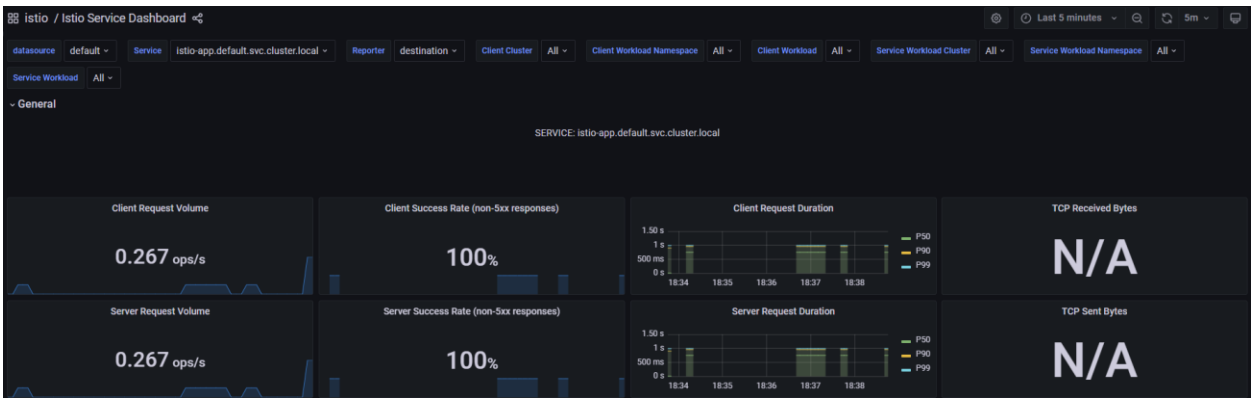
Para verlo en funcionamiento, realizamos una petición.



El nodo *unknown* es la API, y lo marca como desconocido ya que es un agente externo al cluster y es muy complicado identificar el origen de todos. Los demás, son el servicio y la aplicación en Kubernetes.

Podemos usar *grafana*, el otro dashboard de Istio, para ver graficas sobre el rendimiento y las peticiones que realiza la aplicación.





Conclusiones

Sin duda alguna, esta actividad fue todo un reto para mi. Las tecnologías nuevas pueden llegar a ser complejas si no se ha tenido un acercamiento previo a ellas, por lo que intentar implementar una aplicación con microservicios usando Kubernetes me resulto bastante complicado. No obstante, después de leer varias paginas de documentación, y analizar videos y comentarios en foros, pude lograr una aplicación funcional.

Considero que lo aprendido aquí me ayudara bastante a desarrollar el proyecto final, así que estoy satisfecho.

Bibliografía

¿Qué es Istio? (s. f.). https://www.redhat.com/es/topics/microservices/what-is-istio

¿Qué es Istio? |. (s. f.). Google Cloud. https://cloud.google.com/learn/what-is-istio?hl=es