Kubernetes

Nombre del alumno: José Ulises Vallejo Sierra

Código: 219747905

Sección: D06

Curso: Computación Tolerante a Fallas

Nombre del profesor: Michel Emanuel López Franco

¿Qué es Kubernetes?

Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa.

Kubernetes ofrece un entorno de administración centrado en contenedores. Kubernetes orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Esto ofrece la simplicidad de las Plataformas como Servicio (PaaS) con la flexibilidad de la Infraestructura como Servicio (IaaS) y permite la portabilidad entre proveedores de infraestructura.

Algunas características de kubernetes son:

- una plataforma de contenedores
- una plataforma de microservicios
- una plataforma portable de nube

¿Por qué usar kubernetes?

Los flujos de trabajo de las aplicaciones pueden optimizarse para acelerar el tiempo de desarrollo. Una solución de orquestación propia puede ser suficiente al principio, pero suele requerir una automatización robusta cuando necesita escalar. Es por ello por lo que Kubernetes fue diseñada como una plataforma: para poder construir un ecosistema de componentes y herramientas que hacen más fácil el desplegar, escalar y administrar aplicaciones.

Las etiquetas, o Labels, les permiten a los usuarios organizar sus recursos como deseen. Las anotaciones, o Annotations, les permiten asignar información arbitraria a un recurso para facilitar sus flujos de trabajo y hacer más fácil a las herramientas administrativas inspeccionar el estado.

Además, el Plano de Control de Kubernetes usa las mismas APIs que usan los desarrolladores y usuarios finales. Los usuarios pueden escribir sus propios controladores, como por ejemplo un planificador o scheduler, usando sus propias APIs desde una herramienta de línea de comandos.

¿Qué es Ingress? Ingress se usa para exponer rutas HTTP y HTTPS desde el exterior a servicios dentro del Cluster. El tráfico entrante es controlado por las reglas que definimos en un archivo de configuración.

Una razón es que por cada servicio del tipo LoadBalancer se necesita aprovisionar un balanceador de carga, y eso puede suponer un sobrecoste innecesario. Además, cada balanceador de carga requiere su propia dirección IP, en cambio, con Ingress una IP es suficiente. Por último, los Ingreeses operan en la séptima capa de red (http) y pueden proveer funcionalidades como las sesiones por Cookie, cosa que los demás servicios no pueden hacer.

Controladores Ingress para kubernetes

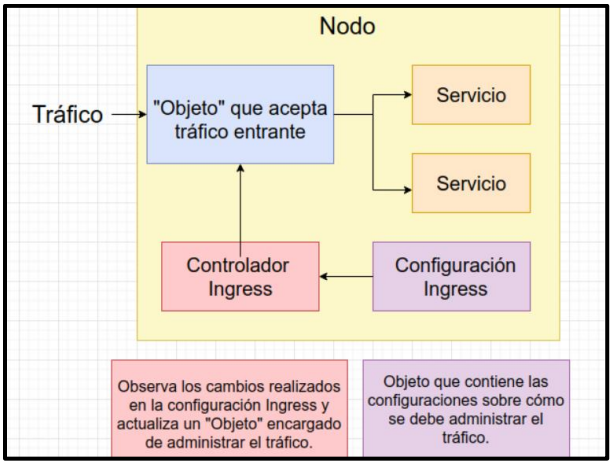
Para que el recurso de entrada funcione, el clúster debe tener un controlador de entrada en ejecución. A diferencia de otros tipos de controladores que se ejecutan como parte del kube-controller-managerbinario, los controladores Ingress no se inician automáticamente con un clúster. Un controlador de Ingress es un Pod o conjunto de Pods que se ejecutan en nuestro Cluster y cuya función es asegurarse de que el tráfico entrante se administra del modo que nosotros hayamos especificado.

Kubernetes como proyecto admite y mantiene los controladores de entrada de

- AWS
- GCE
- Nginx

Componentes de Ingress

Para que Ingress funcione en nuestro Cluster necesitamos dos cosas. La primera es el controlador, que ya hemos visto, y la segunda es una configuración Ingress.Esta configuración es objeto de Kubernetes que usamos para describir dónde dirigir el tráfico entrante. El diagrama de más abajo muestra de forma general cómo funciona Ingress:



¿Qué es LoadBalancer?

Cuando creamos un servicio del tipo LoadBalancer, automáticamente se aprovisiona un balanceador de carga externo al Cluster que nos proporciona una dirección IP a través de la que acceder a nuestros servicios.

Para conseguir eso, Kubernetes se sincroniza con la API del proveedor Cloud que estemos usando. Por cada servicio que creamos se aprovisiona un balanceador de carga diferente. Cuando el servicio es eliminado, el balanceador de carga que tiene asociado también es eliminado.

Si Kubernetes está corriendo en un entorno que no da soporte a LoadBalancer, como es el caso con Minikube, el balanceador de carga asociado no va a ser aprovisionado, pero el servicio sí que va a funcionar. Se va a comportar como un servicio NodePort.

Eso es porque el servicio LoadBalancer, por debajo, es un NodePort con la capacidad de comunicarse con una API exterior para aprovisionar un balanceador de carga.

¿Cuándo usar LoadBalancer?

Si quieres exponer un servicio en producción, este es el método por defecto. Todo el tráfico del puerto que especifiques será enviado al servicio.

La parte negativa es que no se puede filtrar y redirigir el tráfico entrante. Además, cada servicio expuesto a través de un LoadBalancer recibe su propia IP y tienes que pagar por los balanceadores de carga que Kubernetes haya aprovisionado.

¿Qué es Rancher?

Al ser de código abierto, Kubernetes tiene varias distribuciones entre las que puedes elegir si pretendes desplegar cargas de trabajo en la nube. Una de las distribuciones que elige es Rancher.

Rancher es una pila de software que se utiliza para gestionar clústeres Kubernetes. Se trata básicamente de un software que DevOps puede utilizar al adoptar el usuario de contenedores. Rancher incluye una distribución completa de Kubernetes, Docker Swarm y Apache Mesos, lo que facilita la gestión de clústeres de contenedores en cualquier plataforma de nube. Algunas de las empresas populares que utilizan Rancher son: Alibaba travelers, Abeja, Trivago, UseInsider, Starbucks, Oxylabs, yousign, y muchas más.

¿Por qué usar Rancher?

Una de las ventajas significativas de Rancher es la capacidad de gestionar múltiples clústeres Kubernetes de forma simplificada. Ofrece una gestión simplificada de múltiples clústeres Kubernetes que pueden ser creados manualmente utilizando la distribución de Kubernetes de Ranchers llamada RKE (Rancher Kubernetes Engine) o importados en el panel de gestión del gestor de clústeres.

Además de Rancher Kubernetes Engine (RKE), Rancher ha iniciado otros proyectos innovadores, y uno de ellos es el K3S, un panel de control de Kubernetes más sencillo que se utiliza principalmente en la computación de borde.

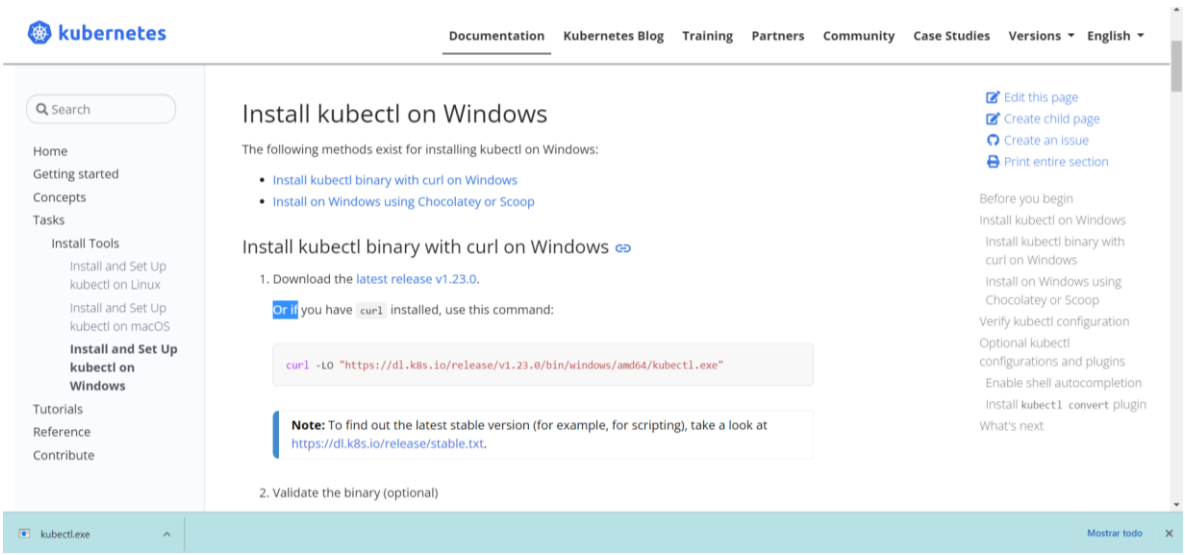
Principales características de Rancher

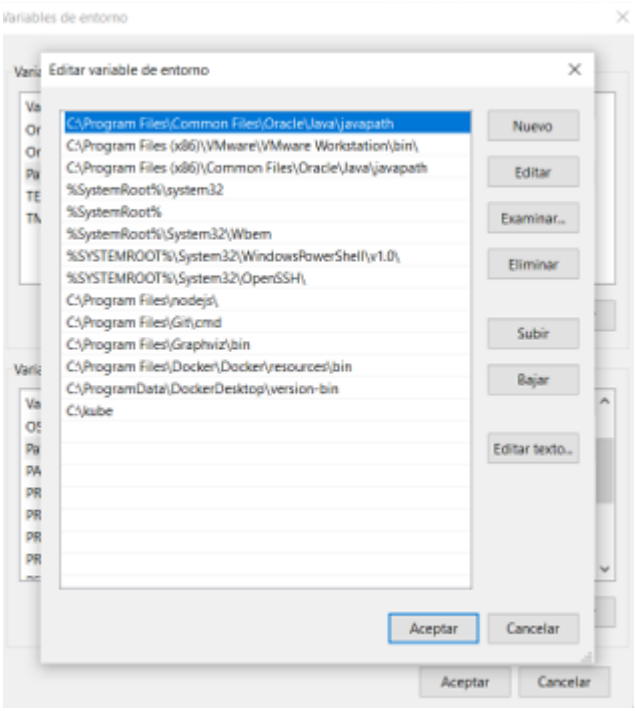
- Gestión de hosts, despliegue de contenedores, supervisión de recursos
- Gestión de usuarios y colaboración
- APIs y herramientas nativas de Docker
- Control y registro
- Conectar contenedores, gestionar discos, desplegar balanceadores de carga

Genera un ejemplo similar al vídeo:

Técnica para utilizar: Kubernetes con Minikube

Primero instalamos los componentes necesarios, empezamos con kubectl en la página oficial de kubernetes y una vez descargado agregamos la ruta al path.

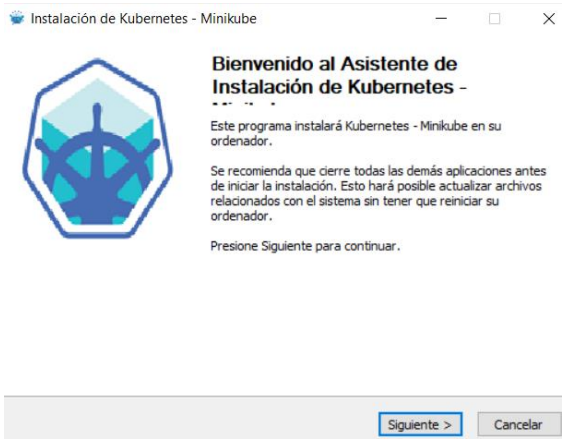




Después abrimos el powershell como administrador para verificar que detecte los comandos relacionados a kubectl y comprobar su instalación

```
PS C:\Users\ulise> kubectl version --client
Client Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.5", GitCommit:"5c99e2ac2ff9a3c549d9ca665e7bc05a3e1
8f07e", GitTreeState:"clean", BuildDate:"2021-12-16T08:38:33Z", GoVersion:"go1.16.12", Compiler:"gc", Platform:"windows/
amd64"}
PS C:\Users\ulise>
```

Ahora descargamos minikube desde la página oficial y realizamos el proceso de instalación



Mediante los comandos sugeridos, también agregamos al Path el bin de minikube mediante el powershell como administradores y comprobamos la instalación

```
PS C:\WINDOWS\system32> $oldPath = [Environment]::GetEnvironmentVariable('Path', [EnvironmentVariableTarget]::Machine)
>> if ($oldPath.Split(';') -inotcontains 'C:\minikube'){ `
>>   [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube' -f $oldPath), [EnvironmentVariableTarget]::Machin
e) `
>> }
PS C:\WINDOWS\system32>
```

```
PS C:\WINDOWS\system32> minikube
minikube provisions and manages local Kubernetes clusters optimized for development workflows.

Basic Commands:
  start          Starts a local Kubernetes cluster
  status         Gets the status of a local Kubernetes cluster
  stop           Stops a running local Kubernetes cluster
  delete         Elimina un cluster de Kubernetes local
  dashboard      Acceder al panel de Kubernetes que corre dentro del cluster minikube
  pause          pause Kubernetes
  unpause        unpause Kubernetes
```

Una vez hecho esto, y con Docker y Flask previamente instalados para la realización de actividades anteriores, podemos empezar a trabajar

Creamos un archivo en visual llamado app.py donde crearemos una app sencilla de demostración para una API sencilla con flask

app.py > ...

```python
from flask import Flask, jsonify
import time

app = Flask(__name__)

app.route("/")
def hello_world():
    return jsonify({"Time to call": time.time()})

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True)
```

Ahora creamos el archivo para los requirements para colocar las dependencias que en nuestro caso solo es flask

requirements.txt

```
flask
```

Para crear la imagen para el contenedor de la aplicación, creaamos un Dockerfile y pasamos las aplicaciones y versiones que el contenedor creará

Dockerfile

```
Dockerfile > ...
1   FROM python:3.7
2   RUN mkdir /app
3   WORKDIR /app/
4   ADD . /app/
5   RUN pip install -r requirements.txt
6   CMD ["python", "/app/app.py"]
```

Después construimos la imagen con el nombre de flask-kubernetes y comprobamos que haya sido creada con éxito

```
C:\Users\ulise\Escritorio\kubernetes>docker build -t flask-kubernetes .
[+] Building 12.0s (5/10)
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 164B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.7
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 482B
 => [1/5] FROM docker.io/library/python:3.7@sha256:55d468198d18e6878f9f029b5efed8a985c63f911ede599c48a3784805a90890
 => => resolve docker.io/library/python:3.7@sha256:55d468198d18e6878f9f029b5efed8a985c63f911ede599c48a3784805a90890
 => => sha256:dbba69284b2786013fe94fefe0c2e66a7d3cecbb20f6d691d71dac891ee37be5 29.36MB / 54.94MB
 => => sha256:9baf437a1badb6aad2dae5f2cd4a7b53a6c7ab6c14cba1ed1ecb42b4822b0e87 5.16MB / 5.16MB
 => => sha256:6ade5c59e324bd7cf369c72ad781c23d37e8fb48c9bbb4abbecafafd9be4cc35 4.19MB / 10.87MB
 => => sha256:55d468198d18e6878f9f029b5efed8a985c63f911ede599c48a3784805a90890 1.86kB / 1.86kB
```

```
C:\Users\ulise\Escritorio\kubernetes>docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
flask-kubernetes    latest    6cffbcd7a99e   10 minutes ago   917MB
imgdocker1          latest    45b258797ed7   2 weeks ago      135MB
```

Ahora, creamos el archivo deployment.yaml para implementar el contenedor para la aplicación en el motor de kubernetes, la extensión se usa para compatibilidad con archivos Docker para kubernetes

Se estructura en dos partes, la primera especifica datos de la aplicación (versión de la api, servicio, metadata, protocolo, puerto, tipo, LoadBalancer, etc), este último, es un servicio que funcionará como equilibrador de carga.

deployment.yaml Dockerfile requirements.txt

deployment.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-test-service
spec:
  selector:
    app: flask-test-app
  ports:
  - protocol: "TCP"
    port: 6000
    targetPort: 5000
  type: LoadBalancer
```

En la segunda parte, trata sobre el deployment el cuál servirá como aplicación para que el usuario presione el LoadBalancer y le distribuya la solicitud en la aplicación.

Aquí se definen las replicas para la escalabilidad por si una instancia bloquea a otra y además, se especifica la imagen “flask-kubernetes” para desplegarla al cluster de Kubernetes.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-test-app
spec:
  selector:
    matchLabels:
      app: flask-test-app
  replicas: 5
  template:
    metadata:
      labels:
        app: flask-test-app
    spec:
      containers:
      - name: flask-test-app
        image: flask-kubernetes
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 5000
```

Ahora usamos el comando start de minikube se puede usar para iniciar el clúster. Y creará y configurará una máquina virtual que ejecuta un clúster de Kubernetes de un solo nodo.

```
C:\Users\ulise\Escritorio\kubernetes>minikube start
😄  minikube v1.25.2 en Microsoft Windows 10 Home Single Language 10.0.19044 Build 19044
✨  Controlador docker seleccionado automáticamente. Otras opciones: virtualbox, ssh
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
💾  Descargando Kubernetes v1.23.3 ...
    > gcr.io/k8s-minikube/kicbase: 379.06 MiB / 379.06 MiB  100.00% 2.16 MiB p/
    > preloaded-images-k8s-v17-v1...: 505.68 MiB / 505.68 MiB  100.00% 2.49 MiB
🔥  Creando docker container (CPUs=2, Memory=2200MB) ...| E0404 19:59:06.969083    1452 kic.go:267] ica
tput - [archivo procesado: C:\Users\ulise\.minikube\machines\minikube\id_rsa
Se procesaron correctamente 1 archivos; error al procesar 0 archivos]

🐳  Preparando Kubernetes v1.23.3 en Docker 20.10.12...
    ▪ kubelet.housekeeping-interval=5m
    ▪ Generando certificados y llaves
    ▪ Iniciando plano de control
    ▪ Configurando reglas RBAC...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Complementos habilitados: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

C:\Users\ulise\Escritorio\kubernetes>
```

Después de esto, ya podemos usar el comando apply que desplegara el servicio y las instancias de la aplicación en kubernetes engine (motor).

```
C:\Users\ulise\Escritorio\kubernetes>kubectl apply -f deployment.yaml
service/flask-test-service created
deployment.apps/flask-test-app created
```

Ahora bien, nos logeamos en Docker para pasar nuestro usuario y la imagen como parámetro para kubernetes y hacer que funcione el servicio de minikube para visualizar mis tags (deployments, pods y replicas) corriendo desde el servidor.

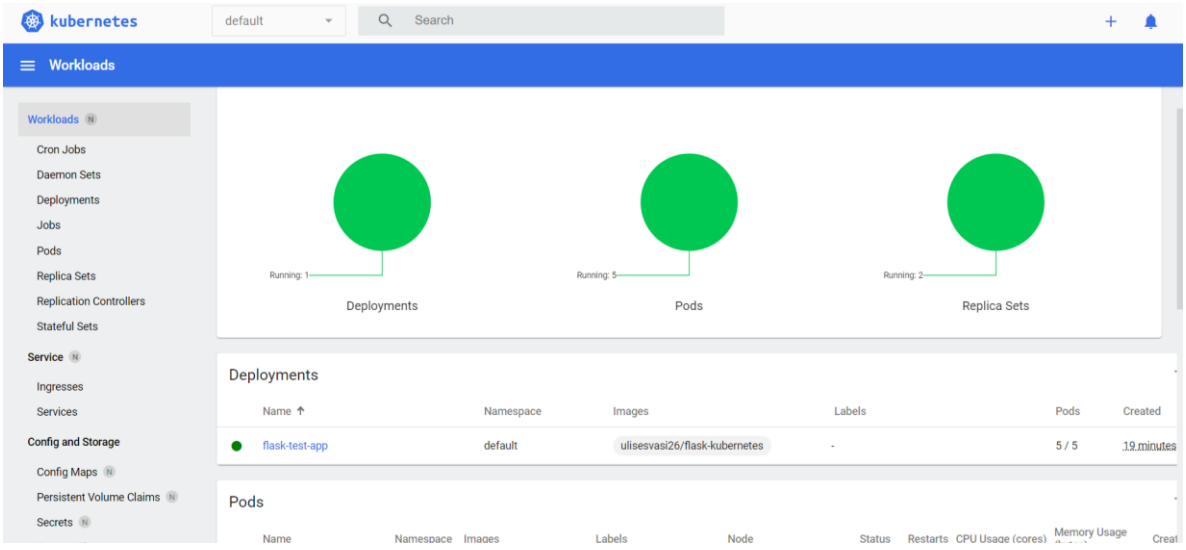
```
C:\Users\ulise\Escritorio\kubernetes>docker tag 6cffbcd7a99e ulisesvasi26/flask-kubernetes

C:\Users\ulise\Escritorio\kubernetes>docker push ulisesvasi26/flask-kubernetes
Using default tag: latest
The push refers to repository [docker.io/ulisesvasi26/flask-kubernetes]
46169095553d: Pushed
2319a73c989c: Pushed
5f70bf18a086: Pushed
71ca3a3c34d2: Pushed
f5e951a0e3c7: Mounted from library/python
3d8c09d52b7b: Mounted from library/python
39ca3c789e6d: Mounted from library/python
89f49a7a4e4a: Mounted from library/python
c5579a205adc: Mounted from library/python
7a7698da17f2: Mounted from library/python
d59769727d80: Mounted from library/python
348622fdcc61: Mounted from library/python
4ac8bc2cd0be: Mounted from library/python
latest: digest: sha256:83b72159cbb1e094d7d5b24b20fc7cdfed35a9e4d9424b16b88411cfc877a535 size: 3049
```

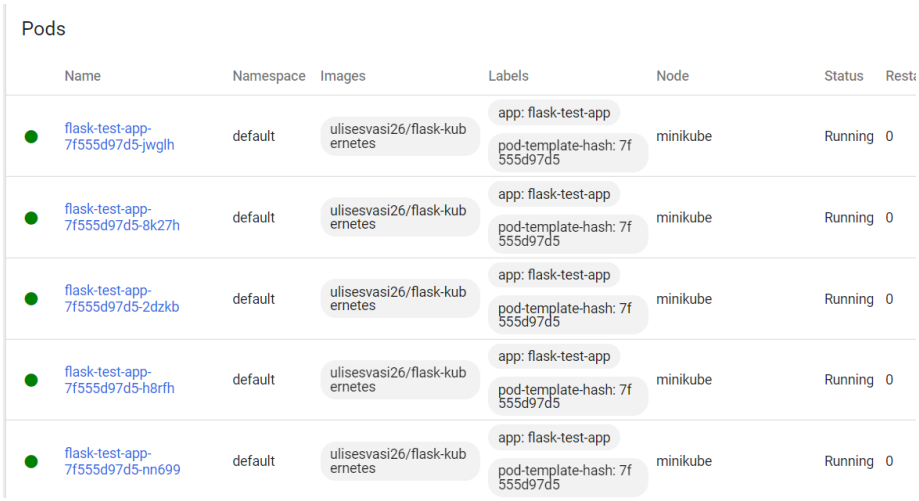
Finalmente, con el comando dashboard de minikube se abre el mismo de manera local ya con la imagen subida al repositorio y la aplicación corriendo y lo comprobamos abriendo el sitio

```
C:\Users\ulise\Escritorio\kubernetes>minikube dashboard
    Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 2.3017705s
    Restarting the docker service may improve performance.
    Verifying dashboard health ...
    Launching proxy ...
    Verifying proxy health ...
    Opening http://127.0.0.1:49693/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```

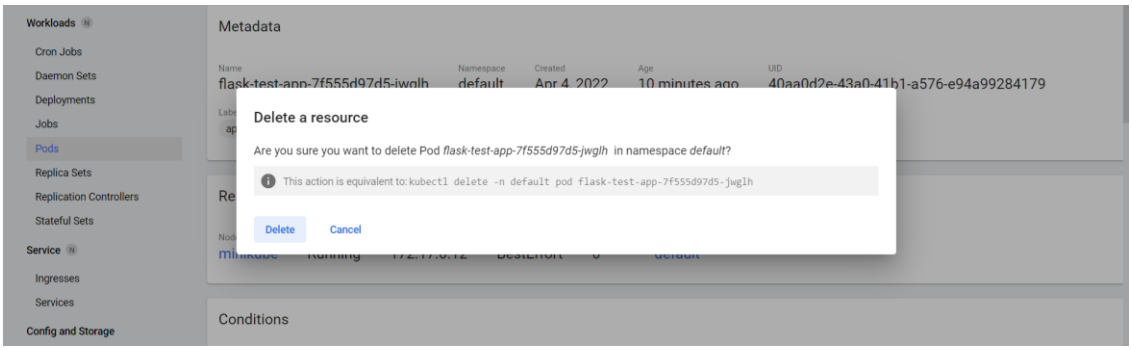
Como se observa, Los pods son los objetos más pequeños y básicos que se pueden implementar en Kubernetes y representan una instancia única de un proceso en ejecución en el clúster, donde cada uno contiene uno o más contenedores.

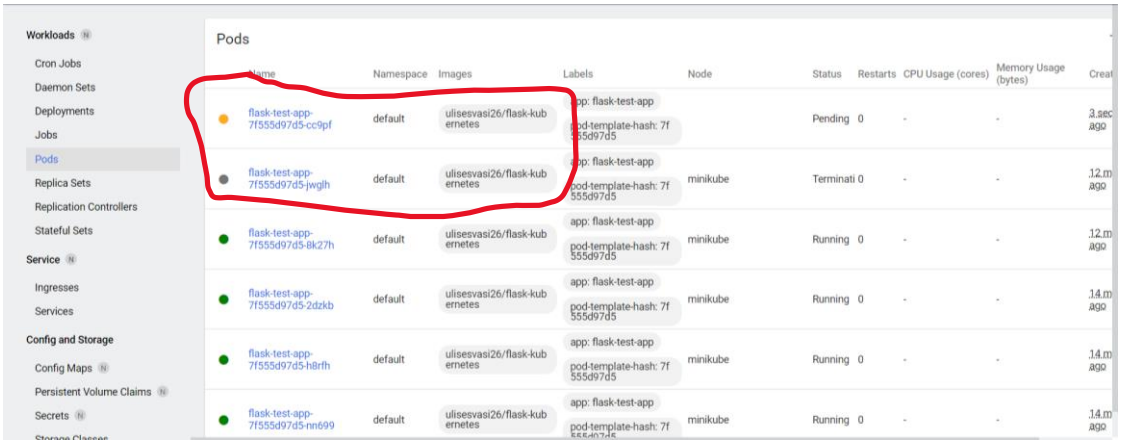
Estas son las instancias de la aplicación asociadas con el LoadBalancer y este distribuirá el número de api calls a las instancias de la app (pods) equitativamente.

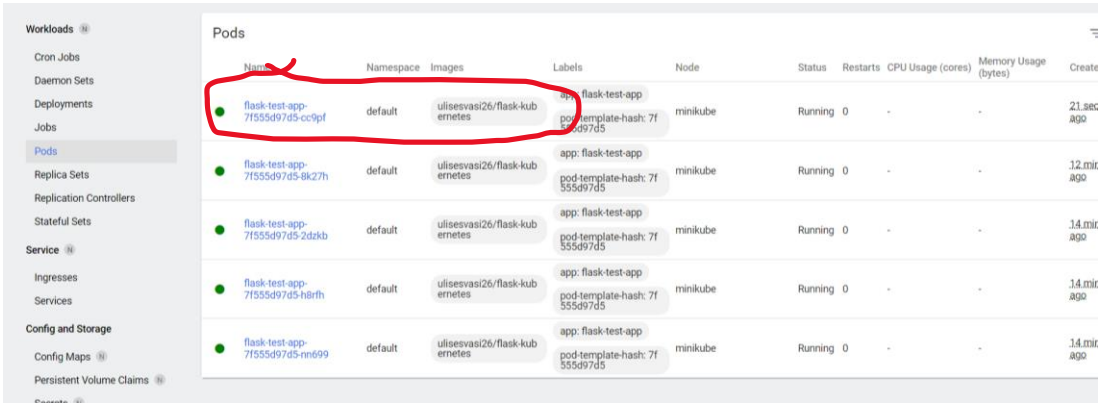




Esto nos permite eliminar alguna de estas instancias, y lo que hará kubernetes con la implementación de minikube será restaurar dichas instancias creandolas desde su especificación en deployment.yaml y con ayuda de la llamada a LoadBalancer.







Comprobamos dicha creación y restauración con el comando get pods

```
C:\Users\ulise\Escritorio\kubernetes>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
flask-test-app-7f555d97d5-2dzkb     1/1     Running   0          12m
flask-test-app-7f555d97d5-8k27h     1/1     Running   0          10m
flask-test-app-7f555d97d5-h8rfh     1/1     Running   0          12m
flask-test-app-7f555d97d5-jwglh     1/1     Running   0          10m
flask-test-app-7f555d97d5-nn699     1/1     Running   0          12m
```

Conclusión

Hoy en día, la creación de aplicaciones esencialmente debe ser colaborativa, por lo que es necesario adaptar dichas aplicaciones y sus módulos o recursos para poder satisfacer este trabajo colaborativo, evitando así errores de versiones o compatibilidad para el desarrollo de una aplicación por mas de un programador; para ello se han implementado novedosas soluciones como el uso de contenedores, que permiten mitigar, prevenir y evitar estos errores logrando que los programas y versiones para las aplicaciones sean previamente definidas y utilizadas en ese esquema sobre cualquier sistema.

La utilización de herramientas con contenedores como Docker, así como la implementación en dichas aplicaciones de kubernetes con minikube que permite procesos de control que monitorean el estado de los nodos, pods y contenedores disponibles en el clúster con una distribución reducida, han traído consigo grandes beneficios en la programación tolerante a fallas como la obtención, creación y recuperación por medio de LoadBalancer y deployment de instancias de tareas que pueden significar un gran alivio y ahorro de tiempo, dinero y esfuerzo para los programadores de hoy en día.

Esta actividad me ha resultado interesante ya que nos plantea un nuevo modo de enfrentar la tolerancia a fallas mediante aplicaciones que inconscientemente creamos creyendo que los errores no tendrán cabida alguna, pero que siempre están susceptibles a las mismas. Enriquecí mi conocimiento en el tema de kubernetes y su implementación en la tolerancia a fallas, aunque me queda claro que estas actividades son aún más provechosas al implementarse en proyectos grandes y colaborativos.

Bibliografías

- Ingress Controllers. (s/f). Kubernetes. Recuperado el 5 de abril de 2022, de https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
- ¿Qué es Kubernetes? (s/f). Kubernetes. Recuperado el 5 de abril de 2022, de https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/
- Sense, D. [DevSense19]. (2020, agosto 22). Deploying Any Dockerized Application To Kubernetes | Ashutosh Hathidara | #kubernetes. Youtube. https://www.youtube.com/watch?v=XQNNAeyMAkk
- sixe. (2021, mayo 4). Todo lo que necesita saber sobre Rancher: gestión de Kubernetes para empresas. SiXe Ingeniería. https://sixe.es/noticias/suse-rancher-kubernetes-toda-la-informacion

Link al repositorio: https://github.com/UlisesVallejo/kubernetes.git