**APP-E-CommercePlugins**

¿Cómo ejecutar el proyecto APP-E-CommercePlugins con Spring Cloud Kubernetes?

**Requisitos:**

Tener instalado Docker, Kubernetes y configurar Kubectl en Kubernetes.

<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

**Notas:**

Es importante que los archivos yaml de deployments (orders y maintenance) la imagen la tengan con la etiqueta latest para forzar la descarga, ejemplo:

*containers:  
- image: alexanderpe/appmic-e-commerceplugins-maintenance:latest*

Spring Cloud necesita permiso a los recursos del API de Spring Cloud Kubernetes, esto requiere permisos para recuperar la lista de accesos de cada pod, puerto que se ejecutan en los servicios. Recuerda que cada servicio ejecuta varias instancias de pods. Entonces lo más fácil sería crear un cluster role binding.

*kubectl create clusterrolebinding admin --clusterrole=cluster-admin --serviceaccount=default:default*

Recuerda que para que el **balanceo de carga** funcione correctamente en Kubernetes, es necesario **tener múltiples réplicas** de tu pod, ya que el balanceador de carga distribuirá el tráfico entre las diferentes instancias de tus pods. Ejemplo:

spec:  
 replicas: 3

Se necesita levantar en orden los archivos YAML, ya que otros recursos lo podrían necesitas, el orden correcto para levantarlo sería primero los Secrets, ConfigMaps, luego sería el **PersistentVolume** **(PV)** y **PersistentVolumeClaim(PVC),** finalmente sería los **Deployments** y los **Services**.

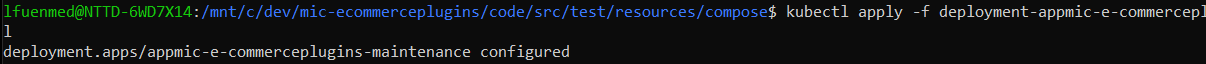
**Comando básicos.**

Kubectl get pods -> Consigue todos los pods.

Kubectl describe pod <Nombre del pod> -> Describe el pod

Kubectl logs <Nombre del pod> -> Consigue los logs del pods

Cualquier cambio en archivo Yaml, será necesario aplicar nuevamente el recurso con el comando kubectl apply -f <nombre-del-recurso>



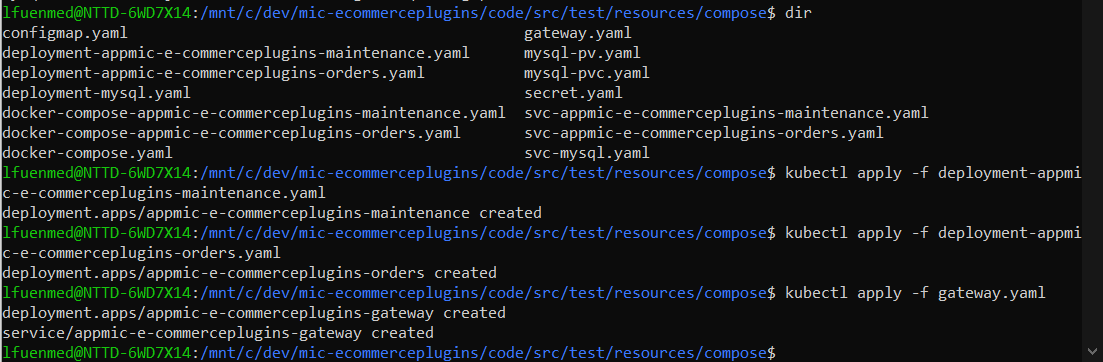
En este caso, se puede observar que hay tres pods ya que en las replicas hemos puesto 3 para probar el balanceo de carga con el controlador de marcas.

Texto

Descripción generada automáticamente

Kubectl delete -f deployment-appmic-e-commerceplugins-maintenance.yaml

Kubectl delete -f deployment-appmic-e-commerceplugins-orders.yaml

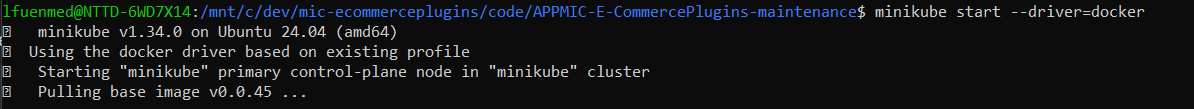


minikube service appmic-e-commerceplugins-maintenance --url -> Obtiene la URL del servicio, devolvería algo como esto. http://192.168.49.2:32000

minikube service --all -> Se ejecutan todos los servicios

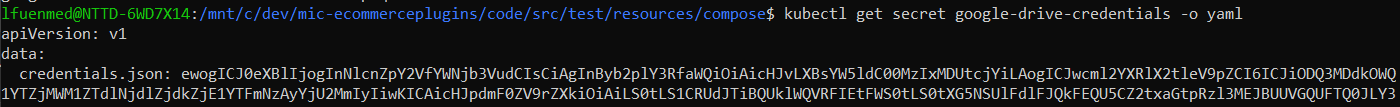
docker rmi <ID de la Imagen> -> Elimina la imagen de Docker

**Levantar Minikube en WSL con Docker como driver**



**Ver los recursos creados como Deployments, Secrets, ConfigMaps, etc.**

Kubectl get secrets, kubectl get configmaps, kubectl get deployments etc.

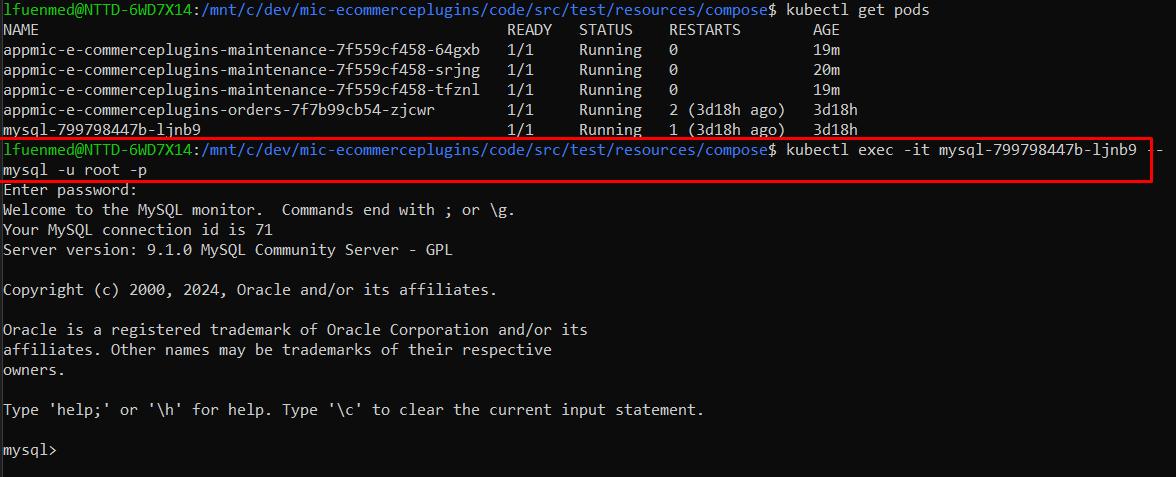
Para ver más detalles de los secretos, debes usar este comando.   


**Base de Datos**

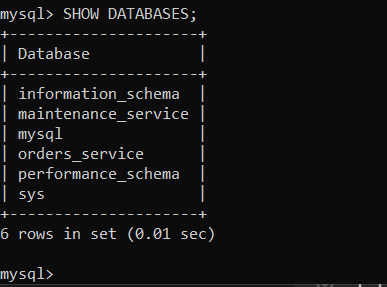
Para acceder al contenedor de MYSQL debemos ejecutar este comando.

kubectl exec -it mysql-799798447b-ljnb9 -- mysql -u root -p

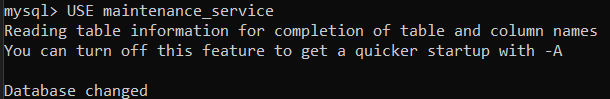
cuando le demos enter, escribimos la contraseña para conectarnos.



Verificar Base de Datos.



Verificar Tablas



Forma

Descripción generada automáticamente con confianza media

Realizar un Select a una tabla en particular.

Texto

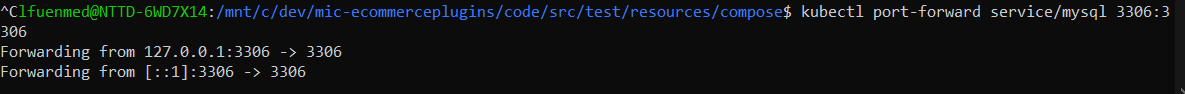
Descripción generada automáticamente

**Para acceder a MySQL desde tu maquina local sin cambiar el tipo de servicio, debes ejecutar este comando.**

Si no deseas cambiar el tipo de servicio y prefieres mantener el servicio como ClusterIP, puedes realizar un port-forward a nivel de servicio, lo cual es útil cuando deseas exponer el puerto de un servicio sin necesidad de modificar la configuración del servicio. El comando sería algo así:

kubectl port-forward service/mysql 3306:3306

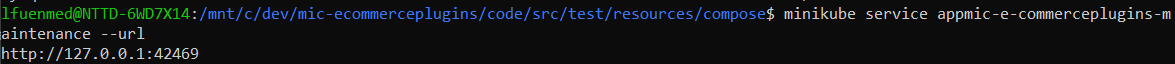
Esto hará que el puerto 3306 del servicio mysql sea accesible en tu máquina local en localhost:3306.



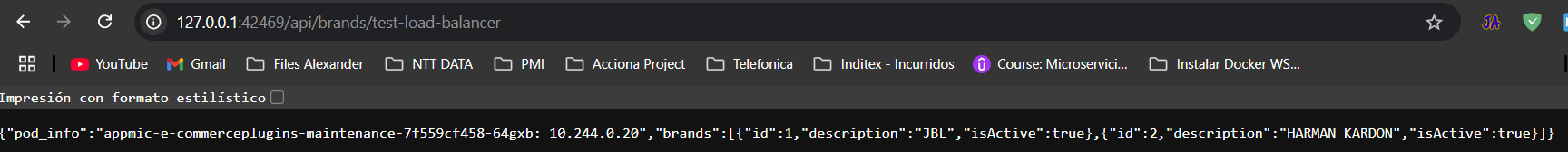
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

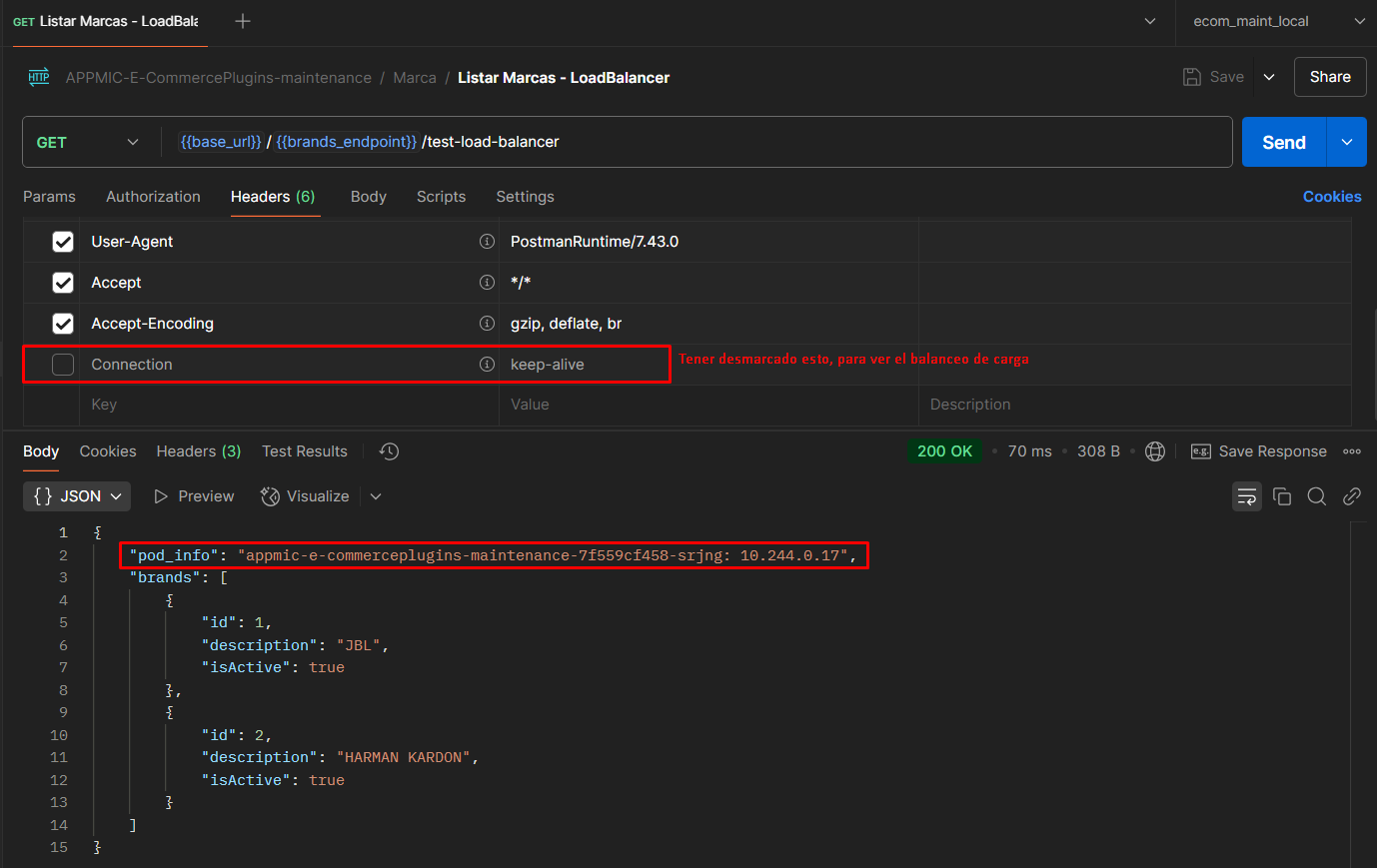
**Balanceo de carga.**



Probamos desde el navegador el balanceo de carga. Si te puedes dar cuenta cada vez que recargamos la página, el pod\_info cambia.



En Postman recuerda tener desactivada la opción de Connection para poder el balanceo de carga.



**Consideraciones importantes.**

**Por qué no usar getResourceAsStream?**

1. **El archivo no está en el classpath:**
   * getResourceAsStream solo funciona para archivos dentro del classpath (generalmente en src/main/resources).
   * Los secretos montados en Kubernetes están en un directorio externo (por ejemplo, /etc/credentials), por lo que no están en el classpath.
2. **Mejor manejo de rutas absolutas:**
   * Usar FileInputStream(credentialsPath) permite acceder a cualquier archivo en el sistema de archivos, incluso si no está en el classpath.
3. **Flexibilidad para Kubernetes:**
   * Este enfoque es más compatible con entornos donde el archivo se monta externamente

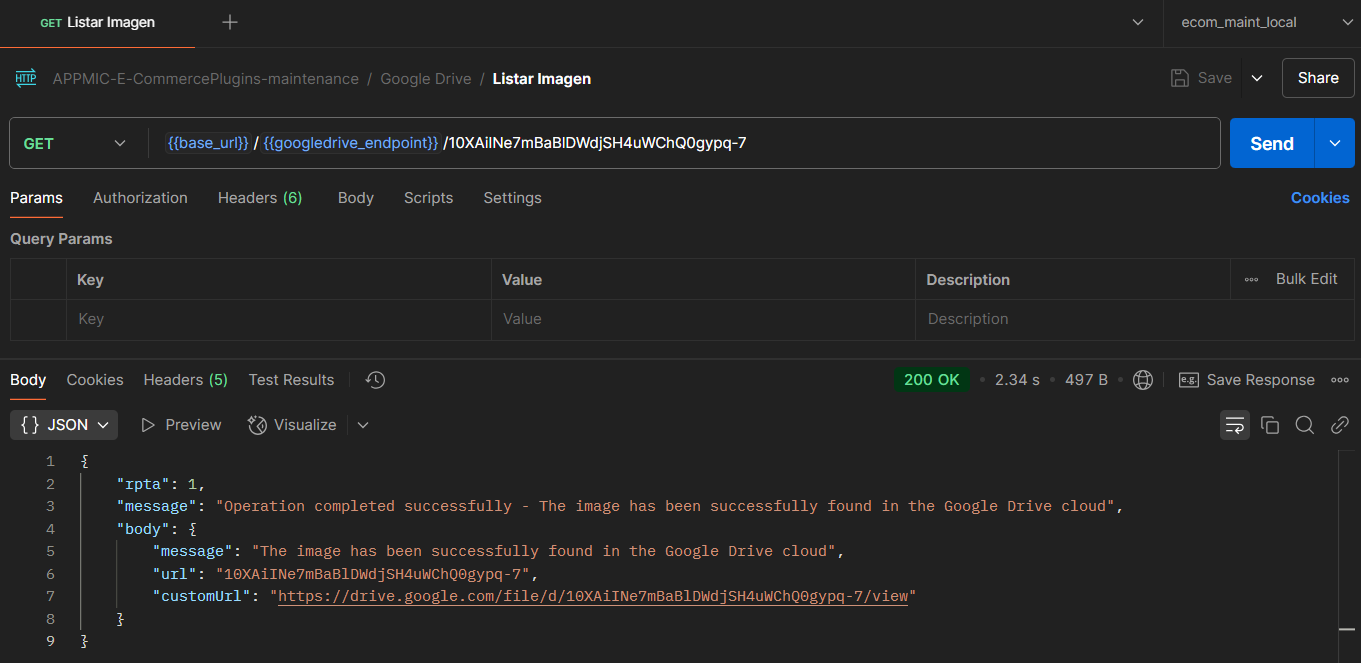
**Nota:** Aquí está el código por si se desea cargar el archivo de configuración desde el ***aplication.properties***

From application.properties  
final InputStream in = GoogleDriveService.class.getResourceAsStream(this.maintenanceConfigBean.getCredentialsPath());  
 if (Objects.isNull(in)) {  
 throw new MaintenanceException(MaintenanceErrorMessage.ERROR\_RESOURCE\_NOT\_FOUND);

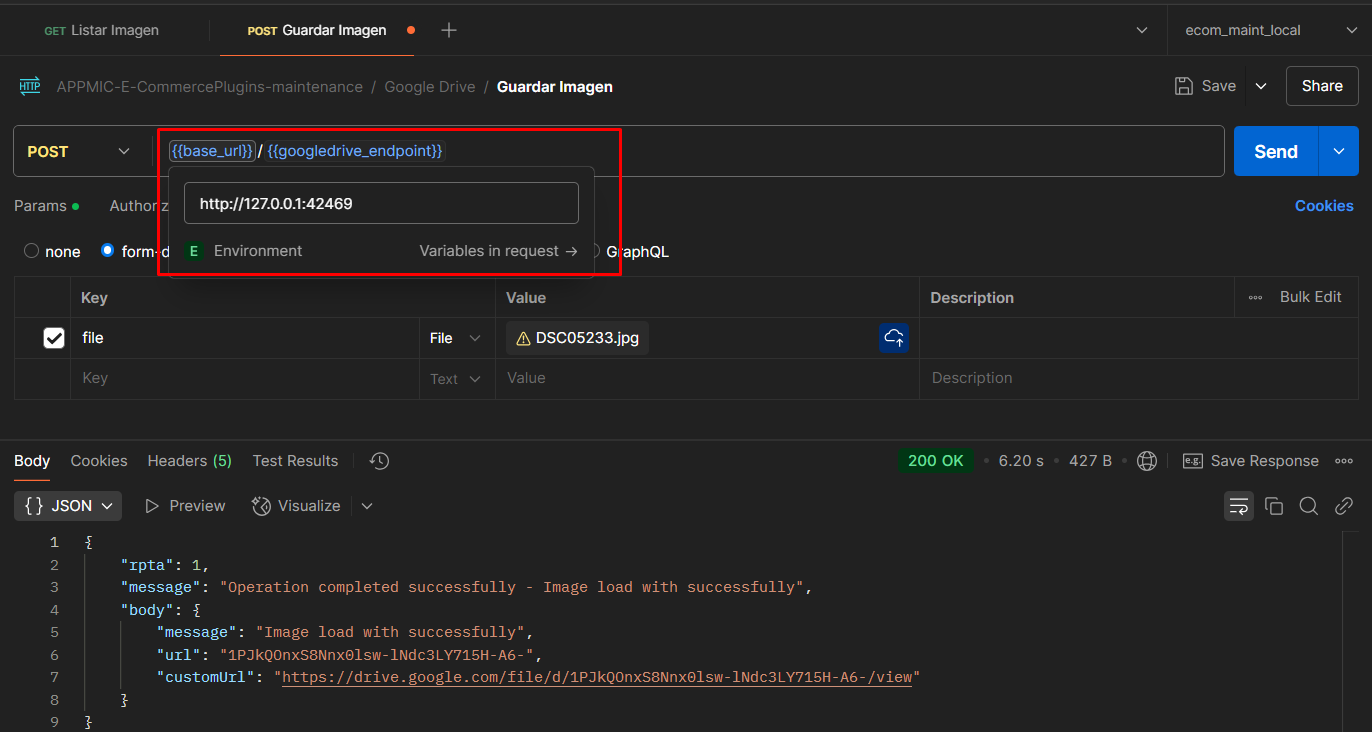
}

Ahora el listado de imágenes ya funciona desde la variables de ambiente de Kubernetes.

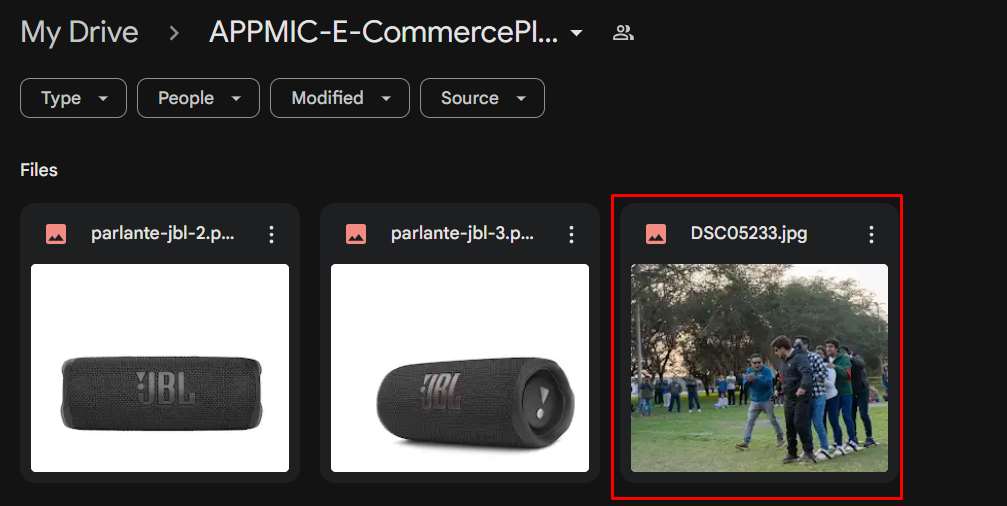
Aquí estamos listando una imagen.



Guardamos una imagen.



Y si nos vamos al navegador, podemos ver que nuestra imagen ha subido con éxito.



Eliminamos la imagen.

