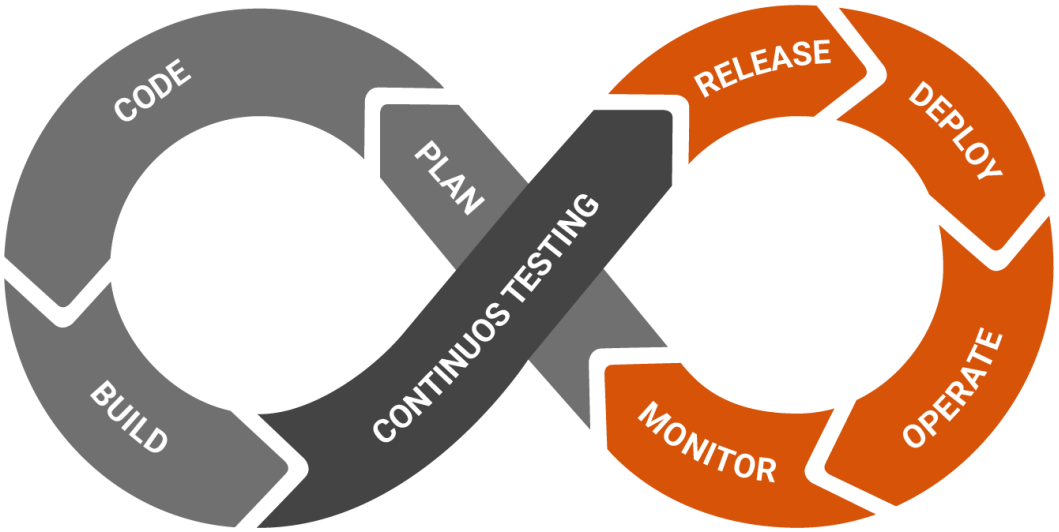
Facultad de Ingeniería



Proyecto Final

Devops & Kubernetes

Integrantes

Ceres Martinez Hanna Sophia

Pedraza Martinez Jose Alberto

Pereda Ceballos Jorge Francisco

Índice

Introducción

En este proyecto, hemos creado una plataforma en línea que permite a los seguidores y admiradores de este renombrado artista adquirir productos únicos y de alta calidad que reflejan su estilo y creatividad. Nuestra página de Ecommerce ha sido desarrollada utilizando tecnologías modernas y se basa en una arquitectura de contenedores con Kubernetes como orquestador. Esta elección nos proporciona una infraestructura escalable y flexible que nos permite gestionar eficientemente el tráfico de usuarios y hacer frente a los picos de demanda durante eventos especiales o lanzamientos de productos.

En este documento, proporcionaremos una visión detallada de las distintas etapas del proyecto, abarcando desde la instalación del clúster de Kubernetes hasta la configuración de seguridad, pasando por la implementación de herramientas de CI/CD y el monitoreo de aplicaciones. Además, presentaremos los aspectos clave de la arquitectura, evaluaremos el correcto funcionamiento de los servidores web y el monitoreo, y realizaremos una demostración del proceso de actualización del sistema.

La documentación también incluirá instrucciones sobre cómo utilizar la página de Ecommerce, resaltando los beneficios que ofrece a los usuarios, así como nuestras conclusiones sobre el desarrollo y despliegue de este apasionante proyecto.

Beneficios

La elección de Kubernetes como orquestador en la arquitectura de nuestra página de Ecommerce trae consigo una serie de beneficios significativos. A continuación, destacaremos algunas de las ventajas clave de utilizar Kubernetes en este proyecto:

- Escalabilidad y Flexibilidad: Kubernetes proporciona una infraestructura altamente escalable y flexible, lo que nos permite adaptarnos fácilmente a las demandas cambiantes del tráfico de usuarios. Podemos aumentar o disminuir la cantidad de réplicas de nuestros contenedores según sea necesario, asegurando un rendimiento óptimo y una experiencia de usuario sin interrupciones, incluso durante picos de carga.

- Automatización de Despliegue: Kubernetes simplifica el proceso de despliegue y actualización de nuestra aplicación. Podemos definir y gestionar fácilmente los recursos necesarios, como contenedores, volúmenes y servicios, a través de archivos de configuración declarativos. Esto nos permite implementar rápidamente nuevas versiones de la aplicación y garantizar la disponibilidad continua sin tiempo de inactividad.

- Gestión Eficiente de Recursos: Con Kubernetes, podemos aprovechar al máximo nuestros recursos de hardware al programar y distribuir los contenedores de manera óptima en los nodos del clúster. El orquestador se encarga de asignar los recursos

necesarios para cada contenedor, evitando la subutilización o la sobreutilización de los recursos disponibles.

- Automatización de Tareas: Kubernetes permite la automatización de tareas repetitivas y administrativas. Podemos utilizar herramientas de CI/CD, como Jenkins para implementar una integración y entrega continuas, lo que agiliza el ciclo de desarrollo y despliegue de nuestra página de Ecommerce.

- Gestión de Estado y Almacenamiento: Con la capacidad de crear volúmenes persistentes, Kubernetes nos proporciona una solución robusta para el almacenamiento de datos. Podemos garantizar la persistencia de la base de datos y la configuración de los servidores web, permitiendo una recuperación confiable de los datos en caso de fallas o reinicios.

- Disponibilidad y Tolerancia a Fallos: Kubernetes ofrece mecanismos integrados para garantizar la alta disponibilidad y la tolerancia a fallos de nuestra aplicación. La capacidad de replicar y distribuir los contenedores en diferentes nodos del clúster asegura que la página de Ecommerce siga funcionando incluso si un nodo o un contenedor experimenta problemas.

- Monitorización y Autoajuste: Con herramientas como Prometheus y Grafana, podemos realizar un monitoreo detallado de los componentes de nuestra página de Ecommerce. Esto nos permite recopilar métricas valiosas sobre el rendimiento, la utilización de recursos y la salud de los contenedores, lo que facilita la detección temprana de problemas y la toma de decisiones informadas para optimizar el sistema.

La adopción de la arquitectura de Kubernetes en nuestra página de Ecommerce brinda una serie de beneficios clave que mejoran la disponibilidad, la escalabilidad, la gestión de recursos y la automatización del despliegue. Estos aspectos contribuyen a una experiencia de usuario mejorada, así como a una mayor eficiencia operativa y una mayor capacidad de respuesta a las demandas del mercado.

Kubernetes

El primer paso en nuestro proyecto es la instalación del clúster de Kubernetes, compuesto por un nodo de control (control plane) y un nodo de trabajo (worker), que son componentes fundamentales para el funcionamiento de nuestra arquitectura.

El nodo de control, conocido como el "cerebro" del clúster, se encarga de gestionar y coordinar todas las operaciones. Incluye componentes clave como etcd, kube-apiserver, kube-controller-manager y kube-scheduler.

Por otro lado, el nodo de trabajo es donde se ejecutarán los contenedores de nuestra aplicación. Actúa como el recurso de procesamiento y alojamiento de los contenedores. En esta etapa, instalaremos componentes esenciales en el nodo de trabajo, como kubelet y kube-proxy.La comunicación adecuada entre el nodo de control y el nodo de trabajo es

crucial para el correcto funcionamiento del clúster. Estableceremos las conexiones necesarias para permitir la comunicación y mantener una sincronización adecuada entre ambos nodos. Una vez completada la instalación, realizaremos verificaciones y validaciones para asegurarnos de que el clúster esté funcionando correctamente. Esto incluirá comprobar la conectividad entre los nodos, verificar que los componentes del nodo de control estén en funcionamiento y confirmar que el nodo de trabajo esté listo para recibir instrucciones.Estas medidas nos asegurarán de que el clúster de Kubernetes esté configurado y listo para soportar nuestras aplicaciones en contenedores. A partir de aquí, podremos avanzar hacia las siguientes etapas del proyecto.

Creación de un deploy

a) Despliegue de al menos 2 contenedores, incluyendo una base de datos

Durante esta fase del proyecto, nos enfocaremos en el despliegue de contenedores, los cuales desempeñarán un papel fundamental en nuestra aplicación. Además, nos aseguraremos de incluir una base de datos para el almacenamiento de información relevante.

Para llevar a cabo el despliegue de los contenedores, aprovecharemos las capacidades de Kubernetes para gestionar y orquestarlos de manera eficiente. Estos contenedores se basarán en imágenes Docker, las cuales contienen todas las dependencias y configuraciones necesarias para su correcto funcionamiento.

En cuanto a la elección de la base de datos, seleccionaremos aquella que mejor se ajuste a las necesidades de nuestra aplicación. La base de datos desempeñará un papel vital al permitirnos almacenar y recuperar datos de manera segura y eficiente.

A continuación, mostraremos los contenedores que formarán parte de nuestra arquitectura:

```
m001@m001:~$ kubectl get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/mariadb-deployment-56bc886475-jnbqt     1/1     Running   0          92m
pod/phpmyadmin-deployment-798858bc-58jzl    1/1     Running   0          92m
pod/phpmyadmin-deployment-798858bc-6p4fm    1/1     Running   0          4s
pod/phpmyadmin-deployment-798858bc-jpk24    1/1     Running   0          4s
pod/phpmyadmin-deployment-798858bc-zscz2    1/1     Running   0          4s

NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP          97m
service/mariadb-service      NodePort    10.97.203.190   <none>        3306:32559/TCP   92m
service/phpmyadmin-service   NodePort    10.96.102.43    <none>        80:30815/TCP     92m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mariadb-deployment      1/1     1            1           92m
deployment.apps/phpmyadmin-deployment   4/4     4            4           92m

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/mariadb-deployment-56bc886475      1         1         1       92m
replicaset.apps/phpmyadmin-deployment-798858bc     4         4         4       92m
m001@m001:~$
```

figura. Contenedores.

Con el despliegue de estos contenedores y la inclusión de una base de datos adecuada, estaremos sentando las bases para el correcto funcionamiento de nuestra aplicación. En las siguientes etapas, nos centraremos en la configuración y la interconexión de estos componentes para lograr una experiencia fluida y confiable para nuestros usuarios.

b) Creación de volúmenes persistentes para la base de datos y la configuración de los servidores web

Para garantizar la persistencia de los datos y la configuración de los servidores web, crearemos volúmenes persistentes en Kubernetes. Estos volúmenes proporcionan almacenamiento duradero y se mantienen incluso cuando los contenedores se detienen o reinician.

Los volúmenes persistentes nos permiten separar los datos y la configuración de los contenedores, lo que facilita la gestión y la recuperación en caso de fallos. Asignaremos volúmenes persistentes tanto a la base de datos como a los servidores web, asegurando que los datos críticos y la configuración se conserven incluso en situaciones inesperadas.

A continuación se enlistan los volúmenes persistentes:

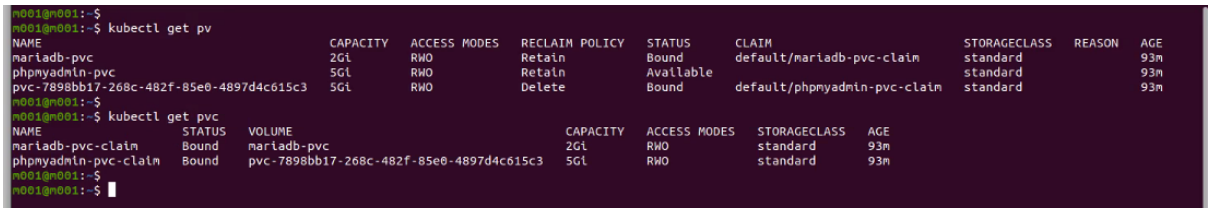


figura. Volúmenes persistentes.

d) Configuración de comunicación en la LAN utilizando Portnode e Ingress

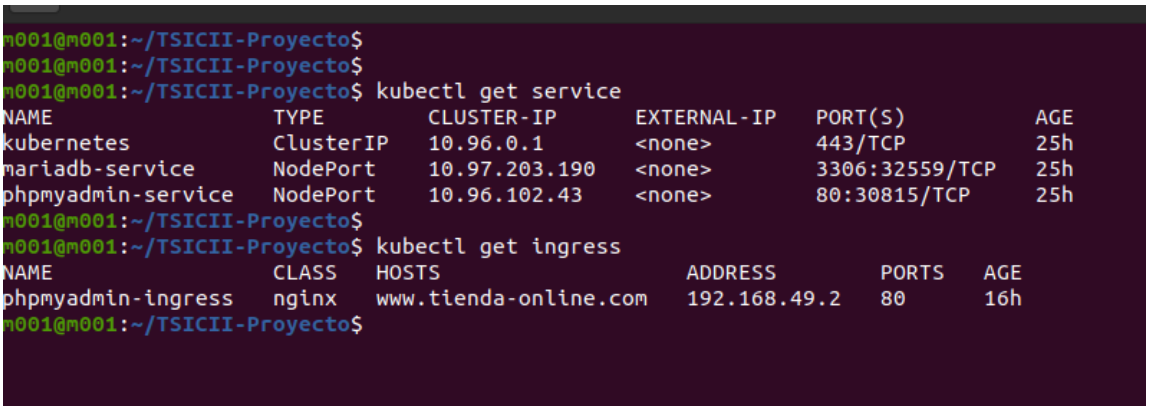
Para habilitar la comunicación en la LAN (Local Area Network) y permitir el acceso a nuestra aplicación, configuraremos Portnode e Ingress en Kubernetes.

Portnode es un componente que nos permite exponer los servicios y contenedores de nuestra aplicación a través de puertos específicos en el clúster. Esto nos permitirá acceder a la aplicación y a los servicios que ofrece.

Ingress, por otro lado, actúa como un controlador de tráfico y equilibrador de carga, dirigiendo las solicitudes de los usuarios a los servicios correspondientes en función de reglas de enrutamiento definidas. Configuraremos Ingress para dirigir el tráfico de la LAN hacia los servicios y contenedores específicos en el clúster de Kubernetes, asegurando así la correcta comunicación con nuestra página de Ecommerce.

Con la configuración de Portnode e Ingress, lograremos establecer una comunicación eficiente y segura en nuestra LAN, permitiendo a los usuarios acceder y utilizar nuestra

aplicación de manera óptima. Esto garantizará una experiencia fluida y una interacción efectiva con los servicios y la base de datos desplegados.



Utilización de una herramienta de CI/CD

La Integración y Distribución Continua (CI/CD) es un conjunto de prácticas altamente relevantes que automatizan las etapas de desarrollo de aplicaciones. Estas prácticas desempeñan un papel fundamental en la eficiencia y confiabilidad de la implementación de actualizaciones de aplicaciones. Para lograr una publicación exitosa, es necesario seguir varias etapas que deben ser aprobadas por el equipo de desarrollo. Estas etapas típicamente incluyen: pruebas, verificación de calidad de código y vulnerabilidades, generación de informes de resultados y publicación en un punto específico. Además, existen herramientas como Terraform para la creación de infraestructura de aplicaciones, así como métodos de seguridad que pueden integrarse en estas etapas.

La elección de esta herramienta se basó en su amplio uso y en su condición de software de código abierto. Jenkins es conocido por su flexibilidad y capacidad de configuración, ya que ofrece un ecosistema de plugins que facilita la integración de diversas herramientas de manera sencilla. Sin embargo, junto con sus beneficios, Jenkins también presenta ciertas desventajas. Por ejemplo, el mantenimiento y las actualizaciones de las instalaciones pueden resultar complicados, y existen limitaciones en algunas características de seguridad. Además, el uso incorrecto de Jenkins, como ejecutar múltiples builds concurrentemente o realizar tareas complejas, puede requerir una cantidad significativa de recursos. A pesar de esto, dado el alcance del proyecto en cuestión, no se requiere un pipeline extremadamente robusto.

a) Despliegue de Jenkins/Argo/Tekton (contenedor).

Antes de implementar el host de Jenkins, se realizó un análisis detallado de las etapas necesarias durante el proceso de despliegue. Dado que este software de automatización requiere un archivo llamado **Jenkinsfile** para ejecutar una serie de etapas, se definieron los siguientes pasos:

1. Identificación del sistema de control de versiones del código.
2. Verificación del contenido del repositorio.
3. Generación de una imagen a partir de un Dockerfile de la aplicación.

4. Publicación de la imagen en el repositorio de imágenes (DockerHub).
5. Conexión a Kubernetes e implementación de los archivos YAML ubicados en el repositorio para publicar la aplicación.

Es importante destacar que para que Jenkins pueda ejecutar estas acciones, se requiere el uso de Docker y configurar las credenciales para la conexión con Kubernetes y las plataformas de control de versiones.

Durante el análisis, se identificaron dos opciones para la instalación del host de Jenkins:

1. En una máquina virtual.

2. En un contenedor.

La primera opción se consideró la más práctica debido a que para realizar la instalación de docker era más sencillo, sin embargo se deliberó por la segunda por motivos de optimización de recursos. En la segunda opción se tuvo que realizar un Docker in docker(dind). Esto puede llegar a ser algo complejo, sin embargo no se profundizó en su manejo y se implementó solo su volumen correspondiente para garantizar persistencia.

```
m001@m001:~/TSICII-Proyecto$ docker ps
CONTAINER ID   IMAGE                                 COMMAND                  CREATED        STATUS        PORTS
                                                                                                              NAMES
e9b088cb6264   jenkins/jenkins:lts                   "/usr/bin/tini -- /u…"   20 hours ago   Up 2 hours    0.0.0.0:8080->8080/tcp,
:::8080->8080/tcp, 0.0.0.0:50000->50000/tcp, :::50000->50000/tcp                                              romantic_solomon
545386f4e1c7   gcr.io/k8s-minikube/kicbase:v0.0.39   "/usr/local/bin/entr…"   25 hours ago   Up 2 hours    127.0.0.1:32772->22/tcp,
 127.0.0.1:32771->2376/tcp, 127.0.0.1:32770->5000/tcp, 127.0.0.1:32769->8443/tcp, 127.0.0.1:32768->32443/tcp   minikube
m001@m001:~/TSICII-Proyecto$
```

figura X. Imagen de contenedor(jenkins/jenkins:lts) con Docker.

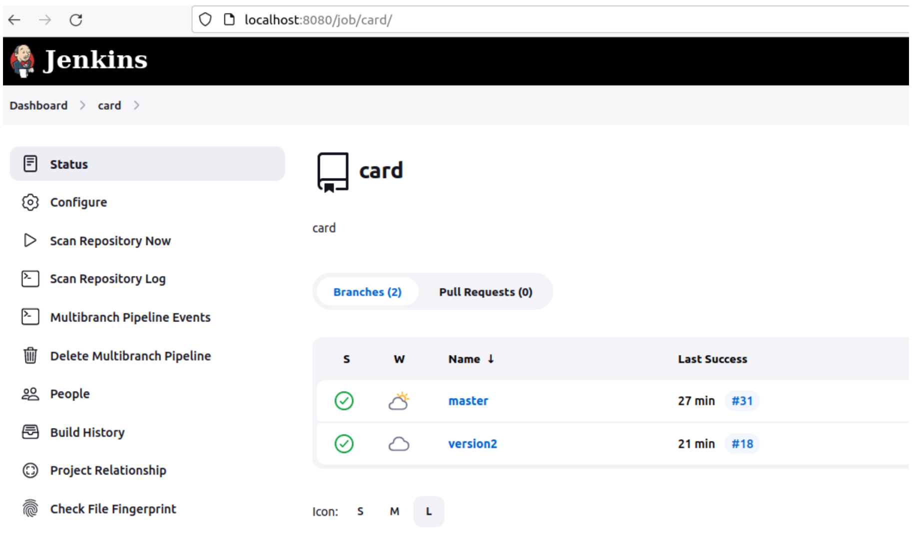


figura Y. Ejecución de imágen en puerto 8080.

Como se puede observar en la figura Y, se cuenta con un proyecto dividido en dos versiones las cuales ya pasaron por etapas de BUILD. La serie de pasos para cargar el proyecto fue la siguiente: create a new Job > Multibranch Pipeline Events > Carga de

credenciales de Github > Configuración de credencial de kubernetes a través de kube config. En este caso, se ingresaron las credenciales únicas generadas por cada cluster ubicadas en /.kube/config o /.minikube/config en minikube . Como el cluster corre con una conexión HTTPS es importante establecer correctamente las llaves de los certificados SSL de lo contrario no se conectará correctamente.

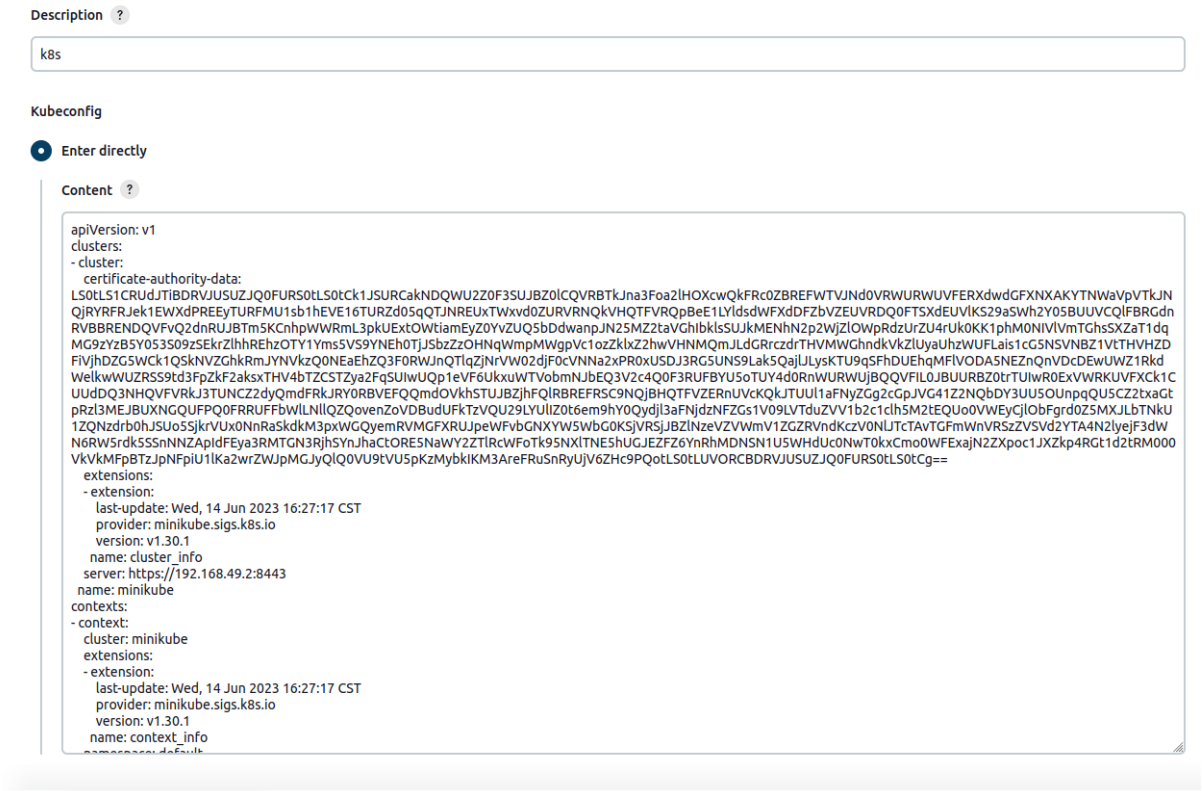


figura Z. Conexión al API de Kubernetes para el despliegue del proyecto.

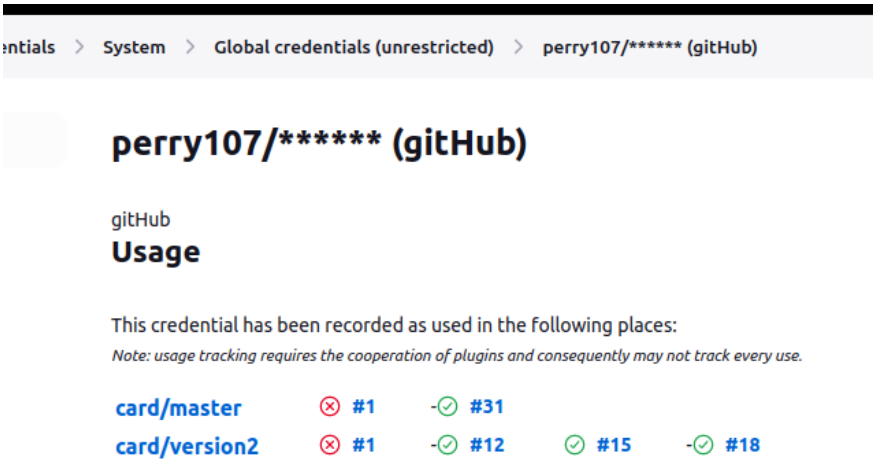


figura W. Conexión con GitHub/GitLab para la gestión del código.

Monitoreo de aplicaciones

Prometheus y Grafana son dos herramientas ampliamente utilizadas en el campo del monitoreo y la visualización de métricas en entornos de infraestructura modernos. Cada una de estas herramientas tiene su propia composición y arquitectura, así como ventajas y desventajas en comparación con otras herramientas de monitoreo disponibles en el mercado. Para su implementación se analizó en detalle Prometheus y Grafana, y por qué son una elección popular en el ámbito del monitoreo.

Prometheus es un sistema de monitoreo de código abierto que se basa en un modelo de recopilación de métricas mediante el scraping. Su arquitectura se compone de varios componentes clave. En primer lugar, está el servidor Prometheus, que recopila y almacena las métricas en una base de datos de series temporales. Luego, están los recolectores de métricas, que son responsables de extraer y exponer las métricas de los componentes de destino a través de puntos de enlace HTTP. Además, Prometheus ofrece un lenguaje de consulta llamado PromQL que permite realizar análisis y alertas en tiempo real basados en las métricas almacenadas.

Por su parte, Grafana es una herramienta de visualización de datos de código abierto que se integra fácilmente con Prometheus y otros sistemas de monitoreo. Su arquitectura se basa en una interfaz web interactiva y personalizable que permite a los usuarios crear paneles de control visualmente atractivos. Grafana es altamente configurable y compatible con una amplia gama de fuentes de datos, lo que facilita la conexión con Prometheus para visualizar las métricas recopiladas. Además, ofrece opciones avanzadas de visualización, como gráficos interactivos, tablas dinámicas y alertas personalizadas.

Una de las principales ventajas de Prometheus y Grafana es su arquitectura escalable y eficiente. Prometheus está diseñado para manejar grandes volúmenes de métricas y puede adaptarse a entornos de escala empresarial sin problemas. Además, ambas herramientas son de código abierto, lo que significa que tienen una comunidad activa que contribuye con mejoras y nuevas características. Esto se traduce en una amplia documentación, recursos y soporte disponibles.

Comparadas con otras herramientas de monitoreo, Prometheus y Grafana ofrecen varias ventajas distintivas. En primer lugar, son altamente flexibles y se adaptan a diferentes entornos y casos de uso. Permiten la recopilación y visualización de métricas personalizadas, lo que brinda a los usuarios un mayor control sobre los aspectos específicos que desean monitorear. Además, la capacidad de consulta en tiempo real y la configuración de alertas avanzadas son características poderosas que permiten a los equipos de operaciones detectar y responder rápidamente a los problemas.

Sin embargo, es importante destacar que Prometheus y Grafana pueden tener una curva de aprendizaje inicial pronunciada para aquellos que no están familiarizados con las herramientas de monitoreo y visualización. Además, aunque Prometheus es excelente para métricas numéricas y series temporales, puede tener limitaciones en la recopilación de otros tipos de datos, como trazas o registros. Para casos de uso específicos, puede ser necesario complementar Prometheus y Grafana con otras herramientas especializadas.

Para este proyecto, se utilizó el repositorio kube-prometheus para su instalación pues fue la más sencilla de realizar. Este repositorio cuenta con diversas configuraciones específicas y personalizaciones en las herramientas de monitoreo. Por ejemplo, en lugar de aplicar un get pods es posible realizar un get buckets, entre otras características. Esta personalización es posible gracias a los custom resources de kubernetes.
De igual manera, cuenta con librerías de gráficos específicos ya implementados en Grafana.
Al implementar un apply en los manifiestos siguiendo la documentación brindada en el repositorio, se ejecutan todos los YAML que componen a servicios, custom resources, deployments, entre otros y son aislados a un namespace denominado monitoring.
Por último, una vez realizada la instalación se expone el puerto correspondiente del servicio de Grafana para empezar a monitorear(3000).

a) Despliegue de Prometheus, Grafana, Kiali, Jaeger, Datadog (mientras más atractivas sean las gráficas, mejor).

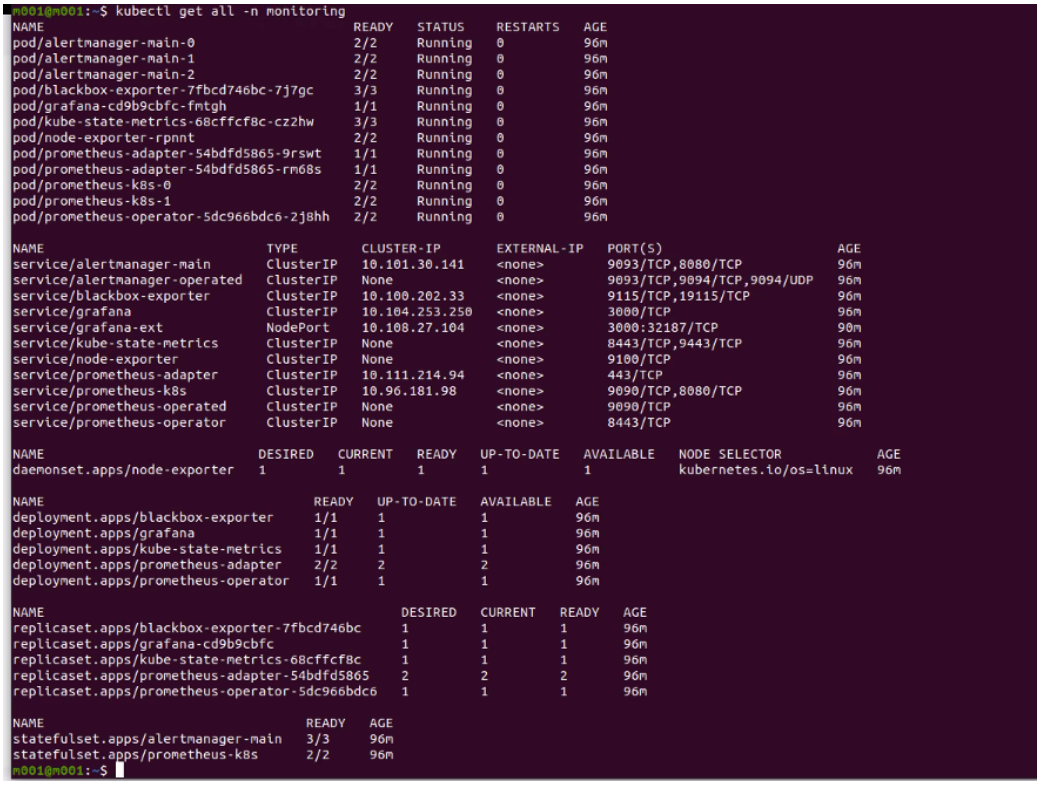


figura R. Instalación correcta y configuración implementada de Prometheus /Grafana.

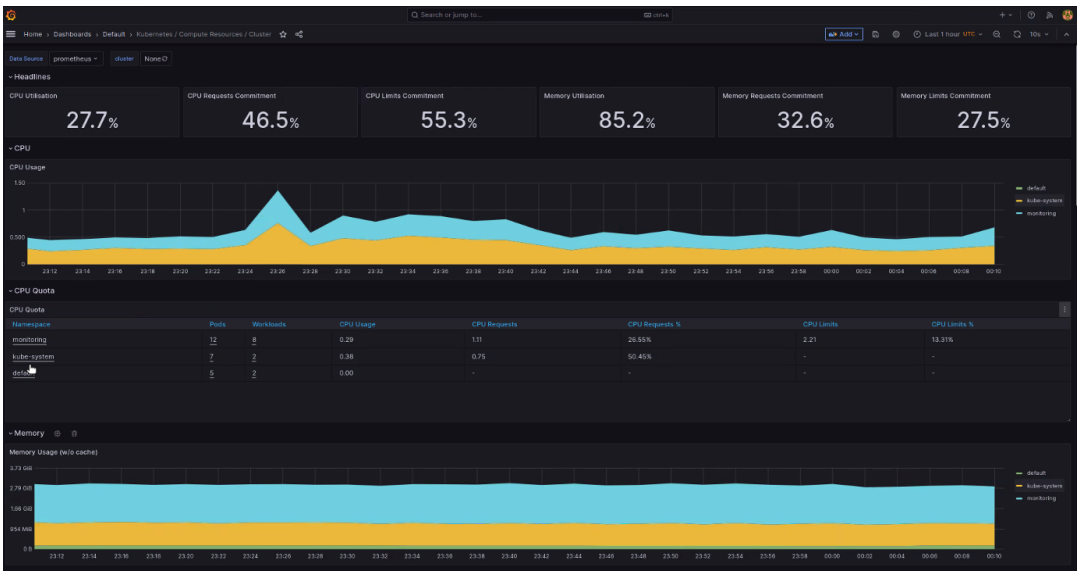


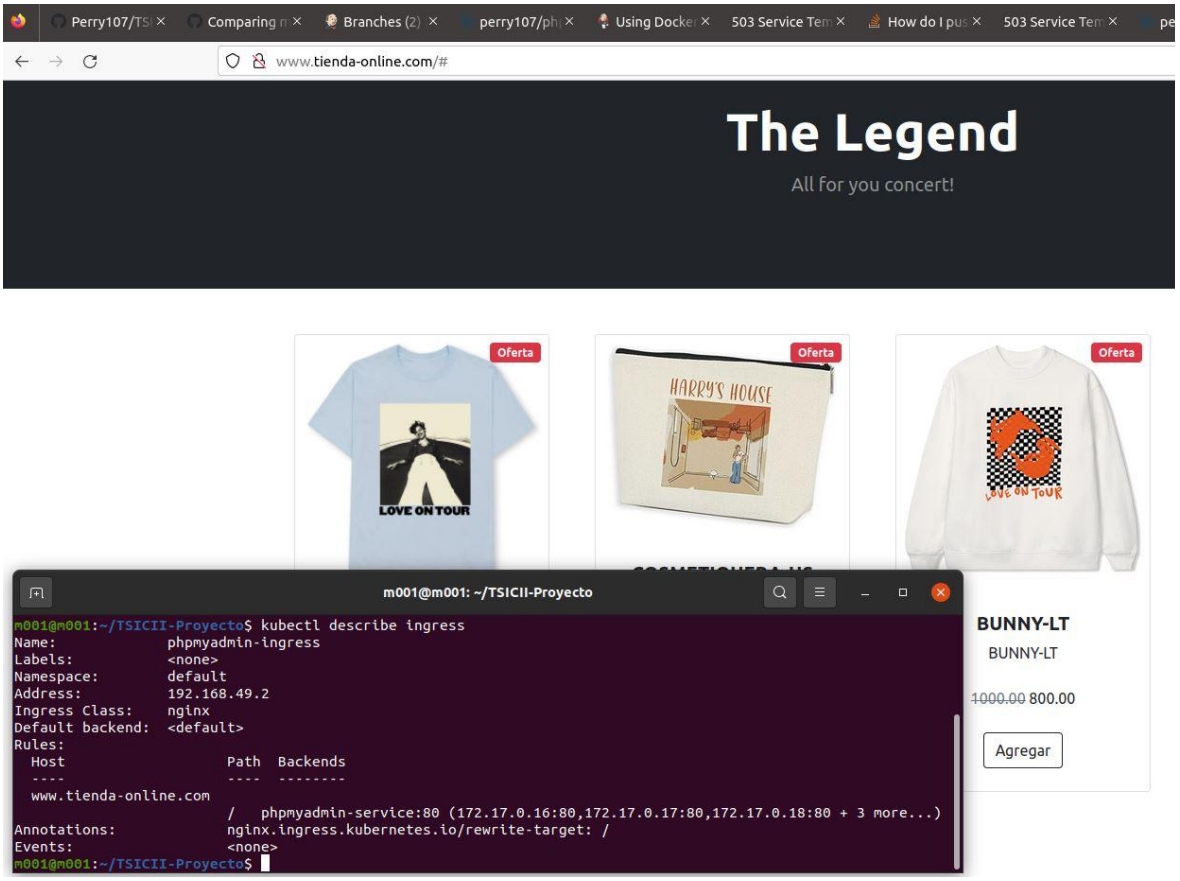
figura S. Generación de gráficas/ Consumo de recursos del Cluster.

figura T. Generación de gráficas/ Consumo de recursos de cada Pod.

Presentación del proyecto

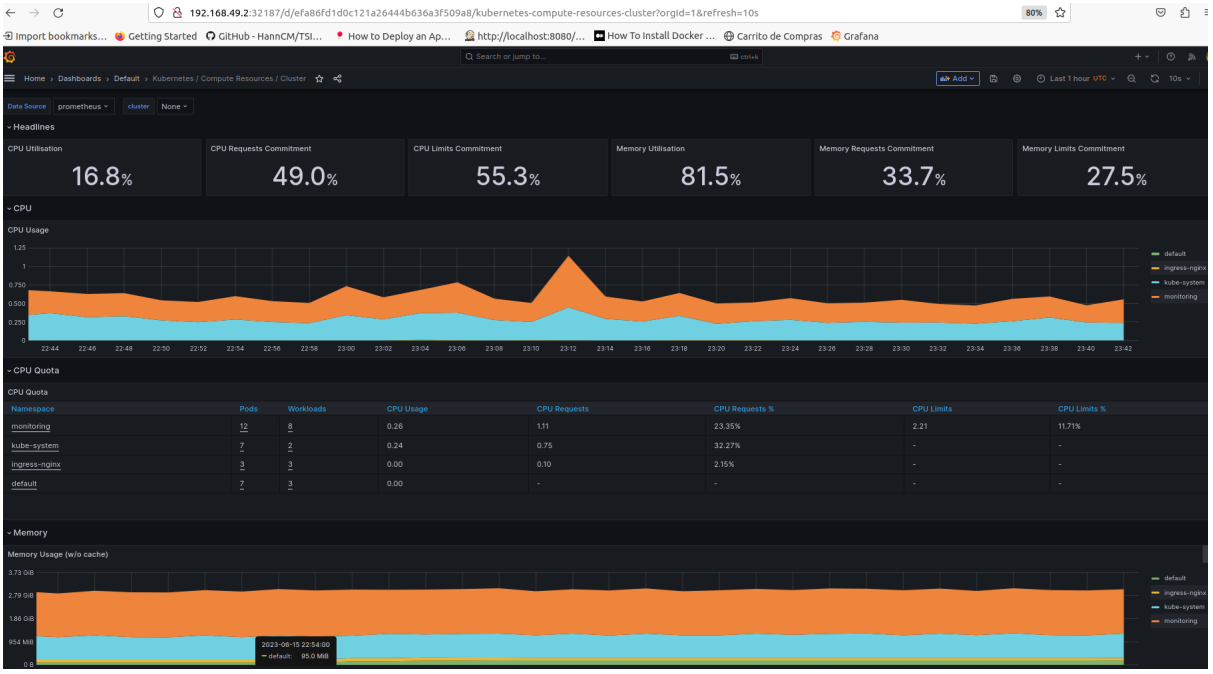
a) Validaciones del servidor web.

Como se puede ver en la **figura .** El cluster cuenta con un ingress configurado para redirigir el tráfico del servicio de nuestra página a la dirección www.tienda-online.com y como se puede ver se tiene conexión con la página.



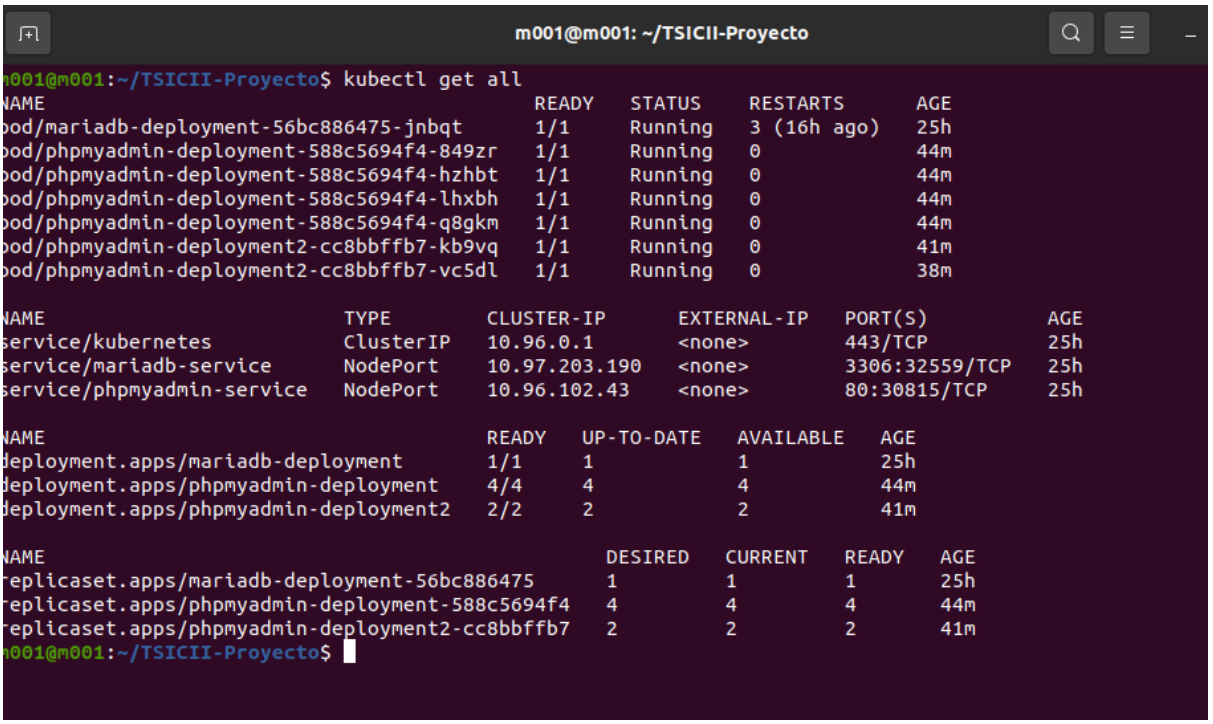
b) Validaciones del monitoreo.

En la imagen podemos ver en funcionamiento Grafa monitoreando el estado de nuestro cluster.



c) Validaciones de la arquitectura.

Como se puede ver todo los elementos de la arquitectura están en funcionamiento.



Estrategia de implementación Canary

Canary es una implementación de versiones basada en un lanzamiento progresivo de una aplicación, esto se logra dividiendo el tráfico entre una versión ya implementada y una versión nueva, se implementa en un subconjunto de usuarios antes del lanzamiento.

Para este proyecto se crearon dos versiones de nuestra página web en el repositorio de gitHub, cada versión se encuentra en una rama diferente y son utilizadas por Jenkins **figura** para crear sus imágenes de docker correspondientes las cuales son implementadas en deployments con el mismo label para que el servicio de conexión las tome como una misma app.

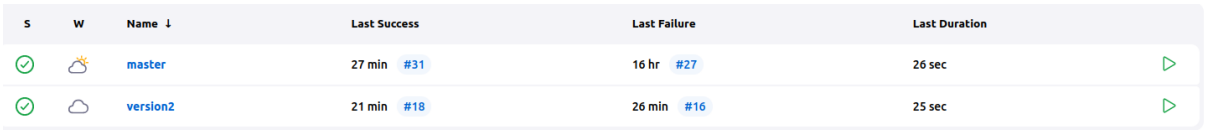


figura . Versiones de la página web.

Como se puede ver en la **figura** en el cluster se encuentran 4 pods con la versión 1 y 2 pods con la versión 2, progresivamente se van aumentado los pods de la versión 2 y disminuyendo los pods de la versión 1 para completar la actualización.

```
m001@m001:~/TSICII-Proyecto$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS      AGE
mariadb-deployment-56bc886475-jnbqt     1/1     Running   3 (16h ago)   24h
phpmyadmin-deployment-588c5694f4-849zr  1/1     Running   0             30m
phpmyadmin-deployment-588c5694f4-hzhbt  1/1     Running   0             30m
phpmyadmin-deployment-588c5694f4-lhxbh  1/1     Running   0             30m
phpmyadmin-deployment-588c5694f4-q8gkm  1/1     Running   0             30m
phpmyadmin-deployment2-cc8bbffb7-kb9vq  1/1     Running   0             27m
phpmyadmin-deployment2-cc8bbffb7-vc5dl  1/1     Running   0             24m
m001@m001:~/TSICII-Proyecto$
```

Figura . Canary

Configuración de seguridad en Kubernetes

a) Implementación de encriptación

Los Secrets son objetos en Kubernetes que se utilizan para almacenar información sensible, como contraseñas, claves de API y certificados. Estos datos confidenciales se almacenan en formato encriptado y solo son accesibles para los contenedores y aplicaciones autorizados.

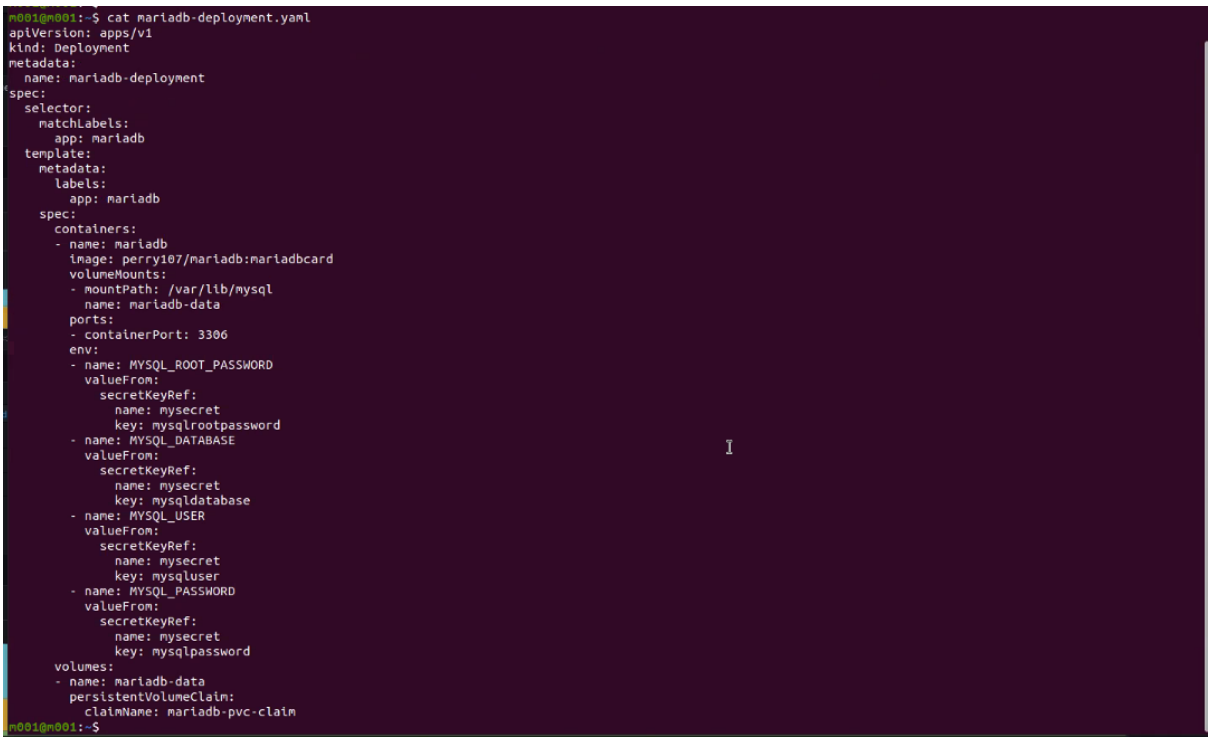
Al utilizar Secrets, podemos proteger de manera efectiva los datos sensibles que se requieren para acceder a servicios externos, bases de datos u otros componentes críticos de nuestra aplicación. Estos Secrets se pueden utilizar en los despliegues y configuraciones de nuestros contenedores, permitiendo que se acceda a la información sensible de manera segura durante el tiempo de ejecución.

Utilizando Secrets en nuestro proyecto de página de Ecommerce, podemos asegurar que la información confidencial, como las credenciales de bases de datos o las claves de acceso a servicios externos, estén protegidas y encriptadas adecuadamente. Esto ayuda a prevenir el acceso no autorizado y garantiza la integridad de nuestros datos sensibles.

A continuación se muestran los Secrets creados así como el archivo yaml:

```
m001@m001:~$ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-xbndw   kubernetes.io/service-account-token   3      102m
mysecret              Opaque                                4      98m
m001@m001:~$
```

figura. Secrets.



```
m001@m001:~$ cat mariadb-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deployment
spec:
  selector:
    matchLabels:
      app: mariadb
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      containers:
      - name: mariadb
        image: perry107/mariadb:mariadbcard
        volumeMounts:
        - mountPath: /var/lib/mysql
          name: mariadb-data
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: mysqlrootpassword
        - name: MYSQL_DATABASE
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: mysqldatabase
        - name: MYSQL_USER
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: mysqluser
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: mysqlpassword
      volumes:
      - name: mariadb-data
        persistentVolumeClaim:
          claimName: mariadb-pvc-claim
m001@m001:~$
```

figura . Secrets -Yaml.

Conclusión

En conclusión, hemos logrado desarrollar una plataforma de Ecommerce innovadora y atractiva para los seguidores y admiradores del artista en cuestión. Gracias a la arquitectura de contenedores con Kubernetes como orquestador, hemos obtenido una infraestructura altamente escalable, flexible y segura.

A lo largo del proyecto, hemos destacado las diversas etapas, desde la instalación del clúster de Kubernetes hasta la implementación de herramientas de CI/CD y el monitoreo de aplicaciones. Estas etapas nos han permitido garantizar un despliegue eficiente, una gestión de recursos óptima y una experiencia de usuario fluida.

Además, hemos evaluado el correcto funcionamiento de los servidores web y el monitoreo, lo que nos ha brindado una visibilidad precisa del rendimiento de la plataforma. Esto nos ha permitido realizar mejoras continuas y asegurar que la página de Ecommerce esté siempre disponible y funcione de manera eficiente.

En nuestra opinión, estamos orgullosos de haber desarrollado esta plataforma de Ecommerce basada en tecnologías modernas y enfocada en ofrecer productos únicos y de alta calidad. Esperamos que los seguidores y admiradores disfruten de esta experiencia y que encuentren en nuestra plataforma una forma emocionante de conectar con el arte y la creatividad del artista en cuestión.

Referencias

- https://www.linuxtechi.com/how-to-install-minikube-on-ubuntu/
- https://kubernetes.io/docs/tasks/configmap-secret/managing-secret-using-config-file/
- https://kubernetes.io/es/docs/concepts/workloads/pods/init-containers/
- https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider
- https://github.com/prometheus-operator/kube-prometheus