Curso: Docker y Kubernetes

Examen Final

Integrantes (Equipo Crédito - Hites)

Claudia Contador - Rodrigo Tobar

Jaime Carcasson - Sergio Chavez

Instructor: Rodrigo Pichiñual

26/05/2023

Parte 1 - Docker Compose (35%)

Debe crear un archivo docker-compose.yaml con las siguientes especificaciones:

- El archivo deberá contener como mínimo la definición de 2 servicios.
- Las imágenes a usar en los servicios se deben previamente subir a su cuenta de DockerHub (repositorio de imágenes) para luego en el campo `image` hacer uso de estas, *`ej: image: rootdrigo0461/node`*
- En el archivo deberá definir y montar un volumen como mínimo.
- Deberá crear una network con el driver bridge sobre la cual se ejecutarán los contenedores.
- Deberá exponer al menos un puerto haciendo un mapeo con el host, de manera que el servicio pueda ser consumido externamente.
- Deberá especificar dependencias entre los contenedores.
- Deberá asignar a los contenedores una política de *restart always*.
- Deberá pasar mínimo 2 variables de entorno a alguno de los contenedores.
- Deberá incluir un archivo Readme, cuyo contenido debe tener las instrucciones de cómo ejecutar los contenedores y testear la aplicación. `ej:localhost port 80`

DESARROLLO

Parte 1 – Docker Compose

Este desarrollo considera la creación de un programa diseñado en el lenguaje Go, que se encarga de operar como un servidor web sencillo. Este servidor interactúa con una base de datos PostgreSQL, con el objetivo de exponer mediante un servicio, datos estadísticos relacionados con las actividades de la base de datos. Para esta tarea, se utiliza la tabla genérica "activity".

El archivo YAML, pieza central de este proyecto, está diseñado de acuerdo con la versión 3.8 de las especificaciones de Docker Compose, una herramienta frecuentemente utilizada para la definición y gestión de múltiples contenedores Docker como si fueran un solo servicio. Dentro de este archivo, se han establecido dos servicios principales: un servidor web y un servidor de base de datos.

El servidor web opera a través de una imagen personalizada que reside en DockerHub, específicamente en rodrigotobarhites/equipocredito-escalab:dcompose. Esta imagen contiene una aplicación web construida con Go, que provee una respuesta JSON. El servicio ha sido configurado para escuchar en el puerto 80, asociando este puerto con su contraparte en el host interno 8080, permitiendo así el acceso al servicio desde el exterior.

El servidor web mencionado anteriormente se enlaza con una red denominada "webnet", la cual se define utilizando el controlador "bridge" de Docker. Este controlador ofrece una red privada interna dentro del host de Docker donde los contenedores pueden comunicarse entre sí. Importante es mencionar que el servidor web depende del servicio "db", lo cual implica que Docker Compose se asegurará de que el servicio "db" se inicie previamente al servidor web.

Con el fin de mantener el servicio en funcionamiento continuo, se implementa una política de reinicio "always". En caso de fallo del contenedor, Docker procurará su reinicio de manera automática. Adicionalmente, se suministran tres variables de entorno al contenedor: DB_HOST, DB_USER y DB_PASSWORD, las cuales establecen el acceso a la base de datos.

El segundo servicio, "db", también emplea una imagen personalizada desde DockerHub: rodrigotobarhites/equipocredito-escalab:postgres13. Basada en PostgreSQL 13, esta imagen se utiliza para el almacenamiento de datos necesarios para la aplicación. Tal como el servidor web, este servicio también se conecta a la red "webnet" y mantiene una política de reinicio "always".

Para este servicio "db", se definen las variables de entorno POSTGRES_USER y POSTGRES_PASSWORD. Dichas variables permiten la configuración del nombre de usuario y la contraseña que PostgreSQL usará para las conexiones de base de datos. Estos valores son indispensables para la seguridad de la base de datos y también necesarios para la interacción de la aplicación web con la misma.

Para la persistencia de los datos de la base de datos, se utiliza un volumen denominado "db_data", que se monta en la ruta /var/lib/postgresql/data dentro del contenedor. De esta manera, se asegura la persistencia de los datos de la base de datos incluso si el contenedor se detiene o se elimina.

Este archivo Docker Compose proporciona un método simple y eficiente para configurar una aplicación con múltiples contenedores, garantizando una comunicación eficaz entre todas las partes necesarias y la preservación de datos vitales. Para poner en marcha la aplicación, solo es necesario ejecutar el comando "docker-compose up" en el directorio que contiene el archivo YAML. Esta estructura simplifica significativamente el proceso de implementación y mantenimiento de la aplicación, y asegura que los servicios esenciales estén disponibles y funcionen correctamente.

Para su visualización dejaremos nuestro proyecto en nuestro repositorio github, e incluiremos el archivo readme con los pasos de la instalación, junto con un video del proceso y el testeo en localhost:80.

https://github.com/rodrigotobarhites/equipocredito-escalab.git

Video Ejecución: https://youtu.be/10JKsie5BDk

Código archivo docker-compose.yaml

docker-compose.yaml

```yaml
version: '3.8'
services:
  web:
    image: rodrigotobarhites/equipocredito-escalab:dcompose
    ports:
      - "80:8080"
    networks:
      - webnet
    depends_on:
      - db
    restart: always
    environment:
      - DB_HOST=db
      - DB_USER=postgres
      - DB_PASSWORD=securepassword
  db:
    image: rodrigotobarhites/equipocredito-escalab:postgres13
    networks:
      - webnet
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=securepassword
    volumes:
      - db_data:/var/lib/postgresql/data
volumes:
  db_data:
networks:
  webnet:
    driver: bridge
```

Parte 2 - Kubernetes (50%)

Descripción del proyecto

Para este proyecto, debes crear una web que tenga un frontend y un backend, utilizando diferentes recursos de Kubernetes para su implementación.

Requisitos de la implementación

La aplicación debe ser implementada utilizando los siguientes recursos de Kubernetes:

- Un Deployment para el frontend.
- Un Deployment para el backend.
- Un Service para el frontend de tipo Clusterip.
- Un Service para el backend de tipo Clusterip.
- Un Hpa para el frontend.
- Un Hpa para el backend.
- Un ConfigMap y/o Secret para que el frontend se pueda comunicar con el backend (definir credenciales o variables de entorno).
- Todos los recursos antes mencionados deben pertenecer a un namespace distinto al default.

Pasos a seguir

1. Crear un archivo YAML para el Namespace.
2. Crear un archivo YAML para el Deployment del frontend.
3. Crear un archivo YAML para el Deployment del backend.
4. Crear un archivo YAML para el Service del frontend.
5. Crear un archivo YAML para el Configmap y/o Secret para el frontend.
6. Crear un archivo YAML para el ConfigMap del backend.
7. Crear un archivo YAML para el Secret de las credenciales base de datos.
8. Crear un archivo YAML para el HPA para ambos deployment.

Nota: Deberá incluir un archivo Readme con las instrucciones de instalación, el instructor ejecutará la aplicación con **port-forward.**

DESARROLLO

Parte 2 – Kubernetes

Este proyecto desarrollando ambas capas en go, interactúan para realizar una aplicación web con la finalidad de mostrar site responsivo para mostrar los integrantes del proyecto, su cargo en hites y sus principales competencias técnicas.
Para esto se desarrollaron 11 archivos yaml que realizan todos los requisitos de implementación y incluiremos en el repositorio mencionado anteriormente.

1. Namespace YAML:

Este archivo crea un espacio de nombres aislado dentro del clúster de Kubernetes llamado "equipocredito". Este namespace permite agrupar y aislar los recursos utilizados por nuestra aplicación del resto del cluster.

2. Deployment YAML del Frontend y Backend:

Se crean dos Deployments para implementar nuestra aplicación web, uno para el frontend y otro para el backend. Ambas partes de la aplicación están construidas usando Go y se alojan como imágenes en Docker Hub ("rodrigotobarhites/equipocredito-escalab:frontend" y "rodrigotobarhites/equipocredito-escalab:backend"). Los Deployments administran el estado deseado para los Pods, creando y actualizando automáticamente los Pods a partir de la definición especificada en la plantilla del Deployment.

Cada Deployment hace referencia a un ConfigMap (para la configuración del entorno de la aplicación) y a un Secret (para las credenciales de la base de datos) que se necesitan para ejecutar correctamente las aplicaciones.

3. Service YAML del Frontend y Backend:

Se definen dos Services para exponer los Deployments dentro del cluster de Kubernetes. Los Services proporcionan una IP estática y un nombre DNS para los Pods, permitiendo la comunicación entre el frontend y el backend (ClusterIP).

4. ConfigMap y Secret YAMLs:

Se crean un ConfigMap y un Secret para proporcionar la configuración y las credenciales de la base de datos a las aplicaciones.

El ConfigMap del frontend contiene la URL del backend a la que se debe conectar, mientras que el ConfigMap del backend contiene un archivo JSON con información de datos que se cargará durante la ejecución.

Por otro lado, el Secret contiene las credenciales para conectarse a la base de datos PostgreSQL, proporcionando una forma segura de almacenar y usar la información sensible.

5. Horizontal Pod Autoscaler (HPA) YAML:

Se crean dos archivos de HPA, uno para el frontend y otro para el backend. Estos HPA permiten que el número de réplicas de los Pods se ajuste dinámicamente en función de la utilización de la CPU, proporcionando escalabilidad automática a la aplicación.

6. LoadBalancer Service YAML:

Este archivo crea un Service de tipo LoadBalancer para el frontend, permitiendo el acceso a la aplicación desde fuera del cluster de Kubernetes a través de una IP externa. Para facilitar el acceso y visualización se trabajo en GCP generando una IP dinámica.

01-namespace.yaml

```yaml
apiVersion: v1

kind: Namespace
metadata:
  name: equipocredito
```

03-frontend-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: equipocredito
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: rodrigotobarhites/equipocredito-escalab:frontend
        ports:
        - containerPort: 8080
        envFrom:
        - configMapRef:
            name: frontend-config
        volumeMounts:
        - name: frontend-config-volume
          mountPath: /template.html
          subPath: template.html
      volumes:
      - name: frontend-config-volume
        configMap:
          name: frontend-config
```

02-frontend-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontend-config
  namespace: equipocredito
data:
  BACKEND_URL: http://backend.equipocredito.svc.cluster.local:8080/obtenerIntegrantes
  template.html: |
    <!DOCTYPE html>
    <html lang="en">
    <head><meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Trabajo Final - Curso Docker y K8s (Escalab)</title>
        <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
rel="stylesheet">
        <style>
            body {
                background: linear-gradient(to right, #065ca3, #0378a6, #0096a9);
            }
            .shadow-text {
                color: #FFFFFF;
                text-shadow: 2px 2px 4px #000000;
            }
            h1 {
                font-size: 6.2vw;
                white-space: nowrap;
                overflow: hidden;
                text-overflow: ellipsis;
            }
            @media (orientation: landscape) {
                h1 {
                    font-size: 4vw;
                }
                .container {
                    max-width: 90%;
                }
            }
        </style>
    </head><body><div class="container">
   <div class="row justify-content-center py-5"><div class="col-12 text-center"><img
src="https://i.ibb.co/grJRFfD/Hites.png" class="img-fluid"><h1 class="mt-4 shadow-text">Equipo
Crédito</h1></div></div>
            {{range .}}
            <div class="row justify-content-center py-3">
                <div class="col-md-6"><div class="card mb-4">
                        <img src="{{.FotoURL}}" class="card-img-top">
                        <div class="card-body">
                            <h5 class="card-title">{{.Nombre}}</h5>
                            <p class="card-text"><strong>Cargo:</strong> {{.Cargo}}</p>
                            <p class="card-text"><strong>Correo:</strong> {{.Correo}}</p>
                            <p class="card-text">{{.Descripcion}}</p>
                        </div>
                    </div>
                </div>
            </div>
            {{end}}
        </div></body></html>
```

04-backend-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-config
  namespace: equipocredito
data:
  datos.json: |
    [
        {
            "foto_url": "https://i.ibb.co/xgf1HZ8/1684462502508.jpg",
            "nombre": "Jaime Carcasson",
            "cargo": "Lider de Fabrica AS400",
            "correo": "jaime.carcasson@hites.cl",
            "descripcion": "Experto en sistemas logísticos y de reposición Hites.
Especialista en Analisis y desarrollo de aplicaciones en diversos lenguajes de programación
y bases de datos relacionales (SQL Server, Oracle, DB2)"
        }, {
            "foto_url": "https://i.ibb.co/25BPMD9/1684462367140.jpg",
            "nombre": "Claudia Contador",
            "cargo": "Arquitecto As400",
            "correo": "claudia.contador@hites.cl",
            "descripcion": "Función definir la compatibilidad e integración de las distintas
plataformas tecnológicas , liderar las estrategias, diseño e implantación de soluciones
técnicas"
        },{
            "foto_url": "https://i.ibb.co/Sx0ChW5/rodrigot.png",
            "nombre": "Rodrigo Tobar",
            "cargo": "Subgerente de Proyectos TI - Crédito",
            "correo": "luis.tobar@hites.cl",
            "descripcion": "Lider Equipo AS400. Experiencia en banca, retail y servicios
financieros. Especialista en Arquitectura de Soluciones Cloud y  Gestión Estratégica.
Dominio técnico en Transformación Digital, Inteligencia Artificial, Cyberseguridad,
Blockchain Empresarial, Data Science y Automatización."
        },{
            "foto_url": "https://i.ibb.co/WgkLKXJ/sergiocha.png",
            "nombre": "Sergio Chavez",
            "cargo": "Arquitecto T.I",
            "correo": "sergio.chavez@hites.cl",
            "descripcion": "Más de 25 años realizando labores como desarrollador de
aplicaciones , jefe de proyecto , jefe de ingeniería de sistema y actualmente como
Arquitecto T.I en Hites , especializado en plataforma IBM i certificado en arquitectura
power IBM y diplomado en ingeniería de software"
        }
    ]
```

05-backend-secrets.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
  namespace: equipocredito
type: Opaque
data:
  DB_HOST: cG9zdGdyZXMuZXF1aXBvY3JlZGl0by5zdmMuY2x1c3Rlci5sb2NhbDo1NDMy  #
"postgres.equipocredito.svc.cluster.local:5432" en base64
  DB_USER: cG9zdGdyZXM=  # "postgres" en base64
  DB_PASSWORD: c2VjdXJlcGFzc3dvcmQ=  # "securepassword" en base64
```

08-backend-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
  namespace: equipocredito
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

07-frontend-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: equipocredito
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

09-loadbalancer.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-lb
  namespace: equipocredito
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

06-backend-deployment.yaml

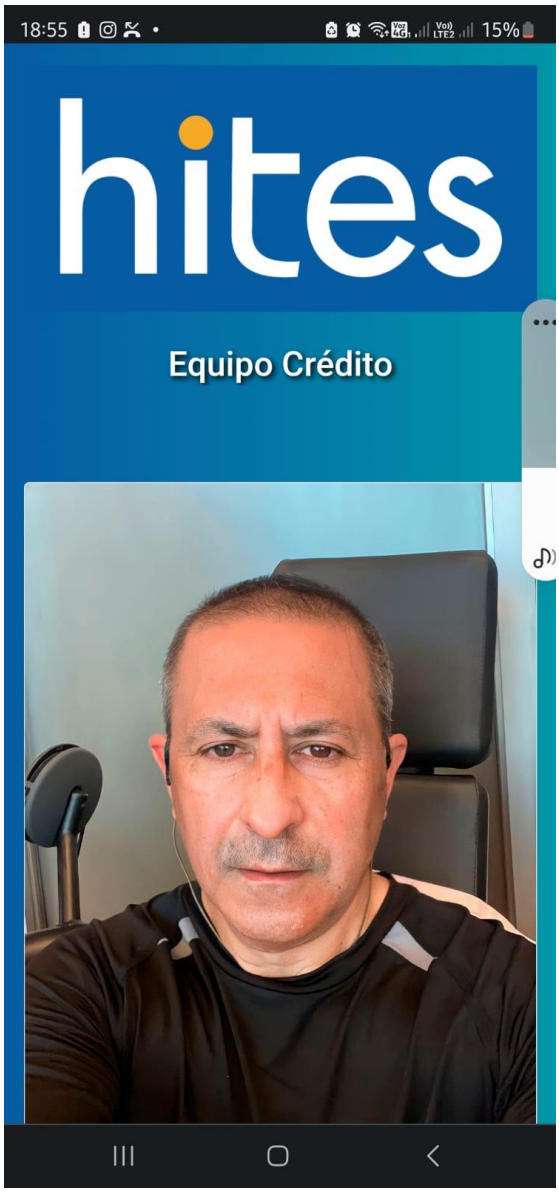
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: equipocredito
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: rodrigotobarhites/equipocredito-escalab:backend
        ports:
        - containerPort: 8080
        envFrom:
        - configMapRef:
            name: backend-config
        env:
          - name: DB_HOST
            valueFrom:
              secretKeyRef:
                name: db-credentials
                key: DB_HOST
          - name: DB_USER
            valueFrom:
              secretKeyRef:
                name: db-credentials
                key: DB_USER
          - name: DB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: db-credentials
                key: DB_PASSWORD
        volumeMounts:
        - name: backend-config-volume
          mountPath: /datos.json
          subPath: datos.json
      volumes:
      - name: backend-config-volume
        configMap:
          name: backend-config
```

10-frontend-hpa.yaml

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-hpa
  namespace: equipocredito
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: frontend
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

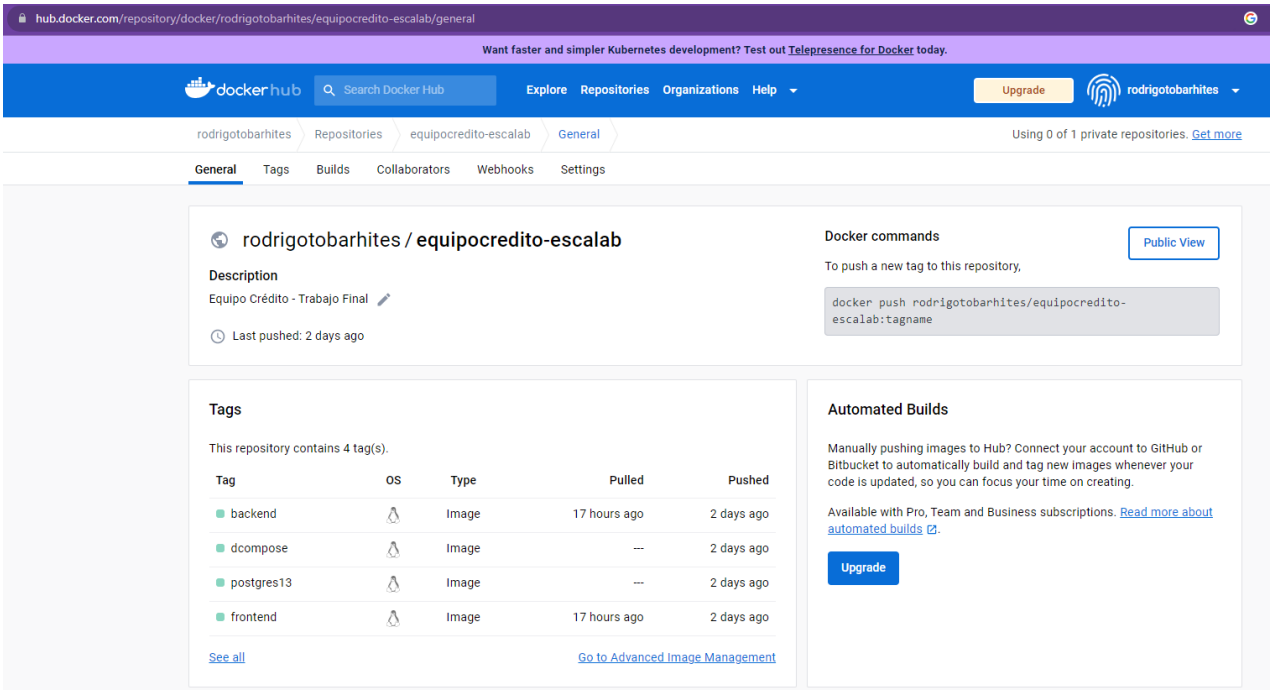
11-backend-hpa.yaml

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: backend-hpa
  namespace: equipocredito
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: backend
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```



DEMO: https://frontend-fzmrmv6xsa-tl.a.run.app/

DOCKER HUB – REPOSITORIO IMÁGENES



https://hub.docker.com/r/rodrigotobarhites/equipocredito-escalab

DEMOSTRACIÓN: https://youtu.be/SWW-_ABuCu0

Parte 3 – Conceptos. (15%)

Descripción de la sección.

En kubernetes existen varios conceptos además de los vistos en clases, como también herramientas de terceros que nos ayudan a la gestión y mantenimiento de manifiestos o del cluster kubernetes en cuestión.

Definir y describir los siguientes conceptos.

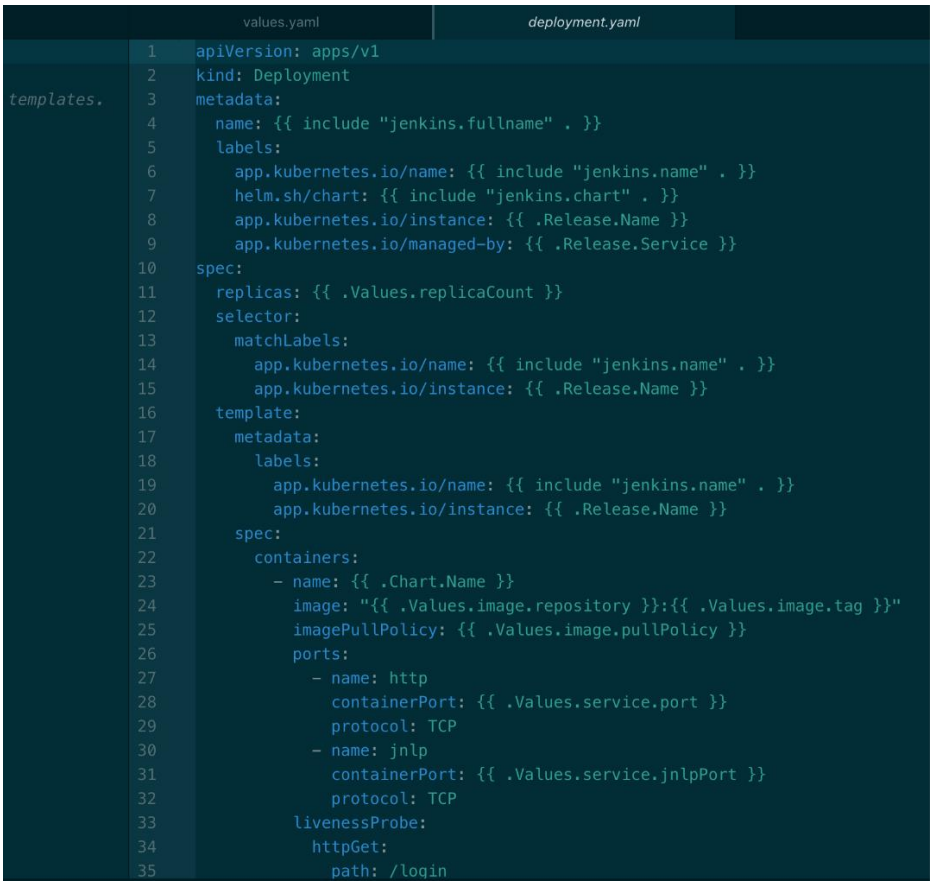
- Kustomize
- Helm
- Kompose
- Ingress
- CertManager

Para cada definición, favor escribir entre 4 a 10 líneas como máximo.

DESARROLLO

Parte 3 - Conceptos.

Kustomize: Es una herramienta de orquestación de Kubernetes que permite la personalización de configuraciones sin la necesidad de alterar los archivos yaml originales. Esta herramienta tiene un enfoque declarativo, permitiendo la superposición de diferentes conjuntos de cambios sobre una base de recursos común. Esto permite a los administradores tener configuraciones diferentes para distintos entornos (producción, desarrollo, pruebas) sin la necesidad de tener múltiples archivos de configuración.



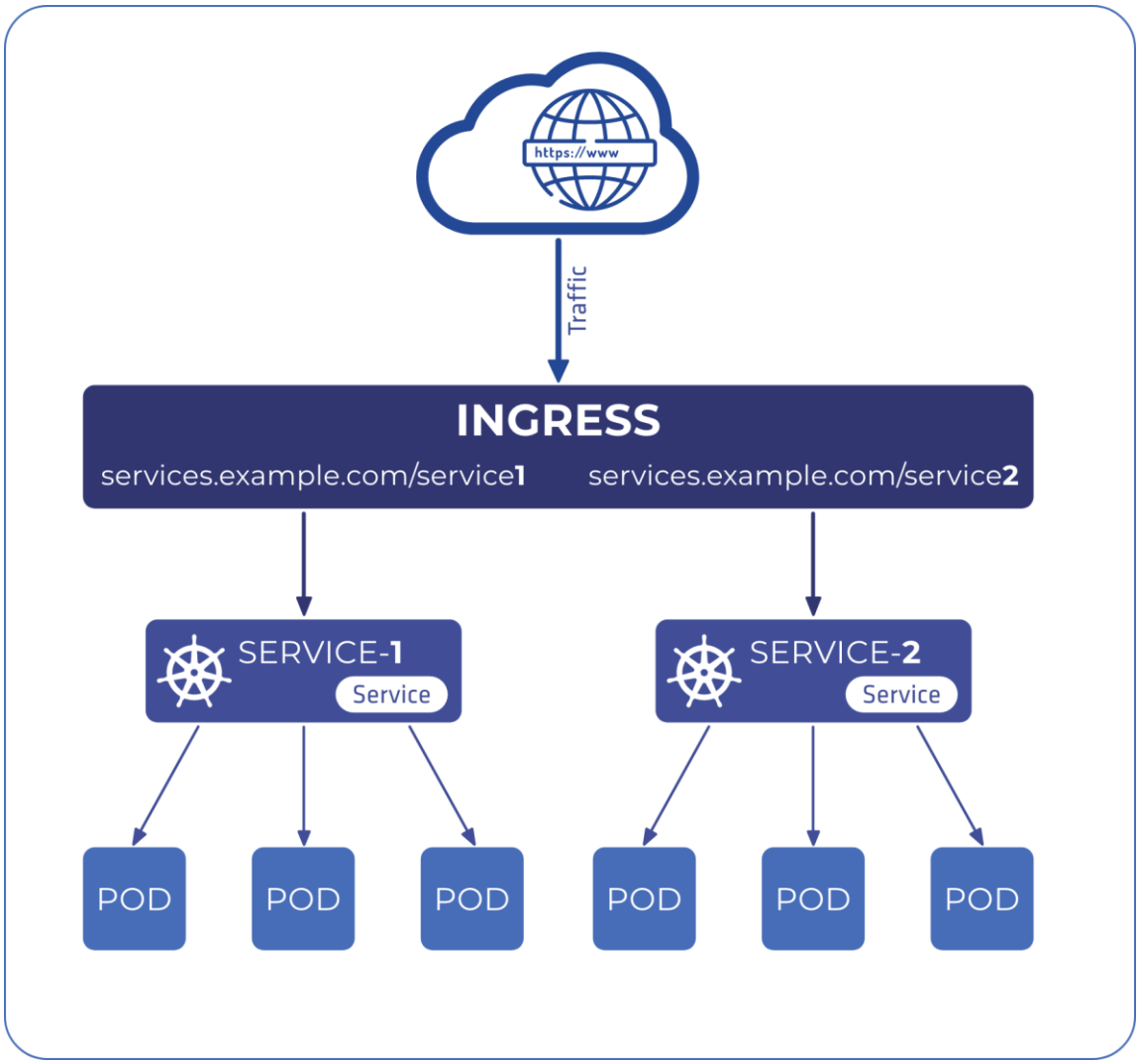
Helm: Es el gestor de paquetes de Kubernetes. Simplifica el despliegue y la gestión de aplicaciones en Kubernetes al agrupar en un solo lugar los múltiples archivos de configuración necesarios para desplegar una aplicación. Los paquetes de Helm, denominados charts, pueden ser compartidos y reutilizados, lo que permite a los equipos trabajar de manera más eficiente y consistente. Un chart, consiste en una colección de archivos que describen un conjunto relacionado de recursos de Kubernetes. Ejemplo: Un chart de Helm podría definir todo lo necesario para desplegar una aplicación web, incluyendo servicios, despliegues y configuraciones.

values.yaml | deployment.yaml

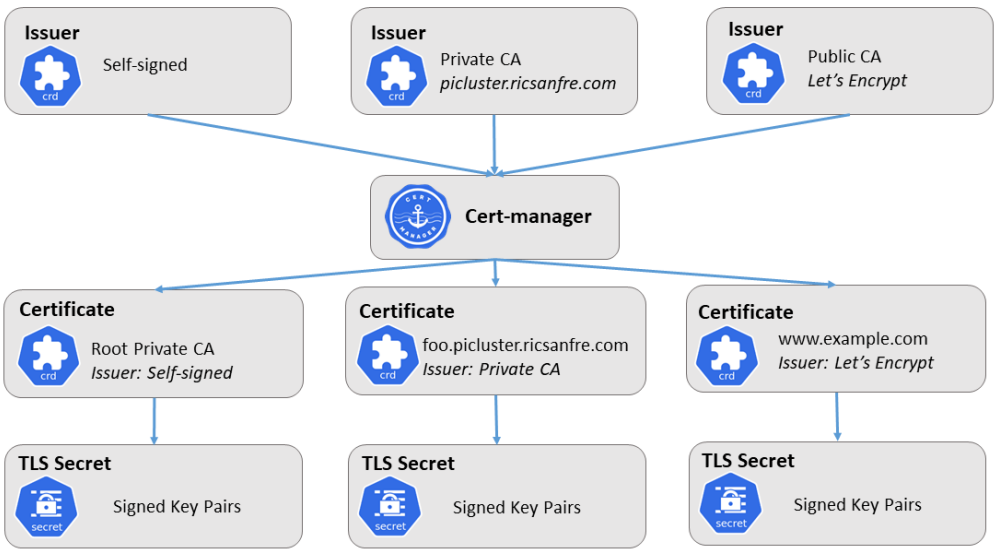
templates.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "jenkins.fullname" . }}
  labels:
    app.kubernetes.io/name: {{ include "jenkins.name" . }}
    helm.sh/chart: {{ include "jenkins.chart" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app.kubernetes.io/name: {{ include "jenkins.name" . }}
      app.kubernetes.io/instance: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app.kubernetes.io/name: {{ include "jenkins.name" . }}
        app.kubernetes.io/instance: {{ .Release.Name }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: {{ .Values.service.port }}
              protocol: TCP
            - name: jnlp
              containerPort: {{ .Values.service.jnlpPort }}
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /login
```

Kompose: Facilita la transición de Docker a Kubernetes. Esta herramienta convierte los archivos Docker Compose en configuraciones de Kubernetes, lo que es especialmente útil para los equipos que están empezando a trabajar con Kubernetes pero ya tienen una infraestructura existente basada en Docker. Esto permite una migración más suave y coherente hacia Kubernetes.



Ingress: Ingress es un objeto API en Kubernetes. El recurso Ingress en Kubernetes proporciona reglas de enrutamiento de HTTP y HTTPS para dirigir el tráfico externo a los servicios dentro del clúster. Este objeto API permite balanceo de carga, terminación SSL y redirecciones de nombres de host, esencial para la exposición de servicios a los usuarios finales.



CertManager:

Es un controlador de Kubernetes nativo que automatiza la gestión de certificados TLS. Interactúa con los emisores de certificados, este solicita certificados y los renueva antes de que expiren. por Ejemplo: se podría configurar CertManager para solicitar un certificado de Let's Encrypt para un sitio web alojado en Kubernetes y renovarlo automáticamente.