

Universidad Tecnológica de Santiago (UTESA)



Tema:

Examen final

Presentado por:

Josué Arismendy Brito Mat. 1-19-0334

Asignatura

Algoritmos Paralelos

Profesor

ING. Ivan Mendoza

Fecha:

11 de abril, 2023


















1) Investigación de Kubernetes

Kubernetes, también conocido como "K8s", es una plataforma de sistema distribuido de código libre que ayuda a automatizar el despliegue, ajuste de escala y manejo de aplicaciones en contenedores. Inicialmente desarrollado por Google y luego donado a la Cloud Native Computing Foundation, Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Su capacidad para facilitar la automatización y la configuración declarativa ha hecho que su popularidad crezca rápidamente y se convierta en una opción popular para las empresas que buscan una solución de administración de contenedores.

Kubernetes se basa en la idea de orquestar contenedores para administrar y mantener aplicaciones de manera efectiva. Los contenedores son una forma de empaquetar una aplicación junto con todas sus dependencias, lo que hace que sea más fácil de mover entre diferentes entornos y plataformas. Kubernetes automatiza el despliegue y la gestión de estos contenedores, lo que permite a los desarrolladores centrarse en escribir código y dejar que Kubernetes maneje la infraestructura subyacente. La plataforma también permite la escala automática de los contenedores, lo que significa que puede aumentar o disminuir la cantidad de recursos asignados a una aplicación en función de la demanda.

En resumen, Kubernetes es una plataforma de sistema distribuido de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores. Su capacidad para facilitar la automatización y la configuración declarativa lo convierte en una opción popular para las empresas que buscan una solución de administración de contenedores. Kubernetes se basa en la idea de orquestar contenedores para administrar y mantener aplicaciones de manera efectiva y automatiza el despliegue y la gestión de estos contenedores, permitiendo a los desarrolladores centrarse en escribir código.

2) Comandos de Kubernetes

-  `kubectl apply`: Crea o actualiza un recurso en función de un archivo de configuración.
-  `kubectl attach`: Conecta a una terminal en ejecución en un contenedor.
-  `kubectl cluster-info`: Muestra información sobre el clúster.
-  `kubectl config`: Administra la configuración de `kubectl`.
-  `kubectl create`: Crea un recurso a partir de un archivo de configuración o de un argumento en línea.
-  `kubectl delete`: Elimina uno o más recursos por nombre o por archivo de configuración.
-  `kubectl describe`: Muestra detalles sobre un recurso.
-  `kubectl exec`: Ejecuta un comando en un contenedor en ejecución.
-  `kubectl get`: Muestra uno o más recursos.
-  `kubectl logs`: Muestra los registros de un contenedor en ejecución.
-  `kubectl port-forward`: Reenvía puertos locales a un pod.
-  `kubectl proxy`: Ejecuta un servidor proxy para Kubernetes API Server.
-  `kubectl rollout`: Gestiona el despliegue de una aplicación.
-  `kubectl run`: Crea y ejecuta un contenedor en una sola operación.
-  `kubectl scale`: Escala el número de réplicas de un conjunto de réplicas o deployment.
-  `kubectl set`: Establece valores específicos en los objetos del clúster.
-  `kubectl version`: Muestra la versión del cliente y del servidor.

3) Entregar archivo de configuración de Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pod1
  template:
    metadata:
      labels:
        app: pod1
    spec:
      containers:
        - name: pod1
          image: pod1:latest
          imagePullPolicy: Never
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
          ports:
            - containerPort: 3000
          env:
            - name: PORT
              value: "3000"
---
apiVersion: v1
kind: Service
metadata:
  name: pod1
spec:
  selector:
    app: pod1
  ports:
    - port: 5000
      targetPort: 3000
  externalTrafficPolicy: Local
  type: LoadBalancer
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pod2
  template:
    metadata:
      labels:
        app: pod2
    spec:
      containers:
      - name: pod2
        image: pod2:latest
        imagePullPolicy: Never
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 4000
        env:
        - name: PORT
          value: "4000"
---
apiVersion: v1
kind: Service
metadata:
  name: pod2
spec:
  selector:
    app: pod2
  ports:
  - port: 6000
    targetPort: 4000
  externalTrafficPolicy: Local
  type: LoadBalancer
```

4) Entregar Proyectos de NodeJS en Github

https://github.com/JosueJB88/Examen_final

5) Explicación de su uso

Explicación de la configuración de los pod:

Este código describe la configuración de dos pods en Kubernetes. Un pod es la unidad básica de Kubernetes que puede contener uno o más contenedores que se ejecutan juntos en el mismo host y comparten la misma dirección IP y puerto de red.

Cada uno de los pods definidos en este código contiene dos contenedores que ejecutan aplicaciones Node.js diferentes. Cada contenedor se inicia a partir de la imagen de Docker de Node.js y expone un puerto diferente. Los límites de CPU y memoria también se establecen para cada contenedor en la sección de recursos.

Además, se utiliza el balanceo de carga para garantizar que los pods estén distribuidos de manera equilibrada entre los nodos del clúster. La sección de afinidad define una regla de "anti-afinidad", que garantiza que los pods no se programen en el mismo nodo a menos que sea necesario. Esto se hace para evitar la pérdida de disponibilidad si el nodo que hospeda el pod falla.

En resumen, este código describe cómo configurar dos pods en Kubernetes para ejecutar diferentes aplicaciones Node.js, cada una en un contenedor separado, con límites de recursos y balanceo de carga.