Kubernetes (K8S)

Kubernetes es un orquestador de contenedores. Sirve para correr muchos contenedores, autoescalarlos, balancearlos, etc…

COMPONENTES:

Control Plane:

En Kubernetes, el "control plane" se refiere a un conjunto de componentes que gestionan el estado general del clúster y toman decisiones sobre qué acciones deben realizarse en el clúster en función del estado actual. El control plane es responsable de mantener la configuración declarativa del clúster y de responder a los cambios en ese estado.

Kubernetes puede conectarse a la API de un Proveedor Cloud, y hacer cosas como crear un Load Balancer, pedir un Disco Virtual, etc…

Estructura de Kubernetes:

Nosotros instalamos en nuestro ordenador un **cliente de Kubernetes**, que será **kubectl**.

Crearemos un **Clúster de Kubernetes** en un proveedor, como puede ser en la nube, por ejemplo AWS, Azure, GCP, Digital Ocean…, o en local con Minikube.

Dentro del Clúster, encontramos dos organizaciones, los Nodos y el Control Plane.

Control Plane: Contiene la división lógica, y a su vez tiene namespaces para dividir los Pods en “carpetas”.

> **Namespace:** Son carpetas lógicas para almacenar los Pods, Kubernetes tiene varias por defecto, y también se crea la default, donde se despliegan los Pods cuando no especificamos ninguno. Un namespace puede contener Pods de distintos Nodos.

Nodos: Son directamente las Máquinas Virtuales, o Instancias que se hayan creado en el proveedor.

Los Pods: Son unidad de K8s mínima, contienen 1 (normalmente) o más contenedores que comparten recursos de sistema, espacio de red y almacenamiento. Un Pod se encuentra en un namespace y además, se encuentra corriéndose en un Nodo.

Utilización:

Primero, para hacer funcionar Kubernetes, es necesario instalar el cliente, en este caso, el más usado es kubectl.

kubectl: Es la interfaz de línea de comandos (CLI) principal para interactuar con clústeres de Kubernetes. Se utiliza para desplegar y gestionar aplicaciones, inspeccionar y gestionar los recursos del clúster, así como para realizar diversas operaciones administrativas.

minikube: Es una herramienta que facilita la creación y gestión de clústeres de Kubernetes locales. Minikube se utiliza para ejecutar un clúster de Kubernetes en un entorno de desarrollo o prueba en una máquina local. Esto es útil para probar aplicaciones en un entorno Kubernetes sin necesidad de un clúster completo en la nube o en un centro de datos.

Para ver si está instalado, usamos:

kubectl version --client=true

Desde la nube que sea, entonces se crea el Clúster de Kubernetes y luego es necesario conectar kubectl con la nube.

Por ejemplo, para Digital Ocean, descargamos el Config File.

Primero exportamos la variable con:

export KUBECONFIG=~/Downloads/(Config File descargado)

Luego nos conectamos con:

kubectl get nodes

No saldrá una lista con los nodos del clúster.

Otro comando, para mostrar los contextos (url+credenciales) usamos:

kubectl config get-context

MANIFIESTOS DE KUBERNETES:

En Kubernetes, un "namespace" es como una carpeta virtual que ayuda a organizar y separar las cosas dentro de tu clúster. Imagina que estás usando un armario grande para guardar cosas, y dentro de ese armario, puedes crear varias carpetas para organizar mejor tus objetos.

En términos sencillos:

División lógica: Los namespaces son como carpetas que te permiten dividir lógicamente los elementos en tu clúster de Kubernetes.

Evita confusiones: Ayudan a evitar confusiones al asegurarse de que los nombres de tus aplicaciones y recursos no entren en conflicto con los de otras personas que usan el mismo clúster.

Aislamiento: Aunque no proporcionan un aislamiento total, te permiten crear "entornos" virtuales donde los recursos (como aplicaciones y servicios) tienen nombres únicos solo dentro de ese espacio.kubectl get ns: Muestra todos los namespace

Un pod es un set de contenedores, es muy probable que un pod ccorra un solo contenedor, pero puede correr varios. Vas a correr 1 generalmente, pero hay casos en los que tienes 2, como por ejemplo cuando tienes un servicio extra que requiera otro proceso, pero no es ideal.

Para ver los pods:

kubectl -n kube-system get pods

Esto muestra los pods dentro del namespace kube-system. Así como la cantidad de contenedores de cada pods, su status o la edad:

kubectl -n kube-system get pods -o wide

Verbosea más, incluye la IP del pod, el nodo al que pertenece…

Para borrar un pod, usamos el comando:

kubectl -n (namespace) delete pod (nombre_pod)

Kubernetes se puede configurar para que levante el pod cada vez que muera (o lo maten sin querer).

Para ver el estado de un pod y sus eventos:

kubectl describe pod (nombre_pod)

MANIFIESTOS:

Los manifiestos son archivos yaml que especifican el deployment. Es como el yaml del Docker-Compose.

Para aplicar manifiestos de Kubernetes:

kubectl apply -f (archivo.yaml)

Si no especificamos ningún namespace, lo coloca en el namespace “default”

Para ejecutar comando, es muy similar a Docker:

kubectl exec -it (nombre_pod) -- sh

No necesitamos levantar ni instalar ssh, ya que entramos por ssh directamente con la API de Kubernetes.

En los manifiestos podemos poner variables de entorno **env:**

Kubernetes no solo nos permite pasar el valor de variable sino obtenerlo, tiene valores que ya puede heredar, por ejemplo, la IP del host donde va a correr el pod (status.hostIP).

También podemos asignarles recursos (**resoruces:**). Tenemos dos tipos:

- **request:** Recursos garantizados, por ejemplo, memoria y cpu
- **limits:** Son los límites que el pod puede llegar a usar, tanto en memoria como en cpu. Si se pasara de lo establecido, el kernel de Linux se encarga de matar el proceso (no lo hace Kubernetes), eso provoca que el pod se muera y Kubernetes simplemente lo levante. Si vemos que un pod se está callendo mucho, es muy posible que sea debido a esto.
- **Readiness Probe (sonda de preparación):** Se utiliza para determinar cuándo un contenedor está listo para servir tráfico. Antes de enviar tráfico al contenedor, Kubernetes verifica la sonda de preparación para asegurarse de que el contenedor esté en un estado "preparado". Si la sonda de preparación falla, el contenedor no recibe tráfico de servicios externos.

- **Liveness Probe (sonda de vitalidad):** Se utiliza para determinar si un contenedor sigue funcionando correctamente. Si la sonda de vitalidad falla, Kubernetes considera que el contenedor está en un estado no saludable y puede tomar medidas, como reiniciar el contenedor.

Para ver el yaml de un Pod en concreto:

kubectl get pod (nombre_pod) -o yaml

DESPLEGANDO DEPLOYMENTS:

kind: Deployment

Un deployment es una plantilla para crear pods, especificando, por ejemplo, cuantas réplicas debe haber funcionando. Puede correr dos pods en un mismo nodo.

Kind: DaemonSet

Un nodo es la máquina virtual que corre los pods de Kubernetes, ya sea en una nube como Azure, donde usaría VMs o en AWS, donde usaría Instancias EC2.

Un DaemonSet es una forma de desplegar un pod que se desplegará en todos los nodos. Un solo pod por cada nodo. No tiene réplicas. Es ideal para servicios de monitoreo.

kind: StatefulSet

Es una forma de crear pods que además tienen un volumen.

Un volumen es un directorio, que va a hacer de disco. De esta forma no se pierden los datos, ideal para Bases de Datos. Si corriéramos un MySQL en un Deployment, estos datos se perderían al morir el Pod. Sí tiene réplicas.

Se montara en la línea

volumeMounts:

-mounthPath: “/data”

-name: csi-pcv

Un storageClassName es un driver de Kubernetes para un proveedor. Por ejemplo, te permite crear un volumen en un proveedor como Digital Ocean o Azure. Se va a conectar a la API y lo va a conectar al Nodo donde va a correr el Pod, todo automáticamente.

Llamamos a los Volúmenes asignados PVC, para listarlos:

kubectl get pvc

Para borrar un pod StatefulSet, podemos usar:

kubectl delete statefulset (nombre_pod) / kubectl delete sts (nombre_pod)

Cuando borramos un StatefulSet, **NO BORRAMOS** el Volúmen PVC, para ello:

kubectl delete pcv (nombre_pvc)

Esto significa que podríamos manejar muchísimas cosas con Kubernetes, sin necesidad de entrar al proveedor de la nube o usar Terraform. Podríamos crear Bases de Datos completas, aplicaciones, Buckets de S3…

NETWORKING:

Cada Pod tiene su propia IP. Que no es la IP de los nodos, sino una interna. Los contenedores dentro del Pod, no tienen IP propia, hay que tener en cuenta que no pueden abrirse dos puertos iguales de dos contenedores que estén en el mismo pod

Para que los pods se comuniquen entre ellos, necesitamos algo que se llama CNI, Clúster Networking Interface, que es como un agente que corre en cada Nodo, que sirve como para crear una especie de VPN para comunicar los Pods que hay en cada nodo.

Un ejemplo de CNI es Calico, que crea rutas diciéndole a cada Instancia dónde esta el Pod que corresponde a cada IP. Toda esta información está guardado en el Etcd.

Servicios de Kubernetes:

Cluster IP: Es una IP fija que no cambia y que sirve de Load Balancer.

Node Port: Servicio que crea un puerto en cada Nodo y que va a recibir el tráfico y se lo va a mandar a los pods que mande.

Load Balancer: Crea un balanceador de carga en nuestro proveedor de nube y redirecciona el tráfico a los distintos Pods.

CLUSTER IP:

Crea una IP que está dentro del Clúster. No la expone, es ideal para conectar entre pods.

Esta es la de por defecto si no lo expecificamos en spec: type:

Como la IP del Pod cambia cuando este muere, no se puede crear un servicio que apunte directamente a esta IP. Para eso usamos **los servicios**, que serán los que estén en “primera línea de batalla”, el que recibirá las peticiones y busacará el pod.

Los Servicios Balancean la carga, tienen una IP que no cambia, contrario a la de los Pods. Para ver la IP:

kubectl describe svc (nombre-servicio)

NODE PORT:

Crea un servicio que encuentra nuestros Pods basado en nuestras etiquetas, necesita tener una IP pública.

Crea un puerto en CADA nodo (aunque el Pod no corra en todos los nodos). Hace lo mismo que Cluster IP, balancea y mantiene la IP. Para ver los nodos:

kubectl get nodes -o wide

NO es útil para exponer aplicaciones públicas.

Load Balancer:

Para verlos:

kubectl get svc

Cuando se despliega, despliega en la nube, como por ejemplo AWS o Digital Ocean, un Load Balancer, por lo que tarda un poco más en crearse. Esto es mejor que un Node Port está atado a la IP de cada nodo, sin embargo, la IP del Load Balancer es persistente.

No es ideal para exponer al pública, es mejor hacer un Ingress.

Ingress:

Necesita una instalación aparte, ya sea en local, por ejemplo, con Helm, o directamente en la nube.

Es un tipo de recurso que nos permite crear acceso a nuestros servicios basados en el Path. Lo que hace ingress es que va a agregar un backend a cada uno de los path.

Ingress en Kubernetes es como un portero que decide a dónde enviar las solicitudes cuando alguien llega a la puerta de tu aplicación. Cuando alguien (un usuario, un navegador web, etc.) intenta acceder a tu aplicación desde fuera del clúster de Kubernetes, el Ingress toma decisiones sobre a qué servicio interno dirigir esa solicitud.

BASICAMENTE: Podemos poner que un deployment se acceda desde http://(ip_publica)/primera_carpeta y que acceda a deployment (o lo que sea) con http://(ip_publica)/segunda_carpeta

Todo esto se puede configurar con una serie de reglas para especificar dónde va cada cosa (al "portero") y por ejemplo, decirle a Ingress que no acepte peticiones de determinado tipo, como solo dejar pasar las que cumplan TSL/SSL con HTTPS.

Para ver los recursos ingress:

kubectl get ing

Cuando instalas Ingress, por lo menos en Digital Ocean, también crea un Load Balancer por defecto que recibirá el tráfico de todo las peticiones a Ingress.

Hay que instalarlo a parte, ya sea con Helm o en la nube directamente.

ConfigMap:

Es un archivo que se hostea en Kubernetes y que se puede acceder desde el Pods. Usado para guardar archivos de configuración.

Esto nos permite usar los valores guardados como variables de entorno.

SECRET:

Es como el ConfigMap, pero codificado en base64. ESTO NO ES SEGURO, pero al menos puede evitar una lectura rápida por parte de ojos humanos. Serviría para poder poner en un repositorio el manifiesto, aunque esto también se puede hacer con el ConfigMap.

Kustomization:

Es una forma de manejar manifiestos más fácilmente. Nos permite con un cliente (kubectl) generar manifiestos.

Nos permite generar manifiestos en base a otros, sin necesidad de cambiar el manifiesto. Es inteligente.

Para usarlo:

kustomize build .

Puedes aplicarlo directamente con:

kustomize build . | kubectl apply -f -

Cada vez que hace un cambio, también cambia el hash del archivo kustomization. Así se asegura que el pod tiene que ser recreado y evita errores.

Stern:

Cuando vemos un log de un pod con:

kubectl logs -f (nombre_pod)

NO es posible usar caracteres comodín como “pod_nginx_*” (suponiendo que existan varios pods que empiecen por “pod_nginx_xxxxxx”. Debemos poner el nombre completo.

Stern (se descarga a parte) es un programita que sirve para esto:

stern pod_nginx

Este comando buscará todos los pods que contengan pod_nginx en su nombre. Devuelve los logs de todos esos pods y pone un color para cada uno de ellos.