Kubernetes.

Paradigma

Contenido.

Autores y contactos.

Fernando Cabadas.

Data Architect.

fcabadas@paradigmadigital.com

01.

Kubernetes

Introduction.

¿Qué es Kubernetes?.

El nombre **Kubernetes** proviene del griego y significa *timonel* o *piloto*.

Kubernetes ofrece un entorno de administración centrado en **contenedores**. Kubernetes orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Una definicón muy simple sería decir que Kubernetes es un **orquestador** de contenedores.

Vamos a enumerar ciertos conceptos importantes.

Contenedor. Incluye las aplicaciones y entornos de software.
Pod. Este elemento de la arquitectura de kubernetes se encarga de agrupar aquellos contenedores que necesitan trabajar juntos para el funcionamiento de una aplicación.
Nodo. Uno o varios pods se ejecutan en un nodo, que puede ser tanto una máquina virtual como física.
Clúster: Es una agrupación de nodos.

02.

Kubernetes

Installation.

Instalación.

(https://minikube.sigs.k8s.io/docs/start/)

Linux

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

Mac

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64

sudo install minikube-darwin-amd64 /usr/local/bin/minikube

Windows

https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe

Iniciamos minikube con el comando *minikube start*

03.

Kubernetes

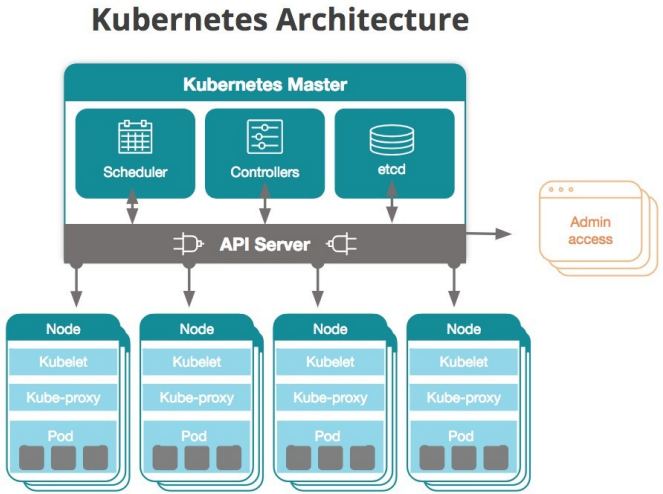
Architecture.

Arquitectura.

La arquitectura de kubernetes se despliega sobre un **clúster** de máquinas (nodos) y está basada en el principio de **maestro/esclavo**.

Los **nodos** se utilizan como esclavos, es decir, que son las partes controladas del sistema y están bajo la administración y el control de los maestros de Kubernetes.

Una de las tareas de un maestro, por ejemplo, es distribuir los pods en los nodos.



Siguiendo con la arquitectura vamos a definir los procesos que se ejecutan tanto en el maestro como en los nodos esclavos y así entender un poco mas como funciona kubernetes.

Nodo maestro

Podemos encontrar los siguientes componentes

kube-apiserver expone la API de kubernetes

etcd almacén de datos persistente, consistente y distribuido de clave-valor utilizado para almacenar toda a la información del clúster de Kubernetes

kube-controller-manager ejecuta los controladores de kubernetes e incluye los siguientes procesos

- **node controller** es el responsable de detectar y responder cuándo un nodo deja de funcionar
- **replication controller** es el responsable de mantener el número correcto de pods para cada controlador de replicación del sistema
- **endpoints controller** responsable de unir servicios y pods
- **service Account & token controllers** crean cuentas y tokens de acceso a la API por defecto para los nuevos Namespaces.

kube-scheduler componente que vigila los pods recién creados sin nodo asignado y selecciona un nodo para que se ejecuten en él

Nodos esclavos

Podemos encontrar los siguientes componentes

kubelet se asegura de que los contenedores se estén ejecutando dentro del pod
kube-proxy es un proxy de red que se ejecuta en cada nodo
container runtime se encarga de ejecutar los contenedores

Container runtime

Es necesario instalar un container runtime en cada nodo del clúster para que los Pods puedan ejecutarse allí.

Nosotros en minikube tenemos instalado *Docker Engine* pero existen otros.

- containerd
- CRI-O
- Docker Engine
- Mirantis Container Runtime

04.

Kubernetes

Concepts.

YAML

Antes de entrar en materia sobre los diferentes objetos que existen en kubernetes hay que decir que el lenguaje utilizado para la creación de estos objetos es YAML. A continuación una descripción copiada.

“YAML (Yet Another Markup Language) es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado en RFC 2822. YAML fue propuesto por Clark Evans en 2001, quien lo diseñó junto a Ingy döt Net y Oren Ben-Kiki”

Es lo mas parecido a un lenguaje de etiquetas.

Es recomendable introducirse un poco en este lenguaje para entender mejor como crear los diferentes objetos de kubernetes aunque para seguir estas sesiones no sea imprescindible.

kubectl

También antes de comenzar a crear objetos de kubernetes debemos saber la existencia del comando kubectl.

kubectl es la herramienta de línea de comandos de Kubernetes.

Se utiliza para desplegar y gestionar aplicaciones en Kubernetes. Usando kubectl, puedes inspeccionar recursos del clúster. Crear, eliminar, y actualizar componentes. Explorar tu nuevo clúster y arrancar aplicaciones de ejemplo.

Para desplegar cualquier recurso de kubernetes ejecutaremos lo siguiente

kubectl apply -f “nombre del fichero yaml”

Kubernetes API

https://kubernetes.io/docs/reference/using-api/

05.

Kubernetes

Core.

Pod.

https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

La unidad más pequeña de kubernetes son los **Pods**, con los que podemos ejecutar contenedores. Un **pod** representa un conjunto de contenedores que comparten almacenamiento y una única IP. Los pods son efímeros, cuando se destruyen se pierde toda la información que contenía. Si queremos desarrollar aplicaciones persistentes tenemos que utilizar volúmenes.

Exploramos *el fichero pod.yml*
vi pod.yml
Editar / Crear *el pod a partir del fichero yaml*
kubectl apply -f pod.yml
Visualizar *los pods*
kubectl get pods
Editar *el pod creado*
kubectl edit pod mipod
Acceder *al pod*
kubectl exec -it mipod -- sh
Eliminar *el pod*
kubectl delete pod mipod

Namespace.

(https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

Los espacios de nombres son una forma de dividir los recursos del clúster. Proporcionan un alcance para los nombres. Los nombres de los recursos deben ser únicos dentro de un espacio de nombres.

Crear namespace
kubectl create ns paradigma
Listar namespaces
kubectl get ns
Exploramos el fichero namespacePod.yml
vi namespacePod.yml
Crear el pod con el namespace asignado
kubectl apply -f namespacePod.yml
Visualizar *los pods del namespace creado*
kubectl get pods -n paradigma
Explorar *la definición del pod con el namespace asignado*
kubectl describe pod minamespacepod -n paradigma

06.

Kubernetes

Configuration.

ConfigMap.

https://kubernetes.io/docs/concepts/configuration/configmap/

Kubernetes ofrece una funcionalidad que ayuda a mantener las configuraciones de las aplicaciones en forma de ConfigMaps. Permite desacoplar la configuración de los propios objetos.

Exploramos *el fichero configMap.yml*
vi configMap.yml
Crear *el ConfigMap a partir del fichero*
kubectl apply -f configMap.yml
Visualizar *los ConfigMaps*
kubectl get cm -n paradigma
Editar *el ConfigMap creado*
kubectl edit cm miconfigmap -n paradigma
Eliminar *el ConfigMap creado*
kubectl delete cm miconfigmap
Explorar *la definición del ConfigMap*
kubectl describe cm miconfigmap -n paradigma

A continuación vemos como pasar datos de un ConfigMap a un contenedor dentro de un pod como una variable de entorno

Exploramos *el fichero configMapPod.yml*

vi configMapPod.yml

Crear *el pod a partir del fichero configMapPod.yml*

kubectl apply -f configMapPod.yml

Visualizar *los pods*

kubectl get pods -n paradigma

Visualizamos *los logs del pod creado para ver el ECHO de la variable*

kubectl logs miconfigmappod -n paradigma

Secrets.

https://kubernetes.io/docs/concepts/configuration/secret/

Siempre es importante almacenar datos confidenciales, como tokens, contraseñas y claves, de forma segura y encriptada. Con el objeto Secret podemos almacenar estos datos de forma segura y proporcionarlos a los contenedores.

Exploramos *el fichero secret.yml*
vi secret.yml
Crear *el Secret a partir del fichero*
kubectl apply -f secret.yml
Visualizar *los Secrets*
kubectl get secrets -n paradigma
Editar *el Secret creado*
kubectl edit secret misecret -n paradigma
Eliminar *el Secret creado*
kubectl delete secret misecret -n paradigma
Explorar *la definición del ConfigMap*
kubectl describe secret misecret -n paradigma

A continuación vemos como pasar datos de un Secret a un contenedor dentro de un pod como una variable de entorno

Exploramos *el fichero secretPod.yml*

vi secretPod.yml

Crear *el pod a partir del fichero secretPod.yml*

kubectl apply -f secretPod.yml

Visualizar *los pods*

kubectl get pods -n paradigma

Visualizamos *los logs del pod creado para ver el ECHO de la variable*

kubectl logs misecretpod -n paradigma

Resource requirements.

A continuación veremos como administrar y utilizar los recursos disponibles para ejecutar contenedores. Las solicitudes y límites de recursos proporcionan un gran control sobre cómo se asignarán los recursos. Estableceremos los límites de recursos a utilizar por el Pod.

Exploramos *el fichero resourcePod.yml*
vi resourcePod.yml
Crear *el Pod a partir del fichero*
kubectl apply -f resourcePod.yml
Visualizar *los Pods*
kubectl get pods -n paradigma

Cuando especificas el recurso **request** para Contenedores en un Pod, el Scheduler de Kubernetes usa esta información para decidir en qué nodo colocar el Pod y reserva al menos la cantidad especificada en request para el contenedor. Cuando especificas el recurso **limit** para un Contenedor, Kubernetes impone estos límites, así que el contenedor no puede utilizar más recursos que el límite que le definimos.

Los tipos de recurso que se pueden especificar son **CPU** y **memoria**. La **memoria** se mide en PetaByte, TeraByte, GigaByte, MegaByte y KiloByte y la **CPU** se mide en unidades de cpu. Una cpu, en Kubernetes, es equivalente a **1 vCPU/Core** para proveedores de cloud y **1 hyperthread** en procesadores Intel. La expresión 0.1 es equivalente a la expresión 100m en CPU.

Resource Quota.

https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/

A continuación veremos como administrar las cuotas de uso dentro de un namespace. Esto permitirá limitar el uso de los recursos del cluster por parte de los pods. Los recursos kubernetes permite limitar son el uso de **cpu** y el uso de **memoria**.

Exploramos *el fichero resourceQuota.yml*
vi resourceQuota.yml
Crear *el ResourceQuota a partir del fichero*
kubectl apply -f resourceQuota.yml
Visualizar *los ResourceQuota*
kubectl get resourcequota -n paradigma
Editar *el ResourceQuota creado*
kubectl edit resourcequota miresourcequota -n paradigma
Eliminar *el ResourceQuota creado*
kubectl delete resourcequota miresourcequota -n paradigma
Explorar *la definición del ResourceQuota*
kubectl describe resourcequota miresourcequota -n paradigma