10.

Kubernetes

Services.

Services.

https://kubernetes.io/docs/concepts/services-networking/service/

Los **deployment** facilitan la creación de un conjunto de pods en réplica que se pueden escalar, actualizar y reemplazar dinámicamente. Sin embargo, es difícil proporcionar acceso de red a esos pods para otros componentes. Los **services** proporcionan una capa de abstracción que resuelve este problema. Los clientes simplemente pueden acceder al service, que representa dinámicamente el tráfico al conjunto actual de réplicas. Los services utilizan la etiqueta **selector** para buscar los pods que tengan esa **label** y dar acceso a ellos.

Existen 3 tipos de servicios.
ClusterIP expone el Service en una dirección IP interna del clúster. Al escoger este valor el Service solo es alcanzable desde el clúster. Este es el ServiceType por defecto.
NodePort expone el Service en cada IP del nodo en un puerto estático (el NodePort). Automáticamente se crea un Service ClusterIP, al cual enruta el NodePort del Service. Podrás alcanzar el Service NodePort desde fuera del clúster, haciendo una petición a **<NodeIP>:<NodePort>**. Si no especificamos el nodeport en el yml kubernetes asigna uno automáticamente en el rango **30000:40000**
LoadBalancer expone el Service externamente usando el balanceador de carga del proveedor de la nube. Son creados automáticamente Services NodePorty ClusterIP, a los cuales el apuntará el balanceador externo. Solo en **CLOUD**.
ExternalName mapea el Service al contenido del campo externalName (ej. paradigmadigital.com), al devolver un registro CNAME con su valor. No se configura ningún tipo de proxy.

ClusterIP.

Exploramos *el fichero deploymentClusterIp.yml*

vi deploymentClusterIp.yml

Crear *el deployment a partir del fichero*

kubectl apply -f deploymentClusterIp.yml

Exploramos *el fichero serviceClusterIp.yml*

vi serviceClusterIp.yml

Crear *el service a partir del fichero*

kubectl apply -f serviceClusterIp.yml

Visualizar *los pods que ha generado el deployment*

kubectl get pods -n paradigma --show-labels

Visualizar *el service que hemos generado*

kubectl get svc -n paradigma

kubectl describe svc miserviceclusterip -n paradigma

NodePort.

Exploramos *el fichero deploymentClusterNodeport.yml*

vi deploymentClusterNodeport.yml

Crear *el deployment a partir del fichero*

kubectl apply -f deploymentClusterNodeport.yml

Exploramos *el fichero serviceClusterNodeport.yml*

vi serviceClusterNodeport.yml

Crear *el service a partir del fichero*

kubectl apply -f serviceClusterNodeport.yml

Visualizar *los pods que ha generado el deployment*

kubectl get pods -n paradigma --show-labels

Visualizar *el service que hemos generado*

kubectl get svc -n paradigma

kubectl describe svc miservicenodeport -n paradigma

Acceder *al service desde nuestra máquina*

curl http://<node-ip>:<node-port>

11.

Kubernetes

Persistence.

Volume.

https://kubernetes.io/docs/concepts/storage/volumes/

En Kubernetes, un **volumen** se puede considerar como un directorio al que pueden acceder los contenedores de un pod. Tenemos diferentes tipos de volúmenes y el tipo define cómo se crea el volumen y su contenido.

Los volúmenes no se limitan a ningún contenedor. Soporta cualquiera o todos los contenedores desplegados dentro del pod de Kubernetes. Una ventaja clave del volumen en Kubernetes es que admite diferentes tipos de almacenamiento en los que el pod puede utilizar varios de ellos al mismo tiempo.

Exiten muchos tipos de volúmenes en Kubernetes
awselasticblockstore, azuredisk, azurefile, cephfs, cinder, configmap, downwardapi, ***emptydir****, fc, flocker, gcepersistentdisk, gitrepo, glusterfs, hostpath, iscsi, local, nfs, persistentvolumeclaim, portworxvolume, projected, quobyte, rbd, secret, vspherevolume*

Un volumen **emptyDir** se crea por primera vez cuando se asigna un Pod a un nodo, y existe mientras ese Pod se esté ejecutando en ese nodo. Como su nombre indica, el volumen emptyDir está inicialmente vacío. Todos los contenedores del Pod pueden leer y escribir los mismos archivos en el volumen emptyDir, aunque ese volumen puede ser montado en la misma o diferente ruta en cada contenedor. Cuando un Pod es eliminado de un nodo por cualquier razón, los datos en el emptyDir son borrados permanentemente.

Exploramos *el fichero volumePod.yml*

vi volumePod.yml

Crear *el pod a partir del fichero*

kubectl apply -f volumePod.yml

Listamos *la carpeta /tmp de ambos contenedores*

kubectl exec -it mivolumepod -c ubuntu -n paradigma -- ls -l /tmp

kubectl exec -it mivolumepod -c nginx -n paradigma -- ls -l /tmp

Crear *un fichero en la carpeta /tmp del contenedor ubuntu*

kubectl exec -it mivolumepod -c ubuntu -n paradigma -- touch /tmp/ubuntu/test.json

Listamos *la carpeta tmp/ubuntu del contenedor ubuntu*

kubectl exec -it mivolumepod -c ubuntu -n paradigma -- ls -l /tmp/ubuntu

Listamos *la carpeta tmp/nginx del contenedor nginx*

kubectl exec -it mivolumepod -c nginx -n paradigma -- ls -l /tmp/nginx

State persistence.

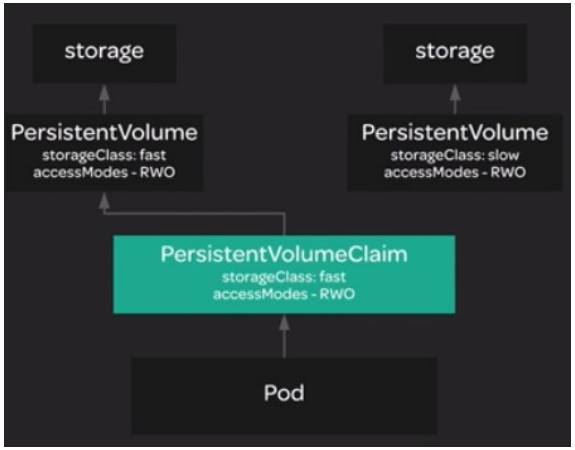
https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Cuando se elimina un contenedor, los datos almacenados dentro del disco interno del contenedor se pierden.
Kubernetes nos permite implementar el almacenamiento persistente utilizando **PersistentVolume** y **PersistentVolumeClaims**.

PersistentVolume (PV) representa un recurso de almacenamiento
PersistentVolumeClaim (PVC) capa de abstracción entre el pod y el PersistentVolume

Los PVCs se vincularán automáticamente a un PV que tenga StorageClass y accessModes compatibles.

Si creamos un PVC con un storageClass:fast y un accessModes:RWO va a buscar un PV con esas carácterísticas y que esté disponible y automáticamente se enlaza con él y es el PVC el que configuraremos al crear el YML del pod.



PersistentVolume.

Vamos a explicar las **labels** importantes del PV

storageClassName define diferentes categorías de almacenamiento
capacity cantidad de almacenamiento para el PV
accessModes determina qué modos de lectura/escritura se pueden utilizar para montar el volumen
hostPath utiliza el sistema de archivos local del nodo para el almacenamiento

Exploramos *el fichero persistentVolume.yml*
vi persistentVolume.yml
Crear *el PV a partir del fichero*
kubectl apply -f persistentVolume.yml
Visualizar *el PV que hemos creado*
kubectl get pv

PersistentVolumeClaim.

Vamos a explicar las **labels** importantes del PVC

storageClassName define diferentes categorías de almacenamiento
accessModes determina qué modos de lectura/escritura se pueden utilizar para montar el volumen
resources.requests.storage define la cantidad de almacenamiento que necesita el **claim**

Exploramos *el fichero persistentVolumeClaim.yml*
vi persistentVolumeClaim.yml
Crear *el PVC a partir del fichero*
kubectl apply -f persistentVolumeClaim.yml
Visualizar *el PVC que hemos creado*
kubectl get pvc

PersistentVolumeClaim Pod.

Exploramos *el fichero persistentVolumePod.yml*

vi persistentVolumePod.yml

Crear *el pod a partir del fichero*

kubectl apply -f persistentVolumePod.yml

Visualizar *el pod que hemos creado*

kubectl get pods -n paradigma

Crear *un fichero en la carpeta del pod que hemos indicado en el volume*

kubectl exec -it mipvcpod -n paradigma -- touch /mnt/storage/testvolume.json

Eliminar *el pod y vemos que el PV y el PVC siguen existiendo*

kubectl delete pod mipvcpod -n paradigma

kubectl get pv & kubectl get pvc -n paradigma

Modificar *mipvcpod y cambiamos la carpeta del volumen*

Crear *de nuevo el pod a partir del fichero*

kubectl apply -f persistentVolumePod.yml

Listar *la carpeta nueva*

kubectl exec -it mipvcpod -n paradigma -- ls -l /mnt/storage

Los datos los encontramos en la ruta de nuestro ordenador ***/var/lib/docker/overlay2*** *sudo find / -name testvolume.json*

StatefulSet.

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

Al igual que un **Deployment**, un **StatefulSet** gestiona Pods que se basan en una especificación idéntica de contenedor. A diferencia de un Deployment, un StatefulSet **mantiene una identidad asociada a sus Pods**. Estos pods se crean a partir de la misma especificación, pero no pueden intercambiarse; cada uno tiene su propio identificador persistente que mantiene a lo largo de cualquier re-programación.

El statefulset tiene las siguientes **limitaciones**.

- El almacenamiento para un Pod determinado **debe** ser aprovisionado por un **PersistentVolume**.
- Al eliminar un statefulset no se elimina el volumen asociado. Esto se hace para garantizar la seguridad de los datos.
- Kubernetes no asegura la eliminación de los pods correspondientes al statefulset cuando el mismo es eliminado.
- Los StatefulSets requieren un servicio responsable de la identidad de red de los Pods. Hay que crearlo manualmente.

Kubernetes crea un PersistentVolume para cada VolumeClaimTemplate. Cuando un Pod se (re)programa en un nodo, sus volumeMounts montan los PersistentVolumes asociados con sus PersistentVolume Claims. Nótese que los PersistentVolumes asociados con los PersistentVolume Claims de los Pods no se eliminan cuando los Pods, o los StatefulSet se eliminan. Esto debe realizarse manualmente.

Exploramos *el fichero stateFulset.yml*

vi stateFulset.yml

Crear *el stateFulset a partir del fichero*

kubectl apply -f stateFulset.yml

Visualizar *los pods que ha generado el statefulset*

kubectl get pods -n paradigma

Visualizar *los PVCs que ha generado el statefulset*

kubectl get pvc -n paradigma

Visualizar *los PVs que ha generado el statefulset*

kubectl get pv

Antes de terminar ejecutamos el siguiente comando ***minikube dashboard*** que nos permiten visualizar en un entorno gráfico todos los objetos de nuestro cluster

¡Muchas Gracias!