Introducción a kubernetes

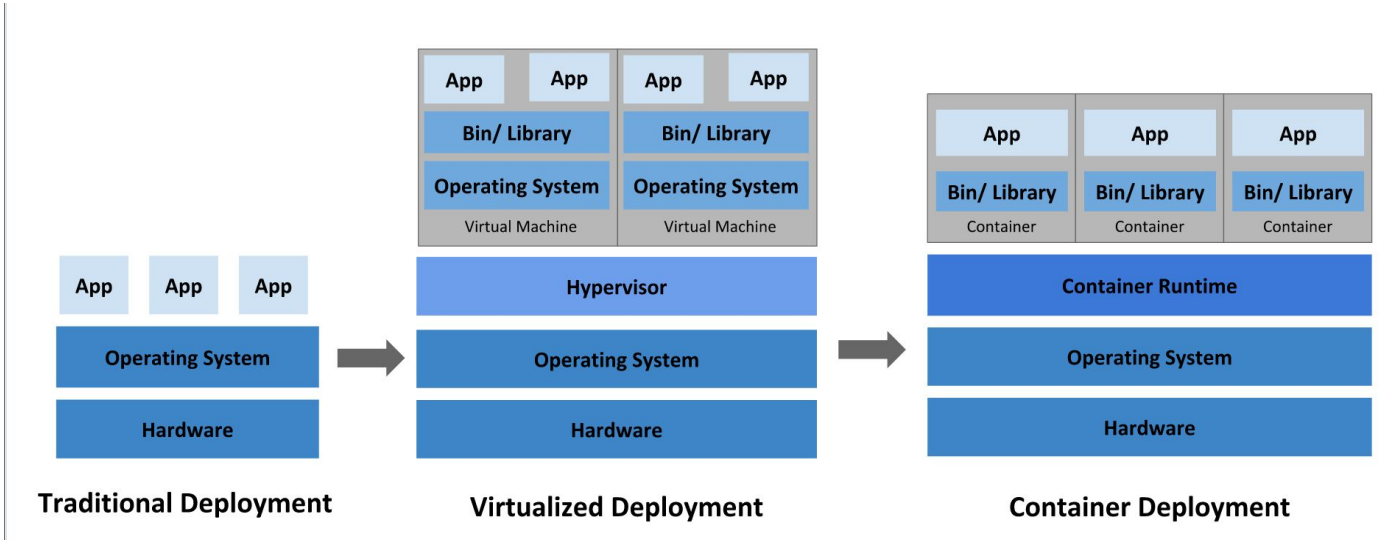
Sistemas Distribuidos

Marcos Novalbos
Marcos.novalbos@live.u-tad.com

Introducción

- Kubernetes es una herramienta de código abierto, diseñada para poder facilitar el despliegue de contenidos y servicios.

- Se basa en el uso de contenedores que aislan las aplicaciones, reaprovechando el sistema operativo en el que estén desplegadas



Introducción

- Qué cosas provee:
  - **Descubrimiento de servicios y balanceo de carga**
  - Permite que las aplicaciones se "anuncien" usando su propia IP o un DNS. Además, puede balancear el tráfico de datos entre varias máquinas/aplicaciones dentro de su red.
  - **Manipulación y configuración del almacenamiento**
  - Kubernetes permite montar automáticamente un sistema de almacenamiento bajo demanda, discos duros locales, remotos (en la nube), etc...
  - **Vuelta atrás a estados anteriores**
  - Se pueden crear "contenedores" con un estado deseado, realizar modificaciones y volver a estados anteriores del mismo.
  - **Empaquetado automático de contenedores**
  - Se puede configurar un cluster de nodos con kubernetes, y configrar sus características. Se puede seleccionar cuanta RAM y cuantas CPUs se usarán en cada uno de los contenedores.
  - **Auto reparado**
  - Puede detectar contenedores que estén fallando, hayan caido o tengan problemas. Se pueden parar y reiniciar atomáticamente.
  - **Accesos seguros**
  - Kubernetes puede gestionar contraseñas y accesos seguros para las aplicaciones que se vayan a distribuir.

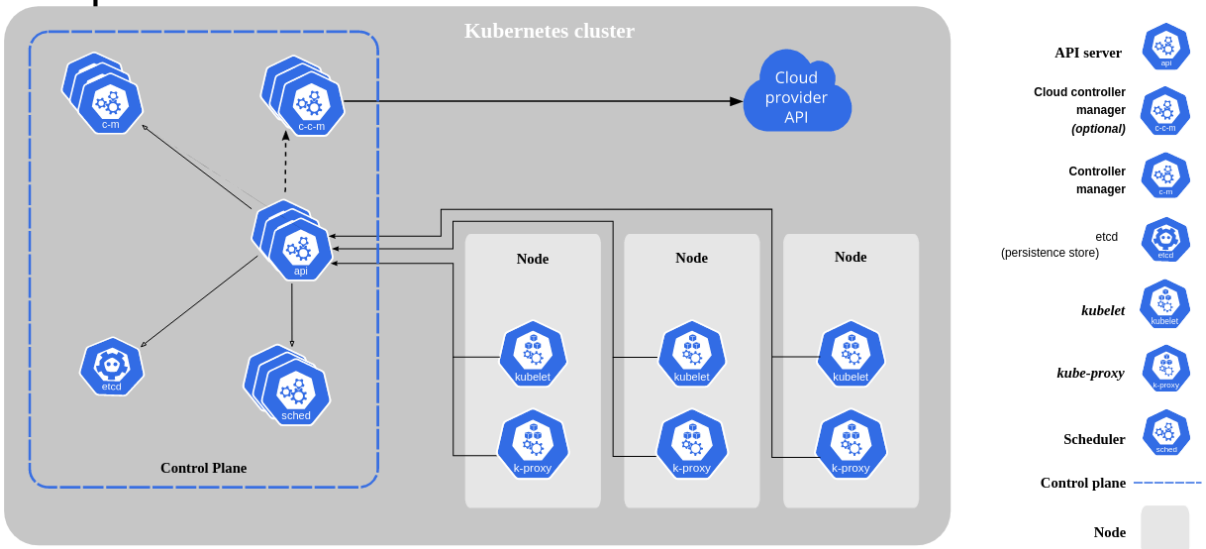
Introducción

- Qué cosas NO provee:
  - No despliega código fuente, ni compila aplicaciones
  - No es un middleware de programación: No provee api de paso de mensajes, sistemas de almacenamiento paralelo, aplicaciones extras como bases de datos... Aunque se podrían instalar usando sus contenedores
  - No recolecta logs, monitorea, alerta...

Cluster de kubernetes

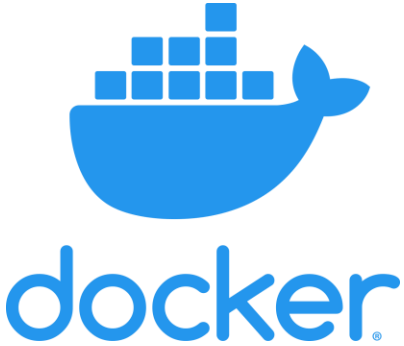
- Un clúster de Kubernetes está formado por nodos de trabajo, que ejecutan aplicaciones en contenedores. Cada clúster tiene al menos un nodo de trabajo.

- Cada nodo aloja los "pods", que son los componentes que conforman la aplicación. El plano de control (control plane) administra los nodos y los pods en el clúster. En entornos de producción, el plano de control generalmente se ejecuta en varios equipos y un clúster generalmente ejecuta varios nodos, lo que proporciona tolerancia a fallos y alta disponibilidad.



Cluster de kubernetes

- **Container:**
  - La unidad básica de despliegue es un contenedor. Consiste en un conjunto de aplicaciones empaquetadas listas para ejecutarse. Contienen todas las librerías y configuraciones instaladas, permitiendo moverse sin necesidad de reconfiguración.
- **Container runtime:**
  - Se usará docker para la creación y configuración de contenedores

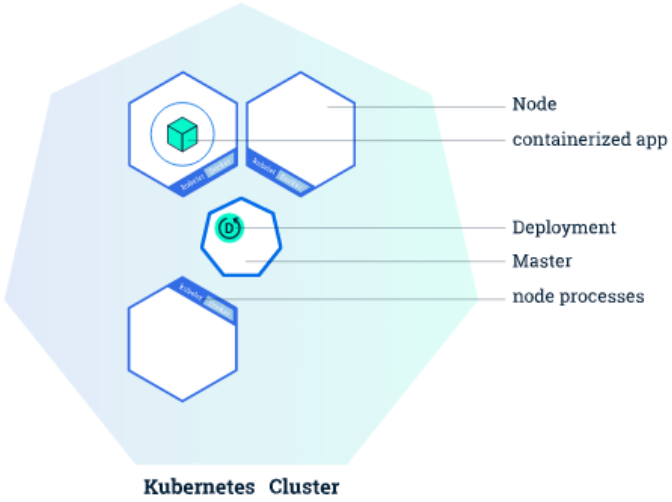


Cluster de kubernetes

- **Pods**
  - Los pods son las unidades de despliegue de aplicaciones manejadas por kubernetes. Consiste en un grupo de uno o más contenedores, que permiten compartir recursos en red. Idealmente, puede representar un conjunto de aplicaciones que trabajan en común, dentro de uno o varios nodos de red
    - Pods que ejecutan un único contenendor: Idealmente, un nodo de la red con una sola imagen de Docker ejecutando sus aplicaciones (lo más común)
    - Pods con varios contenedores trabajando juntos: En algunos casos, se pueden unir varios contenedores para tener una aplicación más grande, pero que no saldrá de este pod. (Casos muy complejos)
- Los pods suelen ser "volátiles": Se crean bajo demanda, realizan su función y se destruyen cuando ya no se necesitan.

Cluster de kubernetes

- **Deployments**
  - Los deployments representan los despliegues de aplicaciones. Están formados por uno o varios pods, ejecutándose en una o varias máquinas. El deployment ofrece un punto de entrada/interacción con la aplicación, que puede redirigirse a alguno de los pods del sistema. En caso de haber pods replicados, intentará balancear las peticiones entre varios nodos.



Capa de red: proveedores CNI

- Para poder conectar las instancias de docker entre los distintos Pods/nodos, es necesario un programa externo.
- Por sí mismo, kubernetes no gestiona la capa de red virtual. Provee una capa "CNI" (Cluster Network Interface), que cualquiera pueda implementar:
  - Weave: Flexibilidad en configuración de la red
  - Flannel: Facilidad de uso, muy popular
  - Calico: Centrado en eficiencia de comunicaciones
  - Canal

Instalación

- En el laboratorio se usará Ubuntu Server 20.04. Los alumnos son libres de usar otra distribución, pero se aconseja ésta para asegurar la efectividad de los comandos usados durante el desarrollo de la asignatura.

- PRERREQUISITOS:
  - Máquina virtual con ubuntu server instalada
  - Configurada con al menos 2 CPUs y 2GB de RAM
  - Disco duro de al menos 10GB de espacio. Al menos 3GB libres de espacio para asegurar que no hay problemas
  - Instalación de linux QUE TENGA LA SWAP DESACTIVADA
  - Varias instancias de linux en red (al menos 2 en los siguientes ejemplos)

Instalación

- Puertos abiertos

TCP	Inbound	6443	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10259	kube-scheduler	Self
TCP	Inbound	10257	kube-controller-manager	Self
TCP	Inbound	30000-32767		

Instalación

- Preparación de las instancias de linux usadas:
  - Dado que se usarán las versiones oficiales de docker y kubernetes, se aconseja desinstalar versiones anteriores preinstaladas por Ubuntu. Por defecto, instala microk8s y docker a través de snap. Los alumnos son libres de usarlas, pero no se verá en los ejemplos.
  - Desinstalando (en caso de haberlas instalado):
    - sudo apt purge docker*
    - sudo rm -r /etc/docker/
    - sudo rm -r /etc/apt/sources.list.d/docker.list
    - sudo apt purge containerd
    - sudo rm -r /etc/containerd

- En caso de no haberlas instalado previamente, esos comandos no hacen nada

Instalación

- Para que el complemento "kubelet" de kubernetes funcione correctamente, es necesario desactivar la swap de nuestras máquinas. Para ello, primero se desactiva la swap que está en el fichero "/swap.img" de las instalaciones linux del laboratorio:
  - `sudo swapoff /swap.img`
- Y luego, se elimina su referencia del fichero "fstab", que intentaría montarlo en siguientes reinicios:
  - `sudo nano /etc/fstab`
- Por último, se borra el fichero /swap.img, para liberar espacio.
  - `sudo rm /swap.img`

Instalación

- En este punto ya se tiene la máquina preparada para instalar kubernetes. El primer paso es añadir el repositorio de kubernetes y docker a la lista de repositorios de APT:

  - `sudo apt-get update && sudo apt-get install -y apt-transport-https gnupg2`
  - `curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -`
  - `echo "deb https://apt.kubernetes.io kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list`
  - `sudo apt-get update`

- E instalar los paquetes:
  - `sudo apt-get install -y kubectl kubeadm kubernetes-cni containerd docker.io conntrack`

Instalación

- Añadir la configuración de red y gestión de procesos para Kubernetes y docker
  - Editar/crear archivo /etc/sysctl.d/kubernetes.conf y escribir:
    - `net.bridge.bridge-nf-call-ip6tables = 1`
    - `net.bridge.bridge-nf-call-iptables = 1`
    - `net.ipv4.ip_forward = 1`
  - Editar/crear archivo /etc/docker/daemon.json y escribir

```
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
```

- Y activar sus servicios, para posteriores reinicios:
  - `sudo systemctl daemon-reload`
  - `sudo systemctl restart docker`
  - `sudo systemctl enable docker.service`
  - `sudo systemctl enable kubelet.service`

Instalación

- En este punto ya están instalados los paquetes, pasamos a iniciar kubernetes. El primer paso es descargar las imágenes con preconfiguraciones de sus componentes. Cada uno de ellos es un contenedor que se quedará guardado en la máquina. Puede tardar varios minutos, depenterá de la conexión, pero sólo hace falta descargarlos la primera vez:
  - `sudo kubeadm config images pull`
- Y se lanza los servicios de control de kubernetes:
  - `sudo kubeadm init --ignore-preflight-errors=NumCPU,Swap`
- La opción para ignorar errores sólo es necesaria si no se desactivó la swap (prerrequisito obligatorio) o si no se tienen CPUs suficientes (opcional).

Instalación

- El comando anterior genera un nodo máster en el ordenador donde se ejecutó, al que se podrán añadir otros nodos esclavos. Éste nodo máster será el que reparta los programas/pods entre los distintos esclavos. El mismo nodo máster también puede actuar como esclavo, sólo es una distinción.
- Para poder añadir más nodos a la red se necesita la IP del nodo máster, un toquen de identificación y una clave/certificado de acceso. Esos datos nos los da el comando "kubeadm init" al final de su ejecución y hay que apuntarlo (no se salva). Tiene el siguiente aspecto:

```
kubeadm join IP_MASTER:PUERTO --token xxxxxxxxxxxx   --discovery-token-ca-cert-hash
sha256:yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
```

- Si se necesita volver a generar un token de para poder unirse a la red (se perdió la clave anterior):

```
kubeadm token create --print-join-command
```

Instalación

- Por último, hay que configurar el directorio del usuario que realizará las ejecuciones de Kubernetes (en nuestro caso, user1). Para eso, deberemos crear una configuración en un directorio local al directorio $HOME del usuario, por defecto se usará ".kube". En caso de haber hecho una instalación correcta en los pasos anteriores, podremos usar una de las configuraciones por defecto que se guardan en el directorio /etc/kubernetes:
  - `mkdir -p $HOME/.kube`
  - `sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
  - `sudo chown $(id -u):$(id -g) $HOME/.kube/config`

Instalación

- Por último, queda configurar un gestor de la capa de transporte de kubernetes, se usará weave. Para instalarlo y configurar kubernetes, es OBLIGATORIO que los comandos anteriores hayan funcionado correctamente (depende de la versión de kubectl y que haya una configuración de kubernetes donde guardar los resultados, debería estar instalado en este momento).

  - `sudo curl -L git.io/weave -o /usr/local/bin/weave`
  - `sudo chmod a+x /usr/local/bin/weave`
  - `kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml`

Instalación

- Por último, si se ha llegado hasta este punto, se debería tener una red de kubernetes con un único nodo funcionando. Se puede comprobar con el siguiente comando:
  - `kubectl get nodes`
- Si todo está instalado, deberá devolver una salida similar a la siguiente:
  - `NAME STATUS ROLES AGE VERSION`
  - `server1 Ready master 25h v1.19.4`
- En caso de no tener "status Ready" en el servidor, se aconseja esperar unos minutos (puede estar iniciando cosas) o reiniciar la máquina virtual.
- Si el server está en estado "ready", podemos pasar a marcarlo como nodo "trabajador":
  - `kubectl taint nodes --all node-role.kubernetes.io/control-plane- node-role.kubernetes.io/master-`
  - `sudo systemctl start docker`

Instalación

- Los servicios de kubernetes están pensados para trabajar con IPs estáticas. Eso significa que si se han creado mientras se estaba trabajando con una red que no asegure las mismas IPs (la red wifi de la universidad) o si se está trabajand offline, kubernetes no se iniciará correctamente.
- Para poder trabajar en estos casos, se aconseja crear un adaptador de red virtual configurado con la IP que se usó al instalar kubernetes:
  - `sudo modprobe dummy`
  - `sudo ip link add eth10 type dummy`
  - `sudo ip address change dev eth10 "IP usada en la instalación"`
  - `ip address`
  - `sudo service kubelet restart`

Desinstalación/Reset de la máquina

- Para poder desinstalar kubernetes y dejar todo en un estado inicial, se pueden seguir los siguientes pasos:
  - `sudo kubeadm reset`
  - `sudo apt-get purge -y kubectl kubeadm kubernetes-cni`
  - `sudo rm -r /etc/kubernetes/`
  - `sudo rm -r /var/lib/etcd/`
  - `sudo rm -r /opt/cni`
  - `rm -r ~/.kube`
- Para resetear los repositorios añadidos a APT, se pueden realizar los siguientes pasos:
  - `sudo rm  /etc/apt/sources.list.d/kubernetes.list`
  - `sudo apt-get update`

Uso de kubernetes

- Tenemos dos comandos principales en kubernetes: kubeadm y kubectl
  - kubeadm:
    - Este comando sirve para gestionar la red virtual y los servicios principales de kubernetes. Se verán los siguientes 3 comandos:
      - kubeadm init: Inicia el servicio de kubernetes en el nodo máster. Configura el sistema de claves para poderse unir a la red virtual. Inicia el servicio kubelet del nodo máster para poder levantar pods/deployments en ese nodo
      - kubeadmin reset: Apaga el servicio kubernetes y destruye la red virtual creada con "kubeadm init". No borra los directorios de configuración creados en ejecuciones anteriores, hay que hacerlo manualmente si se quiere generar una nueva configuración.
      - kubeadm join: Se ejecuta desde un nodos "esclavo". Sirve para añadir un nodo nuevo a la red de kubernetes. Se necesita ejecutar en modo superusuario, y se debe ejecutar en un nodo que no tenga el servidor/master ejecutando.

Scripts suministrados

- OJO, se ejecutan en modo usuario (SIN SUDO, ya piden la contraseña si es necesario)

  - kub_install.sh: Script que realiza todas las operaciones de instalación de kubernetes.
  - kub_reset.sh: Desinstala kubernetes y limpia el sistema de configuraciones anteriores.

Uso de kubernetes

- kubectl:
  - Este comando sirve para configurar las aplicaciones lanzadas dentro de kubernetes. Podemos controlar pods, deployments y servicios abiertos dentro de nuestra red. Se puede encontrar la lista de opciones completa de este comando y sus añadiendo "-h" al final del comando.
  - Algunas de las opciones generales:
    - kubectl <comando> <elemento> <ID elemento> <opciones de ese elemento>
      - "Comando" : La operación que queremos realizar: get, describe, delete, log ,exec, create, etc....
      - "Elemento": Objeto sobre el que se quiere realizar la operación indicada con "comando". En general, usaremos "pods", "deployments", "nodes", "services".
      - "ID elemento": En caso de usar "Elemento" en singular (Ej. "pod" en vez de "pods"), algunas operaciones permiten añadir el identificador de ese elemento, en caso de conocerlo. Si se añade, el resto de operaciones se realizarán sólo en ese elemento.
      - "Opciones de ese elemento": Dependiendo de la operación y del tipo de elemento que hayamos seleccionado, podemos tener una lista de opciones adicionales para especificar. Para obtener una lista completa de modificadores, se puede conseguir una descripción general añadiendo "-h" al final de la orden.
        - Ej: el siguiente comando mostrará todas las opciones disponibles para crear despliegues
          - kubectl create deployment -h

Uso de kubernetes

- kubectl get <elemento>: La opción "get" mostrará una lista de los elementos seleccionados como último parámetro. Algunos ejemplos:
  - kubectl get nodes
    - Lista de los nodos que forman nuestra red
  - kubectl get pods
    - Lista de pods activos en este momento
  - kubectl get deployments
    - Lista de despliegues hechos en nuestra red
  - kubectl get services
    - Lista de puertos/servicios abiertos desde los despliegues creados anteriormente

Uso de kubernetes

- kubectl describe <elemento>: La opción "describe" mostrará el estado actual de alguno de los elementos seleccionables (pods, deployments, nodes...). Algunos ejemplos:
  - kubectl describe nodes
    - Lista características y estados de los nodos que forman nuestra red
  - kubectl get pods
    - Lista Lista características y estados de pods activos en este momento
  - kubectl get deployments
    - Lista de despliegues hechos en nuestra red
  - kubectl get services
    - Lista de puertos/servicios abiertos desde los despliegues creados anteriormente

Uso de kubernetes

- Creando un despliegue:
  - El comando "kubectl create" se puede utilizar para crear despliegues dentro de nuestra red. Admite varios parámetros, en concreto se verán "--image" y "--réplicas"
    - kubectl create deployment <nombre> --image=<docker container> --replicas=<num copias>
  - Para crear un despliegue, se necesita un contenedor docker disponible (online o descargada previamente) . A ese despliegue se le pondrá un nombre elegido por el usuario, y se ejecutará por defecto en la máquina máster donde se haya ejecutado. Adicionalmente, se pueden definir un número de copias a ejecutar dentro del sistema
  - El siguiente comando crea un despliegue con el contenedor "kubernetes-bootcamp", versión 1.0, alojado en "gcr.io/google-samples/kubernetes-bootcamp". El despliegue tendrá 3 copias ejecutándose en el nodo máster:

    - kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 --replicas=3

Uso de kubernetes

- Accediendo a un pod en ejecución:
  - Podemos ejecutar los programas instalados dentro del pod que esté en ejecución, y modificar su contenido. Para ello, disponemos el comando:
    - kubectl exec <pod ID> -- <comando>
  - El comando anterior sólo sirve para ver la salida de un comando básico, no sirve para interactuar con programas por consola. Si necesitamos interactuar, se debe añadir la opción "-ti":
    - kubectl exec -ti <pod ID> -- <comando>
  - Por ejemplo, para abrir una shell dentro del pod "pod1", podemos realizarlo de la siguiente manera:
    - kubectl exec -ti pod1 -- bash

Uso de kubernetes

- Exponiendo un servicio al exterior de la red de kubernetes:
  - Una vez creado un despliegue, necesitamos redirigir los puertos internos de los pods a puertos locales de la máquina donde se está ejecutando el nodo máster de kubernetes. Para eso, usaremos el siguiente comando:
    - kubectl expose deployment <nombre despliegue> --type="NodePort" --port <puerto interno a exponer>
  - El comando anterior enlazará un puerto interno del despliegue de kubernetes con uno de los puertos del ordenador donde está ejecutándose el servicio de kubernetes (normalmente, el nodo máster). Por ejemplo, para exponer el puerto 8080 del despliegue "kubernetes-bootcamp" podemos ejecutar:
    - kubectl expose deployment kubernetes-bootcamp --type="NodePort" --port 8080

Uso de kubernetes

- Exponiendo un servicio al exterior de la red de kubernetes:
  - Una vez expuesto, podemos consultar el puerto asignado al servicio a través del comando:
    - kubectl get services

```
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
apache2      NodePort    10.111.238.113   <none>        80:31678/TCP   6h9m
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        30h
```

  - En la sección "ports" se pueden ver los puertos enlazados para cada una de los despliegues hechos. Suponiendo que la aplicación haya levantado un servidor web, podemos usar el comando "curl" para conectar a ese puerto y ver la respuesta del servidor:
    - curl localhost:puerto

Uso de kubernetes

- Destruyendo despliegues,servicos, pods...
  - Para apagar los servicios levantados anteriormente, podemos usar el comando:
    - kubectl delete <elemento>
  - Ejemplos:
    - kubectl delete deployment kubernetes-bootcam
      - El comando anterior destruirá los pods creados en el despliegue del servicio anterior, liberando sus recursos.
    - kubectl delete service kubernetes-bootcam
      - El comando anterior borrará el servicio kubernetes-bootcamp, liberando los puertos de la máquina seleccionados.

Uso de kubernetes

- Archivos "yaml":
  - Kubernetes usa el formato "yaml" para aplicar configuraciones/modificaciones a la red. Se puede usar por ejemplo para levantar despliegues, pods, etc.. más complejos en nuestra red. Se puede realizar la operación con el siguiente comando:
    - kubectl apply -f <archivo.yaml>

Uso de kubernetes

- Archivos "yaml":
  - Campos de un archivo yaml:
    - ApiVersion: Versión a usar
    - Kind: Tipo de objeto que se quiere crear
    - Metadata: Datos generales sobre el objeto en cuestión (cada objeto tendrá unos datos propios)
    - Spec: Estado del objeto que se pretende crear. Igual que Metadata, depende del tipo de objeto y puede contener subcategorías

Uso de kubernetes

- Ejemplo yaml: Crear un despliegue con el contenedor kubernetes-bootcamp y tres réplicas (mismo caso de antes)

```
apiVersion:  apps/v1
kind:  Deployment
metadata:
  name: kubernetes-bootcamp
  namespace:  default
spec:
  replicas:  3
  selector:
    matchLabels:
      app: bootcamp
  template:
    metadata:
      labels:
        app: bootcamp
    spec:
      containers:
      - name:  kubernetes-bootcamp
        image: gcr.io/google-samples/kubernetes-bootcamp:v1
```

Uso de kubernetes

- Ejemplo yaml: Exponer un servicio para el despliegue creado anteriormente

```
apiVersion:  v1
kind: Service
metadata:
  name: bootcamp-entrypoint
  namespace: default
spec:
  type:   NodePort
  selector:
    app: bootcamp
  ports:
  - port:       8080
    targetPort: 8080
    nodePort:   31000
  externalTrafficPolicy:  Cluster
```

Uso de kubernetes

- Uso de directorios para datos "persistentes":
  - Los pods creados usan sistemas de ficheros "volátiles":
    - Al borrar un pod, se pierde todo el trabajo desarrollado
  - Para poder guardar datos que se necesiten para ejecuciones futuras, se usan "volúmenes":
    - Un volumen es un objeto de kubernetes similar a un disco duro. Contiene ficheros que no deberían borrar al borrar los pods.
      - Al desplegar por segunda vez el mismo pod, recuperará la información guardada
  - Los volúmenes pueden estar implementados usando cualquier sistema/driver de disco duro:
    - Directorios locales: hostPath
    - Directoriosen red: nfs
    - Directorios "en la nube": Gdrive, AWS S3, etc...

Uso de kubernetes

- Ejemplo:
  - Despliegue del servidor bootcamp usando un directorio local compartido con el pod en modo "hostPath":

```
apiVersion:  apps/v1
kind:  Deployment
metadata:
  name: kubernetes-bootcamp
  namespace:  default
spec:
  replicas:  1
  selector:
    matchLabels:
      app: bootcamp
  template:
    metadata:
      labels:
        app: bootcamp
    spec:
      containers:
      - name:  kubernetes-bootcamp
        image: gcr.io/google-samples/kubernetes-bootcamp:v1
        volumeMounts:
        - mountPath: /prueba
          name: directorio-prueba
      volumes:
      - name: directorio-prueba
        hostPath:
          path: /home/ubuntu/compartido
          type: Directory
```

Uso de docker

- Docker permite crear contenedores de aplicaciones con todas las librerías del sistema operativo usado. Facilita la migración de las aplicaciones, levantando sus servicios de forma aislada dentro del sistema operativo host.
- Kubernetes usa Docker como sistema contenedor de los pods que se ejecuten en su red. Descargará de forma automática el contenedor indicado en las opciones al crear el despliegue. Este contenedor se puede descargar únicamente después de haberlo creado y añadido a algún repositorio Docker.
- Opcionalmente, si se descargaron esas imágenes a un directorio local, las podrá buscar en la máquina donde se ejecute kubernetes sin necesidad de acceder a la red.

Uso de docker

- Comandos básicos de docker (ejecutar con sudo en el laboratorio):
  - docker login
    - Realiza el login con un usuario y contraseña en los repositorios de hub.docker.com
  - docker images
    - Da una lista de imágenes descargadas en la máquina local.
  - docker image rm <id>
    - Elimina una imagen docker local identificada por su ID
  - docker pull <id>
    - Descarga una imagen de alguno de los servidores configurados dentro de docker. Por defecto, puede descargar de los servidores de ejemplo usados por google. En caso de haber hecho login anteriormente, podremos seleccionar imágenes creadas por nosotros.
  - docker push <id>
    - Si hemos creado una imagen nueva ó modificación, podremos actualizarla en el repositorio donde hayamos hecho el login anteriormente.

Uso de docker

- Creando una nueva imagen docker
  - Para crear una nueva imagen, debemos crear un directorio con los siguientes recursos:
    - Dockerfile
      - Archivo con las reglas de construcción de la imagen docker
    - Archivos extras a añadir a la configuración
      - Si queremos copiar nuevos archivos a la imagen docker, debemos colocarlos junto al archivo Dockerfile
  - Para construir la nueva imagen:
    - docker build <directorio_con_dockerfile>
      - El último parámetro debe apuntar a un directorio con el fichero dockerfile. Una vez ejecutado el comando anterior, nos devolverá el identificador de la nueva imagen guardada en nuestro ordenador. Para consultarla, podemos ejecutar "docker images".

Uso de docker

- Creando un archivo Dockerfile:
  - Los archivos Dockerfile consisten en una serie de reglas de construcción para nuestro contenedor. Tendrán al menos las siguientes secciones (pueden alternarse, pero se aconseja seguir este esquema por simplicidad):
    - Sección "selección de sistema base":
      - Esta sección permite partir de una distribución del sistema operativo básica. Contendrá las librerías indispensables para poder ejecutar una versión recortada de la distribución del sistema operativo del que se parte.
    - Sección "instalación de programas genéricos":
      - Esta sección permite añadir los programas que se usarán dentro del contenedor. Estos programas normalmente se encuentran en repositorios de la distribución seleccionada anteriormente, por lo que se consideran "genéricos".
    - Sección "configuración de ejecución":
      - Esta sección asadirá configuraciones finales y programas externos que no se encuentren en la distribución de la que se parte. Se usará para copiar archivos propios, programas generados por los usuarios, librerías que no se encuentren en los repositorios, apertura de puertos de red, etc....

Uso de docker

- Selección de sistema base:
  - Comando "FROM distribución:versión"
    - El comando FROM permite seleccionar una distribución/contenedor base desde la que partir, usando su nombre y la versión de la misma.
    - Esta distribución debe estar previamente almacenada como un contenedor dentro de alguno de los repositorios configurados de docker.
    - Ej:
      - FROM ubuntu:20.04
    - El comando anterior buscará un contenedor con una versión básica de ubuntu 20.04 en los repositorios, lo descargará y usará como base para la instalación de nuevas aplicaciones

Uso de docker

- Sección "instalación de programas genéricos":
  - Una vez seleccionada la imagen base, pasamos a configurar el resto de programas que habrá en el contenedor. En esta sección disponemos del comando "RUN", que permite ejecutar configuraciones dentro de la imagen para poder instalar programas "genéricos". En concreto, podemos usar los gestores de paquetes "apt" si nos encontramos en una distribución "ubuntu".
  - Se pueden realizar más configuraciones aparte de instalar programas, para ello solo hay que ejecutar el comando que se necesite, que deberá ser un local a la distribución/contenedor que estamos configurando.
    - Ej:
      - RUN apt-get update
      - RUN apt-get install apache2

Uso de docker

- Sección "configuración de ejecución":
  - Normalmente, después de haber instalado las aplicaciones de la distribución, queda configurar estas aplicaciones. Algunas de las operaciones genéricas son:
    - Apertura de puertos de red: comando EXPOSE <puerto>
      - Ej:
        - EXPOSE 80
    - Comando/script a ejecutar cuando se lance el contenedor
      - Comando CMD <ejecutable> <opciones del ejecutable>
        - Ej: CMD apachectl –DFOREGROUND
    - Copia de archivos externos a directorios locales del contenedor:
      - COPY <archivos origen> <directorio destino>
        - Ej: COPY index.html /var/www/html/

Uso de docker

- Ejemplo de archivo Dockerfile:

```
FROM ubuntu:20.04
RUN apt-get update
RUN apt-get install -y software-properties-common
RUN apt-get update
RUN apt-get install -y apache2
EXPOSE 80
CMD apachectl -DFOREGROUND
COPY index.html /var/www/html/
```

Uso de docker

- Construyendo la imagen:
  - docker build <directorio con dockerfile>
  - Apuntar el identificador de imagen docker generado al final de la construcción
- Marcar/nombrar la imagen para poderla subir a un repositorio docker
  - docker images
    - Deberá aparecer una imagen con nuestro identificador
  - docker tag <id imagen> <tag>
    - El tag deberá contener el nombre del repositorio, el nombre de la imagen, y la versión de la imagen.
    - Ej: para una imagen con identificador bc41f499bca3
      - docker tag bc41f499bca3 mnovalbos/apache_ubuntu:1.0
- Actualizar la imagen en el repositorio online
  - docker push docker.io/mnovalbos/apache:1.0

Preguntas, dudas…