

Orquestación de contenedores Docker sobre múltiples máquinas físicas con Kubernetes.

APRENDIENDO SOBRE

KUBERNETES

CDA

2018

Autores

| DNI | Nombre(s) | Apellidos |
|-----------|---------------|-------------------|
| 77482941N | Diego Enrique | Fontán Lorenzo |
| 44844497V | Jordan | Oreiro Vieites |
| 54151674Z | Diego Stéphan | Jeandon Rodríguez |

Tabla de contenidos

- Contextualización
 - Objetivo del proyecto
 - Antes de nada... ¿Qué es Kubernetes?
 - Kubernetes 101
 - Kubernetes vs Docker Swarm
- Herramientas y entorno
 - Productos empleados
 - Instalación del entorno
 - Configuración del entorno
 - Configuración de las comunicaciones
- Desplegando el entorno
 - Creación de las aplicaciones
- Bibliografía

Contextualización

Objetivo del proyecto

Conseguir desplegar contenedores Docker en varias máquinas físicas y orquestarlas desde una máquina maestra mediante Kubernetes.

Antes de nada... ¿Qué es Kubernetes?

Kubernetes (o k8s) es un sistema de código abierto creado por Google. Su principal objetivo es la gestión y orquestación de contenedores Docker.

A grandes rasgos, Kubernetes suple las carencias que tiene Docker tales como el despliegue, escalado y monitorización de los contenedores. De esta forma, podemos decir que **k8s** es, básicamente, un **Docker con superpoderes**.

Kubernetes 101

Kubernetes tiene una estructura jerárquica de máquinas dentro de un `clúster`. Explicada de manera simple, se basa en la coordinación entre una máquina `master` y varias máquinas esclavas llamadas `nodes`.

Cada `node` está formado a su vez por un conjunto de aplicaciones llamadas `pods`, sus respectivos servicios denominados `services`, sistemas de almacenamiento conocidos como `volumes` y otros nodos virtuales (`namespaces`).

Dentro de los `pods` es donde corremos nuestros contenedores **Docker**.

Kubernetes vs Docker Swarm

Si es cierto que **Kubernetes** mejora **Docker**, este último cuenta con una herramienta propia llamada **Docker Swarm** con la que orquestar contenedores.

A pesar de que **Docker Swarm** implementa muchas más opciones a la administración de contenedores, deja mucho que desear respecto al rendimiento y la escalabilidad. Por ello **Kubernetes** se ha posicionado como la opción más popular, consiguiendo más del *80% del interés en artículos de noticias, gran cantidad de repositorios en Github y grandes resultados en búsquedas web*.

Aún así, **Docker Swarm** es mucho más sencillo de implementar, a lo que **Kubernetes** ha respondido sacando herramientas como `minikube` o `kubeadm`.

Herramientas y entorno

Productos empleados

Para realizar este proyecto utilizaremos las siguientes herramientas:

- Dos o más máquinas físicas conectadas a una misma red
- Docker Community Edition
- Kubernetes: Minikube, Kubeadm y Kubectl
- Tres mentes curiosas y mucha paciencia

Instalación del entorno

Antes de nada debemos instalar las herramientas necesarias en nuestros dispositivos o máquinas virtuales.

Debido a la cantidad de requisitos necesarios (tales como `VirtualBox`, `Minikube`, `Kubectl`, `Kubeadm` ...), nos hemos dado el lujo de crear un script que permita la instalación automática de éstos en sistemas operativos *Ubuntu/Debian-based*.

El script puede ser encontrado en el un repositorio Github perteneciente a un miembro del grupo y puede ser clonado mediante `git` con el siguiente comando:

```
git clone https://github.com/Student-Puma/Homelab --branch k8s
```

El script de instalación es el siguiente:

```
#!/bin/bash
# file: env-setup.sh
# Only run as root necessary commands
[ $EUID -eq 0 ] && sudo='' || sudo='sudo'
[[ $SUDO_USER = '' ]] && user=$USER || user=$SUDO_USER
[[ $(lsb_release -is) = 'elementary' ]] && dist='xenial' || dist=$(lsb_release -cs)
# Install the dependencies
dependencies=( curl virtualbox-qt apt-transport-https software-properties-common
  ca-certificates kubectl kubeadm )
for dep in ${dependencies[@]};do
  # Kubectl needs prerequisites
  if [ $dep = kubectl ]; then
    if [ ! -f /etc/apt/sources.list.d/kubernetes.list ]; then
      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 -y
    fi
  fi
# Install missing dependencies
if ! dpkg -l $dep >&/dev/null; then
```

```

    $sudo apt-get install -y $dep
fi
done
# Docker Setup
if ! which docker &>/dev/null; then
    curl -fsSL https://download.docker.com/linux/ubuntu/gpg | $sudo apt-key add -
    $sudo add-apt-repository "deb [arch=amd64]
        https://download.docker.com/linux/ubuntu $dist stable"
    $sudo apt-get update -y
    $sudo apt-get install -y docker-ce=$(apt-cache madison docker-ce |
        cut -d" " -f4 | grep 18.06.0)
fi
# Get the latest version name
MINIKUBE_VERSION() {
    curl --silent
        "https://api.github.com/repos/kubernetes/minikube/releases/latest" |
        grep '"tag_name":' | sed -E 's/.*"([^"]+)"*/\1/'
}
# Download and install Minikube
if ! which minikube &>/dev/null; then
    curl -Lo minikube
        https://storage.googleapis.com/minikube/releases/
        $(MINIKUBE_VERSION)/minikube-linux-amd64
    chmod +x minikube
    $sudo mv minikube /usr/local/bin/
fi

```

Configurando el entorno

Lo primero será iniciar el `master node` (aquel que manejará el resto de nodos) usando el comando.

Empezaremos levantando `minikube`:

```
minikube start --vm-driver virtualbox
```

Descargaremos de antemano las imágenes Docker necesarias ejecutando:

```
kubeadm config images pull
```

Esto nos dará algo de velocidad a la hora de iniciar el servicio. Para ello escribiremos el comando:

```

kubeadm init --pod-network-cidr <Rango de IPs>
# en nuestro caso: kubeadm init --pod-network-cidr 192.168.0.0/16
# en el caso de estar conectados vía WiFi en vez de Ethernet,
# debemos añadir '--apiserver-advertise-address=<wlan ip>'

```

Una vez completado el comando, nos dará una salida similar a esta:

```
Your Kubernetes master has initialized successfully!
To start using your cluster, you need to run the following as a regular user:
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
[...]
You can now join any number of machines by running the following on each
node as root:
  kubeadm join 192.168.0.18:6443 --token ybm1yx.umimbi3emsbhb59t
    --discovery-token-ca-cert-hash
    sha256:c5ef3003e785d68684a7bf9ec5ba9ee4411421be988de65a17b1a2e22b7af3ed
```

El siguiente paso será copiar el archivo de configuración como un usuario sin privilegios, tal como nos indica el mensaje:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Para agregar los nodos esclavos a nuestro clúster, ejecutaremos en cada uno de ellos el comando mostrado como resultado de `kubeadm init`. En nuestro caso será:

```
kubeadm join 192.168.0.18:6443 --token ybm1yx.umimbi3emsbhb59t --discovery-token-
ca-cert-hash
sha256:c5ef3003e785d68684a7bf9ec5ba9ee4411421be988de65a17b1a2e22b7af3ed
```

Después de unos pocos segundos podemos observar que ya tenemos los `nodes` en nuestro `cluster` usando el siguiente comando en nuestro `master node`:

```
kubectl get nodes
```

Configuración de las comunicaciones

Los `nodes` no se pueden comunicar entre sí sin que exista un `network pod`, así que ejecutaremos:

```
kubectl apply --filename
"https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version |
base64 | tr -d '\n')
```

`kubectl apply --filename <archivo de configuración.yaml>` nos permite crear o modificar `pods` dentro del `cluster`

Desplegando el entorno

Creación de las aplicaciones

Ahora procederemos a desplegar una aplicación en nuestro `master node` , la cual se ejecutará en uno de los `nodes` del `cluster` .

En este caso desplegaremos el **Kubernetes Dashboard** sacado del repositorio oficial de Kubernetes que nos permitirá acceder a una interfaz visual del `cluster` .

Para ello, sólomente deberemos escribir:

```
kubectl apply --filename https://bit.ly/2Lb76yP
```

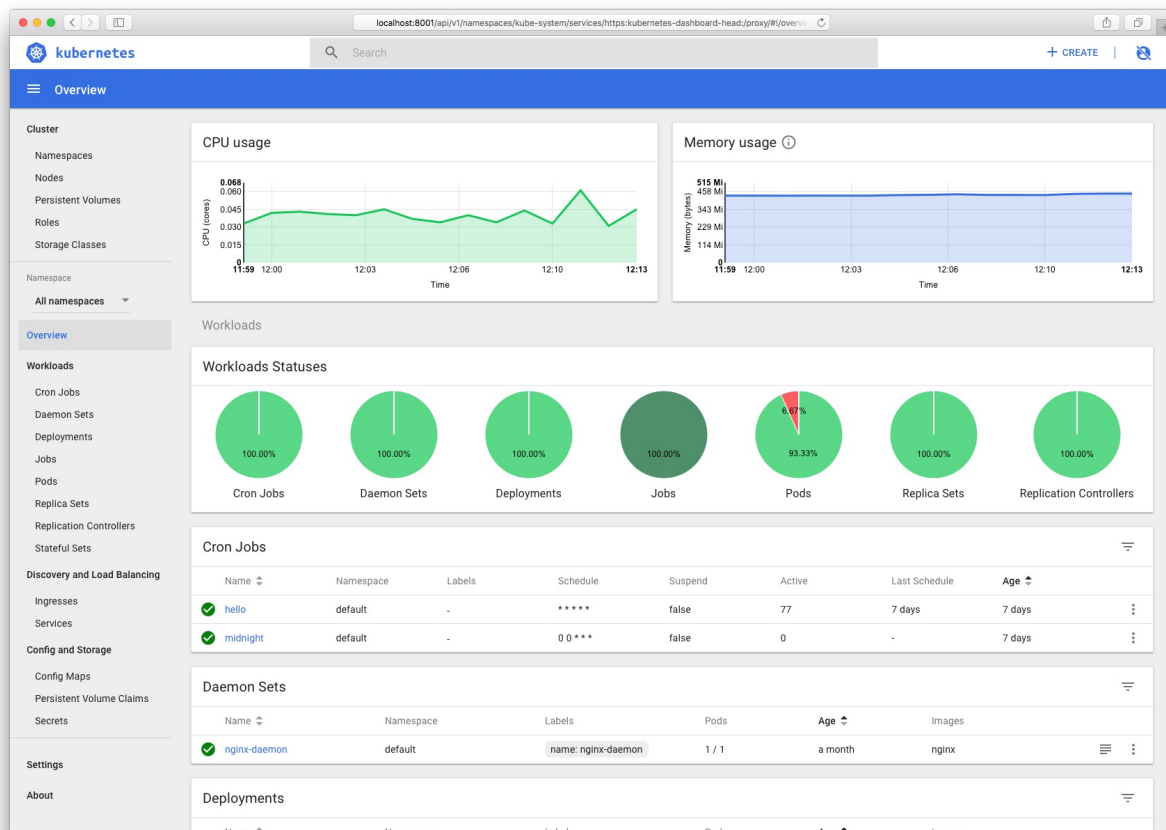
Cambiamos la configuración del Dashboard de `ClusterIP` a `NodePort` para poder acceder a él:

```
kubectl -n kube-system edit service kubernetes-dashboard
```

Sólamente nos falta obtener el puerto que está utilizando para poder acceder desde el navegador con la dirección `https://localhost:<port>` :














```
kubectl -n kube-system get service kubernetes-dashboard -o  
template --template="{{ (index .spec.ports 0).nodePort }}" |  
xargs echo
```

¡Ahora ya podemos acceder a nuestro Dashboard!




En la presentación se propondrá otro tipo de aplicación a desplegar en nuestro `cluster`. Este tipo de aplicación será una formada por un **Frontend** y siete **Backends**

Bibliografía

-  <https://github.com/Student-Puma/Homelab>
-  <https://kubernetes.io/docs/home/>
-  <https://enmilocalfunciona.io/introduccion-a-kubernetes-i/>
-  <https://www.nubersia.com/es/blog/kubernetes-vs-docker-swarm/>
-  <https://kubernetes.io/docs/setup/independent/install-kubeadm/>
-  <https://www.weave.works/blog/weave-net-kubernetes-integration/>
-  <https://gist.github.com/alexellis/fdbc90de7691a1b9edb545c17da2d975>
-  <https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/>
-  <https://blog.hypriot.com/post/setup-kubernetes-raspberry-pi-cluster/>
-  <https://downey.io/blog/how-to-build-raspberry-pi-kubernetes-cluster/>
-  <https://blog.sicara.com/build-own-cloud-kubernetes-raspberry-pi-9e5a98741b49>
-  <http://www.javiergarzas.com/2016/02/kubernetes-for-dummies-explicado-en-10-minutos.html>
-  <https://kubecloud.io/setting-up-a-kubernetes-1-11-raspberry-pi-cluster-using-kubeadm->

952bbda329c8

-  <https://kubecloud.io/setup-a-kubernetes-1-9-0-raspberry-pi-cluster-on-raspbian-using-kubeadm-f8b3b85bc2d1>