# 

## **Facultad de Ingeniería en Sistemas**

## **Sistemas Operativos II**

## **Sección B**

# **Orquestador de microservicios en alta disponibilidad utilizando Kubernetes**

**Gabriel Alejandro Díaz Méndez 3190-02-4284**

**Rony Misael Paz Juárez 3190-17-5011**

**Kevin Giovanni Monterroso Concuá 3190-17-838**

**Erick Antonio Diaz Guzmán 3190-12-16419**

**José Manuel Urbina Ávila 3190-17-10153**

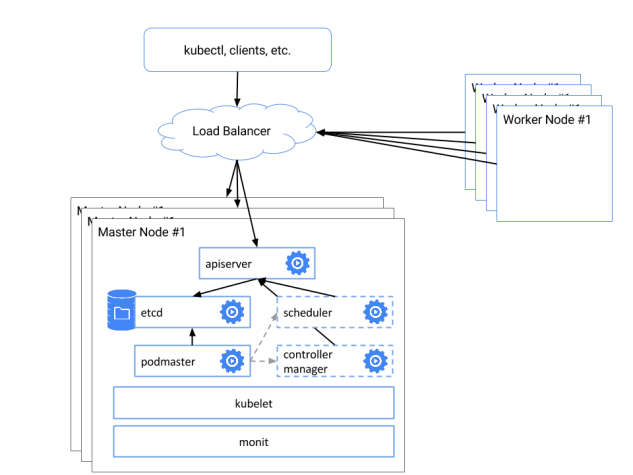
**Víctor Manuel Verlaine Gutiérrez 3190-17-15751**

**David Esteve Monzón Reyes 3190-17-2298**

# **ARQUITECTURA**

* Inventario de servidores

|  |  |  |
| --- | --- | --- |
| NOMBRE DEL SERVIDOR | IP | ROL |
| Master1 | 192.168.1.43 | Master node |
| Master2 | 192.168.1.42 | Master node |
| Balancer | 192.168.1.41 | Load balancer |
| Slave1 | 192.168.1.40 | Worker node |
| Slave2 | 192.168.1.39 | Worker node |
| Registro | 192.168.1.33 | Repositorio privado |



# **REQUISITOS**

* Configuración de máquina virtual para nuestro Balancer
  + Le asignamos 10G de disco duro
  + Le asignamos 2G de RAM
  + Le asignamos 3 Procesador
  + Ubuntu 18.4

Captura de pantalla de un celular

Descripción generada automáticamente

* Configuración de máquinas virtuales para los Masters
  + Le asignamos 10G de disco duro
  + Le asignamos 3G de RAM
  + Le asignamos 3 Procesadores
  + Ubuntu 18.4

Captura de pantalla de un celular

Descripción generada automáticamente

* Configuración de máquina virtual para los esclavos
  + Le asignamos 10G de disco duro
  + Le asignamos 2G de RAM
  + Le asignamos 2 Procesador
  + Ubuntu 18.4

Captura de pantalla de un celular

Descripción generada automáticamente

# **CONFIGURACION PRINCIPAL**

* Actualizar el sistema operativo

sudo apt update && sudo apt -y upgrade

* Entrar en como usuarios root

sudo su

* Instalar los paquetes para permitir que apt use el repositorio HTTPS

apt-get update && apt-get install -y apt-transport-https ca-certificates curl software-properties-common gnupg2

* Agregar la llave GPG oficial de Docker

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

* Agregar el repositorio de docker

add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

* Instalar Docker CE

apt-get update && apt-get install -y containerd.io=1.2.13-1 \

docker-ce=5:19.03.8~3-0~ubuntu-$(lsb\_release -cs) \

docker-ce-cli=5:19.03.8~3-0~ubuntu-$(lsb\_release -cs)

* Configurar Demonio

cat > /etc/docker/daemon.json <<EOF

{

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2"

}

EOF

* Crear la carpeta de Docker Service

mkdir -p /etc/systemd/system/docker.service.d

* Recargar el demonio

systemctl daemon-reload

* Reiniciar Docker

systemctl restart Docker

* Verificamos el estatus de nuestro Docker

systemctl status docker

# **INSTALACION KUBERNETES**

* Agregar la llave GPG oficial de Kubernetes

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

* Agregar el repositorio de Kubernetes

apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

* Instalar Kubeadm

apt install -y kubeadm

* Verificamos el estatus de Kubernetes

systemctl status kubelet

# **CONFIGURACION BALANCER**

* Entrar como usuario root

sudo su

* Instalar Nginx

apt install -y nginx

* Crear una carpeta

mkdir -p /etc/nginx/tcpconf.d

* Abrir el archivo de NGINX

nano /etc/nginx/nginx.conf

* Agregar la carpeta creada al final del archivo de NGINX

include /etc/nginx/tcpconf.d/\*;

* Agregar configuración de Kuberntes

cat <<EOF | sudo tee /etc/nginx/tcpconf.d/kubernetes.conf

stream {

upstream kubernetes {

server 192.168.1.43:6443;

server 192.168.1.42:6443;

}

server {

listen 6443;

listen 443;

proxy\_pass kubernetes;

}

}

EOF

* Recargamos los cambios

nginx -s reload

* Miramos la ip del balancer

Ifconfig

* Miramos el estatus del balancer

systemctl status nginx

# **LEVANTAR PRIMER MAESTRO**

* Apagar la SWAP

swapoff -a

* Levantamos nuestro master

kubeadm init --control-plane-endpoint "192.168.1.41:6443" --upload-certs

* Generación de tokens

kubeadm join 192.168.1.41:6443 --token br3kl2.fj4zzeeh4q5o45hz \

--discovery-token-ca-cert-hash sha256:dfcf1aef179b857d98c401e324f86a76c27de57471f3c832ab5965f0a64de633 \

--control-plane --certificate-key b524df7a3cf54bffb7ff803af6412726a0fb4aaa3872e4b914b4dc561ac59e14

kubeadm join 192.168.1.41:6443 --token br3kl2.fj4zzeeh4q5o45hz \

--discovery-token-ca-cert-hash sha256:dfcf1aef179b857d98c401e324f86a76c27de57471f3c832ab5965f0a64de633

* Finalizar configuración del master

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

* Aplicar el complemento CNI de Kubernetes

kubectl apply -f ”[https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')](https://cloud.weave.works/k8s/net?k8s-version=$(kubectl%20version%20|%20base64%20|%20tr%20-d%20'\n'))”

* Vemos el estado de los pods

kubectl get pod -n kube-system -w

kubectl get pods --all-namespaces

* Volvemos a ver los estados de los pods

kubectl get pod -n kube-system

# **LEVANTAR SEGUNDO MAESTRO**

* Apagar swap

swapoff -a

* Copiar Token

kubeadm join 192.168.1.36:6443 --token f244cw.rizof8lfx4hu1tx9 \

--discovery-token-ca-cert-hash sha256:86264cd7d615dc17742c5fdf305eab76a9ae09143bd553e13c580b80b2c6cea7 \

--control-plane --certificate-key e6edb976f294937cd1b5d0cd641c21a7a18e900e284ac81878f34d9ac7dce608

* Finalizar configuración del master 2

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

# **AGREGAR LOS ESCLAVOS**

* Entrar al esclavo
* Apagar swap

swapoff -a

* Copiar token

kubeadm join 192.168.1.36:6443 --token g6fmvg.k6wno5ash85s7ayj --discovery-token-ca-cert-hash sha256:aaeabb43befb7a8ef75d0c0f3eb2e8abb9404122ac94159d8df53bfc17964048

# **RESET DE CLUSTER**

* Resetear cluster

Kubeadm reset

* Borrar archivos de configuración

rm -rf $HOME/.kube/config

sudo sysctl net.bridge.bridge-nf-call-iptables=1

* Apagar swap

swapoff -a

* Repetir los pasos de levantar para cada miembro del cluster

## **CONFIGURACION SERVIDOR DE IMÁGENES**

* modificar archivo openssl

sudo gedit /etc/ssl/openssl.cnf

* agregar la siguiente línea en el archivo

subjectAltName=IP:192.168.1.33 #ip del servidor de imagenes

* Descargar imagen de docker registry

sudo docker pull registry:2

* Crear carpeta local para almacenar los certificados

mkdir -p /docker\_data/certs/

* Crear certificados

openssl req \

-newkey rsa:4096 -nodes -sha256 -keyout /docker\_data/certs/domain.key \

-x509 -days 365 -out /docker\_data/certs/domain.crt

* En el log de la consola agregar la IP del servidor de imagenes

Common Name (eg, your name or your server's hostname) []:ip-del-servidor

* Crear un directorio para almacenar las imagenes de docker

mkdir -p /docker\_data/images

* Ejecutar un contenedor de docker registry en servidor local

docker run -d -p 5000:5000 \

-v /docker\_data/images:/var/lib/registry \

-v /docker\_data/certs:/certs \

-e REGISTRY\_HTTP\_TLS\_CERTIFICATE=/certs/domain.crt \

-e REGISTRY\_HTTP\_TLS\_KEY=/certs/domain.key \

--restart on-failure \

--name myregistry \

registry:2

* Crear Docker Host

export DOCKER\_HOST=tcp://192.168.1.33:5000

## **CONFIGURACION DEL CLIENTE**

* Copiar el archivo /docker\_data/certs/domain.crt en la carpeta root de los clientes
* Crear carpeta para aprobar el certificado dentro de etc/docker/certs.d

mkdir -p /etc/docker/certs.d/192.168.1.33:5000/

* La anterior es el nombre del servidor de images seguido del puerto
* Copiar el archivo /root/domain.crt a la carpeta /etc/docker/certs.d/192.168.1.33:5000/

cp -rf /root/domain.crt /etc/docker/certs.d/192.168.1.33:5000/

* Subir imagen al servidor de imagenes

docker push 192.168.1.33:5000/imagen:version

* Descargar imagen del servidor

## **CONFIGURACION DEL FRONTEND**

* Configuración del archivo dockercompose.yaml

nginx-proxy:

image: jwilder/nginx-proxy

restart: always

ports:

"80:80"

"443:443"

volumes:

/var/run/docker.sock:/tmp/docker.sock:ro

/etc/nginx/vhost.d

/usr/share/nginx/html

./certs:/etc/nginx/certs:ro

./nginx-custom.conf:/etc/nginx/conf.d/custom.conf:ro

labels:

com.github.jrcs.letsencrypt\_nginx\_proxy\_companion.nginx\_proxy

letsencrypt:

image: jrcs/letsencrypt-nginx-proxy-companion

restart: always

volumes:

./certs:/etc/nginx/certs:rw

/var/run/docker.sock:/var/run/docker.sock:ro

volumes\_from:

nginx-proxy:rw

ui:

image: jc21/registry-ui

expose:

"80"

environment:

REGISTRY\_HOST: 192.168.1.33:5000

REGISTRY\_SSL: "true"

REGISTRY\_DOMAIN: 192.168.1.33

# Tienes que crear este usuario para que ui se pueda conectar al registry con auth

# You gotta create this user for the ui to connect to the registry using auth

REGISTRY\_USER: docker

REGISTRY\_PASS: pass

REGISTRY\_STORAGE\_DELETE\_ENABLED: "true"

restart: always

links:

registry:192.168.1.33

proxy:

image: nginx

expose:

"80"

environment:

VIRTUAL\_HOST: 192.168.1.33

VIRTUAL\_PORT: 80

LETSENCRYPT\_HOST: 192.168.1.33

LETSENCRYPT\_EMAIL: dmonzonr@miumg.edu.gt

volumes:

./proxy.conf:/etc/nginx/conf.d/proxy.conf:ro

depends\_on:

ui

registry

nginx-proxy

links:

ui

registry

restart: always

configuracion archivo nginx-custom.conf

# disable any limits to avoid HTTP 413 for large image uploads

client\_max\_body\_size 0;

* Configurar archivo proxy.conf

# Logging

log\_format proxy 'http [$time\_local] $status $request\_method $scheme $host "$request\_uri" [Client $remote\_addr] [Length $body\_bytes\_sent] [Time $request\_time] "$http\_user\_agent" "$http\_referer"';

upstream registry {

server registry:5000;

}

upstream ui {

server ui:80;

}

## Set a variable to help us decide if we need to add the

## 'Docker-Distribution-Api-Version' header.

## The registry always sets this header.

## In the case of nginx performing auth, the header is unset

## since nginx is auth-ing before proxying.

map $upstream\_http\_docker\_distribution\_api\_version $docker\_distribution\_api\_version {

'' 'registry/2.0';

}

server {

server\_name local;

listen 80 default;

access\_log /dev/stdout proxy;

charset utf-8;

proxy\_set\_header X-Forwarded-Scheme $scheme;

server\_tokens off;

# disable any limits to avoid HTTP 413 for large image uploads

client\_max\_body\_size 0;

# required to avoid HTTP 411: see Issue #1486 (https://github.com/moby/moby/issues/1486)

chunked\_transfer\_encoding on;

location /health {

allow all;

access\_log off;

return 200 'OK';

}

# Registry passthrough

location ~ /v2.\* {

# Do not allow connections from docker 1.5 and earlier

# docker pre-1.6.0 did not properly set the user agent on ping, catch "Go \*" user agents

if ($http\_user\_agent ~ "^(docker\/1\.(3|4|5(?!\.[0-9]-dev))|Go ).\*$" ) {

return 404;

}

## If $docker\_distribution\_api\_version is empty, the header is not added.

## See the map directive above where this variable is defined.

add\_header 'Docker-Distribution-Api-Version' $docker\_distribution\_api\_version always;

# kill cache

add\_header Last-Modified $date\_gmt;

add\_header Cache-Control 'no-store, no-cache, must-revalidate, proxy-revalidate, max-age=0';

if\_modified\_since off;

expires off;

etag off;

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_read\_timeout 900;

proxy\_pass https://registry;

}

# Frontend

location / {

# kill cache

add\_header Last-Modified $date\_gmt;

add\_header Cache-Control 'no-store, no-cache, must-revalidate, proxy-revalidate, max-age=0';

if\_modified\_since off;

expires off;

etag off;

proxy\_set\_header Host $host;

proxy\_pass http://ui;

}

}

* Ejecución del docker compose

docker-compose up -d

## **DESPLEGAR PODS Y APP**

* Crear nuestro deployment y a la vez bajamos la imagen de docker hub

kubectl create deployment hello-http --image=192.168.1.33:5000/my-hello-world:v1

* Ver el estado de los pods

kubectl get pods

* Ver los detalles de nuestro despliegue

kubectl describe pod hello-http

* Crear un archivo yml

nano app.yml

* Agregamos nuestra configuracion

---

apiVersion: v1

kind: Pod

metadata:

  name: mi-app

  labels:

    app: web

spec:

  containers:

    - name: front-end

      image: nginx

      ports:

        - containerPort: 80

    - name: back-end

      image: redis

* Desplegamos el conjunto de contenedores descritos en el fichero app.yml

kubectl apply -f app.yml

* exponemos nuestro servicio

kubectl expose deployment hello-http --type=LoadBalancer --port=80

* Vemos nuestros servicios Activos

kubectl get services

* Para cambiar o agregar el número de replicas que queremos

kubectl scale deployment --replicas=3 hello-http

* Eliminamos el despliegue de la imagen httpd:latest

kubectl delete deployment hello-http

* Eliminamos el despliegue hecho con el fichero app.yml

kubectl delete -f app.yml

* Eliminamos el servicio expuesto hello.http

kubectl delete service hello-http

# **DESPLEGAR DASHBOARD**

* Deploying de Dashboard para la interfaz

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/deploy/recommended.yaml>

* Creamos el archivo dashboar-admin.yml, para el Rol

nano dashboar-admin.yml

* Agregamos la configuración al archivo dashboar-admin.yml

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kube-system

* Aplicamos el archivo de configuración a kubernetes, para el rol de admin

kubectl apply -f dashboar-admin.yml

* Creamos un archivo para el ClusterRol

nano admin-role-binding.yml

* Agregamos la configuración al archivo admin-role-binding.yml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kube-system

* Aplicamos el archivo de configuración a kubernetes, para el ClusterRol

kubectl apply -f admin-role-binding.yml

* Generar token, para poder iniciar sesión en la página del Dashboard a través de token

kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

* Para poder acceder al panel, utilizamos lo siguiente

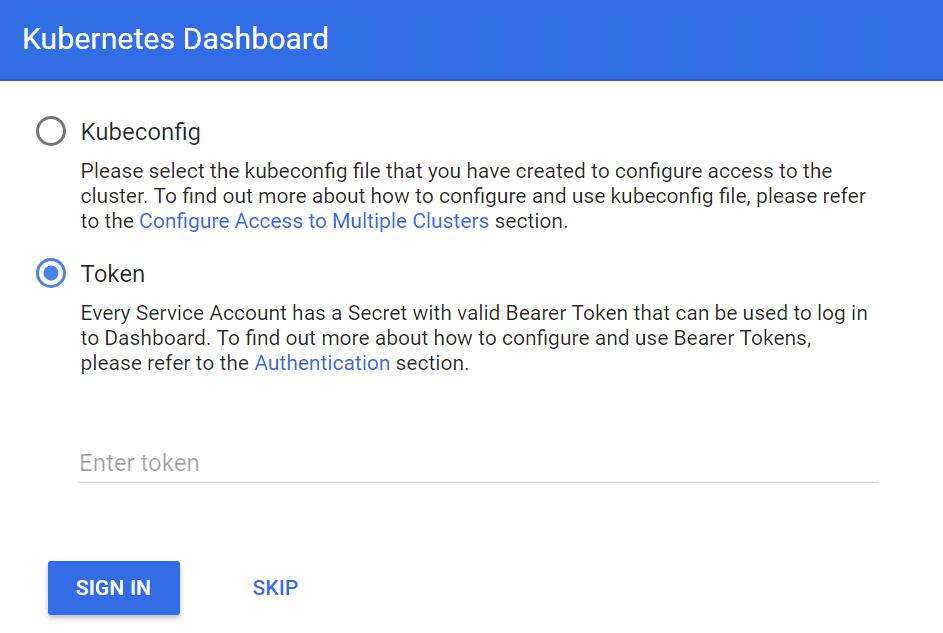
kubectl proxy

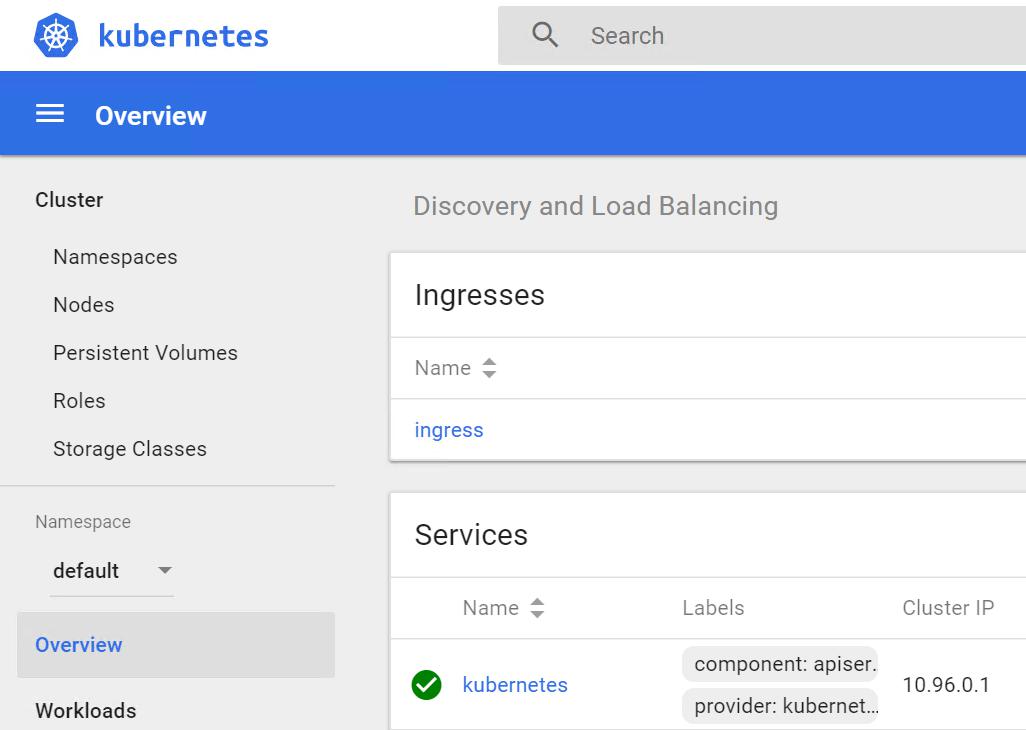
* Comprobar si todo está correcto, abrimos nuestro navegador y podemos lo siguiente

localhost:8001

* Ahora entramos a la página web del Dashboard

<http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>





Link de video tutorial:

https://youtu.be/gMPFPpMgBBc