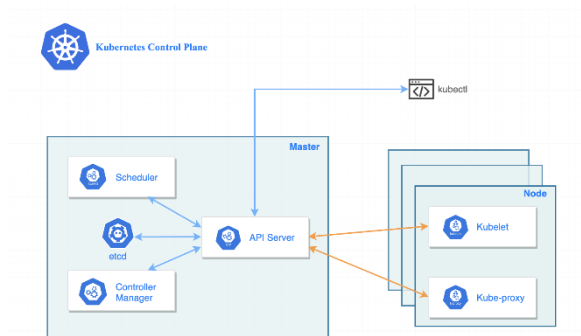


Instalación cluster kubernetes con Kubeadm



En este laboratorio utilizaremos tres mv, creadas en vagrant a través de este **Vagrantfile**, el cual crea tres servidores con CentOS 7.6 y a través de ansible, instala en todos los nodos **docker**, desactiva selinux y firewalld:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
##Las mv de vagrant el usuario es:
#vagrant
#vagrant
#root
#vagrant
Vagrant.configure("2") do |config|

  config.vm.box = "bento/centos-7.6"
  config.vm.provision "ansible_local" do |ansible|
    ansible.playbook = "playbook_centos_install_docker.yaml"
    config.vm.provider "virtualbox" do |vb|
      vb.gui = false
      vb.memory = "2048"
      vb.cpus = "2"
    end
  end

  # master
  config.vm.define "master" do |app|
    app.vm.hostname = "master"
    app.vm.network "private_network", ip: "192.168.2.10"
    #app.vm.provision "shell", path: "provision/provision-for-balancer.sh"
  end

  # worker1
  config.vm.define "worker1" do |app|
    app.vm.hostname = "worker1"
    app.vm.network "private_network", ip: "192.168.2.11"
    #app.vm.provision "shell", path: "provision/provision-for-nginx.sh"
  end

  # worker2
  config.vm.define "worker2" do |app|
    app.vm.hostname = "worker2"
    app.vm.network "private_network", ip: "192.168.2.12"
    #app.vm.provision "shell", path: "provision/provision-for-nginx.sh"
  end
end
end
end
```

Preparar los servidores CentOS 7.x

Características

- Tres máquinas virtuales con al menos 2,5G de RAM y 20 HDD de espacio en disco .
- Debemos configurarla con al menos 2 procesadores, de lo contrario no funciona la instalación con kubeadm
 - Tendremos un master y dos workers

Preparación en todos los servidores

Kubeadm automatiza la instalación y la configuración de componentes de Kubernetes como el servidor de API, Controller Manager y Kube DNS.

Kubeadm es una herramienta que nos ayuda a iniciar clusters de Kubernetes siguiendo las mejores prácticas en la infraestructura existente. Su principal ventaja es la capacidad de lanzar grupos de Kubernetes mínimos viables en cualquier lugar, es decir, realiza las acciones necesarias para que un cluster sea mínimamente viable y funcione de manera fácil para el usuario. Kubeadm automatiza bastantes pasos difíciles en la implementación de un clúster Kubernetes, incluida la emisión y coordinación de los certificados de seguridad de cada nodo, así como los permisos necesarios para el control de acceso basado en roles (RBAC).

Kubeadm no puede proveer la infraestructura y tampoco incluye instalación de addons y la configuración de red. Kubeadm se pretende que sea un componente compositivo de herramientas de nivel superior, es decir, se espera que se construyan herramientas de nivel superior y más personalizadas sobre kubeadm, e idealmente usen kubeadm como la base de todas las implementaciones.

Es una buena opción para las instalaciones en baremetal de Kubernetes.

Según la documentación oficial, kubeadm se puede utilizar en los siguientes escenarios:

- Para probar Kubernetes por primera vez.
- Implementar un clúster mínimo para probar una aplicación
- Para ser explotado como un bloque de construcción en otros sistemas complejos

Kops

Creación automatizada de infraestructura y despliegue de clusters. La mejor opción para desplegar un cluster k8s en una nube pública. Recomendado para ambientes de producción.

Kubespray

Playbooks de Ansible, es mejor idea usar kubespray si vas a desplegar Kubernetes pero no quieres depender de una plataforma de nube o si vas a desplegar Kubernetes en una nube privada o en una nube no soportada por Kops. Recomendado para ambientes de producción.

- Debemos trabajar como **root**, las credenciales de los servidores:

User: **root**

Password: **vagrant**

- En primer lugar debemos deshabilitar el swap.

En todos los nodos:

- **Tener instalado docker.**
- **Asegurarse de tener desactivado selinux y firewalld.**

Change the Docker cgroup to systemd by editing the Docker service with the following command:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/troubleshoot/docker_cgroup.html

Añadimos la línea que esta en rojo al ExecStart

systemctl edit --full docker.service

```
ExecStart=/usr/bin/dockerd --exec-opt native.cgroupdriver=systemd
```

Restart the Docker service by running the following command:

```
systemctl daemon-reload
systemctl restart docker
systemctl status docker
```

Step 1: Configure Kubernetes Repository

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-
x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
```

Step 2: Install *kubelet*, *kubeadm*, and *kubectl*

```
yum install -y kubelet kubeadm kubectl bash-completion
systemctl enable kubelet
systemctl start kubelet
```

Step 3: Disable SWAP

```
sed -i '/swap/d' /etc/fstab
swapoff -a
```

En todos los nodos:

Set the *net.bridge.bridge-nf-call-iptables* to '1' in your *sysctl* config file. This ensures that packets are properly processed by IP tables during filtering and port forwarding.

```
#vi /etc/sysctl.conf
```

```
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

```
#sysctl -p
```

How to Deploy a Kubernetes Cluster

Step 1: En el nodo master, que tiene la ip 192.168.2.10, de nuestro vagrant:

```
kubeadm init \
  --ignore-preflight-errors=SystemVerification \
  --apiserver-advertise-address=192.168.2.10 \
  --pod-network-cidr=10.244.0.0/16 \
```

Si por cualquier motivo, **tenemos problemas** con el comando, para borrar toda la información creada en el nodo con kubeadm init, utilizaremos el comando:

kubeadm reset (NO UTILIZAR EN EL LABORATORIO)

Step 2: Configure kubectl

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

Step 3: Configure flannel

Configuramos flannel, como red, en este caso, nos descargamos el archivo de configuración a crear, y lo modificamos para que escuche por la interface **eth1**, que es el que tenemos configurada en nuestra mv de vagrant, recordar que el interfaz eth0 es el de nat.

```
#cd /root
```

```
curl -Lo kube-flannel.yml \
  https://github.com/coreos/flannel/raw/master/Documentation/kube-
  flannel.yml
```

```
sed -i.bak -e "s/ip-masq/ip-masq\\n          - --iface=eth1/g" kube-
  flannel.yml
```

```
kubectcl create -f kube-flannel.yml
```

Step 4: Check Status of Cluster

```
kubectcl get pods --all-namespaces
```

```
kubectcl get nodes
```

Step 5: Join Worker Node to Cluster

- Nos conectamos a los nodos que queremos añadir al cluster.
- Ejecutamos el join que se ha indicado en el momento de hacer el “init”, en el master

Ejemplo:

```
kubeadm join 192.168.2.10:6443 --token 79l7r0.8qolwhf45i13s7o0 \
```

```
--discovery-token-ca-cert-hash
```

```
sha256:0be52c968bcfc085953af90fa3c52f665a2242865cc2d589b955a43fedf4c3ae
```

Replace the codes with the ones from your master server. Repeat this action for each worker node on your cluster. Now check the status of the nodes.

Esperamos durante unos minutos y tendremos que ver que los nodos se han unido al cluster:

From the master server enter:

```
[root@master ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	5m42s	v1.18.2
worker1	Ready	<none>	64s	v1.18.2
worker2	Ready	<none>	59s	v1.18.2

```
[root@master ~]# kubectl get pods --all-namespaces -o wide
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP
NODE	NOMINATED NODE	READINESS GATES				
kube-system	coredns-66bff467f8-6q4s5	1/1	Running	0	4m48s	
10.244.0.3	master <none>	<none>				
kube-system	coredns-66bff467f8-9dl1j	1/1	Running	0	4m49s	
10.244.0.2	master <none>	<none>				
kube-system	etcd-master	1/1	Running	0	5m4s	
10.0.2.15	master <none>	<none>				
kube-system	kube-apiserver-master	1/1	Running	0	5m4s	
10.0.2.15	master <none>	<none>				
kube-system	kube-controller-manager-master	1/1	Running	0	5m4s	
10.0.2.15	master <none>	<none>				
kube-system	kube-flannel-ds-amd64-28kfx	1/1	Running	0	29s	
10.0.2.15	worker1 <none>	<none>				
kube-system	kube-flannel-ds-amd64-fb8jn	1/1	Running	0	24s	
10.0.2.15	worker2 <none>	<none>				
kube-system	kube-flannel-ds-amd64-qxx4b	1/1	Running	0	97s	
10.0.2.15	master <none>	<none>				
kube-system	kube-proxy-g7988	1/1	Running	0	24s	
10.0.2.15	worker2 <none>	<none>				
kube-system	kube-proxy-lbjjk	1/1	Running	0	4m49s	
10.0.2.15	master <none>	<none>				
kube-system	kube-proxy-qdl7l	1/1	Running	0	29s	
10.0.2.15	worker1 <none>	<none>				
kube-system	kube-scheduler-master	1/1	Running	0	5m3s	
10.0.2.15	master <none>	<none>				

Ahora modificaciones en la instalación de kubeadm para poder conectar a un pod con instalacion vagran-kubernetes

Las modificaciones se hacen **en cada nodo del cluster**, (master, worker1, worker2), poniendo la ip de cada nodo en su **/etc/sysconfig/kubelet**

```
# vi /etc/sysconfig/kubelet
```

```
KUBELET_EXTRA_ARGS=--node-ip=192.168.2.10
```

```
#systemctl daemon-reload
```

```
#systemctl restart kubelet.service
```

Esta tendría que ser la salida correcta tras la configuración:

```
#kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE
KERNEL-VERSION			CONTAINER-RUNTIME				
master	Ready	master	165m	v1.18.6	192.168.2.10	<none>	CentOS Linux 7 (Core) 3.10.0-957.21.3.el7.x86_64 docker://19.3.12
worker1	Ready	<none>	161m	v1.18.6	192.168.2.11	<none>	CentOS Linux 7 (Core) 3.10.0-957.21.3.el7.x86_64 docker://19.3.12
worker2	Ready	<none>	161m	v1.18.6	192.168.2.12	<none>	CentOS Linux 7 (Core) 3.10.0-957.21.3.el7.x86_64 docker://19.3.12

Ahora podemos comprobar el comportamiento del nuevo clúster desplegando una aplicación:

```
[root@master ~]# kubectl run my-nginx --image=nbrown/nginxhello:1.12.1 --port=80
```

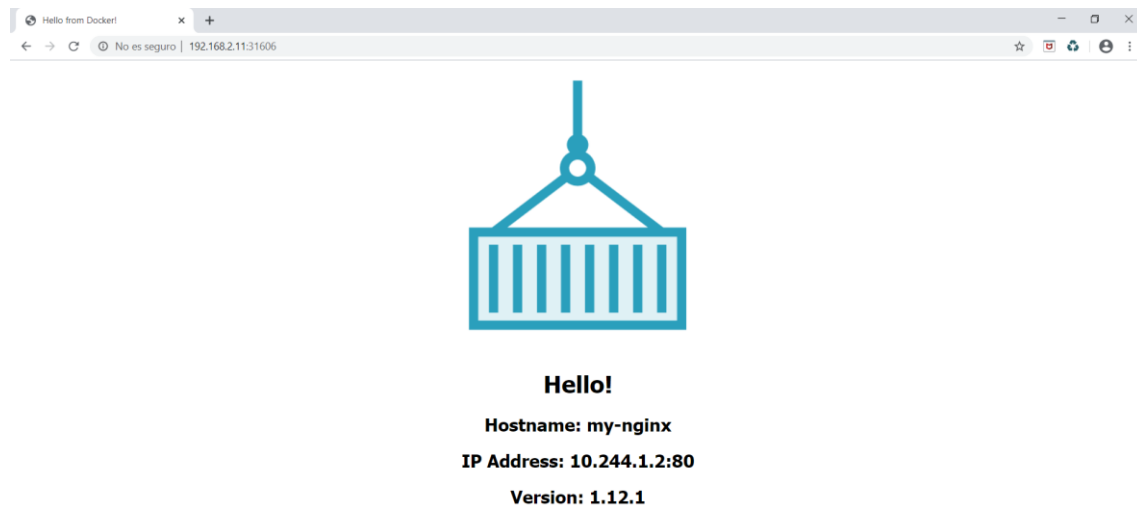
```
[root@master ~]# kubectl get pod
```

```
[root@master ~]# kubectl expose pod my-nginx --type=NodePort --name my-nginx-service
```

```
[root@master ~]# kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	21m
my-nginx-service	NodePort	10.109.2.145	<none>	80:31606/TCP	32s

<http://192.168.2.11:31606/>



Step : Removing a Worker Node from the Cluster

To remove a Kubernetes worker node from the cluster, perform the following operations.

- Migrate pods from the node:

```
kubectl drain <node-name> --delete-local-data --ignore-daemonsets
```

```
[root@master ~]# kubectl drain worker2 --delete-local-data --ignore-daemonsets
```

```
[root@master ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	27m	v1.18.2
worker1	Ready	<none>	22m	v1.18.2
worker2	Ready,SchedulingDisabled	<none>	22m	v1.18.2

- Prevent a node from scheduling new pods use – Mark node as *unschedulable*

```
kubectl cordon <node-name>
```

Eliminar definitivamente desde el master

```
[root@master ~]# kubectl delete node worker2
node "worker2" deleted
```

```
[root@master ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	29m	v1.18.2
worker1	Ready	<none>	24m	v1.18.2

Pasado unos minutos, ya no tendremos ningún pod corriendo en el nodo worker2:

```
[root@master ~]# kubectl get pod --all-namespaces -o wide
```

NAMESPACE	NAME	READY	STATUS	
RESTARTS	AGE	IP	NOMINATED NODE	READINESS
GATES				
default	my-nginx	1/1	Running	0
9m40s	10.244.1.2	worker1	<none>	
kube-system	coredns-66bff467f8-6q4s5	1/1	Running	0
29m	10.244.0.3	master	<none>	
kube-system	coredns-66bff467f8-9dl1j	1/1	Running	0
29m	10.244.0.2	master	<none>	
kube-system	etcd-master	1/1	Running	0
29m	10.0.2.15	master	<none>	
kube-system	kube-apiserver-master	1/1	Running	0
29m	10.0.2.15	master	<none>	
kube-system	kube-controller-manager-master	1/1	Running	0
29m	10.0.2.15	master	<none>	
kube-system	kube-flannel-ds-amd64-28kfx	1/1	Running	1
25m	10.0.2.15	worker1	<none>	
kube-system	kube-flannel-ds-amd64-qxx4b	1/1	Running	0
26m	10.0.2.15	master	<none>	

```

kube-system    kube-proxy-lbjjk          1/1      Running    0
29m           10.0.2.15      master    <none>     <none>
kube-system    kube-proxy-qdl7l          1/1      Running    0
25m           10.0.2.15      worker1    <none>     <none>
kube-system    kube-scheduler-master     1/1      Running    0
29m           10.0.2.15      master    <none>     <none>

```

- Revert changes made to the node by 'kubeadm join' – Run on worker node to be removed

```
sudo kubeadm reset
```

- You can then redo the same process of joining a new node to the cluster once the *kubeadm reset* command has been executed successfully.

Step 1: Get join Token

A token is required when joining a new worker node to the Kubernetes cluster. When you bootstrap a cluster with *kubeadm*, a token is generated which expires after 24 hours.

Check if you have a token – Run the command on Control node:

```

$ kubeadm token list
TOKEN                                TTL      EXPIRES                                USAGES
DESCRIPTION                          EXTRA
GROUPS
nx1jjq.u42y27ip3bhmj8vj             21h      2020-01-10T20:33:08Z
authentication,signing               The default bootstrap token generated by
'kubeadm init'.                      system:bootstrappers:kubeadm:default-node-token

```

If the token is expired, generate a new one with the command:

You can also generate token and print the join command:

```

[root@master ~]# kubeadm token create --print-join-command

W0510 10:39:21.912124    2480 configset.go:202] WARNING: kubeadm
cannot validate component configs for API groups
[kubelet.config.k8s.io kubeproxy.config.k8s.io]
kubeadm join 192.168.2.10:6443 --token twnf4b.x5akz2fmjdr2rs7r --
discovery-token-ca-cert-hash
sha256:4a0dfe2721c78e1358ac6426dea9b14e7563155998bb7dff6c0cec45d80da4a
3

$ kubeadm token list

```

Ahora podemos probar en el nodo **worker2** volver a añadirlo:

```
[root@worker2 ~]# kubeadm join 192.168.2.10:6443 --token twnf4b.x5akz2fmjdr2rs7r --  
discovery-token-ca-cert-hash  
sha256:4a0dfe2721c78e1358ac6426dea9b14e7563155998bb7dff6c0cec45d80da4a3 --ignore-  
preflight-errors=all
```

--ignore-preflight-errors=all, utilizamos este parámetro para que sobrescriba toda la información que teníamos anteriormente.

También **podemos utilizar** el comando **kubeadm reset**, en el nodo para eliminar la configuración que tengamos anteriormente creada.

Habilitar el auto-completado en kubectl

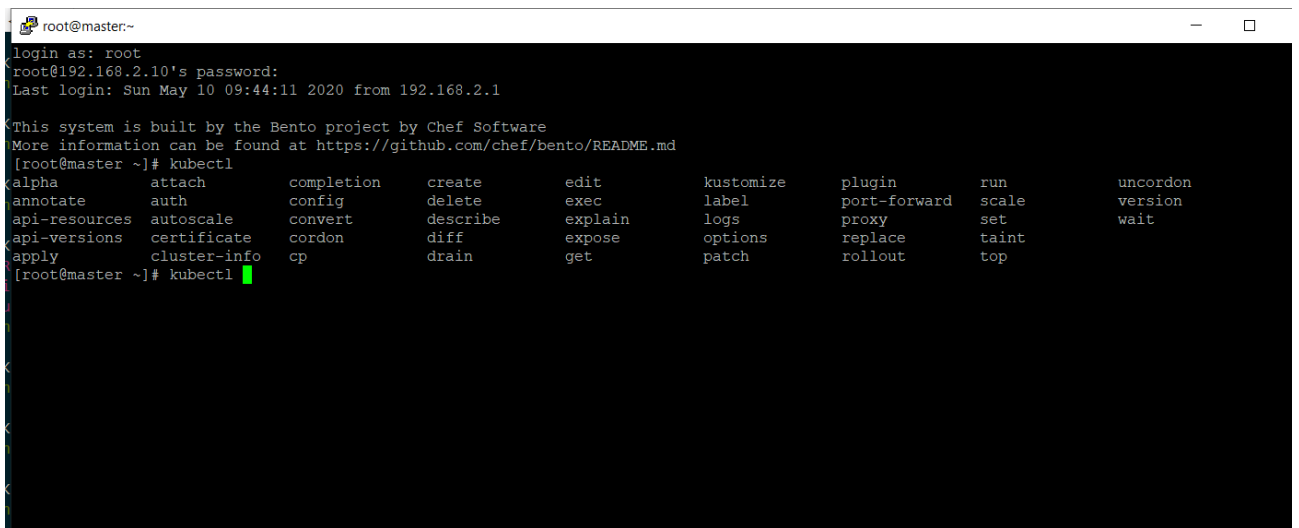
<https://kubernetes.io/es/docs/tasks/tools/install-kubectl/>

```
[root@master ~]# yum install bash-completion -y
```

```
[root@master ~]# echo 'source <(kubectl completion bash)' >> ~/.bashrc
```

```
[root@master ~]# kubectl completion bash >/etc/bash_completion.d/kubectl
```

Tras recargar tu intérprete de comandos, el auto-completado de kubectl debería estar funcionando:

A terminal window titled 'root@master:~' showing the installation and usage of kubectl completion. The terminal output includes login information, system details (Bento project by Chef Software), and a list of kubectl commands. The command 'kubectl' is entered, and a list of suggestions is displayed. The suggestions are: alpha, annotate, api-resources, api-versions, apply, completion, attach, auth, autoscale, certificate, cluster-info, create, config, convert, cordon, cp, delete, describe, diff, drain, edit, exec, explain, expose, get, kustomize, label, logs, options, patch, plugin, port-forward, proxy, replace, rollout, run, scale, set, taint, top, uncordon, version, wait. The cursor is positioned at the end of the 'kubectl' command.

```
root@master:~  
login as: root  
root@192.168.2.10's password:  
Last login: Sun May 10 09:44:11 2020 from 192.168.2.1  
  
This system is built by the Bento project by Chef Software  
More information can be found at https://github.com/chef/bento/README.md  
[root@master ~]# kubectl  
alpha      attach      completion  create      edit         kustomize   plugin      run          uncordon  
annotate   auth        config      delete      exec         label       port-forward scale      version  
api-resources autoscale  convert    describe   explain      logs        proxy       set        wait  
api-versions certificate cordon     diff        expose      options     replace     taint  
apply      cluster-info cp          drain       get         patch       rollout     top  
[root@master ~]# kubectl
```