Clúster con 3 raspberry

https://www.raspberrypi.org/blog/docker-comes-to-raspberry-pi/

Instalamos Raspbian. Tarda unos 5 minutos https://www.raspberrypi.org/downloads/raspbian/

```
xz --decompress 2023-05-03-raspios-bullseye-arm64-lite.img.xz
```

```
dd if=2023-05-03-raspios-bullseye-arm64-lite.img of=/dev/mmcblk0
```

Esto nos crea dos particiones en la tarjeta:

```
bootfs
rootfs
```

Modificamos la tarjeta para poder acceder sin monitor:

En la partición rootfs, ponemos IP fija cambiando el fichero añadiendo al final:

```
rootfs/etc/dhcpcd.conf
```

```
interface eth0
static ip_address=192.168.69.1/24
static routers=192.168.69.1
static domain_name_servers=192.168.69.1
```

Habilitamos ssh dejando un fichero en la partición boot que se llame ssh (da igual el contenido o si está vacio)

```
touch bootfs/ssh
```

A partir de la versión bulleyes tenemos que crear otro usuario para poder acceder:

Crear el fichero en boot llamado userconf con el contenido:

```
vim bootfs/userconf
```

```
username:encrypted_password
```

Por ejemplo, para ruth:odin sacamos la password:

```
echo 'odin' | openssl passwd -6 -stdin
$6$S3pAIx36rcMzDYsK$vzl8eX.2k07Rbje9nJ4zsFQdieKw8Wg296javxQ.VW7SdknBlk03vFKh
0eI8i4VGwPxWHiJCJNnCd7E72Sh8c0
```

Y sería:

```
echo
```

```
'ruth:$6$S3pAIx36rcMzDYsK$vzl8eX.2k07Rbje9nJ4zsFQdieKw8Wg296javxQ.VW7SdknBlk
03vFKh0eI8i4VGwPxWHiJCJNnCd7E72Sh8c0' > bootfs/userconf
```

Modificamos la memoria SWAP que está a 100 y la ponemos a 1024:

```
rootfs/etc/dphys-swapfile
```

```
CONF_SWAPSIZE=1024
```

Deshabilitamos IPV6

https://sleeplessbeastie.eu/2022/07/20/how-to-disable-ipv6-on-raspberry-pi-4/

```
ip -br a
```

```
lo               UNKNOWN        127.0.0.1/8 ::1/128
eth0             UP             172.16.1.101/16 fe80::14b8:2700:4354:ec24/64
wlan0            DOWN
```

```
echo net.ipv6.conf.all.disable_ipv6=1 | sudo tee /etc/sysctl.d/disable-
ipv6.conf
```

Grabamos cambios:

```
sudo sysctl --system
```

Esto para gurb pero no me convence

Añadimos en la línea del fichero

```
/boot/cmdline.txt
```

Que tiene que ser algo así:

```
dwc_otg.lpm_enable=0 console=serial0,115200 console=tty1
root=PARTUUID=ee25660b-02 rootfstype=ext4 elevator=deadline fsck.repair=yes
rootwait quiet init=/usr/lib/raspi-config/init_resize.sh
```

El siguiente parámetro

```
ipv6.disable=1
```

Ya podemos acceder por ssh
Nos conectamos y seguimos modificando

Con raspi-config, modificamos:

-Hostname

```
2. Network Options
```

```
N1 Hostname
```

-Expandimos FileSystem (ya viene hecho):

```
7 Advanced Options
A1 Expand Filesystem
```

-Cambiamos arranque a consola (no hace falta con lite)

```
3 Boot Options
B1 Desktop / CLI
B1 Console
```

-Cambiadmos pais Wifi:

```
4 Localisation Options
I4 Change Wi-fi Country
ES Spain
```

Wifi

https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

El primer nodo se conectará a una red wifi y dará internet al resto.

Escanemos las redes:

```
iwlist wlan0 scan
```

Añadimos la red y la contraseña al final del fichero

```
/etc/wpa_supplicant/wpa_supplicant.conf
```

```
p2p_disabled=1
network={
    ssid="mi_red_wifi"
    psk="***************"
}
```

Probamos de conectarnos a mano:

```
wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf
```

Reiniciamos el servicio:

```
wpa_cli -i wlan0 reconfigure
```

Podemos poner mas de una red en el fichero wpa_supplicant.conf y las seleccionamos con esta

herramienta interactiva:

```
wpa_cli
```

```
list_networks
select_network 1
```

Para el primer nodo le tenemos que quitar el default gw de la red 192.168.69.x. Editamos el fichero:

```
/etc/dhcpcd.conf
```

Y quitamos la línea:

```
static routers=192.168.69.1
```

Para compartir internet:
wlan0: red con internet
eth0: red interna por cable eth0 entre raspberris 192.168.69.1, 192.168.69.2 y 192.168.69.3

Permitimos reenvio entre interfaces editando /etc/sysctl.conf:

```
net.ipv4.ip_forward = 1
```

Lo añadimos en caliente

```
sysctl -p
```

Añadimos reglas iptables

```
iptables -A FORWARD -o wlan0 -i eth0 -m conntrack --ctstate NEW -j ACCEPT
iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
```

Lo hacemos persistente Hay dos formas, guardar las reglas de iptables, no me convence porque puede haber mas y no se si se sobreescriben:

```
iptables-save > /etc/iptables.ipv4.nat
```

Añadimos antes del exit 0 en /etc/rc.local

```
iptables-restore < /etc/iptables.ipv4.nat
```

Debe quedar mas o menos así:

```
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

iptables-restore < /etc/iptables.ipv4.nat
```

```
exit 0
```

La segunda es poner los 3 comandos de iptables en /etc/rc.local antes del exit 0

Instalamos dnsmasq para que funcione el DNS:

```
apt-get install dnsmasq
```

Para poner una entrada, las añadimos en el fichero

```
/etc/dnsmasq.conf
```

Al final ponemos:

```
address=/jenkins.rpicluster.com/192.168.69.2
```

Cada vez que añadamos una tenemos que reinicar el servicio:

```
systemctl restart dnsmasq
```

Punto de acceso

https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md

El tercer nodo hará de Punto de acceso para que otros portátiles se conecten al clúster

```
apt-get install dnsmasq hostapd
```

```
systemctl stop dnsmasq
sudo systemctl stop hostapd
```

/etc/dhcpcd.conf

```
interface wlan0
static ip_address=192.168.4.1/24
```

service dhcpcd restart

mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig

/etc/dnsmasq.conf

```
interface=wlan0
  dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
```

```
/etc/hostapd/hostapd.conf
```

```
interface=wlan0
```

```
driver=nl80211
ssid=raspicluster
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=clusterraspi
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

```
/etc/default/hostapd
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Reiniciamos servicios:

```
systemctl start hostapd
systemctl start dnsmasq
```

Si da un error:

```
Failed to start hostapd.service: Unit hostapd.service is masked.
```

Hacemos esto:

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
sudo systemctl start hostapd
```

Al arrancar el servidor, arranca el servicio de hostapd pero no el AP. Pero si reiniciamos el servicio después de que arranque el servidor si que arranca el AP. Por lo tanto vamos a asegurarnos que el servicio hostapd arranca el útimo: Modificamos la seccion [Unit] del fichero

```
/usr/lib/systemd/system/hostapd.service
```

```
After=network.target network-online.target
Wants=network-online.target
```

Tiene que quedar algo así:

```
[Unit]
Description=Access point and authentication server for Wi-Fi and Ethernet
Documentation=man:hostapd(8)
After=network.target network-online.target
Wants=network-online.target
```

Ahora hacemos que enruten las dos redes:

```
/etc/sysctl.conf
net.ipv4.ip_forward=1
```

Para que aplique hay que reiniciar o lanzar:

```
sysctl -p
```

También añadir iptables. Para hacerlo persistente lo añadimos justo antes del exit 0 en

```
/etc/rc.local
```

```
iptables -t nat -A  POSTROUTING -o eth0 -j MASQUERADE
```

Vemos si funciona:

```
# iwconfig
wlan0     IEEE 802.11  Mode:Master  Tx-Power=31 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:on
```

Podemos grabar logs añadiendo en:

```
/etc/default/hostapd
```

```
DAEMON_OPTS="-dd -t -f /var/log/hostapd.log"

`-dd` -> "more logging"
`-t` -> include timestamps
`-f <path>` -> tells hostapd to log your data to `<path>`
```

DOCKER

Al arrancar los docker, usar:

```
-e TZ=Europe/Madrid
```

Instalamos Docker

```
curl -sSL https://get.docker.com | sh
```

Añadimos el usuario pi al grupo docker

```
usermod -aG docker ruth
```

Para que los docker resuelvan bien dnsmasq y que conecte con docker registry hay que añadir:

```
/etc/docker/daemon.json
```

```
{
  "dns": [
    "192.168.69.1",
        "8.8.8.8",
        "8.8.4.4"
  ],
  "insecure-registries": ["docker.raspi"]
}
```

```
sudo service docker restart
```

Docker Registry

Ponemos el puerto 80 porque el registry lo haremos inseguro para no tener que crear certificados:

```
docker pull registry:2
docker run -d -p 80:5000 --restart=always --name registry registry:2
```

Creamos una imagen, fichero Dockerfile, por ejemplo una simple con git:

```
FROM debian
RUN apt-get update && \
    apt-get install -y git

CMD bash
```

Creamos la imagen:

```
docker build -t docker.raspi/git .
```

La subimos a nuestro Registry:

```
docker push docker.raspi/git
```

Ahora si vamos a otro nodo, la podemos descargar:

```
docker run -ti docker.raspi/git
```

Docker Swarm (deprecated)

```
docker run -ti resin/rpi-raspbian:latest
cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 8 (jessie)"
```

```
docker run -ti arm32v6/alpine:3.5
```

Apache:

```
docker run -ti -p 8080:80 hypriot/rpi-busybox-httpd
```

Si accedemos a 192.168.1.201:8080 nos sale web de apache

Ponemos las siguientes IPs:

```
raspiswarm1: 192.168.1.201
raspiswarm2: 192.168.1.202
raspiswarm3: 192.168.1.203
```

Iniciamos swarm en el nodo1:

```
docker swarm init
```

Para añadir otros nodos nos indica:

```
docker swarm join \
    --token
SWMTKN-1-0kn5tua6jptohuvohoh5v46vb75qscdz7b35hjecynl94xne40-520qqlthfm2bhukh
qpj352sxi \
    192.168.1.201:2377
```

Los añadimos y nos dice:

```
This node joined a swarm as a worker.
```

Ejecutamos un servicio:

```
docker service create --name apache  --replicas=2 -p 8080:80 hypriot/rpi-
busybox-httpd
```

Si hacemos un docker ps vemos que lo ha levantado en la 1 y la 2, pero es accesible desde las 3 ips y va balanceando. Para parar el servicio:

```
docker service rm apache
```

Podemos crearlo mapeando una unidad local para ver que balancea, por ejemplo:

```
docker service create --name apache --replicas=3 -p 8080:80 --mount
type=bind,src=/raspi/www,dst=/www  hypriot/rpi-busybox-httpd
```

Si creamos un /raspi/www/index.html diferente para cada servidor, veremos como balancea

kubernetes

https://kubecloud.io/setup-a-kubernetes-1-9-0-raspberry-pi-cluster-on-raspbian-using-kubeadm-f8b3b

85bc2d1

Tenemos que deshabilitar swap

```
dphys-swapfile swapoff
dphys-swapfile uninstall
update-rc.d dphys-swapfile remove
```

Modificamos:

```
/boot/cmdline.txt
```

```
cgroup_enable=memory
```

Instalamos Docker:

```
curl -sSL get.docker.com | sh && sudo usermod pi -aG docker
```

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key
add - && \
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list && \
sudo apt-get update -q && \
sudo apt-get install -qy kubeadm
```

Inicializamos el clúster. Ponemos el rango de IPs por si tenemos varios, en este caso wlan0 que sale a internet y eth0 que es la red local:

```
kubeadm init --apiserver-advertise-address=192.168.69.1
```

Cuando acaba da lo siguiente:

```
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 should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each
node
as root:

  kubeadm join 192.168.69.1:6443 --token 08607b.fqzo9x9lh7fhp40r --
```

```
discovery-token-ca-cert-hash
sha256:3df520e18e42608c133b01af692b8881f6834319751e461c5deb8534a2055466
```

Ejecutamos esos comandos con el usuario pi.

Miramos el estado de los nodos:

```
kubectl get nodes

NAME          STATUS    ROLES     AGE       VERSION
kubernetes1   NotReady     master    8m        v1.10.2
```

Pone not ready porque falta configurar la red. El pod de dns se queda pendiente

```
kubectl get pods --all-namespaces
NAMESPACE     NAME                                  READY     STATUS
RESTARTS   AGE
kube-system   etcd-kubernetes1                      1/1       Running   0
2m
kube-system   kube-apiserver-kubernetes1            1/1       Running   0
2m
kube-system   kube-controller-manager-kubernetes1   1/1       Running   0
1m
kube-system   kube-dns-686d6fb9c-dwtmh              0/3       Pending   0
2m
kube-system   kube-proxy-j9tmc                      1/1       Running   0
2m
kube-system   kube-scheduler-kubernetes1            1/1       Running   0
2m
```

Lo podemos ver en los logs:

```
sudo journalctl -r -u kubelet

May 02 23:46:39 kubernetes1 kubelet[16196]: E0502 23:46:39.401514   16196
kubelet.go:2125] Container runtime network not ready: NetworkReady=false rea
May 02 23:46:39 kubernetes1 kubelet[16196]: W0502 23:46:39.401016   16196
cni.go:171] Unable to update cni config: No networks found in /etc/cni/net.d
```

Creamos la RED:

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

serviceaccount "weave-net" created
clusterrole.rbac.authorization.k8s.io "weave-net" created
clusterrolebinding.rbac.authorization.k8s.io "weave-net" created
role.rbac.authorization.k8s.io "weave-net" created
rolebinding.rbac.authorization.k8s.io "weave-net" created
daemonset.extensions "weave-net" created
```

Ahora vemos que ya están todos arrancados

```
kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY     STATUS
RESTARTS   AGE
kube-system   etcd-kubernetes1                       1/1       Running   0
6m
kube-system   kube-apiserver-kubernetes1             1/1       Running   0
6m
kube-system   kube-controller-manager-kubernetes1    1/1       Running   0
5m
kube-system   kube-dns-686d6fb9c-dwtmh               3/3       Running   0
6m
kube-system   kube-proxy-j9tmc                       1/1       Running   0
6m
kube-system   kube-scheduler-kubernetes1             1/1       Running   0
6m
kube-system   weave-net-59r4p                        2/2       Running   0
1m
```

Y ya aparece bien en el listado de nodos:

```
kubectl get nodes

NAME          STATUS    ROLES     AGE       VERSION
kubernetes1   Ready     master    8m        v1.10.2
```

Añadimos un nodo al cluster:

```
kubeadm join 192.168.69.1:6443 --token 08607b.fqzo9x9lh7fhp40r --discovery-
token-ca-cert-hash
sha256:3df520e18e42608c133b01af692b8881f6834319751e461c5deb8534a2055466
```

```
This node has joined the cluster:
* Certificate signing request was sent to master and a response
  was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.
```

Desde el master miramos los nodos:

```
pi@kubernetes1:~ $ kubectl get nodes
NAME          STATUS     ROLES     AGE       VERSION
kubernetes1   Ready   master    45m       v1.10.2
kubernetes2   NotReady   <none>    3m        v1.10.2
```

Aparece **NotReady** porque hay que comentar la red cni en los otros nodos. Comentamos la línea que pone **KUBELET_NETWORK_ARGS** /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

#Environment="KUBELET_NETWORK_ARGS=–network-plugin=cni –cni-conf-dir=/etc/cni/net.d –cni-bin-dir=/opt/cni/bin"

Dashboard

Instalamos Dashboard:

```
echo -n 'apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard
  labels:
    k8s-app: kubernetes-dashboard
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system' | kubectl apply -f -
```

Se crea el servicio:

```
clusterrolebinding.rbac.authorization.k8s.io "kubernetes-dashboard" created
```

No he probado:

```
$ kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/alt
ernative/kubernetes-dashboard-arm.yaml
```

Hay que probar este:

```
pi@kubernetes1:~ $ kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/alt
ernative/kubernetes-dashboard-arm.yaml
serviceaccount "kubernetes-dashboard" created
role.rbac.authorization.k8s.io "kubernetes-dashboard-minimal" created
rolebinding.rbac.authorization.k8s.io "kubernetes-dashboard-minimal" created
deployment.apps "kubernetes-dashboard" created
service "kubernetes-dashboard" created
```

Para ver donde se está ejecutando:

```
# kubectl -n kube-system get service kubernetes-dashboard
```

```
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
```

```
AGE
kubernetes-dashboard   ClusterIP   10.103.58.176   <none>        80/TCP
13m
```

Con el proxy debería funcionar pero no lo he conseguido:

```
kubectl proxy
```

Yo he hecho un tunel y abierto en local:

```
ssh -L 8001:10.103.58.176:80 pi@192.168.69.1
```

Abrimos http://127.0.0.1:8001

Podemos acceder directamente al nodo, pero hay que habilitar la autenticación
Documentación:
https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/#deploying-the-dashboard-ui

```
https://192.168.69.1:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/
```

DESKTOP

Descargar la imagen con Desktop:
https://www.raspberrypi.com/software/operating-systems/

```
Raspberry Pi OS (64-bit) > Raspberry Pi OS with desktop
```

O instalamos las X

```
sudo apt-get install lightdm
```

En raspi-config seleccionamos la opción

```
1. System Options
S5 Boot / Auto Login
B4 Desktop Autologin Desktop GUI, automatically logged in as 'ruth' user
```

Para poder conectar con las X por vnc configuramos desde raspi-config:

```
3 Interface Options
I3 VNC           Enable/disable graphical remote access using RealVNC
```

Creo que es lo mismo que instalar:

```
realvnc-vnc-server
```

Para no poner contraseña:

```
/root/.vnc/config.d/vncserver-x11
```

```
Authentication=None
Encryption=AlwaysOff
```

Si queremos contraseña ojo que tiene que ser de 6 a 8 carácteres con:

```
sudo vncpasswd -service -legacy
```

Para reiniciar el servicio

```
systemctl restart vncserver-x11-serviced.service
```

Arranque de una aplicación automáticamente

Nos aseguramos que tengamos instalado chromium-browser

```
~/.config/lxsession/LXDE/autostart
```

```
@chromium-browser --start-fullscreen --app=http://web.raspi
```

Como cambiar de ventanas

Instalamos las aplicaciones necesarias

```
apt-get install xdotool wmctrl
```

Exportamos DISPLAY para conectarnos a la pantalla:

```
export DISPLAY=:0
```

Para listar las aplicaciones abiertas:

```
wmctrl -l
```

```
0x01200004  0 raspberrypi VNC Server
0x01600002  0 raspberrypi Sign in [Jenkins]
0x0200000b  0 raspberrypi EmulationStation
```

Seleccionamos la que queremos llevar al frente

```
wmctrl -i -a 0x01200004
```

Ahora la podemos cerrar por ejemplo:

```
xdotool keydown Alt key F4
```

Vemos que se ha cerrado:

```
wmctrl -l
```

```
0x01600002  0 raspberrypi Sign in [Jenkins]
0x0200000b  0 raspberrypi EmulationStation
```

También podríamos cambiar entre aplicaciones simulando ALT+TAB

```
xdotool keydown Alt key Tab keyup Alt
```

Y con xdotool simular cualquier pulsación de teclas

Retropie

```
git clone --depth=1 https://github.com/RetroPie/RetroPie-Setup.git
```

```
cd RetroPie-Setup
chmod +x retropie_setup.sh
sudo ./retropie_setup.sh
```

Seleccionar “Basic Install”

Para cambiar de una aplicación a otra remotamente:

```
sudo apt-get install xdotool
```

Cambiamos desde la conexión ssh al display del monitor:

```
export DISPLAY=:0
```

Lanzamos el comando ALT+TAB

```
xdotool keydown Alt key Tab keyup Alt
```

Monitorización

https://www.bogotobogo.com/DevOps/Docker/Docker_Prometheus_Grafana.php

Vienen 2 repositorios, he levantado este docker compose y monitoriza los dockers del nodo donde se ejecuta:

https://github.com/stefanprodan/dockprom

Botones

https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

Con el comando pinout muestra un mapa de los pins

Instalamos python paquetes necesarios:

```
apt-get install python3-pip
pip3 install RPi.GPIO
```