**Custom bridge network**

---------------------------------

* A **custom bridge network** in Docker is useful for scenarios where you need more control over the networking behaviour of your containers.
* It allows you to define your own subnets, gateways, and ensures that containers can communicate securely within a defined network.
* Using custom bridge networks gives you greater flexibility and isolation when managing containerized applications
* docker network create --driver bridge my-custom
* docker network ls
* docker network inspect my-custom
* docker run -it --name alpine1 --network my-custom alpine ash
* docker ps
* docker container inspect d0
* docker network create --driver bridge --subnet 172.21.0.0/24 --gateway 172.21.0.1 custom-bridge
* docker network inspect
* docker network ls
* docker run -it --name alpine2 --network my-custom alpine ash
* docker run -it --name alpine3 --network my-custom alpine ash
* docker exec -it <container\_id\_alpine3> ash
* ping alpine2
* docker ps
* docker container inspect 1f
* docker exec -it <container\_id> ash
* docker network create --driver bridge --subnet 172.21.0.0/24 --gateway 172.21.0.1 custom-bridge
* docker network inspect
* docker network ls
* docker network inspect d9
* docker network connect custom-bridge container-name
* docker network disconnect custom-bridge container-name

**Difference between default and custom bridge ???**

**Default bridge**

* **Configuration**: It uses Docker's default settings, meaning it assigns IP addresses from a default subnet (typically 172.17.0.0/16), with the default gateway 172.17.0.1.

Docker assigns a subnet from its default IPAM (IP Address Management) pool, which often includes subnets like 172.17.0.0/16, 172.18.0.0/16, and sometimes 172.24.0.0/16 depending on the available ranges.

* **Container Communication**: Containers attached to the default bridge network can communicate with each other using their IP addresses,

**Custom bridge**

* **Configuration**: You have full control over the network’s configuration, including:
* **Subnet**: You can define a custom subnet (e.g., 192.168.100.0/24) to isolate containers from other networks or the host.
* **Gateway**: You can specify a custom gateway (e.g., 192.168.100.1).

**Container Communication**: Containers connected to a custom bridge network can automatically communicate with each other using their container names or IP address

apt-cache madison kubeadm | tac

sudo apt-get install -y kubelet=1.30.0-1.1 kubectl=1.30.0-1.1 kubeadm=1.30.0-1.1 🡪 0th version of 30

kubectl describe node kmaster | grep "Taint"

kubectl taint node kmaster node-role.kubernetes.io/control-plane:NoSchedule-

day1

1 hostnamectl set-hostname kmaster

2 hostname

3 cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

4 overlay

5 br\_netfilter

6 EOF

7 sudo modprobe overlay

8 sudo modprobe br\_netfilter

9 cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

10 net.bridge.bridge-nf-call-iptables = 1

11 net.bridge.bridge-nf-call-ip6tables = 1

12 net.ipv4.ip\_forward = 1

13 EOF

14 sudo sysctl --system

15 sudo swapoff -a

16 (crontab -l 2>/dev/null; echo "@reboot /sbin/swapoff -a") | crontab - || true

17 sudo apt-get update -y

18 sudo apt-get install -y software-properties-common gpg curl apt-transport-https ca-certificates

19 curl -fsSL https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/Release.key | gpg --dearmor -o /etc/apt/keyrings/cri-o-apt-keyring.gpg

20 echo "deb [signed-by=/etc/apt/keyrings/cri-o-apt-keyring.gpg] https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/ /" | tee /etc/apt/sources.list.d/cri-o.list

21 sudo apt-get update -y

22 sudo apt-get install -y cri-o

23 sudo systemctl daemon-reload

24 sudo systemctl enable crio --now

25 sudo systemctl start crio.service

26 VERSION="v1.30.0"

27 wget https://github.com/kubernetes-sigs/cri-tools/releases/download/$VERSION/crictl-$VERSION-linux-amd64.tar.gz

28 sudo tar zxvf crictl-$VERSION-linux-amd64.tar.gz -C /usr/local/bin

29 rm -f crictl-$VERSION-linux-amd64.tar.gz

30 KUBERNETES\_VERSION=1.30

31 sudo mkdir -p /etc/apt/keyrings

32 curl -fsSL https://pkgs.k8s.io/core:/stable:/v$KUBERNETES\_VERSION/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

33 echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v$KUBERNETES\_VERSION/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list

34 sudo apt-get update -y

35 apt-cache madison kubeadm | tac

36 sudo apt-get install -y kubelet=1.30.10-1.1 kubectl=1.30.10-1.1 kubeadm=1.30.10-1.1

37 sudo apt-get install -y jq

38 local\_ip="$(ip --json addr show eth0 | jq -r '.[0].addr\_info[] | select(.family == "inet") | .local')"

39 cat > /etc/default/kubelet << EOF

40 KUBELET\_EXTRA\_ARGS=--node-ip=$local\_ip

41 EOF

42 kubeadm init

43 mkdir -p $HOME/.kube

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

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

46 exit

47 kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml

48 watch -n kubectl get nodes

49 watch -n 1 kubectl get nodes

50 watch -n 1 kubectl get pods -n kube-system

51 kubectl run selmipod --image=nginx

52 kubectl get pod

53 kubectl describe pod selmipod

54 kubectl describe node kmaster | grep "Taint"

55 kubectl taint node kmaster node-role.kubernetes.io/control-plane:NoSchedule

56 kubectl taint node kmaster node-role.kubernetes.io/control-plane:NoSchedule-

57 kubectl get pod