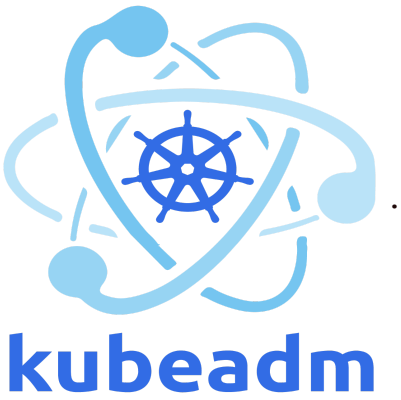
Installing kubernetes on Debian 11

This guide is intended to help you install k8s on debian 11 bullseye.

This guide was written starting from the official Kubernetes pages Installation. There is more than one method to install k8s, this method is based on *kubeadmnin*.



Prerequisites

- Machines with Debian 11 installed;
- At least 2GB of ram per host;
- 2 CPU or VCPU per host;
- Full network connectivity between hosts;
- Unique *Hostname*, *mac address* e *product uuid* per host.
    - To check the mac addresses you can use the command **ip link** or **ifconfig -a**;
    - The *product uuid* is obtained with the command **sudo cat /sys/class/dmi/id/product_uuid**;
- Swap must be disabled for each host (**swapoff -a**). It should be disabled directly on the */etc/fstab* file to avoid having to execute **swapoff -a** each time the host is restarted;
- Installation of the *container runtime*.

Checking the ports

Specific ports must be enabled for hosts in the cluster. For this argument we need to distinguish between **master** node and **slave**/**worker** nodes.

Nodo Master

PROTOCOL	DIRECTION	PORTS	DESCRIPTION
TCP	ingress	6443	Kubernetes API server
TCP	ingress	2379-2380	**etcd** server client API
TCP	ingress	10250	**Kubelet** API
TCP	ingress	10259	**kube-scheduler**
TCP	ingress	10257	**kube-controller-manager**

Nodi slave o worker

PROTOCOL	DIRECTION	PORTS	DESCRIPTION
TCP	ingress	10250	**Kubelet** API
TCP	ingress	30000-32767	**NodePort** services

Container runtime

A *container runtime* must be installed on each node. With version 1.27 of kubernetes, you can only use runtimes that are **Container Runtime Interface (CRI)** compliant.

Come runtime ve ne sono di diversi. Ad esempio, si potrebbe usare **docker** ma non sarebbe sufficiente in quanto **docker** non è aderente allo standard **CRI** e pertanto andrà affiancato da una componente aggiuntiva che è cri-dockerd. In alternativa a **docker** si userà containerd.

Configurazione della rete

Eseguiamo il comando:

```
$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# sysctl params required by setup, params persist across reboots
$ cat <<EOF| sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Apply sysctl params without reboot
$ sudo sysctl --system
```

Bisogna verificare che i moduli del kernel **br_netfilter** e **overlay** sono stati caricati:

```
$ lsmod | grep br_netfilter
$ lsmod | grep overlay
```

Verifichiamo che **net.bridge.bridge-nf-call-iptables**, **net.bridge.bridge-nf-call-ip6tables**, **net.ipv4.ip_forward** sono impostati a 1:

```
$ sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-
call-ip6tables net.ipv4.ip_forward
```

Fatto questo, per ogni nodo bisogna aggiungere nel file */etc/hosts* la tabella degli *ip* e *dns* di ogni nodo del cluster. In alternativa si può usare un server *dns* come **bind**.

Installazione di containderd

In debian si può installare tramite *apt* ma aggiungendo la versione rilasciata da *docker*. La versione fornita con debian non è compatibile con l'ultima versione di *kubernetes*.

Installiamo **curl** e il supporto ad **apt** per il protocollo https:

```
$ sudo apt-get update
$ sudo apt-get install -y apt-transport-https ca-certificates curl
```

Aggiungiamo il repository per docker e containerd. Per le versioni dalla 11 in giù la cartella */etc/apt/keyrings* non esiste e va creata prima di eseguire le istruzioni di aggiunta del repository:

```
$ sudo install -m 0755 -d /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/debian/gpg| sudo gpg -
-dearmor -o /etc/apt/keyrings/docker.gpg
$ sudo chmod a+r /etc/apt/keyrings/docker.gpg

$ echo \
  "deb [arch="$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable"| \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-
buildx-plugin docker-compose-plugin
```

Applichiamo la configurazione di default:

```
$ sudo containerd config default| sudo tee
/etc/containerd/config.toml >/dev/null 2>&1
```

Bisogna modificare la configurazione in modo da usare come driver per i *cgroup* **systemd** al posto di cgroupfs.

Modifichiamo il file */etc/containerd/config.toml* in modo che dopo la riga:

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options
]
```

ci sia **SystemdCgroup = true**.

Avviamo **containerd** e lo abilitiamo per lo startup all'avvio:

```
$ sudo systemctl restart containerd
$ sudo systemctl enable containerd
```

Installazione di kubernetes

Per prima cosa bisogna abilitare il repository di *kubernetes* su debian:

Aggiungiamo il repository:

```
$ sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
$ sudo echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-
keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main"| sudo
tee /etc/apt/sources.list.d/kubernetes.list
```

Installiamo **kubeadm**, **kubelet** e **kubectl** e ne blocchiamo le versioni:

```
$ sudo apt-get update
$ sudo apt-get install -y kubelet kubeadm kubectl
$ sudo apt-mark hold kubelet kubeadm kubectl
```

Attivazione del cluster

L'installazione del cluster avviene con il comando:

```
$ sudo kubeadmin init
```

Se tutto è andato per il verso giusto, avremo come risposta qualcosa di simile a questo:

> *Your Kubernetes control-plane 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*
>
> *Alternatively, if you are the root user, you can run:*
>
> *export KUBECONFIG=/etc/kubernetes/admin.conf*

> *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 control-plane nodes by copying certificate authorities and service account keys on each node and then running the following as root:*
>
> *kubeadm join master:6443 --token wb4p8t.kethcu6y2a7dz75n *
> *--discovery-token-ca-cert-hash*
> *sha256:a870710a2002c3b793e54045ab52095226934e854d613e3a80bf6630ad9d01c9*
> **
> *--control-plane*
>
> *Then you can join any number of worker nodes by running the following on each as root:*
>
> *kubeadm join master:6443 --token wb4p8t.kethcu6y2a7dz75n *
> *--discovery-token-ca-cert-hash*
> *sha256:a870710a2002c3b793e54045ab52095226934e854d613e3a80bf6630ad9d01c9*

Aggiunta dei nodi al cluster

Per interagire con il cluster dobbiamo configurare l'utente che eseguira i comandi con **kubectl**. Pertanto, come utente **non** *root* eseguiamo i comandi:

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

Per riavere o rigenerare il token, dal nodo *master*, eseguire il comando:

```
$ sudo kubeadm token create --print-join-command
```

Dai nodi *slave* eseguiamo il comando:

```
# xxxxx dipende da cosa viene restituito dal comando di generazione
del token
$ sudo kubeadm join master:6443 --token xxxxx --discovery-token-ca-
cert-hash sha256:xxxxx
```

Dopo aver aggiunto tutti i nodi al cluster, eseguiamo il comando:

```
$ kubectl get nodes
```

Avremo una risposta simile a questa:

```
user@master:~$ kubectl get nodes
NAME     STATUS     ROLES           AGE    VERSION
master   NotReady   control-plane   40h    v1.27.1
slave1   NotReady   <none>          175m   v1.27.1
slave2   NotReady   <none>          20s    v1.27.0
```

Lo status è **NotReady** per via del fatto che manca la rete overlay per i **pod**.

La rete *overlay* è una rete che si posiziona "sopra" i nodi del cluseter e viene usata per far comunicare i nodi tra di loro.

Attivazione della rete overlay per i pod

Perchè i pod possano interagire tra loro è necessario installare un *POD network addon*. Un addon del genere non fa altro che installare una rete overlay per i pod. Ne esistono di diversi, al link Network plugin si ha un elenco delle possibili scelte.

Per semplicità, le scelte possibili ricadono su:

- Calico;
- Flannel;
- Weave net.

Nel nostro caso useremo **Weave net**. Dal nodo master e come utente "normale":

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

Avremmo come risultato:

```
$ user@master:~$ kubectl apply -f
"https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-
daemonset-k8s.yaml"
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.apps/weave-net created
```

L'esecuzione del comando `kubectl get nodes` fornirà:

```
user@master:~$ kubectl get nodes
NAME     STATUS   ROLES           AGE    VERSION
master   Ready    control-plane   40h    v1.27.1
slave1   Ready    <none>          3h7m   v1.27.1
slave2   Ready    <none>          12m    v1.27.0
```

Tutti i nodi sono **Ready**.

Test del cluster

Per testare il cluster, creiamo un deploy:

```
$ kubectl create deployment --image nginx my-nginx --replicas 3
```

Abbiamo usato l'immagine di **nginx** con 3 repliche. Il nome di questo *deployment* è **my-nginx**.

Per verificare lo stato della creazione del *deployment*:

```
$ user@master:~$ kubectl get pods
NAME                       READY   STATUS              RESTARTS   AGE
my-nginx-b8dd4cd6-fpwpz    0/1     ContainerCreating   0          40s
my-nginx-b8dd4cd6-qfxk7    0/1     ContainerCreating   0          40s
my-nginx-b8dd4cd6-v24hl    0/1     ContainerCreating   0          40s
```

I container sono, ancora, in fase di creazione.

Dopo qualche minuto:

```
$ user@master:~$ kubectl get pods
$ NAME                       READY   STATUS    RESTARTS   AGE
$ my-nginx-b8dd4cd6-fpwpz    1/1     Running   0          3m54s
$ my-nginx-b8dd4cd6-qfxk7    1/1     Running   0          3m54s
$ my-nginx-b8dd4cd6-v24hl    1/1     Running   0          3m54s
```

Vediamo che le tre repliche sono state create.

Altro comando che ci fa vedere che le tre repliche sono up:

```
$ user@master:~$ kubectl get deployment
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
my-nginx   3/3     3            3           9m29s
```

Se vogliamo avere il dettaglio dei pod creati:

```
$ user@master:~$ kubectl describe pod my-nginx
Name:             my-nginx-b8dd4cd6-fpwpz
Namespace:        default
Priority:         0
Service Account:  default
Node:             slave1/192.168.0.176
Start Time:       Tue, 18 Apr 2023 13:36:01 +0000
Labels:           app=my-nginx
                  pod-template-hash=b8dd4cd6
Annotations:      <none>
Status:           Running
IP:               10.40.0.3
IPs:
  IP:           10.40.0.3
Controlled By:  ReplicaSet/my-nginx-b8dd4cd6
Containers:
  nginx:
    Container ID:
containerd://784852248f6e1c0d56ccaaf40fc6217aefea34f29afe2f9359ed6940
88119dfc
    Image:          nginx
    Image ID:
docker.io/library/nginx@sha256:63b44e8ddb83d5dd8020327c1f40436e37a6ff
fd3ef2498a6204df23be6e7e94
...
...
```

Vediamo che l'ip del pod è **10.40.0.3**, possiamo testare il server http nginx:

```
$ user@master:~$ curl 10.40.0.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
```

```
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully
installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Vediamo che **nginx** ha risposto correttamente.

Volendo, possiamo esporre **my-ngnix** fuori dalla rete dei pod:

```
$ kubectl expose deployment my-nginx --name=nginx-http --type
NodePort --port 80 --target-port 80
service/nginx-http exposed
$ kubectl describe svc nginx-http
Name:                     nginx-http
Namespace:                default
Labels:                   app=my-nginx
Annotations:              <none>
Selector:                 app=my-nginx
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.101.79.21
IPs:                      10.101.79.21
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  31561/TCP
```

```
Endpoints:                10.40.0.3:80,10.40.0.4:80,10.40.0.5:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Possiamo interrogare **my-nginx** su uno dei nodi usando la porta **31561**.

```
$ user@master:~$ curl master:31561
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully
installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```