

# Creating the containercap cluster

## Installing requirements

**Make sure you install go before you start this manual:**

```
Sudo apt install golang-go
```

1. First of all we are going to install an NFS-server that will be used for accessing files of the network:

```
sudo apt install libnfsidmap-dev
sudo apt install libnfsidmap1
sudo apt install nfs-common
sudo apt install nfs-kernel-server
```

Secondly we need to create a directory that we use as a mount location for containercap.

You can make a directory on your system (I put this on my desktop for comfort):

```
cd /mnt
mkdir /ContainerCap
```

Now we setup this directory with:

```
echo "mnt/ContainerCap
*(rw,insecure,sync,no_subtree_check,no_root_squash)" | sudo tee
/etc/exports

sudo systemctl start nfs-server.service
```

We can check our nfs-server with the following command:

```
(kali㉿kali)-[~]
└─$ sudo systemctl status nfs-server.service
● nfs-server.service - NFS server and services
   Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; preset: disabled)
   Active: active (exited) since Mon 2023-02-27 15:34:16 CET; 22min ago
     Process: 690 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=1/FAILURE)
    Process: 692 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
    Main PID: 692 (code=exited, status=0/SUCCESS)
      CPU: 126ms
```

2. Now we are going to install docker (this is the method given by the official kali website):

```
sudo apt update
sudo apt install -y docker.io
sudo systemctl enable docker
```

To use docker without sudo:

```
sudo usermod -aG docker $USER
```

Now we are going to turn off swap for Kubernetes (this is currently in the works) with the following commands:

```
sudo swapoff -a
sudo nano /etc/fstab (Comment out the second line)
```

Next we need to check if we have no boundary on the amount of tasks we can run:

```
sudo systemctl edit docker
```

Make sure that `TasksMax=infinity`

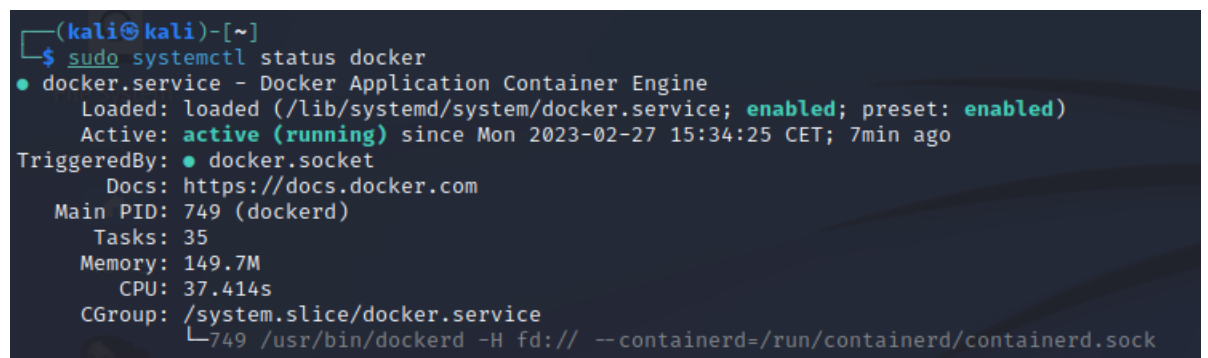
Now we can reload the docker daemon and docker itself:

```
sudo systemctl daemon-reload
sudo systemctl reload docker
```

Now we enable (when the system boots up, docker will automatically start) and start docker:

```
sudo systemctl enable docker
sudo systemctl start docker
```

We can see if docker is up and running by the following command:



```
(kali㉿kali)-[~]
└─$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enabled)
   Active: active (running) since Mon 2023-02-27 15:34:25 CET; 7min ago
     TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
    Main PID: 749 (dockerd)
      Tasks: 35
     Memory: 149.7M
        CPU: 37.414s
    CGroup: /system.slice/docker.service
            └─749 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

If docker is not active, we just restart docker with:

```
Sudo systemctl restart docker
```

### 3. We install docker-ce, docker-ce-cli and containerd to prepare our system Docker Engine

Because we are working on a Debian system we need to use the “bullseye” version because of stability:

```
printf '%s\n' "deb https://download.docker.com/linux/debian
bullseye stable" | sudo tee /etc/apt/sources.list.d/docker-ce.list

curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg
--dearmor -o /etc/apt/trusted.gpg.d/docker-ce-archive-keyring.gpg
```

Now we install the latest version of docker-ce:

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

#### 4. Now we will install Kubernetes (kubelet, kubect!, kubeadm)

First of all make sure that the required ports([required ports](#)) are open. Because I used a brand new kali vm without firewall options, this was no problem.

You can check the ports like this:

```
nc 127.0.0.1 6443
```

We need to make sure we have setup our Container Runtime Interface (CRI), We are going to use cri-dockerd that works as an Docker Engine (Run these commands as a root):

```
git clone https://github.com/Mirantis/cri-dockerd.git
# Run these commands as root
###Install GO###
wget https://storage.googleapis.com/golang/getgo/installer_linux
chmod +x ./installer_linux
./installer_linux
source ~/.bash_profile

cd cri-dockerd
mkdir bin
go build -o bin/cri-dockerd
mkdir -p /usr/local/bin
install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-dockerd
cp -a packaging/systemd/* /etc/systemd/system
sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service
systemctl daemon-reload
systemctl enable cri-docker.service
systemctl enable --now cri-docker.socket
```

Now we will update the package index and install packages needed to use the k8s apt repository:

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

Now download the Google Cloud public signing key and add the apt repository:

```
sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
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
```

Lastly you can install and pin the versions of kubelet, kubeadm and kubectl:

```
sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

sudo systemctl enable kubelet.service
```

5. The following commands will be used whenever we change/lose connection to the internet on our device (we are basically making a fixed interface for a single-node Kubernetes cluster on a local host):

```
sudo ip link add kube_dummy type dummy
sudo ip addr add 10.10.131.44/24 dev kube_dummy
sudo ip link set kube_dummy up
```

We can check our interface with the following command:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:22:46:4f brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
        valid_lft 61765sec preferred_lft 61765sec
    inet6 fe80::c58b:de52:3c93:5562/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:00:a0:2a:7b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
4: kube_dummy: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether d6:89:9c:6e:62:05 brd ff:ff:ff:ff:ff:ff
    inet 10.10.131.44/24 scope global kube_dummy
        valid_lft forever preferred_lft forever
    inet6 fe80::d489:9c6e:6205/64 scope link
        valid_lft forever preferred_lft forever
```

6. In this step we will setup our Kubernetes cluster

First of all we will run the following command:

```
sudo apt-get update && sudo apt-get upgrade
```

Now we can create the cluster:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --cri-socket=unix:///var/run/cri-dockerd.sock --control-plane-endpoint 10.10.131.44:6443 --apiserver-advertise-address 10.10.131.44 --upload-certs
```

The control-plane node is the machine where the control plane components run, including `etcd` (the cluster database) and the `API Server` (which the `kubectl` command line tool communicates with).

- (Recommended) If you have plans to upgrade this single control-plane `kubeadm` cluster to high availability you should specify the `--control-plane-endpoint` to set the shared endpoint for all control-plane nodes. Such an endpoint can be either a DNS name or an IP address of a load-balancer.
- Choose a Pod network add-on, and verify whether it requires any arguments to be passed to `kubeadm init`. Depending on which third-party provider you choose, you might need to set the `--pod-network-cidr` to a provider-specific value. See [Installing a Pod network add-on](#).
- (Optional) `kubeadm` tries to detect the container runtime by using a list of well known endpoints. To use different container runtime or if there are more than one installed on the provisioned node, specify the `--cri-socket` argument to `kubeadm`. See [Installing a runtime](#).
- (Optional) Unless otherwise specified, `kubeadm` uses the network interface associated with the default gateway to set the advertise address for this particular control-plane node's API server. To use a different network interface, specify the `--apiserver-advertise-address=<ip-address>` argument to `kubeadm init`. To deploy an IPv6 Kubernetes cluster using IPv6 addressing, you must specify an IPv6 address, for example `--apiserver-advertise-address=2001:db8::101`

We used the Docker Engine Container Runtime Interface to run the pods (cri-dockerd.sock)

You will get the following screen:

```
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 the control-plane node running the following command on each as root:

kubeadm join 10.0.2.15:6443 --token 9akn9w.wt2cul95ncssq9v5 \
--discovery-token-ca-cert-hash sha256:4f835a855a62c3b8ae5c79605465c36ce12fed908e5c0df8152f50ee7471ff3d \
--control-plane --certificate-key 39ddd64ddef6c5992f17bc6ecb34733e17febcb86ae95b914ff33d2cdb1fca4fe

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.0.2.15:6443 --token 9akn9w.wt2cul95ncssq9v5 \
--discovery-token-ca-cert-hash sha256:4f835a855a62c3b8ae5c79605465c36ce12fed908e5c0df8152f50ee7471ff3d
```

To make kubectl work for your non-root users, use these commands:

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

Next we need to taint our masternode (we can check the name of the node with `kubectl get nodes`):

```
(kali@kali)-[~/Desktop/nfs]
$ kubectl get nodes
NAME     STATUS    ROLES    AGE   VERSION
kali     Ready     control-plane 17h   v1.26.1
```

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```

7. After creating a cluster, we are going to install some Pod network add-ons that help us in networking:

To be able to install some add-ons we need to run this command:

```
sudo apt-get install containernetworking-plugins
```

- The first addon is WeaveNet ([WeaveNet](https://weave.works/docs/weave/latest/en/Getting-Started/))

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

To check if our cluster is running we can use:

```

L$ kubectl get pods -A -o wide

```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
kube-system	coredns-787d4945fb-r9gr7	1/1	Running	0	5m12s	10.32.0.3	kali	<none>	<none>
kube-system	coredns-787d4945fb-t7xwv	1/1	Running	0	5m12s	10.32.0.4	kali	<none>	<none>
kube-system	etcd-kali	1/1	Running	0	5m22s	10.0.2.15	kali	<none>	<none>
kube-system	kube-apiserver-kali	1/1	Running	0	5m22s	10.0.2.15	kali	<none>	<none>
kube-system	kube-controller-manager-kali	1/1	Running	0	5m27s	10.0.2.15	kali	<none>	<none>
kube-system	kube-proxy-fwwvs	1/1	Running	0	5m12s	10.0.2.15	kali	<none>	<none>
kube-system	kube-scheduler-kali	1/1	Running	0	5m22s	10.0.2.15	kali	<none>	<none>
kube-system	weave-net-rljw4	2/2	Running	0	5m1s	10.0.2.15	kali	<none>	<none>

We see that all of our pods are running so we can say that the cluster is fully working.

- The second addon is Weave scope ([Weave scope](#))

```
kubectl apply -f
https://github.com/weaveworks/scope/releases/download/v1.13.2/k
8s-scope.yaml
```

This addon gives us a tool for graphically visualizing your containers, pods, services etc

We can start the tool in a terminal with the following command:

```
kubectl port-forward -n weave "$ (kubectl get -n weave pod --
selector=weave-scope-component=app -o
jsonpath='{.items..metadata.name}')" 4040
```

## 8. (Optional) Install the Kubernetes dashboard add-on

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
```

If you check your pods again, you can see that the Kubernetes-dashboard pods are in pending state (and won't change). Therefore you can use the article from [Shashank Srivastava](#) until you get the login screen.

Now we need to find the token we can use to log in. Execute the following command:

```
kubectl -n kubernetes-dashboard create token admin-user
```

Now you get a line like this:

[illegible]

Copy this token and put it in the enter token field -> now you can sign in

Kubernetes Dashboard

☐ Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the [Configure Access to Multiple Clusters](#) section.

☒ Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the [Authentication](#) section.

Enter token

.....

Sign in

To end the cluster part we need to create the PersistentVolume and PersistentVolumeClaim (this can only be done in the containercap directory so this step needs to be done after step 9):

```
kubectl create -f pv-nfs.yaml
kubectl create -f pvc-nfs.yaml
```

## 9. Now we can clone the project and setup

First you clone the project in a chosen directory:

```
Git clone https://gitlab.ilabt.imec.be/lpdhooge/containercap.git
```

Git Add personal access token to push/pull from this repository.

Add personal access token to push/pull from containercap-imagery.

```
docker login gitlab.ilabt.imec.be:4567 -u <user> -p <access_token>
```

You can check whether you can pull images using this line:

```
docker run --rm -it gitlab.ilabt.imec.be:4567/lpdhooge/containercap-imagery/joy:latest
```

We need to create a go project like the tree below:

```

go
├── bin
│
├── pkg
│   ├── linux_amd64
│   ├── mod
│   └── sumdb
├── src
│   ├── gitlab.ilabt.imec.be
│   │   └── lpdhooge
│   │       └── containercap
│   │           └── ...

```

If not, you can just create the bin, pkg and src folders inside a go directory and go into src/gitlab.ilabt.imec.be/lpdhooge/containercap/ and do a git pull and go get here.

When you are in the containercap directory, you can update/add all the dependencies for the project:

```
go get -u ./...
```

Then we are going to build the project (do this command in the containercap directory):

```
go build
```

10. We just need to setup our mount location for the project:

```
cd /mnt/ContainerCap
sudo mkdir containercap-captures containercap-completed containercap-
scenarios containercap-transformed
```

11. If you have problems with building the go project, try using the following command

```
sudo apt-get install libpcap-dev
```

## Reconfiguring the cluster when switching network

Use the following startup bash script in order to fix swap file and fixed interface for kubernetes:

```
#!/bin/bash
# Startup script for containercap
# Initializing dummy network connection for kubernetes
sudo swapoff -a
sudo ip link add kube_dummy type dummy
sudo ip addr add 10.10.131.44/24 dev kube_dummy
sudo ip link set kube_dummy up
echo "Loading..."
sleep 7
kubectl cluster-info
```

If you get errors using `kubectl get pods -A` we are going to reset our cluster (do step by step):

```
sudo apt-get update && sudo apt-get upgrade
sudo kubeadm reset --cri-socket=unix:///var/run/cri-dockerd.sock

sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --cri-
socket=unix:///var/run/cri-dockerd.sock --control-plane-endpoint
10.10.131.44:6443 --apiserver-advertise-address 10.10.131.44 --upload-certs

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
kubectl taint nodes --all node-role.kubernetes.io/control-plane-

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



```
kubectl apply -f
https://github.com/weaveworks/scope/releases/download/v1.13.2/k8s-
scope.yaml

kubectl create -f pv-nfs.yaml
kubectl create -f pvc-nfs.yaml
```

**To start the scope:**

```
kubectl port-forward -n weave "$(kubectl get -n weave pod --selector=weave-
scope-component=app -o jsonpath='{.items..metadata.name}')" 4040
```

**If you want to run vscode as a root:**

```
sudo code --user-data-dir=/home/kali/.config/Code/ --no-sandbox --disable-
gpu-sandbox
```

Now check again, normally this will do.