

Année Universitaire 2024 – 2025  
UFR de Mathématiques & Informatique  
Systèmes ubiquitaires (Edge, Vehicular et UAV)  
Master 2 Informatique



Université Paris Cité

**INFORMATIQUE RÉSEAUX &  
SYSTÈMES AUTONOMES**

---

# Rapport de TP Free5GC

---

**KASDI Sarah  
&**

**VETTIVELKUMARAN Arrthy**

15 Décembre 2024

## Sommaire

0.1	Introduction . . . . .	3
1	Partie 1 : Installation de la VM (Machine Virtual)	4
2	Partie 2 : Installation de kinD & kubectI	6
3	Partie 3 : Install Multus CNI	9
4	Partie 4 : deploiment du Free5GC	12
4.1	Configurez un proxy chaussettes avec un proxy foxy et un DynamicForward vers la machine virtuelle . . . . .	13
4.2	Conclusion . . . . .	16

## 0.1 Introduction

Dans le cadre de ce travail pratique, nous avons exploré la mise en œuvre d'un réseau 5G basé sur la virtualisation des fonctions réseau (NFV). L'objectif principal était de comprendre les concepts fondamentaux liés à la conteneurisation et à l'intégration des technologies cloud-native dans le domaine des télécommunications modernes. Ce TP a permis de mettre en pratique des outils essentiels au déploiement de réseaux 5G, tels que Kubernetes, Docker, et Helm.

Pour atteindre cet objectif, une simulation a été réalisée en s'appuyant sur le dépôt GitHub WillemBerr/upc-free5gc. Ce projet a été réalisé sur plusieurs étapes :

Prérequis :

Mise en place d'une machine virtuelle pour servir d'environnement de travail. Téléchargement et configuration du dépôt GitHub towards5gc-helm. Installation des commandes essentielles, notamment kind, helm, et kubectl.

Création du cluster Kubernetes :

Configuration d'un cluster Kubernetes composé de deux nœuds :

**Worker** : Responsable de l'exécution des charges de travail.

**Control-plane** : Chargé de l'orchestration et de la gestion du cluster.

Configuration réseau :

Téléchargement et installation des plugins CNI pour assurer la connectivité entre les conteneurs et les interfaces réseau.

**Container Network Interface (CNI)** est un framework permettant de configurer dynamiquement les ressources réseau. Il utilise un groupe de bibliothèques et de spécifications écrites en Go. La spécification du plug-in définit une interface pour configurer le réseau, provisionner les adresses IP et maintenir la connectivité avec plusieurs hôtes.

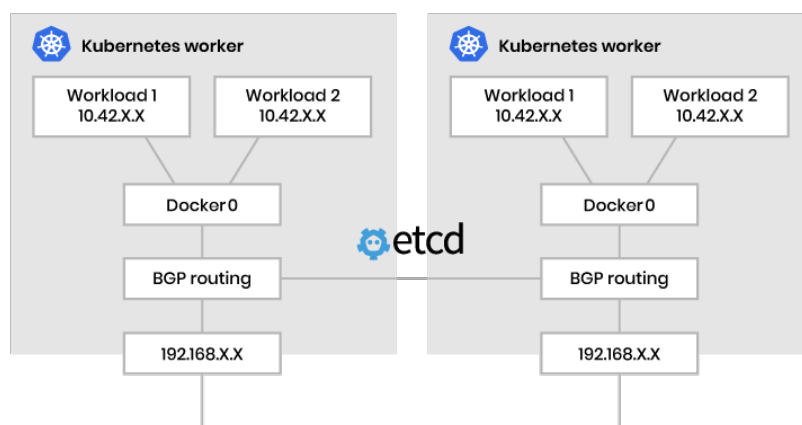


FIGURE 1 – CNI

Déploiement de Free5GC : Installation et configuration de la solution Free5GC sur le cluster Kubernetes. Connexion à l'interface utilisateur Web (WebUI) pour superviser et valider les fonctionnalités du réseau.

Ce travail pratique a permis de mettre en relation les concepts théoriques étudiés en cours avec leur application concrète. En particulier, il a mis en lumière les défis techniques et les solutions associées au déploiement de réseaux 5G, notamment dans le cadre d'architectures cloud-native. À travers cette expérience, nous avons également découvert le rôle clé des technologies modernes comme Kubernetes, NFV, et Docker pour la construction de réseaux modernes, évolutifs et performants.

## 1 Partie 1 : Installation de la VM (Machine Virtual)

On va installer sur VirtualBox ubuntu server, version 22.04.5 amd64. On trouve l'image sur le lien <https://releases.ubuntu.com/jammy/>

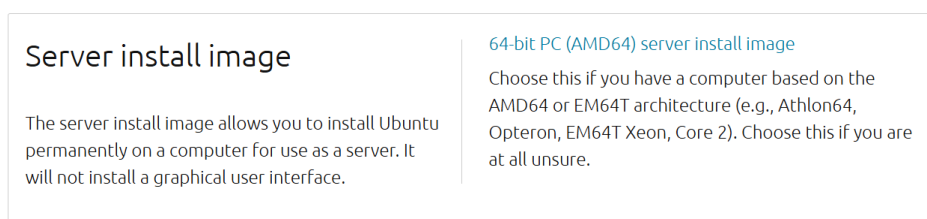


FIGURE 2 – image Ubuntu

- On va ensuite configurer une nouvelle machine virtuelle sous virtualbox avec l'image installer.

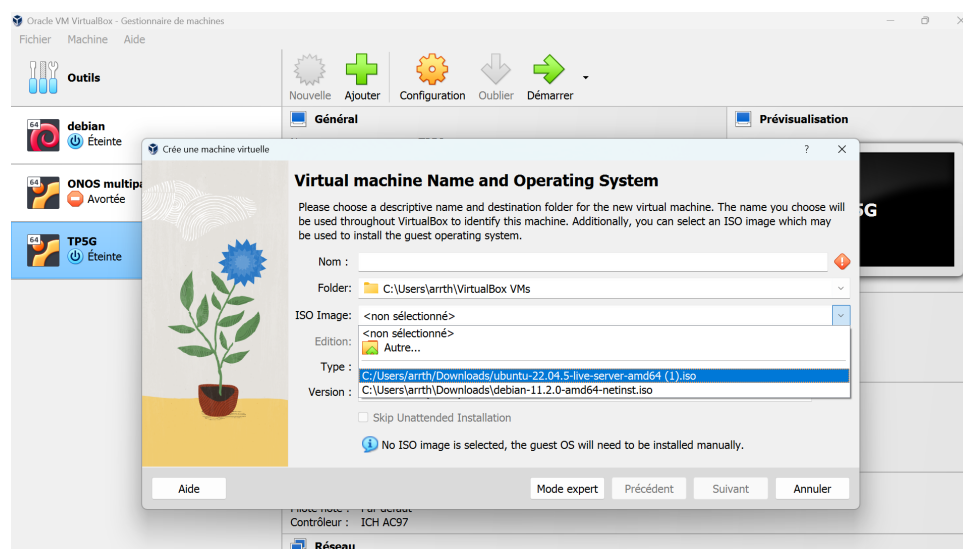


FIGURE 3 – Machine Virtuelle VM

- Notre machine doit être configurée ainsi : 8GB de Ram, 4 CPU, et au moins 25GB de stockage.

Une fois la machine prête on va devoir installer un serveur ssh pour pouvoir effectuer des copier coller depuis notre ordinateur.

Pour cela on va aller dans la configuration de la machine virtuelle et aller dans Réseau et ensuite dans redirection de port.

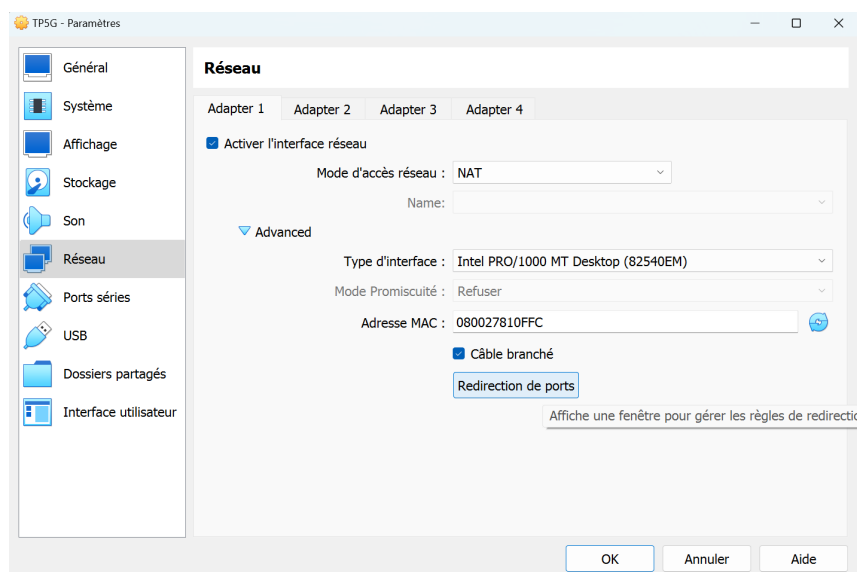


FIGURE 4 – Machine Virtuelle Paramètres Réseau

On fait une redirection de port via les outils **de redirection de port de virtualBox** :

**127.0.0.1 2222 -> IP de votre VM port : 22**

- **IP de la VM** peut être retrouver en tapant sur la ligne de commande **“ip a”**

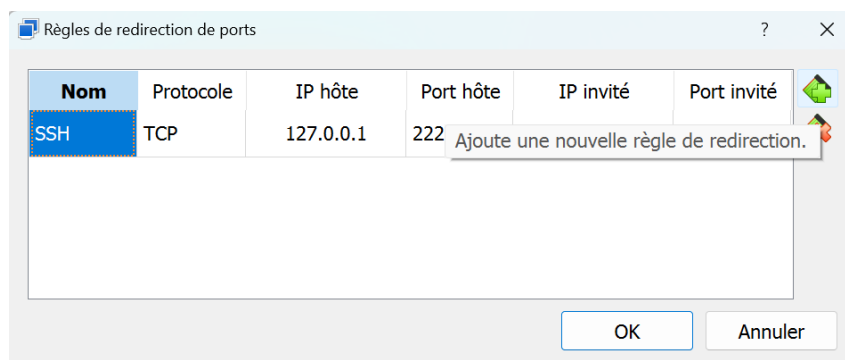


FIGURE 5 – Redirection de port de virtualBox

**Le Port 22 représente :** Secure Shell (SSH). SSH est l'un des nombreux protocoles de tunneling qui créent des connexions réseau sécurisées. - il est utilisé pour la communication Secure Shell (SSH) et permet l'accès à l'administration à distance de la machine virtuelle . En général, le trafic est chiffré à l'aide de l'authentification par mot de passe.

Une fois la redirection effectuée on pourra à partir de la ligne de commande de l'ordinateur se connecter à la ligne de commande de la VM

```

arrthy@vbox: ~
arrth@HUB-PF39KKQJ MINGW64 ~
$ ssh arrthy@127.0.0.1 -p 2222
arrthy@127.0.0.1's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-126-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of jeu. 05 déc. 2024 14:03:34 UTC

System load:  0.28           Processes:           140
Usage of /:   23.3% of 13.82GB Users logged in:         1
Memory usage: 3%            IPv4 address for enp0s3: 10.0.2.15
Swap usage:   0%

La maintenance de sécurité étendue pour Applications n'est pas activée.
20 mises à jour peuvent être appliquées immédiatement.
Pour afficher ces mises à jour supplémentaires, exécuter : apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
New release '24.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Dec  5 14:00:18 2024
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

arrthy@vbox:~$

```

FIGURE 6 – connexion avec la VM à partir de la ligne de commande

## 2 Partie 2 : Installation de kind & kubectl

- Télécharger kind & kubectl

```

arrthy@vbox:~$ [ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.25.0/kind-linux-amd64
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 97 100 97 0 0 483 0 --:--:-- --:--:-- --:--:-- 485
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
100 9697k 100 9697k 0 0 1751k 0 0:00:05 0:00:05 --:--:-- 2196k
arrthy@vbox:~$ chmod +x ./kind
chmod: cannot access './kind': No such file or directory
arrthy@vbox:~$ chmod +x ./kind
arrthy@vbox:~$ sudo mv ./kind /usr/local/bin/kind
[sudo] password for arrthy:
arrthy@vbox:~$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 138 100 138 0 0 775 0 --:--:-- --:--:-- --:--:-- 779
100 53.7M 100 53.7M 0 0 1374k 0 0:00:40 0:00:40 --:--:-- 1681k
arrthy@vbox:~$

```

FIGURE 7 – installation de kind &amp; kubectl

installer kubectl + verification que kubectl c'est bien installée

```

arrthy@vbox:~$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
arrthy@vbox:~$ kubectl version --client
Client Version: v1.31.3
Kustomize Version: v5.4.2
arrthy@vbox:~$

```

FIGURE 8 – installer kubectl + verification que kubectl c'est bien installée

Pour créer un cluster on tape la commande `sudo kind create cluster --name sarah --name sarah` pour ajouter un nom, (par défaut kind) **problème ! il faut installer docker**

```
arrthy@vbox:~$ kind create cluster --name sarah
ERROR: failed to create cluster: failed to get docker info: command "docker info --format '{{json .}}'" failed
with error: exec: "docker": executable file not found in $PATH
Command Output:
```

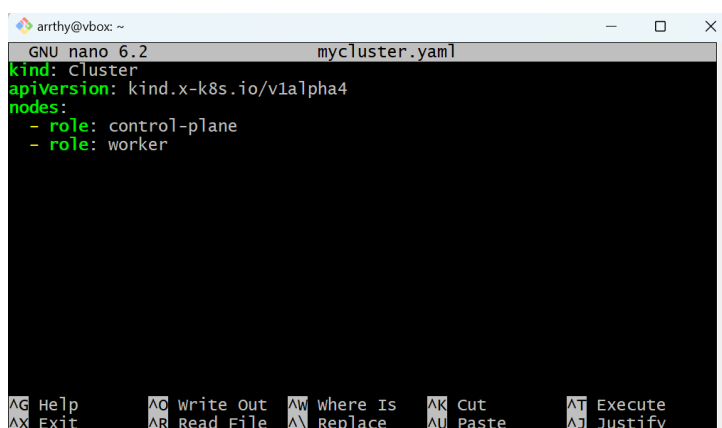
FIGURE 9 – erreur avec kind

On va donc installer docker avec la commande `sudo snap install docker`

```
arrthy@vbox:~$ sudo snap install docker
docker 27.2.0 from Canonical✓ installed
```

FIGURE 10 – installation de Docker

pour créer un cluster on va créer un fichier `.yaml` qu'on a nommer ***mycluster.yaml*** en utilisant **nano** on va editer le fichier :



```
GNU nano 6.2 mycluster.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
```

FIGURE 11 – créer un cluster .yaml

ensuite on créer le cluster en utilisant la commande :

`sudo kind create cluster --config mycluster.yaml`

```
arrthy@vbox:~$ nano mycluster.yaml
arrthy@vbox:~$ cat mycluster.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
arrthy@vbox:~$ sudo kind create cluster --config mycluster.yaml
Creating cluster "kind" ...
✓ Ensuring node image (kindest/node:v1.31.2)
✓ Preparing nodes
✓ Writing configuration
✓ Starting control-plane
✓ Installing CNI
✓ Installing StorageClass
✓ Joining worker nodes
set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s.io/#community
```

FIGURE 12 – création d'un Cluster

après avoir créer un dossier cni on va télécharger : <https://github.com/containernetworking/plugins/releases/download/v1.6.0/cni-plugins-linux-amd64-v1.6.0.tgz>

```

arrthy@arrthy:~/cni$ curl -L -O https://github.com/containernetworking/plugins/releases/download/v1.6.0/cni-plugins-linux-amd64-v1.6.0.tgz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           % Done    0         0              0             0      0     0    0
100 50.2M  100 50.2M    0     0  4445k      0  0:00:11  0:00:11 --:--:-- 5777k

```

FIGURE 13 – Téléchargement du CNI

On a télécharger les binaires du cni puis on le extrait :

```

arrthy@arrthy:~/cni$ ls
cni-plugins-linux-amd64-v1.6.0.tgz
arrthy@arrthy:~/cni$ tar -xvzf cni-plugins-linux-amd64-v1.6.0.tgz
./
./vrf
./firewall
./LICENSE
./macvlan
./static
./host-device
./host-local
./loopback
./sbr
./tuning
./bridge
./README.md
./ptp
./bandwidth
./vlan
./portmap
./ipvlan
./dummy
./tap
./dhcp
arrthy@arrthy:~/cni$ ls
bandwidth  cni-plugins-linux-amd64-v1.6.0.tgz  dummy  host-device  ipvlan  loopback  portmap  README.md  static  tuning  vrf
bridge     dhcp  firewall  host-local  LICENSE  macvlan  ptp      sbr        tap      vlan

```

FIGURE 14 – Téléchargement et extraction des binaires du cni

**-erreur :** on a du extraire le fichier de cni dans un répertoire dédié à celui-ci car sinon docker ne nous laissait pas copier les fichiers /opt/cni/bin

```

arrthy@vbox:~$ mkdir cni
arrthy@vbox:~$ mv cni
cni/
cni/ cni-plugins-linux-amd64-v1.6.0.tgz
arrthy@vbox:~$ mv cni-plugins-linux-amd64-v1.6.0.tgz cni

```

FIGURE 15

Copier les binaire dans les cluster docker (pods)

```

arrthy@vbox:~/cni$ sudo docker cp . 193d2f623f17:/opt/cni/bin/
Successfully copied 145MB to 193d2f623f17:/opt/cni/bin/
arrthy@vbox:~/cni$ |

```

FIGURE 16 – Copier les binaire dans les cluster docker

```

arrthy@vbox:~/cni$ sudo docker cp . 1e5760502c6e:/opt/cni/bin/
Successfully copied 145MB to 1e5760502c6e:/opt/cni/bin/
arrthy@vbox:~/cni$ cd ..

```

FIGURE 17 – Docker ps



On peut voir qu'à présent nous avons tout ça :

```
arrthy@arrthy:~$ ls
cni  get_helm.sh  kubect1  mycluster.yaml
arrthy@arrthy:~$
```

FIGURE 18 – ls

### 3 Partie 3 : Install Multus CNI

**Multus CNI** : est un plugin d'interface réseau de conteneurs (CNI) pour Amazon EKS qui permet de connecter plusieurs interfaces réseau à un Pod.

- on a installé multus cni avec la commande : `git clone https://github.com/k8snetworkplumbingwg/multus-cni` et on a rencontré **un problème l'installation ne s'est pas terminée**, lorsque **on a rajouté sudo** et cela a fonctionné !

installation du multus cni :

```
arrthy@arrthy:~$ git clone https://github.com/k8snetworkplumbingwg/multus-cni
Cloning into 'multus-cni'...
remote: Enumerating objects: 45579, done.
remote: Counting objects: 100% (7022/7022), done.
remote: Compressing objects: 100% (2166/2166), done.
remote: Total 45579 (delta 4919), reused 5664 (delta 4785), pack-reused 38557 (from 1)
Receiving objects: 100% (45579/45579), 52.27 MiB | 3.60 MiB/s, done.
Resolving deltas: 100% (23276/23276), done.
Updating files: 100% (4604/4604), done.
```

FIGURE 19 – installation de Multus CNI

déploiement :

```
arrthy@arrthy:~$ cd multus-cni/
arrthy@arrthy:~/multus-cni$ cat ./deployments/multus-daemonset-thick.yml | sudo kubectl apply -f -
customresourcedefinition.apiextensions.k8s.io/network-attachment-definitions.k8s.cni.cncf.io created
clusterrole.rbac.authorization.k8s.io/multus created
clusterrolebinding.rbac.authorization.k8s.io/multus created
serviceaccount/multus created
configmap/multus-daemon-config created
daemonset.apps/kube-multus-ds created
```

FIGURE 20 – déploiement de Multus CNI

On installe ensuite le **free5gc chart** qu'on trouve ici :

- <https://github.com/free5gc/free5gc-helm/tree/main>

**On installe tout d'abord la commande helm** : `curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3`  
`chmod 700 get_helm.sh`  
`./get_helm.sh`

**Helm** est un gestionnaire de packages pour Kubernetes. Helm est un projet open source créé à l'origine par DeisLabs et donné à la Cloud Native Foundation (CNCF). La CNCF maintient et a gradué le projet.

```

arrthy@vbox:~$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
arrthy@vbox:~$ chmod 700 get_helm.sh
chmod: cannot access 'get_helm.sh': No such file or directory
arrthy@vbox:~$ ls
cni free5gc-helm get_helm.sh kubect1 mycluster.yaml
arrthy@vbox:~$ chmod 700 get_helm.sh
arrthy@vbox:~$ ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.16.3-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm

```

FIGURE 21 – installation de helm

On execute le pod worker nod dans le bash avec  
**docker exec -it bb53fa4e462d /bin/bash** pour pouvoir récupérer les ip a et ip r

```

arrthy@vbox:~/multus-cni$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
f6897a8ac471   kindest/node:v1.31.2               "/usr/local/bin/entr..." 3 days ago    Up 51 minutes    127.0.0.1:37039->6443/tcp          kind-worker
1e5760502c6e   kindest/node:v1.31.2               "/usr/local/bin/entr..." 3 days ago    Up 51 minutes    127.0.0.1:37039->6443/tcp          kind-control-plane
193a2f623f17   kindest/node:v1.31.2               "/usr/local/bin/entr..." 3 days ago    Up 51 minutes    127.0.0.1:33109->6443/tcp          sarah-control-plane
arrthy@vbox:~/multus-cni$ sudo docker exec -it f6897a8ac471 /bin/bash
root@kind-worker:/# ip a
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
5: eth0@91f6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fc00:f853:cdd:e793:12/64 scope global nodad
        valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe12:12/64 scope link
        valid_lft forever preferred_lft forever
root@kind-worker:/# ip r
default via 172.18.0.1 dev eth0
10.244.0.0/24 via 172.18.0.1 dev eth0
172.18.0.0/16 dev eth0 proto kernel scope link src 172.18.0.2

```

FIGURE 22 – kind worker ip a et ip r

ensuite on va configurer les fichiers yaml values et volume

```

excludeIP: 10.100.50.230
n6network:
  enabled: true
  name: n6network
  type: ipvlan
  masterIf: eth0
  subnetIP: 172.18.0.0
  cidr: 16
  gatewayIP: 172.18.0.0
  excludeIP: 172.18.0.0
n9network:
  enabled: true
  name: n9network
  type: ipvlan
  masterIf: eth0
  subnetIP: 10.100.50.224
  cidr: 29
  gatewayIP: 10.100.50.230
  excludeIP: 10.100.50.230

```

FIGURE 23 – fichier values.yaml

```
# network paramters
n3if: # GTP-U
  ipAddress: 10.100.50.233
n4if: # PFCP
  ipAddress: 10.100.50.241
n6if: # DN
  ipAddress: 172.18.0.22
```

FIGURE 24 – fichier values.yaml

On va créer un dossier kubedata dans le worker node :

```
cni free5gc-helm get_helm.sh kubectl mycluster.yaml snap
arrthy@arrthy:~$ sudo docker exec -it 251afc4016ce /bin/bash
[sudo] password for arrthy:
Sorry, try again.
[sudo] password for arrthy:
root@kind-worker:/#
root@kind-worker:/# mkdir /home/kubedata
```

FIGURE 25 – créer un dossier kubedata

puis on applique le volume de création

```
arrthy@arrthy:~$ sudo kubectl apply -f volume.yaml
persistentvolume/example-local-pv9 created
```

FIGURE 26 – APPLY ymal

pour pouvoir avoir la list des pods on tape :

la commande kubectl get pods -A

lorsqu'on a rajoute pas l'argument -A il n'y a pas d'epods qui s'affiche. il ne faut surtout pas oublier cette argument.

```
arrthy@vbox:~$ sudo kubectl get pods
No resources found in default namespace.
```

FIGURE 27 – no resources found in default namespace

```
arrthy@vbox:~$ sudo kubectl get pods -A
NAMESPACE      NAME                                READY   STATUS    RESTARTS   AGE
kube-system     coredns-7c65d6cfc9-swvn4          1/1     Running   0           4m19s
kube-system     coredns-7c65d6cfc9-vnr9l          1/1     Running   0           4m19s
kube-system     etcd-kind-control-plane            1/1     Running   0           4m24s
kube-system     kindnet-9bcr1                     1/1     Running   0           4m19s
kube-system     kindnet-cqdbq                     1/1     Running   0           4m12s
kube-system     kube-apiserver-kind-control-plane  1/1     Running   0           4m24s
kube-system     kube-controller-manager-kind-control-plane  1/1     Running   0           4m24s
kube-system     kube-proxy-9xzcf                   1/1     Running   0           4m19s
kube-system     kube-proxy-fcpth                   1/1     Running   0           4m12s
kube-system     kube-scheduler-kind-control-plane  1/1     Running   0           4m24s
local-path-storage  local-path-provisioner-57c5987fd4-rk4ff  1/1     Running   0           4m19s
arrthy@vbox:~$
```

FIGURE 28 – Running

## 4 Partie 4 : deploiement du Free5GC

Creation du namespace :

```
arrthy@arrthy:~$ sudo kubectl create ns free5gc
namespace/free5gc created
```

FIGURE 29 – namespace

On déploie le free5gc premier a partir du free5gc

```
LAST DEPLOYED: Mon Dec 9 14:59:47 2024
NAMESPACE: free5gc
STATUS: deployed
REVISION: 1
NOTES:
# Source: free5gc-1.0.0
# Software Name : towards5gs-helm
# SPDX-FileCopyrightText: Copyright (c) 2021 Orange
# SPDX-License-Identifier: Apache-2.0
#
# This software is distributed under the Apache License 2.0,
# the text of which is available at https://github.com/Orange-OpenSource/towards5gs-helm/blob/main/LICENSE
# or see the "LICENSE" file for more details.
#
# Author: Abderrahmane KHICHANE, Ithem FAJJARI
# Software description: An open-source project providing Helm charts to deploy 5G components (Core + RAN) on top of Kubernetes
#
# Visit the project at https://github.com/Orange-OpenSource/towards5gs-helm
#
# 1. Get the list of created Pods by running:
# kubectl get pods --namespace free5gc -l "project=free5gc"
#
# Release notes (What's changed in this version):
# - Fix TLS configuration for SBI communications
arrthy@arrthy:~$
```

FIGURE 30 – deploiement

On va ensuite regarder l'état des pods

```
arrthy@arrthy:~$ sudo kubectl get pods -n free5gc
NAME                                READY   STATUS    RESTARTS   AGE
free5gc-premier-free5gc-amf-amf-5d687c54f7-rr7kv    0/1     Init:0/1   0           114s
free5gc-premier-free5gc-ausf-ausf-5449bf6856-p6phb   0/1     Init:0/1   0           115s
free5gc-premier-free5gc-dbp-dbp-698df74c84-kxbk4    0/1     Init:0/1   0           115s
free5gc-premier-free5gc-nrf-nrf-5fd7c65474-dxw9m    0/1     Init:0/1   0           116s
free5gc-premier-free5gc-nssf-nssf-68b48f6686-7m4ql   0/1     Init:0/1   0           116s
free5gc-premier-free5gc-pcf-pcf-775c565f4d-fflqx    0/1     Init:0/1   0           115s
free5gc-premier-free5gc-smf-smf-7d79b5fd58-d82qd    0/1     Init:0/1   0           113s
free5gc-premier-free5gc-udm-udm-5c969b4868-k6sk5    0/1     Init:0/1   0           116s
free5gc-premier-free5gc-udr-udr-6cb7899d9b-98bwh    0/1     Init:0/1   0           116s
free5gc-premier-free5gc-upf-upf-5b85f8479c-tf12w    0/1     ContainerCreating 0           115s
free5gc-premier-free5gc-webui-webui-5bbf8b7889-7m9fn 0/1     Init:0/1   0           116s
mongod-0                                           0/1     Running    0           22s
```

FIGURE 31 – l'état des pods

les pods sont dans l'état d'initialisation : on re tape la commande pour re verifier

```
arrthy@arrthy:~/free5gc/charts/free5gc-upf$ sudo kubectl get pods -n free5gc
NAME                                READY   STATUS    RESTARTS   AGE
free5gc-premier-free5gc-amf-amf-5d687c54f7-rr7kv    0/1     Init:0/1   0           22s
free5gc-premier-free5gc-ausf-ausf-5449bf6856-p6phb   1/1     Running    0           22s
free5gc-premier-free5gc-dbp-dbp-698df74c84-kxbk4    1/1     Running    0           22s
free5gc-premier-free5gc-nrf-nrf-5fd7c65474-dxw9m    1/1     Running    0           22s
free5gc-premier-free5gc-nssf-nssf-68b48f6686-7m4ql   1/1     Running    0           22s
free5gc-premier-free5gc-pcf-pcf-775c565f4d-fflqx    1/1     Running    0           22s
free5gc-premier-free5gc-smf-smf-7d79b5fd58-d82qd    0/1     Init:0/1   0           22s
free5gc-premier-free5gc-udm-udm-5c969b4868-k6sk5    1/1     Running    0           22s
free5gc-premier-free5gc-udr-udr-6cb7899d9b-98bwh    1/1     Running    0           22s
free5gc-premier-free5gc-upf-upf-5b85f8479c-tf12w    0/1     ContainerCreating 0           22s
free5gc-premier-free5gc-webui-webui-5bbf8b7889-7m9fn 1/1     Running    0           22s
mongod-0                                           0/1     Running    0           22s

arrthy@arrthy:~/free5gc/charts/free5gc-upf$ sudo kubectl get svc -n free5gc
NAME                                TYPE               CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
free5gc-premier-free5gc-amf-service ClusterIP           10.96.190.223   <none>            80/TCP           23s
free5gc-premier-free5gc-ausf-service ClusterIP           10.96.194.70    <none>            80/TCP           23s
free5gc-premier-free5gc-dbp-service ClusterIP           10.96.104.226   <none>            80/TCP           23s
free5gc-premier-free5gc-nrf-service ClusterIP           10.96.163.246   <none>            80/TCP           23s
free5gc-premier-free5gc-nssf-service ClusterIP           10.96.114.48    <none>            80/TCP           23s
free5gc-premier-free5gc-pcf-service ClusterIP           10.96.219.190   <none>            80/TCP           23s
free5gc-premier-free5gc-udr-service ClusterIP           10.96.222.241   <none>            80/TCP           23s
free5gc-premier-free5gc-upf-service ClusterIP           10.96.13.254    <none>            27017/TCP        23s
free5gc-premier-free5gc-webui-service ClusterIP           10.96.166.157   <none>            8000/TCP          23s
webui-service NodePort           10.96.127.90    <none>            3000:30500/TCP    23s

arrthy@arrthy:~/free5gc/charts/free5gc-upf$ sudo kubectl get nodes -o wide
NAME              STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
kind-control-plane Ready    control-plane 3d2h   v1.31.2   172.18.0.1    <none>         Debian GNU/Linux 12 (bookworm) 5.15.0-126-generic containerd://1.7.18
kind-worker       Ready    <none>      3d2h   v1.31.2   172.18.0.2    <none>         Debian GNU/Linux 12 (bookworm) 5.15.0-126-generic containerd://1.7.18
```

FIGURE 32 – Les pods son en running

#### 4.1 Configurez un proxy chaussettes avec un proxy foxy et un DynamicForward vers la machine virtuelle

##### - Télécharger extension foxy proxy

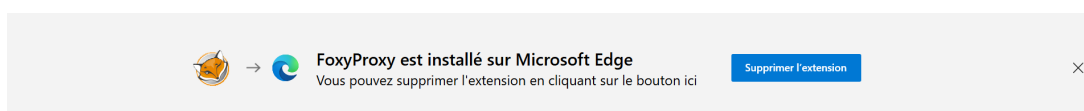


FIGURE 33 – foxyproxy

- La configuration de foxy proxy on doit ajouté le Hostname : 127.0.0.1  
numéro de port : 1080  
LE TYPE : SOCKS5  
Nom : SOCK\_SARAH

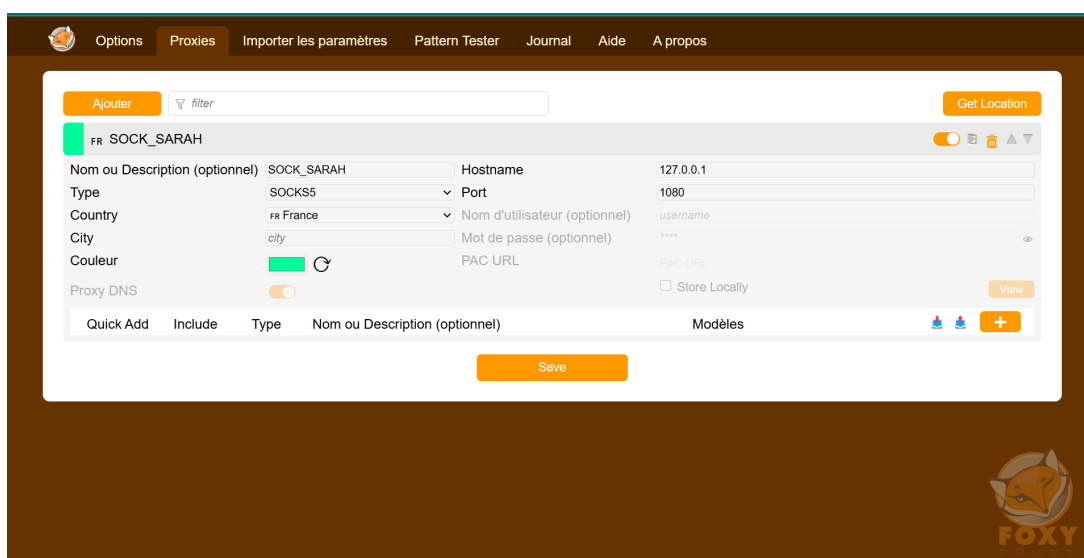


FIGURE 34 – Configuration de foxy proxy

- active le foxy proxy :

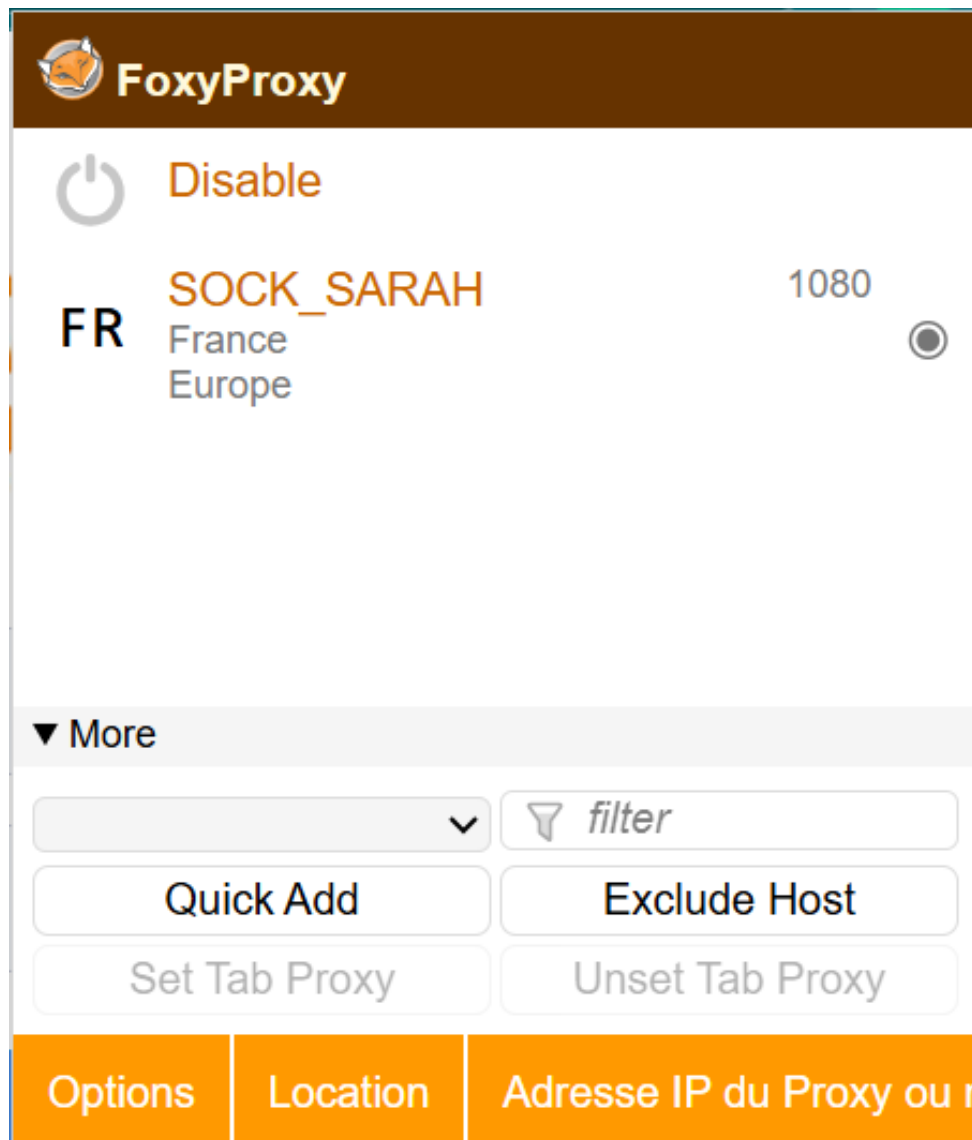


FIGURE 35 – Activé Le FoxyProxy

- Connectez-vous à l'interface webUI : <http://172.18.0.2:30500>

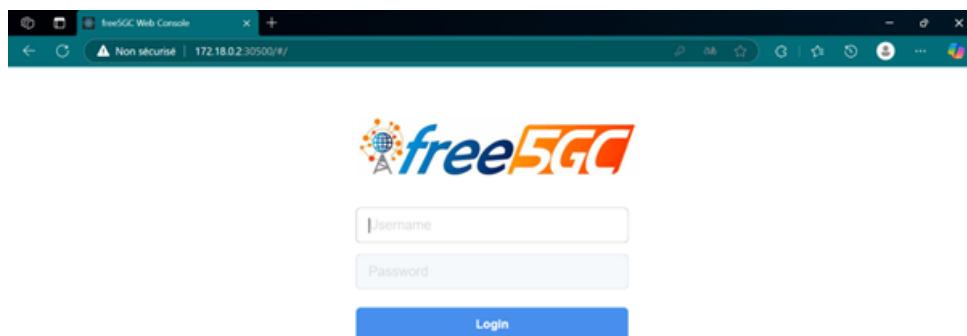


FIGURE 36 – connexion à l'interface webUI

### Connexion admin/free5gc

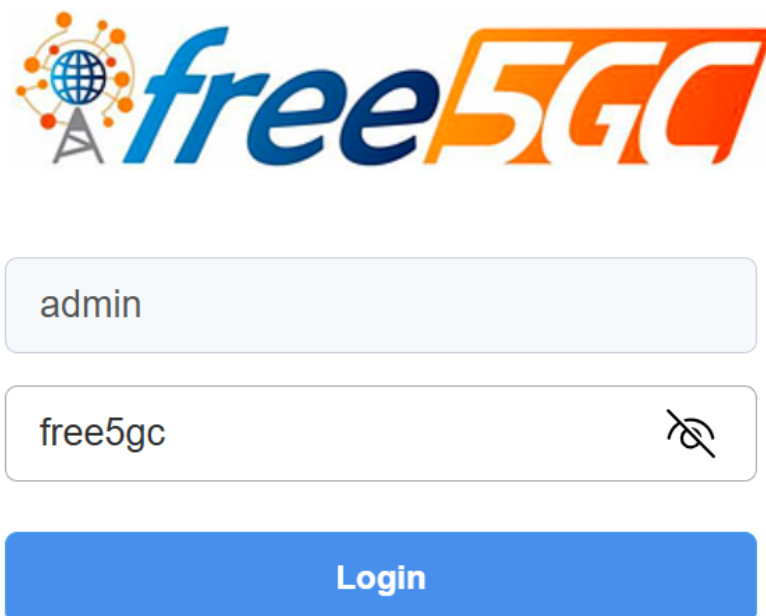


FIGURE 37 – connexion à l'interface webUI

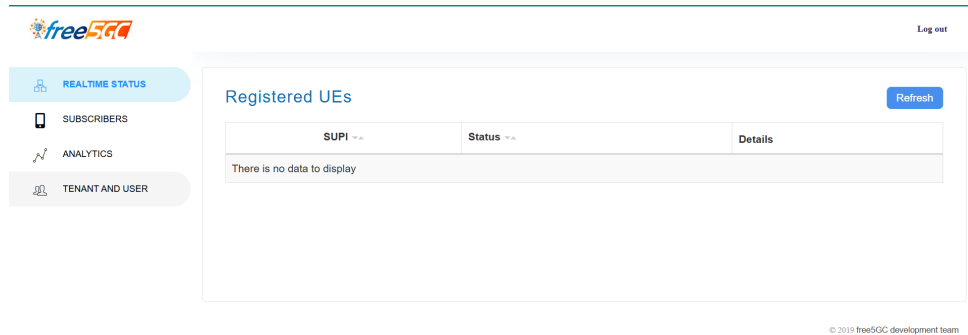


FIGURE 38 – connexion à l’interface webUI

**- On Peut ajouter un abonné par défaut via l'interface utilisateur**

New Subscriber

Subscriber data number (auto-increased with SUPI)\*

1

PLMN ID\*

20893

SUPI (IMSI)\*

208930000000003

MSISDN

Authentication Method\*

5G\_AKA

K\*

8baf473f2f8fd09487cccbd7097c6862

Operator Code Type\*

OPc

Operator Code Value\*

FIGURE 39 – connexion à l'interface webUI

## 4.2 Conclusion

Ce TP nous a permis de mettre en œuvre un réseau 5G Core en utilisant des technologies modernes telles que Kubernetes, Docker, et des outils associés comme kind, kubectl, et Helm. À travers les différentes étapes, nous avons approfondi notre compréhension des concepts suivants :



**Virtualisation des fonctions réseau (NFV) :**

La création d'un cluster Kubernetes pour héberger des fonctions réseau montre la transition des réseaux traditionnels vers des architectures cloud-native. Ce paradigme offre des avantages tels que la modularité et la flexibilité.

**Configuration des CNI (Container Network Interfaces) :**

L'installation et la gestion des plugins de connectivité, comme Multus CNI, ont illustré l'importance des réseaux conteneurisés pour assurer la communication entre les composants du réseau 5G.

**Déploiement et gestion d'applications distribuées :**

Le déploiement de Free5GC a démontré les avantages des outils comme Helm pour automatiser et simplifier la gestion des applications conteneurisées.

**Défis techniques rencontrés :**

L'installation de Docker, la configuration des fichiers YAML, et la gestion des pods ont mis en évidence les aspects pratiques et les problématiques courantes des environnements virtualisés.