



MASTER 2 Réseaux et Systèmes Autonomes

RAPPORT TP : UPC RESEAU 5G

Systèmes ubiquitaires (Edge, Véhiculer et UAV)

Étudiants :

SEBIHI Ahmed Nidhar -22419743-

Encadrant :

willem berroubache

LAST UPDATE 15/12/2024

Table Des Matières

1. Introduction	2
2. Prerequisites.....	2
2.1 Prérequis Matériels	2
2.1.1 Setup la machine virtuelle.....	2
2.2 Prérequis Softwar	6
2.2.1 Installation du module GTP5G	6
2.2.2 Installation de Kind et Creation de cluser	6
2.2.3 Installation de Docker	8
2.2.4 Relance la création du cluster avec Kind.....	9
2.2.5 Instalation des plugins CNI.....	9
2.2.6 Configuration des plugins CNI pour les conteneurs KinD	10
2.2.7 Installation de kubectl	10
2.2.8 Installation de Multus CNI.....	11
2.2.9 Installer Helm via Snap	12
2.2.10 Téléchargement de le chart Helm de Free5GC.....	12
3. Configuration des Charts Helm et et Vérification Réseau dans le Conteneur Worker	13
3.1 Vérifiez les informations réseau dans le conteneur :	13
3.2 Configurer la Valeur pour le Chart Master Helm.....	14
3.3 Configurer la Valeur pour le Chart UPF	14
3.4 Création d'un Volume Persistant	15
3.5 Création d'un dossier sur le nœud worker et application du volume persistant	16
4. Deploying Free5GC.....	17
4.1 Déploiement de Free5GC dans un Cluster Kubernetes	17
Étape 1 : Création du Namespace.....	17
Étape 2 : Déploiement avec Helm	17
Étape 3 : Vérification du Déploiement	18
4.2 Configuration de l'Adresse IP Externe du Cluster pour l'Accès à la WebUI	19
4.2.1 Identification de l'Adresse IP du Nœud du Cluster	19
4.2.2 Configuration d'un Proxy SOCKS avec FoxyProxy	19
4.2.3 Connexion à la WebUI de Free5GC	20
4.2.4 Ajout d'Abonnés via l'Interface WebUI	21
4.3 Déploiement et Tests de Connectivité avec UERANSIM.....	21

1. Introduction

Ce travail pratique (TP) explore l'installation et la configuration de Free5GC sur un cluster KinD hébergé dans une machine virtuelle Linux. L'objectif est de mettre en place les composants réseau essentiels d'un système 5G, en suivant les étapes de déploiement, y compris l'installation des modules CNI et Multus, la création d'un cluster KinD, et la configuration des services Free5GC. Les participants découvriront également comment tester la connectivité à l'aide de UERANSIM afin de vérifier le bon fonctionnement du réseau 5G déployé. Dans les prochaines étapes, vous trouverez les détails complets de chaque phase de mise en œuvre.

2. Prerequisites

2.1 Prérequis Matériels

Côté matériel, j'ai utilisé une machine virtuelle avec les spécifications suivantes : 9 Go de RAM, 45 Go de disque, et Ubuntu 22.04 LTS sans interface graphique comme système d'exploitation. Concernant le processeur de la machine hôte, il est important de noter que ce TP ne fonctionne pas sur une architecture ARM64, comme celle des processeurs Apple Silicon (M1, M2). Pour vérifier l'architecture de votre processeur, vous pouvez utiliser la commande **uname -m** sur un Mac : si le processeur est compatible, cette commande affichera **x86_64**

2.1.1 Setup la machine virtuelle

Sur le bouton **Nouvelle machine** en bleu, cliquez pour accéder à l'interface **Créer une machine virtuelle**. Dans cette interface, vous devez ajouter le chemin du fichier ISO. Si vous souhaitez que VirtualBox effectue automatiquement l'installation d'Ubuntu, vous pouvez sélectionner un nom d'utilisateur et un mot de passe pour votre machine. Ensuite, dans la section matérielle, j'ai choisi une configuration avec 9 Go de RAM et 4 processeurs CPU. Dans la dernière étape concernant le disque dur, j'ai opté pour une taille de 45 Go. Enfin, cliquez sur le bouton **Terminer** et patientez quelques minutes jusqu'à ce que votre machine virtuelle soit prête à être utilisée.

Dans les 4 figures suivantes, vous trouverez les 4 configurations nécessaires à effectuer pour lancer l'installation de la machine virtuelle (VM):

Étape 1 :

▼ Name and Operating System

Nom :

Folder:

ISO Image:

Edition:

Type:

Subtype:

Version:

☐ Skip Unattended Installation

> Unattended Install

> Hardware

> Hard Disk

Figure 1

Étape 2 :

> Name and Operating System

▼ Unattended Install

Username and Password

Username:

Password:

Repeat Password:

Additional Options

Product Key:

Hostname:

Domain Name:

☐ Install in Background

☐ Guest Additions

Guest Additions ISO:

> Hardware

> Hard Disk

Figure 2

Étape 3 :

> Name and Operating System

> Unattended Install

▼ Hardware

Mémoire vive :

4 Mo 16384 Mo

Processors:

CPU 1 CPUs 8

☐ Enable EFI (special OSes only)

> Hard Disk

Figure 3

Étape 4 :

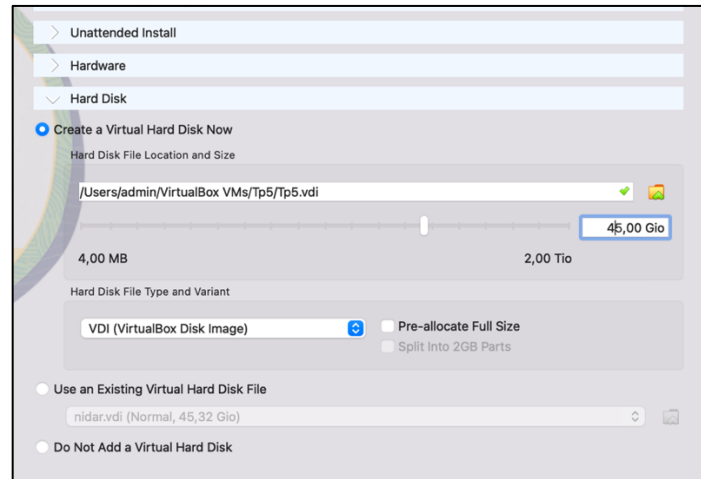


Figure 4

Utilisation de SSH :

Pour une meilleure utilisation et interaction avec votre machine virtuelle depuis le terminal de votre machine hôte, il est recommandé d'installer SSH. Pour cela, commencez par installer le serveur SSH sur votre VM avec les commandes suivantes :

- `sudo apt update`
- `sudo apt install -y openssh-server`

Ensuite, vérifiez le statut du service SSH à l'aide de la commande suivante :

- `sudo systemctl status ssh`

```
nidar@nidar:~$ sudo systemctl status sshd
sudo: systemctl: command not found
nidar@nidar:~$ sudo systemctl status sshd
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: en
   Active: active (running) since Mon 2024-12-09 21:38:18 UTC; 3min 43s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Main PID: 1370 (sshd)
     Tasks: 1 (limit: 10284)
    Memory: 1.8M
       CPU: 26ms
   CGroup: /system.slice/ssh.service
           └─1370 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

Figure 5

Après l'installation de SSH, il faut configurer les paramètres réseau nécessaires dans VirtualBox. Accédez à la section : Configuration → réseau, puis dans l'onglet Adaptateur NAT , cliquez sur Redirection de ports et ajoutez les paramètres suivants :

- Protocole : SSH
- IP hôte : 127.0.0.1 (adresse locale de votre machine hôte)
- Port hôte : 2222 (vous pouvez choisir un autre port si nécessaire)
- IP invité : 10.0.2.15 (c'est l'adresse IP de votre machine virtuelle ; vous pouvez la trouver en exécutant la commande "ip a" dans le terminal de la VM)
- Port invité : 22

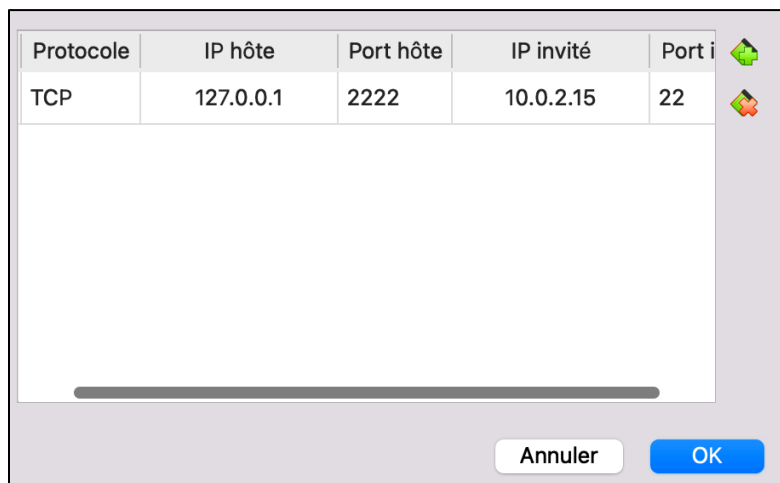


Figure 6

Une fois ces étapes terminées, votre machine virtuelle sera prête à être utilisée avec SSH pour une interaction fluide depuis votre terminal hôte.

En utilisant la commande : `ssh nidar@127.0.0.1 -p 2222`

Explication :

1. **ssh** : Se connecter via le protocole SSH.
2. **nidar** : Nom d'utilisateur sur la machine distante.
3. **127.0.0.1** : Adresse IP locale (machine hôte).
4. **-p 2222** : Utilise le port **2222** pour la connexion (redirigé vers le port 22 de la VM).

2.2 Prérequis Softwar

Avant de procéder à l'installation, il est essentiel de passer en mode **root** avec la commande suivante :

- `sudo su`

2.2.1 Installation du module GTP5G

Le module **GTP5G** est indispensable pour gérer le protocole GTP (GPRS Tunneling Protocol), utilisé dans les réseaux 5G. Pour l'installer correctement, suivez ces étapes :

1. **Installer les outils nécessaires :**
 - `apt install -y make gcc unzip`
2. **Télécharger le code source du module GTP5G :**
 - `curl -LO https://github.com/free5gc/gtp5g/archive/refs/tags/v0.8.10.zip`
3. **Décompresser l'archive et accéder au dossier :**
 - `unzip v0.8.10 && cd gtp5g-0.8.10`
4. **Installer le compilateur GCC (si ce n'est pas déjà fait) :**
 - `apt install gcc`
5. **Nettoyer les fichiers de compilation précédents et compiler le module :**
 - `make clean && make`
6. **Installer le module compilé :**
 - `sudo make install`

Avec ces étapes, le module **GTP5G** sera correctement installé et prêt à être utilisé dans votre configuration réseau 5G.

```
root@nidar:/home/nidar/gtp5g-0.8.10# sudo make install
cp gtp5g.ko /lib/modules/5.15.0-126-generic/kernel/drivers/net
modprobe udp_tunnel
/sbin/depmod -a
modprobe gtp5g
echo "gtp5g" > /etc/modules-load.d/gtp5g.conf
```

Figure 7

2.2.2 Installation de Kind et Creation de cluser

Pour l'installation de Kind, j'ai suivi les étapes indiquées sur le site officiel : [Kind Quick Start Guide](#). Voici les commandes que j'ai exécutées :

1. **Téléchargement de Kind :**
 - `curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64`
2. **Rendre le fichier exécutable :**
 - `chmod +x ./kind`
3. **Déplacer Kind vers un répertoire accessible globalement :**
 - `sudo mv ./kind /usr/local/bin/kind`
4. **Vérifier l'installation :**
 - `kind --version`

```
root@nidar:/home/nidar/gtp5g-0.8.10# kind --version
kind version 0.20.0
```

Figure 8

2.2.3.1 Configurer le fichier de cluster :

J'ai créé un fichier de configuration YAML pour définir la configuration du cluster. Voici le fichier `mycluster.yaml` contenant un nœud pour le plan de contrôle (*control-plane*) et un nœud worker :

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
```

Cette configuration définit un cluster **Kind** avec deux nœuds :

- Un nœud **control-plane** pour gérer le cluster.
- Un nœud **worker** pour exécuter les charges de travail.

Maintenant le fichier `mycluster.yaml` est prêt à être utilisé pour créer le cluster.

2.2.3.1 Créer le cluster avec Kind

J'ai utilisé la commande suivante pour créer le cluster à partir du fichier de configuration :

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

Ce que fait cette commande :

- Elle prépare les nœuds Docker.
- Elle crée un plan de contrôle Kubernetes et des nœuds workers.

- Elle configure le réseau et le stockage nécessaires.

ERROR : Cependant, après avoir exécuté cette commande, j'ai rencontré une erreur qui m'a indiqué que Docker devait être installé au préalable. Cela m'a fait comprendre que Kind dépend de Docker pour créer et gérer les nœuds du cluster.

Comme expliqué plus haut, cette commande prépare les nœuds Docker, mais elle nécessite que Docker soit déjà installé et opérationnel sur la machine. J'ai donc dû installer Docker avant de relancer cette commande.

```
root@nidar:/home/nidar/gtp5g-0.8.10# sudo kind create cluster --config mycluster.yaml
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

2.2.3 Installation de Docker

Pour l'installation de docker j'ai fait ces étapes :

1. **Installer les dépendances nécessaires :**
 - `sudo apt install -y ca-certificates curl gnupg lsb-release`
2. **Ajouter la clé GPG officielle de Docker :**
 - `sudo mkdir -p /etc/apt/keyrings`
 - `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg`
3. **Ajouter le dépôt Docker au gestionnaire de paquets :**
 - `echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`
4. **Installer Docker :**
 - `sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin`
5. **Vérifier l'installation :**
 - `docker --version`

```
root@nidar:/home/nidar/gtp5g-0.8.10# docker --version
Docker version 27.4.0, build bde2b89
```

Figure 10

2.2.4 Relance la création du cluster avec Kind

Après l'installation réussie de Docker, j'ai réessayé de relancer la commande de création du cluster. Cette fois, le cluster a été créé avec succès, et je peux maintenant confirmer que tout fonctionne correctement.

```
root@nidar:/home/nidar/gtp5g-0.8.10# sudo kind create cluster --config mycluster.yaml
Creating cluster "kind" ...
  ✓ Ensuring node image (kindest/node:v1.27.3) 📦
  ✓ 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

Not sure what to do next? 😊 Check out https://kind.sigs.k8s.io/docs/user/quick-start/
root@nidar:/home/nidar/gtp5g-0.8.10#
```

Figure 11

2.2.5 Installation des plugins CNI

Les **plugins CNI** (Container Network Interface) gèrent la mise en réseau des conteneurs en attribuant des adresses IP, en assurant leur connectivité et en appliquant des règles d'isolation réseau dans des environnements comme Kubernetes. Et voici comment j'ai téléchargé :

1. Téléchargez la dernière version des plugins CNI depuis le dépôt GitHub :

- `curl -LO https://github.com/containernetworking/plugins/releases/download/v1.6.0/cni-plugins-linux-amd64-v1.6.0.tgz`

2. Extrayez les fichiers téléchargés :

- `tar -xvzf cni-plugins-linux-amd64-v1.6.0.tgz`

```
root@nidar:/home/nidar/gtp5g-0.8.10# ls
bandwidth          dhcp              gtp5g.ko
bridge             dummy            gtp5g.mod
cni-plugins-linux-amd64-v1.6.0.tgz  firewall         gtp5g.mod.c
root@nidar:/home/nidar/gtp5g-0.8.10#
```

Figure 12

2.2.6 Configuration des plugins CNI pour les conteneurs KinD

Listez les conteneurs Docker utilisés pour exécuter les nœuds du cluster KinD avec la commande suivante :

- `sudo docker ps`

Voici la sortie obtenue :

```
root@nidar:/home/nidar/gtp5g-0.8.10# sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
e1f5800ecc29	kindest/node:v1.27.3	"/usr/local/bin/entr..."	10 minutes ago	Up 10 minutes	
	kind-worker				
2f51d85c25d2	kindest/node:v1.27.3	"/usr/local/bin/entr..."	10 minutes ago	Up 10 minutes	127.0.0.1:41895->64
	43/tcp kind-control-plane				

Figure 13

Identifiez les conteneurs pour le **control-plane** et les **workers**.

Copier les fichiers CNI dans les conteneurs:

- `sudo docker cp . 2f51d85c25d2:/opt/cni/bin/`
- `sudo docker cp . e1f5800ecc29:/opt/cni/bin/`

```
root@nidar:/home/nidar/gtp5g-0.8.10# sudo docker cp . 2f51d85c25d2:/opt/cni/bin/
Successfully copied 173MB to 2f51d85c25d2:/opt/cni/bin/
root@nidar:/home/nidar/gtp5g-0.8.10# sudo docker cp . e1f5800ecc29:/opt/cni/bin/
Successfully copied 173MB to e1f5800ecc29:/opt/cni/bin/
```

Figure 14

2.2.7 Installation de kubectl

kubectl est l'outil en ligne de commande que j'utilise pour interagir avec un cluster Kubernetes. Il permet de déployer, gérer et vérifier l'état des ressources Kubernetes. Voici les étapes que j'ai suivies pour installer kubectl :

1. **Télécharger le binaire de kubectl :**
 - `curl -LO "https://dl.k8s.io/release/$(curl -L -https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"`
2. **Rendre le fichier exécutable :**
 - `chmod +x ./kubectl`
3. **Déplacer le fichier dans un répertoire accessible globalement :**

- `sudo mv ./kubectl /usr/local/bin/`
- 4. **Vérifier l'installation :**
- `kubectl version --client`

Après avoir suivi ces étapes, je peux maintenant utiliser kubectl pour gérer et interagir avec mon cluster Kubernetes.

```

root@nidar:/home/nidar/gtp5g-0.8.10# curl -LO "https://dl.k8s.io/release/$(curl -
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
100  138  100  138    0     0   452      0  --:--:-- --:--:-- --:--:--   452
100 54.6M  100 54.6M    0     0  9733k      0  0:00:05  0:00:05 --:--:-- 10.5M
root@nidar:/home/nidar/gtp5g-0.8.10# chmod +x ./kubectl
root@nidar:/home/nidar/gtp5g-0.8.10# sudo mv ./kubectl /usr/local/bin/
root@nidar:/home/nidar/gtp5g-0.8.10# kubectl version --client
Client Version: v1.32.0
Kustomize Version: v5.5.0

```

Figure 15

2.2.8 Installation de Multus CNI

Dans le cadre de mon TP, Multus CNI joue un rôle clé pour configurer le réseau du cluster KinD en permettant aux pods d'avoir plusieurs interfaces réseau. Cette fonctionnalité est essentielle pour intégrer correctement les composants de Free5GC, notamment pour gérer les différentes interfaces (comme N6) nécessaires au bon fonctionnement des réseaux 5G.

1. **Clonez le dépôt GitHub de Multus CNI :**
 - `git clone https://github.com/k8snetworkplumbingwg/multus-cni`
 - `cd multus-cni`
2. **Appliquez le fichier multus-daemonset-thick.yml :**
 - `cat ./deployments/multus-daemonset-thick.yml | sudo kubectl apply -f -`
3. **Vérifiez que les ressources ont été créées avec succès et Vérifiez que le DaemonSet pour Multus est en cours d'exécution :**
 - `kubectl get crd`
 - `get pods -n kube-system`

```

root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni# kubectl get crd
NAME                                     CREATED AT
network-attachment-definitions.k8s.cni.cncf.io  2024-12-14T00:48:54Z
root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni# kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-5d78c9869d-2jdrf           1/1     Running   0           24m
coredns-5d78c9869d-fh5tc           1/1     Running   0           24m
etcd-kind-control-plane             1/1     Running   0           24m
kindnet-q85lm                      1/1     Running   0           24m
kindnet-qrk2g                      1/1     Running   0           24m
kube-apiserver-kind-control-plane   1/1     Running   0           24m
kube-controller-manager-kind-control-plane 1/1     Running   0           24m
kube-multus-ds-gmlkg               0/1     Init:0/1   0           36s
kube-multus-ds-kgk82               0/1     Init:0/1   0           36s
kube-proxy-hqmg5                   1/1     Running   0           24m
kube-proxy-w8gq5                   1/1     Running   0           24m
kube-scheduler-kind-control-plane   1/1     Running   0           24m

```

Figure 16

Après un moment le statut changera **a running**

kube-system	kube-apiserver-kind-control-plane	1/1	Running	0
kube-system	kube-controller-manager-kind-control-plane	1/1	Running	0
kube-system	kube-multus-ds-gmlkg	1/1	Running	0
kube-system	kube-multus-ds-kgk82	1/1	Running	0

Figure 17

Après un moment le statut changera **a running**

2.2.9 Installer Helm via Snap

Pour pouvoir télécharger le chart de Free5GC, je dois d'abord installer Helm. J'ai utilisé la commande suivante :

- `snap install helm --classic`

2.2.10 Téléchargement de le chart Helm de Free5GC

Voici les étapes que j'ai réalisées pour télécharger le chart Helm de Free5GC :

1. **Cloner le dépôt Free5GC Helm Charts :**
 - `git clone https://github.com/free5gc/free5gc-helm.git`
 - `cd free5gc-helm`
2. **Ajouter le dépôt Helm de Towards5GS** (remarque : j'ai fait cette étape pour consulter les versions disponibles dans le dépôt et choisir la version dans le pull) :
 - `helm repo add towards5gs 'https://raw.githubusercontent.com/Orange-OpenSource/towards5gs-helm/main/repo/'`
3. **Lister les versions disponibles du chart Free5GC :**

- helm search repo towards5gs/free5gc --versions

```
root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni/free5gc-helm# helm repo add towards5gs 'https://raw.githubusercontent.com/towards5gs/free5gc-helm/main/repo/'
"towards5gs" has been added to your repositories
root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni/free5gc-helm# helm search repo towards5gs/free5gc --versions
```

NAME	CHART VERSION	APP VERSION	DESCRIPTION
towards5gs/free5gc	1.1.7	v3.3.0	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.1.6	v3.3.0	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.1.5	v3.2.1	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.1.4	v3.2.1	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.1.3	v3.2.0	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.1.2	v3.1.1	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.1.1	v3.1.1	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.0.2	v3.0.6	A Helm chart to deploy Free5gc
towards5gs/free5gc	1.0.0	v3.0.5	A Helm chart to deploy Free5gc
towards5gs/free5gc	0.1.2	v3.0.5	A Helm chart to deploy Free5gc
towards5gs/free5gc	0.1.1	v3.0.5	A Helm chart to deploy Free5gc
towards5gs/free5gc	0.1.0	3.0.4	A Helm chart to deploy Free5gc
towards5gs/free5gc-amf	0.2.7	v2.2.0	A Helm chart to deploy the Free5GC AMF

Figure 18

J'ai utilisé la commande suivante pour télécharger et extraire le chart Helm :

- helm pull towards5gs/free5gc --untar --version 1.1.7

J'ai choisi la version 1.1.7 car c'est la dernière version disponible.

3. Configuration des Charts Helm et et Vérification Réseau dans le Conteneur Worker

3.1 Vérifiez les informations réseau dans le conteneur :

Pour effectuer cette vérification, commencez par accéder au conteneur Docker correspondant au worker avec la commande :

- sudo docker exec -it e1f5800ecc29 /bin/bash

Ensuite, exécutez les commandes suivantes :

- ip a
- ip r

Cela permet de noter la configuration actuelle. Comme illustré dans la figure, l'interface eth0 est configurée avec le réseau 172.18.0.0/16.

```

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
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.3/16 brd 172.18.255.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fc00:f853:ccd:e793::3/64 scope global nodad
        valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe12:3/64 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 172.18.0.2 dev eth0
172.18.0.0/16 dev eth0 proto kernel scope link src 172.18.0.3

```

Figure 19

3.2 Configurer la Valeur pour le Chart Master Helm

J'ai accédé au fichier values.yaml pour Free5GC dans le répertoire free5gc-helm/charts et J'ai modifié ou ajouté les paramètres dans la section n6network pour configurer le réseau de l'interface N6, en utilisant les informations que j'ai collectées . Comme vous pouvez voir dans la figure :

```

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

```

Figure 20

3.3 Configurer la Valeur pour le Chart UPF

J'ai accédé au fichier values.yaml pour le composant UPF dans le répertoire : free5gc/charts /free5gc-upf/values.yaml

Après j'ai modifié la section n6if à la ligne 79 pour définir l'adresse IP spécifique de l'interface N6 du UPF.

```

n6if: # DN
  ipAddress: 172.18.0.22

```

Figure 21

Et aussi j'ai modifié ou ajouté les paramètres suivants sous la section `n6network` pour spécifier la configuration réseau de l'interface N6 :

```

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

```

Figure 22

3.4 Création d'un Volume Persistant

Un Persistent Volume est une ressource de stockage dans Kubernetes conçue pour préserver les données, même lorsque les pods sont recréés ou supprimés. Cette étape met en place un volume persistant dédié au projet Free5GC, garantissant le stockage durable des données essentielles.

Étapes détaillées :

Création d'un fichier de configuration pour le volume persistant nommé `volume.yaml` est créé dans le home et Contenu du fichier est :


```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-local-pv9
  labels:
    project: free5gc
spec:
  capacity:
    storage: 8Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  local:
    path: /home/kubedata
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - kind-control-plane

```

Figure 23

3.5 Création d'un dossier sur le nœud worker et application du volume persistant

Cette étape permet de lier le volume persistant au cluster Kubernetes, en s'assurant que les données nécessaires seront stockées dans le répertoire /home/kubedata du nœud worker.

1. Identifiez l'ID du conteneur Docker pour le nœud worker et Accédez au conteneur Docker correspondant au worker :
 - `sudo docker exec -it e1f580ecc29 /bin/bash`
2. Une fois dans le conteneur du worker, j'ai créé le dossier /home/kubedata avec :
 - `mkdir /home/kubedata`
3. Ensuite, j'ai appliqué la configuration définie dans le fichier volume.yaml à l'aide de la commande suivante :
 - `sudo kubectl apply -f volume.yaml`

```

root@nidar:/home# sudo kubectl apply -f volume.yaml
persistentvolume/example-local-pv9 created

```

Figure 24

4. Deploying Free5GC

4.1 Déploiement de Free5GC dans un Cluster Kubernetes

Étape 1 : Création du Namespace

Grace Cette commande j'ai crée un namespace nommé free5gc dans mon cluster Kubernetes. Cela garantit que toutes les ressources liées à Free5GC seront regroupées dans ce namespace, facilitant ainsi leur gestion et leur surveillance.

- `sudo kubectl create ns free5gc`

Étape 2 : Déploiement avec Helm

Dans le répertoire où les charts Helm pour Free5GC sont stockés. J'ai exécuter la commande suivante :

- `sudo helm -n free5gc install free5gc-premier ./free5gc/`

Cette commande me permet d'installer Free5GC dans mon cluster Kubernetes créé à l'aide de Helm. Elle déploie tous les composants nécessaires, tels que AMF, SMF, PCF, NRF, et bien d'autres.

```

root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni/free5gc-helm# sudo helm -n free5gc install free5gc-premier ./free5gc/
NAME: free5gc-premier
LAST DEPLOYED: Sat Dec 14 02:03:22 2024
NAMESPACE: free5gc
STATUS: deployed
REVISION: 1
NOTES:
#
# 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: Abderaouf KHICHANE, Ilhem 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="

Release notes (What's changed in this version):
- Fix TLS configuration for SBI communications

```

Figure 25

Étape 3 : Vérification du Déploiement

Une fois l'installation terminée, il est nécessaire de vérifier que tous les pods de Free5GC fonctionnent correctement à l'aide de la commande suivante :

- `sudo kubectl get pods -n free5gc`

Voici la sortie obtenue : ce qui indique que l'installation a été réalisée avec succès et que Free5GC fonctionne correctement ou on peut voir à la fin le webui-service .

```

root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni/free5gc-helm# sudo kubectl get svc -n free5gc
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
free5gc-premier-free5gc-amf-service ClusterIP           10.96.99.88      <none>            80/TCP           6m18s
free5gc-premier-free5gc-ausf-service ClusterIP           10.96.25.147     <none>            80/TCP           6m18s
free5gc-premier-free5gc-nssf-service ClusterIP           10.96.26.62      <none>            80/TCP           6m18s
free5gc-premier-free5gc-pcf-service ClusterIP           10.96.91.39      <none>            80/TCP           6m18s
free5gc-premier-free5gc-smf-service ClusterIP           10.96.102.230    <none>            80/TCP           6m18s
free5gc-premier-free5gc-udm-service ClusterIP           10.96.167.118    <none>            80/TCP           6m18s
free5gc-premier-free5gc-udr-service ClusterIP           10.96.217.107    <none>            80/TCP           6m18s
mongodb                            ClusterIP           10.96.242.153    <none>            27017/TCP        6m18s
nrf-nnrf                           ClusterIP           10.96.12.65      <none>            8000/TCP         6m18s
webui-service                       NodePort            10.96.154.10     <none>            5000:30500/TCP   6m18s

```

Figure 26

4.2 Configuration de l'Adresse IP Externe du Cluster pour l'Accès à la WebUI

4.2.1 Identification de l'Adresse IP du Nœud du Cluster

Pour cela, j'ai utilisé la commande suivante :

- `kubectl get nodes -o wide`

Cette commande permet d'identifier les adresses internes et externes des nœuds. Voici la sortie obtenue. Comme on peut le voir, l'adresse **172.18.0.2** correspond à celle du **control-plane** et doit être utilisée pour accéder au service.

```
root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni/free5gc-helm# kubectl get nodes -o wide
```

NAME	KERNEL-VERSION	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE
kind-control-plane	5.15.0-126-generic	Ready	control-plane	107m	v1.27.3	172.18.0.2	<none>	Debian GNU/Linux 11 (bullseye)
kind-worker	5.15.0-126-generic	Ready	<none>	106m	v1.27.3	172.18.0.3	<none>	Debian GNU/Linux 11 (bullseye)

Figure 27

4.2.2 Configuration d'un Proxy SOCKS avec FoxyProxy

Tout d'abord, sur la machine hôte, j'ai téléchargé et ajouté l'extension FoxyProxy dans Firefox. Ensuite, j'ai ouvert l'extension, accédé à l'onglet **Proxies**, puis ajouté un nouveau proxy. Une fenêtre s'affiche, et il faut la remplir comme suit :

- **Nom** : Choisissez un nom pour identifier le proxy (par exemple, "Mon Proxy TP 5G").
- **Type** : Sélectionnez **SOCKS5**.
- **Hostname** : Entrez l'adresse locale de la machine, c'est-à-dire 127.0.0.1.
- **Port** : J'ai choisi 1080 comme port.

Après avoir rempli ces informations, cliquez sur **Save** pour enregistrer la configuration.

Ajouter filter

so

Nom ou Description (optionnel) so Hostname 127.0.0.1

Type SOCKS5 Port 1080

Country France Nom d'utilisateur (optionnel) username

City city Mot de passe (optionnel) ****

Couleur PAC URL

Proxy DNS Store Locally

Quick Add Include Type Nom ou Description (optionnel) Modèles

Save

Figure 28

4.2.3 Connexion à la WebUI de Free5GC

Maintenant, pour accéder à la WebUI en utilisant l'adresse et le port NodePort du service webui-service via le navigateur.

Étapes :

1. **Activation du proxy :**
 - Activez le proxy en cliquant sur le nom de votre proxy créé dans l'extension FoxyProxy.
2. **Accéder à la WebUI :**
 - <http://127.18.0.2:30500>.

Problème rencontré :

Lorsque j'ai essayé d'accéder à la WebUI, cela n'a pas fonctionné. Le problème provenait de la configuration SSH. Pour résoudre cela, j'ai dû :

- Me déconnecter de la session SSH existante.
- Recréer un tunnel SSH avec l'option -D pour rediriger correctement le trafic via un proxy SOCKS.

Solution :

L'option -D configure un proxy SOCKS dynamique avec le port choisi dans la configuration de FoxyProxy. Dans mon cas, le port configuré était 1080. Voici la commande SSH que j'ai utilisée pour créer le tunnel :

- `ssh -D 1080 nidar@127.0.0.1 -p 2222`

Maintenant, j'ai relancé l'adresse de mon service WebUI : `http://127.18.0.2:30500`, et j'ai pu accéder à l'interface Web. Comme vous pouvez le voir dans la figure, je me suis connecté en utilisant le **nom d'utilisateur** : admin et le **mot de passe** : free5gc.

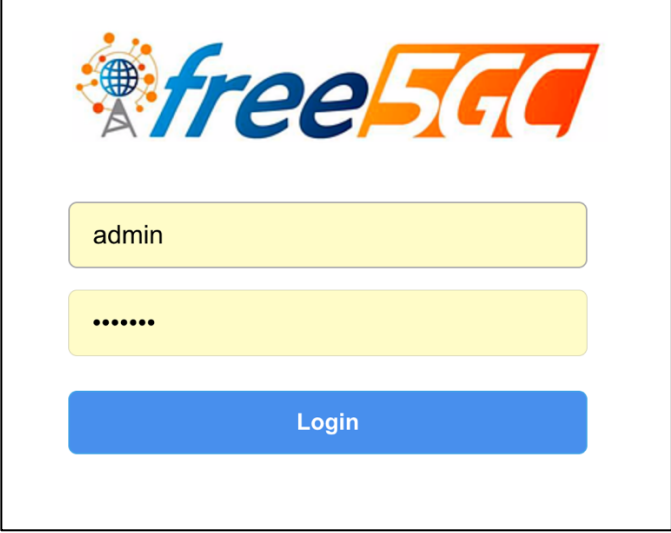
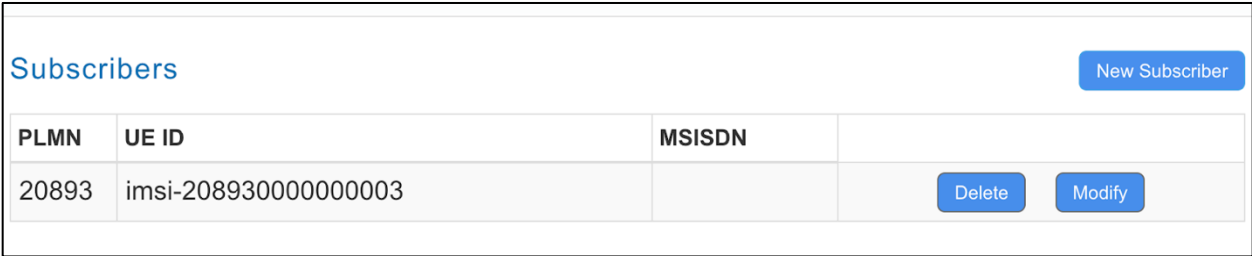


Figure 29

4.2.4 Ajout d'Abonnés via l'Interface WebUI

Maintenant, il est possible de configurer directement des abonnés par défaut ou personnalisés dans l'interface utilisateur.



PLMN	UE ID	MSISDN	
20893	imsi-208930000000003		<button>Delete</button> <button>Modify</button>

Figure 30

4.3 Déploiement et Tests de Connectivité avec UERANSIM

D'abord, il faut lancer la commande suivante depuis le répertoire contenant les charts Helm de UERANSIM.

- `helm -n free5gc install ueransim-premier ./ueransim/`

```

root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni/free5gc-helm/charts# sudo helm -n free5gc install ueransim-premier
/ueransim/
NAME: ueransim-premier
LAST DEPLOYED: Sat Dec 14 03:45:24 2024
NAMESPACE: free5gc
STATUS: deployed
REVISION: 1
NOTES:
#
# 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: Abderaouf KHICHANE, Ilhem FAJJARI
# Software description: An open-source project providing Helm charts to deploy 5G components (Core + RAN) on top o
Kubernetes
#
#
# Visit the project at https://github.com/Orange-OpenSource/towards5gs-helm
#
1. Run UE connectivity test by running these commands:
  helm --namespace free5gc test ueransim-premier

If you want to run connectivity tests manually, follow:

1. Get the UE Pod name by running:
  export POD_NAME=$(kubectl get pods --namespace free5gc -l "component=ue" -o jsonpath="{.items[0].metadata.name}")

2. Check that uesimtun0 interface has been created by running these commands:
  kubectl --namespace free5gc logs $POD_NAME
  kubectl --namespace free5gc exec -it $POD_NAME -- ip address

```

Figure 31

Ensuite, grâce à la commande suivante :

- `kubectl get pods -A`

J'ai pu visualiser le statut des pods, où l'on peut voir que **UERANSIM** est dans le statut **Running**.

```
root@nidar:/home/nidar/gtp5g-0.8.10/multus-cni/free5gc-helm/charts# kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
free5gc	free5gc-premier-free5gc-amf-amf-75b48f959d-v4f88	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-ausf-ausf-8694b78fff-6lm9d	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-dbpython-dbpython-7cd595f6b9-b9qcj	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-nrf-nrf-74579c5897-xtb2p	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-nssf-nssf-9987c5f49-99bhz	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-pcf-pcf-5778596ccf-qvn5k	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-smf-smf-64d76d858f-qbc4m	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-udm-udm-55975f967f-ktg2r	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-udr-udr-76976f6779-b5jb8	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-upf-upf-697d4c85b6-gspzs	1/1	Running	0	14h
free5gc	free5gc-premier-free5gc-webui-webui-5d9c5c9fdb-hn8f1	1/1	Running	0	14h
free5gc	mongodb-0	1/1	Running	0	14h
free5gc	ueransim-premier-gnb-86cf5bb4b-d6z8t	1/1	Running	0	12h
free5gc	ueransim-premier-ue-77d955dfb8-pvbmw	1/1	Running	0	12h
kube-system	coredns-5d78c9869d-2jdrf	1/1	Running	0	15h
kube-system	coredns-5d78c9869d-fh5tc	1/1	Running	0	15h
kube-system	etcd-kind-control-plane	1/1	Running	0	15h
kube-system	kindnet-q85lm	1/1	Running	0	15h
kube-system	kindnet-qrk2g	1/1	Running	0	15h
kube-system	kube-apiserver-kind-control-plane	1/1	Running	0	15h
kube-system	kube-controller-manager-kind-control-plane	1/1	Running	0	15h
kube-system	kube-multus-ds-gmlkg	1/1	Running	0	15h
kube-system	kube-multus-ds-kgk82	1/1	Running	0	15h
kube-system	kube-proxy-hqmg5	1/1	Running	0	15h
kube-system	kube-proxy-w8gq5	1/1	Running	0	15h
kube-system	kube-scheduler-kind-control-plane	1/1	Running	0	15h
local-path-storage	local-path-provisioner-6bc4bddd6b-fb7p5	1/1	Running	0	15h

Figure 32