



Ecole Supérieure Polytechnique de Dakar
Département génie informatique option réseaux et
télécommunications
Module : Réseaux mobiles



.....
.....

Rapport du projet intitulé ...

Conception d'un réseau 4G/5G déployé dans le Cloud

Rédigé par :

Professeur : DR. DIOUM

- Alioune Baldé
- Mohamadou Lamine Dioum
- Abbas Lamine Gueye

Table des matières

Introduction.....	4
I. Le réseau central : OPEN5GS.....	5
1. Architecture du réseau 5G NSA.....	6
a. Rôle de chaque entité	6
II. Le réseau cœur srsRAN	7
III. Technologies utilisé pour la virtualisation et la conteneurisations de notre réseaux 4G 5G	8
1. Docker	8
2. Kubernetes	9
3. Prometheus.....	10
IV. Conception du système	10
1. Conteneurisation de la partie accès srsRAN	10
2. Virtualisation de la partie cœur : OPEN5GS.....	11
a. Déploiement de open5gs avec Kubernetes.....	11
V. Test de fonctionnement de notre réseau.....	14
1. Lancement de notre partie Accès.....	14
2. Lancement de la partie cœur	14
CONCLUSION	15

Figure 1: les différents composants de notre système	4
Figure 2: Open5GS.....	5
Figure 3: architecture open5gs	6
Figure 4: srsRAN.....	7
Figure 5: DOCKER	8
Figure 6:Docker Status	9
Figure 7: KUBERNETES	9
Figure 8: PROMETHEUS.....	10
Figure 9:construction de l'image	11
Figure 10:verification de la disponiblité de l'image de srsran	11
Figure 11:vérification du statut du cluster.....	11
Figure 12: installation chart open5gs	12
Figure 13: visualisation du déploiement.....	12
Figure 14: Visualisation du déploiement.....	13

Introduction

L'avènement de la technologie 5G a révolutionné le monde de la connectivité en offrant une bande passante plus rapide, une latence plus faible et une capacité accrue de connecter un grand nombre d'appareils simultanément. Cependant, la mise en place d'un réseau 5G peut être complexe et coûteuse, ce qui a conduit de nombreuses entreprises à explorer des solutions plus économiques et flexibles, telles que l'utilisation de technologies de virtualisation et de conteneurisation.

Dans cette optique, la conception d'un réseau 5G déployé dans le cloud en utilisant Docker pour la conteneurisation et Kubernetes pour l'orchestration représente une solution prometteuse pour répondre aux besoins de connectivité évolués des entreprises. Dans ce type de déploiement, les fonctions réseau telles que la sécurité, l'optimisation de la bande passante, et la gestion des utilisateurs sont encapsulées dans des conteneurs Docker, puis orchestrées par Kubernetes.

L'objectif de ce projet consiste à mettre en place un réseau 4G 5G basé sur le cloud. Cette infrastructure est composée de parties virtualisées et de fonctionnalités conteneurisées. La figure 1 illustre les différents composants de cette conception.

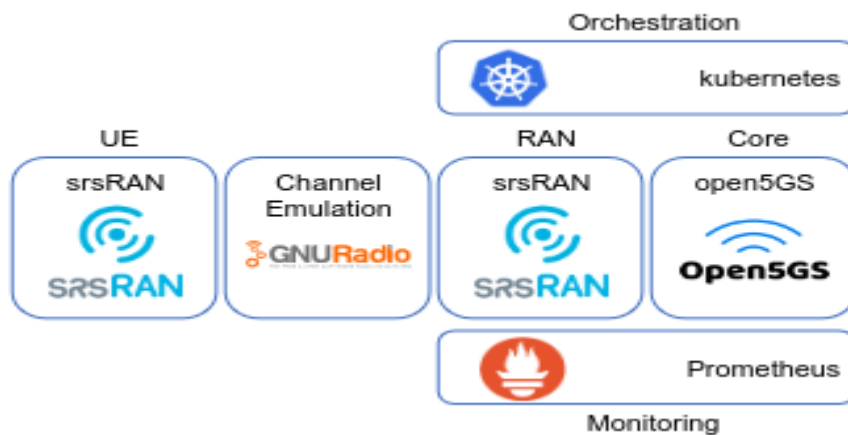


Figure 1: les différents composants de notre système

Pour orchestrer le déploiement et la gestion d'applications conteneurisées, nous avons opté pour Kubernetes, une plateforme open source. Pour le réseau central, nous avons choisi Open5Gs, qui s'adapte parfaitement au paradigme du cloud. Pour les éléments de l'UE et du réseau d'accès, nous utilisons srsRAN, un logiciel open source qui fournit une pile de réseau mobile complète. Afin de simplifier le déploiement et d'assurer sa répétabilité, nous avons créé un émulateur de canal simple qui permet de mettre en place un réseau entièrement programmable sans nécessiter de matériel onéreux. Enfin, nous avons choisi Prometheus pour surveiller l'état de l'ensemble du système.

I. Le réseau central : OPEN5GS



Figure 2: Open5GS

Open5GS est une implémentation de réseau mobile open source qui prend en charge les réseaux 5G et les versions précédentes, y compris 4G, 3G et 2G. Il s'agit d'un logiciel gratuit et open source sous licence AGPLv3, qui peut être utilisé, modifié et distribué librement. Il est conçu pour être déployé sur des infrastructures de cloud computing, telles que des serveurs virtuels, et peut fonctionner sur plusieurs plates-formes, notamment Linux, macOS et Windows. Il offre une gamme de fonctionnalités pour les réseaux mobiles, notamment la gestion des abonnés, l'authentification, l'autorisation et la comptabilité, ainsi que la gestion des appels et des sessions de données.

Open5GS est conçu pour être évolutif et flexible, ce qui le rend adapté aux cas d'utilisation de la 5G, tels que les réseaux privés 5G pour les entreprises, les villes intelligentes et les usines connectées. Il contient une série de composants logiciels et de fonctions réseau qui implémentent les fonctions principales 4G / 5G NSA et 5G SA. **Open5GS** contient une série de composants logiciels et de fonctions réseau qui mettent en oeuvre les entités du coeur du réseau **4G/5G NSA** et **5G SA**. Le coeur du coeur 4G/ 5G NSA Open5GS contient les composants suivants :

- **MME** – Mobility Management Entity (Entité de gestion de la mobilité)
- **HSS** - Home Subscriber Server (serveur d'abonné domestique)
- **PCRF** - Policy and Charging Rules Function (fonction de politique et de règles de tarification)
- **SGWC** - Serving Gateway Control Plane (Plan de contrôle de la passerelle de desserte)
- **SGWU** - Serving Gateway User Plane (plan d'utilisateur de la passerelle de desserte)
- **PGWC/SMF** - Packet Gateway Control Plane / (composant contenu dans Open5GS SMF)
- **PGWU/UPF** - Packet Gateway User Plane / (composant contenu dans Open5GS UPF)

Le coeur comporte deux plans principaux : le plan de contrôle et le plan utilisateur. Ils sont physiquement séparés dans Open5GS car **CUPS Control/User Plane Separation** (séparation des plans de contrôle et d'utilisateur) est mis en oeuvre. Le **MME** est le principal centre du **plan de contrôle** du coeur. Il gère principalement les sessions, la mobilité, la radiomessagerie et les supports. Il est relié au **HSS**, qui génère les vecteurs d'authentification SIM et détient le profil de l'abonné, ainsi qu'au **SGWC** et au **PGWC/SMF**, qui sont les plans de contrôle des serveurs passerelles. Tous les eNB du réseau mobile (stations de base 4G) se connectent au **MME**. Le dernier élément du plan de contrôle est le **PCRF**, qui se trouve entre le **PGWC/SMF** et le **HSS**, et qui gère la facturation et applique les politiques des abonnés.

Le **plan utilisateur** transporte les paquets de données de l'utilisateur entre l'eNB/gNB NSA (stations de base 5G NSA) et le réseau étendu externe. Les deux composants centraux du plan utilisateur sont le **SGWU** et le **PGWU/UPF**. Chacun d'entre eux se connecte à son homologue du plan de contrôle. Les eNB/gNB NSA se connectent à la **SGWU**, qui se connecte à la **PGWU/UPF**, puis au WAN. En séparant ainsi physiquement les plans de contrôle et d'utilisateur, vous pouvez déployer plusieurs serveurs de plan d'utilisateur sur le terrain (par exemple, dans un endroit disposant d'une connexion Internet à haut débit), tout en gardant la fonctionnalité de contrôle centralisée. Cela permet de prendre en charge les cas d'utilisation MEC (Multi-access Edge Computing), par exemple. Tous ces composants Open5GS ont des fichiers de configuration. Chaque fichier de configuration contient les adresses IP de liaison/les noms d'interface locale du composant et les adresses IP/noms DNS des autres composants auxquels il doit se connecter.

1. Architecture du réseau 5G NSA

Pour permettre la double connectivité, l'infrastructure 4G prendra en charge la connexion à une station de base **5G NR (gNodeB)**. La figure suivante montre un **gNodeB 5G** connecté à l'**EPC 4G** au niveau du plan de données. Le **gNodeB 5G** ne se connecte pas au **MME**. Le gNodeB se connecte à l'**eNodeB LTE** pour recevoir les demandes d'activation et de désactivation des supports **5G**.

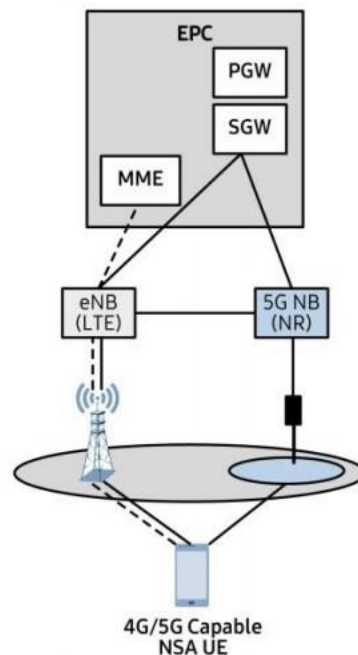


Figure 3: architecture open5gs

a. Rôle de chaque entité

L'UE peut se connecter à la station de base LTE et 5G NR. Une configuration de base pour un tel schéma est :

- L'UE se connecte au réseau LTE. L'UE signale au réseau qu'il peut se connecter simultanément aux réseaux 4G et 5G.
- Le réseau central vérifie si l'UE est autorisée à se connecter aux réseaux 4G et 5G. Le 4G **eNodeB** est informé que l'UE est autorisée à se connecter au réseau 5G.
- L'**eNodeB** prend alors la décision d'activer un support sur le gNodeB 5G.
- Le 4G **eNodeB** et le 5G gNodeB communique pour configurer le support sur le 5G gNodeB.
- L'UE est notifié du support 5G via le message de reconfiguration de la connexion RRC (Radio Resource Control).
- L'UE se connecte ensuite au réseau 5G tout en maintenant la connectivité au réseau 4G.

➤ **gNodeB**

Le 5G gNB agit comme un noeud secondaire dans une configuration à double connectivité. Le 5G gNB fournit le chemin des données 5G. Le gNodeB traite la demande d'ajout de porteur secondaire de l'eNodeB. Le gNodeB attribue des ressources radio 5G à la session et répond à l'eNodeB. Le gNodeB reçoit un message de reconfiguration complète de l'eNodeB. Ce message signale que l'UE a accepté la demande de reconfiguration qui ajoute une porteur 5G dans un mode de double connectivité. Le gNodeB gère ensuite la procédure d'accès aléatoire à partir de l'UE. Le flux de données commence sur les supports 5G NR après que l'eNodeB 4G a changé le chemin SGW.

➤ eNodeB

Le LTE eNB joue un rôle clé dans la configuration de la connexion. Une séquence d'actions simplifiée est répertoriée. L'eNodeB gère la configuration de la connexion 4G RRC à partir de l'UE. L'eNodeB reçoit le premier message NAS (Non Access Stratum) Attach et le transporte vers le MME (Mobility Management Entity) dans le message UE initiale. L'eNodeB agit comme un conduit pour la signalisation NAS entre l'UE et le MME pour la procédure d'authentification et de sécurité NAS. Le MME envoie ensuite la demande de configuration initiale du contexte à l'eNodeB. L'eNodeB effectue la prise de contact de capacité UE avec l'UE (si le MME n'avait pas inclus les capacités UE dans la demande de configuration initiale du contexte). L'eNodeB exécute ensuite la procédure de sécurité AS et le support par défaut 4G. L'UE envoie un rapport de mesure qui signale une bonne qualité de signal pour les supports 5G. L'eNodeB envoie la demande d'ajout SgNodeB au gNodeB 5G demandant un support 5G. L'eNodeB reçoit l'accusé de réception de demande d'ajout SgNodeB qui porte la configuration de support NR qui est signalée à l'UE via le message de reconfiguration de connexion LTE RRC. L'eNodeB envoie le message SgNodeB Reconfiguration Complete au 5G gNodeB à la réception du message LTE RRC Connection Reconfiguration Complete. Ce message porte le message NR RRC Reconfiguration Complete comme charge utile. L'eNodeB commence alors à copier les données vers le gNodeB. Il commute également le SGW afin qu'il puisse maintenant commencer directement à envoyer les données du support au 5G gNodeB.

➤ EPC

L'EPC vérifie si l'UE est autorisé pour un accès à double connectivité pendant la procédure d'attachement. Le réseau 4G EPC prend également en charge la commutation de supports entre les eNodeB 4G et les gNodeB 5G.

II. Le réseau cœur srsRAN



Figure 4: srsRAN

srsRAN est un logiciel open source de station de base 5G et 4G, qui permet aux utilisateurs de créer leur propre réseau mobile en utilisant des équipements basés sur des cartes SDR (Software Defined Radio). Il prend en charge une variété de protocoles de communication, y compris la 5G, la 4G, la 3G et la 2G, ainsi que les normes de réseau sans fil, telles que Wi-Fi.

Il est conçu pour être modulaire et évolutif, ce qui permet aux utilisateurs d'ajouter des fonctionnalités et des protocoles supplémentaires en fonction de leurs besoins. Il prend en charge plusieurs plates-formes, notamment Linux, macOS et Windows, ainsi que les processeurs ARM et x86.

srsRAN est souvent utilisé dans les tests et la recherche sur les réseaux 5G, en particulier pour les expériences en environnement contrôlé, car il offre un contrôle complet sur la configuration du réseau et de la station de base. Les constituants de srsRan sont

- **Le RAN (Radio Access Network)** : C'est le composant de srsRAN qui est responsable de la communication radio entre les terminaux mobiles et les stations de base. Il est composé de plusieurs éléments, notamment le eNodeB pour les réseaux 4G et le gNodeB pour les réseaux 5G.
- **Le Core Network** : C'est la partie centrale du réseau qui est responsable de la gestion de la signalisation, de l'authentification et de l'acheminement des données. Le cœur de srsRAN utilise le protocole S1AP pour la signalisation entre le RAN et le Core Network.
- **La Base Station** : La base station est l'équipement qui gère la communication radio entre les terminaux mobiles et le réseau. Elle est constituée de l'équipement radio et du contrôleur radio qui gère l'ensemble des fonctions radio.
- **L'UE (User Equipment)** : C'est le terminal mobile (smartphone, tablette, etc.) qui se connecte au réseau pour accéder à des services de communication mobile.
- **Le MME (Mobility Management Entity)** : Il s'agit d'un élément du réseau qui gère la mobilité des terminaux mobiles, notamment la gestion de la signalisation pour l'attachement et le détachement des terminaux mobiles.
- **L'HSS (Home Subscriber Server)** : C'est une base de données qui contient des informations sur les abonnés, notamment les profils des utilisateurs et les informations d'authentification.

III. Technologies utilisé pour la virtualisation et la conteneurisations de notre réseaux 4G 5G

1. Docker

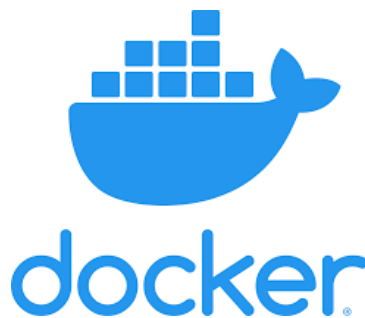


Figure 5: DOCKER

Docker est une plateforme open-source qui permet de créer, de déployer et de gérer des applications dans des conteneurs. Un conteneur Docker est une unité de logiciel autonome qui contient tous les éléments nécessaires pour exécuter une application, y compris le code, les dépendances et les bibliothèques. Les conteneurs Docker sont légers, portables et peuvent être exécutés sur n'importe quel système d'exploitation, ce qui permet de réduire les coûts de développement et de déploiement d'applications. Docker fournit également un écosystème riche en outils pour faciliter la création, la gestion et la mise à l'échelle de conteneurs.

Pour l'installation de **Docker** sur un système UBUNTU on peut suivre les instructions suivantes :

En premier lieu on fait une mise à jour des packages ensuite faire l'installation des dépendances nécessaire. Puis faire l'ajout de la clé GPG de Docker et du référentiel Docker et de remettre à jour les paquets. On installe Docker et le démarrer, faire l'ajout de l'utilisateur actuel au groupe Docker


```
root@balde:~/projetDIOUM# service docker status
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2023-03-26 16:04:16 GMT; 16h ago
     Docs: https://docs.docker.com
    Main PID: 1112 (dockerd)
      Tasks: 17
     CGroup: /system.slice/docker.service
             └─1112 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

mar 27 08:14:24 balde dockerd[1112]: time="2023-03-27T08:14:24.798692734Z" level=info msg="N
mar 27 08:14:24 balde dockerd[1112]: time="2023-03-27T08:14:24.798741868Z" level=info msg="I
mar 27 08:14:26 balde dockerd[1112]: time="2023-03-27T08:14:26.894418144Z" level=info msg="N
mar 27 08:14:26 balde dockerd[1112]: time="2023-03-27T08:14:26.894483624Z" level=info msg="I
mar 27 08:16:23 balde dockerd[1112]: time="2023-03-27T08:16:23.133563653Z" level=info msg="C
mar 27 08:16:23 balde dockerd[1112]: time="2023-03-27T08:16:23.133697216Z" level=info msg="C
mar 27 08:16:23 balde dockerd[1112]: time="2023-03-27T08:16:23.269034737Z" level=info msg="i
mar 27 08:16:23 balde dockerd[1112]: time="2023-03-27T08:16:23.278568925Z" level=info msg="i
mar 27 08:16:33 balde dockerd[1112]: time="2023-03-27T08:16:33.799412273Z" level=info msg="C
mar 27 08:16:33 balde dockerd[1112]: time="2023-03-27T08:16:33.963954609Z" level=info msg="i
lines 1-19/19 (END)
```

Figure 6: Docker Status

2. Kubernetes



Figure 7: KUBERNETES

Kubernetes est un système open-source de gestion de conteneurs qui permet de déployer, de mettre à l'échelle et de gérer des applications conteneurisées. Il offre une automatisation de la gestion de la configuration, du déploiement et de la mise à l'échelle des conteneurs. Il peut fonctionner sur des clusters de machines physiques ou virtuelles, et il permet de gérer de manière efficace des applications distribuées et des charges de travail à grande échelle.

Pour installer Kubernetes on peut suivre les instructions suivantes :

- On procède d'abord à l'installation de kubeadm, kubectl et kubelet :

```
sudo apt-get update && sudo apt-get install -y apt-transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
```

Ensuite on fait l'initialisation du cluster : `sudo kubeadm init`

- Déploiement du réseau : `kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml`
- Vérification : `kubectl get nodes`

3. Prometheus



Figure 8: PROMETHEUS

L'outil de monitoring Prometheus est un système open-source de surveillance et d'alerte pour les applications conteneurisées. Il permet de collecter des métriques à partir des conteneurs et de surveiller l'état de l'ensemble du système, y compris les conteneurs, les machines hôtes et les services. Il offre également des fonctionnalités d'alerte pour signaler les problèmes de manière proactive. Prometheus est conçu pour fonctionner avec Kubernetes et peut être intégré à d'autres outils de surveillance et d'alerte pour offrir une visibilité complète sur l'ensemble du système.

IV. Conception du système

1. Conteneurisation de la partie accès srsRAN

Principe : nous voulons créer une image qui va contenir toutes les dépendances nécessaires au fonctionnement de la partie accès et ensuite nous lançons l'image dans un conteneur.

En effet, ici on a le choix entre le fait de conteneuriser chaque entité de la partie accès (eNB, UE) et de les lancer dans différents conteneurs isolés dont il faut probablement gérer la mise en réseau de ces derniers avant de les lancer ou de regrouper tous les entités dans un même conteneur et de le déployer.

A travers un fichier Docker file (disponible en annexe) nous allons grâce à une image ubuntu 18.04 de base définir toutes les dépendances nécessaires au bon fonctionnement de notre partie accès srsRAN et construire son image. Ensuite on lance les conteneurs grâce à « docker compose » (on créera éventuellement un fichier docker-compose.yml ou on va définir comment sera lancé chaque service et si des services dépendent d'autres pour fonctionner etc....).

```

root@balde:~/projetDIOUM# docker build -t srsran-image .
[+] Building 1457.4s (13/13) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 823B                                              0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/ubuntu:18.04                  1.2s
=> [1/9] FROM docker.io/library/ubuntu:18.04@sha256:8aa9c2798215f99544d1ce7439ea9c3a6dfd82de607da1cec3a8a2fae005931b  0.0s
=> CACHED [2/9] RUN apt update                                                  0.0s
=> CACHED [3/9] RUN apt install -y cmake libbftw3-dev libmbedtls-dev libboost-program-options-dev libconfig++-dev libsctp-dev libtool  0.0s
=> CACHED [4/9] RUN git clone https://github.com/zeromq/libzmq.git && cd libzmq && ./autogen.sh && ./configure && make && make install  0.0s
=> CACHED [5/9] RUN git clone https://github.com/zeromq/czmq.git                0.0s
=> CACHED [6/9] RUN cd czmq && ./autogen.sh && ./configure && make && make install && ldconfig  0.0s
=> [7/9] RUN cd                                                                0.4s
=> [8/9] RUN git clone https://github.com/srsRAN/srsRAN.git                    53.4s
=> [9/9] RUN cd srsRAN && mkdir build && cd build && cmake ../ && make          1397.5s
=> exporting to image                                                            4.7s
=> => exporting layers                                                            4.7s
=> => writing image sha256:044836eeb7f558afc0608817fb049f018bc3be6c17cd401a23dfefb4a3a6a710  0.0s
=> => naming to docker.io/library/srsran-image                                0.0s
root@balde:~/projetDIOUM#

```

Figure 9:construction de l'image

```

root@balde:~/projetDIOUM# docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
srsran-image        latest       044836eeb7f5     3 hours ago     1.53GB
<none>              <none>       9a8a6198785e     3 hours ago     888MB
open5gs-image       latest      a9519cb45504     4 hours ago     1.23GB
hello-world         latest      feb5d9fea6a5     18 months ago   13.3kB
root@balde:~/projetDIOUM#

```

Figure 10:verification de la disponibilité de l'image de srsran

2. Virtualisation de la partie cœur : OPEN5GS

Ici nous avons le choix entre :

- Installer et configurer toutes les entités de open5gs localement ensuite on crée une image docker pour open5gs et après utilise cette image pour le déploiement sur kubernetes.
- Utiliser Helm et pour déployer rapidement notre réseau 5g sur kubernetes

Helm est un outil de gestion de packages pour Kubernetes. Il permet de simplifier le déploiement et la gestion d'applications Kubernetes en utilisant des packages appelés "charts".

a. Déploiement de open5gs avec Kubernetes

Pour un déploiement en local on va utiliser minikube qui est un outil open-source qui permet de déployer une instance locale de Kubernetes sur un ordinateur.

Après l'installation de minikube on crée un cluster kubernetes avec la commande 'minikube start'. Pour vérifier l'état du cluster créé on utilise la commande 'minikube status'.

```

alg@alg-ThinkPad:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

```

Figure 11:vérification du statut du cluster

Une fois le cluster démarré on va utiliser helm pour déployer facilement notre réseau 5g en utilisant la charte télécharger depuis le dépôt officiel de open5gs.

Le chart de Open5GS est utilisé pour déployer Open5GS sur un cluster Kubernetes à l'aide de Helm. Le chart contient des fichiers YAML qui décrivent les ressources Kubernetes nécessaires pour le déploiement, comme les pods, les services, les déploiements, les volumes, etc...

```
alg@alg-ThinkPad:~$ helm install open5gs openverso/open5gs
NAME: open5gs
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figure 12: installation chart open5gs

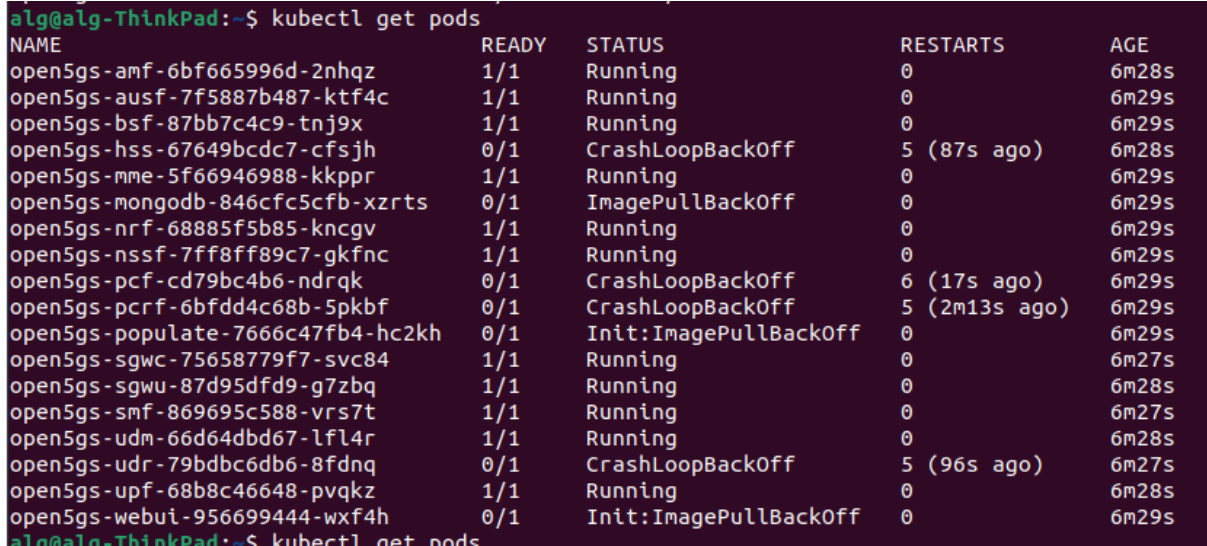
Après l'installation de notre chart pour déployer open5gs dans notre cluster kubernetes on vas utiliser la commande kubectl get pods pour voir le déploiement.

```
alg@alg-ThinkPad:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
open5gs-amf-6bf665996d-2nhqz	0/1	ContainerCreating	0	50s
open5gs-ausf-7f5887b487-ktf4c	0/1	Running	0	51s
open5gs-bsf-87bb7c4c9-tnj9x	0/1	ContainerCreating	0	51s
open5gs-hss-67649bcd7-cfsjh	0/1	ContainerCreating	0	50s
open5gs-mme-5f66946988-kkppr	1/1	Running	0	51s
open5gs-mongodb-846cfc5cfb-xzrts	0/1	ContainerCreating	0	51s
open5gs-nrf-68885f5b85-kncgv	0/1	ContainerCreating	0	51s
open5gs-nssf-7ff8ff89c7-gkfnc	0/1	ContainerCreating	0	51s
open5gs-pcf-cd79bc4b6-ndrpk	0/1	CrashLoopBackOff	2 (5s ago)	51s
open5gs-pcrf-6bfdd4c68b-5pkbf	1/1	Running	1 (10s ago)	51s
open5gs-populate-7666c47fb4-hc2kh	0/1	Init:0/1	0	51s
open5gs-sgwc-75658779f7-svc84	0/1	ContainerCreating	0	49s
open5gs-sgwu-87d95dfd9-g7zbq	0/1	ContainerCreating	0	50s
open5gs-smf-869695c588-vrs7t	0/1	ContainerCreating	0	49s
open5gs-udm-66d64dbd67-lfl4r	0/1	ContainerCreating	0	50s
open5gs-udr-79bdb6c6db6-8fdnq	0/1	ContainerCreating	0	49s
open5gs-upf-68b8c46648-pvqkz	0/1	Init:0/1	0	50s
open5gs-webui-956699444-wxf4h	0/1	Init:0/1	0	51s

Figure 13: visualisation du déploiement

On voit sur la figure que certains pods sont en train d'être créés et d'autres ont déjà été créés avec succès (ceux avec le status running).



NAME	READY	STATUS	RESTARTS	AGE
open5gs-amf-6bf665996d-2nhqz	1/1	Running	0	6m28s
open5gs-ausf-7f5887b487-ktf4c	1/1	Running	0	6m29s
open5gs-bsf-87bb7c4c9-tnj9x	1/1	Running	0	6m29s
open5gs-hss-67649bcd7-cfsjh	0/1	CrashLoopBackOff	5 (87s ago)	6m28s
open5gs-mme-5f66946988-kkppr	1/1	Running	0	6m29s
open5gs-mongodb-846cfc5cfb-xzrts	0/1	ImagePullBackOff	0	6m29s
open5gs-nrf-68885f5b85-kncgv	1/1	Running	0	6m29s
open5gs-nssf-7ff8ff89c7-gkfnc	1/1	Running	0	6m29s
open5gs-pcf-cd79bc4b6-ndrqk	0/1	CrashLoopBackOff	6 (17s ago)	6m29s
open5gs-pcrf-6bfd4c68b-5pkbf	0/1	CrashLoopBackOff	5 (2m13s ago)	6m29s
open5gs-populate-7666c47fb4-hc2kh	0/1	Init:ImagePullBackOff	0	6m29s
open5gs-sgw-75658779f7-svc84	1/1	Running	0	6m27s
open5gs-sgw-87d95dfd9-g7zbq	1/1	Running	0	6m28s
open5gs-smf-869695c588-vrs7t	1/1	Running	0	6m27s
open5gs-udm-66d64dbd67-lfl4r	1/1	Running	0	6m28s
open5gs-udr-79bdbc6db6-8fdnq	0/1	CrashLoopBackOff	5 (96s ago)	6m27s
open5gs-upf-68b8c46648-pvqkz	1/1	Running	0	6m28s
open5gs-webui-956699444-wxf4h	0/1	Init:ImagePullBackOff	0	6m29s

Figure 14: Visualisation du déploiement

6 minutes après la création du cluster on a fait un `kubectl get pods` pour voir si tous les pods ont été bien créés mais certains ne marchent pas.

On voit qu'il y a 7 entités qui ne démarrent pas correctement, pour résoudre ce problème on devra modifier les charts de ces pods et changer l'image de ces entités. Mais par faute de temps on n'a pas pu trouver et configurer correctement les images qu'il nous faut pour résoudre notre problème.

V. Test de fonctionnement de notre réseau

1. Lancement de notre partie Accès

- eNB :

```
root@balde:~/projetDI0UM/open5gs_srsRAN# cd .. && docker compose up
[+] Running 3/3
  :: Container virtual-srsenb   Recreated
  :: Container virtual-srsepc   Recreated
  :: Container virtual-srsue     Recreated
Attaching to virtual-srsenb, virtual-srsepc, virtual-srsue
virtual-srsenb | Active RF plugins: librsran_rf_zmq.so
virtual-srsenb | Inactive RF plugins:
virtual-srsenb | --- Software Radio Systems LTE eNodeB ---
virtual-srsenb |
virtual-srsenb | Couldn't open , trying /root/.config/srsran/enb.conf
virtual-srsenb | Couldn't open /root/.config/srsran/enb.conf either, trying /etc/srsran/enb.conf
virtual-srsenb | Reading configuration file /etc/srsran/enb.conf...
virtual-srsenb | Couldn't open sib.conf, trying /root/.config/srsran/sib.conf
virtual-srsenb | Couldn't open /root/.config/srsran/sib.conf either, trying /etc/srsran/sib.conf
virtual-srsenb | Couldn't open rr.conf, trying /root/.config/srsran/rr.conf
virtual-srsenb | Couldn't open /root/.config/srsran/rr.conf either, trying /etc/srsran/rr.conf
virtual-srsenb | Couldn't open rb.conf, trying /root/.config/srsran/rb.conf
virtual-srsenb | Couldn't open /root/.config/srsran/rb.conf either, trying /etc/srsran/rb.conf
```

- UE :

```
virtual-srsue | Active RF plugins: librsran_rf_zmq.so
virtual-srsue | Inactive RF plugins:
virtual-srsue | Couldn't open , trying /root/.config/srsran/ue.conf
virtual-srsue | Couldn't open /root/.config/srsran/ue.conf either, trying /etc/srsran/ue.conf
virtual-srsue | Reading configuration file /etc/srsran/ue.conf...
virtual-srsue |
virtual-srsue | Built in Release mode using commit 00c972ac4 on branch master.
virtual-srsue |
virtual-srsue | Opening 1 channels in RF device=zmq with args=tx_port=tcp://*:2001,rx_port=tcp://localhost:2000,id=ue,base_srate=23.04e6
virtual-srsue | Supported RF device list: zmq file
virtual-srsue | CHx base_srate=23.04e6
virtual-srsue | CHx id=ue
virtual-srsue | Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
virtual-srsue | CH0 rx_port=tcp://localhost:2000
virtual-srsue | CH0 tx_port=tcp://*:2001
```

2. Lancement de la partie cœur

```
alg@alg-ThinkPad:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
open5gs-amf-6bf665996d-2nhqz	0/1	ContainerCreating	0	50s
open5gs-ausf-7f5887b487-ktf4c	0/1	Running	0	51s
open5gs-bsf-87bb7c4c9-tnj9x	0/1	ContainerCreating	0	51s
open5gs-hss-67649bcd7-cfsjh	0/1	ContainerCreating	0	50s
open5gs-mme-5f66946988-kkppr	1/1	Running	0	51s
open5gs-mongodb-846cfc5cfb-xzrts	0/1	ContainerCreating	0	51s
open5gs-nrf-68885f5b85-kncgv	0/1	ContainerCreating	0	51s
open5gs-nssf-7ff8ff89c7-gkfnc	0/1	ContainerCreating	0	51s
open5gs-pcf-cd79bc4b6-ndrqk	0/1	CrashLoopBackOff	2 (5s ago)	51s
open5gs-pcrf-6bfdd4c68b-5pkbf	1/1	Running	1 (10s ago)	51s
open5gs-populate-7666c47fb4-hc2kh	0/1	Init:0/1	0	51s
open5gs-sgw-75658779f7-svc84	0/1	ContainerCreating	0	49s
open5gs-sgw-87d95dfd9-g7zbq	0/1	ContainerCreating	0	50s
open5gs-smf-869695c588-vrs7t	0/1	ContainerCreating	0	49s
open5gs-udm-66d64dbd67-lfl4r	0/1	ContainerCreating	0	50s
open5gs-udr-79bdb6c6db6-8fdnq	0/1	ContainerCreating	0	49s
open5gs-upf-68b8c46648-pvqkz	0/1	Init:0/1	0	50s
open5gs-webui-956699444-wxf4h	0/1	Init:0/1	0	51s

CONCLUSION

Dans ce projet, nous avons essayé de déployer un réseau 5G NSA qui se concentre sur le haut débit mobile amélioré plus concrètement un réseau 5G qui s'appuie sur l'infrastructure 4G existante afin de bénéficier de la couverture des bandes basses 4G, ainsi que de la connexion vers un réseau cœur 4G évolué, auquel on y ajoute les fonctionnalités nécessaires à la prise en compte du nouveau standard 5G. En effet, le travail effectué ici, nous a permis de comprendre ce qu'on a vu en cours tel que l'architecture des réseaux 4G 5G, la communication entre les différentes entités mais aussi de faire un pas vers les nouvelles technologies tels que la conteneurisation et la mise en cloud.

En perspective, nous comptons continuer la suite du projet afin d'atteindre les objectifs demandés.