الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière : Télécommunications

Spécialité : Réseaux et télécommunications

Présenté par

MENNA Safa

Cloud native pour les applications 5G-VNFs évolutifs basés sur des micro-sevices avec Kubernetes

Proposé par : Mlle. DIFALLAH Rania - Ingénieur exploitation à ATM Mobilis

Co-promoteur : Mr. AIT SAADI Hocine - Chef de département et professeur à l'université Saad DAHLAB Blida 1

Année Universitaire 2021-2022

Remerciements

Au terme de ce travail, et avec des larmes de joie, après les deux ans les plus difficiles de ma vie, Je remercie Dieu le tout puissant de m'avoir sauvé la vie, donné la santé, et le courage ; m'entourer avec des gens très gentils que je porterai dans mon cœur jusqu'à la fin de mes temps…

Je remercie infiniment Allah sobhanaho w taàla et les gens magnifiques qui ont fait de leur mieux pour m'aider à me remettre debout et reprendre mes études pour pouvoir réaliser ce modeste travail.

Mes chaleureux remerciements vont à mes chers parents qui ont toujours été là pour moi, et qui m'ont donné un magnifique modèle de labeur et de persévérance. J'espère qu'ils trouveront, dans ce travail, toute ma gratitude, ma reconnaissance et mon amour. Si je suis devenue qui je suis et je suis arrivée ici aujourd'hui, c'est grâce à vous !

Je remercie respectueusement mon Co-promoteur Monsieur AIT SAADI Hocine, pour sa disponibilité, et surtout pour ces aides précieuses en dépit de ses nombreuses occupations. Vraiment, je ne l'oublierai jamais !

Je remercie chaleureusement mon professeur Monsieur BERSALI Mahdi de m'avoir considéré comme sa fille, je le remercie pour tous ses conseils, ses encouragements, et pour tous led efforts qu'il a faits pour que je puisse arriver ici !

J'ai eu la chance et l'honneur de travailler avec une femme tellement gentille…

Je ne remercierai jamais assez ma promotrice, Madame DIFALLAH Rania pour toute sa gentillesse, et ses mots encourageants. Pour son aide, pour les conseils très précieux qu'elle m'a donnés. Elle m'a montré mon chemin comme la lumière dans le sombre. Je la remercie beaucoup d'avoir été à mes côtés dans les moments les plus difficiles.

Je remercie vivement les membres du jury d'avoir accepté de juger ce modeste travail.

Je suis reconnaissante à madame AMIROUCHE, madame BERKET, monsieur CHIKHI, monsieur BENDOUMIA, et monsieur MAHDI, ainsi qu'à tous mes enseignants qui ont contribué à ma formation et à toute personne qui m'appris une lettre ou une phrase, qu'ils trouvent ici mes reconnaissances et un petit fruit de leurs sueurs.

Mes vifs remerciements vont également à madame LOUANCHI Karima, madame Saida, et madame Fatima, ainsi qu'à toute l'équipe de ATM Mobilis pour leur accueil chaleureux et leur soutien tout au long de mon stage.

Enfin, Je tiens également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Dédicace

A ma chère maman

Aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que j'ai toujours eu pour toi. Rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien être.
Ce travail est le fruit des sacrifices que tu as consentis pour mon éducation.
Que Dieu t'accorde santé, bonheur, et t'accueille dans son éternel paradis.

A mon cher papa

Je croie que la meilleure façon de te montrer l'amour et l'affection que j'éprouve pour toi est de te dédier ce travail. Tu m'as comblé avec ta tendresse et ta protection tout au long de mon parcours.
Tout ton souci pour que je sois à la hauteur, le sérieux, l'endurance, et le travail dur que tu m'as appris, tu n'as cessé de me soutenir et de m'encourager, tu as toujours été présent à mes côtés quand j'en avais besoin.
En ce jour mémorable, pour moi ainsi que pour toi, reçoit ce travail en signe de ma vive reconnaissance et ma profonde estime.
Puisse Dieu t'accorde santé, bonheur et t'accueille dans son immense paradis.

A mon frère et ma petite sœur adorable

J'ai toujours fait de mon mieux pour être un bon exemple pour vous, et j'espère que je l'étais. Je vous dédie ce travail tout en vous souhaitant santé et réussite dans vos études et dans votre vie.

A mon médecin Dr.F CHELHA

Vous étiez la seule médecin qui a pu diagnostiquer ma maladie. Avec votre sens de devoir, vos compétences, votre sympathie avec moi, vos conseils pleins d'amour…
Vous avez été mon soutien, fidèle, quel que soit le jour et l'heure de ma visite et appel…
Vous êtes un médecin que chacun aimerait avoir sur son chemin. Avec humanité et

bonté, vous avez su m'épauler, me donner de la force, et me montrer que l'espoir n'était pas vain.
Grace à Allah et à vous je vais mieux aujourd'hui, je vous remercie énormément ainsi qu'au nom de mes parents, pour tous ce que vous avez fait pour moi, et je ne manquerai pas de vous tenir au courant de chaque progrès.

A mes chers oncles Yacine & Chakibe et leurs femmes, mes chères tantes Fella & Fethiya, je n'oublierai jamais tout ce que vous avez fait pour moi pour que je guérisse.

A ma chère cousine Ghada, en témoignage de l'amitié sincère qui nous a liées et de meilleurs et pires moments passés ensemble, je te dédie ce travail en te souhaitant tout le bonheur du monde.

A mes chers amis.

A toutes personnes que j'ai sentie redoutable, je leur dédie ce modeste travail en terme d'amour et de profonde gratitude.

ملخص: سيتم بناء الوظيفة الأساسية لشبكة الجيل الخامس من خلال "التطبيقات السحابية الأصلية" و المبدأ هو تقسيم البرامج إلى عناصر أصغر تسمى الخدمات الصغيرة ، و التي يهل إدارتها و نشرها على بنية أساسية سحابية موزعة تعتمد على الحاويات باستخدام نظام الافتراضية .

يستخدم نهج السحابة الأصلي آليات Kubernetes مثل إكتشاف الخدمة.الهدف الرئيسي من هذا المشروع هو تقديم تقنية السحابة الأصليةومبادئ تصميمها لتلبية متطلبات شبكة الجيل الخامس 5G الأساسية .

كلمات المفاتيح:

نهج السحابة الأصلية، نظام الإفتراضية 5Gالوظيفة الأساسية لشبكة الجيل الخامس.

Résumé : Dans ce projet nous allons présenter la technologie Cloud native ainsi que les principes de sa conception. Toute en mettant en place une application pour démontrer l'avantage de cette nouvelle technologie qui répond aux exigences du réseau cœur 5G.

L'idée est de décomposer un logiciel en éléments plus petits appelés Microservices, ces derniers sont plus faciles à gérer et à déployer à l'aide des mécanismes Kubernetes sur une infrastructure cloud distribuée basée sur des conteneurs.

Mots clés : Fonctions Réseaux 5G,Cloud Native, Virtualisation .

Abstract : The NF 5G core will be built through "Cloud native applications". The principle consists of breaking down software into smaller elements called Micro-services, which are easier to manage and deploy on a distributed cloud infrastructure based on containers in using virtualization.

The cloud native approach uses Kubernetes mechanisms such as service discovery. The main goal of this project is to present the native Cloud technology and the principles of its design, to meet the requirements of the 5G core network.

Keywords : Network Functions 5G , Cloud Native , Virtualization .

Listes des acronymes et abréviations

A

AF: Application Function
AMF: Access and Mobility Function
API: Application Programming Interface

ARP: Address Resolution Protocol
AUSF: Authentication Server Function

B

BSS: Business Support System

C

CIDR: Classless Inter-Domain Routing
CLI: Command Line Interface
CRI: Container Runtime
CUPS: Control and User Plane Separation
CPU: Central Processing Unit

E

EAP: Extensible Authentication Protocol
eMBB : Enhanced Mobile Broadband
EPC: Evolved Packet Core
E-UTRAN: Evolved Universal Mobile Telecommunications System Terrestrial Radio Access Network

H

HSS: Home Subscriber Server

L

LTE: Long Term Evolution
LXC: Linux Container

M

MAC: Media Access Control
MME: Mobility Management Entity
mMTC:Massive Machine Types

Communications

N

NAT: Network Address Translation
NAS: Non Acces Stratum
NFV: Network Functions Virtualization
NFVI: Network Functions Virtualization Infrastructure

ng-Nb: Next Generation Nb
NG-RAN: Next Generation Radio Acess Network
NSSF: Network Slice Selection
NSSAI: Network Slice Selection Assistance Information

O

OS : Operating System
OSS: Operations support system

I

IMS: IP Multimedia Sub-System
IOT: Internet of Things
IP: Internet Protocol.
ITU: Internet Control Message Protocol.

P

PCF: Policy Control Function
PCRF: Policy Control and Charging Rules
PDN: Public Data Network
P-GW: Packet Gateway Function
POP: Point of Presence

S

SBA: Service Based Architecture
SDN: Software Defined Networking
S-GW: Serving Gateway
SMF: Session Management Function

U

UDM: Unified Data Management
UDR: Unified Data Repository
UE: User Equipment
UPF: User Plan Function
URLLC: Ultra Reliable & Low Latency Communications

V

VM: Virtual Machine
VNFM: Virtual Network Function Manager
VIM: Virtualized Infrastructure Manager

Liste des figures

Chapitre I : Introduction aux réseaux 5G

Chapitre II : L'approche cloud native avec NFV

Chapitre III : Implémentation et réalisation de l'application

Table des matières

Introduction générale

A nos jours, un téléphone portable avec accès à Internet est devenu un besoin fondamental, et il devient de plus en plus essentiel. La diversité d'utilisation et les différentes applications dans tous les domaines, passer un appel vidéo de bonne qualité, ou regarder un film en streaming sans interruption…etc… nécessite une vitesse de téléchargement supérieure à 300 Mb/s, (celle qui est actuellement disponible sur les réseaux 4G).

Elle ne sera pas en mesure de répondre aux futures demandes des utilisateurs telle que l'Internet des objets (IoT), la télé médecine, les voitures autonomes qui visent l'hyper connectivité et la possibilité de connecter simultanément nos maisons (maisons intelligentes), véhicules, montres et villes au réseau.

Ces multiples cas d'usage pourront donc supporter un nombre important de connexions, absorber une grosse quantité de données, et nécessiter de nouveaux types de performances améliorées.

La clé du succès de ces cas d'usage est la latence presque nulle, et la faible consommation d'énergie ce qui permettra aux objets connectés de fonctionner pendant des mois, voire des années, sans assistance humaine.

Le but des anciennes générations, la 3G et la 4G était d'améliorer la vitesse afin de supporter l'usage croissant de l'Internet mobile. Mais, elles ne sont pas spécifiquement conçues pour répondre à trois nouvelles problématiques : l'Internet des objets (IOT), le besoin de débit élevé partout sur le territoire, et les économies d'énergie ; ce qui remet en question ces méthodes traditionnelles de construction des réseaux cellulaires.

Les opérateurs de téléphonie mobile recherchent de nouvelles solutions pour améliorer la capacité du réseau, la couverture et l'expérience des utilisateurs, tout en réduisant les délais de mise sur le marché de nouveaux services et les coûts.

La cinquième génération du réseau mobile est là pour répondre à plusieurs problématiques : l'augmentation des débits nécessaires pour les nouveaux usages numériques, la réduction de la latence mais aussi la densification du réseau.

La 5G aura donc une capacité supérieure à la 4G, ce qui sera particulièrement utile pour les zones denses comme les centres villes.

Premièrement, elle va travailler sur l'augmentation des débits pour réaliser ce que l'on appelle le Très Haut Débit (VHF) mais aussi l'Ultra Haut Débit (UHF).

Cette augmentation de débit sera utile pour divers usages : la réalité virtuelle, le contenu à très haute définition (4K, 8K), le cloud (plusieurs Go et To à transférer). Cela va permettre d'accéder à de nouveaux services et d'améliorer des services actuels.

Enfin, la 5G va s'intéresser à la densification du réseau : en offrant une meilleure capacité, plusieurs appareils seront connectés en même temps tout en assurant une qualité de connexion et une fiabilité de fonctionnement. Ce qui permettra aux millions d'objets connectés de fonctionner, mais aussi à des dizaines de milliers d'utilisateurs d'accéder au réseau dans des zones très denses, comme un stade par exemple.

Pour ce faire, les opérateurs de téléphonie mobile doivent adopter une architecture d'application réseau de base qui doit prendre en compte les exigences de déploiement flexible pour gagner en rapidité de mise en œuvre, en souplesse, en qualité et en sécurité de fonctionnement.

Notre projet intitulé « Cloud Native pour les applications 5G-VNFs évolutifs basés sur des micro-services avec Kubernetes » a pour but principale de présenter la technologie Cloud native et les principes de sa conception, ainsi que mettre en place une application pour démontrer l'avantage de cette nouvelle technologie qui constitue la clé pour répondre aux exigences du réseau cœur 5G.

Cette solution vise à répondre aux objectifs suivants :

- ✓ Réduction de la consommation de la RAM de 30 fois.
- ✓ Rapidité de déploiement, au lieu des heures il sera en quelque secondes.
- ✓ Réduction des couts d'investissement et d'exploitation, par exemple 100 conteneurs pourraient être exécutés en parallèle sur un seul serveur.
- ✓ Développer et exécuter des applications réactives, évolutives et résistantes aux pannes.

- ✓ Mettre à jour rapidement des applications en quelques heures au lieu des jours ou bien en quelques minutes au lieu des heures.
- ✓ Augmentation de la qualité de prestation des services.
- ✓ Meilleure satisfaction des clients.
- ✓ Prévision d'une augmentation du débit de 100Mps jusqu'à 10Gps et une latence presque nulle (de 35ms jusqu'à 4 ms).

Le présent mémoire sera présenté en trois chapitres :

Le premier chapitre : va contenir une présentation des réseaux 4G et leur amélioration vers les réseaux 5G.

Le deuxième chapitre : parlera de la description de la technologie « Cloud native » et les principes de sa conception, ses avantages dans la partie cœur du réseau 5G, ainsi que les fonctions de virtualisation de réseaux.

Le troisième chapitre : traitera la partie pratique de notre travail, il donne une présentation de notre solution, les différentes étapes d'installation de l'application, ainsi que les tests faits et les résultats obtenus.

Chapitre I

Introduction aux réseaux 5G

I.1 Introduction

La prochaine génération de technologie sans fil mobile sera dénommée la « 5G ». Comme son nom l'indique, elle fait suite aux précédentes générations de la téléphonie mobile.

La première génération de téléphones mobiles vers 1980 « 1G », ces téléphones fonctionnaient comme des radios et utilisaient leurs fréquences en mode analogique.

Au début des années 1990, les téléphones de deuxième génération « 2G » ont été développés avec une évolution vers les réseaux numériques : les débits d'échanges de données pour ces téléphones étaient inférieurs à 1000 bits par seconde (1000 bps) mais des améliorations significatives en termes de performance ont été introduites en l'an 2000 « 2,5G ».

Peu de temps après, la troisième génération est apparue « 3G », et la vitesse de débit de données a augmenté jusqu'à 100000 bps; l'amélioration était considérable, puisqu'il devenait possible de transmettre des appels vidéo limités, et de fournir des connexions Internet à des vitesses raisonnables.

Des améliorations ont été introduites dans le codage numérique de la communication « 3.5G », vers 2009 ; et « 3.9G » en 2012.

En 2015, avec la norme suivante (dite génération « 4G »), des vitesses de transmission dix fois supérieures devinrent possible. Depuis le passage vers les réseaux cellulaires au début des années 1990.

La nouvelle génération « 5G » rendra possible un certain nombre de nouvelles applications. On pourra bénéficier de vidéos en ligne avec une très bonne qualité de réception, même dans des zones denses comma un stade par exemple, où chaque spectateur utilise un dispositif connecté.

En effet, la 5G devrait désormais proposer de nouveaux cas d'utilisation et de nouveaux modèles commerciaux aux consommateurs, aux entreprises et à l'industrie.

Voitures autonomes, robotique avancée, objets connectés, gestion de l'énergie en direct. Autant d'usages qui cherchent un réseau dense, rapide et accessible pour les

gouverner tous.

Dans ce chapitre nous allons présenter l'évolution du réseau de la cinquième génération et les concepts clés qui ont permis la mise à niveau de son infrastructure.

I.1.1 Le réseau de la 4ème génération « 4G » LTE

La 4G est la quatrième génération de réseau mobile. C'est la norme précédente à la 3G. La 3GPP (*3rd Generation Partnership Project*) a défini cette technologie comme suite au succès que les réseaux 2G et 3G ont connu.

Elle permet le transfert de données à très haut débit, avec une portée plus importante, un nombre d'appels par cellule supérieur et une latence plus faible.

I.1.1.1 L'architecture d'un réseau LTE [1]

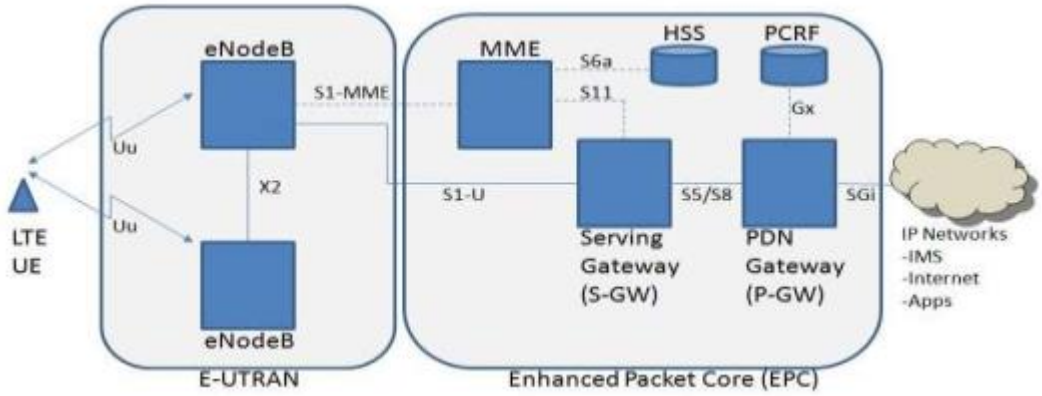

Figure I.1 : L'architecture d'un réseau LTE

L'architecture d'un réseau LTE est composée de deux entités principales comme la figure ci-dessus le montre.

- Partie accès radio E-UTRAN (*Evolved-UTRAN*).
- Partie réseau cœur EPC (*Evolved PacketCore*).

Chaque partie contient des entités qui sont reliées entre elles par des interfaces.

I.1.1.1.1 Le réseau d’accès radio e-utran (evolved-utran)

La partie radio du réseau est appelée "E-UTRAN" est simplifiée par l'intégration de stations de base "eNodeB".

E-UTRAN fournit des connexions entre les terminaux mobiles et les réseaux centraux des opérateurs mobiles par ondes radio. Les deux entités qui existent dans E-UTRAN sont UE (User Equipment) et eNodeB:

- **UE** (**équipement utilisateur**) : il s'agit de téléphones mobiles et d'autres appareils prenant en charge la norme LTE.
- **E-NodeB** : les nœuds E-UTRAN B ou eNodeB sont répartis sur l'ensemble du réseau de l'opérateur mobile. Ils connectent le terminal mobile au réseau central via une interface radio.

I.1.1.1.2 Le réseau cœur epc (evolved packet core)

Le réseau cœur est le cerveau du système. Il est constitué de commutateurs téléphoniques qui permettent aux différents utilisateurs d’utiliser différents services. Les périphériques réseaux centraux connectent les périphériques mobiles du réseau mobile. Ils connectent également le réseau mobile au réseau de téléphonie fixe et à Internet.

L’EPC qui est le cœur du réseau, Constitué principalement de :

- **MME** (Mobility Management Entity).
- **SGW** (Serving Gateway).
- **PGW** (Packet Data Network Gateway).

Le biais de l’E-UTRAN permet l’UE de communiquer avec l’EPC. Lorsque un UE est allumé, l’EPC est responsable de l’authentification et l’établissement de la connexion nécessaire pour toute la communication. Le LTE a une architecture différente à celle de l’UMTS, dite toute IP, qui supporte uniquement les données à commutation par paquet.

Le réseau principal LTE est appelé EPC (Evolved Packet Core) et il est constitué de cinq nœuds :

✦ **MME :** est le nœud de contrôle du réseau EPC. Il est responsable de la signalisation, de la mobilité et de la sécurité.

✦ **S-GW :** interconnecte le réseau d'accès radio avec le réseau EPC, Et il est connecté au P-GW.

✦ **P-GW :** passerelle PDN (réseau de données par paquets) connecte le réseau EPC aux réseaux externes. Il achemine le trafic depuis et vers les réseaux PDN.

✦ **HSS :** est la base de données de tous les utilisateurs mobiles qui inclut toutes les données des abonnés. Il est responsable de l'authentification, de la configuration de l'appel et de la session.

✦ **PCRF :** est le nœud responsable des règles de politique en temps réel et de la facturation dans le réseau EPC.

L'utilisation croissante de la technologie LTE cause une surcharge énorme sur l'EPC. Alors les opérateurs cherchent toujours un réseau qui peut répondre à des problématiques de coût, de performance et de souplesse dans le fonctionnement.

Cependant la dépendance entre les fonctions réseaux et le matériel limite la flexibilité du réseau, car ces fonctions réseaux (routage, commutation, équilibrage de charge, gestion de la voix, gestion de données, et stockage des clients etc…) sont réalisées sur des boitiers hardware dédiés.

Chaque fonction est dépendante du hardware sur lequel elle fonctionne. Cela signifie que chaque nœud du réseau a pour objectif de faire fonctionner une application réseaux spécifique (elle lui est dédiée) Exemples :
Les routeurs, HSS, MME, PCRF...etc. Cette architecture représente les problèmes suivants:

✓ Manque d'évolutivité.
✓ Manque de flexibilité.
✓ Manque de rapidité.

Afin de répondre à ces problèmes, les entreprises ont de plus en plus recourt à la virtualisation de leur infrastructure.

I.2 VIRTUALISATION DES FONCTIONS RESEAUX NFV (NETWORK FUNCTIONS VIRTUALISATION)

I.2.1 DEFINITION

L'approche NFV (*Network Functions Virtualization*) est basé sur la virtualisation informatique permettant la dissociation des fonctions réseau des équipements matériels dédiés
et les placer dans des machines virtuelles.
La volonté d'automatiser l'orchestration et la gestion du réseau, du stockage et des ressources de calcul constitue un facteur clé en matière de développement NFV.
L'automatisation permet de créer ou de supprimer rapidement les fonctions réseau virtualisées
(VNF) pour les adapter automatiquement à une demande dynamique.

I.2.2 L'architecture NFV [2]

La figure I.2 montre l'architecture NFV

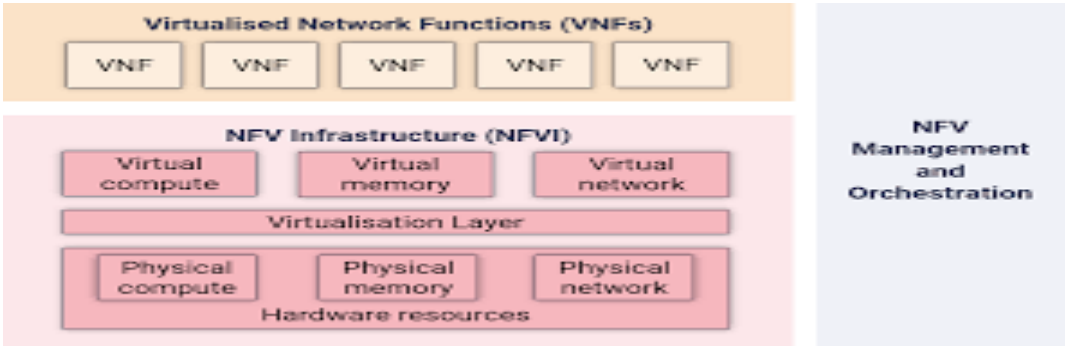

Figure I.2 : Architecture NFV

L'architecture NFV est constituée de :

✦ **L'infrastructure NFV :** NFVI (*Network Functions Virtualization Infrastructure*) fournit les ressources matérielles (serveurs, cartes électroniques, …) et le logiciel de virtualisation. Le NFVI qui est composé de :

- Domaine de calcul virtualisé (processeurs, accélérateurs, …).
- Domaine de l'hyperviseur : système d'exploitation supportant la virtualisation (machines virtuelles) et/ou des conteneurs pour faire fonctionner les VNF,

ainsi que les capacités de commutateurs virtuels (vSwitch).
- Domaine réseau de l'infrastructure.

✦ **Le VNF (*Virtualized Network Functions*) :** correspond aux fonctions réseaux virtualisées pouvant être exécutées sur les équipements de NFVI.

✦ **NFV M&O (*Management and Orchestration*) :** permettant de gérer les services réseaux de bout en bout, il contient 3 blocs :
- L'orchestrateur (NFV Orchestrator) : L'entité d'orchestration est responsable de cycle de vie des services réseau tant au niveau logiciel que matériel sur plusieurs domaines en contrôlant les VIM de chaque domaine.
- Un gestionnaire (VNFM) en charge du cycle de vie des VNFs.
- Un gestionnaire (VIM) en charge de la gestion et le contrôle des ressources du NFVI à l'intérieur d'un domaine.

Les services OSS/BSS doivent pouvoir transmettre au bloc NFV M&O des informations sur
le profil des utilisateurs, la facturation, les politiques de qualités, les accords entre domaine.

I.2.3 Les avantages de la NFV

La NFV consiste à séparer les fonctions réseau du matériel fourni, ceci a apporté beaucoup d'avantages comme :

✦Réduction des couts matériels du réseau.

✦Réduction de l'espace nécessaire aux équipements matériel du réseau.

✦Réduction de la consommation électrique du réseau.

✦Réduction des couts de maintenance du réseau.

I.3 La cinquième génération des réseaux mobiles 5G

I.3.1 Définition

La nouvelle génération « 5G » des standards de la téléphonie mobile prend le relais de la 4G. Elle a été validée par l'ITU (*International Telecommunication Union)* et le 3GPP (*3rd GenerationPartnership Project*). Cette technologie de

télécommunication sans fil est conçue pour répondre à la très grande croissance des données et à la connectivité de la société moderne.

Avec la 5G, des débits très élevés seront atteints, et le temps de latence sera considérablement diminué.

I.3.2 L'évolution de l'architecture 4G « LTE » vers l'architecture du réseau cœur 5G

Le réseau cœur 5G devra prendre en charge un nombre de solutions très élevé, ainsi, optimiser les offres de services existantes simultanément, Alors Il doit être conçu d'une manière qui maximise la flexibilité et l'agilité.
Des technologies spécifiques ont été inclues comme le CUPS, Network slicing et le cloud distribué permettent un changement remarquable dans l'architecture réseau.

I.3.2.1 CUPS : Séparation de plan de contrôle du plan d'utilisateur [3]

Le CUPS consiste à séparer en deux parties les entités SGW et PGW qui sont le plan de contrôle et plan de données.

- SGW → SGW-C et SGW-U
- PGW → PGW-C et PGW-U

Cette séparation permettra une exploitation de réseau flexible, grâce à un déploiement distribué ou centralisé et à une mise à l'échelle indépendante entre les fonctions du plan de contrôle et du plan d'utilisateur, sans affecter la fonctionnalité des nœuds existants soumis à cette séparation.

La figure ci-dessous illustre l'évolution de l'architecture 4G LTE vers l'architecture de réseau cœur 5GC.

Figure I.3 : L'évolution de l'architecture 4G à 5G

Architecture basée sur les services (SBA)
Une fonction réseau NF peut exposer un ensemble de services (Producteur de service NF) aux autres fonctions réseaux (Consommateur de service NF) via deux interfaces élémentaires :

- Demande-réponse : Un plan de contrôle NF B (Consommateur de service NF)

demande à un plan de contrôle NF A (Producteur de service NF) de fournir un certain
service NF, pouvant inclure l'exécution d'une action et / ou la fourniture d'informations, La réponse de NF A fourni des résultats de service de NF sur la base des informations fournies par NF B dans sa demande.

- Abonnement-notification : un plan de contrôle NF A (consommateur de service NF)
s'abonne au service NF proposé par un autre plan de contrôle NF B (producteur de service NF). La NF B communique les résultats de ce service aux NFs intéressées ayant souscrit à ce service. La demande d'abonnement du consommateur peut inclure
une demande de notification pour des mises à jour périodiques ou une notification déclenchée par certains événements (par exemple, les informations demandées sont
modifiées, atteignent un certain seuil, etc.)

I.4 L'architecture du réseau 5G

I.4.1 Réseau cœur 5G « 5G Core »

La figure ci-dessous illustre l'architecture du réseau central 5GC :

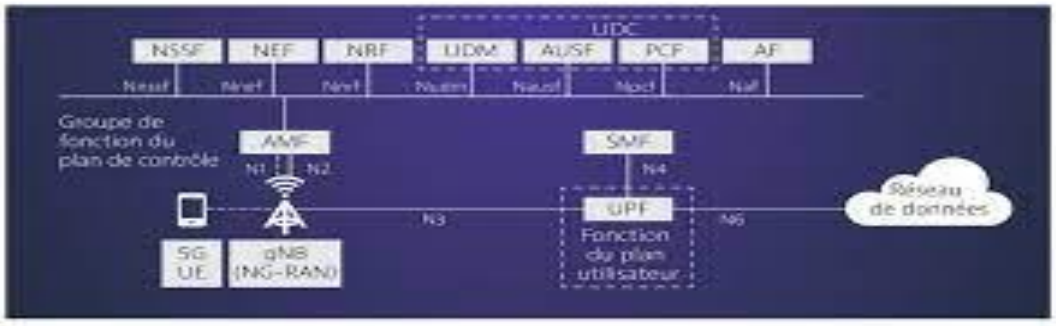

Figure I.4 : L'architecture du réseau 5G

AMF *(Access and Mobility Function*)
L'AMF exécute la plupart des fonctions que le MME exécute dans un réseau 4G.

- ✦ Authentification de l'UE.
- ✦ Enregistrement (attachement) des mobiles UE.
- ✦ Gestion des inscriptions.
- ✦ Gestion de la connexion.
- ✦ Gestion de l'accessibilité.
- ✦ Gestion de la mobilité.

SMF (*Session Management Function*)
La fonction SMF exécute les fonctions de gestion de session gérées par : 4G MME, SGW-C et PGW-C.
✦ Allocation des adresses IP aux UE.
✦ Signalisation NAS pour la gestion de session.
✦ Envoie des informations de qualité de service et de politique au NG-RAN via l'AMF. ✦ Agit comme interface pour toutes les communications liées aux services de plan
d'utilisateur proposés.

UPF (*User Plan Function*)
L'UPF est essentiellement une fusion des parties du plan de données du SGW et du PGW.
Dans le contexte de l'architecture CUPS : EPC SGW-U + EPC PGW-U → 5G UPF
L'UPF exécute les fonctions suivantes
✦ Routage et transmission des paquets.
✦ Inspection des paquets et gestion de la qualité de service.
✦ Connexion à Internet POP (Point of Presence).

La fonction UPF peut éventuellement
intégrer les fonctions de pare-feu et de traduction d'adresses réseau (NAT).

PCF (*Policy Control Function*)
La PCF 5G exécute la même fonction que la PCRF dans les réseaux 4G.
✦ Fournit des règles de stratégie pour les fonctions du plan de contrôle. Cela inclut la
gestion du découpage en réseau, de l'itinérance et de la mobilité.
✦ Prend en charge la nouvelle politique de qualité de service 5G et les fonctions de contrôle de facturation.

UDR (*Unified Data Management*)
C’est une base de données qui stocke le profil utilisateur (son abonnement, ses droits, …).

UDM (*Unified Data Management*)
L'UDM exécute des parties de la fonction HSS en 4G.
✦ Identification de l'utilisateur.
✦ Autorisation d'accès.
✦ Gestion des abonnements.

AUSF (*Fonction serveur d'authentification*)
L'AUSF exécute la fonction d'authentification du HSS en 4G.
✦ Implémente le serveur d'authentification EAP.
✦ Stocke les clés.

AF (*Application Function*)
✦ contrôle des politiques.

NRF (*NF RepositoryFunction*)
✦Enregistrement de service et fonction de découverte permettant aux fonctions réseau
de se découvrir mutuellement. Maintient le profil NF et les instances NF disponibles.

NEF (*Network Exposure Function*)
NEF fournit un mécanisme permettant d'exposer en toute sécurité les services et les
fonctionnalités du réseau cœur de la 5G.
✦ Sécurisation de la fourniture d'informations d'une application externe au réseau 3GPP.
✦ Traduction d'informations internes / externes.

NSSF (*Network Slice Selection Function*)
NSSF redirige le trafic vers une tranche de réseau. Les tranches de réseau peuvent être
définies pour différentes classes d'abonnés. La NSSF exécute les fonctions suivantes :
✦ Sélection des instances de tranche de réseau pour desservir l'UE.
✦ Détermination de l'ensemble AMF à utiliser pour desservir l'UE

I.4.2 L'accès radio 5G [4]

L'accès radio 5G est constitué de stations de base de nouvelle génération qui forment
le nœud de connexion des mobiles avec le cœur du réseau 5G.
Les mobiles UE communiquent avec les stations de base soient par un lien radio 5G, soit par un lien radio 4G. Si la communication est en 5G, la station de base se nomme gNB
(nextGenerationNode Base Station), si la communication est en 4G, la station de base

est une
station de base 4G eNB évoluée pour s'interconnecter avec le cœur du réseau 5G. La station
de base se nomme ng-Nb (NextGeneration-Nb).
Les fonctions de la station de base gNb sont assez similaires avec l'entité eNB.
Cependant, les différences concernent la gestion de la qualité de service par flux et non par
support (*bearer*) et la gestion des tranches de réseau (Slices) sur l'interface radio

I.4.3 Network Slicing [5]

Le découpage réseau en tranches (network slicing) est l'élément clé pour exploiter pleinement le potentiel de l'architecture 5G.

Cette technologie apporte une dimension supplémentaire à la NFV en permettant à de multiples réseaux logiques de fonctionner simultanément sur une infrastructure physique de réseau partagée.

Elle devient ainsi une partie intégrante de l'architecture 5G en créant des réseaux de bout en bout virtuels qui comprennent des fonctions de mise en réseau et de stockage.

Le découpage réseau en tranche est extrêmement utile pour les applications telles que l'Internet des objets (IOT) pour lesquelles le nombre d'utilisateurs peut être très élevé alors que la demande générale en bande passante reste faible.

Chaque verticale 5G aura ses propres exigences.

C'est pourquoi il est important de tenir compte de découpage réseau en tranche.

Dans la conception de l'architecture réseau 5G, le cout, la gestion des ressources et la flexibilité des configurations des réseaux peuvent être optimisés grâce à ce nouveau niveau de personnalisation.

De plus, le network slicing permet d'effectuer des essais plus rapidement pour les nouveaux services potentiels de la 5G, tout en réduisant le délai de mise sur le marché.

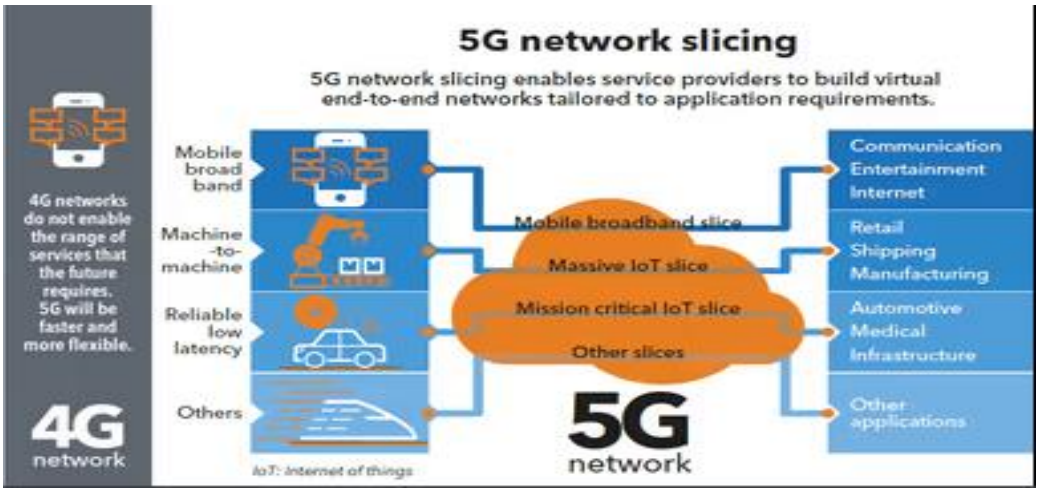

Figure I.5 : Network Slicing

I.4.4 LE CLOUD DITRIBUE [6]

Un Cloud distribué est une architecture dans laquelle plusieurs Clouds sont utilisés pour répondre aux besoins de conformité, aux exigences relatives aux performances ou pour prendre en charge l'Edge Computing tout en étant gérés de manière centralisée par le fournisseur de Cloud public.

En effet, un service de Cloud distribué est un Cloud public qui est exécuté sur plusieurs sites, notamment :

- L'infrastructure du fournisseur de Cloud public
- Dans le Data Center d'un autre fournisseur de Cloud
- Sur du matériel tiers ou de centre de colocalisation

L'apparition de l'Internet des objets (IoT) et de l'Edge Computing a été un moteur essentiel aux déploiements des Clouds distribués.

Les applications d'intelligence artificielle (IA) qui déplacent de grandes quantités de données d'emplacements périphériques vers le Cloud exigent que les services Cloud soient aussi proches que possible de ces emplacements.

En outre, le déplacement des ressources Cloud vers ces emplacements périphériques peut considérablement améliorer les performances de ces applications.

En rapprochant les services Cloud d'un utilisateur, d'une application ou des données, les Clouds distribués peuvent offrir :

- Une réduction de la latence.
- Une congestion du réseau réduite ou supprimée.
- Une qualité de service (Quality Of Service) (QoS) garantie pour les applications stratégiques et les utilisateurs mobiles
- Répondre aux exigences des cas d'utilisation nécessitant une bande passante élevée.

I.5 Conclusion

Les avantages attrayants de la nouvelle génération 5G tels que le débit très élevé et la latence presque nulle, vont servir à de nouveaux cas d'usages ; ainsi qu'à la possibilité de connecter plusieurs nouvelles technologies comme l'Internet des objets (IOT), les voitures autonomes, l'intelligence artificielle (IA), et la réalité virtuelle.

Ceci a eu naissance grâce à la conception de la 5G qui est basée sur l'application de la technologie cloud et de la NFV qui feront l'objectif du prochain chapitre.

Chapitre II

L’approche « cloud native avec NFV »

II.1 Introductions

Les technologies de l'information et de la communication évoluent et révolutionnent nos modes de vie et de travail. La technologie cloud et la NFV sont apparues ces dernières années comme des nouveaux modèles de gestion et d'utilisation des systèmes informatiques. Ces technologies ont bouleversé l'ordre établi dans presque tous les secteurs. La raison est simple, grâce à eux, les entreprises ont pu réduire les investissements consacrés à leurs centres de données internes au profit de ressources de calcul illimitées, disponibles à la demande et facturées à l'utilisation.

Dans ce chapitre nous allons présenter le cloud native et la virtualisation, les expliquer, et voir les liens existants entre ces deux technologies.

II.2 Cloud native

Le cloud native est une nouvelle technologie qui a comme concept de créer et d'exécuter des applications pour tirer parti de l'informatique distribuée offerte par le modèle de fourniture cloud. Les applications cloud native ont été conçues et construites pour exploiter l'évolutivité, l'élasticité, la résilience et la flexibilité du cloud.

Les entreprises cloud native peuvent créer et exécuter des applications évolutives dans des clouds publics, privés et hybrides. Des caractéristiques telles que les conteneurs, les maillages de services, les microservices, les infrastructures immuables et les interfaces de programmation d'applications (API) déclaratives illustrent parfaitement cette approche.

Le principe de cloud native consiste à créer des logiciels sur des microservices conçus et déployés indépendamment les uns des autres. Ces microservices communiquent via des interfaces d'applications et peuvent être déployés sur une infrastructure cloud distribuée basée sur des conteneurs avec un système d'orchestration central.
Ces applications sont plus rapides à mettre à jour, à être distribuées sur des portées lointaines, et requièrent moins d'espace de stockage.

✦ **Les avantages des applications cloud native**

• **Compétitivité**

À l'ère du logiciel, les grandes gagnantes seront les entreprises capables de développer et de déployer rapidement des applications en réponse aux besoins de leurs clients. Une fois déployées, les applications doivent s'exécuter sous forme de services scalables et disponibles en continu.

• **Flexibilité**

Les entreprises peuvent créer des applications qui s'exécuteront sur n'importe quel cloud, sans aucune modification. Leurs équipes ont ainsi la possibilité de migrer ou distribuer leurs applications entre divers clouds publics et privés pour satisfaire leurs priorités et réduire leurs coûts.

• **Développement optimal**

Avec l'approche cloud native, les développeurs n'ont plus à tenir compte des contraintes d'exécution et de montée en charge sur une diversité d'infrastructures cloud. Ils se recentrent ainsi sur l'écriture de code à plus forte valeur ajoutée.

II.3 Les microservices

II.3.1 Définition

Les microservices est une approche de développement logiciel. Elle consiste à décomposer les applications en éléments plus petits appelés « services », ces derniers sont indépendants les uns des autres. Chaque service exécute un processus unique et gère sa propre base de données, ce processus est donc un microservice.

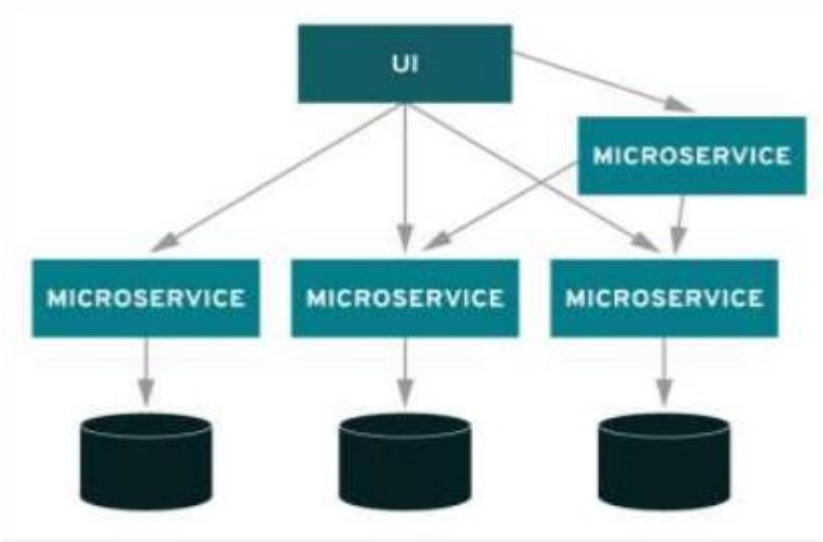

Figure II.1 : L'architecture des microservices

II.3.2 Caractéristiques des microservices [7]

Les microservices ne fonctionnent pas de manière standard. Cependant, on peut trouver des caractéristiques communes.

- **Caractéristiques liées à son logiciel**

Il peut être décomposé en plusieurs parties indépendantes, et chacun des services peut être déployé et modifié sans affecter les autres fonctions de l'application.

- **Caractéristiques liées à son organisation**

Son organisation est faite d'une manière qui implique un contraste avec l'environnement monolithique, car ils prennent en compte des aspects tels que les capacités de l'entreprise où ils doivent être mis en œuvre.

- **Caractéristiques liées à son architecture**

Les modules sont indépendants, puisque chacun d'entre eux possède sa propre base de données, tous les services ne se connectent pas à la même base de données. Donc, la surcharge et les temps d'arrêt des applications sont évités.

- **Caractéristiques liées à leurs avertissements et actions**

Comme plusieurs services sont communiqués, on a besoin de systèmes d'alerte et d'action en cas de panne sur ces services. Autrement dit, on a un avertissement, un e-mail sera envoyé au support…etc.... Ce système est avantageux, car il favorise un bon fonctionnement entre les modules.

II.3.3 Les avantages des microservices

Parmi les avantages les plus bénéfiques des microservices on cite :

- La rapidité
- La haute évolutivité
- La résilience
- La facilité de déploiement
- L'accessibilité
- Liberté des développeurs pour développer et déployer des services de manière indépendante.

II.4 La technologie des conteneurs

L'exécution de nouvelles applications nécessite l'intervention des microservices. Pour ce faire, il est important de faire appel aux conteneurs. Dans ce cas, et le Docker, est considéré comme la meilleur solution pour le déploiement de ces microservices dans les conteneurs dédiés.

II.4.1 Da la virtualisation par VM à la virtualisation par conteneurs

II.4.1.1 La virtualisation [8]

Dans cette section nous commencerons par définir le concept de virtualisation. Ensuite nous allons présenter les différents objectifs, les différentes techniques et les différents types de la virtualisation.

La virtualisation recouvre l'ensemble des techniques matérielles et/ou logiciels qui permettent d'exécuter plusieurs machines virtuelles dites invitées (guest) sur une seule machine dite hôte (host). La virtualisation constitue le socle du Cloud Computing, car elle offre la possibilité de mettre en commun des ressources informatiques à partir de clustersde serveurs, et ainsi d'affecter ou réaffecter dynamiquement des machines virtuelles aux applications à la demande. Pour bénéficier de cette technologie, il suffit d'équiper une machine d'un logiciel de virtualisation permettant d'ajouter une couche de virtualisation, appelée hyperviseur.(GERVAISE, 2012)

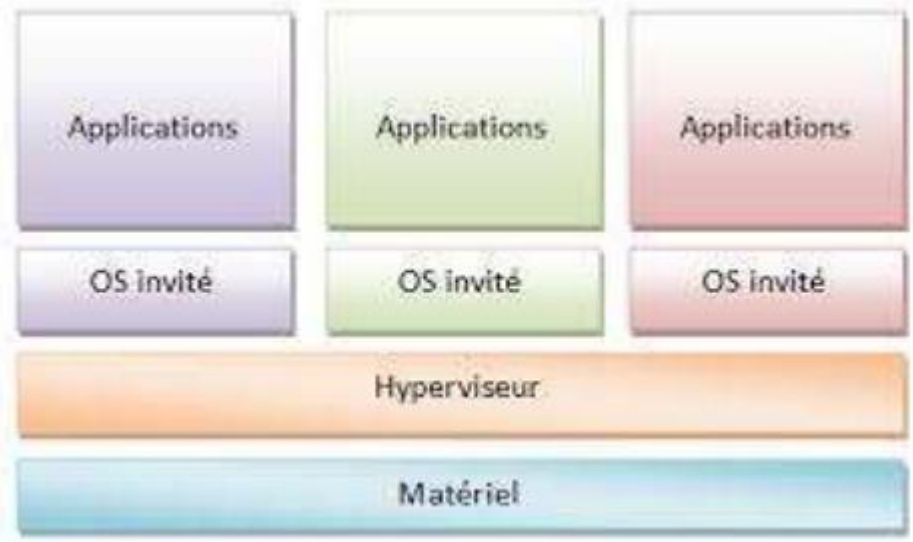

Figure II.2 : L'architecture d'une machine virtuelle

II.4.1.2 La conteneurisation

Un conteneur consiste en un environnement d'exécution complet : une application, ses dépendances, ses bibliothèques et autres fichiers binaires, ainsi que les fichiers de configurations nécessaires pour l'exécuter, regroupés dans un seul package. [9]

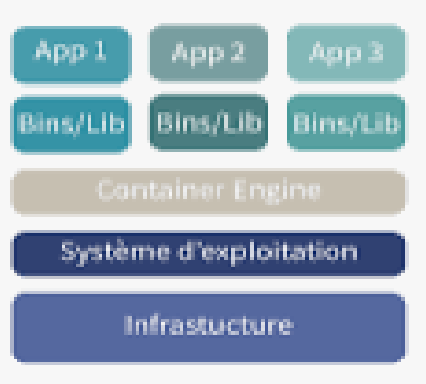

Figure II.3 : L'architecture d'un conteneur

La virtualisation par conteneurs se base sur la virtualisation Linux LXC, pour *Linux Containers*. Il s'agit d'une méthode de cloisonnement au niveau de l'OS.

Le principe est de faire tourner des environnements Linux isolés les uns des autres dans des conteneurs partageant le même noyau.
Ce qui veut dire que contrairement aux machines virtuelles traditionnelles, un conteneur n'inclut pas d'OS, puisqu'il s'appuie sur les fonctionnalités de l'OS de la machine hôte. Les conteneurs accèdent alors à l'OS hôte de manière totalement isolée les uns des autres.
Le conteneur virtualise l'environnement d'exécution (comme le processeur, la mémoire vive ou le système de fichier…) et ne virtualise donc pas la machine.

C'est pour cela que l'on parle de « conteneur » et non de machine virtuelle (VM). LXC repose principalement sur deux fonctionnalités du noyau Linux :

1. **Cgroups (Control groups) :** Ont pour but de contrôler les ressources systèmes utilisées par un ou plusieurs processus. Les processus sous contrôle sont affectés dans des groupes sur lesquels agissent des contrôleurs de ressources. Chaque contrôleur gère un type de ressources :

• cpuset : allocation CPU (numéro CPU/core CPU).
• cpuacct : consommation CPU (nombre de cycles).
• memory : contrôle la mémoire vive et la mémoire swap.
• devices : contrôle l'accès aux périphériques.
• blkio : contrôle l'accès aux périphériques de type bloc (ex : disque dur).
• net_cls : contrôle l'accès aux périphériques réseau.

2. **Les espaces de noms (names-spaces)** : Les conteneurs Linux proposent également une isolation complète de leurs espaces de noms. Chaque type d'espace de noms s'applique à une ressource spécifique. Et chaque espace de noms crée des barrières entre les processus. Ces obstacles peuvent être à différents niveaux. Les types des espaces de noms que docker utilise :

• pid : isole les processus (pid : Process ID).
• net : permet la gestion des interfaces réseaux (net : networking).
• ipc : gère les accès inter-processus (ipc : InterProcess Communication).
• mnt : gestion des points de montage (mnt : mount).

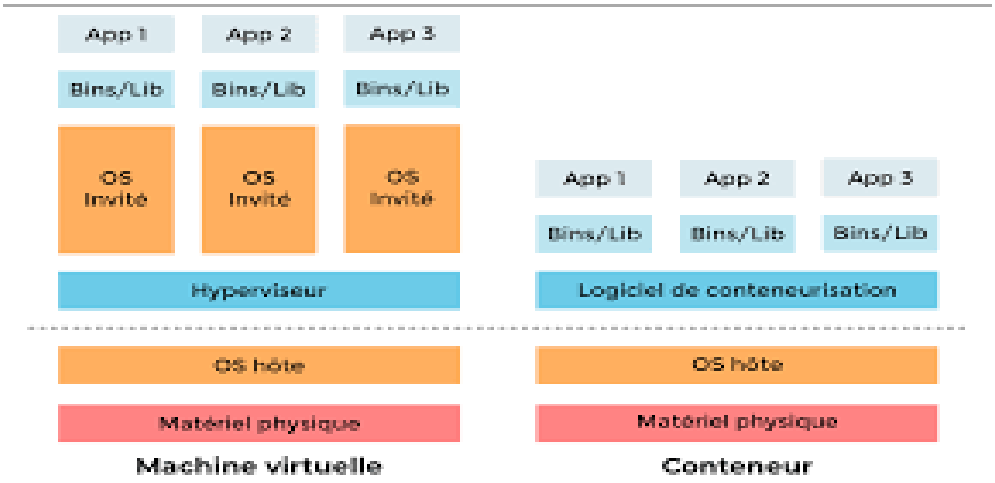

Figure II.4 : Comparaison entre la virtualisation par VM & la conteneurisation

II.4.1.3 Docker

Le Docker est probablement le logiciel libre qui s'est le plus popularisé ces dernières années. Il permet d'automatiser le déploiement d'applications et de leurs dépendances dans différents conteneurs isolés sur un n'importe quel serveur Linux. Il a été développé par Solomon Hykes à la base comme projet interne de DotCloud en tant

qu'évolution de solutions open-source déjà existantes au sein de la société. Docker a été distribué en open-source la première fois le 13 mars 2013. Début 2017, il y a 1600 contributeurs sur le Github de l'application. [10]

II.4.1.3.1 L'architecture de Docker

Docker utilise une architecture client-serveur. Le client Docker parle au démon Docker, qui s'occupe de la construction, de l'exécution et de la distribution des conteneurs Docker.

Le client et le démon Docker peuvent s'exécuter sur le même système, ou on peut connecter un client Docker à un démon Docker distant. Le client et le démon Docker communiquent à l'aide d'une API REST, via des sockets UNIX ou une interface réseau. Un autre client Docker est Docker Compose, qui permet de travailler avec des applications composées d'un ensemble de conteneurs. [11]

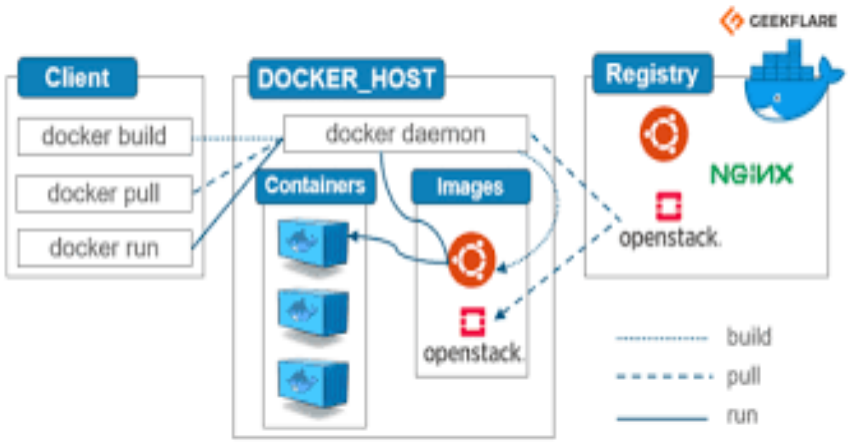

Figure II.5 : L'architecture de docker

1.Docker engine : Est la couche sur laquelle Docker est exécuté. C'est un outil d'exécution et de gestion léger qui gère les conteneurs, les images, les versions, etc.
2. Le client Docker : Le client docker est l'interface utilisateur principal de docker, il communique avec le Docker *daemon.*
3. Le démon Docker : Exécute réellement les commandes envoyées par le client Docker telles que la construction, l'exécution et la distribution des conteneurs.
4. Docker file : Est un ensemble de fichiers permettant de construire une image

Docker étape par étape d'une façon automatisée.
5. Une image Docker : est un template avec des instructions pour la création de conteneur, elle peut être construite à partir du Dockerfile ou être récupérée du « Registre Docker ». [12]

II.4.1.4 Kubernetes [13]

Kubernetes est un orchestrateur de conteneurs, il automatise le déploiement, l'évolution, la maintenance, la planification et le fonctionnement de nombreux conteneurs
d'application sur des clusters. Il contient des outils d'orchestration, de catalogue de services et
d'équilibrage de charge utilisables avec les conteneurs Docker .
Kubernetes aide les utilisateurs à s'assurer que les applications conteneurisées s'exécutent où et quand ils le souhaitent, et les aide à trouver les ressources et les outils dont
elles ont besoin pour travailler.

II.4.1.4.1 Architecture de Kubernetes [14]

L'architecture Kubernetes suit une architecture client-serveur, elle est définie par les
types de nœuds suivants :

- **Un nœud maître** « appelé Master » : chargé d'orchestrer le cluster, et c'est le point d'accès des tâches administratives par une interface API ou CLI.
- **Des nœuds esclaves** « appelés worker node » : correspondant à des hôtes physiques ou virtuels qui exécutent une application Docker. Ils sont contrôlés par le Master node.
- **Key-value store (**etcd) : Elle joue le rôle d'une base de données du cluster.

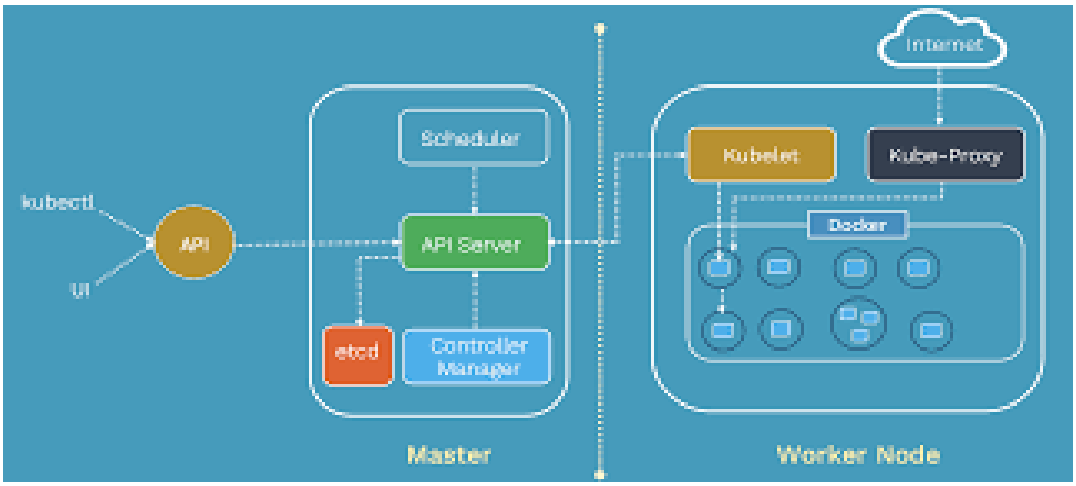

Figure II.6 : Architecture de Kubernetes

Dans ce cas, le nœud master est installé sur une machine et les worker sur des machines Linux distinctes. La communication entre le master et les worker (ainsi qu'à l'intérieur des nœuds) est permise via un réseau privé interne au cluster.

Les composantes de « master node » : Le nœud master contient les composantes suivantes :

1. Controller manager : ce composant gère les différentes boucles de contrôle qui régulent l'état du cluster Kubernetes. Chacune de ces boucles de contrôle connaît l'état
souhaité du cluster, l'état des objets qu'elle gère et surveille l'état actuel du cluster via le serveur API. Dans une boucle de contrôle, si l'état actuel des objets qu'elle gère ne correspond pas à l'état souhaité, la boucle de contrôle prend des mesures correctives pour s'assurer que l'état actuel est identique à l'état souhaité.

2. API server : il permet d'exécuter toutes les tâches administratives et fournit toutes les opérations sur le cluster à l'aide d'une interface (API, CLI).

3. Scheduler : il est responsable de la planification, la répartition, et le suivi de la charge de travail entre les nœuds esclaves.

4. ETCD : est utilisé pour stocker l'état du cluster ainsi que les informations de configuration pouvant être utilisées par chacun des nœuds du cluster.

Les composantes de « worker node »

1. Runtime : aide à exécuter les conteneurs d'applications encapsulés dans un environnement d'exploitation relativement isolé mais léger.

2. Kubelet : Il s'agit d'un petit service dans chaque nœud chargé de relayer les informations vers et depuis le master node. Il reçoit la définition du pod via différents moyens (principalement via le serveur API) et exécute les conteneurs associés au pod, Il s'assure également que les conteneurs qui font partie des pods sont fonctionnels à tout moment.

3. Kube proxy : Il s'agit d'un service proxy qui s'exécute sur chaque nœud et aide à rendre les services disponibles pour l'hôte externe.

II.4.1.4.2 Objets de Kubernetes

Ce sont des abstractions qu'utilise Kubernetes pour représenter l'état souhaité du
cluster, Ce modèle d'objet est très riche, il décrit :

- Quelles applications conteneurisées sont exécutées et sur quel nœud.
- La consommation des ressources d'application.
- Les différentes stratégies attachées aux applications, telles que les stratégies de redémarrage / mise à niveau, la tolérance aux pannes, etc.
- Comment maintenir l'état actuel du cluster selon l'état souhaité.

Ces objets sont :

1.Les pods : ce sont une collection logique d'un ou plusieurs conteneurs, qui :

- Sont programmés ensemble sur le même hôte.
- Partagent le même espace de noms réseau.
- Montent le même stockage externe.

2. Les Volumes : Ils permettent de stocker et de gérer les fichiers de l'application, en les partageant entre les pods. De plus ils conservent les fichiers en cas de crash d'un conteneur.

3. Les Replicasets : Il s'assure que le nombre spécifié de pod (replicas) est en cours d'exécution à un moment donné.

4. Les déploiements : Ils représentent la manière dont les pods sont déployés et en particulier la manière dont ils sont créés ou supprimés lors d'une montée en charge.

5. Les Services : C'est un ensemble logique de pods ouverts au reste du cluster Kubernetes. Par défaut, le pod est uniquement accessible par son adresse IP interne dans le cluster Kubernetes. Pour rendre le conteneur accessible de l'extérieur du réseau
Kubernetes, le pod doit être exposé en tant qu'un service Kubernetes.

6. Les Names-spaces : Kubernetes fournit un partitionnement des ressources qu'il gère en différents ensembles qui ne se chevauchent pas appelés « namespaces ».

7. Ingress : c'est un objet API qui gère l'accès externe aux services dans un cluster, il donne à ces services des URLs accessibles de l'extérieur.

- **Gestion des objets Kubernetes**

Kubernetes fournit des mécanismes qui permettent de gérer, sélectionner ou manipuler ses objets :

• **Labels :** Ce sont des paires key/value qui sont attachées à n'importe quel objet du système. Lorsqu'un service est défini, des labels sont utilisés pour sélectionner l'ensemble de pods vers lesquelles le trafic sera acheminé.

• **Selectors :** Ils permettent de sélectionner les objets de Kubernetes, contrairement aux labels la sélection est basée sur les valeurs d'attribut inhérentes à l'objet sélectionné.

II.5 Conclusion

La nouvelle génération 5G est sensée être le point clé pour arriver à la réalisation et à la mise en pratique des nouvelles technologies qui vont changer nos mode de vie et travail, tel que l'internet des objets (IOT), l'intelligence artificielle, et les voitures autonomes…etc…

Les opérateurs de téléphonie mobile continuent de vanter les attentes théoriques de nouveaux services que la 5G est censée annoncer, cette nouvelle technologie doit reposer sur la nouvelle approche « cloud native avec NFV ».

Dans ce chapitre nous avons présenté l'approche « cloud native avec NFV », ainsi que la manière dont elle favorise le déploiement des applications de télécommunication dans une infrastructure Cloud. Ce qui permet au réseau cœur 5G de répondre aux nouvelles demandes dans le secteur de télécommunication espérées par les utilisateurs.

Le chapitre prochain sera dédié à la mise en pratique de toutes ces notions théoriques.

Chapitre III

Implémentation et réalisation de l’application

III.1 Introduction

Nous avons présenté toute une étude théorique sur la nouvelle approche des applications « cloud native ». Cette approche est comme on l'avait dit ; considérée comme la clé de la conception et la mise en pratique des applications télécom des éseaux cœur 5G. Nous allons donc mettre ces notions théoriques en pratique.

Ce chapitre va présenter la mise en pratique de l'application « nginx » dans un cluster Kubernetes. Comme, le principe de fonctionnement de cette application est le même que celui d'une application 5G, elle a été donc choisie pour faire nos tests.

Nous allons aussi voire à travers ces tests de fonctionnement les bénéfices de « nginx » avec l'approche « cloud native » pour l'implémentation de ses fonctions réseau.

III.2 La mise en place de l'application « nginx »

La mise en place de l'application « nginx » dans un environnement Cloud native doit passer par les étapes suivantes :

III.2.1 Installation de Kubernetes

Pour créer un cluster kubernetes nous avons utilisé l'outil « Minikube », ce dernier facilite l'installation d'un cluster kubernetes. Nous avons procédé à l'installation de notre cluster comme suit :

1. Installation d'une version récente de VirtualBox pour exécuter Minikube
2. L'installation de « Minikube ».
3. Le démarrage de « Minikube » en utilisant la commande « minikube start » et la vérification de son bon état avec la commande « minikube status ».

4. Accès à Minikube.

La figure ci-dessous montre l'installation de kubernetes

Figure III.1 : Démarrage de Kubernetes

Un cluster Kubernetes en cours d'exécution est accessible via les méthodes suivantes :

- **Interface de ligne de commande (CLI) :** « kubectl » est l'outil de ligne de commande permettant d'exécuter des commandes sur les clusters Kubernetes, de déployer des applications, d'inspecter et de gérer les ressources et les applications du cluster Kubernetes.
- **Interface utilisateur graphique (GUI) :** « Kubernetes dashboard » fournit une interface utilisateur graphique pour interagir avec ses ressources et ses applications conteneurisées.
- **APIs :**

Pour accéder à notre cluster kubernetes, nous avons utilisé l'outil « Kubectl », Et on a téléchargé la dernière version v1.21.0 comme la figure suivante le montre :

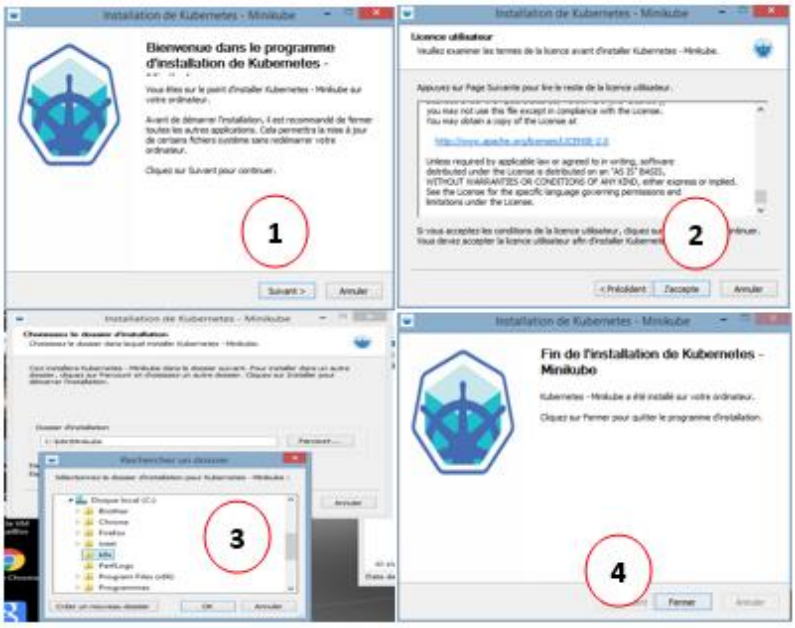

Figure III.2 : L'installation de l'outil Kubectl

Pour avoir des informations sur le cluster nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl cluster-info
I0605 15:50:36.356383    5984 request.go:668] Waited for 2.3661464s due to client-side throttling, not priority and fair
ness, request: GET:https://192.168.99.100:8443/apis/networking.k8s.io/v1?timeout=32s
Kubernetes control plane is running at https://192.168.99.100:8443
KubeDNS is running at https://192.168.99.100:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\B I P>
```

Figure III.3 : Vérification de l'état de cluster

L'affichage montre que le cluster a été bien créé, il contient le nœud master dont l'adresse IP est 192.168.99.100 attribué par un DHCP.

III.2.2 Déploiement de l'application

Pour assurer un bon déploiement de l'application « nginx », nous avons créé un fichier YAML de type déploiement dans lequel nous avons défini notre état souhaité pour le cluster, y compris l'image docker de l'application « nginx ». Notre fichier de déploiement YAML se présente comme ceci :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Figure III.4 : Fichier YAML de déploiement

Ce fichier « YAML » contient les spécifications suivantes :

apiVersion : Ceci spécifie la version de l'API de l'objet « déploiement Kubernetes ».

kind : il décrit le type d'objet à créer. Dans ce cas, c'est un objet de type déploiement.

metadata : C'est l'ensemble de données permettants d'identifier de manière unique le déploiement, il s'agit essentiellement du nom de déploiement.

spec : Sous spec, nous avons déclaré l'état et les caractéristiques de déploiementsouhaitées, nous avons spécifié le nombre de réplicas, le nom de l'image docker de « nginx » et sa version nginx 1.14.2

Pour exécuter ce déploiement dans le cluster, nous avons transmis le fichier YAML comme une spécification d'objet en utilisant l'option **-f** avec la commande **kubectl apply** :

```
PS C:\Users\B I P> kubectl apply -f D:\Deployment1.yml
deployment.apps/nginx-deployment created
PS C:\Users\B I P>
```

Figure III.5 : La création du déploiement

La figure ci-dessus montre que notre déploiement « nginx-deployment » a été bien créé.

Pour voir les détails de ce déploiement, nous avons utilisé la commande suivante :

Figure III.6 : Détails de déploiement

Le résultat obtenu (présenté dans la figure ci-dessus montre) que ce déploiement contient 2 pods qui sont en cours d'exécution, chaque pod a une adresse IP unique dans le cluster, 1 Replicaset qui pointe vers ces 2 pods exécutants l'image « nginx ». Donc l'état actuel du cluster correspond à l'état souhaité.

III.2.3 Tests Fonctionnels

Notre application est prête. Maintenant, nous allons effectuer un ensemble de test pour s'assurer de son fonctionnement.

III.2.3.1 Self Healing

Parmi les caractéristiques de Kubernetes, il vérifie de manière constante l'état du cluster pour s'assurer qu'il correspond à l'état souhaité par l'utilisateur, cela se fait à travers des boucles de contrôle.

Si un objet tombe en panne ou ne fonctionne plus, Kubernetes réagit immédiatement pour le corriger afin de garder cet état souhaité sans interventions de l'utilisateur.

Pour montrer le « Self healing » de Kubernetes, nous avons effectué ce test comme suit :

```
PS C:\Users\B I P> kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           4m37s
PS C:\Users\B I P>
```

Figure III.7 : Voir le déploiement

La figure précédente montre que notre déploiement nginx-deployment qu'il est en cours d'exécution. Pour voir les pods de ce déploiement nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-66b6c48dd5-jmgtd   1/1     Running   0          5m24s
nginx-deployment-66b6c48dd5-x7826   1/1     Running   0          5m24s
PS C:\Users\B I P>
```

Figure III.8 : Voir les Pods de déploiement

La Figure ci-dessus montre que notre application nginx est déployée sur les 2 pods que nous avons spécifiés dans le fichier YAML.

Maintenant, pour tester le comportement de Kubernetes, nous allons supprimé un pod parmi ces 2 pods en utilisant la commande montrée dans la figure suivante :

```
PS C:\Users\B I P> kubectl delete pod nginx-deployment-66b6c48dd5-jmgtd
pod "nginx-deployment-66b6c48dd5-jmgtd" deleted
PS C:\Users\B I P>
```

Figure III.9 : Suppression d'un Pod

Le pod précisé a été bien supprimé.

Pour voir comment Kubernetes va réagir face à ce problème nous allons utiliser la commande suivante :

```
PS C:\Users\B I P> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-66b6c48dd5-2wg9p   1/1     Running   0          52s
nginx-deployment-66b6c48dd5-x7826   1/1     Running   0          13m
PS C:\Users\B I P>
```

Figure III.10 : Vérification des Pods

Cette figure montre qu'un nouveau pod vient juste d'être créé par kubernetes et que notre déploiement contient 2 pods au lieu d'un seul pod.

Cela signifie que Kubernetes a rapidement recréé un nouveau pod pour remplacer celui que nous avons supprimé, et cela d'une manière automatique (sans intervention de l'utilisateur).

III.2.3.2 ReplicaSet

La fonction d'un ReplicaSet est de garantitir qu'un nombre spécifié de réplicas de Pod soient exécutés à un moment donné. On peut changer le nombre de replicas dans notre fichier YAML, ce qui implique un changement de nombre de pods dans notre deploiement.

Cette opération est montrée dans la figure suivante :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # Update the replicas from 2 to 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Figure III.11 : Fichier YAML avec 4 Replicas

Pour exécuter ce déploiement dans le cluster, Nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl apply -f D:\DeploymentReplicas.yml
deployment.apps/nginx-deployment configured
PS C:\Users\B I P>
```

Figure III.12 : Exécution de déploiement

La figure présentée montre, le nouveau déploiement a été bien configuré dans le cluster. Pour voir ce déploiement (Changement de nombre de replicas) nous allons utiliser la commande suivante :

```
PS C:\Users\B I P> kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-66b6c48dd5-2wg9p    1/1     Running   0          5m54s
pod/nginx-deployment-66b6c48dd5-99dcs    1/1     Running   0          2m32s
pod/nginx-deployment-66b6c48dd5-jc5x7    1/1     Running   0          2m32s
pod/nginx-deployment-66b6c48dd5-x7826    1/1     Running   0          18m

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   80m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment    4/4     4            4           18m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-66b6c48dd5    4         4         4       18m
PS C:\Users\B I P>
```

Figure III.13 : Voir les Pods

On remarque bien que Kubernetes a rajouté les 2 pods , et que notre application nginx est déployée sur les 4 pods que nous avons spécifiés dans le fichier YAML.

```
PS C:\Users\B I P> kubectl get all -o wide
NAME                                     READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
pod/nginx-deployment-66b6c48dd5-2wg9p    1/1     Running   0          14m     172.17.0.3   minikube   <none>           <none>
pod/nginx-deployment-66b6c48dd5-9g6bb    1/1     Running   0          3m28s   172.17.0.6   minikube   <none>           <none>
pod/nginx-deployment-66b6c48dd5-jc5x7    1/1     Running   0          10m     172.17.0.5   minikube   <none>           <none>
pod/nginx-deployment-66b6c48dd5-x7826    1/1     Running   0          27m     172.17.0.4   minikube   <none>           <none>

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE   SELECTOR
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   88m   <none>

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES         SELECTOR
deployment.apps/nginx-deployment    4/4     4            4           27m   nginx        nginx:1.14.2   app=nginx

NAME                                           DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES         SELECTOR
replicaset.apps/nginx-deployment-66b6c48dd5    4         4         4       27m   nginx        nginx:1.14.2   app=nginx,pod-template-hash=66b6c48dd5
PS C:\Users\B I P>
```

Figure III.14 : Etat de déploiement

Comme la figure précédente le montre, notre cluster contient maintenant 4 pods, donc il est à jour avec notre **nouveau** fichier YAML.

Si l'un de ces pod tombe en panne ou va être supprimé, Kubernetes va recréer un nouveau pod.

Pour confirmer ça on a supprimé un pod et voir combien de pods restent, en exécutant la commande suivante :

```
PS C:\Users\B I P> kubectl delete pod nginx-deployment-66b6c48dd5-99dcs
pod "nginx-deployment-66b6c48dd5-99dcs" deleted
```

Figure III.15 : Suppression d'un pod

Le pod a été supprimé avec succès.

Pour vérifier les pods restants dans notre cluster, nous allons utiliser la commande suivante :

```
PS C:\Users\B I P> kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-66b6c48dd5-2wg9p   1/1     Running   0          12m
pod/nginx-deployment-66b6c48dd5-9g6bb   1/1     Running   0          91s
pod/nginx-deployment-66b6c48dd5-jc5x7   1/1     Running   0          9m2s
pod/nginx-deployment-66b6c48dd5-x7826   1/1     Running   0          25m

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   86m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment    4/4     4            4           25m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-66b6c48dd5    4         4         4       25m
PS C:\Users\B I P>
```

Figure III.16 : Vérification des pods

Il existe toujours 4 Pods dans le cluster. Cette figure montre qu'un nouveau pod vient juste d'être créé par kubernetes et que notre déploiement contient toujours 4 pods au lieu de 3 pods. Cela signifie que Kubernetes a rapidement recréé un nouveau pod pour remplacer celui que nous avons supprimé, et cela d'une manière automatique (sans intervention de l'utilisateur).

III.2.3.3 Rolling Update

Dans le meme fichier YAML précédent, nous avons modifié la version de l'image nginx de (nginx-1.14.2) vers la version la plus récente (nginx-1.16.1).

L'opération est présentée dans la figure ci-dessous :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16.1 # Update the version of nginx from 1.14.2 to 1.16.1
        ports:
        - containerPort: 80
```

Figure III.17 : Fichier YAML de mise à jour

Pour exécuter ce déploiement dans le cluster, nous allons utiliser la commande suivante :

```
PS C:\Users\B I P> kubectl apply -f D:\DeploymentUpdate.yml
deployment.apps/nginx-deployment configured
PS C:\Users\B I P>
```

Figure III.18 : Exécution et déploiement de la mise à jour

Notre déploiement a bien été exécuté.

Pour voir ce déploiement (la mise à jour de l'application), nous avons utilisé la commande montrée dans **la figure III.19 :**

```
PS C:\Users\B I P> kubectl get all -o wide
NAME                                    READY   STATUS        RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
pod/nginx-deployment-559d658b74-8cxbr   1/1     Running       0          5s      172.17.0.3   minikube   <none>           <none>
pod/nginx-deployment-559d658b74-m7jlm   1/1     Running       0          7s      172.17.0.5   minikube   <none>           <none>
pod/nginx-deployment-559d658b74-nbcbm   1/1     Running       0          8m19s   172.17.0.6   minikube   <none>           <none>
pod/nginx-deployment-559d658b74-xkq4n   1/1     Running       0          8m19s   172.17.0.7   minikube   <none>           <none>
pod/nginx-deployment-66b6c48dd5-2wg9p   0/1     Terminating   0          29m     172.17.0.3   minikube   <none>           <none>
pod/nginx-deployment-66b6c48dd5-jc5x7   0/1     Terminating   0          25m     172.17.0.5   minikube   <none>           <none>
pod/nginx-deployment-66b6c48dd5-x7826   0/1     Terminating   0          41m     172.17.0.4   minikube   <none>           <none>

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE    SELECTOR
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   103m   <none>

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES         SELECTOR
deployment.apps/nginx-deployment    4/4     4            4           41m   nginx        nginx:1.16.1   app=nginx

NAME                                          DESIRED   CURRENT   READY   AGE     CONTAINERS   IMAGES         SELECTOR
replicaset.apps/nginx-deployment-559d658b74   4         4         4       8m19s   nginx        nginx:1.16.1   app=nginx,pod-template-hash=559d658b74
replicaset.apps/nginx-deployment-66b6c48dd5   0         0         0       41m     nginx        nginx:1.14.2   app=nginx,pod-template-hash=66b6c48dd5
PS C:\Users\B I P>
```

Figure III.19 : Détails de déploiement

La figure ci-dessus montre que notre nouveau déploiement exécute la version récente de l'image nginx 1.14.2.

A ce niveau-là nous avons remarqué qu'un nouveau replicaset a été créé, ce replicaset contient 4 pods exécutant la version récente de l'image nginx 1.16.1 tandis que l'ancien replicaset ne contient aucun pod, cela signifie que les pods ont été supprimés de l'ancien replicaset et recrées dans le nouveau replicaset un par un sans interruption de service (les pods sont toujours en cours d'exécution).

Donc notre déploiement nginx-deployment est exécuté sur ce nouveau replicaset.

De plus, nous avons remarqué que l'ancien replicasets (avec la version de l'image 1.14.2) n'a pas été supprimé.

Pour voir l'état du Rolling-update nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl rollout status deployment nginx-deployment
deployment "nginx-deployment" successfully rolled out
PS C:\Users\B I P>
```

Figure III.20 : Etat de mise à jour

Pour voir l'historique de la mise à jour de notre application, nous avons utilisé la commande présentée dans la figure suivante :

```
PS C:\Users\B I P> kubectl rollout history deployment nginx-deployment
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
1         <none>
2         <none>

PS C:\Users\B I P>
```

Figure III.21 : Historique de déploiement

La figure montre les 2 révisions du Rolling-update, l'ancien déploiement est représenté par la première révision, tandis que le nouveau déploiement est représenté par la deuxième révision.

- **Utilisation de Kubectl**

Nous avons vu dans le Rolling-update précédent que la modification a été faite dans fichier YAML. Cependant, il existe une deuxième façon pour ce même but.

Kubernetes permet d'effectuer un Rolling-update via Kubectl. Pour ce faire nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl set image deployment nginx-deployment nginx=nginx:1.15
deployment.apps/nginx-deployment image updated
PS C:\Users\B I P>
```

Figure III.22 : Rolling Update via Kubectl

Cette figure montre que nous avons changé la version de l'image nginx 1.16.1, vers la version nginx 1.15.

Pour voir l'état de ce déploiement nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl get replicasets -o wide
NAME                          DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES         SELECTOR
nginx-deployment-559d658b74   0         0         0       16m    nginx        nginx:1.16.1   app=nginx,pod-template-hash=559d658b74
nginx-deployment-66b6c48dd5   0         0         0       50m    nginx        nginx:1.14.2   app=nginx,pod-template-hash=66b6c48dd5
nginx-deployment-698676d7f8   4         4         4       2m11s  nginx        nginx:1.15     app=nginx,pod-template-hash=698676d7f8
PS C:\Users\B I P>
```

Figure III.23 : Etat de déploiement avec l'image 1.15

Cette figure montre que le troisième replicaset (avec la version 1.15) a été bien créé.

Pour voir les détails de ce déploiement nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl get all -o wide
NAME                                   READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
pod/nginx-deployment-698676d7f8-646bq  1/1     Running   0          2m55s   172.17.0.4   minikube   <none>           <none>
pod/nginx-deployment-698676d7f8-dsk4s  1/1     Running   0          60s     172.17.0.5   minikube   <none>           <none>
pod/nginx-deployment-698676d7f8-jdn7m  1/1     Running   0          58s     172.17.0.8   minikube   <none>           <none>
pod/nginx-deployment-698676d7f8-zkz6n  1/1     Running   0          2m55s   172.17.0.3   minikube   <none>           <none>

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE   SELECTOR
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   112m  <none>

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES       SELECTOR
deployment.apps/nginx-deployment   4/4     4            4           51m   nginx        nginx:1.15   app=nginx

NAME                                         DESIRED   CURRENT   READY   AGE     CONTAINERS   IMAGES         SELECTOR
replicaset.apps/nginx-deployment-559d658b74  0         0         0       17m     nginx        nginx:1.16.1   app=nginx,pod-template-hash=559d658b74
replicaset.apps/nginx-deployment-66b6c48dd5  0         0         0       51m     nginx        nginx:1.14.2   app=nginx,pod-template-hash=66b6c48dd5
replicaset.apps/nginx-deployment-698676d7f8  4         4         4       2m55s   nginx        nginx:1.15     app=nginx,pod-template-hash=698676d7f8
PS C:\Users\B I P>
```

Figure III.24 : Détails de Déploiement avec l'image 1.15

Cette figure montre que le nouveau replicaset contient 4 pods exécutants la nouvelle version de l'image « nginx 1.15 », cela signifie que les pods ont été supprimés de l'ancienne révision « nginx 1.16.1 » et sont recréés dans la nouvelle révision (la version 1.15) un par un **sans interruption de service**.

Pour voir l'historique des mises à jour, nous avons exécuté la commande suivante :

```
PS C:\Users\B I P> kubectl rollout history deployment nginx-deployment
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
3         <none>

PS C:\Users\B I P>
```

Figure III.25 : Historique des mises à jour

Cette figure montre les trois révisions du Rolling-update. Ce qui signifie que les anciennes révisions n'ont pas été supprimées mais qu'elles ne soient pas utilisées. Kubernetes les garde dans le cas où l'utilisateur souhaite revenir sur une révision précédente.

Pour revenir à la révision précédente nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl rollout undo deployment nginx-deployment
deployment.apps/nginx-deployment rolled back
PS C:\Users\B I P>
```

Figure III.26 : Revenir sur une révision précédente

Pour vérifier que les pods sont exécutés sur la deuxième révision « nginx1.16.1 » nous avons utilisé la commande suivante :

```
PS C:\Users\B I P> kubectl get deployments -o wide
NAME               READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES         SELECTOR
nginx-deployment   4/4     4            4           54m   nginx        nginx:1.16.1   app=nginx
PS C:\Users\B I P>
```

Figure III.27 : Vérification d'une révision

La figure montre que les pods sont supprimés de la version de l'image nginx 1.15, et sont recréés dans la version précédente nginx 1.16.1, un par un sans interruption de service.

Dans le cas où la version de l'image de l'application est **inexistante** ou **fausse**, le Rolling- update **ne marchera pas**.

Pour le montrer, nous avons utilisé la commande suivante en spécifiant une version inexistante de l'image nginx :

```
PS C:\Users\B I P> kubectl set image deployment nginx-deployment nginx=nginx:0.0.0
deployment.apps/nginx-deployment image updated
PS C:\Users\B I P>
```

Figure III.28 : Exécution de l'image nginx 0.0.0

La figure montre que notre déploiement à bien configuré la version « nginx 0.0.0 ».

```
PS C:\Users\B I P> kubectl get replicasets -o wide
NAME                          DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES         SELECTOR
nginx-deployment-559d658b74   3         3         3       27m    nginx        nginx:1.16.1   app=nginx,pod-template-hash=559d658b74
nginx-deployment-66b6c48dd5   0         0         0       60m    nginx        nginx:1.14.2   app=nginx,pod-template-hash=66b6c48dd5
nginx-deployment-698676d7f8   0         0         0       12m    nginx        nginx:1.15     app=nginx,pod-template-hash=698676d7f8
nginx-deployment-8565f5669    2         2         0       4m56s  nginx        nginx:0.0.0    app=nginx,pod-template-hash=8565f5669
PS C:\Users\B I P>
```

Figure III.29 Exécution de l'image nginx 0.0.0 inexistante

La figure III.29 montre que le Rolling-update n'a pas marché. e Rolling-update met à jour les pods un par un et ne fait tomber qu'un seul pod à la fois, et vu que la version de l'image est inexistante, le pod indisponible n'a pas été mis à jour donc le Rolling-update n'est pas passé vers le prochain pod.

III.3 Conclusion

Au cours de ce chapitre nous avons montré comment sera la conception des fonctions réseaux cœur 5G à travers las applications « cloud native ».

Nous avons commencé par le déploiement de l'application nginx, qui présente ces NF sous formes de microservices, conçus et déployés indépendamment

les uns des autres. Ces microservices sont exécutés dans des conteneurs Docker et orchestrés par Kubernetes.

Et à la fin de ce chapitre nous avons effectué un ensemble de tests de fonctionnement pour approuver que les applications du réseau 5G bénéficient de l'approche « cloud native ».

Conclusion générale

A nos jours, il est devenu possible de changer notre façon de penser envers l'approche « cloud native ». Car, le développement de ses applications est basé sur les pratiques DevOps. Ses conteneurs, ses microservices, et son cloud hybride.

Aujourd'hui, nous avons la possibilité d'adopter l'approche « cloud native » et concevoir des services plus rapidement, et de manière plus sécurisée.

L'approche « cloud native » permet de créer des applications sous forme de services plus petits appelés microservices, indépendants et faiblement couplés, et de les exécuter sur des plates-formes conteneurisées, ce qui fournit aux développeurs d'applications modernes un moyen d'itérer et de déployer rapidement des mises à jour logicielles plusieurs fois par jour dans un environnement rapide, distribué et isolé permettant d'exploiter pleinement les avantages du cloud.

Kubernetes est rapidement devenu la solution la plus populaire d'orchestration de ces applications conteneurisées sur un cluster de machines en utilisant des méthodes de prédictibilité, de scalabilité et de haute disponibilité.

Il prend en charge Docker qui est le standard le plus recommandé à la création et l'exécution d'applications conteneurisées.

En outre, la guerre pour la 5G a commencé et les plans d'architecture réseau 5G commencent à prendre forme, ils se tournent distinctement vers le cloud.

Certains opérateurs de réseau réalisent qu'ils devront créer des réseaux de manière très différente pour prendre en charge les services et applications basés sur la 5G ce qui a conduit à la virtualisation du réseau qui sera un élément crucial de la conception future du réseau 5G, mais cela ne ressemblera pas aux premières implémentations de la virtualisation des fonctions de réseau (NFV) mais plutôt les applications cloud native.

Ce projet avait pour objectif de montrer le rôle important de l'approche « cloud native » dans la conception des applications du réseau cœur 5G. Pour ce faire,

il suffit d'orchestrer un cluster Kubernetes contenant deux types de nœuds : « master node », qui est chargé de l'exécution du système d'orchestration Kubernetes qui commande et contrôle toutes les autres machines du cluster, et « worker nodes », ce dernier exécutent les applications 5G.

Dans notre cas nous avons choisi l'application nginx qui est résponsable du load balancing dans un réseau cœur 5G. Ces applications sont conteneurisées et déployées dans des pods ou chaque pod représente une seule instance d'une application ou d'un processus en cours d'exécution sur Kubernetes.

[1] SEIDE.G ,« Planification d'un réseau de quatrième génération à Partir D'un Réseau De Troisième Génération », Mémoire en vue de l'obtention du diplôme de maîtrise des sciences appliquées (génie informatique), Université de MONTREAL, 2011.

[2] Guillaume Di Maiol (Directeur des opérations de la société JeChange)**,** Article « Tout savoir sur la 4G », France, 2020.

[3] Kamran Sharief , « What is Network Functions Virtualization (NFV) » , 2019.

[4] « Accès radio réseau 5G », Extrait de l'article « Brochure.com », France, 2021.

[5] « Network slicing », Article « viavi solution.com », France, 2022.

[6] « VMware, INC 2022 », Article publié sur le site VMware.com, France.

[7] « Chakray.com », Article « Que sont les microservices ? Définition, caractéristiques, avantages et inconvénients », France,2020.

[8] HAMDANI Nadir, KERROUM Salima, OUALLOUCHE Nacera, « Etude et comparaison des failles de sécurité d'Open Strack et Open Nebula », Mémoire en vue de l'obtention de diplôme de Master professionnel en ingénierie des systèmes d'information, Université de Tizi Ouzzou, année 2018-2019.

[9] Guillaume SAYER, « Introduction à la conteneurisation avec Docker »,11 octobre 2015.

[10] Myagile Partner, « Docker tutorial pdf », 2019.

[11] Jean-Sébastien AVRILA, « Qu'est-ce qu'une application "Cloud Native" ? », le 15 aout 2017.

[12] Rajesh RADHAKRISHNAN, « Docker Architecture And Components », le 04 Aout 2018.

[13] Margaret Rouse, « Kubernetes », mars 2016.

[14] Samarpit, « Understanding Kubernetes Architecture », le 22 Mai 2019.