الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي والبحث 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 Télécommunication
Spécialité Réseaux Et Télécommunications

Cloud Native pour les applications 5G–VNFs évolutives basées sur des micro-services avec Kubernetes

Présenté par :

Mme. OUISSOU Wafa

Encadré par :

Mme. DIFALLAH Rania

M. AIT SAADI Hocine

Année Universitaire 2020-2021

Remerciements

Je remercie tout d'abord « Allah » de m'avoir donné le courage d'entamer et de finir ce mémoire dans de bonnes conditions.

Je remercie mes très chers parents, qui ont toujours été là pour moi. Je remercie mes sœurs Ikram et Hadil, et mon frère Mohamed, pour leurs encouragements.

Je remercie vivement ma directrice de mémoire L'ingénieure Mme. DIFALLAH Rania, d'avoir encadré ce travail avec beaucoup de compétences
Merci pour votre indéfectible disponibilité, votre rigueur scientifique et la confiance que vous m'avez accordée au cours de l'élaboration de ce mémoire, Ainsi que toute l'équipe d'ATM Mobilis

Je remercie également les membres du jury d'avoir accepté d'évaluer ce travail

Mes remerciements vont également à mon professeur M. AIT SAADI Hocine, pour les efforts consentis afin de m'assurer de meilleures conditions de travail

Je remercie tous les enseignants qui ont contribué à ma formation de la graduation à la poste graduation
À tous ces intervenants, je présente mes remerciements, mon respect et ma gratitude.

Dédicaces

A mes très chers Parents

Je dédie ce mémoire à mes parents, pour l'amour qu'ils m'ont toujours donné, leurs encouragements et toute l'aide qu'ils m'ont apportée durant mes études.

Aucun mot, aucune dédicace ne pourrait exprimer mon respect, ma considération, et mon amour pour les sacrifices qu'ils ont consentis pour mon instruction et mon bien-être.

Trouvez ici, chère mère et cher père, dans ce modeste travail, le fruit de tant de dévouement et de sacrifices ainsi que l'expression de ma gratitude et de mon profond amour.

Puisse Dieu leur accorder santé, bonheur, prospérité et longue vie afin que je puisse un jour combler de joie leurs vieux jours.

A mon Frère et Sœurs

Mon cher frère et sœurs pour leur dévouement, leur compréhension et leur grande tendresse, qui en plus de m'avoir encouragé tout le long de mes études, m'ont consacré beaucoup de temps et disponibilité, et qui par leur soutien, leurs conseils et leur amour, m'ont permis d'arriver jusqu'à ici car ils ont toujours cru en moi

Merci d'avoir toujours soutenu et merci pour tous les bons moments passés ensemble, et ce n'est pas fini. A ma famille et toutes les personnes que j'aime

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 يسهل إدارتها و نشرها على بنية أساسية سحابية موزعة تعتمد على الحاويات باستخدام نظام الافتراضية .

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

كلمات المفاتيح: الوظيفة الأساسية لشبكة الجيل الخامس 5G ، نهج السحابة الأصلية، نظام الإفتراضية، تقنية الحاوية.

Résumé : Les noyaux "NF 5G" seront construits à travers des « Cloud native applications ». Le principe consiste à décomposer un logiciel en éléments plus petits appelés Micro-services, qui sont plus faciles à gérer et à déployer sur une infrastructure Cloud distribuée basée sur des conteneurs en utilisant la virtualisation. L'approche Cloud native utilise des mécanismes Kubernetes telles que la découverte de services. Le but principal de ce projet est de présenter la technologie Cloud native et les principes de sa conception, pour répondre aux exigences du réseau cœur 5G.

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

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, Kubernetes.

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

I

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

L

LTE: Long Term Evolution
LXC: Linux Container

M

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

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

V

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

U

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

Table des matières

Liste des figures

Introduction générale

Ces dernières années, nous avons observé une expansion progressive et continue des réseaux téléphoniques sans fil concentrés sur l'être humain. L'apparition de ces réseaux a encouragé les chercheurs à réfléchir à de nouvelles solutions pour assurer une évaluation efficace et une conception appropriée des protocoles de communication. En fait, ces réseaux sont soumis à de multiples limites, telles que: Manque d'infrastructure, de topologie dynamique, de ressources limitées et de qualité de service, ainsi que la sécurité des informations préliminaires.

Les opérateurs souhaitent déployer leurs réseaux et introduire de nouveaux et divers services plus rapidement et à moindre cout. Pour cela, ils doivent avoir des environnements de travail flexibles. C'est ici qu'intervient la virtualisation des fonctions réseaux (qu'on va étudier en détail par la suite) en permettant aux fournisseurs de services la possibilité d'évoluer plus rapidement et plus aisément au sein des nouvelles infrastructures telles que le LTE (Long Term Evolution). Elle permet aussi d'automatiser et de donner une certaine intelligence au réseau et d'utiliser plus efficacement les ressources pour accroitre ou diminuer les services. La virtualisation va apporter un plus en ce qui concerne la réduction des couts et les délais de commercialisation des services réseaux.

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.

Les bénéfices de la 5G sont vastes pour les sociétés comme pour les consommateurs. La vitesse est le plus grand atout des réseaux de cinquième génération. Les débits en 5G sont 10 fois plus élevés que ceux de la 4G. Un univers à ultra haut débit s'appelle IoT « l'internet des objets connectés », qui utilise un débit fort et demande un temps de latence faible. Les outils de surveillance ou la vidéo immersive en temps réel sont à inclure dans l'IoT haut débit.

La 5G est donc supposée apporter des avancées à un réseau de télécommunications toujours plus exigeant, sur lequel l'architecture NFV constitue la base, cette approche vise à la création des partitions du réseau isolées sur une infrastructure réseau partagée afin de permettre une automatisation de bout en bout et de prendre en charge des services dédiés à un spectre spécifique ou à des cas d'utilisateurs (découpage en tranches de réseau, par exemple).

Alors que les plans d'architecture réseau 5G commencent à prendre forme, ils se tournent distinctement vers le Cloud. L'architecture «Cloud native» est considérée comme l'architecture principale pour la conception et le développement d'applications de réseau dans le futur. Elle permet aux organisations de créer et d'exécuter des applications évolutives dans des environnements modernes et dynamiques tels que les Clouds publics, privés et hybrides. Le fait que les applications seront déployées dans des conteneurs cela constitue une étape cruciale pour faire évoluer les innovations et les cas d'utilisation. De plus les applications monolithiques, seront fragmentées pour créer des micro-services pouvant être développées, mises à l'échelle, corrigées indépendamment et communiquant entre eux via des API riches. Kubernetes a évolué pour gérer de telles charges de travail en fournissant une orchestration dynamique pour les applications.

La technologie « Cloud native » présente les avantages d'efficacité, de flexibilité et d'ouverture par rapport aux architectures de réseau conventionnelles. Elle est devenue le meilleur choix pour les opérateurs d'ajouter ou de remplacer leurs périphériques réseaux centraux.

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

Le présent mémoire s'articule autour de trois chapitres :

Le premier chapitre donnera une introduction qui contient une présentation des réseaux 4G et leur évolution vers les réseaux 5G.

Le deuxième chapitre sera dédié à la description de la technologie « Cloud native » et les principes de sa conception ainsi que ses avantages dans le cœur du réseau 5G, ainsi que les fonctions de virtualisation des 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 et les résultats obtenus.

Chapitre I

Introduction aux réseaux 5G

Chapitre I Introduction aux réseaux 5G

Introduction

La 4G/LTE a permis des services hauts débit mobiles utilisables, et la 5G devrait désormais apporter de nouveaux cas d'utilisation et modèles commerciaux aux consommateurs, aux entreprises et aux industries. Voitures autonomes, robotique avancée, objets connectés, gestion de l'énergie en temps réel. De nombreux usages recherchent des réseaux denses, rapides et accessibles pour les gérer. Dans ce chapitre Introduction aux réseaux 5G, nous présenterons l'évolution du réseau de cinquième génération et les concepts clés qui permettent de mettre à niveau son infrastructure.

I.1 Les réseaux de la quatrième génération « 4G » LTE

La 4G constitue la quatrième génération des technologies de téléphonie mobile. Elle repose sur la nouvelle norme « LTE » ou Long Term Evolution, et succède directement à la technologie 3G et à la 3G+, avec un débit plus élevé.

I.1.1 L'architecture d'un réseau LTE

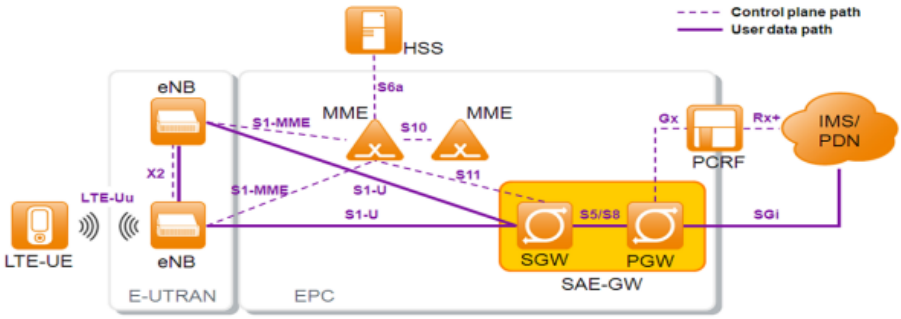

Figure 1.I : L'architecture d'un réseau LTE

L'architecture générale du système LTE est composée de deux entités principales comme la figure ci-dessus illustre.

- 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 Partie accès radio e-utran (evolved-utran)

La partie radio du réseau est appelée "E-UTRAN" et 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.1.1.2 Partie réseau cœur epc (evolved packet core)

L'EPC qui est le cœur du réseau, constitué principalement de :

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

L'UE communique avec l'EPC par le biais de l'E-UTRAN. Lorsqu'un UE est allumé, l'EPC est responsable de l'authentification et l'établissement de la connexion nécessaire pour toute la communication. A la différence de l'UMTS, le LTE a une architecture dite toute IP, qui supporte uniquement les données à commutation par paquet, Le réseau principal LTE est appelé EPC (Evolved Packet Core) est constitué des cinq nœuds suivants :

- **MME :** est le nœud de contrôle du réseau EPC. Il est responsable de la signalisation, de la mobilité et de la sécurité.
- **S-GW :** Il interconnecte également 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.

I.2 Virtualisation des fonctions réseaux «NFV»

I.2.1 Définition

La virtualisation des fonctions réseaux (NFV) est une approche réseau en développement qui permet le remplacement d'appareils matériels dédiés et coûteux tels que les routeurs, pare-feu avec des périphériques réseaux basés sur des logiciels qui fonctionnent comme des machines virtuelles sur les serveurs standard de l'industrie.

Les administrateurs réseau gèrent les fonctions d'orchestration et d'administration via un système d'exploitation qui coordonne les périphériques virtuels qui s'exécutent sur un réseau. Comme les machines virtuelles, les appareils virtuels peuvent être sélectionnés et déployés en fonction des besoins du réseau. NFV remplace le matériel réseau coûteux et dédié par un équipement logiciel.

I.2.2 L'architecture NFV

L'architecture NFV définie par l'ETSI est représentée sur la **figure 2.I**. La couche horizontale VNF correspond aux fonctions réseaux virtualisées VNF (Virtualised Network Function). Il s'agit de machines virtuelles (VM) fonctionnant sur l'infrastructure NFV (NFVI). L'infrastructure NFVI est une infrastructure physique de stockage, de calcul et de réseau.

La gestion et l'orchestration des VM sont sous la responsabilité de la fonction MANO (Management and orchestration). La fonction MANO doit gérer les ressources de l'infrastructure NFVI et la durée de vie des fonctions virtuelles en fonctionnement sur l'infrastructure NFVI (création et allocation des VMs).

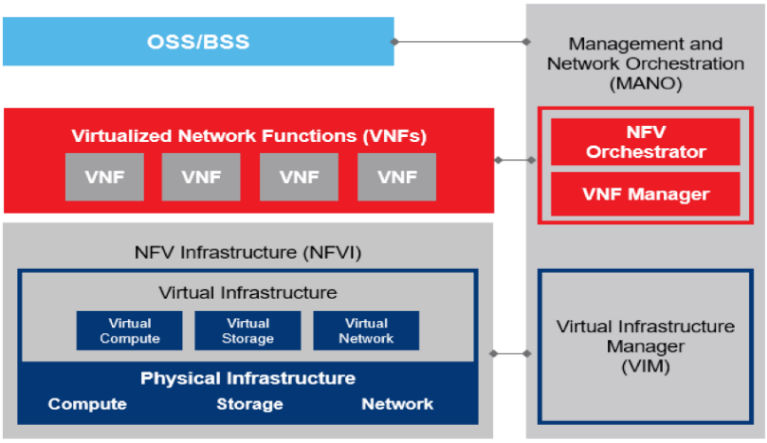

Figure 2.I : L'architecture NFV

L'architecture NFV est constituée par :

Le service OSS / BSS, qui doit pouvoir transmettre des informations sur les profils d'utilisateurs, les factures, etc. à l'unité NFV M&O.

L'infrastructure NFV, NFVI fournissait des ressources matérielles (serveurs, cartes électroniques, etc.) et des logiciels de virtualisation. Par conséquent, NFVI comprend :

- Interface matérielle (stockage, réseau, informatique).
- Interface virtuelle (stockage, réseau, informatique).
- Couche de virtualisation entre matériel et logiciel.

VNF (Virtual Network Function), correspond à la fonction de réseau virtuel qui peut être exécutée sur le périphérique NFVI.

NFV M&O (Management and Orchestration), pour la gestion des services réseau de bout en bout, elle se compose de trois modules :

- NFV Orchestrator : l'entité d'orchestration est responsable du cycle de vie du service réseau au niveau logiciel et matériel sur plusieurs domaines en contrôlant le VIM de chaque domaine.
- Manager en charge du cycle de vie VNF (VNFM).
- Le gestionnaire (VIM) responsable de la gestion des ressources NFVI dans le domaine.

I.2.3 Les Avantages de la technologie NFV

La NFV permet aux opérateurs réseau de fournir dynamiquement de nouveaux services, sans qu'il soit nécessaire d'installer de nouveaux éléments matériels, et donc NFV présente des avantages importants, notamment :

- ✓ La faible consommation d'énergie du réseau.
- ✓ Réduction du coût d'entretien du réseau.
- ✓ Mises à jour réseau plus simples et plus rapides.
- ✓ Préparation de l'architecture 5G.

I.3 Les Réseaux de la cinquième génération 5G

I.3.1 Définition

Le réseau 5G englobe un ensemble de technologies correspondant à la cinquième génération du standard pour la téléphonie mobile. La 5G devrait permettre de gérer le nombre toujours plus grand d'appareils connectés. Le réseau 5G réduira considérablement les temps de latence à 1 ms contrairement aux 30-40 ms observés aujourd'hui.

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

Dans la technologie 4G, le « cœur » (core) du réseau est la partie centralisée du système qui utilise des ordinateurs pour trier et traiter toutes les données. En gros, tant que le cœur est protégé, les données le sont aussi. Avec la 5G, le traitement des données sera plus proche des utilisateurs et de leurs appareils. L'augmentation de la puissance de traitement dans toutes les parties du réseau rend la 5G plus rapide.

Des technologies comme le CUPS, Network slicing et le Cloud distribué permettent un changement drastique dans l'architecture réseau.

I.3.2.1 CUPS « Control User Plane Separation »

Le CUPS propose de séparer les entités SGW et PGW. Le contrôleur se nomme SGW-C et PGW-C et le plan de données deviennent les entités SGW-U et PGW-U.

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

Cela permet un déploiement et une exploitation réseau flexible, grâce à une mise à l'échelle indépendante entre les fonctions du plan de contrôle et du plan d'utilisateur.

La figure ci-dessous montre l'évolution de l'architecture 4G EPC vers l'architecture de réseau cœur 5G.

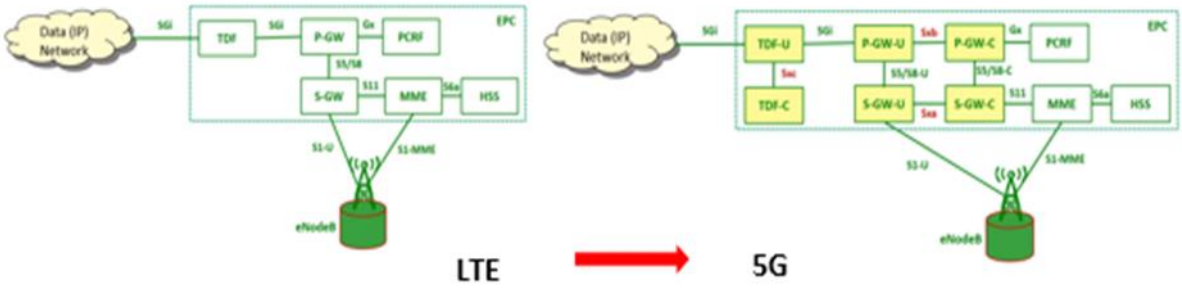

Figure 3.I : L'évolution de l'architecture 4G EPC à 5G

I.4 L'architecture de réseau 5G

I.4.1 Réseau cœur 5G « 5G Core »

Le réseau central 5G (5GC) adopte une architecture de contrôle et de transfert distincte, qui permet une gestion indépendante de la mobilité et de la gestion de session.

La figure ci-dessous montre L'architecture de réseau cœur 5G.

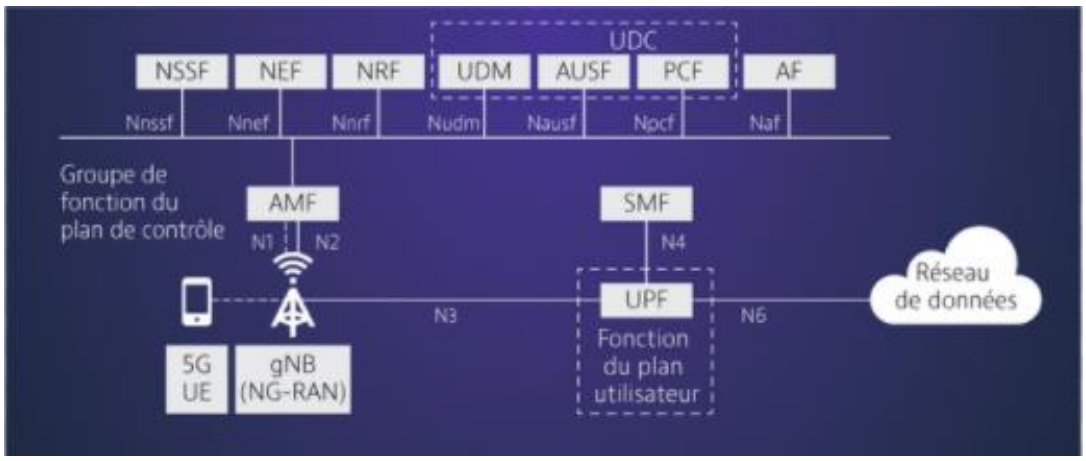

Figure 4.I : L'architecture de réseau 5G

- **AMF (Access and Mobility Function)**

L'AMF exécute la plupart des fonctions que le MME exécute dans un réseau 4G.

- L'authentification de l'UE.
- Gestion de l'accessibilité.
- Gestion de mobilité et de connexion.

❖ **SMF (Session Management Function)**

SMF exécute les fonctions de gestion de session suivantes :

- Allocation des adresses IP aux UE.
- Signalisation pour la gestion de session.
- Création de la mise à jour et de la suppression des sessions PDU (Protocol Data Unit)

❖ **UPF (User Plan Function)**

L'UPF exécute les fonctions suivantes :

- Intégrer les fonctions de pare-feu et de traduction d'adresses réseau (NAT).
- Routage et transmission de paquets.

❖ **PCF (Policy Control Function)**

PCF 5G exécute les mêmes fonctions que le PCRF dans un réseau LTE

- Prend en charge la nouvelle politique de qualité de service 5G.
- Prend en charge les fonctions de contrôle de facturation.

❖ **UDR (Unified Data Repository)**

C'est une base de donnée 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 et autorisation d'accès.
- Gestion des abonnements.

❖ **AUSF (Fonction serveur d'authentification)**

- Implémente le serveur d'authentification.
- Stocke les clés.

❖ **AF (Application Function)**

- Contrôle des politiques.

❖ **NRF (NF RepositoryFunction)**

- 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.

❖ **NSSF (Network Slice Selection Function)**

La NSSF redirige le trafic vers une tranche de réseau.

I.4.2 L'accès radio 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 (next Generation Node 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 (Next Generation-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 et la gestion des tranches de réseau (Slices) sur l'interface radio.

I.4.3 Network Slicing

Le découpage de réseau 5G est une architecture de réseau qui permet le multiplexage de réseaux logiques virtualisés et indépendants sur la même infrastructure de réseau physique. Chaque tranche de réseau est un réseau de bout en bout isolé conçu pour répondre aux diverses exigences requises par une application particulière. Pour cette raison, cette technologie assume un rôle central pour prendre en charge les réseaux mobiles 5G.

Network Slicing, s'agit de fournir un service réseau dont les caractéristiques sont spécifiques à la nature de l'usage qui en est fait. Par exemple : un acte de télémédecine requiert une fiabilité beaucoup plus importante que la visualisation d'une chaîne Youtube. Cette adaptation se fait en utilisant les technologies SDN (Software-Defined Networking) et NFV. Ces deux approches complémentaires ont révolutionné la façon de mettre en œuvre et d'opérer les réseaux.

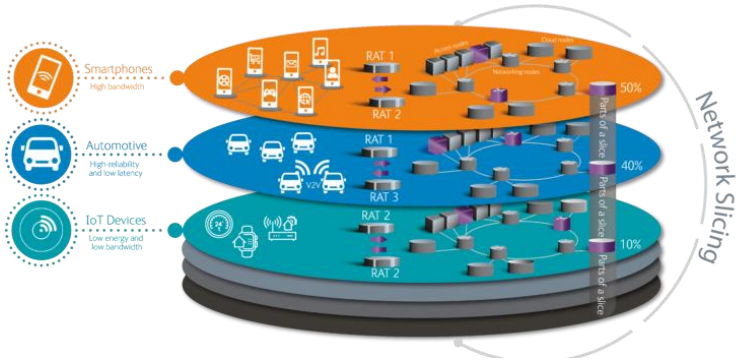

Figure 5.I : Network Slicing

I.4.4 Cloud distribué

L'infrastructure « Cloud distribué » est plus décentralisée que les architectures Cloud traditionnelle. Elle permet de répartir le trafic utilisateur physiquement à proximité du réseau d'accès, les données ne devraient pas avoir à parcourir beaucoup de trajets sur le réseau pour atteindre l'utilisateur. Cela permet de :

- Réduire le temps de latence.
- L'évolutivité de réseau.
- Répondre aux exigences des cas d'utilisation nécessitant une bande passante élevée.

Conclusion

La 5G offre plusieurs caractéristiques nouvelles dont notamment, le très haut débit garanti et la faible latence qui vont bouleverser les usages et les marchés. La 5G permet de connecter plusieurs technologies (le Cloud, l'intelligence artificielle, l'internet des objets et la réalité virtuelle) qui se nourriront les unes des autres, tout ça basé sur le concept des applications Cloud native avec NFV qui fera l'objectif du chapitre prochain.

Chapitre II

L’approche Cloud Native avec NFV

Chapitre II L'approche Cloud Native avec NFV

Introduction

Nous présentons dans ce deuxième chapitre les concepts liés à l'infonuagique (Cloud Native), à la virtualisation traditionnelle et à la conteneurisation. L'objectif est de décrire et expliquer les liens qui existent entre eux.

II.1 Cloud native

Le Cloud-native est la clé pour libérer tout le potentiel de revenus des applications 5G. Cloud-native, en substance, signifie que le logiciel a été conçu pour le déploiement dans le Cloud. Le logiciel est constitué sur des micro-services indépendants et peut s'exécuter sur une plate-forme de conteneur.

❖ **Le principe et la conception des applications Cloud native**

Le concept principal consiste à créer des logiciels sur des micro-services conçus et déployés indépendamment les uns des autres. Ces micro-services 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 peuvent être plus rapidement mises à jour, distribuées sur des territoires plus vastes, et requièrent moins d'espace de stockage en plus d'être réparties sur plusieurs serveurs.

❖ **Avantages des applications Cloud native**

- La réduction des coûts liés à l'hébergement de l'application.
- Le développement rapide des applications.
- Disponibilité de service : en cas de panne d'un équipement du DataCenter du fournisseur de Cloud, un autre équipement de ce même DataCenter peut prendre le relais automatiquement.

II.1.1 Les micro-services

II.1.1.1 Définition

Les micro-services désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments plus simples, indépendants les uns des autres. Contrairement à une approche monolithique classique, selon laquelle tous les composants forment une entité indissociable, les micro-services fonctionnent en synergie pour accomplir les mêmes tâches, tout en étant séparés. Chacun de ces composants ou processus est un micro-service.

La figure 6.II montre la différence entre l'architecture d'une approche monolithique vers les micro-services.

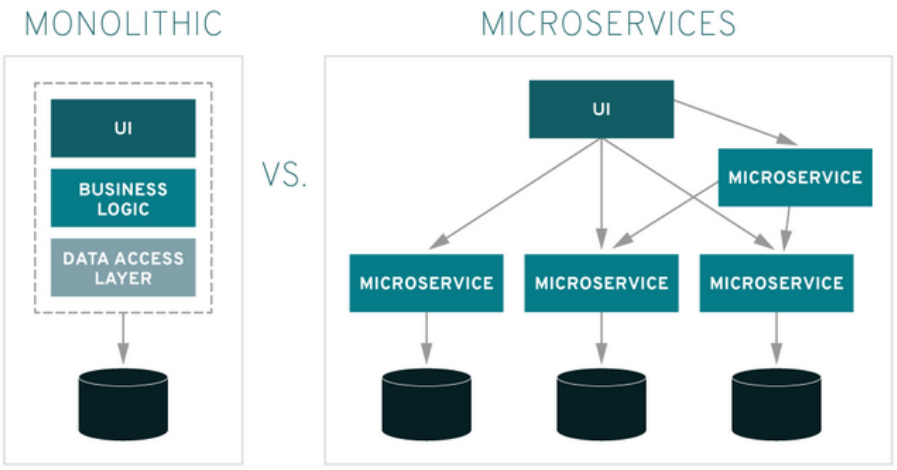

Figure 6.II : L'architecture d'une application monolithique VS les micro-services

II.1.1.2 Le concept des micro-services

Les micro-services désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments plus simples appelés services, indépendants les uns des autres. Chaque service exécute un processus unique et gère sa propre base de données, ce processus est un micro-service.

II.1.1.3 Les avantages des micro-services

- **Rapidité**

Un micro-service doit être capable de proposer de nouvelles fonctionnalités. Vous pouvez en même temps modifier un composant du système, le tester et l'utiliser dans la production. C'est une façon beaucoup plus rapide et plus agile de développer de nouvelles fonctionnalités pour un logiciel.

- **Utilisation d'outils adaptés au travail**

En raison de l'isolement et de l'indépendance des micro-services, les services individuels peuvent être polyglottes en matière de langage de programmation, ce qui offre la possibilité d'utiliser le bon outil pour le travail.

- **Dynamisme et flexibilité**

Les micro-services offrent plus de dynamisme. Ainsi, il est possible d'augmenter ou de réduire l'usage du Cloud en fonction de la charge de l'application. Grâce aux micro-services, les entreprises pourront mettre en place un système beaucoup plus flexible par rapport aux services individuels.

- **Encapsulation**

Les avantages des micro-services sont similaires aux résultats des objets et services qui communiquent via des interfaces bien définies. C'est un bon moyen de concevoir le code et la classe, mais aussi au niveau systémique, rendant les constituants du système plus simples et plus faciles à maîtriser. Ce qui les rend réutilisables.

- **Résistance**

Les micro-services, ainsi que la façon dont ils sont abordés, offrent plus de résistance, surtout quand les entreprises sollicitent plus des systèmes. Ce qui équilibre le degré de la charge.

II.1.2 La Technologie des conteneurs

Pour exécuter les applications modernes qui font appel à des micro-services, les conteneurs sont l'option préférée et Docker étant la meilleure solution de déploiement de ces micro-services dans des conteneurs.

II.1.2.1 De la virtualisation par VM à la virtualisation par conteneur

❖ **La virtualisation par VM**

La virtualisation est un mécanisme informatique qui consiste à faire fonctionner plusieurs systèmes, serveurs ou applications, sur un même serveur physique comme s'ils fonctionnaient sur des machines distinctes.

Une machine virtuelle est essentiellement une émulation d'un ordinateur réel, elle exécute des programmes dans une machine hôte à l'aide d'un «hyperviseur». Un hyperviseur, à son tour, s'exécute sur la machine hôte. Il est donc le logiciel de virtualisation sur lequel les machines virtuelles s'exécutent.

La machine virtuelle utilise les ressources informatiques de la machine physique, notamment la RAM et le Processeur. Ces ressources sont réparties entre plusieurs machines virtuelles. Et donc on peut avoir plusieurs machines virtuelles sur une seule machine physique, et chaque VM à son propre système d'exploitation.

- **Fonctionnement de la virtualisation par VM**

Il permet aux organisations de partitionner un seul ordinateur physique ou serveur en plusieurs machines virtuelles (VM)**.** Chaque machine virtuelle peut alors interagir indépendamment et exécuter différents systèmes d'exploitation ou applications tout en partageant les ressources d'un seul ordinateur.

Le logiciel hyperviseur facilite la virtualisation**.** Un hyperviseur se trouve au-dessus d'un système d'exploitation. Mais, nous pouvons également avoir des hyperviseurs qui sont installés directement sur le matériel. Les hyperviseurs prennent des ressources physiques et les répartissent afin que les environnements virtuels puissent les utiliser**.**
La figure 7.II montre l'architecture des machines virtuelles VM's

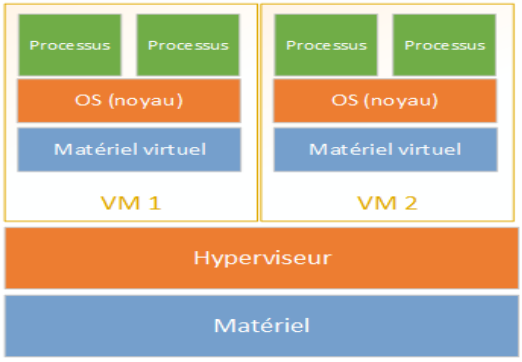

Figure 7.II : Architecture des machines virtuelles VM's

❖ **La conteneurisation**

Un conteneur est une solution logicielle permettant d'exécuter des processus dans un environnement isolé du reste système d'exploitation. Autrement dit, au sein d'un conteneur, vous verrez uniquement ce qui concerne ce conteneur et vous serez limité dans vos interactions avec le reste du système d'exploitation ou avec les autres conteneurs.

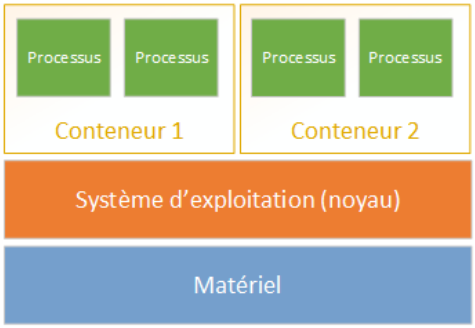

Figure 8.II : Architecture d'un conteneur

➢ **Le principe de la virtualisation par 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 fichiers…) 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).

II.1.2.2 Docker

Avec la technologie Docker, on peut traiter les conteneurs comme des machines virtuelles très légères et modulaires. En outre, ces conteneurs nous offrent une grande flexibilité : créer, déployer, copier et déplacer d'un environnement à un autre, ce qui vous permet d'optimiser vos applications pour le Cloud.

- **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, où 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.

La figure 9.II montre l'architecture de Docker

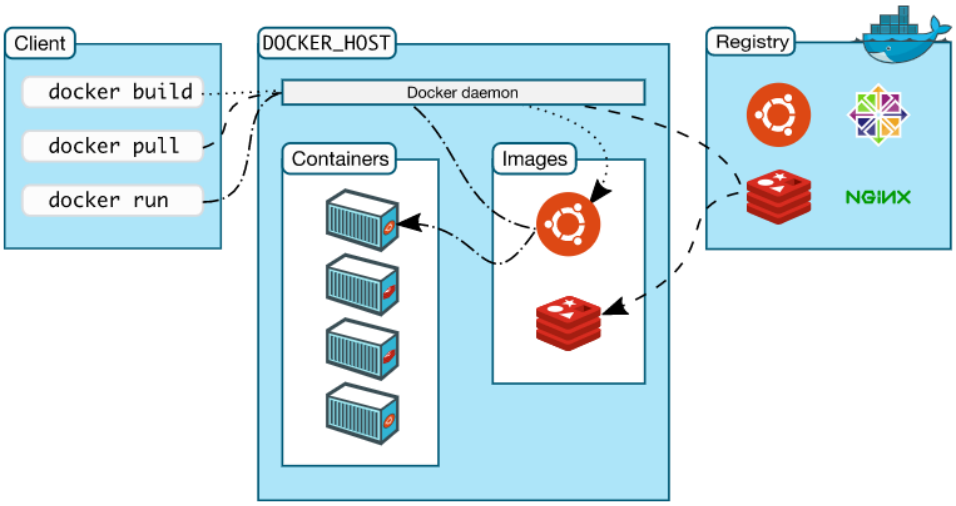

Figure 9.II : Architecture de Docker

1. **Le démon Docker**

Exécute réellement les commandes envoyées par le client Docker telles que la construction, l'exécution et la distribution des conteneurs.

2. **Le client Docker**

Le client Docker est l'interface utilisateur principal de docker, il communique avec le Docker daemon.

3. **Registre Docker**

Un registre Docker stocke les images Docker. Docker Hub est un registre public que tout le monde peut utiliser, et Docker est configuré pour rechercher des images sur Docker Hub par défaut.

4. **Image Docker**

Une image est un modèle en lecture seule avec des instructions pour créer un conteneur Docker.

5. **Objets Docker**

Lorsqu'on utilise Docker, on peut créer et utiliser des images, des conteneurs, des réseaux, des volumes, etc. Cette section est un bref aperçu de certains de ces objets.

II.1.3 Kubernetes

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 charges utilisables avec les conteneurs Docker.

II.1.3.1 L'architecture de Kubernetes

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.
- **Worker Node :** correspondant à des hôtes physiques ou virtuels qui exécutent une application Docker. Ils sont contrôlés par le Master node.
- **ETCD:** Elle joue le rôle d'une base de données du cluster.

La figure ci-dessous montre l'architecture de Kubernetes :

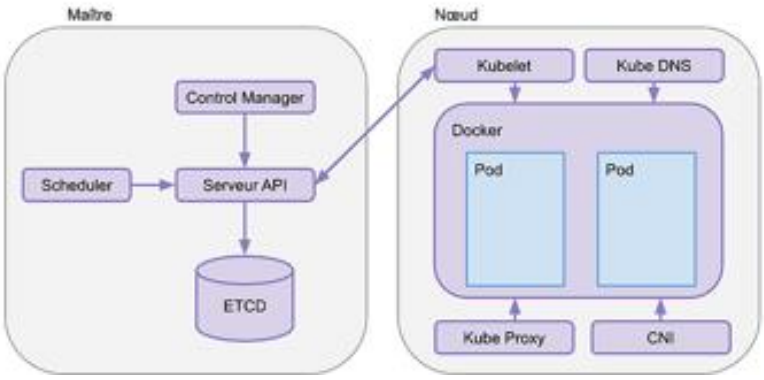

Figure 10.II : Architecture de Kubernetes

Les composantes de « master node » :

- **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.

- **API serveur :** il permet d'exécuter toutes les tâches administratives et fournit toutes les opérations sur le cluster à l'aide d'une interface (API, CLI).
- **Scheduler :** il est responsable de la planification, la répartition, et le suivi de la charge de travail entre les nœuds esclaves.
- **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 »

- **Runtime :** Aide à exécuter les conteneurs d'applications encapsulés dans un environnement d'exploitation relativement isolé mais léger.
- **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.
- **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.1.3.2 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.

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.

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 keys/values 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.

Conclusion

Alors que les opérateurs de téléphonie mobile continuent de vanter les attentes théoriques de nouveaux services et sources de revenus que la 5G est censée annoncer, cette nouvelle technologie doit reposer sur la nouvelle approche « Cloud native ».

Dans ce chapitre nous avons présenté l'approche « Cloud native », nous avons également montré comment elle favorise le déploiement des applications de télécommunication dans une infrastructure Cloud, permettant au réseau cœur 5G de répondre à ses promesses.

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

Chapitre III

Implémentation et Réalisation de l'Application

Chapitre III Implémentation et Réalisation de l'application

Introduction

Après avoir accompli toute une étude théorique sur la nouvelle approche des applications «Cloud native» considérée comme l'élément-clé de la conception des applications télécom des réseaux cœur 5G. Nous allons mettre ces notions théoriques en pratique.

Ce chapitre sera dédié à la description de la mise en œuvre de l'application «nginx» dans un cluster Kubernetes où elle a le même principe de fonctionnement d'une application 5G et voir comment elle bénéficie de l'approche « Cloud native » pour l'implémentation de ses fonctions réseau, avec un ensemble de tests de fonctionnement.

III.1 La mise en place de l'application « nginx »

Pour mettre en place l'application « nginx » dans un environnement Cloud native nous sommes passées par les étapes suivantes :

III.1.1 Installation de Kubernetes

Pour créer un cluster kubernetes nous avons utilisé l'outil « Minikube » qui est un moyen facilitant 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

```
PS C:\Users\B I P> minikube start
Enable colors error: Paramètre incorrect.
* minikube v1.20.0 sur Microsoft Windows 8.1 Pro 6.3.9600 Build 9600
* Choix automatique du pilote virtualbox
* Démarrage du noeud de plan de contrôle minikube dans le cluster minikube
* Création de VM virtualbox (CPUs=2, Mémoire=2200MB, Disque=20000MB)...
* Préparation de Kubernetes v1.20.2 sur Docker 20.10.6...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Figure 11.III : 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 :

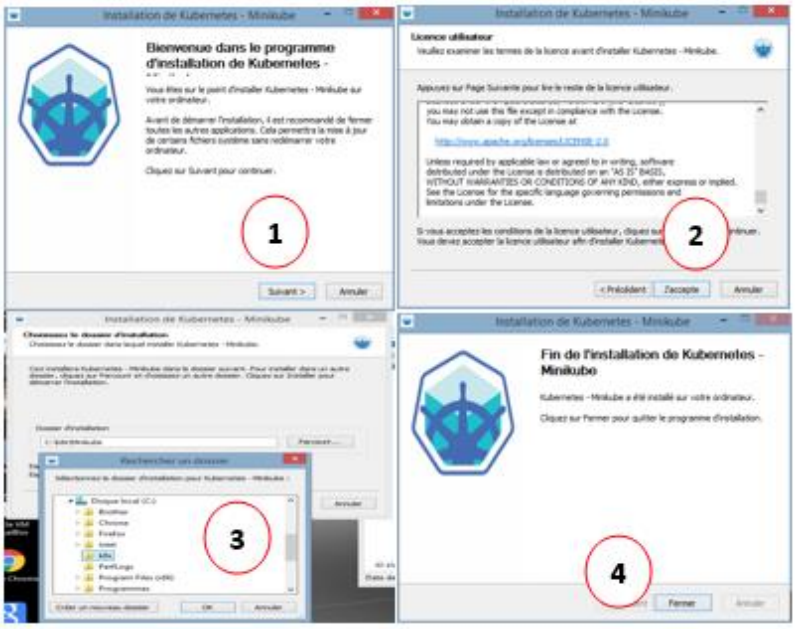

Figure 12.III : l'installation de l'outil Kubectl

❖ Il faut ajouter le binaire à notre Path, en suivant les étapes suivantes, étalées sur les figures 13.III à 15.III.

- Etape 1 : Aller au système, en choisissant « Paramètre système avancés ».
- Etape 2 : Cliquer sur Variables d'environnement.
- Etape 3 : Ajouter le Path.

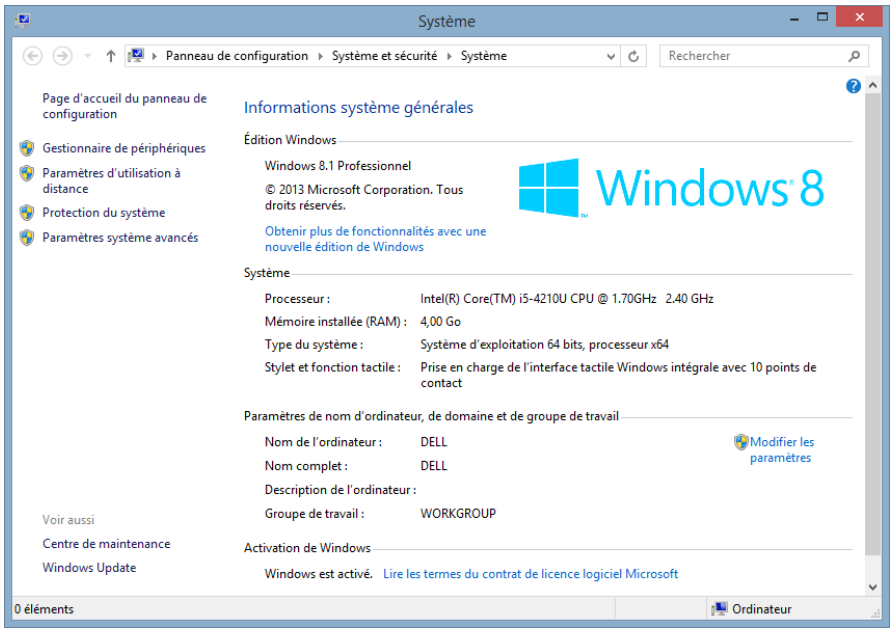

Figure 13.III : Ajouter le binaire au Path (Etape 1)

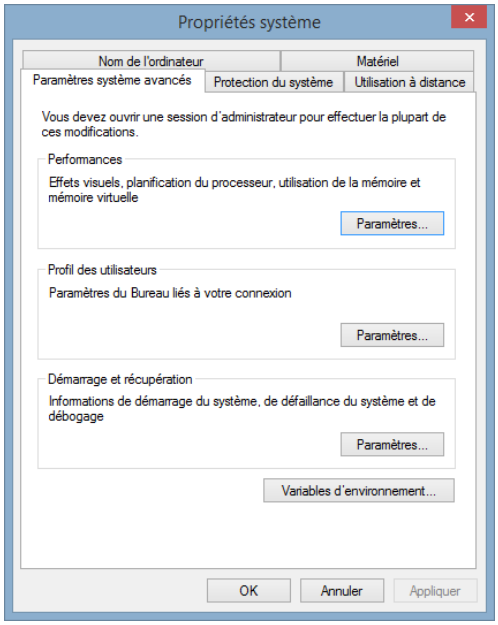

Figure 14.III : Ajouter le binaire au Path (Etape 2)

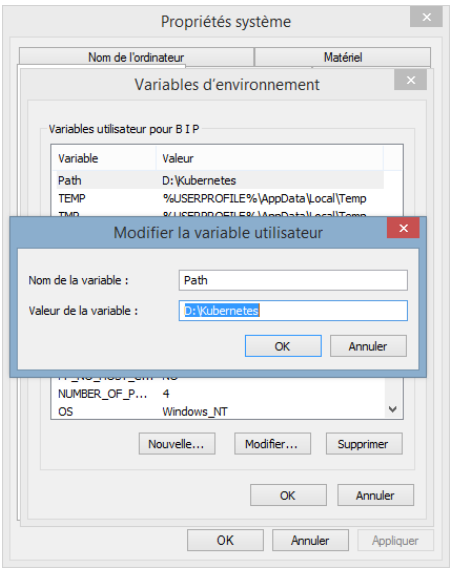

Figure 15.III : Ajouter le binaire au Path (Etape 3)

❖ 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 16.III : Vérification de l'état de cluster

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

III.1.2 Déploiement de l'application

Pour le 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 est le suivant :

```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 17.III : Fichier YAML de déploiement

Le fichier YAML de **la figure 17.III** contient les principales 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 permettant d'identifier de manière unique ledéploiement il s'agit essentiellement du nom de déploiement.

spec : Sous spec, nous avons déclaré l'état et les caractéristiques de déploiement souhaitées, 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 :

Figure 18.III : La création du déploiement

Comme la figure ci-dessus montre, notre déploiement « nginx-deployment » a été bien crée. Pour voir les détails de ce déploiement, nous avons utilisé la commande suivante :

Figure 19.III : Détails de déploiement

La figure ci-dessus montre que ce déploiement contient 2 pods qui sont en cours d'exécution, chaque pod possède 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.1.3 Tests Fonctionnels

Une fois notre application est bien déployée « prête », nous avons effectué un ensemble de tests pour s'assurer de son bon fonctionnement.

III.1.3.1 Self Healing

Kubernetes vérifie constamment l'état du cluster pour s'assurer qu'il correspond à l'état souhaité par l'utilisateur à travers des boucles de contrôle.

Si un objet tombe en panne ou n'est plus fonctionnel, Kubernetes réagit immédiatement pour le corriger afin de garder cet état souhaité sans intervention de l'utilisateur.

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

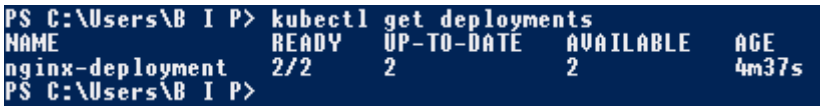

Figure 20.III : Voir le déploiement

Cette figure montre notre déploiement nginx-deployment, qu'il est en cours d'exécution. Et pour voir les pods de ce déploiement nous avons utilisé la commande suivante :

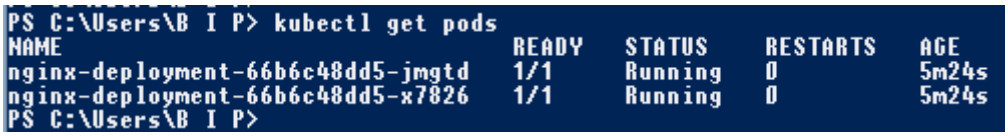

Figure 21.III : Voir les Pods de déploiement

La **figure 21.III** montre que notre application nginx est déployée sur les 2 pods que nous avons spécifiés dans le fichier YAML.

Pour tester le comportement de Kubernetes, nous avons supprimé un pod parmi ces 2 pods en utilisant la commande présentée dans la **figure 22.III**

Figure 22.III : Suppression d'un Pod

Le pod ciblé a été bien supprimé.

Pour voir comment le Kubernetes va réagir face à ce problème on a utilisé 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 23.III : 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.1.3.2 ReplicaSet

Un ReplicaSet garantit qu'un nombre spécifié de réplicas de Pod soit exécuté à un moment donné. Et on peut changer le nombre de replicas dans notre fichier YAML, ce qui implique un changement de nombre de pods dans notre déploiement, comme c'est montré 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 24.III : Fichier YAML avec 4 Replicas

Pour exécuter ce déploiement dans le cluster, on a 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 25.III : Exécution de déploiement

La figure 25.III confirme que le nouveau déploiement a été bien configuré dans le cluster. Pour voir ce déploiement (Changement de nombre de replicas) nous avons utilisé 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 26.III : Voir les Pods

On remarque bien que Kubernetes a rajouté les deux pods, et que notre application nginx est déployée sur les quatre 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 27.III : Etat de Déploiement

Comme **la figure 27.III** montre, notre cluster contient maintenant 4 pods, donc il est à jour avec notre fichier YAML.

Si l'un de ces pods tombe, Kubernetes va recréer un nouveau pod. Pour confirmer ceci, nous supprimons un pod et nous vérifions le nombre de pods restant, 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 28.III : Suppression d'un pod

Le pod a été bien supprimé.

Pour vérifier les pods restant dans notre cluster, on a utilisé 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 29.III : Vérification des pods

Le cluster contient toujours les 4 pods. Cette figure montre qu'un nouveau pod vient juste d'être créé par kubernetes et que notre déploiement contient 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.1.3.3 Rolling Update

A partir du fichier YAML, nous avons modifié la version de l'image nginx de (nginx-1.14.2) vers la version la plus récente (nginx-1.16.1) comme suit :

```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 30.III : Fichier YAML de mise à jour

Pour exécuter ce déploiement dans le cluster, nous avons utilisé 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 31.III : 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 32.III**

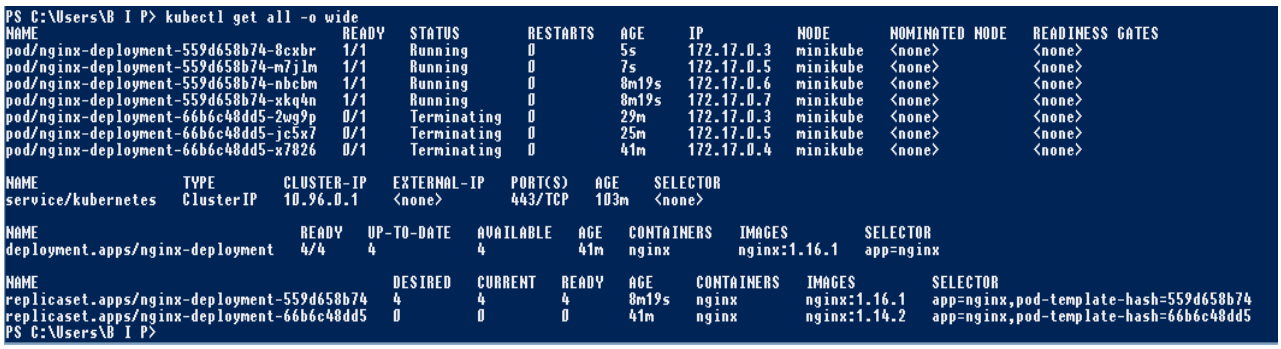

Figure 32.III : Détails de déploiement

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 recréés dans le nouveau replicaset un par un sans interruption de service (les pods sont toujours en cours d'exécution).

Donc, notre déploiement nginx-deploy est exécuté sur ce nouveau replicaset. Nous avons remarqué également 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 :

Figure 33.III : Etat de mise à jour

Pour voir l'historique de la mise à jour de notre application, nous avons utilisé la commande montrée dans **la figure 34.III**

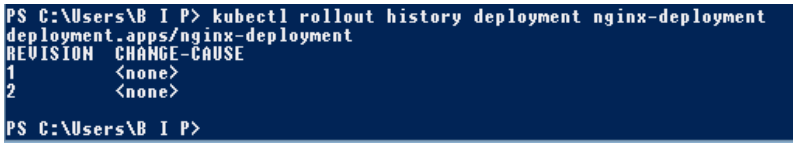

Figure 34.III : Historique de déploiement

La figure montre les deux 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

Kubernetes permet d'effectuer un Rolling-update via Kubectl. Pour le 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 35.III : 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 36.III : Etat de déploiement avec l'image 1.15

Cette figure montre que le troisième replicaset 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 37.III : 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 38.III : Historique des mises à jour

La figure 38.III 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 39.III : 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 40.III : 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 41.III : 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 42.III : Exécution de l'image nginx 0.0.0 inexistante

La figure 42.III montre que le Rolling-update n'a pas marché. Le 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.

Conclusion

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

Nous avons commencé par le déploiement de l'application nginx, qui présente ces NF sous forme de micro-services, conçus et déployés indépendamment les uns des autres. Ces micro-services sont exécutés dans des conteneurs Docker et orchestrés par Kubernetes.

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

Conclusion générale

Le développement d'applications Cloud-native est axé sur les pratiques DevOps, avec ses conteneurs, ses micro-services et son Cloud hybride. Maintenant c'est possible de changer notre façon de penser pour adopter cette approche et concevoir des services d'applications plus rapidement.

L'approche « Cloud native » permet de créer des applications sous forme de services plus petits appelés micro-services, 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 devenu un outil indispensable pour les applications Cloud Native, conçues pour être hébergées dans le Cloud, ainsi que les micro-services, il est donc la solution la plus populaire d'orchestration de ces applications conteneurisées sur un cluster de machine, en utilisant des méthodes de haute disponibilité.

La 5G représente l'avenir. La 5G requiert une évolution vers une infrastructure virtualisée, automatisée, software-defined, basée sur la plate-forme NFV. Elle nécessite la modernisation du réseau en un Cloud. La plate-forme de services NFV éprouvée en production de VMware est optimisée pour la 4G et la 5G avec une architecture unique. Elle prend en charge les réseaux actuels tout en offrant le moyen le plus rapide de passer efficacement à la 5G.

Mon projet avait pour objectif de montrer comment la conception des applications du réseau cœur 5G bénéficiera de l'approche « Cloud native ». Pour ce faire notre solution consiste à orchestrer un cluster Kubernetes contenant deux types de nœuds « master node » et « worker nodes », le premier est chargé de l'exécution du système d'orchestration Kubernetes, qui commande et contrôle toutes les autres machines du cluster. Les « worker nodes » quant à eux exécutent les applications 5G. Dans notre cas nous avons choisi l'application nginx qui est responsable 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.

Malgré les difficultés confrontées tout au long de ce projet en matière de nouveauté technologique (les réseaux de nouvelle génération 5G, les applications Cloud native, l'orchestration des applications conteneurisées avec Kubernetes), j'ai pu m'adapter à chaque situation afin d'accomplir au mieux les tâches qui m'ont été données, et ce grâce à cette expérience professionnelle au sein d'ATM Mobilis.

De plus, le stage pratique était très enrichissant autant sur l'aspect « techniques de développement des réseaux mobiles » que sur l'aspect de « création et exploitation de l'approche Cloud native ». Ce qui m'a permis d'approfondir les connaissances acquises durant notre formation à l'Université de Saad Dahleb et de mieux nous intégrer dans le milieu professionnel.

Bibliographie

[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] **Meihui Gao**, «Models and Methods for Network Function Virtualization (NFV) Architectures », Université de Lorrain - IAEM - Ecole Doctorale Informatique, Automatique, Électronique - Électrotechnique, Mathématiques, 2019-03-19.

[5] **Eventhelix**, «5G Service-Based Architecture (SBA) », 20 Octobre 2018.

[6] **Marcin Dryjanski** , « 5G Core Network Functions », 02 Mars 2018.

[7] **Bernadette Villeforceix**, Network slicing : « Une connectivité 5G innovante pour les véhicules connectés », dimanche 18 mars 2018.

[8] **Malgorzata Svensson**, **Christer Boberg**, **Benedek Kovács**, « Distributed cloud – a key enabler of automotive and industry 4.0 use cases », 20 November 2018.

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

[10] **Jonathan Morin** et **Shinie Shaw**, Network Virtualization 2nd VMware Special Edition.

[11] **Pat Gelsigner,** VMware NSX Network Virtualization Fondamentals, Gustavo A. A. Santana.

[12] **Cherifa GHERSI AMARZOURGUI**,« Tutoriel pour apprendre sous quelles conditions utiliser les microservices », 25 octobre 2017.

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

[14] **Ismail KABOUBI** (Ingénieur Smilien Système DevOps), Smile IT is Open, Livre Blanc Kubernetes, Novembre 2019