UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Comunicaciones

Implementación de OpenSlice y OSM en la Red privada 5G del iTEAM-UPV

Trabajo Fin de Máster

Máster Universitario en Tecnologías, Sistemas y Redes de Comunicaciones

AUTOR/A: Echeverría Martínez, María Isabel

Tutor/a: Gómez Barquero, David

CURSO ACADÉMICO: 2023/2024

Objetivos – Para este trabajo se plantean los siguientes objetivos:

- **Objetivo general:**

  Evaluar el rendimiento de la implementación y orquestación de un núcleo de red 5G basado en Open5GS utilizando Open Source MANO (OSM) y OpenSlice en un entorno de Kubernetes, analizando las ventajas de la virtualización de funciones de red en términos de automatización, escalabilidad y eficiencia operativa.

- **Objetivos específicos:**

  - Instalar y configurar Open Source MANO (OSM) para la orquestación y gestión del core de red 5G.
  - Instalar y probar OpenSlice como herramienta de apoyo para la gestión de servicios de red.
  - Probar la integración entre OSM y OpenSlice.
  - Evaluar las ventajas y desafíos de la utilización de OSM en el despliegue de un núcleo 5G.

Metodología – La metodología seguida en este trabajo se centra en la implementación práctica y el análisis comparativo de diferentes herramientas de código abierto para el despliegue y orquestación de un core 5G en un entorno de Kubernetes. El enfoque metodológico combina elementos experimentales con un análisis detallado de la integración de estas herramientas en un entorno virtualizado. Este es el procedimiento que se siguió para el desarrollo de este trabajo:

- Instalación y Configuración de OSM
- Instalación y Configuración de OpenSlice
- Despliegue del core 5G de Open5GS
- Análisis comparativo del despliegue de un core 5G

Resumen – Este trabajo presenta la implementación de Open Source MANO (OSM) y OpenSlice para el despliegue de un core de red 5G, utilizando Open5GS como base. Se destaca cómo la virtualización de funciones de red (NFV) es un componente esencial para la implementación de redes 5G, al permitir una flexibilidad y eficiencia superiores en la gestión y operación de los servicios de red. La virtualización desacopla las funciones de red del hardware específico, lo que facilita la creación de redes más dinámicas y escalables, capaces de adaptarse rápidamente a las demandas cambiantes del entorno digital. En este proyecto, se utiliza OSM para orquestar y automatizar el despliegue de un núcleo 5G, optimizando y simplificando significativamente la gestión de la red. Además, se integra OpenSlice a OSM como una herramienta complementaria que optimiza el flujo de trabajo y la gestión de los catálogos de servicios de red, facilitando la definición, despliegue y orquestación coherente de servicios desde su concepción inicial hasta su operación continua.

A través de esta implementación se demuestra cómo las tecnologías de código abierto pueden ser una solución efectiva para enfrentar los retos actuales y futuros en el despliegue de redes 5G.

Abstract – This work presents the implementation of Open Source MANO (OSM) and OpenSlice for the deployment of a 5G core network, using Open5GS core. It highlights how Network Function Virtualization (NFV) is an essential component for the implementation of 5G networks, enabling superior flexibility and efficiency in the management and operation of network services. Virtualization decouples network functions from specific hardware, facilitating the creation of more dynamic and scalable networks that can quickly adapt to the changing demands of the digital environment. In this project, OSM is used to orchestrate and automate the deployment of a 5G core, significantly optimizing and simplifying network management. Additionally, OpenSlice is integrated with OSM as a complementary tool that optimizes workflow and service catalog management, streamlining the definition, deployment, and coherent orchestration of services from initial conception to continuous operation. This implementation demonstrates how open-source technologies can be an effective solution to address current and future challenges in 5G network deployment.

Autor: María Isabel Echeverría, email: mechmar@teleco.upv.es
Director 1: David Gómez Barquero, email: dagobar@iteam.upv.es
Director Experimental: Borja Iñesta Hernández, email:borieher@iteam.upv.es
Fecha de entrega: 05-09-2024

Lista de figuras

Lista de tablas

Índice

1. Introducción

La virtualización y los contenedores han sido tecnologías fundamentales en el avance hacia la quinta generación de redes móviles (5G). La virtualización desvincula las funciones de red del hardware dedicado, lo que permite la creación de redes más flexibles y escalables, capaces de adaptarse rápidamente a las cambiantes demandas del entorno digital. Este enfoque no solo optimiza el uso de los recursos mediante el despliegue dinámico de funciones de red, sino que también aumenta la flexibilidad en la gestión de la infraestructura. Sin embargo, la virtualización también introduce nuevos desafíos, ya que es necesario gestionar y orquestar eficazmente estas infraestructuras virtualizadas para garantizar un rendimiento óptimo.

Reconociendo la importancia de la virtualización, el Instituto Europeo de Normas de Telecomunicaciones (ETSI) ha definido una arquitectura para la Gestión y Orquestación de las Funciones de Red Virtualizadas (MANO) [1]. MANO, del inglés *Management and Orchestration*, es un componente clave en la virtualización de funciones de red, ya que gestiona los recursos de red virtualizados y optimiza el proceso de despliegue. Su papel es esencial para garantizar una gestión eficiente y efectiva de las infraestructuras de red virtualizadas.

En este trabajo, se utilizó Open Source MANO (OSM) para la orquestación y automatización del despliegue de un core 5G en un clúster de Kubernetes. También se instaló y configuró OpenSlice, una plataforma de soporte de operaciones (OSS, *Operations Support Systems*) de código abierto. OpenSlice, orientado principalmente a la gestión de catálogos de servicios y la automatización de flujos de trabajo, se presenta como una herramienta complementaria.

Además, se presenta un análisis comparativo de las ventajas y desafíos que surgen al realizar el despliegue del core 5G directamente en Kubernetes en contraste con su despliegue a través de OSM. Se evalua el impacto de la orquestación en términos de automatización, escalabilidad y eficiencia operativa, así como la facilidad de integración y gestión del entorno.

La estructura de este trabajo se divide en cinco partes. Comienza con la presentación de los fundamentos teóricos sobre la virtualización de funciones de red y la arquitectura del core 5G. Seguido, se ofrece una descripción detallada de OSM como plataforma, junto con su proceso de instalación y configuración. Posteriormente, se introduce OpenSlice y se detalla su integración con OSM. El capítulo siguiente describe el despliegue del core 5G de Open5GS utilizando OSM. Finalmente, se presenta un análisis comparativo entre un despliegue directo en Kubernetes y el despliegue orquestado mediante OSM.

2. Fundamentos Teóricos

2.1. Virtualización y contenedores

La virtualización como su nombre lo indica, consiste en la representación virtual de máquinas físicas, es decir que en un mismo sistema físico se pueden tener múltiples máquinas virtuales. Su principal objetivo es desvincular las aplicaciones del hardware en que se ejecutan.
Hay 2 tipos de tecnología de virtualización, virtualización a nivel de hardware y virtualización a nivel de sistema operativo. En la virtualización a nivel de hardware se virtualiza el hardware de un servidor y se crean máquinas virtuales que simulan una máquina física. Para este tipo de virtualización se utiliza un hipervisor que virtualiza los recursos del servidor, como CPU y memoria, y se los asigna a cada máquina virtual. El hipervisor también es responsable de aislar una máquina virtual de las otras. En este tipo de virtualización cada máquina ejecuta su propio sistema operativo [2].
En la virtualización a nivel de sistema operativo, los contenedores comparten el kernel del *host*. A esta forma de virtualización se le conoce como contenedorización. El kernel se encarga de asignar los recursos de memoria y CPU a cada contenedor, así como de aislar los contenedores del *host* [2]. En la Figura 1 se puede ver la diferencia en la estructura de una máquina virtual y un contenedor.

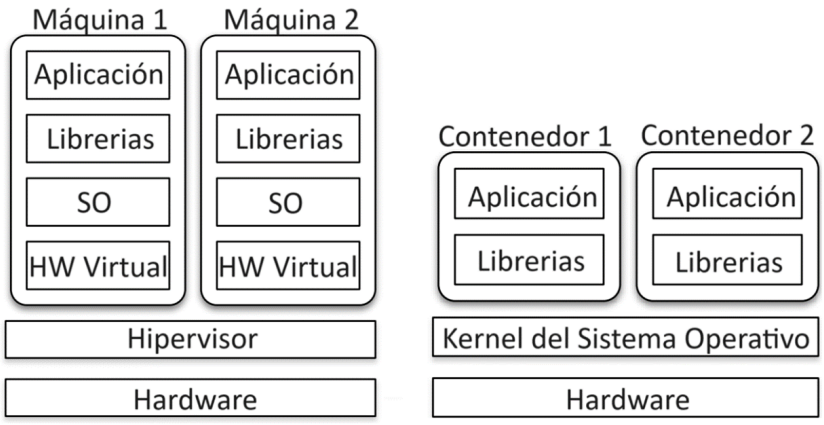

Fig. 1: Diferencia entre máquina virtual y contenedor

La virtualización optimiza el uso de un servidor porque permite dividirlo en servidores virtuales más pequeños, cada uno dedicado a una aplicación especifica [3]. Los contenedores han ganado popularidad en los últimos años debido a su menor consumo de recursos y mejor rendimiento en comparación con las máquinas virtuales. A continuación, se introducirán las tecnologías de contenedores Docker y Kubernetes ya que se utilizarán en el desarrollo de este trabajo.

2.1.1. Docker

Docker es una plataforma de contenedores que permite construir, ejecutar, desplegar, actualizar y modificar contenedores ejecutando comandos simples. Docker utiliza un archivo de *script* llamado Dockerfile que contiene el procedimiento para generar una imagen. Una imagen es la base sobre la que se crea un contenedor [4].

2.1.2. Kubernetes

Es una herramienta para la gestión y orquestación de contenedores[5]. Un clúster de Kubernetes está compuesto por nodos, que pueden ser una máquina virtual o física. Cada clúster debe tener un nodo de trabajo mínimo. En los nodos se ejecutan los pods, que son la unidad más pequeña en Kubernetes y dentro de los pods están los contenederos [6]. A continuación, se presentan algunos conceptos clave que se utilizarán en el desarrollo de este trabajo:

- **Namespace**: es un mecanismo para aislar recursos dentro de un clúster. Los nombres de los recursos deben ser únicos dentro de un namespace. En un namespace se despliegan objetos como *Pods, Services y Deployments* [7]. La finalidad de los namespace es permitir que varios proyectos puedan compartir el mimso clúster de Kubernetes.

- **Pod**: puede ser uno o más contenedores que comparten recursos de almacenamiento y red. Un Pod representa un host lógico específico para la aplicación, donde los contenedores están integrados y ejecutados en un contexto compartido [8]. Lo más común es que contenga un solo contenedor por lo que un Pod en Kubernetes es equivalente a un contenedor.

- **Service**: es el objeto que define cómo se accede a las aplicaciones dentro del clúster [9]. Por ejemplo, si hay un grupo de Pods que funcionan en conjunto, el Service se encarga de agrupar estos Pods y hacerlos accesibles. Esto permite que los Pods interactuen entre sí y que otros componentes del clúster puedan interactuar con ellos.

- **ReplicaSet**: asegura que un número constante de réplicas de Pods esté ejectutándose siempre para garantizar la disponibilidad continua de una cantidad de Pods idénticos. Se define un campo para indicar la cantidad deseada de réplicas, lo que permite gestionar el número adecuado de Pods. El ReplicaSet puede crear o eliminar Pods según sea necesarios para mantener siempre el número de réplicas definido [10].

- **Deployment**: es un recurso que facilita la actualización de Pods y ReplicaSets. Este permite especificar el estado deseado para una aplicación y asi el controlador del Deployment se encarga de ajustar el estado de los Pods y ReplicaSets para que coincida con el estado definido [11].

Docker y Kubernetes, aunque complementarios, cumplen roles diferentes en el manejo de contenedores. Docker se utiliza para crear y ejecutar contenedores, siendo su principal caso de uso el desarrollo y despliegue rápido de aplicaciones, donde se necesita un entorno consistente que funcione de la misma manera en diferentes sistemas. Kubernetes, por otro lado, es una herramienta que se encarga de gestionar y orquestar contenedores a gran escala. Su propósito es coordinar el despliegue, escalado y operación de múltiples contenedores distribuidos en un clúster de nodos. Kubernetes es esencial para gestionar aplicaciones complejas y distribuidas que requieren alta disponibilidad, despliegue automático de contenedores, escalabilidad y autorreparación. Mientras Docker facilita la creación y ejecución de contenedores individuales, Kubernetes asegura que estos contenedores funcionen de manera conjunta. En resumen, Docker ofrece una manera simple y eficiente de ejecutar y gestionar contenedores, mientras que Kubernetes proporciona características más avanzadas [12].

2.2. Virtualización de funciones de red (NFV)

Una función de red es una tarea o proceso que ejecuta un equipo de red, como un Firewall, un balanceador de carga o un sistema de protección de intrusos (IPS). La virtualización de funciones de red busca desacoplar estas funciones de un hardware especifico. Tradicionalmente, como se observa en la Figura 2, cada función se ejecuta en un hardware dedicado, lo que implica altos costos de mantenimiento y una gestión más compleja debido a la cantidad de equipos [13].
La virtualización de funciones de red busca reducir la dependencia de hardware propietario para que los servicios se ejecuten en máquinas virtuales sobre hardware de propósito general, que es más fácil de reemplazar y gestionar.

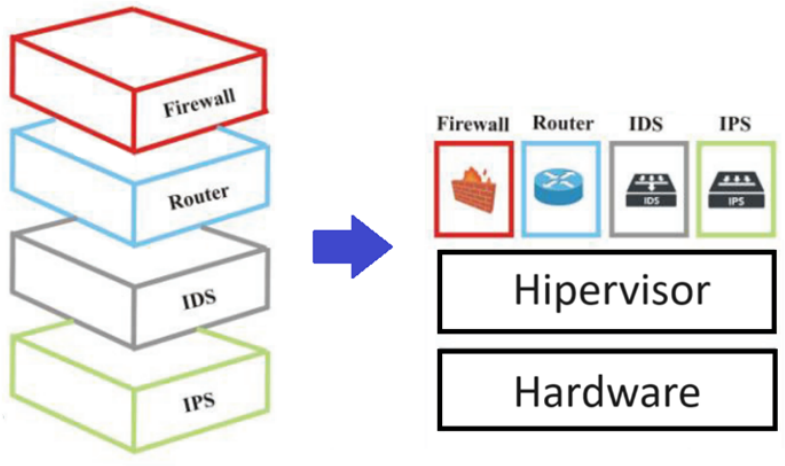

Fig. 2: Virtualización de funciones de red.

La virtualización de funciones de red se puede dividir en 3 partes fundamentales.

- **Infraestructura que contiene las máquinas virtuales y los contenedores**: Esta infraestructura es gestionada por el Gestor de Infraestructura Virtual (VIM,

por sus siglas en inglés). El VIM asigna los recursos relacionados con la computación, el almacenamiento y la red a los recursos virtuales necesarios para funciones específicas [14].

- **Funciones de Red Virtualizadas (VNFs)**: Las VNFs son composiciones interconectadas de máquinas virtuales y/o contenedores que, en conjunto, forman un servicio de red. Estas funciones acceden a sus respectivos recursos, los cuales están configurados y supervisados globalmente.
- **Orquestador**: El Orquestador se encarga de la gestión del ciclo de vida y la configuración de las funciones de red [15]. Además, es responsable de conectar las funciones de red y gestionar la orquestación de los recursos de la infraestructura de virtualización a través de VIMs.

En resumen, la infraestructura física contiene las máquinas virtuales y contenedores, los cuales, al interconectarse, forman funciones y servicios de red, y finalmente, existe un componente que gestiona y orquesta estas funciones y servicios de red.

2.3. Componentes y arquitectura del core de red 5G

La principal diferencia entre la arquitectura de generaciones anteriores y la de 5G es que, anteriormente, se utilizaban elementos de red conectados entre sí a través de interfaces dedicadas. En cambio, en el core de 5G se utilizan funciones de red que se comunican mediante interacciones basadas en servicios. Esto quiere decir que cada función de red ofrece o consume servicios mediante interfaces orientadas a servicios, eliminando la necesidad de interfaces punto a punto típicas de las arquitecturas de redes tradicionales donde se necesitaba una conexión por cada pareja de equipo que se necesitara comunicar.
En la arquitectura de 5G, las funciones de red exponen los servicios que ofrecen a través de APIs, por lo que cuando una función de red necesita un servicio utiliza un API para comunicarse con la función de red que lo proporciona. En resumen, las funciones de red del core 5G ofrecen y consumen servicios entre sí mediante interfaces basadas en servicios, que se implementan a través de APIs.
El método de comunicación que utilizan las funciones de red es HTTP REST. Este es un conjunto de reglas que define cómo las tecnologías de comunicación web acceden a servicios de aplicaciones a través de APIs [16]. HTTP REST utiliza los mismos métodos que HTTP para realizar operaciones, los más conocidos son:

- **GET**: se utiliza para obtener datos sin realizar modificaciones.
- **POST**: se utiliza para enviar datos y que se cree un nuevo recurso.
- **PUT**: se utiliza para actualizar un recurso existente.
- **DELETE**: se utiliza para eliminar un recurso.

La función de red que solicita un servicio se le conoce como *consumidora*, mientras que la función de red que lo proporciona se llama *productora*. Para que la función consumidora pueda localizar a la productora se utiliza el concepto de descubrimiento de servicio. Esto requiere que una función de red mantenga un registro de todos los servicios disponibles y de qué función de red ofrece cada uno, esto y las demás funciones de red se explican en la siguiente sección.

2.3.1. Funciones de red del core 5G

El core de la red 5G se divide en dos componentes principales: el plano de control y el plano de usuario. El plano de control gestiona los protocolos de señalización, mientras que el plano de usuario se encarga de transportar los datos de los usuarios. Además, la RAN (Red de Acceso Radio) constituye la parte central de la comunicación inalámbrica. La RAN está compuesta por la unidad de radio y la unidad de procesamiento, encargadas de la transmisión, recepción y asignación de recursos [17]. En la Figura 3 se ilustra la estructura del core 5G y las funciones de red básicas que debe incluir todo despliegue. Estas funciones se describen a continuación:

- **UPF** (*User Plane Function*): es la única función que forma parte de el plano de usuario y su principal función es reenviar y procesar los datos de los usuarios. Además, actúa como punto de conexión desde redes externas, es decir que los paquetes provenientes de Internet o redes externas son enrutados hacia el UPF asociado al dispositivo.
- **AMF** (*Access and Mobility Management Function*): La función de gestión de acceso y movilidad es responsable de interactuar con los dispositivos móviles y la RAN. Su principal tarea es permitir que los dispositivos se registren, sean autenticados y se desplacen sin interrupciones entre diferentes celdas de la red radio. Además, garantiza que el usuario sea localizable y pueda recibir notificaciones de llamadas o mensajes entrantes. También asegura que el usuario pueda iniciar comunicaciones con otros usuarios, manteniendo la conectividad incluso cuando cambia de ubicación.
- **SMF** (*Session Management Function*): como su nombre lo indica, esta función se encarga de gestionar las sesiones con los dispositivos móviles. Para establecer las sesiones con el usuario final, el SMF reenvía los mensajes a través del AMF.
- **NRF** (*Network Repository Function*): mantiene un registro de todos los servicios ofrecidos por las funciones de red. Esto implica que cada función de red debe estar configurada con la dirección IP de la NRF para registrarse y así no necesita conocer la dirección IP de todas las demás funciones de red.
- **UDR** (*Unified Data Repository*): es la base de datos que almacena información como políticas de usuario y suscripciones.
- **UDM** (*Unified Data Management Function*): Utiliza la información almacenada en el UDR para autorizar el acceso a los usuarios basado en los datos de suscripción y también genera los datos de autenticación para que los dispositivos se conecten.

- **AUSF** (*Authentication Server Function*): utiliza las credenciales de autenticación generadas por el UDM para proveer el servicio de autenticación a los dispositivos. También provee servicios criptográficos para asegurar la actualización de información de roaming y otros parámetros del dispositivo.
- **PCF** (*Policy Control Function*): gestiona las sesiones de datos de los usuarios aplicando reglas que dirigen el tráfico y garantizan el cumplimiento de las políticas de servicio. Además, controla el acceso del usuario a la red, especificando qué tecnologías de acceso radio se pueden utilizar y delimitando las áreas geográficas donde se puede establecer conectividad.

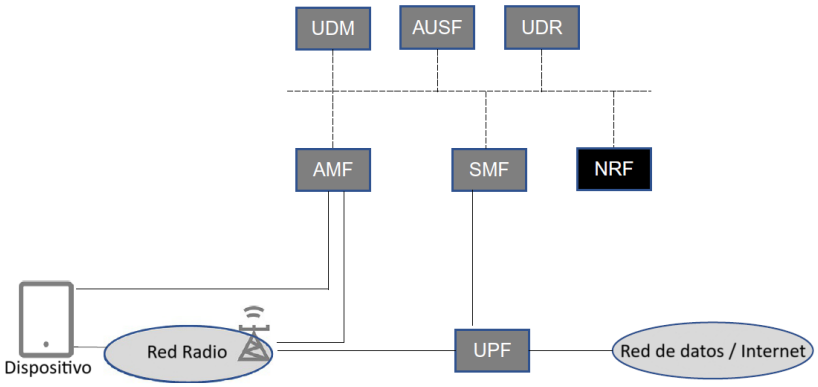

Fig. 3: Arquitectura basada en servicios del Core 5G.

Para que un dispositivo móvil se conecte a internet necesita establecer una sesión PDU (Protocol Data Units). Una sesión PDU es una conexión lógica entre un dispositivo y una red de datos, como internet. A través de esta sesión se transporta el protocolo de usuario final, pueden ser paquetes IP o tramas Ethernet. Un usuario puede requerir multiples sesiones PDU en paralelo. En la Figura 4 se observa el proceso para que un dispositivo se conecte a una red. A continuación se explica cada paso:

1. El dispositivo hace una solicitud para establecer una sesión PDU. Esta solicitud pasa a través de la RAN y se dirige a la AMF, que se encarga de gestionar la movilidad y el acceso del dispositivo. La AMF, a su vez, comunica la solicitud a la SMF, que es responsable de gestionar la sesión de datos.

2. La SMF se comunica con la UDM para obtener los datos de suscripción necesarios. Estos datos incluyen la identidad del usuario y las credenciales de seguridad. La UDM proporciona la información sobre el perfil del usuario, que es esencial para autenticar y autorizar el acceso a la red.

3. La SMF consulta a la PCF para obtener las reglas de políticas que se aplican a la sesión de datos para este usuario. Estas reglas determinan cómo debe ser gestionado el tráfico de datos del usuario, incluyendo aspectos como la calidad del servicio y las restricciones de uso.

4. Se establece la sesión entre la SMF y el UPF para el plano de usuario. La principal función de la SMF en este paso es gestionar la conexión del plano de usuario, asegurando que el tráfico de datos se maneje adecuadamente y que se establezca la sesión PDU. La SMF coordina la creación del túnel de datos entre el dispositivo y el UPF.

5. La SMF se comunica con la RAN para asignar los recursos de radio necesarios. Este paso implica coordinar con la RAN para asegurarse de que el dispositivo final tenga acceso a los recursos necesarios para la sesión de datos.

6. La RAN gestiona los recursos de radio para el dispositivo del usuario final. La RAN asigna los recursos de radio necesarios para que el dispositivo pueda establecer y mantener la conexión con la red.

7. La RAN responde la petición a la SMF. Una vez que los recursos de radio están asignados y configurados, la RAN envía una confirmación a la SMF, indicando que la configuración de los recursos está completa y que el dispositivo está listo para la comunicación de datos.

8. Se establece la comunicación entre la SMF y el UPF para configurar el túnel con la RAN. La SMF coordina con el UPF para asegurar que el túnel de datos entre el dispositivo y la RAN esté configurado correctamente, permitiendo el flujo continuo de datos.

9. Se establece el túnel desde el dispositivo del usuario final hasta el UPF. Finalmente, se crea un túnel de datos que conecta directamente el dispositivo del usuario final con el UPF, completando el establecimiento de la sesión PDU y permitiendo la transmisión de datos entre el dispositivo y la red.

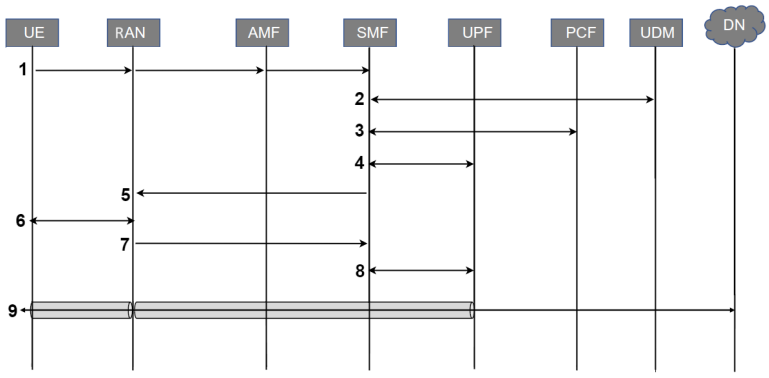

Fig. 4: Procedimiento para establecer una sesión PDU.

3. Metodología

El desarrollo de este trabajo seguirá un enfoque experimental orientado a la implementación y análisis comparativo de herramientas de código abierto para el despliegue y orquestación de un core 5G en un entorno de Kubernetes. Los pasos clave son los siguientes:

- **Instalación y Configuración de Open Source MANO (OSM)**:

  Se instalará y configurará OSM, una plataforma de código abierto alineada con los estándares de ETSI para la orquestación de funciones de red virtualizadas. OSM se utilizará como el orquestador encargado de gestionar el ciclo de vida de las funciones de red y automatizar el core 5G. La instalación se llevará a cabo en una máquina virtual con recursos específicos, detallados en la guía de instalación de OSM [18].

- **Instalación y Configuración de OpenSlice**:

  Se procederá con la instalación de OpenSlice y posteriormente su integración con OSM para facilitar la gestión de servicios de red. OpenSlice se utilizará como una plataforma de soporte de operaciones (OSS) y su instalación se realizará mediante Docker Compose. Se utilizará la interfaz norte (NBI) de OSM para la integración con OpenSlice.

- **Despliegue del Core 5G de Open5GS**:

  Se desplegará el core 5G de Open5GS en un clúster de Kubernetes. Open5GS, como solución de código abierto para implementar redes 5G, incluirá funciones de red como AMF, SMF, UPF, entre otras. Para automatizar el despliegue en Kubernetes, se utilizará un repositorio Helm y descriptores en OSM para automatizar el despliegue.

- **Análisis Comparativo del Despliegue**:

  Se realizará un análisis comparativo entre el despliegue directo en Kubernetes y el despliegue orquestado por OSM. Este análisis evaluará los beneficios de la orquestación en términos de automatización, escalabilidad y eficiencia operativa, considerando también los desafíos que pueden surgir al utilizar OSM para gestionar y desplegar el core 5G.

Este enfoque metodológico está diseñado para cumplir con el objetivo general de evaluar el rendimiento de la implementación y orquestación de el core de red 5G de Open5GS, utilizando OSM, y analizar las ventajas de la virtualización de funciones de red en términos de automatización, escalabilidad y eficiencia operativa.

4. OSM como Orquestador

4.1. Descripción y funciones de OSM

La necesidad de orquestar las funciones de red surge cuando estas se virtualizan, ya que al virtualizarlas estas se desacoplan de la infraestructura que las contiene. Este desacoplo hace que surjan nuevas necesidades de establecer una relación entre las funciones de red virtualizadas (VNFs) y la infraestructura que se utiliza para virtualizar las funciones de red [19].
MANO viene de las siglas en inglés *Management and Orchestration*, como su nombre lo indica, gestiona y orquesta los recursos de red. Open Source MANO es una solución de código abierto para gestionar el ciclo de vida, la configuración y los aspectos en tiempo real de las funciones de red. Utiliza un modelo en capas que permite crear servicios compuestos más complejos hasta construir objetos de servicio.
La idea principal de OSM es que sea orientado a modelos, de manera que estos modelos sirvan como plantillas para desplegar servicios y que tenga parámetros personalizables para cada servicio que se necesite crear.

4.2. Modelo en Capas de OSM

OSM es un software MANO alineado con ETSI NFV, por lo que utiliza su modelo en capas, que consiste en combinar varias capas de servicio para crear lo que se denomina objetos de servicio. Un objeto de servicio es un conjunto de servicios de red interconectados y un servicio de red está compuesto de una o varias funciones de red, que pueden ser un conjunto de máquinas virtuales y/o contenedores. En la Figura 5 se puede observar como se crea un objeto de servicio a partir de un componente básico, como una máquina virtual o un contenedor. La principal ventaja que aporta OSM, es que una vez se tenga el paquete del servicio de red, este funciona como modelo para crear otras instancias de red a las que se les asigna un ID único que a su vez sirve para gestionar su ciclo de vida.

Fig. 5: Modelo en capas de OSM.

4.3. Descriptores y Paquetes

OSM tiene como objetivo implementar servicios bajo demanda, y para lograrlo de manera eficiente se utilizan descriptores que funcionan como plantillas de despliegue. Existen dos tipos principales de descriptores: los descriptores de funciones de red (VNFD) y los descriptores de servicios de red (NSD). El descriptor de una función de red especifica los requisitos y configuraciones necesarios, permitiendo a OSM comprender y gestionar

diferentes tipos de funciones de red en función de los datos proporcionados en el descriptor. A continuación se describe cada tipo de función de red:

- **PNF** (*Physical Network Function*): son funciones de red que se ejecutan físicamente y se despliegan fuera del ámbito de un orquestador, pero son gestionadas como funciones de red dentro de un servicio de red. Aunque las PNFs no se despliegan directamente mediante un orquestador, siguen siendo modeladas dentro del marco de un VNFD (Virtual Network Function Descriptor).
- **VNF** (*Virtual Network Function*): es una función de red que ha sido virtualizada. El descriptor VNFD detalla la implementación de unidades de despliegue virtual (VDU) y la interconexión interna entre los diferentes componentes dentro de la VNF, asegurando su correcta operación dentro del entorno virtualizado.
- **KNF** (*Kubernetes-based Network Functions*): es una función de red nativa de la nube que se despliega en un entorno Kubernetes. El despliegue de una KNF se realiza en un clúster de Kubernetes operativo. OSM admite la implementación de KNFs utilizando tanto Helm Charts como Juju Bundles [20].

Un servicio de red (*network service*) describe la relación entre las funciones de red. El descriptor de servicios de red (NSD) incluye la información de las funciones de red que lo constituyen y los parámetros de conectividad, detallando cómo deben integrarse unas funciones con otras dentro del entorno de red. Esta información permite establecer enlaces virtuales entre múltiples funciones de red o entre los elementos necesarios para el servicio, permitiendo que las funciones de red provenientes de distintos proveedores se integren de manera transparente en un servicio de red unificado. Además, estos descriptores aseguran que los servicios de red puedan ser implementados de forma automatizada y repetible, minimizando la intervención manual y optimizando el tiempo de despliegue.
Tanto el descriptor de la funcion de red como el del servicio de red se empaquetan. El paquete de la función de red contiene el descriptor (VNFD) en un archivo yaml y otros archivos de configuración opcionales que puedan ser necesarios para el despliegue. A continuación se muestra un ejemplo de la estructura de este tipo de paquete:

```
nombre_vnfd/
    | - charms
    |
    | - checksums.txt
    |
    | - vnfd.yaml
```

El paquete del servicio de red puede tener una estructura como la que se muestra a continuación:

```
nombre_nsd/
    | - README
    |
    | - nsd.yaml
```

OSM, como plataforma, tiene la capacidad de crear un objeto compuesto complejo. Un aspecto clave de este tipo de arquitecturas es que una plataforma no está limitada a proporcionar objetos de servicio a una única plataforma 'superior'. En su lugar, puede ofrecer estos objetos de servicio a cualquier plataforma que lo solicite a través de su API, utilizando su Interfaz Norte (NBI) para gestionar la comunicación.

4.4. Instalación de OSM

Se instaló la versión 15.0.1 de OSM en una máquina virtual cuyas especificaciones se detallan en la Tabla 1. Durante el proceso de instalación, se siguió el proceso recomendado en la guía de instalación [18], que incluye la instalación de un clúster de Kubernetes independiente en la máquina virtual, sobre el cual se desplegó OSM.

Recurso	**Especificación**
CPU	4
RAM	16 GB
Disco	120 GB
Imagen base	Ubuntu 22.04

Tabla 1: Especificaciones de la máquina virtual para OSM.

4.5. Pruebas e integración a la red

Para comprobar que se instaló correctamente se accede al portal de OSM, utilizando la dirección IP de la máquina virtual en la que se instaló y se obtiene lo que se observa en la figura 6.

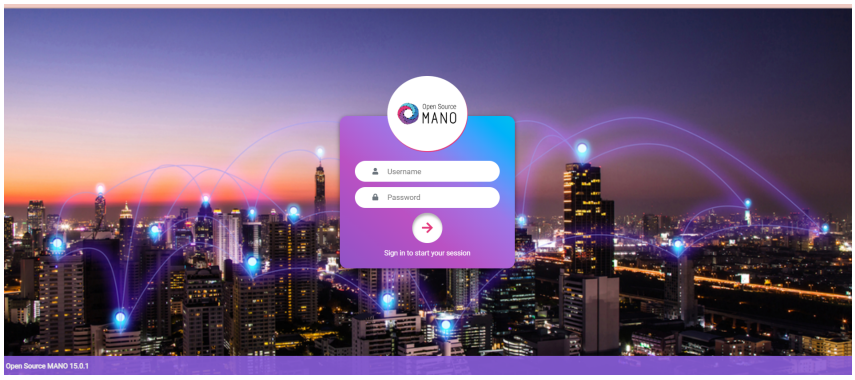

Fig. 6: Portal web de OSM.

También se realizaron validaciones para asegurar que todos los componentes se desplegaron correctamente en el clúster de Kubernetes. Se verificaron todos los objetos creados utilizando el comando:

```
kubectl get all -n osm
```

Todo el despliegue se realizó en el *namespace* denominado osm. En la Figura 7 se muestran los pods que se están ejecutando, mientras que la Figura 8 presenta los servicios correspondientes. Entre los pods más relevantes se encuentran lcm-78b9fc86bb-92cxb y nbi-5b5576dfb8-wf9mp.
El pod lcm (*Life Cycle Management*) es responsable de gestionar el ciclo de vida de las funciones de red (VNF). Además de gestionar, controla cómo se despliegan y configuran estas funciones. Por otro lado, el pod nbi-5b5576dfb8-wf9mp maneja la interfaz norte (NBI, *North Bound Interface*), que permite la interacción de OSM con otros sistemas de gestión, esta interfaz sigue el estándar ETSI SOL005. OSM utiliza su *North Bound Interface* (NBI) para servir HTTPS de manera predeterminada en el puerto 9999, lo cual se puede verificar en la Figura 8, donde se observa que el servicio nbi expone dicho puerto.

Consola

```
mechmar@osm:~$ kubectl get all -n osm
NAME                                                    READY   STATUS    RESTARTS        AGE
pod/airflow-postgresql-0                                1/1     Running   0               38d
pod/airflow-redis-0                                     1/1     Running   0               38d
pod/airflow-scheduler-79b86c5d74-xvkqg                  2/2     Running   0               38d
pod/airflow-statsd-59cf4979b6-6pvtd                     1/1     Running   0               38d
pod/airflow-triggerer-659fb58bd6-dmlx7                  2/2     Running   0               38d
pod/airflow-webserver-77db7dc89b-dfngb                  1/1     Running   0               38d
pod/airflow-worker-0                                    2/2     Running   0               38d
pod/alertmanager-0                                      1/1     Running   0               38d
pod/grafana-76bfd9c969-77khj                            2/2     Running   0               38d
pod/kafka-controller-0                                  1/1     Running   0               38d
pod/kafka-controller-1                                  1/1     Running   0               38d
pod/kafka-controller-2                                  1/1     Running   0               38d
pod/keystone-cd496f96c-wlbw8                            1/1     Running   0               38d
pod/lcm-78b9fc86bb-92cxb                                1/1     Running   0               38d
pod/mon-c6dc95cb8-vtjdl                                 1/1     Running   0               38d
pod/mongo-client                                        1/1     Running   0               21d
pod/mongodb-k8s-0                                       1/1     Running   7 (5d14h ago)   38d
pod/mongodb-k8s-1                                       1/1     Running   5 (4d18h ago)   38d
pod/mongodb-k8s-arbiter-0                               1/1     Running   0               38d
pod/mysql-0                                             1/1     Running   0               38d
pod/nbi-5b5576dfb8-wf9mp                                1/1     Running   0               38d
pod/ngui-69f78cccc8-7xrxp                               1/1     Running   0               38d
pod/prometheus-0                                        2/2     Running   0               38d
pod/pushgateway-prometheus-pushgateway-79f8f686d9-2twgk 1/1     Running   0               38d
pod/ro-889df4d56-drt9n                                  1/1     Running   0               38d
pod/webhook-translator-64ff7d7ff4-nzcwg                 1/1     Running   0               38d
pod/zookeeper-0                                         1/1     Running   0               38d
```

Fig. 7: Pods al instalar OSM sobre Kubernetes.

Consola

```
NAME                                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                       AGE
service/airflow-postgresql                  ClusterIP   10.98.55.106     <none>        5432/TCP                      129d
service/airflow-postgresql-hl               ClusterIP   None             <none>        5432/TCP                      129d
service/airflow-redis                       ClusterIP   10.99.210.117    <none>        6379/TCP                      129d
service/airflow-statsd                      ClusterIP   10.98.13.52      <none>        9125/UDP,9102/TCP             129d
service/airflow-webserver                   NodePort    10.97.114.64     <none>        8080:439/TCP                  129d
service/airflow-worker                      ClusterIP   None             <none>        8793/TCP                      129d
service/alertmanager                        NodePort    10.111.145.26    <none>        9093:9093/TCP                 129d
service/alertmanager-headless               ClusterIP   None             <none>        9093/TCP                      129d
service/grafana                             NodePort    10.109.183.36    <none>        3000:3000/TCP                 129d
service/kafka                               ClusterIP   10.110.165.138   <none>        9092/TCP                      129d
service/kafka-controller-headless           ClusterIP   None             <none>        9094/TCP,9092/TCP,9093/TCP    129d
service/keystone                            ClusterIP   None             <none>        5000/TCP                      129d
service/mongodb-k8s                         ClusterIP   None             <none>        27017/TCP                     129d
service/mongodb-k8s-arbiter-headless        ClusterIP   None             <none>        27017/TCP                     129d
service/mysql                               ClusterIP   10.101.195.179   <none>        3306/TCP                      129d
service/mysql-headless                      ClusterIP   None             <none>        3306/TCP                      129d
service/nbi                                 NodePort    10.109.198.26    <none>        9999:9999/TCP                 129d
service/ng-ui                               NodePort    10.96.152.4      <none>        80:80/TCP                     129d
service/prometheus                          NodePort    10.102.142.253   <none>        9090:9091/TCP                 129d
service/pushgateway-prometheus-pushgateway  ClusterIP   10.111.46.17     <none>        9091/TCP                      129d
service/ro                                  ClusterIP   None             <none>        9090/TCP                      129d
service/webhook-translator                  NodePort    10.97.132.20     <none>        9998:9998/TCP                 129d
service/zookeeper                           ClusterIP   10.105.141.209   <none>        2181/TCP,2888/TCP,3888/TCP    129d
service/zookeeper-headless                  ClusterIP   None             <none>        2181/TCP,2888/TCP,3888/TCP    129d
```

Fig. 8: Servicios al instalar OSM sobre Kubernetes.

5. OpenSlice como OSS

5.1. Descripción y funcionamiento de OpenSlice

OpenSlice es un sistema de soporte de operaciones (OSS, por sus siglas en inglés, *Operations Support System*) de código abierto que ofrece a los usuarios un catálogo de servicios de red. Este sistema proporciona una interfaz que permite a los usuarios solicitar, exponer y gestionar estos servicios de manera eficiente.
El proceso comienza cuando el usuario emite una orden de servicio desde el catálogo disponible. En esta orden, se especifican los detalles y requerimientos de la instancia de servicio que se desea desplegar. Una vez procesada, la orden se comunica con el orquestador. El orquestador es responsable de asignar y gestionar los recursos de red necesarios para desplegar el servicio solicitado por el usuario.
Los catálogos de servicios en OpenSlice incluyen servicios de red predefinidos basados en plantillas para funciones comunes, como firewalls, VPNs, balanceadores de carga y otros. Todos estos servicios pueden ser solicitados y gestionados a través de un portal de autoservicio, lo que facilita la interacción del usuario con el sistema.
OpenSlice está diseñado para orquestar el flujo de trabajo de extremo a extremo, asegurando una gestión completa y automatizada del ciclo de vida de los servicios. Las fases principales de este proceso son:

- **Solicitud de Servicios**: Los usuarios pueden solicitar servicios a través del portal de autoservicio. Esta solicitud incluye las especificaciones detalladas del servicio requerido.

- **Orquestación de Servicios**: El orquestador evalúa la solicitud, determina los recursos necesarios y activa los flujos de trabajo automatizados, interactuando con los componentes subyacentes para provisionar y configurar los servicios.
- **Aprovisionamiento y Configuración**: Los servicios y funciones de red (VNFs) son instanciados y configurados de acuerdo con las especificaciones de la solicitud, asegurando que los recursos estén optimizados para el rendimiento y las políticas establecidas.
- **Entrega del Servicio**: Una vez configurado, el servicio es activado y puesto a disposición del usuario, garantizando que esté operativo y accesible según lo solicitado.
- **Gestión del Ciclo de Vida**: OpenSlice maneja las actualizaciones, escalabilidad y cualquier modificación necesaria durante todo el ciclo de vida del servicio, asegurando su disponibilidad y rendimiento continuo.

5.2. Instalación de OpenSlice

Se instaló OpenSlice mediante Docker Compose siguiendo la guía [21]. Se utilizó la versión v2.28.1 de Docker Compose y se utilizó una máquina virtual con las especificaciones que se muestran en la tabla 2.

Recurso	**Especificación**
CPU	4
RAM	8 GB
Disco	60 GB
Imagen base	Ubuntu 22.04.3

Tabla 2: Especificaciones de la máquina virtual para OpenSlice.

5.3. Pruebas e integración con OSM

Tras verificar el estado de los contenedores Docker y confirmar el acceso al portal web, como se muestra en la Figura 9, OpenSlice ofrece varios portales de servicio según la función o el uso requerido. Estos son los portales disponibles:

- Desplegar servicios utilizando APIs abiertas y estandarizadas
- Gestionar recursos utilizando APIs abiertas y estandarizadas
- Desplegar VNFs y NSDs en el Orquestador asignado
- Diseñar pruebas utilizando APIs abiertas y estandarizadas
- Diseñar productos utilizando APIs abiertas y estandarizadas

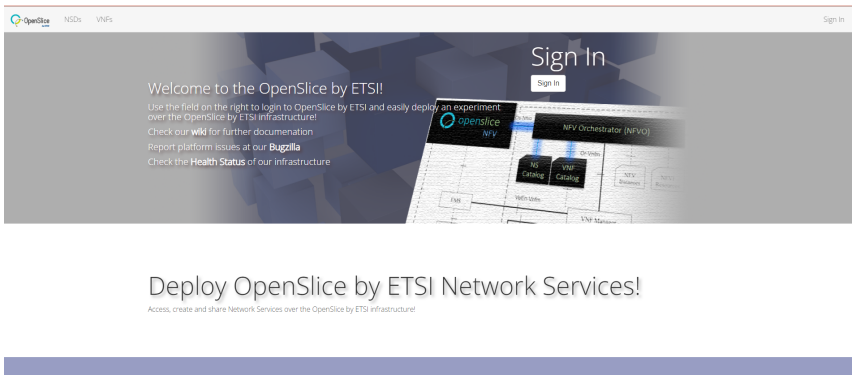

Fig. 9: Portal web de OpenSlice.

El objetivo es utilizar OSM como orquestador de servicios, integrándolo a OpenSlice. Para lograr esto, se accede al portal que permite desplegar VNFs y NSDs en el orquestador. Esto permite gestionar y compartir funciones y servicios de red entre las plataformas.
Una vez dentro del portal, en el menú desplegable de la sección 'Admin', se selecciona 'Add new MANO Platform'. Esta acción nos lleva a la interfaz mostrada en la Figura 10. En el menú desplegable de 'version', que ofrece opciones desde OSMvEIGHT hasta OSMvTHIRTEEN, se debe seleccionar la versión correspondiente de OSM. En este caso, como se está utilizando la versión 15 de OSM, se elige 'GenericSOL005'. Esta opción hace referencia al estándar ETSI SOL005, que define las características de la interfaz para la orquestación de servicios en un entorno de virtualización. OSM adopta este estándar en su interfaz North Bound (NBI).
Tras definir la plataforma de orquestación a utilizar, es necesario agregar el proveedor de orquestación. Este proceso se muestra en la Figura 11. En este paso, se debe definir el 'endpoint' que OpenSlice utilizará para comunicarse con el orquestador. Este 'endpoint' es esencial para establecer la conexión y asegurar la correcta integración entre OpenSlice y el sistema de orquestación OSM. En este caso, se utiliza como endpoint la dirección IP de la máquina virtual de OSM y el puerto 9999, que es el puerto mediante el cual se expone la API.

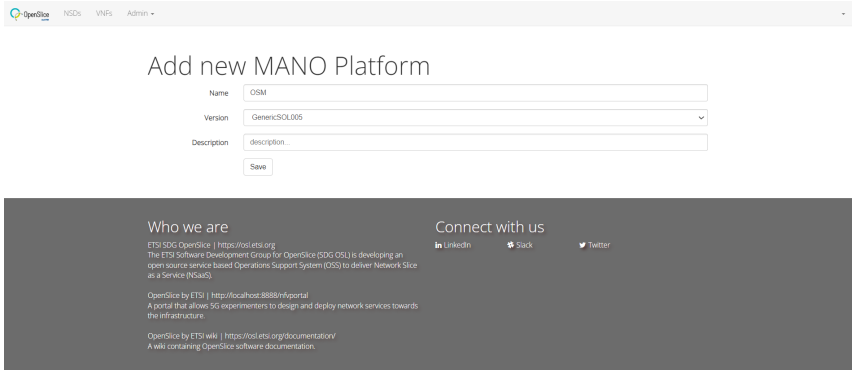

Fig. 10: Definir plataforma MANO en OpenSlice.

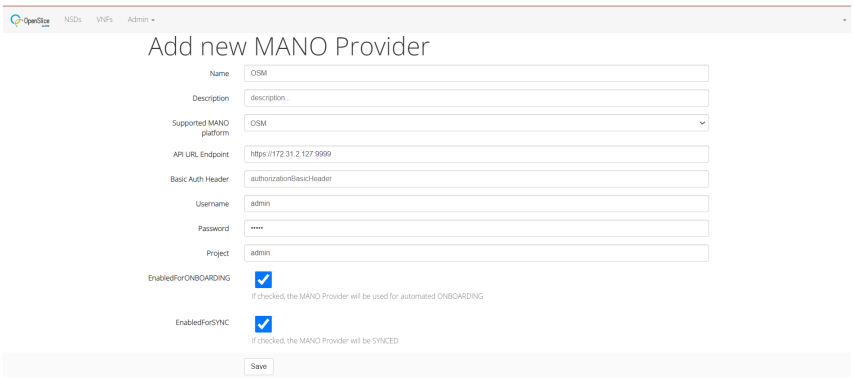

Fig. 11: Agregar el proveedor MANO en OpenSlice.

En el despliegue de OpenSlice, se incluye un contenedor denominado 'manoclient'. En los logs de este contenedor, como se muestra en la Figura 12, se puede observar el proceso de sincronización, que inicia con la detección del endpoint y continúa con la sincronización de la información. Posteriormente, en la Figura 13, se puede ver que dentro del menú desplegable 'Admin', en la opción 'Manage Infrastructures', se encuentra el VIM asociado a OSM.

Consola

```
mechmar@openslice:~$docker logs manoclient

2024-08-09 15:08:23,676 INFO  org.etsi.osl.mano.MANOClient getMANOprovidersForSync
1080:  Found EndPoint https://172.31.2.127:9999

2024-08-09 15:08:23,683 INFO  org.etsi.osl.mano.MANOClient getMANOproviderByID 1110:
The MANOprovider with name OSM has endpoint https://172.31.2.127:9999

2024-08-09 15:08:23,714 INFO  org.etsi.osl.mano.MANOController synchronizeVIMs 864:
Synchronize VIMs for MANOProvider OSM

2024-08-09 15:08:23,721 INFO  org.etsi.osl.mano.MANOClient getInfrastructures 161:
Infrastructure mylocation1 with VIM id:87243042-e1e4-44c0-b585-ed086163cf66 is loaded

2024-08-09 15:08:23,748 INFO  org.etsi.osl.mano.MANOController synchronizeVIMs 941:
synchronizeVIMs: Infrastructure 87243042-e1e4-44c0-b585-ed086163cf66 updated
Infrastructure status to OSM_PRESENT
```

Fig. 12: Logs de sincronización de OSM y OpenSlice.

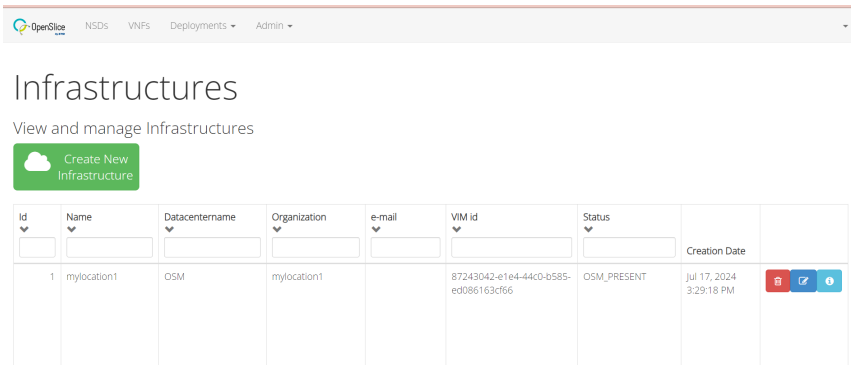

Fig. 13: Infraestructura de OSM vinculada a OpenSlice.

6. Despliegue de Open5GS con OSM

6.1. Open5GS

Open5GS es un proyecto de código abierto que permite desplegar redes 5G [22]. Este proyecto permite implementar un core 5G completo y funcional, facilitando la creación y gestión de servicios de red avanzados. Open5GS está diseñado para ser flexible y escalable, permitiendo su uso en diversas aplicaciones, desde entornos de prueba hasta redes de producción. Las funciones de red que se desplegarán incluyen:

- AMF
- AUSF
- BSF
- DB
- NRF
- NSSF
- PCF
- SMF
- UDM
- UDR
- UPF
- WEBUI

Como se detalló en la sección 2.3.1, el AMF es responsable de la gestión de conexiones y movilidad. El UDM, AUSF y UDR se ocupan de la autenticación y la administración del perfil del suscriptor. La SMF gestiona las sesiones, mientras que el PCF se encarga de la facturación y la aplicación de políticas de suscripción. Por su parte, el UPF maneja el plano de usuario [23].

6.2. Preparación del entorno

El objetivo es desplegar el core de Open5GS en un clúster de Kubernetes. Para esto, se realizaron los siguientes pasos:

6.2.1. Asociar el *cluster* de Kubernetes a OSM

Se utilizó un clúster de Kubernetes con las características que se muestran en la Tabla 3. El clúster debe estar asociado a un VIM, contar con un *LoadBalancer* y tener configurado un *storageclass* predeterminado [24].

El VIM es el componente de OSM en el que se hará el despliegue, por lo que este permite la conexión entre OSM y el clúster de Kubernetes. En general, OSM considera que el clúster está localizado en un VIM.
El *LoadBalancer* en Kubernetes, además de distribuir el tráfico entre los *pods*, también proporciona una dirección IP para exponer los KNFs a la red. En este caso, se utilizó MetalLB como *loadbalancer*, como se observa en la Figura 15.
Por último, el *storageclass* es un recurso que define la clase de almacenamiento que un *cluster* puede ofrecer. En este caso, se utilizó OpenEBS, como se observa en la Figura 14.

Consola

```
mechmar@k8s-vim:$ kubectl get storageclass
NAME                 PROV                REC     VOL                    ALLOW   AGE
openebs-device       openebs.io/local    Delete  WaitForFirstConsumer   false   123d
openebs-hostpath     openebs.io/local    Delete  WaitForFirstConsumer   false   123d
(default)
```

Fig. 14: Storageclass del clúster de Kubernetes.

Consola

```
mechmar@k8s-vim:~$ kubectl get ipaddresspools.metallb.io -n metallb-system
NAME      AUTO ASSIGN   AVOID BUGGY IPS   ADDRESSES
default   true          false             ["172.21.248.10-172.21.248.250"]
```

Fig. 15: Loadbalancer y el rango de IPs en el cluster de Kubernetes

Característica	**Descripción**
RAM	8 GiB
CPU	4
Sistema Operativo	Ubuntu 22.04.3 LTS
Versión de Kubernetes	v1.29.4

Tabla 3: Características del *cluster* de Kubernetes

Al ser un clúster de Kubernetes separado de OSM en el que se hizo el despliegue del KNF, se creó un VIM "dummy" de destino con la siguiente línea:

```
osm vim-create --name mylocation1 --user u --password p --tenant
p --account_type dummy --auth_url http://localhost/dummy
```

Con esta línea se asocia el clúster de Kubernetes al VIM:

```
osm k8scluster-add cluster --creds .kube/config --vim mylocation1
version " v1.29.4"
```

En dónde *.kube/config* contiene el archivo de configuración del clúster y *version* es la versión del clúster de Kubernetes.

6.2.2. Repositorio HELM con el despliegue de Open5GS

Un HELM chart es un conjunto de archivos que describe cómo desplegar aplicaciones y servicios en Kubernetes. Estos archivos se suelen comprimir en un solo archivo de formato tgz (Tar GZip).
Un repositorio HELM es un servidor HTTP que contiene un 'index.yaml' y uno o varios archivos tgz. El 'index.yaml' es un índice de todas los charts que contiene el repositorio. En este caso se utilizó un servidor HTTP local, en el que se colocó el archivo tgz que contiene el Chart para desplegar Open5GS y con el comando 'helm repo index' se creó el archivo 'index.yaml'.
El servidor HTTP utilizado tiene la siguiente estructura:

```
Stable/
 |
 | - index.yaml
 |
 | - open5gs-0.3.0.tgz
```

Para agregar un repositorio Helm se utiliza la instrucción:

```
helm repo add [NAME] [URL]
```

En este caso se agregó el repositorio bajo el nombre 'stable' con la siguiente instrucción:

```
helm repo add stable http://172.31.2.127:8181/stable
```

Para confirmar que se ha agregado correctamente, se puede verificar el nombre y la URL del repositorio utilizando el comando mostrado en la Figura 16.

Consola

```
mechmar@osm:~$ helm repo list
NAME                    URL
stable                  http://172.31.2.127:8181/stable
```

Fig. 16: Repositorio HELM agregado a OSM.

El archivo 'index.yaml' que se encuentra en el repositorio HELM, contiene la dirección del Chart denominado *open5gs*. En la línea 1 de la Figura 17 se muestra el nombre del Chart, y en la línea 4, la URL para acceder a él.

Código

```
1 name: open5gs
2 type: application
3 urls:
4 - http://172.31.2.127:8181/stable/open5gs-0.3.0.tgz
5 version: 0.3.0
```

Fig. 17: Contenido del archivo index.yaml.

6.2.3. Crear los descriptores: KNF y NSD

A partir de la versión 9, OSM utiliza el estándar NFV-SOL006 definido por ETSI, que establece la estructura para la redacción de descriptores.
En la Figura 18 se muestra el descriptor del KNF utilizado. Este KNF se basa en el Helm Chart descrito en la sección 6.2.2 y requiere la versión 3 de Helm para su despliegue. El KNF define tanto el identificador como el nombre. A continuación, se presentan varios conceptos clave para comprender el KNF.
El 'df', abreviatura de 'deployment flavor' en inglés, agrupa aspectos de configuración para el despliegue de una VNF. Esto permite seleccionar la configuración más adecuada en función de los requerimientos específicos.
En cuanto a los puntos de conexión (CP) y los descriptores de enlaces virtuales (VLD), hay tanto internos como externos para cada tipo. Estos se describen en la Tabla 4. Los puntos de conexión son interfaces para conectar componentes y los enlaces virtuales interconectan los puntos de conexión.

Elemento	**Definición**
VLD interno	Interconecta la unidad de despliegue virtual (VDU) con la función de red (VNF).
VLD externo	Interconecta las funciones de red (VNFs) dentro de un servicio de red (NS).
CP interno	Relaciona interfaces de VDU con VLD internos.
CP externo	Relaciona interfaces de VDU con VLD externos.

Tabla 4: Elementos de conexión.

En este despliegue se utilizó un punto de conexión (CP) externo, cuyas características se definen mediante la instrucción 'Ext-cpd', como se muestra en la línea 5 de la Figura 18.

A este punto de conexión externa se le asigna un nombre o identificador, que en este caso es 'mgmt-ext'. Además, en este punto de conexión externa se define que está relacionado con la red o conexión 'mgmtnet' del clúster de Kubernetes.
'Kdu' proviene de *Kubernetes deployment unit*, y esta sección del descriptor hace referencia al HELM Chart que contiene los archivos que se quieren desplegar. En el campo 'Helm-chart' del descriptor se debe especificar lo siguiente:

`nombre_del_repositorio/ruta_al_chart.`

Donde `nombre_del_repositorio` es el nombre con el que se agregó el servidor HTTP a HELM y `ruta_al_chart` es el nombre del chart que va a buscar en el `index.yaml`.
Se observa en la línea 14 de la Figura 18 que se coloca primero el nombre repositorio, que en este caso es 'stable' y luego el nombre del chart que es 'open5gs'.
El campo 'Mgmt-cp' en la línea 15 de la Figura 18, hace referencia al punto de conexión que se utiliza para la gestión del KNF. En este caso las operaciones de gestión del KNF se realizarán a través del punto de conexión externa definido en la línea 6.

Código

```
1  vnfd:
2    description: open5gs core descriptor
3    df:
4      - id: default-df
5    ext-cpd:
6    - id: mgmt-ext
7      k8s-cluster-net: mgmtnet
8    id: open5gs_knf
9    k8s-cluster:
10     nets:
11     - id: mgmtnet
12   kdu:
13   - name: open5gsc
14     helm-chart: stable/open5gs
15   mgmt-cp: mgmt-ext
16   product-name: open5gs_knf
17   version: 1.0
```

Fig. 18: Descriptor del KNF.

Además del descriptor del KNF, es necesario definir el descriptor del servicio de red (NSD). En la Figura 19 se muestra el NSD utilizado para este despliegue. Dentro del perfil de despliegue (df), el NSD especifica tanto el perfil del KNF como la conectividad de los enlaces virtuales. En la línea 12 de la Figura 19 se define el punto de conexión dentro de la KNF, y en la siguiente línea se asocia este punto de conexión, mgmt-ext, con el enlace virtual mgmtnet.

En la interfaz gráfica de OSM se puede observar la topología del servicio de red (NS). La Figura 20 muestra cómo la función de red utiliza el punto de conexión externo 'mgmt-ext' para interconectarse con la red de Kubernetes, a través del enlace virtual 'mgmtnet'.

Código

```
nsd:
  nsd:
    - description: open5gs descriptor
      designer: NGUI Composer
      df:
        - id: default-df
          vnf-profile:
            - id: open5gs_knf
              virtual-link-connectivity:
              - constituent-cpd-id:
                - constituent-base-element-id: open5gs_knf
                  constituent-cpd-id: mgmt-ext
                virtual-link-profile-id: mgmtnet
              vnfd-id: open5gs_knf
      id: open5gs
      name: open5gs
      version: '1.0'
      virtual-link-desc:
      - id: mgmtnet
        mgmt-network: true
      vnfd-id:
      - open5gs_knf
      mgmt-cp: mgmt-ext
  product-name: open5gs_knf
  version: 1.0
```

Fig. 19: Descriptor del servicio de red.

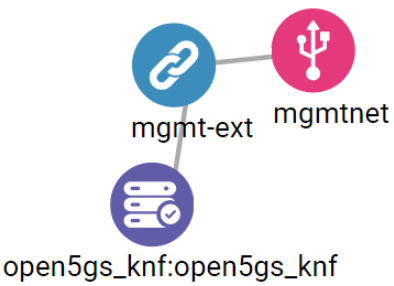

Fig. 20: Topología del servicio de red

6.3. Despliegue

Una vez que los elementos descritos en la Subsección 6.2 estén preparados, se puede proceder con el despliegue. Para instanciar un servicio de red, se accede a la interfaz mostrada en la Figura 21, donde se encuentran los paquetes de los servicios de red. Se hace clic en el ícono de instanciar, señalado en la Figura 21, y luego se ingresa el nombre, la descripción y la cuenta del VIM en la que se realizará el despliegue, como se muestra en la Figura 22. En este caso, se selecciona 'mylocation1' porque es el VIM asociado al clúster de Kubernetes.

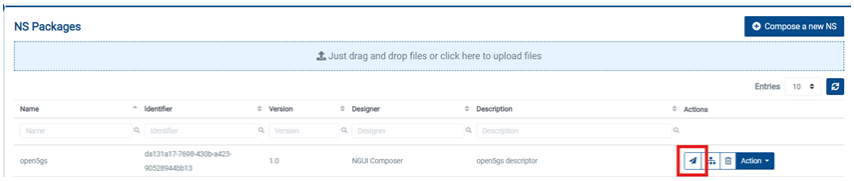

Fig. 21: Paquetes de los servicios de red en OSM.

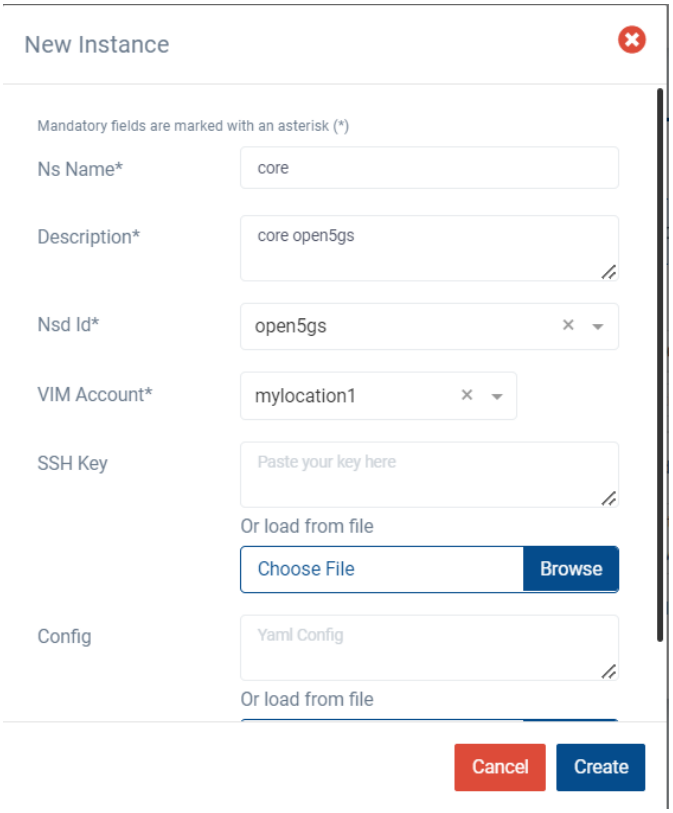

Fig. 22: Datos para instanciar el core de Open5GS.

En la Figura 22 se puede observar un campo denominado 'Config' que permite cargar un archivo en formato YAML para especificar características del despliegue.

En los logs del pod 'lcm' (*Lifecycle Management*) de OSM se observa claramente el proceso del despliegue. Se inicia con la preparación del entorno, donde se crean los directorios necesarios para almacenar la información del despliegue. Luego, se sincronizan los datos relevantes, incluyendo la obtención de descriptores de servicio y registros de funciones de red desde la base de datos de OSM. También se sincronizan los repositorios HELM en el clúster de Kubernetes y, finalmente, se configura el clúster de Kubernetes para hacer el despliegue del core.
A continuación, se analisan algunos de los puntos más relevantes de la instanciación del core. En la Figura 23 se observa el log de validación del namespace: *117a5027-09cb-4db8-8415-13a8ae435920*. Es necesario verificar en el clúster de Kubernetes la existencia de este namespace, ya que será el utilizado para el despliegue del core. Además, la figura 24 confirma que el identificador del clúster de Kubernetes coincide con el 'cluster-id' que aparece en el log.

Consola

```
2024-08-11T13:26:32 DEBUG lcm.ns k8s_helm3_conn.py:245 checking if namespace
    117a5027-09cb-4db8-8415-13a8ae435920 exists cluster_id 57f02c0e-c468-4eb2-91
    e4-af325fc3b916
```

Fig. 23: Revisión del namespace en el clúster de Kubernetes.

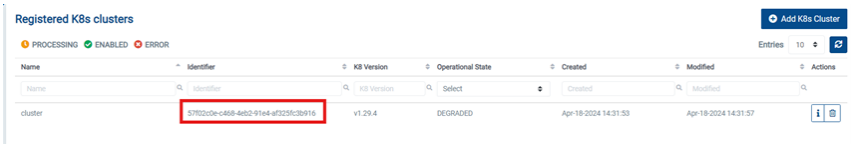

Fig. 24: ID del cluster de Kubernetes asociado al VIM.

En la Figura 25 se muestra el log de la instalación del Helm chart `stable/open5gs` en el namespace: `'117a5027-09cb-4db8-8415-13a8ae435920'` del cluster.

Consola

```
2024-08-11T13:26:33 DEBUG lcm.ns k8s_helm_base_conn.py:440 installing:
env KUBECONFIG=/app/storage/57f02c0e-c468-4eb2-91e4-af325fc3b916/.kube/config
/usr/local/bin/helm3 install stable-open5gs-0095088503 --atomic --output yaml
--timeout 1800s --namespace 117a5027-09cb-4db8-8415-13a8ae435920 stable/open5gs
```

Fig. 25: Instalación del Helm Chart en el clúster de Kubernetes.

Para confirmar que el despliegue se ha realizado correctamente, se revisa el clúster de Kubernetes y se verifica que todas las funciones de red del core estén operando sin problemas, como se muestra en la Figura 26.

Consola

```
mechmar@k8s-vim:~$ kubectl get pods -A
NAMESPACE                              NAME                                                        READY   STATUS
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-amf-deployment-7866b96959-xb2z7   1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-ausf-deployment-b4994587c-n25b5   1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-bsf-deployment-69f5bb447f-5hnwg   1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-db-deployment-696675d888-615ts    1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-nrf-deployment-f854c6f98-d4gbz    1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-nssf-deployment-74f844bdc6-bwbqq  1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-pcf-deployment-cb84b65d9-zlqbb    1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-smf-deployment-587fc7b46b-zng7c   1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-udm-deployment-57d74b44c6-tdprf   1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-udr-deployment-5677488d97-khzq4   1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-upf-deployment-f86585f65-zxprf    1/1     Running
117a5027-09cb-4db8-8415-13a8ae435920   stable-open5gs-0095088503-webui-deployment-7f49ddfdcd-j5djb  1/1     Running
```

Fig. 26: Estado del core en el cluster de Kubernetes.

7. Análisis del Despliegue

Este apartado presenta un análisis comparativo del despliegue y configuración de Open5GS en un clúster de Kubernetes de dos formas: mediante una instalación directa y utilizando Open Source MANO (OSM). Los resultados se estructuran en torno a los principales criterios de evaluación: facilidad de despliegue, recuperación y escalabilidad, destacando las diferencias y ventajas de cada enfoque.

7.1. Facilidad de Despliegue

Se evaluó la complejidad del despliegue de Open5GS mediante OSM en comparación con un despliegue directo en Kubernetes. Se observó que el uso de OSM facilitó la integración y la configuración automatizada del sistema.

- Proceso de Despliegue: El despliegue de las funciones de red del core 5G se realizó utilizando Helm charts y descriptores específicos creados para OSM. Una vez creado el paquete, se instanció el servicio y todas las funciones de red se desplegaron correctamente en el clúster.

  Para el despliegue directo en Kubernetes se utiliza el mismo repositorio HELM. Sin embargo, este proceso requiere ejecutar una serie de instrucciones bash para preparar y configurar el entorno. Por ejemplo, es necesario crear el *namespace* en el que se quiere desplegar el core, lo que requiere un conocimiento técnico más avanzado.

- Configuración Simplificada: La utilización de descriptores estandarizados en OSM simplifica significativamente el proceso de despliegue una vez que estos han sido

creados. Esta estandarización permite una mayor reutilización y flexibilidad en la configuración del entorno. Con los descriptores listos, el tiempo requerido para el despliegue se reduce considerablemente gracias a la automatización proporcionada por OSM. Además, la interfaz gráfica de OSM facilita el proceso, permitiendo que cualquier persona pueda desplegar un servicio con mayor facilidad.

7.2. Recuperación y Escalabilidad

Se evaluaron las capacidades de recuperación ante fallos y la escalabilidad del sistema en función de la demanda de servicios.

- Robustez del Sistema: Cuando se despliega un core 5G en Kubernetes, la plataforma ofrece una robustez inherente gracias a su capacidad para detectar y reemplazar automáticamente los pods que fallan. Sin embargo, la instalación mediante OSM añade una capa adicional de robustez al gestionar de manera más efectiva fallos a gran escala. En una instalación directa de Kubernetes, la recuperación de un fallo mayor podría requerir intervención manual para reiniciar o reconfigurar los servicios, lo que resulta en tiempos de respuesta más lentos y potencialmente menos eficientes. OSM, en cambio, permite una automatización y orquestación avanzada que puede minimizar la necesidad de intervención manual, mejorando la resiliencia del sistema en situaciones de fallo crítico.
- Escalabilidad: OSM facilita la gestión de despliegues 5G complejos que incluyen múltiples funciones de red interconectadas. Esta capacidad de orquestación avanzada permite que el sistema no solo escale a nivel de pod, como lo haría Kubernetes de manera predeterminada, sino que escale de manera eficiente a nivel de servicio completo. Esto es particularmente relevante en redes 5G, donde múltiples componentes y funciones deben escalar de manera coordinada para satisfacer las demandas de tráfico.

8. Conclusiones

La virtualización de funciones de red ha transformado radicalmente la forma en que se despliegan y gestionan los servicios. Al permitir un despliegue y modificación de servicios mucho más ágil, esta tecnología reduce notablemente los costos de implementación, al eliminar la dependencia de hardware dedicado. Estas ventajas no solo optimizan los gastos operativos y de capital, sino que también aceleran la innovación, permitiendo a los operadores adaptarse rápidamente a nuevas demandas del mercado.

Una de las mayores fortalezas de Open Source MANO (OSM) es su capacidad para gestionar múltiples tipos de funciones de red. OSM facilita la creación de servicios más complejos gracias a su arquitectura en capas, lo que permite una orquestación más precisa y adaptable. Una vez definido un paquete de servicio de red, este se convierte en un modelo reutilizable, simplificando la creación de nuevas instancias de red.

Este trabajo ha demostrado la eficacia de OSM para el despliegue y orquestación de un core 5G basado en Open5GS dentro de un clúster de Kubernetes. OSM ha mostrado claras ventajas en términos de automatización y gestión, permitiendo una implementación más eficiente y menos propensa a errores en comparación con un despliegue directo en Kubernetes. La capacidad de OSM para manejar descriptores y automatizar configuraciones reduce considerablemente la complejidad del proceso, lo que se traduce en una mayor eficiencia operativa y una escalabilidad mejorada del sistema.

Sin embargo, es importante señalar que el proceso no estuvo exento de desafíos. Lograr el despliegue de Open5GS en Kubernetes y la elaboración de los descriptores de red presentaron complejidades que requirieron un manejo cuidadoso y un conocimiento de las tecnologías involucradas. Además, estos desafíos destacan la necesidad de mantener la documentación actualizada y accesible, para facilitar el despliegue de servicios y una implementación efectiva de las plataformas.

Finalmente, cabe destacar que el uso de tecnologías de código abierto ha sido fundamental para el desarrollo de este trabajo. La elección de soluciones como OSM, OpenSlice y Open5GS, todas ellas basadas en código abierto, ha proporcionado flexibilidad para adaptarse a las necesidades específicas de cada proyecto. Estas herramientas no solo facilitan la personalización y la integración, sino que también fomentan la innovación continua y la colaboración dentro de la comunidad.

Referencias

[1] ETSI, "Open source mano (osm) introduction," https://www.etsi.org/technologies/open-source-mano, 2024.

[2] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th International Middleware Conference, Middleware 2016*. Association for Computing Machinery, Inc, Nov. 2016.

[3] S. D. Team, "Virtualization," https://libraetd.lib.virginia.edu/downloads/pz50gx15r?filename=Sundstrom_Bronte_STS_Research_Paper.pdf, [Online; Accessed: Aug. 10, 2024].

[4] D. D. Team, "Docker architecture," https://docs.docker.com/guides/docker-overview/, [Online; Accessed: Aug. 16, 2024].

[5] V. B. Mahajan and S. B. Mane, "Detection, analysis and countermeasures for container based misconfiguration using docker and kubernetes," in *Proceedings of International Conference on Computing, Communication, Security and Intelligent Systems, IC3SIS 2022*. Institute of Electrical and Electronics Engineers Inc., 2022.

[6] K. D. Team, "Kubernetes nodes," https://kubernetes.io/docs/concepts/architecture/nodes/, [Online; Accessed: Aug. 19, 2024].

[7] K. N. Documentation, "Namespaces," https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/, 2024, accessed: 2024-08-25.

[8] K. P. Documentation, "Pods," https://kubernetes.io/es/docs/concepts/workloads/pods/pod/, 2024, accessed: 2024-08-25.

[9] K. S. Documentation, "Service," https://kubernetes.io/es/docs/concepts/services-networking/service/, 2024, accessed: 2024-08-25.

[10] K. R. Documentation, "Replicaset," https://kubernetes.io/es/docs/concepts/workloads/controllers/replicaset/, 2024, accessed: 2024-08-25.

[11] K. D. Documentation, "Deployment," https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/, 2024, accessed: 2024-08-25.

[12] S. Banerjee, "Kubernetes vs docker: A comprehensive comparison," https://www.civo.com/blog/kubernetes-vs-docker-a-comprehensive-comparison, 2024.

[13] I. C. Committee, "2020 8th international conference on reliability, infocom technologies and optimization (trends and future directions)," in *Proceedings of the 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)*, IEEE. IEEE, 2020.

[14] A. Fornes-Leal, I. Lacalle, R. Vaño, C. E. Palau, and F. Boronat, "Evolution of mano towards the cloud-native paradigm for the edge computing," in *Proceedings of the International Conference on Edge Computing and Scalable Cloud (ICESCC)*. Springer, 2022, pp. 1–22.

[15] O. D. Team, "Osm scope, functionality, operation and integration guidelines," https://osm-download.etsi.org/ftp/Documentation/201902-osm-scope-white-paper/#!index.md, [Online; Accessed: Aug. 16, 2024].

[16] G. N. D. Team, "5g core networks."

[17] R. Reddy, M. Gundall, C. Lipps, and H. D. Schotten, "Open source 5g core network implementations: A qualitative and quantitative analysis," in *2023 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE, 2023, pp. 253–258, accessed: August 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10299755

[18] ETSI, "Open source mano (osm) - quickstart guide," 2024, Último acceso: Agosto 25, 2024. [Online]. Available: https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html#installing-osm

[19] NFV, "Gs nfv-man 001 - v1.1.1 - network functions virtualisation (nfv); management and orchestration," http://portal.etsi.org/chaircor/ETSI_support.asp, 2014, [Online].

[20] V. R. S, "Osm#10 hackfest - hd1.4 network services and nf models," in *OSM#10 Hackfest*. ETSI, 2020, overview of Network Service and Network Function models.

[21] O. Documentation, "Openslice deployment guide with docker compose," https://osl.etsi.org/documentation/latest/deploymentCompose/, 2024, accessed: 2024-08-26.

[22] R. Zhang, Y. Lin, S. Chen, and Z. Mo, "A multi-node 5g core network testbed developed from open5gs," in *2023 9th International Conference on Computer and Communications, ICCC 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 1038–1043.

[23] O. Documentation, "Introduction to open5gs," https://open5gs.org/open5gs/docs/guide/01-quickstart/, 2024.

[24] K. D. Team, "Kubernetes installation," https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html#osm-kubernetes-requirements.