Universidad de Valladolid

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Diseño de un microCPD Edge Hiperconvergente mediante herramientas OpenSource

Autor:

Álvaro Curto Merino

Tutores:

Juan Carlos Aguado Manzano

Jesús Macías Portela

Valladolid, junio de 2023

RESUMEN

La irrupción de las tecnologías 5G, el Internet de las cosas y la Inteligencia Artificial han supuesto un gran aumento en la cantidad de dispositivos conectados a la red, así como en la cantidad de datos generados. Con el objetivo de reducir la latencia en el procesamiento de datos y de liberar el tráfico hacia el centro de la red, surge como solución la computación en el borde o *edge computing*. Es por ello necesario contar con un diseño y arquitectura inicial que sirva como base a la hora de desplegar Centros de Procesamiento de Datos (CPD) *edge* para cualquier tipo de infraestructura.

El CPD estudiado en este proyecto parte de un paradigma de hiperconvergencia que permita la presencia del menor número de dispositivos posibles y la virtualización de sus recursos (computación, redes y almacenamiento). Estos recursos virtualizados pueden ser gestionados vía API del mismo modo que se haría con las nubes públicas más habituales permitiendo la creación de nubes híbridas, esto es, la combinación de recursos privados y públicos. La arquitectura *software* del proyecto se diseña pensando en el uso de herramientas *open source* para otorgar mayor libertad en el diseño y evitar *vendor locking*. Por lo tanto, en este proyecto se presenta una solución sencilla de CPD *edge* mediante herramientas de virtualización (OpenNebula) y de almacenamiento (Ceph) *open source*, implementable en un gran número de contextos, y desplegable de manera automatizada mediante un script de Ansible.

Palabras clave: Edge Computing, Hiperconvergencia, OpenNebula, Ceph

Índice

LISTADO DE ACRÓNIMOS

API: Application Programming Interface

AWS: Amazon Web Services

CI: Converged Infrastructure

CLI: Command Line Interface

CMP: Cloud Management Platform

CPD: Centro de Procesamiento de Datos

CRUSH: Controlled Replication Under Scalable Hashing

EDC: Edge Data Center

FC: Fiber Channel

Fio: Flexible I/O

GUI: Graphical User Interface

HA: High Availability

HCI: Hyper Converged Infrastructure

HCL: Hashicorp Configuration Language

IaaS: Infrastructure as a Service

IaC: Infrastructure as Code

IoT: Internet of Things

iSCSI: Internet Small Computer System Interface

KVM: Kernel-based Virtual Machine

LACP: Link Aggregation Control Protocol

LAN: Local Area Network

LUN: Logical Unit Number

LVM: Local Volume Manager

MDS: MetaData Server

MGR: Manager

MON: Monitor

NAS: Network Attached Storage

NFS: Network File System

NIC: Network Interface Card

NTP: Network Time Protocol

OCA: OpenNebula Cloud

OSD: Object Storage Devices

PG: Placement Group

QEMU: Quick emulator

RADOS: Reliable Autonomic Distributed Object Store

RBD: Rados Block Device

RPC: Remote Procedure Call

SAN: Storage Area Network

SDDC: Software Defined Data Center

SDS: Software Defined Storage

TSDB: Time Series Data Base

UI: User Interface

VIP: Virtual IP

VM: Virtual Machine

VNet: Virtual Network

YAML: Yet Another Markup Language

1. INTRODUCCIÓN

1.1. Motivación

El *edge computing* o computación en el borde hace referencia a la ejecución de procesos o gestión de datos en una ubicación física lo más cercana al usuario final. Su objetivo es acelerar los procesos y el acceso a la información para ofrecer servicios en tiempo real, confiables y eficientes. En un entorno híbrido clásico, las empresas podían ejecutar procesos y acceder a datos tanto desde sus servidores locales (*on-premise*) como desde la nube (*public cloud*). Con el *edge computing* se expande esa posibilidad a muchas más ubicaciones, gracias a Centros de Procesamientos de Datos (CPDs) repartidos geográficamente. El *edge* no busca reemplazar a la computación centralizada tradicional, sino que la complementa solucionando de paso sus principales limitaciones como son la latencia o el ancho de banda compartido.

Son cada vez más los casos de uso en los que se hace evidente la necesidad de este nuevo tipo de CPD. Sin ir más lejos, la actual implementación de redes móviles 5G habilita el acceso masivo al intercambio de datos a una gran variedad de dispositivos, generando la necesidad de procesar grandes volúmenes de datos de manera acorde a los requerimientos de la nueva generación de servicios. La computación en el borde es el complemento perfecto al despliegue de las nuevas redes de comunicaciones, al posibilitar la ejecución de procesos y la gestión de datos en nodos intermedios, sin necesidad de enviar constantemente todos los datos al *core* de la red [1].

Según estimaciones de la UE hechas públicas por Roberto Sánchez [2], anterior secretario de Estado de Telecomunicaciones, Europa contempla la construcción de 10.000 nodos *edge* para 2030, por lo que España, según su peso dentro de la UE, se enfrenta a la construcción de 1.000 en nueve años. En el último barómetro publicado por la empresa Barbara IoT con información de más de 100 empresas industriales españolas [3], el 40% duplicaron sus inversiones en *edge computing* en 2022 respecto a 2021 (principalmente eléctricas y Telcos) llegando a una media de 400.000€ por empresa. Globalmente se estima que las inversiones en *edge computing* aumenten un 13.1% entre 2022 y 2023 hasta los 208 mil millones de dólares [4], y que el mercado de los *Edge Data Centers* (EDCs), valorado en 9.000 millones de dólares en 2022, crezca por encima del 15% entre 2023 y 2032 [5]. Entre 2020 y 2026 se estima que la inversión en miniCPDs en concreto aumente en un 26%, pasando de los 4.000 millones de dólares en 2019 a superar los 15.000 millones [6], y ya en 2022 estos constituyen el 26.7% de la cuota de mercado de todos los *EDCs* [5].

No obstante, y pese a los datos expuestos, la mayoría de las empresas no cuentan con un proyecto definido ni plan sobre cómo poner en marcha su infraestructura *edge*. Este proyecto realizado con Telefónica busca satisfacer este vacío e idear un modelo base de miniCPD lo más sencillo posible, pero gestionable por APIs (Application Programming Interfaces) para la ejecución de aplicaciones de propósito general como si fuera una nube.

Para que el miniCPD *edge* sirva para cualquier tipo de aplicaciones, este debe ser flexible, modular y eficiente en *hardware* en propiedad (*on-premise*). La solución pasa por la utilización de una infraestructura Hiperconvergente (HCI), la cual consiste en poder reunir virtualizados todos los componentes del centro de datos (almacenamiento, procesamiento, red y gestión) en un número reducido de servidores. Las HCI no solo suponen un mejor aprovechamiento de los recursos, sino que han sido una tendencia en alza en el mercado de los CPDs durante los últimos años. Por cuantificar, el tamaño del mercado global de Infraestructuras Hiperconvergentes fue de 5,88 mil millones de dólares en 2020 y se prevé que pase de 6,79 mil millones en 2021 a 32,19 mil millones de dólares en 2028, a una tasa de crecimiento anual compuesta del 24,9% en el periodo 2021-2028 [7]. .

1.2. Objetivo

Este trabajo busca definir e implementar una propuesta de arquitectura de miniCPD para nodos *edge*, basada en el paradigma de la hiperconvergencia y sobre *software open source*. Un diseño portable orientado al despliegue y orquestación de aplicaciones, fácilmente escalable gracias a la virtualización del *hardware*, e integrable dentro de una nube híbrida (que combine *hardware on-premise* con *cloud*) para la redistribución dinámica de cargas de trabajo. Otros objetivos para el proyecto son:

- Asegurar la calidad de servicio y la alta disponibilidad (*High Availability* o HA). Los componentes del CPD deben procurar garantizar la ausencia de un solo punto de fallo, para asegurar la continuidad en el servicio de las aplicaciones.
- Incluir en el diseño un sistema de monitorización continua del estado del *cluster*, que permita tanto el envío de alarmas, como la ejecución de medidas autocorrectivas.
- Ofrecer un método rápido, fiable y flexible de hacer despliegues automatizados en entornos de producción mediante (una vez más) herramientas *open source*.

1.3. Fases y métodos

Para el desarrollo de este proyecto, las fases seguidas han sido las siguientes:

En primer lugar, hubo una fase de conceptualización, en la que se afianzaron conceptos, se estudiaron las tendencias actuales en arquitecturas de CPDs, y se esclarecieron las necesidades *software* a la hora de diseñar uno.

A continuación, se estudió qué herramientas *software* de código abierto serían las más idóneas para la arquitectura del CPD. Estas tenían que ser compatibles entre ellas, ofrecer las funcionalidades deseadas, y suponer el menor coste de soporte posible.

Con las herramientas decididas, se continuó con el diseño teórico del CPD. En este diseño se tuvieron en cuenta no solo las interacciones entre componentes a varios niveles (reparto de

recursos *hardware*, redes, etc.), sino también los ajustes a realizar de acuerdo con las posibilidades de cada herramienta.

El siguiente paso fue el despliegue del diseño en un entorno de laboratorio. Primeramente, se hizo de forma "manual" para poder definir detalladamente los pasos necesarios para instalar y configurar cada componente. Ya con los pasos afianzados, se procedió desarrollar los *scripts* que permiten hacer dicha instalación de forma automatizada y con la menor interacción humana posible (*Zero Touch*).

Con el prototipo de CPD levantado, el siguiente paso fueron las pruebas de rendimiento o *benchmarks*, y el despliegue de una aplicación de prueba para un caso de uso real como puede ser albergar un *core* de 5G en el *edge*.

Como última fase de este proyecto, estuvo la elaboración de esta misma documentación, junto con la definición de nuevas líneas de desarrollo, limitaciones conocidas y funcionalidades implementables.

1.4. Material

Para la elaboración de este trabajo, se contó con una serie de servidores ubicados en el centro I+D de Telefónica en el parque tecnológico de Boecillo.

Al comienzo de mi investigación tuve acceso a un conjunto de 4 servidores Supermicro X9DRFR. Estos poseen 24 CPUs Intel Xeon de 2.10GHz, 62.9GB de memoria RAM (más 8GB de swap), dos discos duros HDD de 465.8GB útiles de almacenamiento y tres tarjetas de red dos de ellas de par trenzado de 1Gbit/s y una de fibra de 10Gbit/s.

En la segunda mitad del proyecto traspasé el desarrollo a otro *cluster* esta vez de 3 servidores siendo uno de ellos desigual a los otros dos. Los dos primeros son servidores GIGABYTE MH70-HD1-ZB-XX con 56 CPUs Intel Xeon de 2.4GHz, 125.8GB de RAM (más 8GB de swap), dos discos SDD uno de 447GB y el otro de 1.8TB, y dos tarjetas de red una de ellas con puerto para par trenzado de 1Gbit/s y la otra para puerto de fibra de 40Gbit/s. El tercer servidor es un TYAN B5631G88V2HR-2T-N con un procesador Intel Xeon Gold de 32 núcleos de 2.1GHz, 62.5GB de RAM (más 8GB de swap), dos discos SSD de 931.5GB y un NVMe de 476.9GB, y tres tarjetas de red dos de ellas para fibra de 10Gbit/s y una de ellas para par trenzado de 1Gbit/s.

Todos los servidores están conectados entre sí a través de dos *Switches*, uno de fibra óptica y otro para los enlaces de par trenzado. Ambos *switches* comparten la misma subred de gestión y poseen acceso a otro *switch* Gateway hacia la red exterior.

1.5. Estructura de la memoria

La estructura de la presente memoria es la siguiente. Después de este primer capítulo de introducción y motivación, se continúa con un capítulo sobre el estado del estado del arte, en el

que se revisan los conceptos de computación en la nube y el *edge computing*, las tendencias actuales en el diseño de arquitecturas de centros de datos entre las que se encuentra la hiperconvergencia, y cómo entra en juego la nube pública en el planteamiento moderno de infraestructura IT.

En el siguiente capítulo, se explica el *software* que se va a utilizar en el despliegue del CPD y las razones para escogerlo, siendo las piezas de este *software* más importantes el gestor de nube y la aplicación para *software* definido por *software*.

En el capítulo de Diseño se muestra la propuesta de arquitectura de CPD, viendo en detalle los requerimientos *hardware* y *software* de los servidores, y se definen directrices y recomendaciones a la hora de ampliar el CPD.

En el capítulo sobre la implementación se describen de manera resumida los pasos necesarios para levantar la arquitectura en un entorno de producción, a la vez que se explican las diferentes herramientas para hacerlo.

En Conclusión y Resultados se muestra el rendimiento logrado con esta arquitectura y los medios disponibles en una serie de *benchmarks* variados, y se analizan los posibles cuellos de botella en el diseño y recomendaciones.

Por último, en líneas futuras, se definirán las líneas para el desarrollo posterior del modelo además de posibles funcionalidades a incluir.

2. ESTADO DEL ARTE.

2.1. Computación Edge

Durante años, la principal tendencia en la industria de los centros de datos ha sido la de seguir un modelo de economía de escala. Los llamados centros de datos de hiperescala buscan abaratar los costes operativos por *rack*, repartiendo los gastos generales en el mayor número posible de servidores [8]. Esta tendencia sigue vigente hoy en día principalmente debido a la migración gradual de gran parte de la infraestructura TI tradicional a la nube pública hospedada por los principales gigantes tecnológicos (Amazon, Google, Microsoft…). No obstante, si bien la centralización geográfica del procesamiento de las aplicaciones modernas tiene sus ventajas, genera a su vez una serie de inconvenientes cada vez más acentuados.

El principal punto débil de la centralización de la nube es cómo todos los datos deben ser transportados desde su origen hacia un gran nodo central para su procesado o almacenamiento. Este proceso conlleva dos ineficiencias, en primer lugar, las aplicaciones intensivas en volumen de datos requieren gran cantidad de ancho de banda desde su origen hasta su destino pudiendo perjudicar el tráfico compartido de otras aplicaciones más livianas. En segundo lugar, en aplicaciones meramente centralizadas la latencia de la red (tiempo que tardan los datos en llegar su destino) suele ser excesiva para gestionar con eficacia determinados casos de uso. A esto hay que añadir como aplicaciones que dependen de un procesamiento continuo y asegurado de datos, quedan vulnerables ante posibles caídas o retrasos del nodo central.

Otro gran problema que abordar es el inminente aumento exponencial en el volumen de datos generados e intercambiados, principalmente por la irrupción de tecnologías como el Internet de las Cosas (IoT), la Inteligencia Artificial o la implantación del 5G como estándar de comunicaciones móviles. Por cuantificar, durante el año 2025 la compañía IDC prevé que existan alrededor de 55.900 millones de dispositivos conectados a la red, y que solamente los relacionados con el IoT generen más 79.4 ZB de información [9]. Ciertas aplicaciones IoT van a requerir de una cantidad ingente de datos para algunas de sus aplicaciones, los cuales van a requerir ser filtrados y agregados antes de poder ser procesados por una nube centralizada. En el caso de los vehículos autónomos, se estima que sus sensores generen entre 3 y 40Gbps por vehículo que deben ser transmitidos [10]. Las redes 5G por otro lado suponen un aumento masivo en los datos al permitir conectar hasta un millón de dispositivos por kilómetro cuadrado [1].

La computación en el borde o *edge computing* surge como solución a estos problemas acercando el procesamiento y almacenamiento de los datos relativos a un proceso o servicio a una ubicación física lo más próxima posible a la fuente que los genera y utiliza. Mantener los datos en la periferia de la red donde se generan y consumen permite gestionar más eficazmente el flujo de datos hacia el usuario, en lugar de tener que conectarse a un gran CPD centralizado [11].

Los centros de datos en el borde no tienen como objetivo reemplazar a la infraestructura ya migrada a una nube centralizada, sino complementarla y librarla de cargas de trabajo más intensivas en el transporte de datos pequeños. Aplicaciones con un volumen masivo de datos podían otrora someter a las redes de acceso al núcleo de la red a un gran estrés, ocupando casi todo el ancho de banda. El *edge* también ayuda con aplicaciones cuyos datos no se pueden o deben migrar a la nube, ya sea por problemas de latencia (aplicaciones en tiempo real) o incluso de seguridad (información sensible, que no queremos que llegue a servidores de terceros).

Un CPD *edge* no tiene por qué ser meramente de *hardware* propietario, algunos proveedores cloud también poseen servicios de computación *edge* con la posibilidad de escoger entre diversas ubicaciones. No obstante, un problema a considerar del uso de la nube pública es que, a pesar de la gran escalabilidad de recursos que esta otorga, numerosas empresas acaban dependiendo en demasía de ella (cayendo en *vendor lock-in*). Aunque un nodo *edge* con *hardware* propietario pueda complementarse con proveedores *cloud* cuando sus recursos se quedan cortos, es deseable que aproveche lo máximo posible sus recursos *on-premise* y estos puedan ser ajustados de forma sencilla dependiendo de la aplicación para la que pretenda ser usado el miniCPD. Se busca, por tanto, que el diseño del nodo *edge* simplifique la adición o sustracción de equipamiento *on-premise* sin complicaciones, además de permitir una rápida migración/replicación de datos/procesos entre servidores/CPDs. Es aquí donde entra el concepto clave de la hiperconvergencia.

2.2. Hiperconvergencia (HCI)

La hiperconvergencia es un término usado para referirse a la integración de los componentes de un CPD tradicional (computación, redes y almacenamiento) en un mismo servidor físico mediante una capa *software* de virtualización [12]. Sin embargo, para entender la razón de la elección de este tipo de modelo para el CPD es preciso explicar y entender las diferentes alternativas de modelo de CPD y cómo han ido evolucionando las tendencias a lo largo del tiempo, desde los modelos tradicionales hasta las actuales infraestructuras hiperconvergentes pasando por los modelos convergentes.

2.2.1. MODELO TRADICIONAL

La arquitectura tradicional de un CPD requiere del uso de *hardware* y *software* especializados para cada uno de los servicios ofrecidos en el centro. En los *racks* de los CPDs tradicionales se pueden encontrar servidores heredados de diferentes marcas para la computación de aplicaciones, equipos dedicados al almacenamiento al servicio de redes SAN (*Storage Area Network*) y/o NAS (*Network Attached Storage*), y *switches* y enrutadores suficientemente potentes para la gestión de las redes SAN y LAN (*Local Area Network*). Los servidores físicos se dedican cada uno a una sola aplicación y se escalan verticalmente, es decir, ampliando los recursos disponibles

como CPU o memoria a medida que aumenta la carga de trabajo. Del mismo modo los sistemas de almacenamiento se dimensionan y escalan independientemente a los servidores de cómputo.

Todo esto genera soluciones complejas con gran cantidad de dispositivos, sistemas y *software* diferentes, además de necesitar personal técnico para cada componente de infraestructura (experto en ciberseguridad, almacenamiento, redes). Esta mezcla heterogénea de componentes requiere de buena planificación previa, resultando en un proceso largo y costoso a la hora de optimizar el rendimiento del conjunto, dando lugar a menudo a problemas de coordinación, interoperabilidad, más demanda de energía, espacio físico y en resumen mayores costes [13]. En la Ilustración 1, se muestra un esquema simplificado del modelo tradicional. Se puede observar cada uno de los componentes separados físicamente, con los inconvenientes de los que ya se ha hablado.

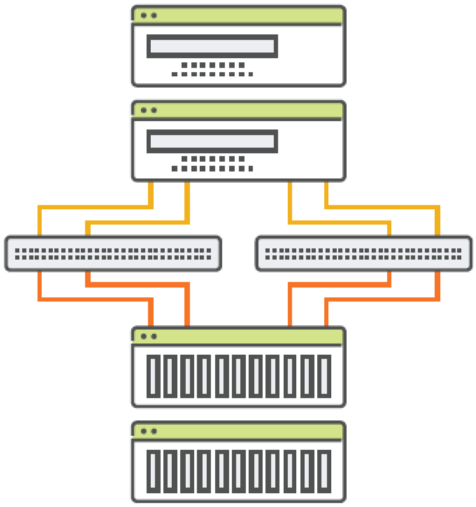


Ilustración 1. Infraestructura tradicional, con separación física entre servidores de almacenamiento y de computación [13]

2.2.2. MODELO CONVERGENTE

Como evolución de los CPD tradicionales surge el concepto de *infraestructura convergente* (CI), procurando solucionar principalmente los problemas de la heterogeneidad. En este tipo de arquitecturas, un único proveedor combina los mismos componentes subyacentes de la arquitectura tradicional en un único *rack* preconfigurado de equipos y herramientas que trabajan en conjunto.

Mediante el uso de bloques preconfigurados, es posible realizar despliegues repetibles y modulares con los que formar o ampliar un CPD, permitiendo a las empresas reducir enormemente el tiempo y gastos que emplearían en diseñar, desplegar, configurar y optimizar sus CPDs. Tampoco tendrían que preocuparse de problemas de compatibilidad e integración entre componentes, ni de optimizar los gastos relativos a cableado, alimentación y refrigeración. En esta infraestructura se estandariza también el uso de una capa de *software* de virtualización o hipervisor en los diferentes componentes de cada servidor para su gestión centralizada vía *software*.

No obstante, a pesar de las ventajas en simplicidad y compatibilidad de tener soluciones preconfiguradas de componentes, la CI aún conserva parte de los problemas de la arquitectura tradicional. Aún tenemos un conjunto de componentes dedicados interconectados, cada uno con su propia capacidad de expansión. Esto sigue suponiendo una limitación en el crecimiento del CPD, la complejidad de tener que gestionar numerosos componentes separados, y una menor eficiencia de cada componente al necesitar cada uno su propia memoria, CPU, almacenamiento interno e hipervisor. A todo esto, se añade el riesgo de caer en *vendor lock-in* con el proveedor de CI.

2.2.3. MODELO HIPERCONVERGENTE

Finalmente, con la explosión de las tecnologías de virtualización en los últimos años, aparece la infraestructura Hiperconvergente (HCI) como una evolución natural de la CI, en la que computación, *networking*, y *storage* se combinan en un mismo servidor físico o *host* a través un único hipervisor, reduciendo al mínimo el *hardware* redundante del CPD [14]. La unión de varios servidores hiperconvergentes es capaz de formar un *cluster* en el que los recursos de cada *host* se combinan aditivamente en una misma plataforma centralizada, y se manipulan libremente por *software* gracias a los hipervisores. Quedan pues los recursos de cada *host* desvinculados y agrupados en *pools* comunes al *cluster*. Con los *pools* se pueden asignar dinámicamente memoria, procesamiento y almacenamiento para el levantamiento servidores virtuales o contenedores, donde finalmente se ejecutan las aplicaciones deseadas.

En la Ilustración 2 se puede apreciar la convergencia los antiguos componentes de un CPD tradicional, en un reducido número de servidores.

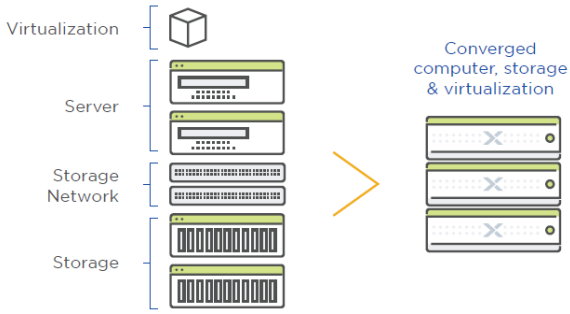


Ilustración 2. Arquitectura Hiperconvergente [13]

Un CPD formado por servidores HCI constituye un Centro de Datos Definido por *Software* o SDDC, donde la infraestructura física se puede aprovisionar y gestionar de forma rápida y flexible empleando plantillas definidas por *software* y distintas APIs. Esta asignación dinámica de recursos permite la creación de nubes privadas que ofrezcan como servicio lo que en términos de *cloud* se denomina Infraestructura como Servicio (IaaS).

Dentro de las ventajas de contener recursos concentrados en un reducido número de *hosts* están la simplicidad en la gestión, mantenimiento y replicación de un menor número de dispositivos, mayor flexibilidad y escalabilidad de servicios (se pueden añadir nuevos *hosts* o mejorar los existentes sin necesidad de detener el servicio), mejor eficiencia energética reduciendo

las necesidades de *hardware* redundante, y la reducción del tiempo de respuesta al mínimo, al disponer las instancias virtuales de todos los recursos necesarios en el mismo *host*.

Si bien una implementación de arquitectura hiperconvergente puede implicar un mayor gasto inicial en comparación con las arquitecturas tradicionales o convergentes, a largo plazo la eficiencia, escalabilidad y facilidad en el despliegue de entornos hiperconvergentes conllevan una clara reducción de costes totales. Un ejemplo de ello es cómo mediante la hiperconvergencia no son necesarios dispositivos dedicados al almacenamiento o la red, como SANs o enrutadores programables, evitando así además el retardo que implicaría su uso [15].

2.3. Nube híbrida y gestores de nube

También en los últimos años se consolida en el mercado la nube pública, y comienza una tendencia de migración a la nube de los CPDs que no se ha detenido hasta nuestros días, teniendo los restantes CPDs que buscar un equilibrio de costes entre la posesión de equipamiento *on-premise*, y el de alquiler en la nube.

Las nubes públicas al igual que las privadas utilizan una serie de tecnologías para: virtualizar los recursos *hardware* en grupos compartidos, agregar un sistema de control administrativo para todo el entorno y crear funciones de autoservicio automatizadas [16]. Son posibles por tanto entornos híbridos, en los que se combinan las capacidades de diferentes infraestructuras de nubes tanto públicas como privadas en un solo *cluster* virtual, aprovechando los beneficios de cada una frente a la otra.

Utilizando las APIs correspondientes a cada tipo de nube, un *software* de gestión de nube posee gran flexibilidad para elegir dónde y cómo distribuir cargas de trabajo, y cómo aprovechar los recursos de nube pública y privada de manera más eficiente. Una tarea puede ejecutarse no solo en una nube pública o privada, sino también en varias al mismo tiempo independientemente de su localización, lo que se denomina *multi-cloud*. En una nube híbrida las aplicaciones se despliegan y controlan de manera unificada mediante las mismas herramientas y procedimientos, indiferentemente a si están en nube pública o privada, lo que posibilita y facilita el balanceo de cargas, la alta disponibilidad de servicios y la migración de datos y servicios entre nubes de forma dinámica y sin tener que interrumpir el servicio.

Para lograr esta integración de nube híbrida, se hace imprescindible la presencia de un gestor de nube que orqueste los procesos de los diferentes *clusters*. Antes de nada, quiero clarificar que con gestor de nube procuro referirme a una herramienta que permita a sus usuarios crear, gestionar y escalar servicios sobre recursos, en nuestro caso virtualizados, provenientes tanto de infraestructura local (nube privada o en propiedad) como remota (nube pública o alquilada). Este término, si bien no está muy extendido, lo uso en referencia al "*cloud management*", término más asentado para la distribución de recursos entre las empresas. Otros términos usados en lugar de gestor de nube son *software* de gestión de nube, plataforma de computación en la nube o *Cloud Management Platform* (CMP).

Un gestor de nube puede tener diferentes funcionalidades, pero las tres características que debe cumplir son:

- Autoservicio: Un buen gestor de nube cuenta con flexibilidad a la hora de acceder a los diferentes recursos *cloud* para poder asignarlos donde se desee. También es capaz de añadir nuevos recursos (*cloud provisioning*) y supervisar el consumo total de ellos que se hace para controlar los costes.
- Automatización del flujo de trabajo: también llamada orquestación, es necesaria para que no sea necesaria intervención humana a la hora de gestionar los recursos *hardware* de la nube. Aumenta la eficiencia operativa al acelerar el despliegue de aplicaciones y reducir los posibles errores humanos que pueda haber al replegarlas.
- Análisis *cloud*: para hacer un seguimiento de las cargas de trabajo en la nube y las experiencias de usuario. Los gestores de nube no solo deben contar con monitorización para todos los servicios que manejan, sino también con una buena observabilidad para los usuarios del estado de las nubes controladas [17].

Es relevante pues escoger bien el gestor de nube a utilizar, así como el resto de las herramientas de virtualización empleadas, desde el hipervisor a una solución de almacenamiento definido por *software*.

3. SOFTWARE EMPLEADO

3.1. Gestor de Nube

Para controlar un miniCPD de *edge computing* allá donde esté, necesitamos contar con un gestor de nube ligero, elástico, dinámico, con un bajo coste de mantenimiento, y que funcione como plataforma IaaS. El gestor debe ser capaz de ser provisionado no solo con recursos de nube pública, sino también con *hardware* en propiedad que el gestor se encargará de virtualizar con los hipervisores que necesite. También es beneficioso que el gestor esté desarrollado en código abierto, para que pueda nutrirse del desarrollo de otros usuarios enriqueciendo la funcionalidad total del sistema.

Partiendo de estos criterios, las opciones que mejor se ajustan y son más empleadas en producción son [18]: OpenStack, OpenNebula, CloudStack, Eucalyptus y Apache CloudStack. Elijo las dos primeras al ser las que tienen mejor comunidad y documentación.

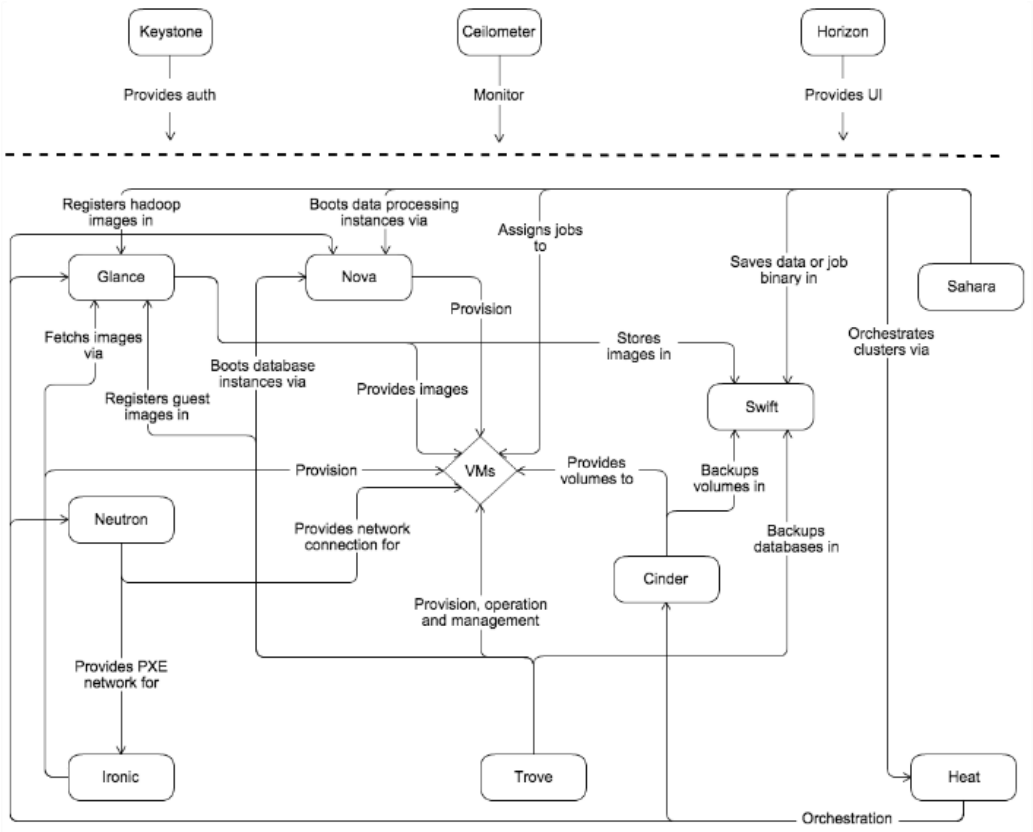


Ilustración 3. Arquitectura de servicios de OpenStack [19]

OpenStack es un gestor de nube *open source* con licencia Apache 2.0, implementado como una colección de servicios y módulos interactivos trabajando juntos mediante APIs. Con OpenStack la nube (pública, privada o híbrida) se puede gestionar tanto por una interfaz de usuario basada en web, por un cliente de línea de comandos, o por peticiones API a sus diferentes servicios. OpenStack ofrece una amplia gama de funcionalidades (redes virtuales, soporte *bare-metal*,

soporte integrado para VMs — *virtual machine*s — y contenedores, etc.) pero generalmente es considerado demasiado complejo y tedioso para desplegar y gestionar. En la Ilustración 3, se puede ver un esquema de su arquitectura conceptual.

OpenNebula es una plataforma *open source* también bajo licencia Apache 2.0, para crear y administrar nubes privadas, públicas e híbridas. Está más orientada a la gestión de entornos heterogéneos y distribuidos que OpenStack, y es compatible con todos los Hipervisores y sistemas operativos más comunes. Como sistema, OpenNebula consiste en una única distribución que ya contiene los subsistemas y elementos necesarios para su funcionamiento. Con OpenNebula la nube se opera tanto desde línea de comandos, como a través de su *Front-end* desarrollado en Ruby.

Ambas soluciones si bien persiguen la misma funcionalidad, son estructuralmente diferentes. Mientras que la arquitectura de OpenStack está constituída por una gran variedad de proyectos *open source* para cada una de sus funciones, OpenNebula sigue una estructura modular que cubre un subconjunto de dichos proyectos OpenStack, con una cantidad mucho menor de variables y mandos [20]. En la Tabla 1 se muestra una comparación simple entre componentes OpenStack y su equivalente en OpenNebula.

	Componente OpenStack	Equivalente OpenNebula
Computación	Nova	Incorporado
Almacenamiento de objetos	Swift	Sin equivalente
Servicios de Imágenes	Glance	Incorporado
Identificación de usuarios	Keystone	Incorporado
Dashboard	Horizon	SunStone
Networking	Neutron	Incorporado
Almacenamiento de bloques	Cinder	Incorporado + plugins
Telemetría	Ceilometer	Incorporado
Orquestación	Heat	OneFlow
Bare-metal	Ironic	Sin equivalente

Tabla 1. Comparación componentes OpenStack y OpenNebula [20]

La mayoría de los servicios de más alto nivel no poseen equivalente en OpenNebula intencionadamente por diseño. Servicios extra como el almacenamiento de objetos pueden ser añadidos mediante VMs externas.

Este enfoque de "todo en uno" no solo facilita despliegues rápidos y sin necesidad de entender y planificar el funcionamiento de decenas de componentes (subproyectos en diferente estado de maduración), sino que también simplifica la gestión de recursos virtualizados. Se podría decir que OpenNebula entiende la nube como si fuera un "DataCenter Virtualizado" tal y como hacen soluciones comerciales como VMware vCloud, mientras que Openstack obedece a una nube más orientada a la provisión *cloud* de infraestructura, como AWS (Amazon Web Services) [21]. Los casos de uso de OpenNebula se centran en una simplificación de la orquestación de recursos virtualizados, mientras que OpenStack busca ser una buena herramienta de provisión de recursos virtualizados bajo demanda (si bien con su gran cantidad de plugins, puede variar su especialización

base). La Ilustración 4 muestra este posicionamiento de los gestores de nube según su modelo y flexibilidad.

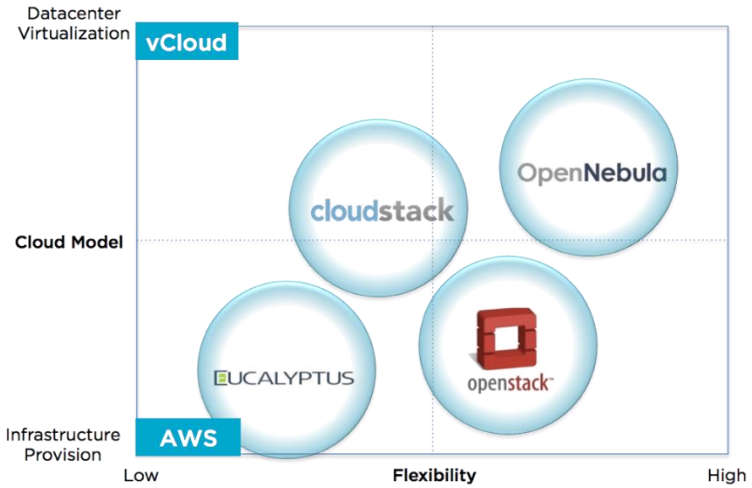


Ilustración 4. Comparación de los principales gestores de nube [21]

OpenStack y OpenNebula también difieren en su visión de producto. OpenStack requiere de una integración compleja para lograr una infraestructura de nube funcional y carece de soporte comercial, haciendo prácticamente necesaria la contratación de "*vendor stacks*", esto es, soluciones comerciales de herramientas OpenSource complejas modificadas para satisfacer determinados casos de uso. Estas distribuciones modificadas de OpenStack (por HP, Red Hat, IBM…) cuentan con funciones extendidas, componentes mejorados y *software* patentado, pero no entrarían en la categoría de *open source* y suponen caer en *vendor lock-in* (imposibilidad manifiesta para cambiar a proveedores alternativos). OpenNebula es un único producto *open source* sencillo de instalar, usar y actualizar, con la posibilidad de contratar soporte comercial oficial a un coste bastante menor que las distribuciones comerciales de OpenStack.

Para miniCPDs en el borde de la red, OpenNebula no tiene carencias funcionales ni de prestaciones que invaliden la selección. Es una solución más ligera, sencilla y compacta que OpenStack, lo que se traduce en beneficios de coste de despliegue, operación y mantenimiento para la infraestructura en el borde de la red.

3.1.1. OPENNEBULA

Tal y como dice su documentación, OpenNebula es una plataforma *open source* de computación *cloud* y *edge* para la creación y gestión de infraestructuras de centros de datos heterogéneas y distribuidas. Combina tecnologías de virtualización y contenerización con una administración multiusuario, provisión automatizada y elasticidad para ofrecer aplicaciones y servicios en entornos privados, *edge* e híbridos. OpenNebula es compatible nativamente con los hipervisores KVM, LXC, Firecracker y VMware, además de poder integrar *hosts* con otros hipervisores como Xen mediante addons [22].

La arquitectura estándar de OpenNebula mostrada en la Ilustración 5, debe poseer:

- Un *cluster* de gestión *cloud* con el *Front-end* de OpenNebula y todas sus dependencias. Son uno o más *hosts* (en caso de configuración de alta disponibilidad) a través de los cuales se controlan el resto de *clusters* del entorno, interconectados a través de una subred de gestión.
- La Infraestructura *cloud*, compuesta por uno o más *clusters* con uno o más "*hosts* hipervisores" y el sistema de almacenamiento escogido. Para poder incluir un *host* dentro de un *cluster*, este simplemente debe poder ser accesible por el *Front-end* por ssh y poseer un paquete de contextualización de OpenNebula según el hipervisor que posea.

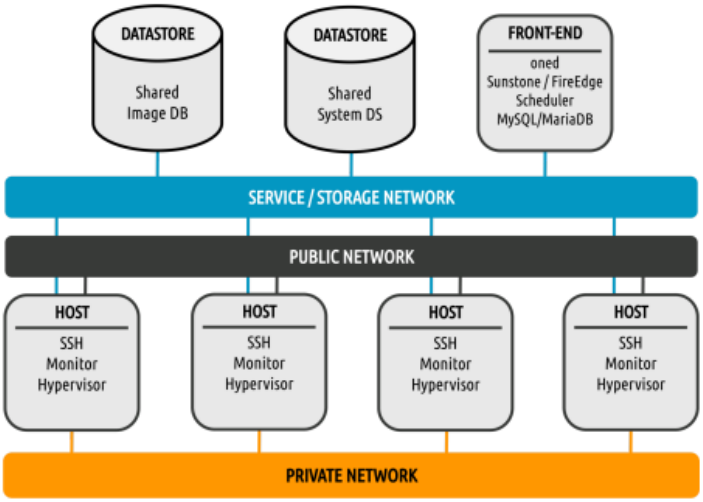


Ilustración 5. Arquitectura de referencia [23]

Partiendo de esta sencilla arquitectura, OpenNebula forma un sistema modular capaz de adaptarse fácilmente a cualquier infraestructura y ampliarse con nuevos componentes *software*. Los *hosts* hipervisores de OpenNebula, si bien pueden usar una gran variedad de hipervisores, solo poseen soporte nativo de KVM (Kernel-based Virtual Machine) para máquinas virtuales (VMs), LXC para contenedores, Firecracker para micromáquinas virtuales (microVMs) y VMware vCenter para infraestructura de VMWare. A partir de ahora para referirme a las instancias virtuales levantadas mencionaré únicamente a las VMs, al ser las más empleadas en OpenNebula.

OpenNebula cuenta con gran cantidad de componentes tal y como se muestra en la **¡Error! No se encuentra el origen de la referencia.**, pero los que tienen más impacto a la hora de definir el CPD son:

- *Daemon* de OpenNebula (oned): Servicio central de la plataforma de gestión de la nube. Es capaz de controlar a los *hosts* hipervisores, así como crear y manipular redes virtuales, el almacenamiento virtual, los grupos y usuarios y sus máquinas virtuales y contenedores. Expone también una API XML-RPC para otros servicios y usuarios finales.
- Base de datos. Almacena el estado de la nube en la base de datos SQL seleccionada. Por defecto es SQLite, aunque a medida que crece la nube es recomendable pasar a MySQL o MariaDB.

- Planificador/*Scheduler*: *Daemon* encargado de planificar las VM pendientes en los *hosts* hipervisores disponibles.
- Edge Cluster Provision: Componente al servicio del cliente OneProvision, para la creación automatizada de *clusters* OpenNebula preconfigurados mediante proveedores de *cloud* pública o *edge*. A pesar de la versatilidad de esta herramienta, los diseños de *cluster* propuestos son limitados y no se adecúan a lo buscado por este trabajo, por lo que no se usa esta funcionalidad.
- Collectd: *Daemon* parte de oned que recopila información del estado de los *hosts* de hipervisores y sus VMs/contenedores. Esta información es enviada periódicamente por los *hosts* hipervisores al *Front-end*.
- OneFlow: Servicio opcional para la definición y orquestación de aplicaciones compuestas por VMs interconectadas con dependencias de despliegue entre ellas.
- OneGate: Otro servicio opcional que permite a las VMs recibir o enviar información de monitoreo desde/hacia OpenNebula, para la recopilación de métricas, detección de problemas en sus aplicaciones o la activación de reglas de elasticidad de OneFlow.

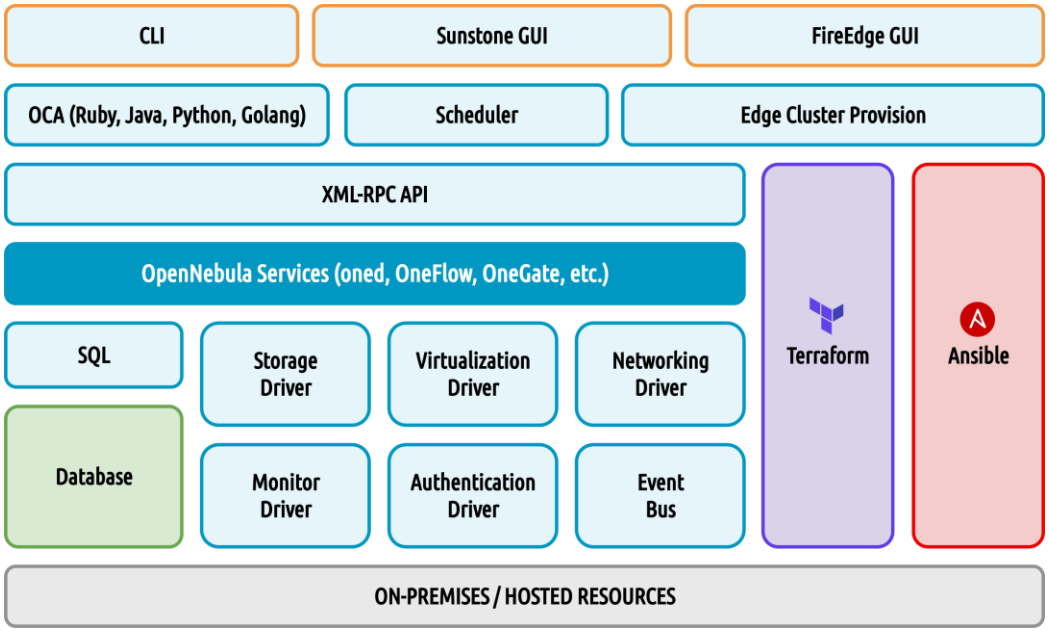


Ilustración 6. Componentes de OpenNebula [22]

En cuanto a las interfaces de sistema de OpenNebula, las más relevantes son:

- Sunstone: Interfaz web de gestión tanto para usuarios finales como para administradores. En ella se pueden realizar gestión de recursos y operaciones típicas.
- FireEdge: Otra Interfaz web de gestión en desarrollo especializada en ofrecer una GUI (Graphical User Interface) para OneProvision. Es útil también al funcionar como proxy de Sunstone para el acceso remoto a las VMs (vía VNC, RDP y ssh)
- CLI: Librerías de comandos para el control individual de cada uno de los componentes de OpenNebula en los *Front-end*.

- API XML-RPC: Interfaz principal para manipular recursos directamente (VMs, redes virtuales, imágenes, usuarios, *hosts*, *clusters*…)
- API OpenNebula Cloud (OCA): interfaz simplificada para usar la API XML-RPC con soporte para Ruby, Java, Golang y Python.
- API OpenNebula OneFlow: API de tipo REST para la creación, control y monitorización de servicios desplegados en VMs interconectadas.

Una vez entendida la arquitectura de OpenNebula, hay que entender cómo esta gestionaría los recursos de nuestro CPD Hiperconvergente. Las aplicaciones desplegadas en *clusters* en OpenNebula usan las capacidades de computación de los *hosts* hipervisores que las contengan. El *networking* entre las VMs y contenedores levantados en los *hosts* se realiza a través de redes virtuales (vnets) definidas por OpenNebula sobre las redes físicas.

No obstante, para la gestión del almacenamiento tanto de las imágenes como de las instancias de VMs/contenedores las opciones se diversifican. Es necesaria una solución unificada y compartida de almacenamiento que permita que todos los datos se encuentren disponibles de manera equitativa para cada *host*.

3.2. Almacenamiento Virtualizado

OpenNebula ofrece una gran variedad de posibilidades para almacenar las Máquinas Virtuales y contenedores desplegados, pero para escoger la opción más adecuada hay que entender primero su concepción de los Datastores.

Un Datastore no es más que un medio de almacenamiento de imágenes de disco al servicio de uno o más *clusters*. Has tres tipos de Datastores siendo solo dos de ellos necesarios.

- **Datastore de Imágenes**: repositorio local de imágenes de sistemas operativos, volúmenes de datos persistentes y CD-ROMs.
- **Datastore de Sistema**: almacenamiento de los discos de las VMs en ejecución. Las imágenes se mueven de/hacia un Datastore de imágenes cuando las VMs se despliegan/terminan.
- **Datastore de Ficheros y Kernels**: alberga ficheros planos como Kernels, RAMdisks o ficheros de contextualización de VMs. No es necesario para el funcionamiento de un *cluster*.

A partir de estos Datastores, OpenNebula permite principalmente cuatro soluciones de *storage*:

- **Local**: Almacenamiento local que emplea los dispositivos conectados directamente a cada *host* hipervisor. Existe también la configuración llamada OneStor, que añade caché y mecanismos de migración de imágenes. Es una solución eficiente, pero carece de tolerancia a fallos en las VMs y al no virtualizar el espacio de sus dispositivos no es hiperconvergente.

- **NFS/NAS**: Los *hosts* hipervisores hacen uso de volúmenes de ficheros compartidos en servidores NAS o NFS (Network File System). Esta solución requiere servidores aparte dedicados meramente al almacenamiento por lo que tampoco cumple con el paradigma de la hiperconvergencia.
- **SAN**: Los *hosts* hipervisores acceden a dispositivos de almacenamiento de bloques (LUNs – Logical Unit Number) montados con LVM (*Local Volume Manager*) a través de protocolos como iSCSI (*Internet Small Computer System Interface*) o Canal de Fibra (FC). Usar volúmenes lógicos en vez de sistemas de ficheros incurre en menos sobrecarga que una red NAS, pero de nuevo estamos empleando dispositivos externos para el almacenamiento.
- **Sistemas de almacenamiento distribuido:** Utilizar herramientas de *Storage* Definido por *Software* (SDS) permite compartir un espacio de almacenamiento común a todos los *hosts* hipervisores ubicado en sus propios dispositivos de almacenamiento. Permiten asegurar alta disponibilidad y escalabilidad para las aplicaciones, pero requieren configuraciones mucho más complejas que las anteriores

De las cuatro opciones las única que cumple el requisito de hiperconvergencia es la de sistemas SDS, sin embargo, existen varias posibilidades que se pueden implementar en OpenNebula, requiriendo por lo tanto realizar un estudio para escoger el mejor sistema de SDS para nuestras necesidades. Según su documentación, OpenNebula es compatible con Ceph, GlusterFS, StorPool y LINBIT para la creación de *pools* de almacenamiento flexibles [24]. Descarto en primer lugar GlusterFS al consistir en realidad en una solución de NFS/NAS no pensada para hiperconvergencia, y StorPool por ser una solución comercial (no *open source*). Entre Ceph y LINBIT, me quedo con la primera al ser la solución más utilizada en entornos similares [25] y por poseer driver nativo en OpenNebula. LINBIT requiere de la instalación de su propio addon y no ha sido probada en demasiados entornos de producción, por lo que la documentación disponible es considerablemente menor.

3.2.1. CEPH

Ceph es un *software open source* de almacenamiento distribuido de una escalabilidad casi ilimitada y con una gran resiliencia ante errores. Permite almacenamiento de bloques, de ficheros y de objetos en un único sistema unificado, gratuito y sencillo de gestionar, siendo el espacio de almacenamiento de bloques la mejor opción para albergar y montar las imágenes de las VMs. Ceph está diseñado para no tener nunca un único punto de fallo y hace innecesaria la presencia de elementos como RAID o la paridad, ya que toda la inteligencia para escalar, proteger y replicar los datos se impulsa por *software*.

Ceph se compone por una serie de procesos o *daemons* individuales distribuidos por todo el *cluster*. Los más importantes son:

- **Monitors (MON)**: se encargan de mapear el *cluster*, es decir, conocer la distribución del resto de *daemons*, y hacer un seguimiento de la salud del *cluster*. También almacenan la

configuración del *cluster* y sus *data pools* y se encargan de la autenticación de usuarios (membresía y estado). El mapeo requiere consenso, por lo que más de la mitad de los MONs deben estar siempre disponibles. Por ello en el *cluster* tiene que haber por lo menos 3 MONs, y a partir de ahí se puede incrementar su número preferiblemente manteniendo una cantidad impar, aunque tampoco es necesario que haya excesivos MON por *cluster*.

- **Managers (MGR)**: se encargan como dice el nombre de gestionar el *cluster*. Su funcionamiento depende de los módulos python que le habilitemos, por ejemplo: recopilar métricas de tiempo de ejecución del *cluster*, levantar una Interfaz de Usuario (UI) web llamada *dashboard* con dichas métricas, gestionar la API REST, etc. Sólo hay un MGR activo por *cluster* en cada momento, estando el resto de *daemons* MGR del *cluster* en reposo/*standby* para cuando el actualmente activo deje de estar disponible. Una de las funciones más importantes del MGR para el proyecto es el módulo del orquestador, que permite tratar a los componentes de Ceph como servicios configurables mediante una plantilla.
- **Object Storage Devices (OSD)**: se encargan de las operaciones de lectura, escritura, replicación, balanceo y recuperación de datos en los dispositivos de almacenamiento. También monitorizan el estado de otros OSDs mediante latidos e informan a los MON y MGR de su estado. Normalmente cada OSD suele tener asignado un dispositivo, aunque hay casos inusuales en los que pueden tener asignados varios para la distribución de diferentes tareas en dispositivos con diferentes velocidades de escritura/lectura. Puede haber desde decenas a miles en un *clúster*, aunque la cantidad mínima suele ser de 3 OSDs para asegurar la redundancia y alta disponibilidad de los datos.
- Luego existen los *daemons* Rados-GW, iSCSI y MDS (*MetaData Server*), pero son para clientes del almacenamiento de objetos, iSCSI o de ficheros respectivamente.

El espacio gestionado por los *daemons* OSD se distribuye en *data pools* o particiones lógicas para diferentes aplicaciones o tipos de almacenamiento. Cada *pool* tiene configurado su tipo de resiliencia, sus *Placement Groups* y sus reglas CRUSH [26].

Por defecto los *data pools* se crean con resiliencia de replicación, en la cual los datos se escriben en varios OSDs, por defecto 3, para asegurar la alta disponibilidad y validación de datos. El otro tipo es el *erasure coding*, en el cual mediante bloques de datos de paridad se consiguen los mismos objetivos que con la replicación empleando menos espacio. Para el diseño del CPD se considera más conveniente usar replicación para la resiliencia, ya que el *erasure coding* requiere más potencia de cómputo, la cual en sistemas hiperconvergentes es compartida con las aplicaciones a desplegar, y la inversión en más dispositivos de almacenamiento siempre es más barata que en mejorar los procesadores.

Los *Placement Groups* (PGs) antes mencionados, son subdivisiones del almacenamiento controlado por los OSDs que se crean cuando se crea un *data pool* en Ceph. Los PGs de un *data pool* se distribuyen lo máximo posible en todos los OSDs en todos los *hosts* de manera que la información se ubique lo más distribuida posible cumpliendo la replicación designada.

Ceph está basado en RADOS (*Reliable Autonomic Distributed Object Store*), un sistema de almacenamiento de objetos fiable, con una escalabilidad teóricamente ilimitada y distribuida en *hosts* de almacenamiento inteligentes que se autoregeneran y autoorganizan [27]. En RADOS, la información se reparte en unidades de almacenamiento llamadas objetos que contienen un ID único, metadatos y la información que guardar en sí. Los objetos, de 4MBs por defecto, se asignan a un PG principal, para a continuación replicarse en otros PGs de OSDs diferentes para garantizar redundancia. La distribución de los objetos en los PGs del *cluster* viene dada por CRUSH (*Controlled Replication Under Scalable Hashing*), un algoritmo de posicionamiento pseudo-aleatorio, determinista y de rápida calculación. CRUSH adapta su comportamiento acordemente a las reglas definidas para cada *pool* del *cluster*, y al peso depositado en los dispositivos para asegurar una distribución de objetos de probabilidad uniforme. En la Ilustración 7 se muestra una representación visual del guardado de información mediante CRUSH.

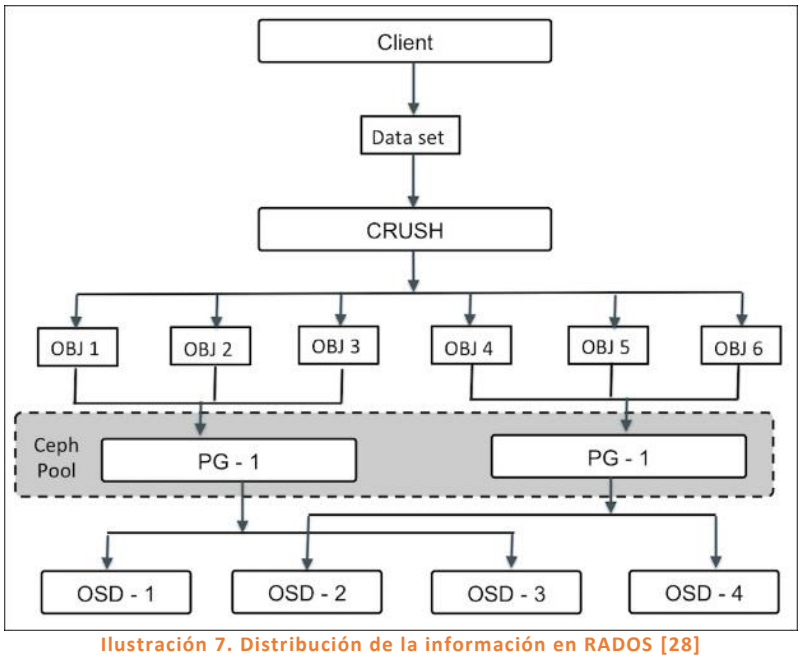


Ilustración 7. Distribución de la información en RADOS [28]

Cuando un cliente de Ceph quiere escribir o leer información, este puede conocer exactamente el PG (y sus OSDs) de los objetos en los que se subdivide ejecutando localmente el algoritmo CRUSH, y sin depender de una tabla central de consulta. Los clientes solamente necesitan ajustar CRUSH mediante el "CRUSH map" del *cluster* que deben consultar a un *daemon* MON. Entre sus funciones, el CRUSH map del *cluster* posee la lista de OSDs disponibles y las reglas que rigen como se replican los datos en cada pool del *cluster*.

RADOS establece la API librados, a partir de la cual se definen las interfaces de los diferentes tipos de almacenamiento y funcionalidades de alto nivel de Ceph. La utilidad para la abstracción del espacio de almacenamiento de bloques es RBD (Rados Block Device), la cual posee su propio módulo para el Kernel de Linux desde la versión 2.6.39 (en 2011) y un *driver* para el hipervisor QEMU/KVM (Quick Emulator), usado por OpenNebula para las VMs, a través de libvirt. A continuación, en la Ilustración 8 se muestra un esquema de la arquitectura básica de Ceph.

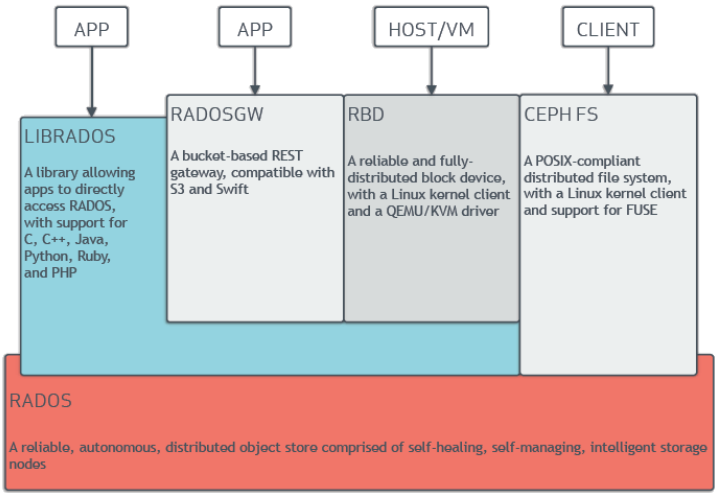


Ilustración 8. Arquitectura de Ceph [29]

Hay que entender que con Ceph, al igual que con cualquier plataforma de SDS, los datos almacenados se depuran periódicamente. Los OSDs están continuamente realizando operaciones en segundo plano como redistribución de carga o comprobación de consistencia, lo que con volúmenes de datos considerables puede requerir gran cantidad de computación y memoria.

Para un diseño hiperconvergente, hay que tener cuidado con que los servicios de Ceph no perjudiquen el rendimiento de las VMs desplegadas en los *hosts* y sus aplicaciones, por lo que es necesario establecer límites en los recursos disponibles para Ceph.

En todo caso, es necesario tener en todo momento una forma fiable de conocer el estado y salud de los *hosts* y sus herramientas en tiempo real y es ahí donde entran en juego las herramientas de monitorización.

3.3 Monitorización

En este apartado no existe la mayor duda sobre qué usar, ya que ambas Ceph y OpenNebula integran y recomiendan el uso del *stack* Prometheus-Grafana para su monitorización. El uso de ambas herramientas — la primera para la recopilación de métricas y la segunda para visualizarlas — se ha convertido en el estándar de facto de la industria para la monitorización de sistemas nativos *cloud* modernos.

3.3.1. PROMETHEUS

Prometheus es un sistema de monitorización y alerta de código abierto escrito en Golang. Permite la recopilación periódica de métricas numéricas de diferentes aplicaciones y servicios formando series temporales. Los principales tipos de métricas que emplea Prometheus son: Counter para valores solamente incrementables o reseteables a cero, Gauge para valores numéricos que pueden aumentar o disminuir, e Histogram para agrupaciones de valores acumulativas en el tiempo.

El ecosistema de Prometheus, mostrado en la Ilustración 9, está compuesto por una serie de componentes, algunos de ellos opcionales:

- **Servidor Prometheus principal**: recoge las métricas, las guarda en su base de datos local (TSDB – Time Series Data Base) y permite su consulta mediante peticiones en formato PromQL, un lenguaje de peticiones propio de Prometheus. En la configuración de Prometheus se definen una serie de tareas (*jobs*) con los objetivos (*targets*) a los que hacer peticiones para obtener las métricas. También es posible definir en él reglas o métricas compuestas, además de alertas si las métricas cumplen determinadas condiciones.
- **Client libraries**: librerías en diferentes lenguajes de programación para implementar la creación de métricas Prometheus en todo tipo de aplicaciones. Estas librerías ya están incluidas tanto en Ceph como en OpenNebula.
- **NodeExporters**: Aparte de las librerías de clientes, existen una serie de *daemon exporters* para exponer métricas a Prometheus, tanto oficiales como de aplicaciones de terceros. Entre los *exporters* oficiales destaca el Node Exporter con diferentes métricas relativas al *hardware* del servidor. De terceros existen tanto el *exporter* de Ceph como el de OpenNebula o Libvirt.
- **Alertmanager**: servicio aparte para el reenvío de alertas generadas por aplicaciones como el servidor Prometheus.

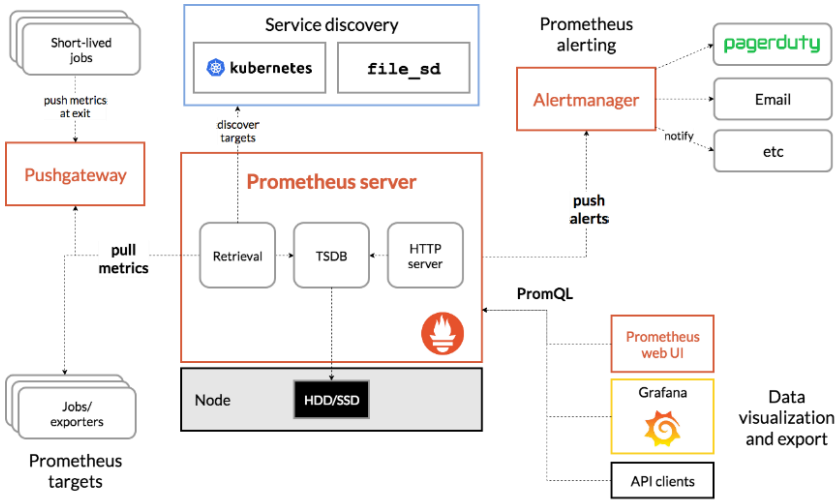


Ilustración 9. Arquitectura del Ecosistema de Prometheus [30]

El servidor Prometheus expone su propia interfaz web para la consulta tanto de métricas como de gráficas sencillas mediante peticiones en PromQL. No obstante, esta visualización de datos cuenta con numerosas limitaciones al estar pensada como una herramienta de test o debug, y por ello está bastante extendido el uso de una herramienta aparte para ello.

3.3.2. GRAFANA

Grafana es una herramienta *open source* creada para la visualización de datos de series temporales en paneles de control (*dashboards*) personalizables. Es un servicio independiente que permite recoger y mostrar información de diferentes bases de datos en una única interfaz totalmente personalizable.

Los *dashboards* en Grafana pueden ser cargados mediante plantillas reusables y personalizables, que toman sus datos en tiempo real a partir de las diferentes fuentes de datos (*data sources*) de las que disponga la instancia de Grafana.

Entre los diferentes *data sources* permitidos, se encuentra Prometheus como uno de los primeros soportados nativamente, lo que significa que Grafana es capaz de hacer peticiones directamente a la API de Prometheus en promQL para la obtención de los datos necesarios para su uso en múltiples *dashboards*.

4. DISEÑO

4.1. Consideraciones Generales

Una vez explicados los componentes *software* a usar, es momento de definir cómo estos se van a integrar entre sí, y los requisitos que imponen en el *cluster*. Ambos, Ceph y OpenNebula, van a estar en su versión más reciente a la fecha de la elaboración del proyecto, usando Ceph en su versión *Quincy* 17.2.5, y OpenNebula en su versión 6.6.0.

4.1.1. NÚMERO MÍNIMO DE HOSTS Y SISTEMA OPERATIVO

Ceph requiere de un *cluster* de 3 *hosts* como mínimo para poder asegurar una replicación de factor 3, y para su versión Quincy, recomienda el uso de los sistemas operativos CentOS 8, Ubuntu 20.04 y Ubuntu 22.04 [31]. Un entorno OpenNebula no impone un número mínimo de *hosts* al necesitar solamente un *Front-end* y un *host* hipervisor, los cuales pueden coincidir en el mismo *host* físico. Los sistemas operativos soportados en la versión 6.6.0 de OpenNebula son Ubuntu 22.04, Ubuntu 20.04, Debian 11, Debian 10, AlmaLinux/RHEL 8,9 y Arch [32]. Concluimos así que el *cluster* va a tener 3 *hosts*, con un sistema operativo Ubuntu al ser el único soportado por ambas herramientas. Se utiliza su versión más reciente, el Ubuntu Server 22.04 LTS, al garantizar mayor plazo de soporte que su versión anterior. El *cluster* es expandible a un mayor número de *hosts* para ampliar sus capacidades, pero lo ideal es mantener el menor número de *hosts* posibles y simplemente aprovisionar más los existentes.

4.1.2. NÚMERO MÍNIMO DE REDES

Como expliqué en el apartado 3.1.1. sobre OpenNebula, su entorno se compone un *cluster Front-end* y otro de *hosts* hipervisores, los cuales se comunican entre sí a través de una red de gestión o *management*. Por lo general, esta subred suele ser accesible desde el exterior — ergo tiene conexión a la red pública — por lo que es en ella donde se suelen exponer las diferentes interfaces de OpenNebula, como su interfaz web *Sunstone* o sus diferentes APIs. Aparte de la red de gestión, es posible definir dentro de OpenNebula diferentes redes virtuales o VNets para la interconectividad de las diferentes VMs entre sí o con el exterior. En la definición de estas VNets se puede detallar tanto la NIC a usar (*Network Interface Card*) como cualquier rango de IPs, por lo que es posible y recomendable emplear una NIC libre del *host* para VNets de aplicaciones que demanden un gran ancho de banda.

Para el *networking* de los *hosts* que componen un *cluster* de Ceph, basta con una sola subred en la que los diferentes *daemons* se comuniquen entre sí. En Ceph son los mismos clientes los que gracias al mapa CRUSH obtenido de un MON realizan peticiones de escritura o lectura directamente a los OSDs que la albergan. Aunque haya replicación de la información, la lectura de cada objeto se realiza siempre en el OSD que contiene su PG principal, por lo que los clientes van haciendo diferentes peticiones a todos los OSDs del *cluster* tanto para escribir como para leer.

No obstante, de la replicación de objetos se encargan los mismos OSDs, que también mediante el mapa CRUSH conocen el resto de OSDs en los que replicar la información. En la figura 10 se puede ver como un objeto no se considera como escrito hasta que su OSD primario no asegura que la última réplica se ha completado.

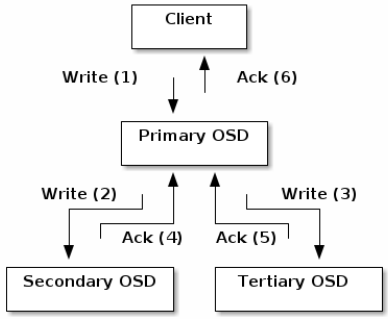


Ilustración 10. Replicación de datos en Ceph [29]

Ceph realiza un uso intensivo de su red para la transferencia de datos entre OSDs; no solo para la replicación de objetos, sino también para redistribuciones de PGs, recuperación de datos, etc. De hecho, el ancho de banda de la comunicación entre OSDs es uno de los principales cuellos de botella del rendimiento de Ceph, por lo que cuanta mayor capacidad tenga y menos tráfico compartido exista mejor.

Para la arquitectura de nuestro CPD, se hace necesaria pues la presencia de una red de alta capacidad (>= 10Gbps) dedicada específicamente a servir a una red de almacenamiento. Para *clusters* grandes es posible también separar el tráfico de los OSDs del resto mediante una subred secundaria exclusiva para ellos [33]. De todas formas, como la arquitectura de este proyecto no es una de gran escala, una sola subred suficientemente rápida dedicada al almacenamiento es suficiente para garantizar un buen rendimiento de escritura y lectura.

4.1.3. ALTA DISPONIBILIDAD

Uno de los objetivos del CPD es el de asegurar la continuidad de sus servicios, sin importar la posible existencia de fallos tanto físicos como de *software*. Para lograr la alta disponibilidad del servicio o *High Availability* (HA), es necesario tener varias réplicas de cada uno de los elementos indispensables que garantizan el funcionamiento del sistema.

Comenzando por Ceph, la HA viene provista por su propia arquitectura RADOS, la cual ya brinda replicación de la información en los OSDs repartidos en los diferentes *hosts* físicos. Ceph también garantiza la existencia de por lo menos tres MONs para que los clientes puedan conocer el mapa CRUSH del *cluster*, y permite la replicación del resto de *daemons* del *cluster*. Para nuestra arquitectura — la cual tiene el mínimo número de *hosts* para un cluster que son tres — no supone ningún inconveniente el poder tener una instancia de cada *daemon* en cada *host*: un OSD, un MON y un MGR.

La alta disponibilidad en OpenNebula es algo más compleja, ya que la continuidad de los servicios de orquestación del *Front-end* y de las VMs funciona de manera completamente diferente. En el caso del *Front-end* con HA, OpenNebula permite implementar el algoritmo de consenso Raft para elección de un *host* líder de entre los disponibles en el *cluster Front-end*, desde el que se hace una gestión centralizada de OpenNebula. El *host* líder, es el encargado de levantar las diferentes interfaces de OpenNebula, y es capaz de levantar una IP flotante o virtual (VIP) que permite la gestión centralizada de OpenNebula siempre desde la misma dirección IP, sin importar cual sea el *host* líder. Entre los requisitos de Raft, el *cluster* de *hosts Front-end* debe tener un número impar de *hosts* (tres es el número recomendado), una base de datos del mismo tipo entre *hosts* (se recomienda MySQL), y la misma configuración en todos los *hosts* [34].

La alta disponibilidad de las VMs en cambio se configura mediante la creación *hooks* programables que se lancen cuando la VM o su *host* se detecten como caídos. Los *hooks* se encargan de realizar el *script* necesario para levantar una nueva VM que sustituya a la caída en otro *host* válido.

4.1.4. MONITORIZACIÓN

La monitorización de los *hosts* y VMs de los *clusters* de Opennebula es tarea del *daemon* monitord ubicado en el *Front-end*. Este configura un agente que ejecuta sondas o *probes* en los *hosts* hipervisores y recibe una serie de métricas periódicamente en una dirección de escucha del *Front-end*. Las métricas recibidas corresponden a información general como puede ser el estado de los *hosts* y VMs o su consumo de CPU y memoria, y esta información puede ser visualizada principalmente desde el *dashboard* del servidor web Sunstone del *Front-end*.

Ceph en diseño es mucho más complejo que OpenNebula y emplea diferentes medios para la monitorización de la salud del *cluster*, de cada componente individual o de la información almacenada. La visualización del estado del *cluster* y sus componentes se puede hacer desde su potente herramienta de CLI, o desde la UI web *dashboard* levantada por el *daemon* MGR activo. Esta UI *dashboard* se expone por defecto en el puerto 8443 de todas las interfaces del *host* con el MGR activo, sucediendo que si accedemos a dicho puerto desde el resto de los *hosts* con MGR en *standby* estos realizan una redirección HTTP (303) a la URL del *dashboard* del *host* con MGR activo.

Por suerte, tanto Ceph como OpenNebula desde su última versión 6.6 tienen sus propios *exporters* para métricas de Prometheus. En el caso de Ceph el *exporter* es un módulo del *daemon* MGR con información acerca de los diversos componentes del *cluster*, mientras que para OpenNebula hay dos *exporters*, para métricas de la nube OpenNebula y de cada VM respectivamente. Ambos sistemas tienen también su propia implementación del "*node exporter*" de Prometheus para métricas relacionadas con el estado de cada *host*, pero basta con levantar uno de ellos. El enfoque basado en llamadas tipo "PULL" de Prometheus, permite tener centralizadas las métricas de ambos sistemas en un único servidor Prometheus (o uno por *host* para tener HA) para la programación conjunta de alarmas relacionadas con el CPD en su conjunto. Tanto Ceph como OpenNebula incluyen también *dashboards* predefinidos para la visualización de métricas en

servidores web Grafana. De hecho, la UI *dashboard* de Ceph requiere obtener sus gráficas de un servidor Grafana.

4.2. Arquitectura de Referencia de Alto Nivel

Para la definición de una arquitectura *cloud* es posible diferenciar entre cuatro planos: orquestación *cloud*, procesamiento, almacenamiento y redes. En la orquestación se encuentran aquellos servicios relativos a la coordinación y automatización de la gestión de la infraestructura del *cluster*, mientras que los planos de procesamiento, almacenamiento y redes comprenden el *hardware* del *cluster* con el que se van a aprovisionar a las VMs. Siguiendo esta diferenciación lógica, la orquestación del CPD (llevada a cabo por el *Front-end* de OpenNebula y la API de gestión de virtualización Libvirt) no tiene por qué coincidir físicamente con los planos de procesamiento, almacenamiento y redes, por lo que es posible separar el *Front-end* de OpenNebula de los *hosts* hipervisores que realizan la virtualización. De esta separación surgió el diseño de la Ilustración 11, el cual fue el primero empleado para mis pruebas.

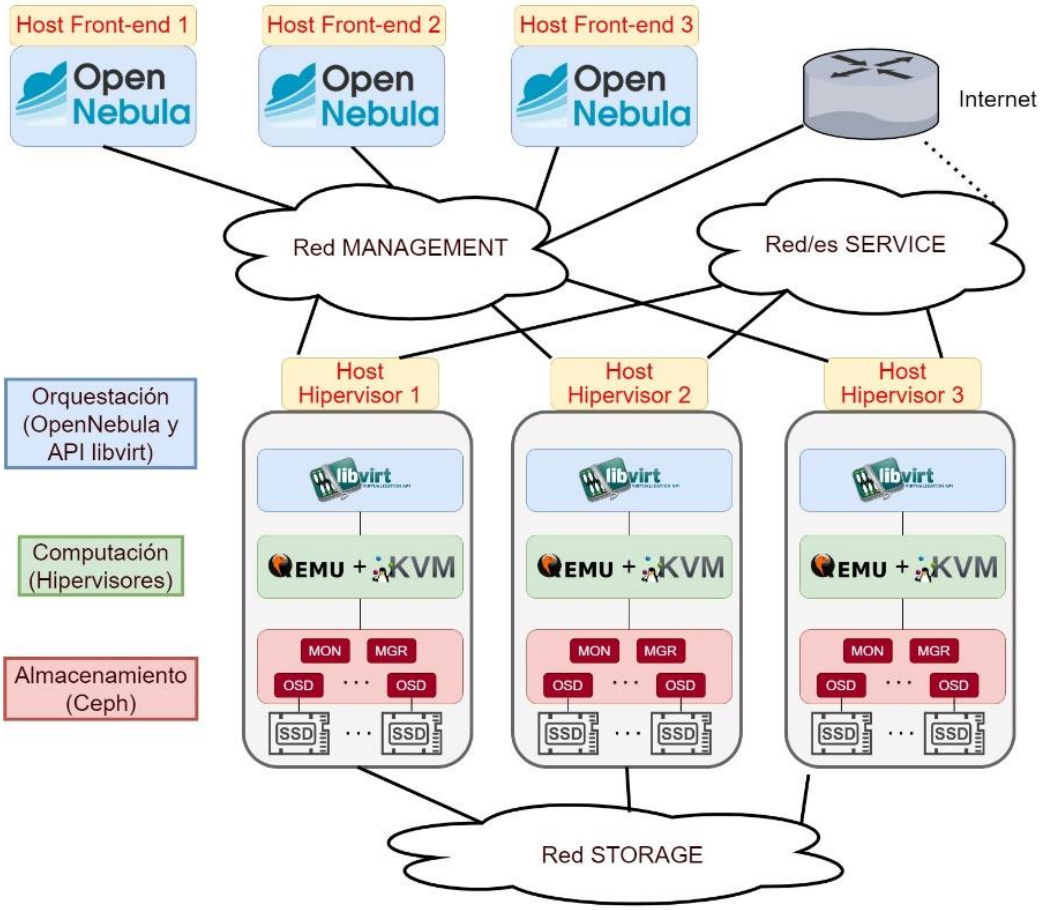


Ilustración 11. Arquitectura con separación entre *Front-end* e hipervisores

En la figura se muestran claramente los planos de orquestación, computación y almacenamiento. El plano que falta en la figura sería el de redes, representado por las tres redes mostradas:

- **Red MANAGEMENT**: Red de acceso a la gestión del *cluster*, usada para transportar el tráfico de orquestación de OpenNebula y exponer APIs. Salvo en picos puntuales, no hay excesiva intensidad de trabajo, por lo que con un ancho de banda de 1Gbps es suficiente.
- **Red STORAGE**: Red dedicada específicamente para la comunicación entre componentes de Ceph. Es uno de los principales cuellos de botella de la red, tanto en lectura como en escritura, por lo que es necesario un ancho de banda de por lo menos 10Gbps.
- **Red/es SERVICE**: Conjunto de redes virtuales (VNets) creadas por OpenNebula desde cualquier NIC de los *hosts* para su uso por parte de las VMs. Dependiendo de la aplicación que contengan las VMs, las VNets pueden necesitar acceso a la red pública, y más o menos ancho de banda. Se pueden levantar VNets sobre los NICs de las redes MANAGEMENT y STORAGE mediante el uso de puentes de red (Linux Bridges).

Los tres *hosts* hipervisores cumplen con el paradigma de la hiperconvergencia al albergar cada uno suficiente poder de computación, memoria, almacenamiento y redes para correr diversas cargas de trabajo sin depender de recursos externos. Solamente no incluirían la "cabeza" que les ordena qué cargas correr que estaría en los *hosts Front-end*.

El número mínimo de *hosts* hipervisores es tres para cumplir con los requisitos de HA de Ceph, pero para el *Front-end* de OpenNebula basta con un solo *host*, por lo que cuatro es el menor número de *hosts* posible para esta arquitectura. La ilustración muestra tres *hosts* de *Front-end* para asegurar la alta disponibilidad de OpenNebula, pero no siempre es necesario tener tantos ya que estos *hosts* no son tan críticos dentro del *cluster*. Las VMs pueden funcionar ininterrumpidamente sin problema durante el tiempo que el *Front-end* de OpenNebula esté caído.

Este tipo de arquitecturas con separación entre *hosts* de control y de virtualización es totalmente funcional, de hecho, con la adición de cierta complejidad es ahora mismo la usada en algunos CPD de producción de Telefónica. Sin embargo, su principal inconveniente es precisamente la necesidad de aumentar el número mínimo de *hosts* a 4 o 6 en HA junto con los costes que esto conlleva. También supone un aumento en el tráfico de la red de MANAGEMENT y depender en mayor medida de los posibles retardos de latencia de una red pensada para tráfico ligero.

Por suerte, es posible ubicar los servicios del *Front-end* de OpenNebula en los propios *hosts* hipervisores permitiendo reducir la cantidad mínima de *hosts* del CPD a 3. Tener los cuatro planos del CPD en cada *host* encaja aún más con concepto de hiperconvergencia que la arquitectura anterior, simplifica el *networking* y permite que varias de las llamadas a los hipervisores de los *hosts* se hagan hacia una dirección "localhost", reduciendo las latencias de orquestación al mínimo posible. En la Ilustración 12 se muestra la proposición de arquitectura de referencia para el CPD.

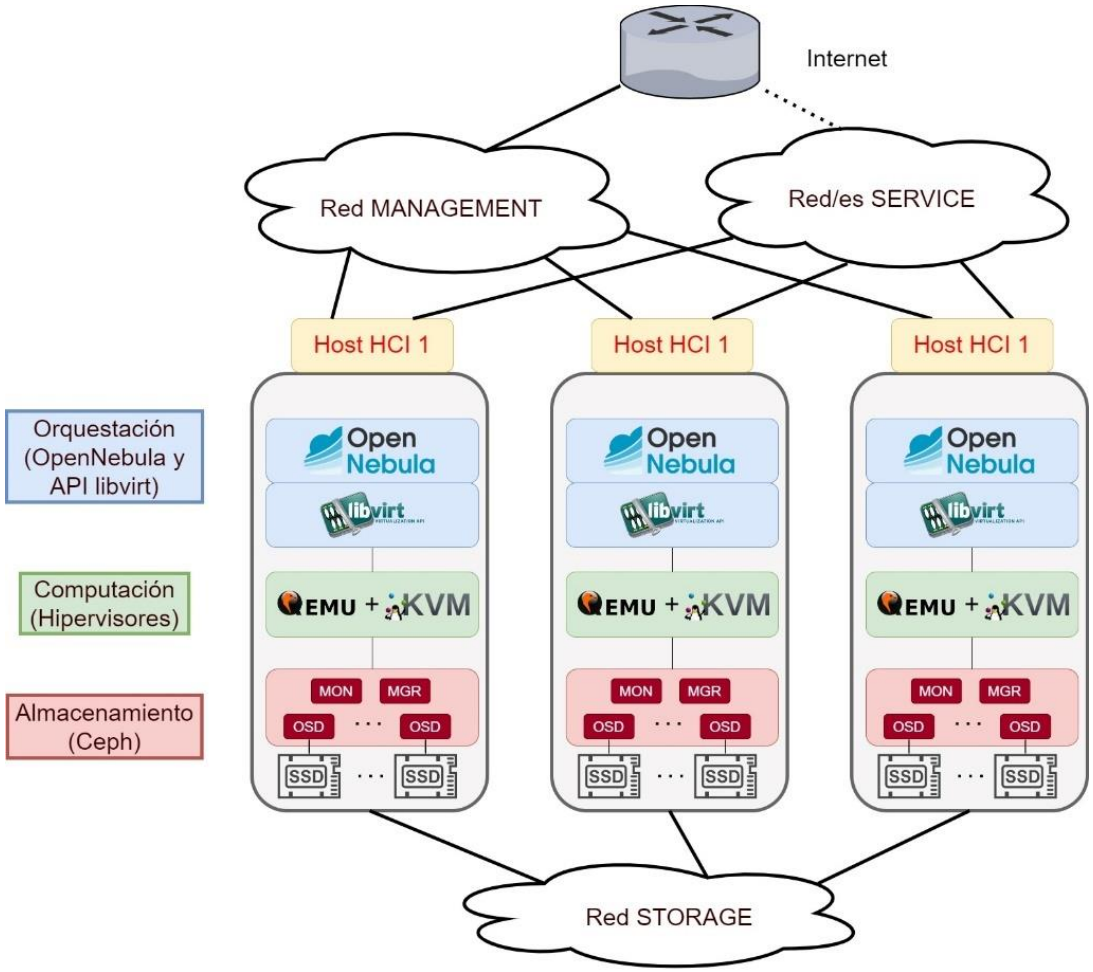


Ilustración 12. Arquitectura con *hosts* alojando *Front-end* e hipervisores al mismo tiempo

Los tres *hosts* contienen los mismos servicios tanto de Ceph como de OpenNebula creando un único *cluster* autónomo útil para todo tipo de aplicaciones. El acceso al *cluster* para su gestión se realiza en la red MANAGEMENT desde la IP flotante levantada por el *host* líder del *Front-end* de OpenNebula. Es en esta IP fija en la que se exponen también todas las interfaces de los servicios internos del *cluster*: la UI Sunstone, las diferentes APIs, las herramientas de visualización, etc.

Queda por último explicar dónde se ubican los servicios de Prometheus y Grafana para la monitorización del *cluster*. Ambas Ceph y OpenNebula tienen cada una su propia versión sobre cómo y dónde deberían desplegarse estos servicios. OpenNebula ofrece sus propios paquetes para la configuración del servidor Prometheus, el alertmanager y los *exporters* necesarios, los cuales están pensados para correr en todos los *hosts* del *Front-end*.

En el caso de Ceph el *host* con el MGR activo puede levantar su propio *ceph-exporter* con las métricas de Ceph como módulo del mismo modo que sucedía con la UI *dashboard* de Ceph. No obstante, para el despliegue del resto de herramientas de monitorización hay dos escenarios posibles [35]:

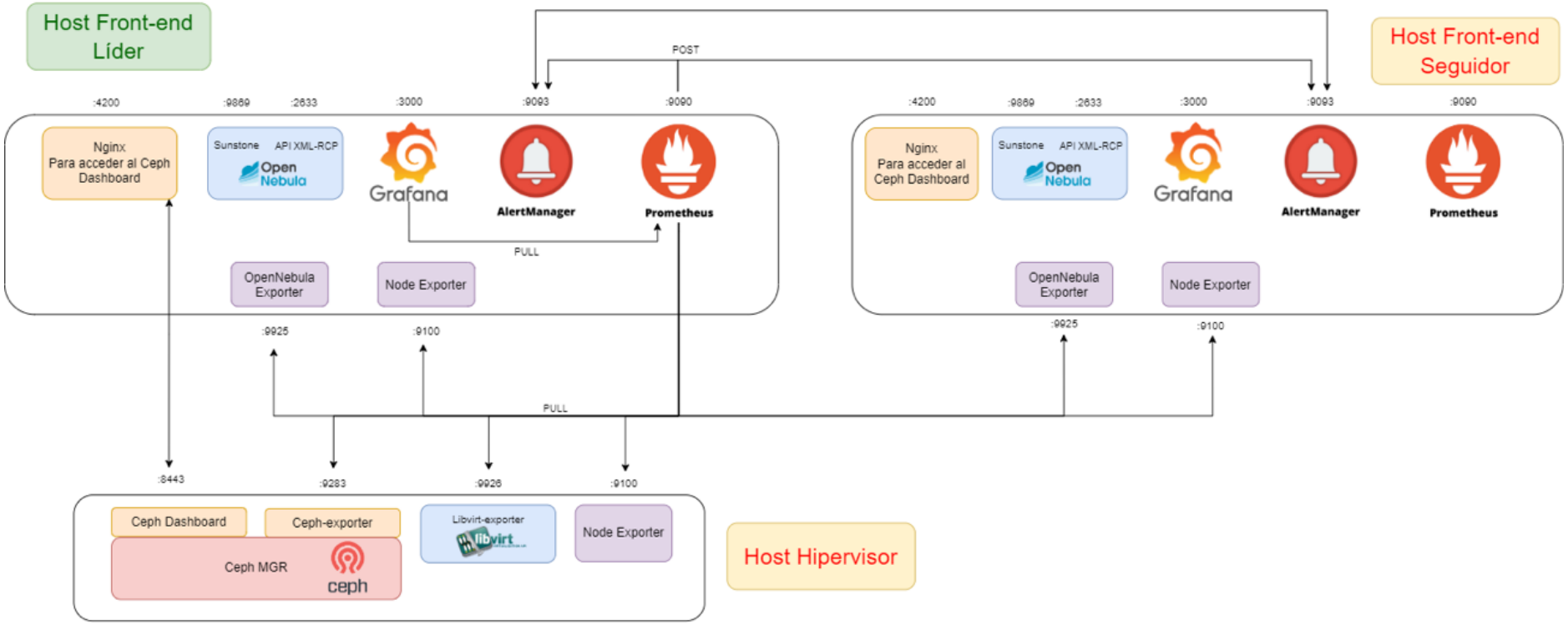


Ilustración 13. Monitorización en arquitectura con separación entre Front-end e hipervisores

- Existencia de un *stack* de monitorización totalmente automatizado y gestionado por el orquestador “cephadm”. Este se encarga de configurar e integrar automáticamente los servicios de Prometheus, alertmanager, Grafana y node-exporter con Ceph desde contenedores. Estos servicios (principalmente Grafana) se despliegan en versiones simplificadas ya que su principal objetivo es servir a la UI dashboard de Ceph.
- Tener estos servicios desplegados aparte de forma manual y configurar el servidor Prometheus para que obtenga métricas del *endpoint* del *ceph-exporter*, y el Grafana para que la UI dashboard pueda leer *dashboards* de ella.

En la implementación actual del CPD se va a hacer uso de este segundo escenario, usando de Ceph solamente su *ceph-exporter* y su UI *dashboard*, y de Opennebula los Prometheus, alertmanager y node-exporter que vienen en sus paquetes. A mayores hay que instalar el servidor Grafana y un contenedor con un servidor nginx para redirigir el tráfico de la IP virtual al *dashboard* de Ceph. En la Ilustración 13 se muestra de manera simplificada el modelo de monitorización en arquitecturas hiperconvergentes con separación entre hipervisores y *Front-end*.

La Ilustración 13 del *host Front-end* líder los servicios expuestos a los que accederá el administrador desde la IP flotante a través de diferentes puertos. Cada *host Front-end* va a contar con los mismos servicios, pero solamente se utilizan los del *host* líder para que en caso de que cambie el líder, poder utilizar siempre la misma URL para los servicios. Estos servicios de derecha a izquierda son:

- Prometheus: Todos los servidores Prometheus (no solo el del *host* líder) obtienen métricas de los diferentes *exporters* de todos los *hosts* del *cluster* mediante peticiones tipo PULL, y las exponen en el puerto 9090 de todas sus direcciones de la red MANAGEMENT. Todos los servidores Prometheus por tanto deberían funcionar de forma paralela exponiendo en todo momento las mismas métricas.
- Alertmanager: En caso de que las métricas recolectadas en Prometheus cumplan alguna condición que active una alarma, se notifica la alarma a todos los servidores Prometheus del *cluster*, no solo el del mismo *host*. El alertmanager, se encarga de evaluar la alarma y enviarla hacia diferentes destinatarios o *receivers* de acuerdo con las condiciones que se establezcan. Todos los servidores alertmanager cuentan con su propio servicio de HA para sincronizar su comportamiento y no enviar notificaciones duplicadas a los diferentes *receivers*. Desde la interfaz del alertmanager en el puerto 9093 se pueden configurar de forma dinámica silencios para acallar alarmas durante un periodo de tiempo.
- Grafana: Cada servidor Grafana obtiene métricas del servidor Prometheus de su mismo *host* y las visualiza en diversos *dashboards* en el puerto 3000.
- OpenNebula: Ilustro solamente Sunstone, la UI web del puerto 9869 la API XML-RCP para gestión programática en el puerto 2633 al ser los servicios más representativos, pero todas las interfaces de OpenNebula descritas en el punto 3.1.1 se expondrían también en la red MANAGEMENT. Aunque la gestión se haga desde un solo *host*, el estado del *cluster* se sincroniza automáticamente en la base de datos de todos los servidores OpenNebula.

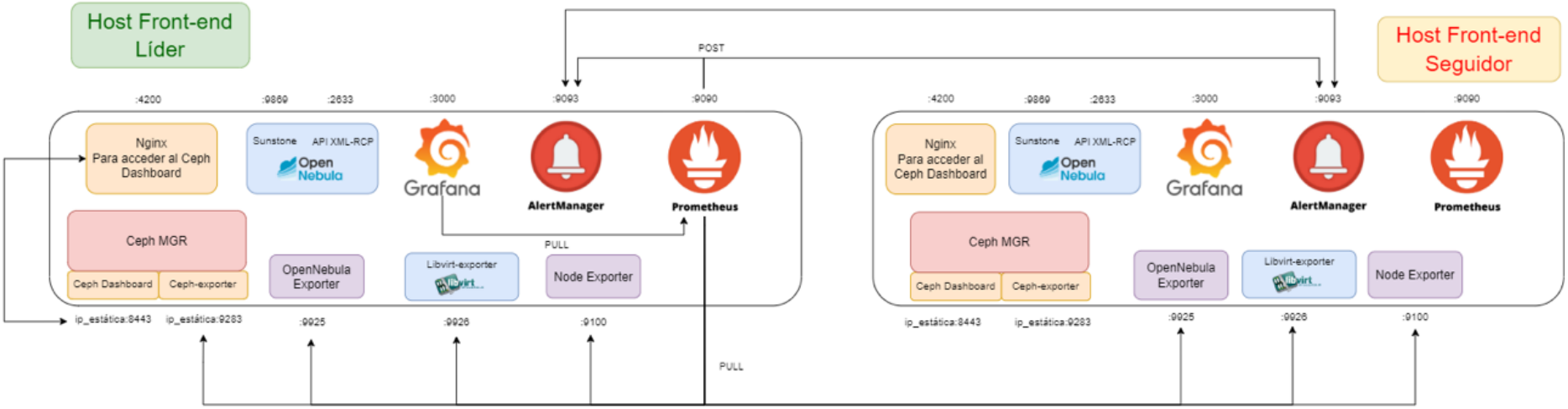


Ilustración 14 Monitorización en arquitectura con *hosts* alojando Front-end e hipervisores al mismo tiempo

- Nginx para el *Dashboard* Ceph: Al ser gestionada por el MGR activo de uno de los *hosts* hipervisores, la interfaz *Dashboard* de Ceph no puede ser expuesta en la IP flotante directamente. Es por esto necesario levantar un servidor nginx ligero (y contenerizado en nuestro caso) que permita redireccionar el tráfico entrante en un puerto arbitrario, el 4200 en nuestro caso, hacia el Dashboard de Ceph en el puerto 8443 de un *host* hipervisor. El *host* al que apunta este nginx no es relevante ya que en caso de apuntar a un MGR en *standby*, este redirige automáticamente hacia el MGR activo.

En el caso de la arquitectura con *hosts* alojando *Front-end* e hipervisores al mismo tiempo, el diagrama es más simple tal y como se ve en la Ilustración 14.

El libvirt-exporter y los servicios de Ceph pasan a estar en el mismo *host* que los servicios de monitorización. El servidor nginx sigue siendo necesario, ya que los MGR de Ceph no son capaces de exponer en la IP flotante debido a su dinamismo, pero por lo menos el redireccionamiento se produce a una dirección del mismo *host* hacia la denominada "ip_estática" de la figura que sería la IP normal del *host* en la red MANAGEMENT.

Como se mencionó anteriormente, el *Dashboard* de Ceph obtiene sus gráficas a partir del servidor Grafana que se le indique, que en el caso anterior era el expuesto en la IP flotante. No obstante, al tener Ceph y monitorización en conjunción en el mismo *host*, esta obtención de *dashboards* se produce una vez más mediante llamadas locales.

4.3. Requisitos *Hardware*

Aunque es difícil cuantificar los recursos mínimos que requiere el CPD para funcionar correctamente, hay que tener en cuenta las recomendaciones de *hardware* documentadas para los diferentes componentes *software* que lo componen.

OpenNebula por su parte no establece de forma oficial parámetros mínimos para su despliegue. Para tener al menos una referencia, se pueden utilizar los requerimientos mínimos para el despliegue de un OpenNebula de un solo *host* (*Front-end* e hipervisor) del *script* de despliegue miniONE. MiniONE pide solamente un *host* (físico o virtual) con 4GiB de memoria RAM y 20GiB libres de almacenamiento [36]. Aparte de estos, están los requisitos del *driver* de virtualización KVM, los cuales son solamente que la CPU de los *hosts* disponga de las funcionalidades IntelVT o AMD-V para habilitar la virtualización [37].

En Ceph en cambio, al haber una serie de implementaciones comerciales detrás de compañías como SUSE, RedHat o IBM, sí es posible encontrar fácilmente documentación diversa acerca del *hardware* mínimo recomendado. Por lo general, la mayoría de los sitios consultados asumen una separación entre *hosts* de almacenamiento con los *daemons* OSD y *hosts* de gestión con los MON, por lo que hay que tener en cuenta que las necesidades reales de cada *host* no tienen por qué corresponderse con la suma de los requisitos de los *daemons* que contenga. Haciendo una

síntesis de las fuentes encontradas [38] [39] [40] [41], algunos de los requisitos de Ceph para los *hosts* son:

- Emplear una red ethernet cableada tolerante a fallos de como mínimo 10Gbps para uso exclusivo de Ceph. Idealmente sería recomendable emplear dos tarjetas de red de 25Gbps (o más) asociadas mediante agregación de enlaces 802.3ad (LACP – Link Aggregation Control Protocol) para ofrecer un rendimiento superior y tolerancia a fallos, pero para ello hay que asegurarse de tener también *switches* que permitan este tipo de enlaces.
- La capacidad de las CPUs no es importante para Ceph. Los *daemons* no realizan un gran esfuerzo de computación, por lo que con tener suficientes *cores* donde contenerizar los diferentes *daemons* es suficiente.
- Tener cuanta más memoria RAM posible. Los *daemons* OSD por ejemplo utilizan memoria para almacenar datos en caché en lugar de depender de la caché de los dispositivos que gestionan [38].
- Garantizar que por cada OSD haya:
    - Un core AMD64 o Intel64, dos si el dispositivo de almacenamiento manejado es SSD y cuatro si es NVMe [40].
    - 4GB de memoria RAM de base, y 1GB extra por cada TB de almacenamiento gestionado. Hay que tener en cuenta que los OSDs pueden tener picos de uso en tareas intensivas como la recuperación de datos, por lo que habría que dejar un margen de memoria para el resto de los servicios del *host*. Para OSDs contenerizados, asignar como mínimo 5GB de RAM, aunque es más recomendable dejarles 8GB en caso de picos de uso [39].
    - Un dispositivo de almacenamiento que gestionar que no sea el mismo que el que utiliza el Sistema Operativo del *host*, no vale con reservarle una partición. Se recomienda almacenamiento SSD por ser más rápido que su contraparte HDD cada vez más anticuada en producción.
- Garantizar del mismo modo que por cada MON haya:
    - Uno o dos cores AMD64 o Intel64.
    - 1GB de RAM. 3GB en caso de MON contenerizado.
    - 10GB de almacenamiento SSD, aunque es más aconsejable asegurar 50GB a cada MON [41].
- Requisitos por cada MGR:
    - Un core AMD64 o Intel64
    - 1GB RAM. 3GB en caso de MGR contenerizado.
    - 5GB de almacenamiento [41].
- Por último, y como consideración general, los *hosts* deberían quedar lo más similares posibles en todos sus recursos, de modo que no funcionen mejor unos que otros. Grandes diferencias de *hardware* entre *hosts* perjudican el rendimiento general del *cluster* igualando el rendimiento general al del peor *host*.

Vista esta información y por resumir, en la Tabla 2 se muestran los requisitos que se consideran mínimos a partir de la información anterior para el montaje del CPD Hiperconvergente.

Elemento	Requisitos mínimos
NICs	1 de 1Gbps para la red Management y otro de 10Gbps para la red Storage. Recomendable aumentar el ancho de banda de ambas redes, añadir duplicidad para asegurar alta disponibilidad en la conexión, y añadir más NICs para las VNETs de las VM.
Dispositivos de almacenamiento	Por lo menos dos en cada *host* hipervisor. Uno para el Sistema Operativo, librerías, base de datos de OpenNebula, almacenamiento de los MONs, etc, y otro para el OSD del *host*. Para el primer dispositivo basta con medio TB, mientras que para el segundo mínimo sugiero mínimo 1TB para albergar las imágenes de las VM. Ambos dispositivos deberían ser SSD para asegurar un buen rendimiento Input-Output.
Procesador	Un procesador AMD64 con AMD-V o Intel64 con IntelVT con gran cantidad de *cores*, más de 6 para poder satisfacer los requerimientos de los *daemons* de Ceph y OpenNebula. La arquitectura *software* del CPD no requiere gran poder de computación, pero son las VM a levantar y sus aplicaciones las que más partido van a sacar a las CPU de los *hosts*. Dependiendo de las cargas de trabajo para las que se va a usar el CPD, se requieren procesadores más o menos potentes con un mayor o menor número de *cores*.
Memoria RAM	Se recomienda no escatimar en memoria la RAM de los *hosts* hipervisores, para poder provisionar sin problema numerosas VMs sin perjudicar a las herramientas del *cluster*. Se partiría como mínimo de 16GB, para cumplir apenas con los requisitos de los OSDs, MONs, MGRs y OpenNebula, pero en un posible pico de trabajo es probable que no sea suficiente. Al igual que con el procesador, se recomienda suministrar la memoria que haga falta para provisionar las VMs de las aplicaciones a partir de un mínimo de 32GB. También puede ser aconsejable utilizar parte del almacenamiento sobrante del SSD principal anteriormente dicho como memoria SWAP.

Tabla 2. Requisitos *hardware* mínimos para cada host del CPD

Para lograr un rendimiento estable en el Ceph, se aconseja que tanto los dispositivos de almacenamiento como los NICs de los *hosts* sean lo más semejantes posible entre todos los *hosts*. En caso de no poder garantizar esto, hay que procurar compensar por otros medios las diferencias significativas en velocidades de escritura/lectura, en el almacenamiento de los OSDs o en el ancho de banda de la red de almacenamiento. Ceph ofrece para ello sistemas como los pesos de los OSDs dentro del algoritmo CRUSH, para que el *pool* exija menos a estos *hosts* desiguales. Tanto Ceph como OpenNebula hacen un uso *best-effort* del *hardware* del que disponen y operan independientemente el uno del otro, por lo que un pico de trabajo de Ceph puede conllevar el deterioro del rendimiento de las VMs de OpenNebula y viceversa.

Para evitar un uso desproporcionado de recursos de los *daemons* (principalmente los OSDs), Ceph permite desplegar sus *daemons* en contenedores, así como limitar dinámicamente la memoria empleada por los OSD mediante el atributo 'osd_memory_target'. En OpenNébula es más fácil hacer un seguimiento de los recursos que consume cada VM ya que se le asignan un número fijo de cores, de memoria y de almacenamiento que forman el *hardware* virtualizado de esta. De todas formas, cuando OpenNebula adjudica *cores* y memoria a las VMs, lo hace sin tener en cuenta el uso de ellos que pueda estar haciendo Ceph en el momento, por lo que a la hora de levantar

VMs hay que mantener unos márgenes de memoria y *cores* más allá de los que se muestran disponibles en las interfaces de OpenNebula.

5. IMPLEMENTACIÓN

5.1. Introducción

Este capítulo explica el proceso seguido para poder montar toda la infraestructura *software* del CPD en los *hosts* partiendo con solamente el Sistema Operativo (Ubuntu-Server 22.04 LTS) recién instalado. En los objetivos del proyecto, se encontraba la capacidad de desplegar toda la infraestructura de manera automatizada sin necesidad de interacción por parte del usuario. Para lograrlo, se ha hecho uso de una serie de herramientas que ejecuten los pasos necesarios, en lugar de tener el administrador que realizarlos manualmente.

5.2. Herramientas usadas en el despliegue

5.2.1. ANSIBLE

Ansible es un *software open source* para la automatización de la configuración de sistemas informáticos, despliegue de *software* y orquestación de tareas avanzadas [42]. Permite la elaboración y ejecución de *scripts* llamados *playbooks* en formato YAML (Yet Another Markup Language) con los que describir tareas y configuraciones de forma legible y comprensible. La configuración se hace a partir de un nodo de control con Ansible instalado, el cual accede mediante SSH a los nodos controlados para ejecutar pequeños programas llamados módulos. Una vez finalizados los módulos, Ansible los elimina y finaliza la conexión SSH. El único requisito para esta interacción es que el nodo de control que ejecuta los playbooks tenga acceso SSH a los nodos controlados. Los roles en ansible equivalen a conjuntos de tareas definidas en un fichero reutilizable en múltiples *playbooks*.

Ansible se ha usado para la práctica totalidad de la configuración automatizada de los *hosts*, pudiéndose mediante un único *playbook* de Ansible realizar el despliegue de todo el CPD, además de llamar al resto de herramientas usadas como Terraform.

5.2.2. TERRAFORM

Terraform es una herramienta de código abierto para crear, modificar y versionar infraestructura como Código (IaC) de forma segura y eficaz. Emplea un lenguaje declarativo llamado HashiCorp Configuration Language (HCL) para la descripción de los recursos y componentes necesarios para configurar la infraestructura. Los *plugins* o *providers* de Terraform le permiten interactuar con plataformas en la nube y otros servicios a través de sus APIs [43].

El principal uso de Terraform en el despliegue del CPD es del de instanciar los diferentes recursos dentro de OpenNebula, ya que mediante su API XML-RPC, OpenNebula puede ser usada como *provider* de Terraform.

5.2.3. CEPHADM

Existen gran variedad de opciones para la instalación de un *cluster* Ceph, tales como Ansible, Salt, Juju, Puppet o la instalación manual. No obstante, el método más recomendado es mediante un orquestador oficial de Ceph, entre los que se encuentran Rook para entornos de Kubernetes y Cephadm para infraestructuras locales (*on-premise*) [44].

Cephadm es una utilidad para la gestión del ciclo de vida completo de un *cluster* Ceph, el cual abarca desde su "arranque" (o *bootstrap*) en un único *host* con un *daemon* MON, hasta su ampliación o actualización. Una vez desplegado el *cluster* Ceph, Cephadm se integra como módulo de orquestación dentro del MGR activo de Ceph, y se encarga de levantar y gestionar con total libertad todos los *daemons* que componen un *cluster* Ceph dentro de contenedores.

Pese a que se podría emplear Ansible para el despliegue del *cluster* Ceph unificando la configuración de los *hosts* en una misma herramienta común, se ha considerado mejor que los *playbooks* usados llamen y empleen Cephadm para sus operaciones. No solo es que los *playbooks* oficiales de Ceph-Ansible carezcan de algunas funcionalidades, principalmente de gestión post-despliegue, sino que Cephadm de partida brinda una gran flexibilidad para la configuración de los servicios, pudiendo definir cada uno de los servicios y *daemons* mediante plantillas en formato YAML.

Para utilizar Cephadm en los *hosts*, se requiere únicamente de la existencia de una herramienta de gestión de contenedores como Podman o Docker, sincronización temporal entre *hosts* (mediante paquetes como chrony o NTP – Network Time Protocol), Python 3, systemd y LVM2 [45]. Las dos primeras se configuran en los *hosts* mediante Ansible, mientras que las tres últimas vienen ya preinstaladas en Ubuntu 22.04.

5.2.4. HERRAMIENTAS DESCARTADAS: MINIONE Y OPENNEBULA PROVISIONS

Para desplegar una nube OpenNebula con un sólo *host Front-end* (con posibilidad de configurarlo también como *host* hipervisor en el proceso), existe la herramienta miniONE [36]. MiniONE consiste en un *script* rápido y sencillo que instala y configura en un solo *host* las librerías necesarias para una nube OpenNebula, pudiendo esta nube ser provisionada y ampliada posteriormente, tanto desde la GUI Sunstone como desde CLI o su API con Terraform.

MiniONE fue ampliamente usada al comienzo de la investigación de este proyecto, sin embargo, la imposibilidad de configurar desde el comienzo aspectos como el *Front-end* con HA, o la base de datos de OpenNebula en MySQL, hicieron que finalmente optara por realizar la configuración de OpenNebula mediante módulos de Ansible.

Otra funcionalidad de automatización de OpenNebula desechada fue la de “OpenNebula Provisions” para la ampliación dinámica de la infraestructura *cloud*. OpenNebula contempla para servidores Equinix, AWS u *on-premise* la posibilidad de utilizarlos para el despliegue de *clusters* de *hosts* hipervisores preconfigurados por completo tanto desde CLI con “oneprovision” y “oneprovider” como desde la GUI FireEdge.

Este tipo de herramientas no son necesarias en configuraciones hiperconvergentes como la perseguida, pero podrían servir en aquellas como la referida en la Ilustración 11 para la incorporación en la nube OpenNebula de un *cluster* de *hosts* con Ceph e hipervisor (escenario igualmente denominado HCI por ONE al ubicar almacenamiento y virtualización en los mismos *hosts*). Descarto igualmente esta posibilidad debido a la limitada flexibilidad de las “provisiones” disponibles. Lo que “oneprovision” realmente hace es ejecutar por debajo una serie de playbooks de Ansible (entre los que se encuentran Ceph-Ansible) en los *hosts* objetivo, los cuales no solo no incluyen las últimas librerías disponibles, sino que tienen algunas limitaciones. Por ejemplo, no se permite escoger la versión de Ceph a instalar (siempre Pacific, la penúltima disponible) y el escenario de configuración de OSDs es siempre *collocated* (escenario de Ceph-Ansible que utiliza la herramienta de gestión de discos Ceph-Disk, obsoleta desde 2017 [46] [47]) .

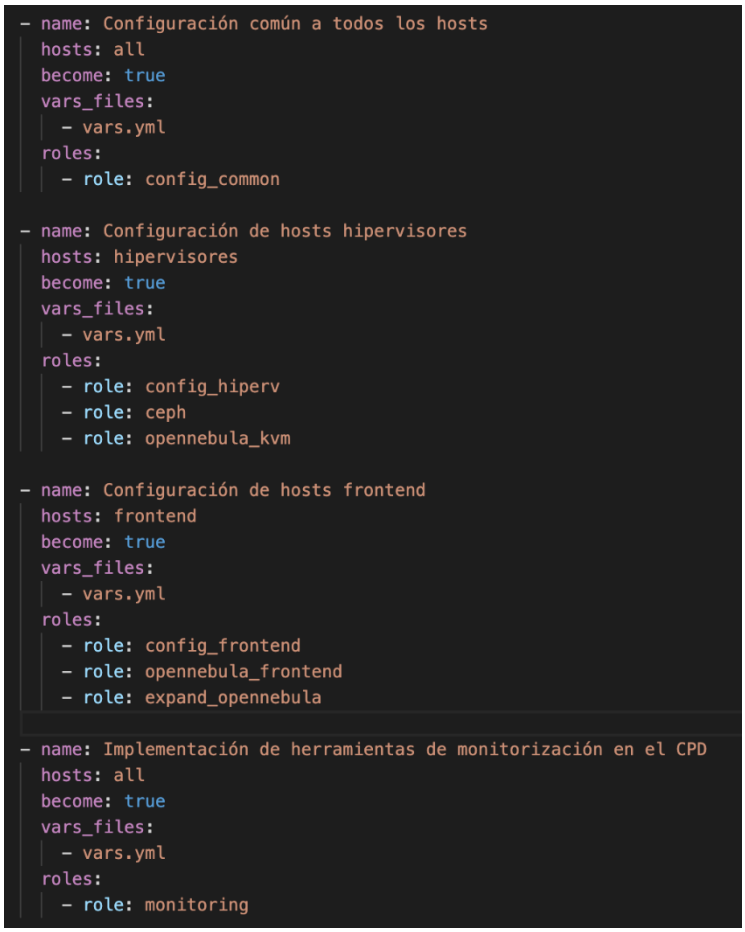
5.3. Pasos seguidos

El despliegue del CPD se puede realizar tanto manualmente desde la CLI, o mediante el *playbook* de Ansible cuya elaboración ha supuesto gran parte del esfuerzo de este proyecto. Los pasos descritos y el *playbook* de Ansible sirven para el despliegue de CPDs hiperconvergentes tanto siguiendo la Arquitectura con separación entre *Front-end* e hipervisores, como la Arquitectura con *hosts* alojando *Front-end* e hipervisores al mismo tiempo.

Para la explicación del procedimiento seguido, como resulta demasiado aparatoso comentar cada instrucción ejecutada o fichero de Ansible diseñado, se hará una descripción de alto nivel de las tareas que una instalación manual debe hacer, usando el *playbook* de ansible como hilo conductor. A grandes rasgos, los pasos seguidos para el despliegue se pueden resumir en los roles de Ansible utilizados en la Ilustración 15.

El primer grupo de roles corresponde a la ejecución de una serie de pasos para la preconfiguración de todos los *hosts* del CPD. El rol “config_common” tiene entre sus tareas:

- Configuración de zona horaria y servidor NTP común a los *hosts*
- Inclusión de *hosts* y su dirección en el fichero `/etc/hosts` de cada *host*.
- Configuración de red. En Ubuntu 22.04 se hace mediante plantillas YAML en la herramienta Netplan.
- Añadir repositorio de OpenNebula.



```yaml
- name: Configuración común a todos los hosts
  hosts: all
  become: true
  vars_files:
    - vars.yml
  roles:
    - role: config_common

- name: Configuración de hosts hipervisores
  hosts: hipervisores
  become: true
  vars_files:
    - vars.yml
  roles:
    - role: config_hiperv
    - role: ceph
    - role: opennebula_kvm

- name: Configuración de hosts frontend
  hosts: frontend
  become: true
  vars_files:
    - vars.yml
  roles:
    - role: config_frontend
    - role: opennebula_frontend
    - role: expand_opennebula

- name: Implementación de herramientas de monitorización en el CPD
  hosts: all
  become: true
  vars_files:
    - vars.yml
  roles:
    - role: monitoring
```

Ilustración 15. Playbook para el despliegue de CPD HCI

El segundo grupo de roles incluye la configuración correspondiente a los *hosts* hiperconvergentes donde se ubica el almacenamiento Ceph y los recursos de computación. Los roles ejecutados junto con sus tareas son:

- config_hiperv: preconfiguración necesaria en los *hosts* hipervisores
    - Depuración de sistemas de ficheros, particiones o cabeceras LVM existentes en los dispositivos de almacenamiento a usar en Ceph
    - Instalación de Docker. Herramienta de contenerización usada por Cephadm para los *daemons*.
- ceph: despliegue y configuración de Ceph
    - Instalación de paquetes “cephadm” y “ceph-common”.
    - Arranque o *bootstrap* del *cluster* Ceph mediante Cephadm desde uno de los *hosts*.
    - Añadir el resto de los *hosts* al *cluster* Ceph.
    - Aplicación de plantilla YAML que indique al orquestador Cephadm la ubicación de los MONs, MGRs y OSDs contenerizados a lo ancho del *cluster*.
    - Creación de *pool* RBD para OpenNebula, y cliente Ceph con permisos de escritura y lectura sobre dicho *pool*.

- opennebula_kvm: resto de configuración de hipervisores una vez desplegado Ceph.
    - Instalación de paquete OpenNebula para hipervisores, que incluye libvirt y el hipervisor KVM.
    - Configuración de libvirt para que tenga acceso al *pool* creado en Ceph con las credenciales de autenticación del cliente Ceph creado.

Los roles dedicados a los *hosts Front-end* son una lista independiente a la lista de hipervisores, por lo que los siguientes pasos pueden realizarse en los mismos *hosts* que antes o en otros aparte.

- config_frontend: preconfiguración necesaria en los *hosts Front-end*
    - Instalación de base de datos local mySQL para almacenar el estado de la nube de OpenNebula.
    - Configuración del MySQL para crear usuario OpenNebula.
- opennebula_frontend: pasos para el arranque de la nube OpenNebula.
    - Instalación de paquetes de OpenNebula
    - Edición de ficheros de configuración de OpenNebula para que use la base de datos MySQL o se modifiquen las credenciales por defecto.
    - Arranque de servicios OpenNebula
    - Reparto de clave pública ssh de OpenNebula en los *hosts* hipervisores para que puedan ser manipulados.
- expand_opennebula: aplicación de ficheros Terraform para la creación de los recursos virtuales básicos para el uso de una nube en OpenNebula. Los recursos que instanciar son principalmente:
    - *Cluster*: Un *cluster* en Opennebula se refiere a un espacio lógico donde albergar recursos físicos por cercanía.
    - *Hosts*: Se incluyen los *hosts* hipervisores dentro del *cluster* creado anteriormente.
    - *Datastores*: Se crean un *datastore* de tipo “system” y otro tipo “image” para el acceso al almacenamiento de Ceph. Estos *datastores* quedan incluidos de nuevo dentro del *cluster*.
    - VNets: Definición de rangos de IPs y las interfaces de red donde se van a ubicar. Las VMs hacen referencia a estas VNets para su configuración de red.

Finalmente, se definen diversas tareas para la implementación del stack de monitorización en el rol “monitoring”. Estas tareas se corresponden al levantamiento y configuración de los *exporters*, el Prometheus, el alertmanager, el Grafana y el *dashboard* de Ceph.

La implementación descrita del CPD es flexible y portable a gran cantidad de escenarios. Lo único necesario es que las interfaces de red que se vayan a usar dentro de OpenNebula se encuentren en puentes Linux nombrados de la misma forma en todos los *hosts*, pero este ajuste en Ubuntu 22.04 se puede hacer mediante la herramienta netplan, ya utilizada en el rol “config_common”. El resto del código se puede encontrar en mi cuenta de github personal: https://github.com/alvarocurt/Hyper-converged-CPD

5.4. Benchmarks

Tras levantar y configurar correctamente el CPD, es momento de medir el rendimiento logrado con él mediante una serie de *benchmarks* de diversa índole. Las pruebas de rendimiento se han realizado sobre la arquitectura descrita en la Ilustración 12, utilizando el equipamiento descrito en la sección 1.4 para la segunda mitad del proyecto.

El CPD consiste en 3 *hosts* hiperconvergentes, "boe01" y "boe02" idénticos, y "boe03" con peores prestaciones. Los *hosts* tienen solamente dos NICs: uno de 1Gbps conectado a internet y utilizado para la red MANAGEMENT, y otro de 40Gbps en los dos primeros *hosts* y de 10Gbps en el tercero, usado para la red de STORAGE. Ambos NICs tienen sus interfaces de red dentro de un *Linux bridge*, y pueden ser usados para la creación de VNETs dentro de OpenNebula (redes SERVICE de la Ilustración 12). Los dos primeros *hosts* utilizan un disco SSD de 447GB para el Sistema Operativo y los servicios, y otro de 1.8TB para el almacenamiento compartido de Ceph en un OSD. El tercer *host* utiliza un NVMe de 476.9GB para el SO y servicios, y utiliza dos SSDs de 931.5GB para el espacio de almacenamiento Ceph combinados en un solo OSD. Se espera por tanto una buena velocidad de escritura y lectura. La comparativa de estas características viene indicada en la Tabla 3.

Elemento	Requisitos mínimos	boe01 y boe02	Boe03
NICs	1 de 1Gbps y otro de 10Gbps	1 de 1Gbps y otro de 40Gbps	1 de 1Gbps y otro de 10Gbps
Dispositivos de almacenamiento	HDD de 300 GB para OS, y 1 HDD de 1 TB para un OSD.	SSD de 447 GB para OS, y SSD de 1.8 TB para un OSD	NVMe de 476.9 GB para OS y dos SSDs de 931.5 GB para un OSD
Procesador	6 CPUs AMD64 con AMD-V o Intel64 con IntelVT	56 CPUs Intel Xeon de 2.4 GHz	32 CPUs Intel Xeon Gold de 2.1 GHz
Memoria RAM	16GB	125.5 GB	62.5 GB

Tabla 3. Comparación de *hardware* entre *hosts* usados

Como se puede ver, el equipamiento usado incumple la recomendación de simetría en los *hosts* del CPD comentada en la sección 4.3, pero para la verificación del potencial del diseño, los recursos de los que disponen son más que suficientes. Con tal cantidad de memoria RAM y de núcleos de procesador en cada *host*, se puede provisionar un gran número de VMs y de servicios; pero si hay cuellos de botella en el CPD, lo más probable a priori es que se encuentren en la capacidad de red y la velocidad de lectura/escritura en el espacio de almacenamiento de Ceph. El primer paso para la medición del rendimiento del CPD, es ver la potencia base del *hardware* antes de la instalación de los servicios de OpenNebula y Ceph.

RED

Haciendo uso de la herramienta "iperf3", se han obtenido los siguientes anchos de banda disponibles en la red:

Hosts involucrados	NICs MANAGEMENT	NICs STORAGE
boe01 <-> boe02	941 Mbps	30.3 Gbps
boe01 <-> boe03	936 Mbps	9.42 Gbps
boe02 <-> boe03	938 Mbps	9.41 Gbps

Tabla 4. Ancho de banda de las conexiones entre *hosts*

El ancho de banda de la red MANAGEMENT muestra un rendimiento de algo menos de 1Gbps tal y como se esperaba, mientras que la velocidad red STORAGE muestra grandes diferencias entre los *hosts* "boe01" y "boe2", y el tercer *host*, con un resultado considerablemente inferior. Los 40Gbps de los NICs de "boe01" y "boe02" tampoco se consiguen aprovechar en todo su potencial siendo en este caso el limitante el *switch* que interconecta los *hosts*, pero esta reducción no afecta demasiado comparativamente, y hasta ayuda a "igualar" las interconexiones.

La siguiente prueba se ha realizado con la herramienta "netperf", con el objetivo de cuantificar la latencia mínima, la latencia media, el percentil 99 y la latencia máxima entre *hosts*. Tras varias repeticiones de la prueba, los resultados medios para cada valor han sido:

Hosts y red involucrados	Latencia mínima	Latencia media	Percentil 99 de latencia	Latencia máxima
boe01 <-> boe02 red MANAGEMENT	38 ms	39 ms	42 ms	296 ms
boe01 <-> boe03 red MANAGEMENT	65 ms	128 ms	138 ms	3066 ms
boe02 <-> boe03 red MANAGEMENT	56 ms	129 ms	138 ms	3493 ms
boe01 <-> boe02 red STORAGE	15 ms	16 ms	20 ms	156 ms
boe01 <-> boe03 red STORAGE	20 ms	21 ms	25 ms	159 ms
boe02 <-> boe03 red STORAGE	20 ms	21 ms	23 ms	228 ms

Tabla 5. Latencia entre *hosts*

En este caso, es la red MANAGEMENT la que sufre una gran diferencia entre las latencias de la conexión entre "boe01" "boe02", y las conexiones con "boe03". Este resultado fue inesperado. Tras investigar acerca de ello, descubrí que la red de STORAGE de "boe03" debía pasar por un *switch* más para llegar a "boe01" y "boe02". En la red STORAGE, como se ve, no hay grandes desigualdades.

DISPOSITIVOS DE ALMACENAMIENTO

Para la medición del rendimiento de escritura de los dispositivos de almacenamiento usados por los OSDs se ha hecho uso de fio (Flexible IO Tester), una popular herramienta de

benchmarking y simulación de cargas de trabajo. Fio permite la creación de ficheros en los que programar el tipo de prueba que se quiere realizar. Las pruebas han sido una de escritura secuencial, y otra de escritura aleatoria, y en el caso de "boe03" solo se ha comprobado la velocidad de uno de sus dispositivos al ser los dos iguales.

Host	Escritura secuencial	Escritura aleatoria
boe01	315 MiB/s (330 MB/s)	245 MiB/s (256 MB/s)
boe02	314 MiB/s (329 MB/s)	273 MiB/s (287 MB/s)
boe03 (uno de los dispositivos)	342 MiB/s (359 MB/s)	345 MiB/s (362 MB/s)

Tabla 6. Velocidades de escritura de los dispositivos de almacenamiento

Se puede ver como la velocidad de escritura secuencial es superior a la velocidad aleatoria, pero las escrituras de objetos en Ceph son escrituras aleatorias de 4MB como máximo, luego es esta segunda velocidad la que realmente resulta de interés. Todos los SSDs ofrecen velocidades similares tanto de escritura secuencial como aleatoria.

TEST OSDS

Hechas las pruebas en el *hardware* disponible, se procede a comprobar su rendimiento una vez incorporado al servicio de Ceph. La herramienta usada para la realización de este *benchmark* es la incluida en el paquete CLI de Ceph. Los OSDs 0 y 1 corresponden a los SSD de los *hosts* "boe01" y "boe02" respectivamente, mientras que el tercer OSD engloba los dos SSDs del *host* "boe03", unidos mediante un volumen lógico LVM.

OSD	Operaciones por segundo	Bytes por segundo
osd.0	90,90	381,255 MB/s
osd.1	89,39	374,908 MB/s
osd.2	113,06	474,218 MB/s

Tabla 7. Rendimiento de los OSDs

Como era de esperar, los OSDs de los dos *hosts* "boe01" y "boe02" tienen un rendimiento ligeramente inferior al OSD del *host* "boe03", del mismo modo que sucedía en el test fio. Los resultados son mejores que los del test fio ya que la monitorización del rendimiento de los OSDs tiene en cuenta las operaciones concurrentes.

TEST RADOS

A continuación, se pueden hacer pruebas sobre el rendimiento de RADOS, que recordemos que es el algoritmo de almacenamiento distribuido que vertebra Ceph. RADOS escribe la información redistribuyendo los objetos de 4MB en los PGs de cada OSD.

	Escritura	Lectura secuencial	Lectura aleatoria
Ancho de banda máximo	393 MB/s	-	-
Ancho de banda medio	448 MB/s	1318 MB/s	1308 MB/s
Ancho de banda mínimo	320 MB/s	-	-

Nº máximo de Operaciones IO	112 IOPS	353 IOPS	356 IOPS
Nº medio de Operaciones IO	98 IOPS	329 IOPS	327 IOPS
Nº mínimo de Operaciones IO	80 IOPS	293 IOPS	297 IOPS
Latencia máxima	260,82 ms	269,97 ms	262,17 ms
Latencia media	162,94 ms	47,77 ms	48,06 ms
Latencia mínima	18,64 ms	11,85 ms	2,27 ms

Tabla 8. Rendimiento de algoritmo RADOS

Se observa como en Ceph no tiene sentido hablar de lectura secuencial o aleatoria, ya que todos los datos se encuentran esparcidos entre los PGs de todos los OSDs. La herramienta "rados bench" toma sus resultados de un test de varias escrituras y lecturas concurrentes a lo ancho del cluster, de ahí que los resultados sean incluso mejores que los del test de OSDs.

TEST RBD

El espacio de almacenamiento de bloques de Ceph también incluye sus propias herramientas de *benchmark* adaptadas. RBD es capaz de montar las imágenes que almacena en el sistema de ficheros de un *host* como si fuera un volumen de almacenamiento más. Aprovechando esta funcionalidad, se han hecho las siguientes pruebas:

	Operaciones por segundo	Bytes por segundo
Escritura secuencial	64376,8 IOPS	251 MiB/s
Escritura aleatoria	22644,8 IOPS	88 MiB/s
Lectura secuencial	41985,7 IOPS	164 MiB/s
Lectura aleatoria	54843,6 IOPS	214 MiB/s

Tabla 9. Rendimiento almacenamiento de bloques RBD

En el espacio RADOS, las escrituras secuencial y aleatoria vuelven a cobrar sentido, ya que corresponden a ubicaciones de los datos abstraídas sobre los objetos de RADOS. Se aprecia en el espacio RBD como era esperable un menor rendimiento de escritura y lectura respecto al que se tenía en RADOS. Resulta curioso como si bien la escritura secuencial es más rápida que la aleatoria con las lecturas secuencial y aleatoria sucede lo contrario.

TEST VELOCIDAD DESPLIEGUE VMS

Para comprobar el tiempo que demora OpenNebula en levantar una VM operativa sobre el espacio RBD del Ceph, se ha elaborado un script que cronometra el tiempo que se tarda desde que se levanta una VM que lance un ping al estar lista, hasta que se recibe dicho ping.

```
#!/bin/bash

onetemplate instantiate Ubuntu\ 22.04-test
echo "levantando maquina"
start_time=$(date +%s)

tcpdump -i br_mgmt -c 1 icmp | grep "ICMP echo request" > /dev/null 2>&1

end_time=$(date +%s)
elapsed_time=$((end_time - start_time))
echo "Tiempo transcurrido: $elapsed_time segundos"
```

Ilustración 16. *Script* para cronometrar el tiempo de arrancado de una VM

Tras la ejecución sucesiva del *script*, los tiempos recogidos fueron de 34, 37 y 34 segundos.

TEST RENDIMIENTO VMS

Por último, se van a realizar pruebas de rendimiento desde dentro de las VMs levantadas desde OpenNebula sobre Ceph. Para la escritura y lectura sin cachear, se va a usar el comando "dd" para escribir una cantidad suficientemente grande de ceros de `/dev/zero` en un fichero, y se va a leer el resultado escrito. La lectura cacheada se calcula mediante la herramienta "hdparm"

Escritura	Lectura sin cachear	Lectura cacheada
93 MB/s	322 MB/s	8806 MB/s

Tabla 10. Velocidad de escritura y lectura en las VMs de OpenNebula

Este rendimiento logrado debería ser suficiente como para servir a gran cantidad de aplicaciones de diversos campos. El CPD es capaz de levantar gran cantidad de instancias virtuales independientes (en las que repartir la memoria RAM y los núcleos de los *hosts*), capaces de soportar cargas de trabajo exigentes, y con comunicación interna y externa a través de los puentes Linux levantados sobre los NICs.

6. CONCLUSIONES Y LÍNEAS FUTURAS

6.1. Conclusiones

Para la elaboración de este trabajo hecho en colaboración con Telefónica se ha llevado a cabo una investigación acerca del estado actual de los Centros de Computación de Datos *Edge*, su potencial y utilidad dentro de las tendencias tecnológicas actuales. Partiendo de la necesidad de desarrollar un prototipo de CPD *Edge*, se han discutido las diferentes arquitecturas que puede seguir la infraestructura del CPD, para concluir que la opción más versátil y flexible es la Hiperconvergente. También se ha hablado de la aparente hegemonía de las nubes públicas, y de la necesidad *de facto* de que el CPD en propiedad a diseñar pueda ser gestionado de la misma forma que una nube pública.

Planteada la conceptualización, se abordó un análisis de las herramientas *software* que podrían ser más adecuadas para implementar el CPD, eligiendo OpenNebula como gestor de la nube y los recursos de computación, y Ceph para la creación de un espacio de almacenamiento compartido entre *hosts*. Con estas herramientas, se han propuesto dos arquitecturas diferentes que garanticen la hiperconvergencia de los *hosts*: una con el plano de control ubicado en *hosts* aparte, y otra con el plano incluido sobre los mismos *hosts* de virtualización. Ambas arquitecturas incluyen su propia solución de monitorizado interno mediante las herramientas estándar para ello actualmente en el mercado: Prometheus y Grafana. A continuación, se han explicado de forma breve y concisa los diferentes pasos seguidos para el despliegue del CPD, los cuales se pueden ejecutar de manera automatizada mediante un *script* Ansible que no requiere de ninguna interacción del administrador. Para terminar, se han hecho varias pruebas de rendimiento para demostrar la utilidad del diseño, el cual se encuentra actualmente en uso como proveedor de VMs para la investigación de otros proyectos en Telefónica.

6.2. Líneas Futuras

Una de las primeras decisiones a la hora de definir el CPD fue la de escoger Ceph como herramienta de Storage Definido por Software (SDS). Aunque esta es una de las opciones más populares en CPDs de producción y la recomendada por OpenNebula, su alto consumo de recursos y enfoque en la escalabilidad hacen que no siempre sea la mejor opción para *hosts* hiperconvergentes. LINBIT a pesar de ser una opción OpenSource no tan valorada, sí se especializa en su concepción en arquitecturas HCI y también se encuentra integrada como solución de almacenamiento dentro de OpenNebula [48].

En caso de continuar con Ceph, añadir nuevos *hosts* al *cluster* no es excesivamente complicado, pero hay que tener en cuenta una serie de pasos. Hay que configurar nuevamente su apartado de red, instalar las librerías necesarias, incorporarlo al *cluster* Ceph, incluirlo en la nube OpenNebula como *hosts* hipervisor, levantar sus *exporters* para el monitorizado, etc. Queda por

definir un nuevo *playbook* de Ansible que sea capaz de hacer toda la configuración para nuevos *hosts* del *cluster*, tanto si van a contribuir al Ceph con otro dispositivo de almacenamiento como si no.

Para seguir optimizando el CPD, es necesario rastrear y localizar los posibles cuellos de botella en el rendimiento. Actualmente es posible hacerlo en parte mediante las métricas obtenidas en Prometheus, pero para rastrear los procesos del *cluster*, Ceph ofrece integración con Jaeger como *backend* de rastreo. El CPD actualmente no habilita este servicio, pero es fácilmente implementable gracias a Cephadm [49]. Con o sin rastreo, se pueden también hacer pruebas de rendimiento con diversas configuraciones extra del CPD aún no probadas, como la posibilidad de emplear tramas "jumbo" de 9000 bytes como MTU en la red de STORAGE.

Es posible hacer más seguro el CPD con medidas como restricciones en los puertos expuestos mediante firewall y/o iptables. Ceph también permite encriptar su tráfico con el protocolo msgr2, pero nuevamente queda por investigar su verdadero impacto en el rendimiento de las VMs de OpenNebula.

Por último, a fecha de la presentación de este trabajo se está estudiando la capacidad del CPD de servir en diversas aplicaciones, entre las que destaca el despliegue de *cores* 5G en el *Edge* mediante la implementación Open5GS [50]. De momento las pruebas están siendo satisfactorias, y se espera que en un futuro próximo los *cores* desplegados en OpenNebula sirvan para su uso en la investigación de otros proyectos europeos como el de 6G-SANDBOX [51]

7. BIBLIOGRAFÍA

[1] C. J. B. Cano, «¿Qué implica la llegada del 5G?,» 2 Enero 2022. [En línea]. Available: https://theconversation.com/que-implica-la-llegada-del-5g-162880.

[2] Economía Digital, «España, ante el reto de construir 1.000 'minicentros' de datos en 2030,» 3 Abril 2021. [En línea]. Available: https://www.expansion.com/economia-digital/2021/04/03/606820ff468aeb03038b45d2.html.

[3] Barbara IOT, «barbaraiot.com,» 2022. [En línea]. Available: https://resources.barbaraiot.com/es/barometro-del-edge-computing-industrial.

[4] IDC, «New IDC Spending Guide Forecasts Edge Computing Investments Will Reach $208 Billion in 2023,» 15 Febrero 2023. [En línea]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS50386323. [Último acceso: 29 Abril 2023].

[5] Global Market Insights, «Edge Data Center Market Size By Component,» Noviembre 2022. [En línea]. Available: https://www.gminsights.com/industry-analysis/edge-data-center-market.

[6] Global Market Insights, «Micro Data Center Market Size by Component,» Septiembre 2020. [En línea]. Available: https://www.gminsights.com/industry-analysis/micro-data-center-market.

[7] Fortune Business Insights, «Hyper Converged Infrastructure Market,» Marzo 2022. [En línea]. Available: https://www.fortunebusinessinsights.com/hyper-converged-infrastructure-market-106444. [Último acceso: 17 Mayo 2023].

[8] S. Besteman, «Why does 5G need Edge Computing in a Micro Data Center?,» Junio 2020. [En línea]. Available: https://www.kingston.com/en/blog/servers-and-data-centers/the-need-for-edge-data-centers.

[9] IDC, «blogs.idc.com,» 4 Noviembre 2019. [En línea]. Available: https://blogs.idc.com/2019/11/04/how-you-contribute-to-todays-growing-datasphere-and-its-enterprise-impact/.

[10] S. Wright, «Autonomous cars generate more than 300 TB of data per year,» 2 Julio 2021. [En línea]. Available: https://www.tuxera.com/blog/autonomous-cars-300-tb-of-data-per-year/.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li y L. Xu, «Edge Computing: Vision and Challenges,» *IEEE INTERNET OF THINGS JOURNAL, VOL. 3, NO. 5,* OCTOBER 2016.

[12] A. Reyes, C. Rodriguez y D. Esenarro, «Hyper Converged Systems Applied (HSA),» *International Journal of Recent Technology and Engineering (IJRTE),* 2019.

[13] Nutanix, The Definitive Guide to Hyperconverged Infrastructure.

[14] B. J. S. Lasluisa, «Estudio de Factibilidad para la Implementación de una Infraestructura de Hiperconvergencia de Alta Disponibilidad en el Data Center Experimental,» 2019. [En línea]. Available: https://dspace.udla.edu.ec/bitstream/33000/10913/1/UDLA-EC-TIRT-2019-03.pdf.

[15] E. Maya-Olalla, M. Dominguez-Limaico, S. Meneses-Narvaez, P. D. Rosero-Montalvo, S. Narvaez-Pupiales, M. Zambrano Vizuete y D. H. Peluffo-Ordóñez, «Design and tests to implement hyperconvergence into a datacenter: preliminary results,» de *Advances in Emerging Trends and Technologies: Volume 1*, Springer International Publishing, 2020, pp. 54-66.

[16] Red Hat, «La nube pública,» 3 Enero 2023. [En línea]. Available: https://www.redhat.com/es/topics/cloud-computing/what-is-public-cloud.

[17] M. Semilof, S. J. Bigelow y K. Casey, «TechTarget,» Julio 2021. [En línea]. Available: https://www.techtarget.com/searchcloudcomputing/definition/cloud-management. [Último acceso: 23 Mayo 2023].

[18] ShapeBlue, «How to Choose a Cloud Management Platform,» 7 Junio 2021. [En línea]. Available: https://www.shapeblue.com/how-to-choose-a-cloud-management-platform/.

[19] Openstack, «Conceptual Architecture,» 19 Abril 2023. [En línea]. Available: https://docs.openstack.org/install-guide/get-started-conceptual-architecture.html. [Último acceso: 29 Abril 2023].

[20] C. Daffara, «Comparing OpenNebula and OpenStack: Two Different Views on the Cloud,» 2 Octubre 2014. [En línea]. Available: https://opennebula.io/comparing-opennebula-and-openstack-two-different-views-on-the-cloud/.

[21] I. M. Llorente, «Eucalyptus, CloudStack, OpenStack and OpenNebula: A Tale of Two Cloud Models,» 4 Febrero 2013. [En línea]. Available: https://opennebula.io/eucalyptus-cloudstack-openstack-and-opennebula-a-tale-of-two-cloud-models/.

[22] OpenNebula, «OpenNebula Overview,» 18 Noviembre 2022. [En línea]. Available: https://docs.opennebula.io/6.6/overview/opennebula_concepts/opennebula_overview.html.

[23] Open Nebula, «Open Cloud Reference Architecture,» Abril 2021. [En línea]. Available: https://support.opennebula.pro/hc/en-us/article_attachments/360019500617/OpenNebula_-_Open_Cloud_Reference_Architecture_r2.2_20210428.pdf.

[24] OpenNebula, «Choosing the Right Storage for Your Cloud,» Abril 2021. [En línea]. Available: https://support.opennebula.pro/hc/en-

us/article_attachments/360019741598/OpenNebula_-_Choosing_the_Right_Storage_for_your_Cloud_r1.0_20210426.pdf.

[25] OpenNebula, «Edge Cloud Reference Architecture,» Mayo 2021. [En línea]. Available: https://support.opennebula.pro/hc/en-us/article_attachments/360019858317/OpenNebula_-_Edge_Cloud_Reference_Architecture_r2.2_20210507.pdf. [Último acceso: 25 Abril 2023].

[26] Ceph, «Pools,» Noviembre 2022. [En línea]. Available: https://docs.ceph.com/en/quincy/rados/operations/pools/.

[27] S. A. Weil, A. W. Leung, S. A. Brandt y C. Maltzahn, «RADOS: A Scalable, Reliable Storage Service for Petabyte-scale,» 11 Noviembre 2007. [En línea]. Available: https://ceph.io/assets/pdfs/weil-rados-pdsw07.pdf.

[28] Packt Publishing, Ceph Cookbook, Birmingham: Packt Publishing Ltd., 2016.

[29] Ceph, «Architecture,» Marzo 2023. [En línea]. Available: https://docs.ceph.com/en/latest/architecture/.

[30] Prometheus, «Overview,» 2023. [En línea]. Available: https://prometheus.io/docs/introduction/overview/.

[31] Ceph, «OS Recommendations,» 5 Abril 2022. [En línea]. Available: https://docs.ceph.com/en/quincy/start/os-recommendations/#platforms. [Último acceso: 25 Abril 2023].

[32] OpenNebula, «Build Dependencies,» 8 Febrero 2023. [En línea]. Available: https://docs.opennebula.io/6.6/integration_and_development/references/build_deps.html. [Último acceso: 25 Abril 2023].

[33] Red Hat, «Ceph network configuration,» 20 Marzo 2023. [En línea]. Available: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/6/html/configuration_guide/ceph-network-configuration. [Último acceso: 25 Abril 2023].

[34] OpenNebula, «OpenNebula Front-end HA,» 29 Agosto 2022. [En línea]. Available: https://docs.opennebula.io/6.6/installation_and_configuration/ha/frontend_ha.html#frontend-ha-setup. [Último acceso: 1 Mayo 2023].

[35] Ceph, «Monitoring Services,» 29 Abril 2023. [En línea]. Available: https://docs.ceph.com/en/quincy/cephadm/services/monitoring/. [Último acceso: 7 Mayo 2023].

[36] OpenNebula, «miniONE,» 15 Marzo 2023. [En línea]. Available: https://github.com/OpenNebula/minione. [Último acceso: 7 Mayo 2023].

[37] OpenNebula, «KVM Driver,» 28 Noviembre 2022. [En línea]. Available: https://docs.opennebula.io/6.6/open_cluster_deployment/kvm_node/kvm_driver.html?highlight=hardware. [Último acceso: 7 Mayo 2023].

[38] Ceph, «Hardware Recommendations,» 14 Julio 2022. [En línea]. Available: https://docs.ceph.com/en/quincy/start/hardware-recommendations/. [Último acceso: 10 Mayo 2023].

[39] Red Hat, «Minimum hardware recommendations for containerized Ceph,» [En línea]. Available: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/6/html/hardware_guide/minimum-hardware-recommendations-for-containerized-ceph_hw. [Último acceso: 10 Mayo 2023].

[40] Suse, «Requisitos y recomendaciones de Hardware,» 17 Octubre 2022. [En línea]. Available: https://documentation.suse.com/es-es/ses/7.1/html/ses-all/storage-bp-hwreq.html. [Último acceso: 10 Mayo 2023].

[41] IBM, «Minimum hardware recommendations for containerized Ceph,» 17 Marzo 2023. [En línea]. Available: https://www.ibm.com/docs/en/storage-ceph/5?topic=hardware-minimum-recommendations-containerized-ceph. [Último acceso: 10 Mayo 2023].

[42] Ansible, «About Ansible,» 30 Marzo 2023. [En línea]. Available: https://docs.ansible.com/ansible/latest/index.html. [Último acceso: 11 Mayo 2023].

[43] Terraform, «What is Infrastructure as Code with Terraform,» [En línea]. Available: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code. [Último acceso: 14 Mayo 2023].

[44] Ceph, «Installing Ceph,» 8 Marzo 2023. [En línea]. Available: https://docs.ceph.com/en/quincy/install/. [Último acceso: 14 Mayo 2023].

[45] Ceph, «Deploying a new Ceph Cluster,» 31 Enero 2023. [En línea]. Available: https://docs.ceph.com/en/quincy/cephadm/install/. [Último acceso: 14 Mayo 2023].

[46] ceph-ansible, «OSD Scenarios,» 29 Agosto 2018. [En línea]. Available: https://docs.ceph.com/projects/ceph-ansible/en/stable-3.1/osds/scenarios.html. [Último acceso: 14 Mayo 2023].

[47] Ceph, «ceph-disk -- Ceph disk utility for OSD,» 4 Diciembre 2017. [En línea]. Available: https://docs.ceph.com/en/mimic/man/8/ceph-disk/. [Último acceso: 14 Mayo 2023].

[48] LINBIT, «Software defined Storage with OpenNebula & LINSTOR,» [En línea]. Available: https://linbit.com/opennebula/. [Último acceso: 14 Mayo 2023].

[49] Ceph, «Tracing Services,» 16 Junio 2022. [En línea]. Available: https://docs.ceph.com/en/latest/cephadm/services/tracing/. [Último acceso: 14 Mayo 2023].

[50] S. Lee, «Introduction to Open5GS,» 13 Junio 2023. [En línea]. Available: https://open5gs.org/open5gs/docs/guide/01-quickstart/. [Último acceso: 14 Junio 2023].

[51] 6G-SANDBOX SNS HE PROJECT, «ABOUT 6G-SANDBOX,» 2022. [En línea]. Available: https://6g-sandbox.eu/. [Último acceso: 14 Junio 2023].