Universidad De Zaragoza

Trabajo De Fin De Grado

Despliegue de una federación *cloud*: instanciación, presentación de recursos, gestión de la pertenencia y monitorización y recuperación ante desastres

Deployment of a *cloud* federation: instantiation, resource submission, membership management and monitoring and disaster recovery.

Lucas Cauhé Viñao

Director: Eduardo Tomás Fiat Gracia
Ponente: Unai Arronategui Arribalzaga

Grado en Ingeniería Informática
Computación

2025

Gracias a mi familia y amigos por acompañarme en este proceso.

Gracias a Eduardo, por guiarme en el desarrollo del proyecto, y a Unai, por aportar una mirada objetiva y sincera.

Resumen

Este proyecto tiene como objetivo definir, implementar y validar un modelo de federación entre distintos despliegues del software de gestión de máquinas virtuales *OpenNebula*. Una solución de recuperación ante desastres para la federación es esencial para incorporar el modelo planteado en un entorno real. Con ello se pretende ampliar la tecnología *cloud* existente, siendo capaz de abordar escenarios complejos.

La arquitectura del modelo de federación está formada por tres planos de acción: el de confianza, donde se definen los mecanismos de interacción entre la infraestructura de las distintas entidades; el de gestión, en el que se definen las políticas, métricas y organización de usuarios para gobernar la federación; y el de uso, el cual presenta el catálogo de recursos de la federación que los usuarios emplean, según dictan las políticas definidas. Los elementos funcionales del plano de gestión son, la seguridad en las comunicaciones entre entidades, el gestor de políticas y el servicio de monitorización. Se interactúa con estos servicios a través del sistema de eventos de *OpenNebula*, que permite ejecutar la lógica personalizada que hace cumplir con las políticas definidas.

El sistema desplegado tiene en consideración la naturaleza de cada componente que lo forma; el almacenamiento persistente y volátil de estados, bases de datos y políticas se guardan en *Ceph*, los servicios de la federación que requieren cómputo se han virtualizado en *OpenNebula* y cada componente basa su configuración en las políticas que se han implementado.

Se ha escogido el modelo de federación para poder incorporar miembros a un entorno de confianza, donde compartir usuarios y recursos de virtualización, como imágenes de disco para máquinas virtuales. Este modelo permite a cada entidad mantener una gestión local de su infraestructura sin tener que acomodarse a las restricciones individuales del resto de entidades. Además, cada entidad es soberana de sus recursos, pudiendo entrar y salir de la federación en cualquier momento sin penalizar al funcionamiento del sistema, siempre que se haga de forma controlada.

El sistema de recuperación ante desastres confeccionado actúa sobre la federación, no cada elemento independiente, homogeneizando las políticas destinadas a este ámbito. Se ha diseñado una arquitectura para el *backup*, donde los servicios están virtualizados en *OpenNebula* y almacena las copias en *Ceph*. Para que la implementación de las estrategias y trabajos de *backup* diseñados sea lo más directa y efectiva posible, se ha valorado positivamente que *OpenNebula* integre la herramienta de *backup*.

La fuente de código en la que están definidos los componentes de la infraestructura y los servicios de la federación cumple con las políticas establecidas. Las herramientas de despliegue automático escogidas interpretan esta fuente de código y despliegan los componentes definidos. Para describir la interacción entre entidades y la gestión de los componentes de la infraestructura en masa, se han definido una serie de manifiestos en un lenguaje declarativo. Esto permite mantener una configuración homogénea que pueda replicarse a las entidades cuando sea pertinente. Así, los planes de despliegue entre entidades serán idénticos, evitando fallos humanos.

Al no contar con la infraestructura necesaria para realizar el despliegue real del sistema diseñado, se ha optado por desarrollar un entorno de simulación en un único nodo físico, donde desplegar y probar los servicios de la federación.

Ha sido satisfactorio el grado de completitud de los objetivos y los conocimientos adquiridos. La puesta en marcha ha sido un éxito y las pruebas han demostrado la corrección del diseño y la implementación.

Abstract

The goal of this project is to define, implement and validate a federation model between different OpenNebula deployments, a virtual machine management software. In order to deploy this model in a real environment, a disaster recovery plan must be designed. This project aims to extend the current cloud technology, being able to face complex cloud designs.

The architecture introduced for the federation model consists of three levels of action: the trust level, where the means of interaction between the infrastructure of the different entities is described; the management level, where the policies, metrics and user organisation for governing the federation are defined; and the usage level, which introduces the federation resources catalog that users interact with, as it is allowed by the defined policies. The working components from the management level are, communications security between entities, policies manager and the monitoring service. The interaction with these services is accomplished with the event system builtin in OpenNebula, that allows executing custom logic, forcing the resources to be compliant with the defined policies.

The system being deployed takes into account the nature of each component that it is made of; persistent and volatile storage, data bases and policies are kept in Ceph, the services that require compute power have been virtualized within OpenNebula and every component's configuration is compliant with the implemented policies.

The federation model has been chosen so that new members can be added to a trusted environment, where to share users and virtual resources, such as disk images for virtual machines. This model allows to locally manage each entity's infrastructure resources, disregarding individual constraints from others. Moreover, each entity has sovereignty over its own resources, and may anter or leave the federation at any time without affecting the system's workflow.

The system designed for disaster recovery acts on the federation as a whole, not each element individually, standarizing policies in this environment. The backup system has its own architecture design, where services are virtualized within OpenNebula and the data is stored in Ceph. Tools that integrate directly with OpenNebula have been upvoted in order to benefit the builtin backup strategies and jobs, this will allow for a more effective implementation of the backup system.

The source code defining the infrastructure components and the federation services is compliant with the stablished policies. The automated deployment tools chosen for the project interpret the source code and deploy the defined components. A series of manifests, written in a declarative programming language, describe the interaction between entities and the infrastructure management at scale. This allows a standarized configuration for every component, able to be replicated to each entity whenever is required. Execution plans will be the same for different entities and free from human errors.

A simulation environment within a single physical node has been developed due to the lack of real infrastructure. Tests and the designed deployment take this into account.

Project goals have been met, the deployment has been successful and tests show the wellness of the design and implementation. Also, a substantial knowledge has been acquired along the way.

Glosario

Bridge huérfano Interfaz *bridge* sin interfaz física como *master*.

Cloud On-Premise Tipo de *cloud* cuya infraestructura, usuarios y software reside en la propia organización y es gestionada por la misma.

Cloud privado Tipo de *cloud* que utiliza una infraestructura de nube dedicada a una organización concreta. Un *cloud* privado puede ser On-Premise o estar construido sobre un *cloud* público.

Device class Etiqueta asignada a un OSD.

Entidad Organización, sede o grupo de personas que participa en la federación y que está ligado a una zona geográfica y unos recursos informáticos concretos.

Federación Medio para habilitar una interacción o colaboración de algún tipo. En este documento se entiende como federación *cloud*: habilidad de dos o más proveedores *cloud* para interactuar o colaborar cooperativamente, estableciendo un espacio de nombres común para los recursos compartidos (usuarios, grupos, políticas, etc...).

Hook OpenNebula Plantilla en texto plano, interpretada por el modelo de ejecución programada de eventos de OpenNebula (*Hook Execution Manager*), que liga la ejecución de un *script* personalizado a un cambio en el estado de un recurso o una llamada API.

Infraestructura Conjunto de recursos informáticos sobre los que se basa la arquitectura de un sistema. En este proyecto se entiende por infraestructura de virtualización o red según el contexto.

Imagen no persistente En *OpenNebula*, tipo de imagen de disco sobre la que no se despliegan máquinas virtuales, solo guarda el estado concreto de una máquina.

Imagen persistente En *OpenNebula*, tipo de imagen de disco sobre la que permanecen las escrituras que realiza una máquina virtual tras la destrucción de la misma. No confundir con el clonado enlazado o total de QEMU.

Imagen volátil Tipo de imagen de disco que solo escribe las diferencias respecto de la imagen que tiene como base (en *OpenNebula*, no persistente). Una vez eliminada la máquina virtual, el disco ligado a este tipo de imagen se destruye.

Nodo Computadora que forma parte de un conjunto de máquinas que sirven un propósito similar.

Política Directrices que rigen la actuación de entidades, usuarios y recursos *cloud* en la federación. Son la base fundamental para establecer la gobernanza en la federación.

Pool Ceph Agrupación lógica de un conjunto de objetos RADOS sobre la que se aplican un conjunto de reglas de replicación y mantienen los datos distribuidos con el uso de *placement groups*.

Relación maestro-esclavo Caracterización de dos o varios sistemas por el que el *maestro* completa cierto tipo de peticiones que el *esclavo* reexpide al *maestro* o desecha. El *maestro* replica su estado en los *esclavos*.

Servicio OpenNebula Plantilla que define el despliegue de una o varias máquinas virtuales y recursos ligados a ella.

Servidor o host Computadora física o virtual,con especial capacidad para cómputo, que ofrece un servicio a un cliente. En este proyecto hay servidores de virtualización, de gestión de políticas y el núcleo de *OpenNebula*

Servidor puente Almacenamiento intermedio donde asentar primero los imágenes al descargarlas o migrarlas en *OpenNebula*.

Zona OpenNebula Instalación del software de *OpenNebula* que tiene cada entidad federada y está en una relación de maestro-esclavo con otra instalación diferente.

Índice

Capítulo 1.

Introducción

1.1. Contexto

Este proyecto se ha realizado en el Servicio de Informática y Comunicaciones de la Universidad de Zaragoza (SICUZ); encargado de mantener la operativa informática de la Universidad y la infraestructura que le da soporte. En este contexto, la gestión de un *cloud* on premise permite controlar la información y servicios ofrecidos desde la Universidad. Este proyecto se alinea con los planes de desarrollo en tecnologías *cloud* de la Unión Europea, entre los que se encuentran: la estrategia de gestión datos, en la que la federación de *clouds* juega un rol vital, y la protección de datos en espacios seguros frente a su almacenamiento en empresas de *cloud* privadas.

En este departamento se está implementando *Boira*[1], un proyecto de colaboración interuniversitario, donde se ha desplegado *OpenNebula* [1] siguiendo una arquitectura distribuida. Distintas universidades aportan parte de su infraestructura tecnológica para dar soporte a este despliegue. Hay tres núcleos de *OpenNebula* repartidos entre tres universidades formando un único *cloud* on-premise. Esto supone homogeneizar la infraestructura (almacenamiento, red y cómputo) para que cada universidad ofrezca y reciba rendimientos similares.

La confianza entre las distintas universidades se establece de palabra y siguiendo una serie de buenas prácticas establecidas de forma general. Así, se ha contemplado la idea de federar cada uno de los *cloud* on-premise de cada universidad, permitiendo una gestión abstracta y gobernada mediante políticas que afectan al uso y los recursos presentados.

1.2. Motivación y alcance

El uso de una federación *cloud* on-premise permite tener una gestión local de la infraestructura y compartida de la información de usuarios. El nivel de confianza entre las entidades federadas rige el control de acceso a los recursos presentados.

El despliegue de un *cloud* on-premise permite, a nivel de mantenimiento y hardware de la infraestructura, tener unos costes fijos; el tráfico puede entrar y salir sin restricciones ajenas a la organización (*egress fees*) y la escalabilidad de los recursos no tiene costes monetarios adicionales. Los sistemas cuyo estado avanza consensuadamente entre los servicios que los componen, son sensibles a las latencias entre ellos. Con un *cloud* on premise hay una mayor flexibilidad para cumplir con los tiempos que estos sistemas requieren. Además, el rendimiento de los programas informáticos ejecutados, no se verá afectado por entornos compartidos con usuarios ajenos.

[1] https://boira.es/

La Unión Europea está apostando por el uso de este tipo de tecnologías que permiten a las organizaciones ser soberanas de su propia información, independizándose de contratos con empresas privadas donde almacenar la información de sus usuarios y clientes. No obstante, la puesta en marcha de estos proyectos supone una gran dedicación en tiempo y personal para llegar a acuerdos en las políticas de uso e incorporación de entidades.

1.3. Objetivos

El objetivo principal del proyecto es definir, implementar y validar un modelo de federación entre dos instancias de cloud on-premise *OpenNebula* usando Ceph [2] como sistema de almacenamiento distribuido. Este se contempla como un prototipo que permita identificar posibles defectos del diseño real y confeccionar el despliegue de los elementos esenciales. También el de contribuir a la creación de tecnología *cloud*, capaz de abordar escenarios complejos en múltiples zonas geográficas.

Se va a dar una solución de recuperación ante desastres para los recursos de la federación, siendo estos: sistemas de ficheros (de máquinas virtuales, de configuración y estado de la federación), estado de máquinas virtuales y catálogo de servicios de la federación (imágenes persistentes y volátiles del Marketplace privado de *OpenNebula*).

El uso de la federación que hagan los usuarios y recursos, se basará en las políticas de gobernanza especificadas. También han de intervenir un sistema de monitorización y otro de control de accesos que validará las políticas definidas.

Los usuarios comunes de la federación en *OpenNebula* son visibles para el administrador. Estos usuarios tienen, por defecto, permisos restringidos a la zona en la que han sido creados. Se va a crear una organización más compleja que permita tener administradores locales y grupos que controlen cierto tipo de recursos.

1.4. Metodología

El proyecto se desarrolla mediante una metodología iterativa, permitiendo ajustar progresivamente el diseño y la implementación de la federación *cloud* en función de los resultados obtenidos en cada ciclo. En una primera fase se definen la arquitectura, la infraestructura base y las políticas iniciales. Posteriormente, se implementan gradualmente los componentes de la federación y del sistema de *backup*, evaluando su funcionamiento a través de la comprobación del estado de cada componente. Los resultados de estas validaciones sirven para refinar y optimizar las políticas diseñadas inicialmente, adaptándolas a las necesidades reales del entorno y consolidando finalmente una infraestructura estable y alineada con los objetivos del proyecto.

1.5. Organización de la memoria

Se da comienzo por el estudio del estado del arte de los conceptos, tecnologías y herramientas relacionadas con la federación *cloud*. Seguidamente se presenta un análisis de los requisitos del problema y el diseño del sistema planteado. El diseño se divide en tres partes. Primero, la arquitectura del prototipo de federación, tomando inspiración del modelo del NIST [3], donde

se comentan las relaciones entre los componentes de cada plano de la federación: mecanismos de confianza entre las entidades, validador de políticas y monitorización y el servicio de catálogo de recursos y usuarios. Después, se aborda la infraestructura que da soporte a la arquitectura de federación, comentando aspectos de red, almacenamiento y despliegue, y se termina con el sistema de recuperación ante desastres.

Se sigue con la implementación del prototipo diseñado, describiendo el despliegue del catálogo, sistema de validación y control de accesos, monitorización e interacción con estos servicios. Se cuenta cómo se ha simulado el entorno real de la infraestructura, cómo cada componente se ha tenido que adecuar al entorno de despliegue y se hace referencia a los problemas encontrados por este motivo. Se termina la implementación con la recuperación ante desastres.

Por último se comentan las pruebas que se han llevado a cabo y los métodos de validación seguidos, para asegurar que la infraestructura se encuentra en el estado deseado en el momento del despliegue de los servicios. En otro apartado se hacen una valoración final y personal del proyecto.

Al final de la memoria hay una serie de anexos donde se documentan en detalle la distribución de carga de trabajo para este proyecto, la especificación de las políticas implementadas, problemas encontrados, una tabla con aspectos de red de los servicios desplegados, el código que valida las políticas y los manifiestos y código personalizado que definen los componentes de la federación.

Capítulo 2.

Estado del arte

El *NIST* [3] presenta un modelo de federación *cloud* dividido en tres planos: confianza, donde cada entidad aplica una serie de protocolos que le permiten compartir información de su sistema con otro; administración, donde se hacen cumplir unas políticas de gobernanza y se definen usuarios y controles de acceso; y el plano de uso, donde los usuarios de la federación emplean los servicios publicados mediante la interacción con un servicio de catálogo. Se entiende por confianza a la esperanza de que la otra entidad satisfaga lo previamente pactado, es decir, sea el cumplimiento de las políticas definidas para la gobernanza, lo que permita a entidades incorporarse al clúster.

La solución *cloud* on-premise viene ya dada para el proyecto, *OpenNebula*, que permite federar varias instancias en lo que se denominan zonas. *OpenNebula* es un software de gestión de máquinas virtuales en nodos de virtualización separados. La gestión de usuarios que ofrece es similar a la de UNIX, basándose en usuarios, grupos y permisos e incorporando otros conceptos como las listas de control de acceso. El método de confianza establecido es la compartición de credenciales de administrador entre todas las zonas. Este protocolo hace que la confianza se establezca de forma manual e implica que varias personas se pongan de acuerdo. Para mejorar este aspecto se ha valorado el uso de *Verifiable Credentials* [4]. Este concepto lo presenta el W3 y permitiría crear una capa de confianza añadida, pero por la complejidad de la solución valorada (red IPFS más sistema de recuperación de credenciales) se ha descartado. Esta solución, no obstante, se ha suplido por el uso de la infraestructura de clave pública (*PKI*) para la replicación entre zonas de *OpenNebula*, ya que la comunicación por defecto es en texto plano.

OpenNebula incorpora conceptos imprescindibles para el diseño, como pueden ser las listas de control de accesos y la alta disponibilidad, de la infraestructura y las máquinas virtuales. Las listas de control de acceso (*ACL* en inglés), son acciones que pueden realizar unos sujetos sobre un objeto. Así, los usuarios de la federación podrán pertenecer a los grupos que tienen permisos sobre máquinas virtuales, pero si no están autorizados a efectuar las acciones descritas en las listas de control de acceso, no podrán interactuar con dichos objetos. La alta disponibilidad (*HA* en inglés), es la característica de un sistema que asegura un cierto rendimiento, normalmente de "Tiempo en línea", por un periodo de tiempo más extenso al normal.

Los componentes de *OpenNebula* de interés para este proyecto son: el núcleo (demonio *oned*), cuya función es guardar y propagar el estado del sistema y gestionar el planificador (*scheduler*); el subsistema de *hooks*, que permite ejecutar lógica personalizada tras haberse disparado un cambio de estado o ejecutado una acción en la API; y el sistema de backup, que presenta la definición de trabajos planificados para grupos de recursos. Además, introduce con el concepto de *datastore*, una capa de abstracción sobre el almacenamiento real que hace uso de drivers específicos para cada solución de almacenamiento soportada.

La arquitectura de federación de *OpenNebula* plantea una replicación jerárquica, donde cada entidad (zona *OpenNebula*) toma el rol de maestro o esclavo. Hay un maestro y uno o más esclavos. Se asegura consistencia secuencial entre zonas, es decir, que las operaciones replicadas están en el mismo orden en el estado de cada zona, sin embargo es posible que los esclavos lean datos anticuados.

De la arquitectura de Ceph, los componentes de mayor valor para el prototipo son RADOS [5], el sistema autónomo de replicación donde se almacenan los objetos del clúster; RadosGW [6], la puerta de enlace con el clúster mediante una interfaz compatible con S3; RBD, que ofrece una representación de un dispositivo de bloque, cuyos sectores de disco se encuentran distribuidos en Ceph, que puede ser montada (*map*) localmente como tal dispositivo o accedido a través de la interfaz de QEMU/KVM; y CephFS [7], el sistema de ficheros compartido compatible con POSIX.

RADOS está formado por monitores, gestores del estado del clúster y recuperación de los datos; OSD, servicios que efectúan el almacenamiento de objetos en disco; *managers*, servicio encargado de exportar métricas del clúster; y MDS, los servicios de gestión de metadatos para CephFS, un estilo de monitores especiales para los sistemas de ficheros. Además se basa en los conceptos matemáticos, no materializados como servicios, de *placement group*, un cálculo estadístico para la distribución (y replicación) de objetos en OSDs, y *pools*, agrupamientos lógicos con unas políticas de replicación personalizables. Los *pools* pueden estar configurados con dos modos o estrategias de replicación: réplica *n* o *erasure code*. Para el modo de réplica *n*, siendo 3 la *n* más utilizada, existen varios modos de copia entre los OSD en el mismo *placement group*, donde *primario-copia* es el más utilizado por tener la menor latencia en la confirmación de las operaciones de escritura. La estrategia de *erasure code* se basa en la definición de *m*, el número de bloques de datos y *k*, bloques de paridad. Esto permite soportar el fallo de hasta *k* OSD (o *k* entidades si se expande el clúster), corregirlo y restaurar la E/S después de su corrección. La primera estrategia permite recuperar datos tras fallo más rápido que la segunda, aunque se penalice con la copia de cada objeto *n* veces.

El proyecto *Gaia X* [8] ha servido de inspiración para encontrar tipos de políticas que implementar en la federación. Estas han sido las de servicio mínimo y resaltar la importancia de que cubran todos los componentes de la infraestructura, usuarios y servicios de la federación. Esto último, ha sido extraída como núcleo conceptual de las políticas definidas en la federación presentada.

XACML [9] es un lenguaje de marcado, basado en XML, para definir políticas de control de accesos basadas en atributos. Es un estándar publicado por *OASIS* que presenta también un arquitectura y un modelo de evaluación de las peticiones de acceso a recursos. Las decisiones de control de acceso se toman en base a un conjunto de reglas, cada una formada por una serie condiciones. Una implementación de este estándar es *Open Policy Agent* [10], que presenta un lenguaje de dominio específico para expresar las políticas y un motor de que valide las peticiones de acceso a recursos. En general, comprueba si el conjunto de reglas de una política se satisfacen dada una entrada y un conjunto de datos.

Las características esenciales que debe presentar un sistema de backup para entornos *cloud* son la gestión de duplicados, compresión y cifrado para el almacenamiento de los datos de tipo objeto. A estos datos se los conoce como datos fríos, es decir, que no se necesitan con frecuencia

pero deben guardarse a largo plazo. *OpenNebula* presenta dos tipos de backup: incremental y completo, con distintos modos de control del sistema de ficheros de las máquinas virtuales. Las herramientas de backup estudiadas han sido Restic y Bacula, siendo la primera la que se ha acabado utilizando. *OpenNebula* cuenta con un driver, aunque incompleto, de Restic, por lo que la definición de trabajos automáticos de backup está integrado. Bacula, pese a ser una herramienta más completa, debería ser desplegada como una máquina virtual y gestionada a parte, como un servicio desconectado de la federación.

Entre las tecnologías estudiadas para el desarrollo de la solución se encuentra la Infraestructura como Código (*IaC* en inglés). Define el proceso de gestión y aprovisionamiento de recursos informáticos en una infraestructura *cloud*, a través de manifiestos interpretables por alguna herramienta. Puppet, Ansible y OpenTofu han sido las herramientas contempladas para la implementación del diseño, pero finalmente se han escogido Puppet [11], dada la capacidad de personalización de los recursos del despliegue, y OpenTofu [12], al ser la herramienta de integración nativa con *OpenNebula* y la rama de código abierto de Terraform.

Capítulo 3.

Análisis y Diseño del sistema

3.1. Análisis del problema

El diseño de La infraestructura que da soporte a las instancias *cloud*, incorpora como propiedades la **tolerancia a fallos y alta disponibilidad**.

Teniendo en cuenta la alta disponibilidad, el entorno *cloud* on-premise requiere bajas latencias entre núcleos para avanzar el estado del sistema de forma consensuada. Se debe utilizar un **sistema de almacenamiento tolerante a fallos y que pueda ser utilizado como datastore de *OpenNebula***. La configuración de cada entidad y el proceso de incorporación a la federación es una parte sensible, ya que hay una relación de orden entre operaciones estricta y precisa. Por ello, **el despliegue debe controlar las relaciones entre recursos y estar automatizado**. Así se evitan fallos recurrentes y la distribución de cambios de configuración es inmediata.

Las políticas que se definan **deben cubrir el uso y control de acceso a los servicios de la federación por parte de los usuarios de las entidades federadas**. Además del almacenamiento, cómputo y estado de la federación. **Debe haber un sistema de monitorización y otro de gestión de las políticas**, que dado un servicio y su uso deseado, dictará si se permite o no el uso del servicio. Las **métricas han de estar disponibles y ser accesibles en un *endpoint*** acordado, de este modo se asegura que la interacción con los servicios sea efectiva.

El **catálogo de servicios de la federación ha de tener una visión unificada del almacenamiento subyacente**, es decir, que independientemente del número de entidades federadas, cada una tiene la responsabilidad de almacenar parte de la información del catálogo. La **ubicación de esa información debe ser transparente al recuperarla**.

La recuperación ante desastres debe cumplir las políticas establecidas, ya que es un servicio más de la federación. El **tipo de datos almacenados en este proceso debe ser de objetos**, ya que responden a la naturaleza del problema: múltiples lecturas, única escritura y recuperación del objeto, del medio de almacenamiento, eficiente. La **información almacenada ha de estar disponible para todas las entidades en cualquier momento**, siendo la recuperación de los datos independiente del estado del servicio de *backup* en cada entidad.

La **definición de los servicios de *backup*, gestor de políticas, monitorización y catálogo debe ser homogénea**, independiente de la configuración local de cada zona y compatible con *OpenNebula*. Esto permite replicar la configuración de estos servicios de manera inmediata entre las distintas entidades federadas.

Por defecto, la propagación del estado de la federación emplea el protocolo HTTP, al estar comunicando información de usuarios, grupos y otros datos sensibles, se considera esencial proteger esta fase, mediante el **cifrado de las comunicaciones entre entidades**. Además,

para asegurar que no cualquier despliegue de *OpenNebula* pueda comenzar a recibir datos de la replicación, se ha de habilitar un **mecanismo de confianza entre entidades a nivel de red (o pertenencia a la federación)**. La infraestructura es naturalmente escalable ya que la capacidad de cómputo y almacenamiento incrementa tras la incorporación de nuevas entidades.

3.2. Diseño de la federación

Se plantea la federación como un *cloud* privado para la información de las entidades federadas y público para los usuarios que utilicen los servicios que presenta. Las políticas que conforman la gobernanza deben ser respetadas a todos los niveles, infraestructura, servicios y aplicaciones, y en cada ámbito, entidades que guardan datos de sus usuarios como *cloud* privado y usuarios que entran a la federación como *cloud* público.

La Figura 1 introduce los componentes principales principales del sistema. Se definen tres planos de abstracción: confianza, gestión y uso. La confianza entre entidades se establecerá mediante los protocolos internos de distribución de credenciales de administración de *OpenNebula*. En el plano de gestión se hará cumplir con las políticas de gobernanza definidas

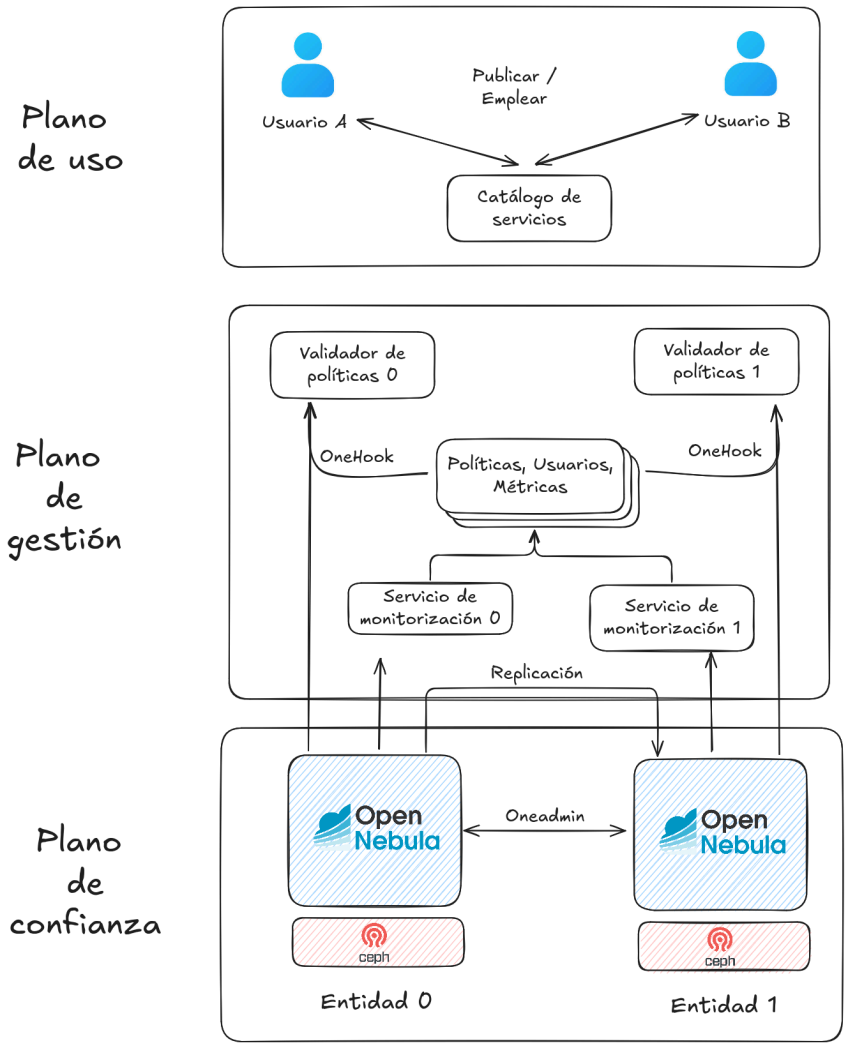

Figura 1: Arquitectura del prototipo de federación.

mediante un sistema de validación. El plano de uso describe la interacción que habrá entre los usuarios de las distintas entidades con los servicios ofrecidos por la federación.

3.2.1. Plano de confianza

El plano de confianza lo forman las instalaciones de *OpenNebula* y *Ceph* locales en cada entidad. A este nivel, y con la arquitectura de federación de *OpenNebula*, se establece la confianza para dos escenarios: pertenencia a la federación y el proceso de replicación entre zonas. Las credenciales del usuario administrador *ondeadmin* se comparten de la entidad *maestra* a la *esclava*, así ambas zonas pertenecen a la misma federación. En la Figura 1 la "Entidad 0" es la *maestra* y la "Entidad 1" la *esclava*. Esta es la primera aproximación a la pertenencia de una entidad a la federación. A nivel de infraestructura, se establece otro mecanismo para gestionar la pertenencia. El proceso de replicación entre entidades es confiable mediante el uso de algún protocolo de comunicación cifrada.

La confianza entre cada instancia de *OpenNebula* y su clúster Ceph se establecerá a través de un usuario de Ceph que tendrá los permisos necesarios para consumir los recursos del clúster y almacenar los datos de *OpenNebula*.

3.2.2. Plano de gestión

En el plano de gestión, una serie de políticas definen la gobernanza de la infraestructura y controlan accesos de usuarios a servicios. El comportamiento y rendimiento del *backup*, sistema de validación de políticas y máquinas virtuales de usuarios, que se ejecutan en la federación, está monitorizado; atendiendo también a las políticas definidas.

Las políticas diseñadas cubren los componentes definidos de la infraestructura y en los distintos planos de la federación. El acceso al almacenamiento, cómputo, recursos de *OpenNebula*, *backup*, monitorización y validador de políticas está controlado mediante alguna política específica para ello. En el Listado 1 se muestra el ejemplo de la política de nombrado para máquinas virtuales, pero también se cubren imágenes y *hooks*; estas son de gran utilidad para administrar estos y más recursos mencionados en la Sección 4.2.1. Se definen políticas de acuerdos de servicio mínimo (*SLA*) que los servicios desplegados en la federación han de respetar. Estas permiten que cada entidad perciba un rendimiento, de los servicios que ofrece la federación, adecuado al nivel de los que tiene ella desplegados. La organización de usuarios en la federación, comentada en la siguiente subsección, está cubierta por políticas respecto a la gestión de usuarios. Más políticas diseñadas y su especificación se pueden encontrar en el Anexo B y su implementación en el Anexo C.

El sistema que valida las políticas diseñadas se plantea en un modo pasivo, es decir, que dicta si se está respetando una política concreta para un recurso presentado, en cuyo caso se deja o no desplegarlo. En un modo activo, el sistema sería capaz de aplicar los cambios necesarios al recurso para que cumpla con las políticas. En el Listado 2 se puede ver en las líneas finales como se aplica este modo de uso. Se ha escogido el modelo pasivo por claridad, simplificar el diseño, viabilidad técnica y disponibilidad de tiempo.

```
Desplegar_recurso := FALSO
Tipo_recurso <- Entrada.accion
Nombre_crudo <- Entrada.nombre_recurso
Entidad, Nombre_usuario, Nombre_recurso <- LLAMAR Separar_cadena para "/"
Nombre_crudo

SI Tipo_recurso == "VM"
    Desplegar_recurso :=
        Entidad == Entrada.metadatos.Nombre_entidad AND
        Nombre_usuario == Entrada.metadatos.Nombre_usuario AND
        Nombre_recurso CONTIENE "A-Za-z"
FIN SI

DEVOLVER Desplegar_recurso
```

Listado 1: Pseudocódigo de política de nombrado.

El sistema de monitorización ofrece la información necesaria para hacer cumplir ciertas políticas definidas. El Listado 2 describe el procedimiento por el cual se interactúa con el punto de acceso a las métricas y con el validador de políticas, obteniendo la información requerida para determinar si un recurso cumple con las políticas que le incumben. En este proceso se consultan y se procesan las métricas exportadas por el servicio de monitorización, que después sirven de entrada para el validador de políticas. El modelo de ejecución programada de eventos que ofrece *OpenNebula* permite ejecutar la lógica personalizada que implementa este flujo de trabajo. El Listado 4 refleja parte de la implementación del pseudocódigo del Listado 2.

Se necesita algún tipo de interacción con los servicios mencionados para este plano, si no, no se podrían validar las políticas implementadas y ejecutar lógica personalizada que permite gestionar trabajos de *backup*. *OpenNebula* ofrece un modelo de ejecución programada de eventos, el *Hook Execution Manager*, que permite suscribir la ejecución de un *script* a eventos del tipo de cambio de estado o llamadas a la API interna.

3.2.3. Plano de uso

En el plano de uso destacan los usuarios y el catálogo de la federación. Los usuarios son nativos de *OpenNebula*, se comparten en la federación y están sujetos a una serie de *ACL*, con permisos sobre imágenes y máquinas virtuales que regulan el uso que hacen de la federación. Además estos usuarios pueden publicar y emplear los recursos ofrecidos en el catálogo de la federación.

El catálogo de la federación es el espacio virtual donde la federación ofrece imágenes de disco para máquinas virtuales y servicios *OpenNebula*, que los usuarios de la federación pueden emplear. *OpenNebula* ofrece un servicio llamado "Marketplace"[2], el cual es accesible desde las zonas de la federación, donde publicar los recursos mencionados ligados a una serie de permisos. Este modelo es el más adecuado dada su integración nativa con *OpenNebula*.

Al crear un usuario, por defecto en *OpenNebula*, se le asigna al grupo de usuarios de la zona en la que se crea. En la organización diseñada, se extienden los permisos mediante

[2]https://docs.opennebula.io/6.10/marketplace/private_marketplaces/overview.html

```
// Trigger
INICIO <- Señal_periodo_monitorización || Señal_creación_recurso
// Contactar con monitorización

FUNCIÓN VALIDAR DEVUELVE BOOL
PARAMETRO ENTRADA Datos_entrada
PARAMETRO ENTRADA política

    Resultado <- VERDADERO
    PARA CADA regla EN política HACER
        SI Datos_entrada no cumple regla
            Resultado <- FALSO
        FIN SI
    FIN PARA
    Devolver Resultado

FIN FUNCIÓN

// Haces comprobaciones de política
Validación_exitosa <- VERDADERO
SI EXISTE Info_recurso
    Info_recurso <- OBTENER información del recurso por RPC XML-API
    PARA CADA política_opennebula que afecta a Info_recurso HACER
         Resultado_validación <- LLAMAR Validar política_opennebula para
Info_recurso
        SI Resultado_validación NO ES VERDADERO
           Validación_exitosa <- FALSO
        FIN SI
    FIN PARA
SINO
    Métricas <- LEER endpoint_monitorización
    PARA CADA política_monitorización
      Resultado_validación <- LLAMAR Validar política_monitorización para
Métricas
      SI Resultado_validación NO ES VERDADERO
       Validación_exitosa <- FALSO
      FIN SI
    FIN PARA
FIN SI

PERMITIR Despliegue de Info_recurso SI Validación_exitosa ES VERDADERO
BLOQUEAR Despliegue de Info_recurso SI NO
```

Listado 2: Pseudocódigo de validación de políticas.

la incorporación a grupos secundarios. Estos grupos adicionales son los sujetos de listas de control de acceso que otorgan privilegios según el propósito de cada grupo. Para encapsular el radio de acción de ciertos grupos se definen cuotas de uso o grupos de recursos.

Por defecto, la figura del administrador es única en la federación de *OpenNebula*. Siguiendo la organización anterior, se definen administradores locales de cada entidad cuyos permisos estarán ligados a su propia entidad.

3.3. Diseño de la Infraestructura

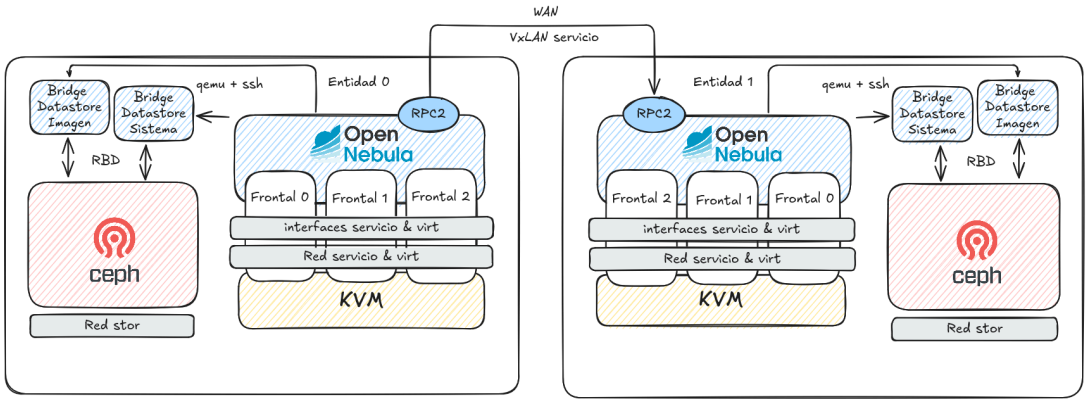

Figura 2: Diseño de la infraestructura multi sede.

La infraestructura desplegada consiste en dos entidades separadas geográficamente, cada una ejecuta su propio software de *OpenNebula* local. Esto constituye un despliegue *cloud* On-Premise en cada entidad. Los núcleos, inicialmente, son máquinas virtuales gestionadas por una interfaz de virtualización local. *OpenNebula* ofrece un mecanismo de adopción por el que máquinas virtuales locales pueden pasar a ser gestionadas por *OpenNebula*. En *OpenNebula* se conoce a este tipo de máquinas como "máquinas salvajes". Adoptando los núcleos como "máquinas salvajes", se consigue tener una infraestructura autocontenida. Cada núcleo se coloca en un servidor distinto, asumiendo que cada entidad cuenta con, al menos, tres servidores de virtualización. Los datastores de *OpenNebula* tienen configurado el driver de Ceph y como servidor puente los nodos de virtualización.

3.3.1. Red

La red interna de cada entidad cumple con políticas de red, lo que permite que los servicios de backup, sistema de validación y catálogo puedan desplegarse de forma transparente en cualquier entidad. Para separar la infraestructura de la federación de cualquier otro servicio con el que ya cuente una entidad, se definen 3 redes distintas: una de servicio, para la comunicación entre servicios de *OpenNebula* y la interacción con los usuarios y otras entidades; administración, donde residen las máquinas físicas; y almacenamiento, independiente del resto y que sirve como punto de entrada al clúster Ceph, el cual cuenta con una red interna de replicación, aunque en otros despliegues pueda ser opcional.

El acceso a la red física desde las máquinas virtuales se consigue mediante las interfaces de red locales de cada nodo de virtualización. Estas deben estar creadas en cada nodo de cada entidad federada; el tipo escogido para estas interfaces es el de conmutación de paquetes. Estas interfaces tienen como entrada el tráfico de alguna de las redes descritas en el primer párrafo, habiendo creado tantas como componentes existen, una para los elementos de Ceph (OSD,

monitores, etc...), otra para los de *OpenNebula* y para las máquinas virtuales y servicios de la federación.

Este diseño permite aislar el tráfico de las máquinas virtuales a una red en un *host*, sin afectar al funcionamiento de la federación. Se configuran en cada nodo de virtualización y las máquinas virtuales ven una interfaz de red ethernet sin alteraciones. En la Sección 4.1.1 puede verse la implementación de la red y en el Anexo H la definición concreta.

El endpoint de cada zona conecta cada entidad de la federación y se comunican a través de sus redes de servicio. Esta comunicación entre endpoints se basa en una solución que permite establecer la pertenencia de una entidad a la federación (VxLAN, VPN o PKI), permitiendo la conexión a través de la red WAN y aislando la comunicación frente a terceros. Se adopta el protocolo TLS, basado en infraestructura de clave pública, para esta comunicación ya que de serie, *OpenNebula* transfiere los datos en texto plano.

3.3.2. Almacenamiento

Cada entidad tiene una serie de discos locales reservados para la federación. Se diferencian dos espacios de discos, uno para explotación (datos calientes) y otro para la recuperación ante desastres (datos fríos).

Cada entidad contará con su propio cluster Ceph. Esta solución de almacenamiento es muy sensible a latencias entre sedes por lo que se recomienda[3] su uso de forma independiente en conexiones por red WAN. Cada clúster seguirá la organización planteada en el diseño de cada componente de la federación (propósito de los pooles definidos, etiquetado de dispositivos y redes internas), pero las reglas internas de gestión son de libre implementación. No obstante, tienen como base las políticas de servicio mínimo implementadas.

En Ceph se almacenan el estado de los núcleos de *OpenNebula*, de las máquinas virtuales y del sistema de monitorización, las imágenes de disco de máquinas virtuales, los backup y los recursos alojados en el *Marketplace*. En el Anexo F.1 se trata en detalle el módulo de almacenamiento implementado.

3.4. Diseño de la Recuperación ante Desastres

El eje central de la recuperación ante desastres es el sistema de *backup*. Se habla del *backup* de la federación más que de sus componentes por separado. Así, se van a tener una serie de políticas que afecten al *backup*, como a cualquier otro servicio, y la infraestructura que le da soporte. Los recursos sobre los que se actúa son: sistemas de ficheros, que incluyen máquinas virtuales, configuraciones, estado de la federación y políticas; y estado de las máquinas virtuales de la federación. Al ser el despliegue de *OpenNebula* autocontenido, la gestión de los *backup* de los servicios de la federación y de los componentes de *OpenNebula* es idéntica.

El diseño del *backup* de los sistemas de ficheros está dirigido por las opciones que ofrece *OpenNebula*. Se emplea un agente nativo que no haga perder disponibilidad del servicio que ofrezca la máquina, si esta se crea desde *OpenNebula*. En caso contrario, se suspende momen-

[3] https://docs.ceph.com/en/squid/radosgw/multisite/#requirements-and-assumptions

táneamente la E/S y se realiza la copia. Los parámetros concretos en *OpenNebula* que ofrecen esta gestión, se detallan en la Sección 4.3.

El servicio de validación de políticas no almacena estado, por lo que para su recuperación en caso de fallo, se almacena la configuración de su despliegue como parte del estado de *OpenNebula*, lo que permite relanzar automáticamente este servicio.

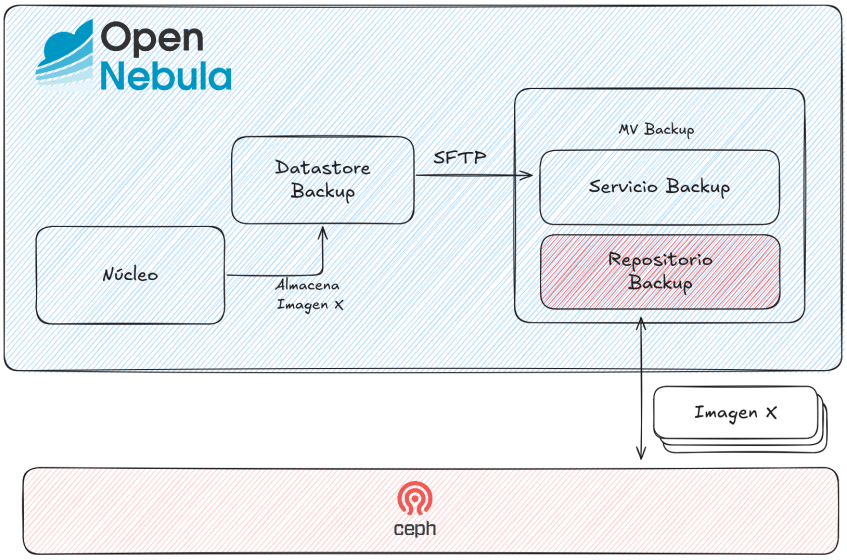

Figura 3: Arquitectura del servicio de backup para cada entidad.

La Figura 3 destaca el uso de un repositorio backup que contacta con Ceph para el almacenamiento de objetos deseado (representados en la figura mediante el cuadro duplicado en el que está escrito "Imagen X"). El repositorio tendrá más implicaciones de red que de almacenamiento *per se*. Al estar comunicado con Ceph, los datos no residirán en la máquina virtual del servicio de backup, si no que lo harán directamente en Ceph. Se emplea el protocolo de comunicación SFTP entre el núcleo de *OpenNebula* y el servicio de backup escogido. Este protocolo es una restricción de *OpenNebula*[4] para conseguir backups nativos; otra opción hubiese sido desaprovechar la capacidad de definición de tareas de backup nativa de *OpenNebula* para emplear un protocolo de almacenamiento de objetos como Ceph S3. Con este diseño, se configura el servicio de backup para que almacene localmente las imágenes de backup de *OpenNebula* y el sistema de ficheros empleado tendrá contacto con Ceph.

Cabe destacar que el término "imagen" en la Figura 3 es el nombre que *OpenNebula* da al resultado de aplicar una estrategia cualquiera de backup sobre la imagen de disco (persistente o no) de una máquina virtual.

3.5. Despliegue

El despliegue de la arquitectura de *OpenNebula* consiste en la instalación, por medio de paquetes software, de los servicios esenciales mencionados en la Sección 2. Con ello, se definen los *datastores* de tipo imagen y sistema con un driver específico de almacenamiento, en este

[4]https://docs.opennebula.io/6.10/management_and_operations/backups/restic.html?highlight=backup#backup-datastore-restic

caso *Ceph*. Por último se incorporan una serie de nodos de virtualización con el hipervisor KVM instalado y cargado como módulo del kernel.

Los núcleos de *OpenNebula* de cada entidad están desplegados formando un clúster en alta disponibilidad. En el plano de gestión, están desplegados en cada zona, el sistema de validación de políticas, el servicio de monitorización y una interfaz que permite la conservación de los recursos alojados en el *Marketplace*, dotando de alta disponibilidad a estos servicios. Así se consigue también la completitud de las métricas ofrecidas. La infraestructura y los servicios que está ofreciendo a la federación son el objeto del servicio de monitorización.

Los recursos siguen un despliegue escalonado (*Rolling Deployment*), se comienza por la entidad de la zona maestra y, tras validar el correcto estado de sus componentes, se sigue por las zonas de las demás entidades. Para cambios o actualizaciones, además de tener en consideración lo anterior para *OpenNebula*, se sigue una estrategia que mantiene la consistencia de los recursos almacenados en el *Marketplace*, que están distribuidos entre todas las entidades.

El despliegue se completa en dos etapas con propósitos distintos, una primera para la infraestructura y después la arquitectura de la federación. Se plantea el despliegue de forma semi-automática, donde cada etapa se completa mediante una herramienta de *IaC* con propósito distinto. Hay ciertos parámetros, como claves de acceso o identificadores de recursos, necesarios en la segunda etapa, que se generan en la primera. De ahí, que el despliegue no pueda ser completamente automático, ciñéndose a estas herramientas.

Durante la primera etapa del despliegue, se declaran varias fases ordenadas por su naturaleza y la relación entre los componentes de la infraestructura. En la primera fase, se prepara el almacenamiento físico, particionando el espacio, y la red, creando las interfaces de red locales. En la segunda fase, se crean las redes virtuales necesarias para comunicar los contenedores Ceph y máquinas virtuales, según la implementación escogida. La siguiente fase corresponde al despliegue del clúster Ceph, que se detalla en el Anexo F.1, subsección de Ceph. Con esto, ya se pueden desplegar los núcleos de *OpenNebula*, ya en la siguiente fase, dotarlos de alta disponibilidad y definir una primera zona maestra. Tras ello, se sigue el mismo procedimiento para la zona esclava y se termina creando los recursos que emplean directamente los servicios de la federación. Esta primera etapa se comenta más en detalle en la Sección 4.4.

En la siguiente etapa de despliegue están definidos los usuarios, grupos y permisos, creados en la primera fase. Con ellos se despliegan el backup, validador de políticas y catálogo de servicios, cada uno en su fase correspondiente. En la Sección 4.4.2 se comenta más en detalle esta etapa.

Capítulo 4.

Implementación

Para implementar el prototipo de federación diseñado se emplean herramientas de despliegue automático, Puppet para la infraestructura y OpenTofu para los servicios de la federación; *Prometheus*, como servicio de monitorización y gestor de métricas personalizadas; el *driver* de *Restic*, con el que llevar a cabo los *backup*; y *Open Policy Agent* (OPA), como sistema de validación de políticas, junto a *Rego* para implementar las políticas. Además se emplea el lenguaje de *scripting Ruby* para definir recursos personalizados en *Puppet* y los *hooks* de *OpenNebula*.

Por cuestiones económicas y materiales, no se contaba con la infraestructura necesaria para desplegar todos los componentes de la infraestructura en distintas máquinas, por lo que se ha optado por simular el entorno real mencionado en el diseño. Este entorno consiste en un único servidor de virtualización, lo que restringe el despliegue real.

El servidor en el que se simula la infraestructura tiene dos procesadores de 16 núcleos, 128GB de memoria RAM y tres discos de 1 TiB de capacidad, dispuestos en RAID5.

4.1. Implementación de la Infraestructura

El despliegue está basado en contenedores *Podman* y dos máquinas virtuales que simulan cada sede geográficamente separada. Sabiendo que el backup y el sistema de control de accesos se despliegan en máquinas virtuales, se ha hecho la infraestructura lo más ligera posible. Cada contenedor se ejecuta en modo no privilegiado, aunque al *gateway* se le asignan las *capabilities*

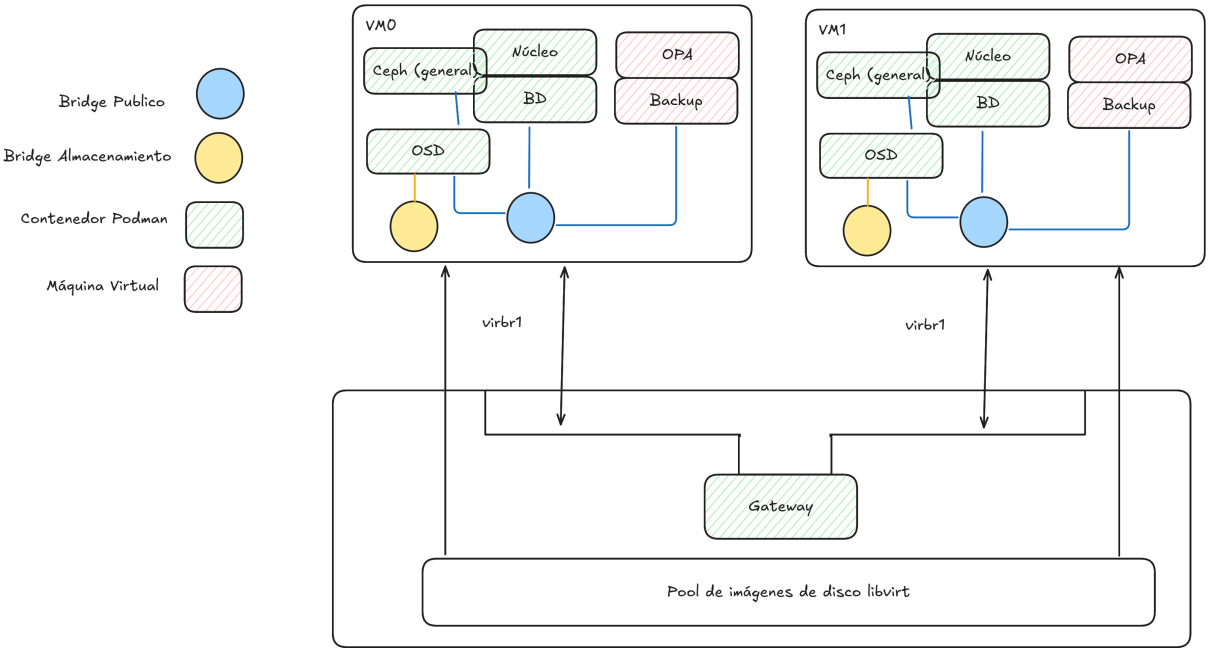

Figura 4: Infraestructura virtualizada

NET_RAW y *NET_ADMIN* que permiten modificar la red. Cualquier acceso a dispositivos físicos, en el caso de OSDs o asegurar la persistencia de las bases de datos, se realiza a través de volúmenes con los dispositivos previamente configurados y montados en una ruta específica en el host.

En el Listado 7 se muestra el despliegue que sigue la federación de *OpenNebula*. La creación de la red, seguida por el *gateway* y los cluster Ceph, son los pilares sobre los que se despliegan los núcleos en alta disponibilidad de la zona maestra, incorporando después la zona esclava, salvando las relaciones de orden entre ellas.

4.1.1. Red

La topología de red presentada en la Figura 4 es una adaptación de la diseñada en la Sección 3.3.1 al entorno de despliegue.

La distinción geográfica entre sedes se consigue haciendo que el tráfico pase por el encaminador más cercano, en este caso el *gateway*. Para ello, cada máquina virtual, que corresponde a una sede distinta, tiene configuradas rutas para su subred local y como puerta de enlace el *gateway*. En el servidor físico están definidas la interfaz tipo bridge para las máquinas virtuales y la red por defecto de contenedores.

Cada máquina virtual cuenta con dos interfaces tipo bridge sobre las que conectar los contenedores. Una de ellas es para el despliegue de los componentes de *OpenNebula*, Ceph y las máquinas virtuales gestionadas por *OpenNebula*. La otra es la red de almacenamiento interno de Ceph. Además, las máquinas virtuales que simulan cada sede están en una misma red de administración, lo cual facilita luego el despliegue.

Se define la red frontal de todo el sistema para que controle el acceso a Internet. Esta se detalla en el Anexo F.2, en el subapartado del *gateway*.

La red de contenedores emplea *macvlan*, redes virtuales que se acoplan a una interfaz física, en este caso los *bridge*. En el Anexo J hay una tabla que refleja los aspectos de red que se tienen en cuenta durante el despliegue.

4.1.2. Almacenamiento

Se ha empleado un *pool* de almacenamiento, definido en *libvirt*, para almacenar las imágenes de disco. Este cuenta con tantas imágenes como necesite cada componente de la infraestructura, y arquitectura. Así, para los OSDs, que en total son seis, se instancian seis imágenes de 20GB, permitiendo el uso de forma más granular del almacenamiento físico. Los espacios de explotación y backup forman parte del clúster Ceph local, pero están diferenciados por el *device class* de sus OSDs que permite asignarlos al pool de backup o explotación.

Los pooles de Ceph definidos están configurados en modo réplica 3 y primario-copia, a excepción del pool de datos del sistema de ficheros para el backup, que emplea la técnica de *erasure code* 3-1. Al tratarse de almacenamiento archivado (datos fríos), 1 bloque de paridad se considera aceptable. Las bases de datos de los núcleos se guardan como imágenes en un pool de bases de datos.

Prometheus almacena sus métricas en el almacenamiento local o, en su defecto, necesita un adaptador en el que almacenar la información y que este se la comunique a la solución de almacenamiento real. Como adaptador se emplea un volumen de contenedor cuyo soporte subyacente es un disco RBD de Ceph. En el pool empleado para las bases de datos hay creada una nueva imagen para este servicio.

OpenNebula consigue almacenar las imágenes de disco de máquinas virtuales mediante el uso de datastores configurados con el driver Ceph. Se emplea el almacenamiento local del nodo de virtualización como "puente" entre la solución de almacenamiento y la instancia de máquina virtual, gestionada por *OpenNebula*. Se puede observar en detalle la definición de estos elementos en el Anexo F.1.

4.2. Implementación de los servicios de la federación

A continuación se abordan los aspectos técnicos más concretos del despliegue de los servicios de validación de políticas, monitorización, catálogo de recursos e interacción con ellos mediante el uso de *hooks* de *OpenNebula*.

4.2.1. Sistema de control de accesos y validación de políticas

Para implementar este sistema se emplea *OPA* (*Open Policy Agent*) [10], que es un validador de políticas expresadas en el lenguaje de dominio específico *Rego*[5]. Dado un conjunto de datos de entrada y un archivo con un política expresada en *Rego*, se valida si el conjunto de datos está conforme con la política.

Este servicio ofrece varias opciones de despliegue, siendo la que mejor se ajusta a *OpenNebula* la opción de contenedores *Docker*[6]. En la Sección 4.4.2 se explica en detalle el despliegue de este servicio.

```
package naming

default vm_allocate := false

vm_allocate if {
    resource_name := split(input.action.resource_name, "/")
    resource_name[0] == input.context.federation_entity
    resource_name[1] == input.context.username
    regex.match(`[A-Za-z]+`, resource_name[2])
}
```

Listado 3: Política de nombrado de máquinas virtuales empleando lenguaje Rego.

Las políticas preliminares implementadas cubren los siguientes aspectos: nomenclatura de recursos, imágenes de disco, máquinas virtuales, etiquetado de recursos y tareas de backup; acuerdos de servicio mínimo, sobrecarga, sobreaprovisionamiento, uptime y endpoints de ser-

[5] https://www.openpolicyagent.org/docs/policy-language
[6] https://www.docker.com/

```ruby
#
# Validación de políticas de nombrado
#

# Preparar entrada para petición a OPA
request_md = naming_policy_md(action, api_info)
opa_input = OPA.create_input(:naming, username, target_entity, metadata =
request_md)

# Resultado tras validar política de nombrado
policy_result, policy_error = OPA.compliance_test(:naming, opa_input, data
= nil)
test_name = action.split(".")[1..].join("_")

# Eliminar el recurso desplegado si no cumple con las políticas
One::delete_resource(resource_type,          resource_name)          unless
policy_result[test_name] or policy_error != nil

#
# Validación de políticas de sobrecarga
#

if action == "one.vm.allocate" or action == "one.image.allocate"
  # Obtención de métricas a través de la función de librería load_metrics
  opa_input = OPA.create_input(:load, username, target_entity, metadata =
load_metrics)

  # Resultado tras validar política de sobrecarga
  policy_result, policy_error = OPA.compliance_test(:load, opa_input, data
= nil)

  # Eliminar el recurso desplegado si no cumple con las políticas
          One::delete_resource(resource_type,     resource_name)     unless
policy_result["compliant"] or policy_error != nil
```

Listado 4: Código Ruby para el control de despliegue tras validación de políticas.

vicios; usuarios de la federación, almacenamiento, compatibilidad técnica, backup y despliegue de aplicaciones finalistas. Estas se encuentran especificadas en el Anexo B.

Las políticas de nomenclatura y sobrecarga se validan mediante el lenguaje *Rego*, visto en el ejemplo del Listado 3, mientras que las demás están implementadas como código del despliegue en Puppet.

En el Listado 4 se resalta una parte de un *hook* implementado, visto en el Anexo E.1, que se ejecuta tras la creación de un recurso en *OpenNebula*, que tenga asociado este script ruby. Esta es la parte de implementación más directa del Listado 2. Destaca el uso de librerías personalizadas, o *middleware*, para la comunicación con *OPA*, la API de *OpenNebula* y el endpoint donde *Prometheus* exporta las métricas de la federación.

4.2.2. Servicio de monitorización

Se emplea *Prometheus*[7] como sistema de monitorización por su integración con *OpenNebula* y porque el tipo de métricas que ofrece permiten implementar las políticas de forma más directa.

Se consiguen cubrir las métricas de la federación, del estado de los servidores de cómputo, el de *OpenNebula*, como servicio, y el almacenamiento de la federación, siguiendo el despliegue mencionado en la Sección 4.4.2. Estas métricas son accesibles desde un mismo *endpoint*. Mediante el uso de métricas personalizadas definidas en *Prometheus*, que se detallan en el Anexo C.3, se implementan algunas políticas de servicio mínimo.

Ceph cuenta con su propio servicio de monitorización, a través de los *managers*, que servirán para consultar el estado del clúster. El driver de Ceph para los datastores de imagen y sistema se encarga de recuperar estas métricas y exponerlas como el estado de los datastores.

4.2.3. Interacción con los servicios

Para ligar los eventos de creación y eliminación de máquinas virtuales, imágenes y trabajos de backup con el sistema de validación se definen dos hooks de *OpenNebula*, ambos son un script Ruby. Uno de ellos interpreta la plantilla en formato XML del recurso en cuestión, valida los parámetros recibidos y el contexto de la operación mediante una petición HTTP a la API REST de OPA y actúa consecuentemente con el resultado obtenido. Opcionalmente, realiza una petición HTTP, con una consulta en PromQL, al endpoint de Prometheus para obtener algunos parámetros que validar posteriormente. En caso de no cumplir la política, el recurso se elimina a través de la interfaz de Ruby que ofrece *OpenNebula*. El otro *hook* actualiza la lista de máquinas a las que aplicar un trabajo de *backup* concreto. La implementación concreta de estos hooks puede encontrarse en el Anexo E.

La otra parte de la definición de un hook es una plantilla que interpreta *OpenNebula*. El Listado 5 muestra una de estas plantillas. En ella se especifica la ubicación del script (*COMMAND*), el evento que inicie su ejecución (*CALL*) y los argumentos para el programa (*ARGUMENTS*). *OpenNebula* ofrece dos tipos de eventos a los que suscribir un hook: cambios de estado (*state hooks*) o peticiones API (*API hooks*). Se suscriben los hooks a las peticiones API de creación (*allocate*) y eliminación (*terminate*), respectivamente. Un mismo script puede ser ejecutado por varias plantillas, mientras no haya dos plantillas que estén suscritas al mismo evento.

```
NAME = naming_vm_creation
TYPE = api
COMMAND = "vm_create.rb"
ARGUMENTS = "$API one.vm.allocate $USER one-10"
CALL = "one.vm.allocate"
```

Listado 5: Plantilla de un hook de OpenNebula.

[7] https://docs.opennebula.io/6.10/management_and_operations/monitor_alert/overview.html

4.2.4. Catálogo de la federación

La arquitectura de almacenamiento, en Ceph [13], escogida para la replicación y visión unificada del contenido del *Marketplace* es la "Multi-zona"[8]. Esta consiste en un reino (*realm*), que es el dominio de acción de la federación, con un *zonegroup* (o región) que cuenta con dos zonas: maestra y secundaria. Hay una zona en cada clúster de las entidades federadas y a las que responden los Rados Gateway (RGW), servicios de Ceph que exponen una interfaz compatible con S3. Hay diferentes tipos de usuarios, uno encargado de la replicación entre zonas, con permisos de administrador, y otro con el que *OpenNebula* accede al almacenamiento del *Marketplace*. Este último es el propietario del *bucket* donde se almacenan los objetos del *Marketplace* privado de la federación. Se puede seguir la definición de estos recursos en el Anexo F.1, en la subsección de *Scripts* y Anexo I, en el apartado del *Marketplace*.

Hay un servicio balanceador que distribuye la carga entre las zonas y que reside en una red alcanzable por todos los nodos de la entidad, la red frontal. Se emplea el software *HAProxy* para este servicio y donde se define un *frontend* cuyo único *backend* ejerce una política de *Round Robin* entre los *RGW*. Con esta arquitectura se consigue la disponibilidad del dato desde cualquier entidad de la federación y tolerancia al fallo de alguna de las zonas dentro del *zonegroup*.

4.3. Implementación de la Recuperación ante Desastres

La implementación se enfoca en las dos necesidades planteadas en el diseño: servicio de backup y repositorio que se comunique con Ceph. Para este servicio se usa *Restic*, el cual cuenta con integración nativa en *OpenNebula* e implementa las necesidades de backup básicas (deduplicación, versionado y encriptado). Para la incorporación del driver a *OpenNebula*, se tiene que crear un nuevo *datastore* de tipo *BACKUP_DS*. Ya que el driver de *Restic* en *OpenNebula* todavía no implementa la comunicación mediante la interfaz S3, se despliega una máquina virtual en la que se monta un sistema de ficheros compartido CephFS. El punto de montaje es un directorio que es accedido por el driver de *Restic* de *OpenNebula*, para ello se emplea el protocolo de red SFTP.

En la puesta en marcha del servicio de backup, se instalan los paquetes *rsync* y *qemu-img*, requeridos por el driver *Restic*; se sigue la política de nomenclatura de máquinas virtuales y etiquetado; y se distribuyen las claves públicas *ssh* de los tres núcleos a la máquina virtual. La definición de la máquina virtual, usuario y *datastore* pueden encontrarse en el manifiesto *OpenTofu* de la Sección I.

Las tareas definidas, tienen en cuenta las políticas de backup. Para cumplir con ellas, se especifican los modos de agente (atributo *FS_FREEZE="AGENT"*) o suspensión (atributo *FS_FREEZE="SUSPEND"*) según la naturaleza de la máquina virtual. Los backup se realizan de manera incremental usando como base el *snapshot* cada cuatro repeticiones. Las copias intermedias emplean la técnica de *Copy On Write* (*CoW*). Siguiendo este método, se guarda también el estado de la máquina virtual de forma consistente en caso de fallo. En el Listado 6

[8] https://docs.ceph.com/en/quincy/radosgw/multisite/#varieties-of-multi-site-configuration

```
NAME = "Servicios básicos"

BACKUP_VMS   = "0, 1, 2"
DATASTORE_ID = 102

FS_FREEZE = "SUSPEND"
KEEP_LAST = "4"
MODE      = "INCREMENT"

PRIORITY  = 90 # prioridad alta
EXECUTION = "SEQUENTIAL" # requerido por Restic

SCHED_ACTION = [
    REPEAT="0", # repetir cada semana
    DAYS="1,5", # lunes, miércoles y viernes
    END_TYPE="0", # tarea permanente
    TIME="18000" ] # 5 AM
```

Listado 6: Definición de la tarea de backup one-10-90 en OpenNebula

se ve la aplicación de una tarea para los servicios básicos de *OpenNebula* (tres núcleos en HA con sus bases de datos).

Para la incorporación de nuevos servicios al sistema de backup, se define un nuevo *hook*. Este leerá la etiqueta de la máquina virtual para determinar su prioridad y asignará su identificador a la tarea que le corresponda según la prioridad. Habrá igualmente otro *hook* definido para el borrado de máquinas donde se eliminará su identificador de la tarea. Ambos pueden encontrarse en el Anexo E.

4.4. Despliegue

4.4.1. Infraestructura

Se opta por la herramienta de despliegue automático Puppet [11] porque permite definir las relaciones entre recursos de manera más granular y la naturaleza del problema permite instalar en cada servidor del despliegue un agente Puppet. Este tipo de herramientas permiten modelizar los recursos de un sistema y automatizar el aprovisionamiento de servidores, sin modificar las características previamente configuradas.

En Puppet se definen los módulos *storage* y *virt* para el despliegue de cada parte de la infraestructura por separado. La relación de despliegue entre estos módulos se rige por la que se ha diseñado en la Sección 3.5. Se ha empleado Bolt [14] para el despliegue distribuido, visto en el Listado 7, de los componentes de la infraestructura y Vagrant [15] para el aprovisionamiento y despliegue de las máquinas virtuales que simulan cada sede. El detalle más fino y el uso de recursos personalizados de Puppet de estos módulos se comenta en el Anexo F.

```
plan cloud_fed::deploy_federation(){
    $targets = get_targets('fed_deploy')
    $scripts_path = lookup('deployment_paths::scripts')
    $templates_path = lookup('deployment_paths::file_templates')


    upload_file('virt/scripts',
        "$scripts_path/virt",
        $targets,
        _run_as => 'root'
    )
    upload_file('storage/scripts',
        "$scripts_path/storage",
        $targets,
        _run_as => 'root'
    )
    upload_file('virt/file_templates/opennebula.repo',
        "$templates_path",
        $targets,
        _run_as => 'root'
    )
    run_plan('facts', 'targets' => $targets)
    $plan_result = catch_errors() || {
        run_plan('cloud_fed::net_ifaces', 'targets' => $targets)
        run_plan('cloud_fed::storage_ceph', 'targets' => $targets)

        run_plan('cloud_fed::nebula_instances')

        run_plan('cloud_fed::nebula_federation',
            'master' => lookup('master_id'),
            'slaves' => lookup('slaves_ids')
        )

        run_plan('cloud_fed::slaves_ha', 'targets' => get_targets('slaves'))

        run_plan('cloud_fed::monitoring', 'targets' => $targets)

    }

    return {
        result => $plan_result
    }
}
```

Listado 7: Despliegue, en Puppet Bolt*, de la red, el clúster Ceph y la federación* OpenNebula.

4.4.2. Servicios de la federación

Se emplea *OpenTofu* [12] para la segunda etapa del despliegue ya que se están desplegando recursos que gestiona *OpenNebula*. OpenTofu guarda en un fichero el estado del despliegue de los recursos, lo que permite no duplicarlos, actualizarlos cuando se modifican o eliminarlos si se desea. Hay que tener cuidado con el borrado de este fichero de estado, ya que el despliegue de futuros recursos sería inconsistente y podría afectar al estado de los servicios existentes.

En el Anexo I sigue de forma ordenada el despliegue de los recursos que se van a presentar. En una primera fase, se crean los usuarios y grupos de cada zona con los que luego se despliegan el resto de recursos. En esta fase se emplea el usuario administrador de la federación, *oneadmin*.

Se crea una imagen de disco, *cloud init*, provista a través del paquete de contextualización[9] de *OpenNebula* e instalado el sistema operativo *Debian*. Esta imagen se crea como no persistente, de forma que pueda ser utilizada por las máquinas virtuales en las que se despliegan el *backup* y *OPA*, como imágenes volátiles.

Al iniciar la máquina se ejecuta un script que instala los paquetes necesarios y monta el sistema de ficheros compartidos que expone las políticas. En esta máquina virtual se instala *Docker* y se despliega *OPA*. En el anexo Sección D.2 se detalla un problema encontrado con este despliegue.

Se define un nuevo sistema de ficheros en Ceph, *policies*, montado en la máquina virtual para que el contenedor pueda acceder a las políticas mediante un volumen. Este actúa como repositorio, comunicando los objetos a guardar con Ceph, a tráves de la red. El contenedor se despliega localmente con exposición en el entorno físico-virtual de la máquina desde el puerto *2345*.

Para la monitorización de la infraestructura, se despliega un contenedor separado que ejecuta el software de Prometheus, este ofrece el *endpoint* en el que se exponen las métricas. Los exportadores de métricas se distribuyen como paquetes de software desde el repositorio oficial de *OpenNebula*; estos instalan agentes de exportación de métricas en los nodos de virtualización y en los núcleos y datastores de *OpenNebula*.

El manifiesto del *Marketplace* de la federación define los permisos de uso a todos los usuarios de la federación, gestión solo al grupo que pertenezca el recurso y administración al propietario. En este manifiesto se declara también la conexión, mediante el protocolo S3, con el balanceador de la red frontal.

[9] https://docs.opennebula.io/6.10/management_and_operations/guest_os/kvm_contextualization.html

Capítulo 5.

Validación y Pruebas realizadas

5.1. Validación

Existen recursos que tienen dependencias que no se pueden expresar mediante las herramientas de despliegue automático utilizadas. Por lo general, las herramientas de despliegue automático emplean lenguajes declarativos y no pueden gestionar dependencias generadas en tiempo de ejecución, de forma nativa (e.g. Esperar a que un recurso alcance un estado concreto). Mediante la ejecución de lógica personalizada, que sirve como barrera para confirmar el estado deseado del clúster, se previene el despliegue prematuro de estos recursos. Así, se valida que el comportamiento final de los componentes es el esperado.

Las relaciones de orden para la configuración de los nodos en alta disponibilidad, establecer la federación y la "Multi-zona" de Ceph son estrictas e involucran mezclar distintos tipos de recursos. Por este motivo, se valida el despliegue automático de la infraestructura, generando el grafo de dependencias que ofrece Puppet, para comprobar el orden en el que se despliegan los recursos. El flujo de *logs* generados por el líder de cada zona de la federación sirve para detectar errores en la configuración o fallos en el despliegue. Creando usuarios en *OpenNebula*, en cualquiera de las zonas, se comprueba que la replicación interna funciona si se pueden listar los usuarios desde ambas zonas. Para Ceph, se controla el estado del clúster en todo momento, consiguiendo determinar que el número de OSDs requeridos para los sistemas de ficheros y la interfaz S3 es de 5, si no, los *placement group* quedaban desprovistos de los OSDs necesarios.

Las políticas implementadas con el lenguaje *Rego* se validan en el entorno controlado[10] de pruebas que ofrece la propia herramienta. Se comprueba como con distintos valores de la entrada, la salida variaba según el comportamiento esperado.

5.2. Pruebas

A continuación se presentan dos pruebas generales que comprueban el correcto funcionamiento del sistema implementado. Previo a ellas, se completan una serie de pruebas. La primera comprueba que la infraestructura se despliegue correctamente, para ello se prueba que los paquetes de red llegan a los distintos componentes dentro de la misma federación y que los núcleos de *OpenNebula* se encuentran en un estado estable. Se sigue por comprobar que se consiguen desplegar las máquinas virtuales definidas en los manifiestos *OpenTofu*. Más tarde se prueba mediante el *script* que utilizan los *hooks*, a modo de test, que se obtiene la respuesta esperada del *endpoint* de *OPA* y que se procedería a la eliminación del recurso desplegado, en el caso correspondiente.

[10] https://play.openpolicyagent.org/

5.2.1. Prueba de sobrecarga

Esta prueba involucra la recolección de métricas del servicio de monitorización, la aplicación de políticas de SLA, sobrecarga en este caso, y su interacción con el subsistema de hooks de *OpenNebula*. Además, fuerza la portabilidad de los servicios en la federación, para su despliegue inmediato en cualquier entidad mediante el uso de *IaC*. Esto permite validar posibles políticas de infraestructura, aquellas que se escapen del alcance del servicio de validación de políticas, y la organización de usuarios en la federación, permitiendo un acceso más restringido a usuarios de zonas ajenas a la que despliega el recurso.

Se despliega una máquina virtual en una zona de *OpenNebula*, ejecutando tareas de cómputo exigentes, tales que lleguen a superar el 80% de utilización de *vCPU*. Entonces, se prueba que se impida el despliegue de una nueva máquina virtual. Este efecto debe ser visible solo para usuarios de una zona diferente a donde se está desplegando el recurso, diferenciando a nivel organizativo usuarios de entidades distintas.

Para llevar a cabo esta prueba, primero se despliega una máquina virtual con imagen base no persistente sobre la que se instala el software *cpuburn*. Se usa esta nueva imagen en la plantilla de máquina virtual que después se despliega hasta conseguir superar el 80% de utilización de *vCPU*.

Se ha visto como la herramienta *cpuburn*[11] es muy agresiva y rápidamente ha aumentado la carga del nodo KVM, alcanzando el porcentaje objetivo en menos tiempo que el periodo de recolecta de métricas. Por ello, se ha reconfigurado el periodo inicial, rebajándolo a 2 segundos.

Aún así, la aplicación de la política ha sido correcta y se ha impedido la creación de una nueva máquina virtual, probando el correcto funcionamiento del sistema de monitorización y la validación de políticas.

5.2.2. Prueba de backup

Para probar el despliegue del backup se va lanza una tarea en *OpenNebula* que almacena el estado de una de las bases de datos de uno de los núcleos. Dado que la base de datos utilizada (MariaDB) ya cuenta con un plan de backup, se va a hacer la copia del contenido de la máquina virtual que la almacena. Para esto, en el entorno de simulación se despliega una nueva máquina virtual que actuará como el cuarto núcleo de *OpenNebula* para la zona *maestra*.

La ejecución del plan ha congelado momentáneamente el sistema de ficheros, hecho probado mediante la ejecución de un script a modo de agente, que lista el directorio raíz y manda una baliza a un puerto escuchando en otra máquina. Tras haberse ejecutado el plan definido, la ocupación del datastore ha aumentado, según muestra la monitorización del mismo ofrecida por *OpenNebula*. Este aumento ha sido de un 40% menos que la imagen original, debido a la compresión. También ha crecido la ocupación de los OSD en el pool de backup y se ha repartido la carga de forma homogénea entre los OSD con device class "backup". Entre los tres OSD suman una carga ligeramente mayor que el tamaño de la copia, debido a los bloques de paridad. Esto enseña el uso de la técnica erasure code en el pool.

[11] https://patrickmn.com/projects/cpuburn/

Seguidamente, se ha eliminado la máquina virtual y se ha recuperado del backup, lo que ha conllevado la replicación entre núcleos, visto en los logs del núcleo recuperado.

Capítulo 6.

Conclusiones y Trabajo futuro

Se ha conseguido el objetivo de desplegar el prototipo de una federación entre dos instancias de *OpenNebula*, que sirva como primer paso para un posterior despliegue real con dos entidades en zonas geográficamente separadas. El correcto despliegue de los componentes de *OpenNebula* y Ceph, ha probado la validez de la infraestructura simulada. Pese a ello, se han encontrado una serie de problemas que han dificultado el despliegue de la infraestructura inicial, pero se han resuelto satisfactoriamente. El uso de los manifiestos de despliegue automático ayudará en parte al despliegue real.

Las pruebas han demostrado que los servicios principales funcionan como se esperaba, que la infraestructura soporta la traslación de recursos virtuales entre zonas de *OpenNebula* y las políticas implementadas aseguran el uso adecuado de la federación.

La red frontal diseñada ha quedado desprotegida, en el contexto de la seguridad. Por ello, habría que considerar la incorporación de un router perimetral que actúe como un firewall. Estaría controlado por software y permitiría cumplir políticas de enrutado y control de acceso, independientemente del hardware utilizado. Un aspecto de relevancia para ciertos tipos de despliegue, sería la definición de "servicios" de *OpenNebula* que permiten desplegar conjuntos de máquinas virtuales con una relación concreta entre ellas. Para ello, se deberán habilitar los servicios de OneFlow y OneGate. Quedan abiertos al diseño otros aspectos como la seguridad, modelado de usuarios más complejo y migración en caliente de máquinas virtuales entre sedes.

Personalmente, considero que este proyecto ha sido un importante motor de conocimiento para proyectar los conceptos aprendidos durante la carrera. No solo me ha permitido conocer en profundidad las herramientas utilizadas, los algoritmos que las hacen funcionar y los conceptos en los que se basan, si no que también entender porqué herramientas de software libre permiten garantizar la soberanía de la información que uno genera.

Bibliografía

[1] «OpenNebula Documentation». [En línea]. Disponible en: https://docs.opennebula.io/6.10/

[2] S. Weil y et al., «Ceph: A Scalable, High-Performance Distributed File System», 2004, *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04)*. [En línea]. Disponible en: https://ceph.io/assets/pdfs/weil-ceph-osdi06.pdf

[3] C. Lee, R. Boh, y M. Michael, «The NIST Cloud Federation Reference Architecture», 1 de febrero de 2020, *National Institute of Standards and Technology, Gaithersburg, MD*. [En línea]. Disponible en: https://doi.org/10.6028/NIST.SP.500-332

[4] M. Sporny, D. Longley, D. Chadwick, y I. Herman, «Verifiable Credentials Data Model v2.0», 15 de mayo de 2025, *World Wide Web Consortium (W3C)*. [En línea]. Disponible en: https://www.w3.org/TR/vc-data-model-2.0

[5] «Ceph Storage Cluster Documentation». [En línea]. Disponible en: https://docs.ceph.com/en/reef/architecture/

[6] «Ceph Object Gateway Documentation». [En línea]. Disponible en: https://docs.ceph.com/en/latest/radosgw/#object-gateway

[7] «Ceph File System Documentation». [En línea]. Disponible en: https://docs.ceph.com/en/latest/cephfs/#ceph-file-system

[8] «Gaia-X: A Federated Secure Data Infrastructure». [En línea]. Disponible en: https://gaia-x.eu/about/

[9] E. Rissanen, Ed., «eXtensible Access Control Markup Language (XACML) Version 3.0.», 22 de enero de 2013, *OASIS Standard*. [En línea]. Disponible en: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[10] «Open Policy Agent Documentation». [En línea]. Disponible en: https://www.openpolicyagent.org/docs

[11] «Puppet Documentation». [En línea]. Disponible en: https://www.puppet.com/docs/index.html

[12] «OpenTofu Documentation». [En línea]. Disponible en: https://opentofu.org/

[13] «Ceph Documentation». [En línea]. Disponible en: https://docs.ceph.com/en/latest/

[14] «Puppet Bolt Documentation». [En línea]. Disponible en: https://help.puppet.com/bolt/current/topics/bolt.htm

[15] «Vagrant libvirt plugin Documentation». [En línea]. Disponible en: https://vagrant-libvirt.github.io/vagrant-libvirt/

[16] «Parámetro host-passthrough». [En línea]. Disponible en: https://forum.opennebula.io/t/cpu-host-passthrough-and-scheduling-vms/3349

[17] «Esquema XML OpenNebula.». [En línea]. Disponible en: https://forum.opennebula.io/t/one-vm-updateconf-invalid-raw-section-cannot-validate-data-with-domain-rng--schema/9866

Anexo A

Diagrama de Gantt

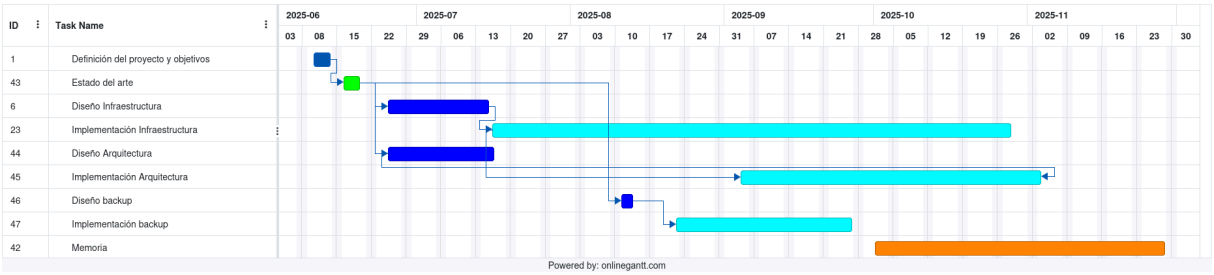

Figura 5: Diagrama de gantt.

Anexo B

Especificación de las políticas

A continuación se especifican todas las políticas que se han implementado usando *Rego*, las métricas de *Prometheus* y otros métodos.

B.1 Nomenclatura

Todas los nombres tendrán como prefijo común el nombre de la entidad. Se emplea una barra lateral (/) como separador. El Listado 3 muestra un ejemplo de una política de nombrado implementada usando el lenguaje *Rego*.

1 **Nombrado de etiquetas:** <nombre-etiqueta> (ej. one-10/prod)
2 **Nombrado de máquinas virtuales:** <nombre-usuario>/<nombre-maquina> (ej. one-10/user10/ntp)
3 **Nombrado de imágenes:** <nombre-usuario>/<nombre-imagen> (ej. one-10/user10/Ubuntu)
4 **Nombrado de tareas de backup:** <prioridad> (ej. one-10/90)
5 **Nombrado de *hooks*:** hook/<nombre-hook> (ej. one-10/hook/update-backupjob)

B.2 Servicio mínimo

1 **Sistema de monitorización:** Cada entidad contará con un sistema que monitorice la actividad local de *OpenNebula*.
   i **Punto de enlace**: Definir una IP virtual donde establecer la comunicación de la federación.
   ii **Exposición de métricas:** Las métricas se expondrán en formato *JSON* accesibles desde el punto de enlace definido en la ruta */metrics* y su esquema en */metrics/schema* que deberá coincidir con el esquema esperado por la federación.
   iii **Periodo de monitorización:** Se establecerá una frecuencia de monitorización de 30 segundos.
   iv **Control de acceso:** Habrá un grupo de usuarios con permisos, exclusivamente, de consulta de métricas llamado "fed-exporter".
2 ***Uptime* (tiempo en línea) de servicios:** 99% de tiempo en línea requerido para cada uno de los servicios etiquetados como producción. Se comprobará que el *LCM STATE* tenga el valor *ACTIVE*
3 **Definición de downtime o caída:** Un servicio se considera caído si no se exportan sus métricas durante un periodo entero y el estado de la máquina virtual en la que se ejecuta en *OpenNebula* es *ERROR*, *POWEROFF* o no existe. En el caso de que la máquina esté activa (estado *ACTIVE*) se mirará el estado de su ciclo de vida o *LCM STATE* y comprobará que está en un estado diferente a *ACTIVE*.
4 **Sobreaprovisionamiento:**

i **Proporción:** No se podrán aprovisionar más *CPU* reales que *vCPU*.
ii **Rango de sobreaprovisionamiento:** El rango por defecto será de hasta 2:1 (2 *vCPU* por cada *CPU* real).
iii **Memoria de respaldo:** La cantidad de *swap* presente en los nodos KVM deberá soportar la carga de sobreaprovisionamiento de memoria asignada en un momento dado.
iv **Etiquetado:** Para incrementar la capacidad de sobreaprovisionar una máquina virtual, se deberá etiquetar como "comp" o "storage" para poder sobreaprovisionar la máquina por encima del por defecto, impidiendo que servicios que no lo necesiten, hagan un uso indebido (política de *opt-in*). El máximo establecido es de 6:1.

5 **Sobrecarga:** Los porcentajes presentados responden a la intuición que se tiene de las capacidades del sistema previsto, la realidad dependerá de la robustez de cada uno de los miembros.
i Una entidad no podrá desplegar más máquinas virtuales si la utilización de *vCPU* está por encima del 80%.
ii Una entidad no podrá desplegar más máquinas virtuales cuyas imágenes sean persistentes si el almacenamiento supera el 90%.

B.3 Backup

1 **Tipo de backup:** Se realizarán backups completos cada 4 incrementales.
2 **Modo de backup:** Si el almacenamiento de un servicio se basa en bloque (bases de datos), se escogerá el modo *CBT* de *OpenNebula* para seguir cambios a nivel de bloque. Si un servicio solo interactúa con el sistema de ficheros, se escogerá el modo *snapshot* con diferenciales de *Copy On Write*.
3 **Proceso de backup:** Si un servicio cuenta con solución de backup integrada, se deberá activar dicha solución antes de empezar el backup desde *OpenNebula*.

B.4 Almacenamiento

1 **Uso de imágenes:** El uso de imágenes persistentes estará restringido a máquinas que solo tengan desplegada una instancia y no haya una imagen persistente ya creada en al que basarse. Se deberá hacer uso de imágenes no persistentes en cualquier otro caso.
2 **Marketplace:**
i Las imágenes publicadas tendrán instalados el paquete de contextualización de *OpenNebula*.
ii El usuario *oneadmin* no podrá publicar imágenes.
3 **Ceph:**
i Los pooles de metadatos tendrán una regla CRUSH que los asigne a OSDs con los discos de menor latencia.
ii El pool de backup tendrá asignados los OSD con device class *backup*.

B.5 Usuarios y grupos

1 **Administradores:** El usuario *oneadmin* es global a la federación y solo tendrá permisos sobre las máquinas desplegadas para el backup. En cada entidad habrá un administrador que tendrá un uso privilegiado sobre su zona y pertenecerá al grupo de administradores de la federación.
2 **Uso de imágenes:** Cada usuario tendrá como grupo primario el de su zona y el administrador de zona podrá añadirlo a los grupos de la federación oportunos.

B.6 Políticas de aplicación

1 **Gateways de acceso a Internet:** Toda aplicación debe tener tener conexión a Internet a través de alguna de las puertas de enlace definidas para ese propósito dentro de la federación.
2 **Etiquetado:** Toda aplicación debe estar correctamente etiquetada. Deberá distinguirse si corresponde a un servicio en producción, en desarrollo o pruebas. Para ello emplear los nombres "prod", "dev" y "test".
3 **Replicación mínima:** Las aplicaciones etiquetadas como producción, deberán tener, al menos, dos instancias en ejecución en entidades federadas distintas.

B.7 Compatibilidad técnica

Cada entidad debe presentar una instancia de *OpenNebula* como software de gestión de máquinas virtuales y, opcionalmente, Ceph como sistema de almacenamiento. En su defecto, el diseño del almacenamiento debe ser similar al planteado en la fase de diseño. Se ha de emplear *KVM* como hipervisor y *Restic*[12] como driver del datastore de backup. El resto de servicios pueden tener una implementación diferente a la presentada en los siguientes apartados.

[12]https://docs.opennebula.io/6.10/management_and_operations/backups/restic.html?highlight=backup

Anexo C

Implementación de políticas

A continuación se muestran las dos políticas implementadas usando el lenguaje *Rego*. La primera implementa las políticas de nombrado de máquinas virtuales, imágenes *OpenNebula*, *hooks* y trabajos de *backup*. La segunda detalla la política de sobrecarga, que forma parte de las políticas de acuerdo de servicio mínimo.

C.1 Política de nombrado

En el código, se obtiene de la entrada (variable *input*) el contexto en el que se está desplegando cada recurso, esto es, el nombre de la entidad y nombre de usuario. En el campo *action* se definen el nombre del recurso dado por el usuario (*resource_name*), el recurso al que afecta la acción y, opcionalmente, la etiquetea (*label*) que tiene asignada. Primero se comprueba que la acción corresponda a algún recurso de los definidos, seguidamente se aplican las reglas establecidas en el Anexo B.

```
  package naming

default vm_allocate := false

vm_allocate if {
    input.action.type == "one.vm.allocate"
    resource_name := split(input.action.resource_name, "/")
    resource_name[0] == input.context.federation_entity
    resource_name[1] == input.context.username
    regex.match(`[A-Za-z]+`, resource_name[2])
}

image_allocate if {
    input.action.type == "one.image.allocate"
    resource_name := split(input.action.resource_name, "/")
    resource_name[0] == input.context.federation_entity
    resource_name[1] == input.context.username
    regex.match(`[A-Za-z]+`, resource_name[2])
}

hook_allocate if {
    resource_name := split(input.action.resource_name, "/")
    resource_name[0] == input.context.federation_entity
    resource_name[1] == "hook"
    regex.match(`[a-zA-Z0-9-_]+`, resource_name[2])
}

backupjob_allocate if {
    resource_name := split(input.action.resource_name, "/")
```

```
    resource_name[0] == input.context.federation_entity
    regex.match(`[0-9]+`, resource_name[1])
}
```

C.2 Política de sobrecarga

Aquí, la variable de entrada solo contiene el campo de acción (*action*) cuyos campos se relacionan directamente con las métricas obtenidas desde *Prometheus*.

```
  package sla


default compliant := false


mem_compliant if input.action.host_mem_allocation_ratio < 0.8

ds_compliant          if          input.action.datastore_free_space          /
input.action.datastore_total_capacity > 0.1

compliant if {
  mem_compliant
  ds_compliant
}
```

C.3 Políticas de *uptime* y *downtime*

```
groups:
  - name: OneVM
    rules:
      - alert: UptimeBelowPolicy
        expr: (
          sum_over_time(opennebula_vm_lcm_state == 3[30d])
          / count_over_time(opennebula_vm_lcm_state[30d])
        ) < 0.99
        for: 1h
        labels:
          severity: warning
        annotations:
          summary: "Uptime bajo para {{ $labels.service }}"
           description: "El servicio {{ $labels.service }} tiene un uptime
inferior al 99% en los últimos 30 días."
      - alert: DowntimeAbovePolicy
        expr: (
          opennebula_vm_state == 8 or opennebula_vm_lcm_state != 3
        )
        for: 1h
        labels:
          severity: warning
        annotations:
          summary: "Downtime elevado para {{ $labels.service }}"
          description: "El servicio {{ $labels.service }} tiene un downtime
superior al acordado."
```

Listado 8: Grupo de reglas Prometheus que detectan las políticas de uptime y downtime.

En el Listado 8 se muestra un ejemplo de implementación de las políticas de uptime y downtime mediante grupos de reglas de Prometheus. La naturaleza de estas políticas invitan a enviar alertas a la federación más que esperar a que suceda algún evento para comprobar si el recurso cumple las políticas. Así, se consigue monitorizar el estado de los servicios de backup y validador de políticas.

Anexo D

Problemas encontrados

D.1 Despliegue en contenedores de Ceph

Ceph está desplegado en contenedores ya que involucra la definición de un número *a priori* indefinido de OSD. No se contempla la posibilidad de definir nodos LXC en *OpenNebula* en esta versión temprana del proyecto, principalmente por la limitación de recursos de los que se dispone (se necesitaría otro servidor de virtualización). Al emplear la tecnología de contenedores, se tiene que hacer uso de un orquestador o una herramienta de despliegue automático que gestione los contenedores.

D.2 Contenedores en máquina virtual

Se ha de habilitar el modo de CPU *host-passthrough* [16], [17] activo para permitir el uso de contenedores en una máquina virtual. Para ello, se ha desactivado el atributo restringido de *OpenNebula VM_RESTRICTED_ARGS="RAW/DATA"*. Se han definido también los siguientes ACL para que los usuarios *backup* y *policies* puedan instanciar sus máquinas virtuales:

oneacl create "@115 VM+NET+IMAGE+TEMPLATE/ CREATE+USE+MANAGE+ADMIN"*

oneacl create "@114 VM+NET+IMAGE+TEMPLATE/ CREATE+USE+MANAGE+ADMIN"*

Anexo E

Implementación del modelo de ejecución de eventos

Todos los *hooks* definidos en *OpenNebula* tienen prefijadas con unas líneas que definen la ubicación de las librerías de ruby que usa *OpenNebula*.

```
  ONE_LOCATION = ENV["ONE_LOCATION"]

if !ONE_LOCATION
  RUBY_LIB_LOCATION = "/usr/lib/one/ruby"
  GEMS_LOCATION = "/usr/share/one/gems"
else
  RUBY_LIB_LOCATION = ONE_LOCATION + "/lib/ruby"
  GEMS_LOCATION = ONE_LOCATION + "/share/gems"
end

if File.directory?(GEMS_LOCATION)
  Gem.use_paths(GEMS_LOCATION)
end

$LOAD_PATH << RUBY_LIB_LOCATION
# carga librerías personalizadas
$LOAD_PATH << "/usr/share/one/hooks-lib"
```

E.1 Hook creación de recursos

Este código es el referenciado en varias partes de la memoria. Se interpreta la plantilla XML que *OpenNebula* pasa como parámetro a cualquier *hook*, codificada en base 64. Se validan las políticas de nombrado y de sobrecarga. Para el recurso de trabajos de *backup*, se añade el identificador del recurso presentado a una la lista de máquinas virtuales en los trabajos de *backup* principales. `#!/usr/bin/env ruby`

```
require "base64"
require "nokogiri"
require "opa"
require "policies/naming" # naming_policy_md
require "policies/load" # load_metrics
require "policies/backupjob" # rename_backupjob
require "nebula" # delete_resource
require "rpcapi/parser"

api_info = Nokogiri::XML(Base64::decode64(ARGV[0]))
```

```
action = ARGV[1]
username = ARGV[2]
fed_entity = ARGV[3]

NEBULA_ENDPOINT = "http://192.168.10.10:2633/RPC2"
One::set_endpoint NEBULA_ENDPOINT

success = api_info.xpath("/CALL_INFO/RESULT").text.to_i == 1

if !success
  puts "Resource wasn't created, hook not executed"
  exit 0
end

resource_type = action.split(".")[1]
resource_name = parse_template(api_info)[:"resource_name"]
target_entity = {
  "username" => username,
  "federation_entity" => fed_entity,
}

#
# Ejecutar políticas de nombrado
#
request_md = naming_policy_md(action, api_info)
opa_input  =  OPA.create_input(:naming,  username,  target_entity,  metadata  =
request_md)
# Contact OPA's appropriate endpoint
policy_result,  policy_error  =  OPA.compliance_test(:naming,  opa_input,  data  =
nil)
test_name = action.split(".")[1..].join("_")
One::delete_resource(resource_type,            resource_name)           unless
policy_result[test_name] or policy_error != nil

#
# Ejecutar políticas de sobrecarga
#

if action == "one.vm.allocate" or action == "one.image.allocate"
  # Obtener métricas y generar entrada
    opa_input  =  OPA.create_input(:load,  username,  target_entity,  metadata  =
load_metrics)

  # Evaluar política
 policy_result, policy_error = OPA.compliance_test(:load, opa_input, data = nil)
```

```
  # Eliminar recurso si no cumple
          One::delete_resource(resource_type,      resource_name)      unless
policy_result["compliant"] or policy_error != nil
elsif action == "one.backupjob.allocate"
  # Renombrar trabajo de backup
  update_backupjob(:insert, api_info)
end
```

E.2 Hook eliminación de recursos

En esta ocasión, solo hay que eliminar de la lista de máquinas virtuales del trabajo de backup correspondiente al recurso que se quiere eliminar.

```
  #!/usr/bin/env ruby

require "base64"
require "nokogiri"
require "rpcapi/parser"
require "policies/backupjob"

#
# Al crear una MV, incluir el id en la tarea correspondiente
#

api_info = Nokogiri::XML(Base64::decode64(ARGV[0]))
action = ARGV[1]
_username = ARGV[2]
_fed_entity = ARGV[3]

success = api_info.xpath("/CALL_INFO/RESULT").text.to_i == 1

if !success
  puts "Resource wasn't created, hook not executed"
  exit 0
end

# Elimina máquina de trabajo de backup si se elimina
update_backupjob(:delete, api_info) if action == "one.vm.terminate"
```

E.3 Librerías para los hooks

A continuación se ofrece el código correspondiente al *middleware* utilizado para interactuar con *OPA*, la interfaz *ruby* de *OpenNebula* y para *parsear* la entrada en formato XML de los *hooks*.

E.3.1 OPA

```
module OPA
  require "json"
```

```
require "ceramic"
require "net/http"

# Dirección según especificado en la tabla
SERVER = "192.168.10.56:2345"

BASE_POLICY_ENDPOINT = "/v1/data"

# Path a cada una de las políticas
POLICY_ENDPOINTS = {
  :naming => "/naming/compliant",
  :load => "/sla/compliant",
}

# Construye el contexto de una entrada de petición a OPA
def self.build_input_context(user, entity)
  {
    "username" => user,
    "federation_entity" => entity[:name],
  }
end

# Construye la entrada de una petición OPA para política de nombrado
def self.build_naming_input(user, entity, metadata)
  {
    "input" => {
      "context" => build_input_context(user, entity),
      "action" => {
        "type" => metadata[:action],
        "resource_name" => metadata[:resource_name],
        "date" => Date.today.to_s,
      },
    },
    "data" => {},
  }
end

# Construye la entrada de una petición OPA para política de sobrecarga
def self.build_load_input(user, entity, metadata)
  {
    "input" => {
      "context" => build_input_context(user, entity),
      "action" => {
        "type" => metadata[:action],
        "host_mem_allocation" => metadata[:host_mem_allocation],
        "datastore_free_space" => metadata[:ds_free_space],
        "datastore_total_capacity" => metadata[:ds_total_capacity],
      },
    },
    "data" => {},
  }
end
```

```ruby
  # Construye la entrada de una petición OPA
  def self.create_input(policy_type, user, entity, metadata = {})
    case policy_type
    when :naming
      build_naming_input(user, entity, metadata)
    when :load
      build_load_input(user, entity, metadata)
    end
  end

  # Valida una política contra el motor OPA
  # policy_type -> símbolo :naming o :load
  # request_input -> entrada para la política
  # data -> datos adicionales que procesar
  def self.compliance_test(policy_type, request_input, data = nil)
    uri = URI("http://#{SERVER}")
    uri.path = BASE_POLICY_ENDPOINT + POLICY_ENDPOINTS[policy_type]
    req = Net::HTTP::Post.new(uri)
    req.content_type = "application/json"
    req.body = request_input.to_json
    response = Net::HTTP.start(uri.hostname, uri.port) do |http|
      res = http.request(req)
      puts "Response status: #{res.code} #{res.message}"
      puts "Response body: #{res.body}"
      JSON::parse(response)
    end
    [response.body, response.error]
  end
end
```

E.3.2 *OpenNebula*

```
require "opennebula"

include OpenNebula

module One
  CREDENTIALS = "oneadmin:xxxxxxx"
  ENDPOINT = nil

  # Inicializar comunicación con API
  def set_endpoint(endpoint)
    @ENDPOINT = endpoint
    @client = Client.new(@ENDPOINT, @CREDENTIALS)
  end

  # Eliminar MV de OpenNebula
  def delete_vm(vmname)
    vm = VirtualMachine.new(VirtualMachine.build_xml(vmname), @client)
    vm.delete
  end

  # Eliminar imagen de OpenNebula
  def delete_image(iname)
    image = Image.new(Image.build_xml(iname), @client)
    image.delete
  end

  # Eliminar hook de OpenNebula
  def delete_hook(hname)
    hook = Hook.new(Hook.build_xml(hname), @client)
    hook.delete
  end

  # Eliminar backupjob de OpenNebula
  def delete_backupjob(bjname)
    image = BackupJob.new(BackupJob.build_xml(bjname), @client)
    image.delete
  end

  # Elimina un recurso de OpenNebula
  # rtype -> tipo de recurso
  # rname -> id del recurso
  def delete_resource(rtype, rname)
    begin
      case rtype
      when "vm"
        delete_vm(rname)
      when "image"
        delete_image(rname)
      when "hook"
```

```
        delete_hook(rname)
      when "backupjob"
        delete_backupjob(rname)
      end
    rescue
      puts "Failed to perform action #{rtype} on #{rname}"
    end
  end
end
```

E.3.3 Parsing

```
# Función que procesa un template XML de OpenNebula
# Devuelve un diccionario con el contenido del template
def parse_template(template_content)
  template_content.split(/\n/).inject({}) do |prev, kv|
    if kv != "]"
      key, value = kv.strip.split("=")
      value = value.strip.gsub('"', "")
      if value != "["
        prev[key] = value
      end
    end
    prev
  end
end
```

Anexo F

Manifiestos Puppet

En este anexo se documenta cada módulo *Puppet* definido en el despliegue de la infraestructura. Cada uno de estos módulos consta de manifiestos en lenguaje *Puppet*, *scripts* en lenguaje *ruby*, plantillas expresadas en metalenguaje *epp* y funciones de utilidad compartidas para todo el módulo, también en lenguaje *ruby*.

F.1 Módulo de almacenamiento

Este módulo define los componentes esenciales para el despliegue de un clúster Ceph en un única máquina. Pese a existir distintos módulos Puppet en la comunidad que despliegan Ceph, no se han podido adecuar a las necesidades concretas de este proyecto. También se define el almacenamiento local del nodo de virtualización, siguiendo la estructura LVM descrita en la implementación.

Los *scripts* de interés en este módulo son el de creación de dispositivos de bloque, que simulan sectores de disco distribuidos en la red, RBD, la creación de la arquitectura "Multi--zona" de *Rados Gateway* y la preparación del particionamiento lógico LVM.

Se ha empleado una plantilla *epp* que genera el fichero de configuración de Ceph, *ceph.conf*.

El módulo contiene tanto el almacenamiento local, con la definición de la estructura de LVM, como los recursos Ceph. El almacenamiento local no tiene dependencias internas ni externas, pero Ceph depende de los recursos de red y del almacenamiento local. Internamente, primero se despliegan los monitores, después los *managers* y posteriormente los OSD y MDS. Se termina con los RGW, la arquitectura "Multi-zona" y la creación de pooles.

F.1.1 Manifiestos

Almacenamiento local

Manifiesto que crea la estructura de LVM mencionada en la implementación. Se instala el paquete de *lvm2* según el sistema operativo y posteriormente se crea un grupo volumen cuyos volúmenes físicos son los dispositivios especificados como parámetro. Los volúmenes lógicos se crean también a partir del parámetro *lvs* de la clase definida.

```
  # Author | Lucas Cauhé Viñao
# Contact | lcauhe@gmail.com
# Description | Storage definition
# Docs -> https://forge.puppet.com/modules/puppetlabs/lvm/readme

class storage::lvm_stor (
  Array[String] $devices = ['/dev/sda'],
  Array[Hash] $lvs = {},
```

```
  ) {
    #
    # Instalar paquete LVM
    #
  $lvm2 = $facts['os']['name'] ? {
    default => lvm2
  }
  $vg_name = 'main'
  package { "lvm":
    ensure  => installed,
    name    => $lvm2,
  }

    #
    # Crea volúmenes físicos a partir de dispositivos
    #
  $devices.each |$device| {
    physical_volume { $device:
      ensure => present,
      require => Package['lvm'],
    }
  }

    #
    # Incorpora los volúmenes físicos a un mismo grupo volumen
    #
  volume_group { $vg_name:
    ensure           => present,
    physical_volumes => $devices,
  }

    #
    # Crea los volúmenes lógicos especificados en la entrada
    #
  $lvs.each |$lv| {
    lvm::logical_volume { $lv[name]:
      ensure       => present,
      volume_group => $vg_name,
      size         => $lv[size],
      yes_flag     => true, # wipe fs signatures
      fs_type      => $lv[fs_type],
      createfs     => $lv[fs_type] ? {
        'raw' => false,
      default => true
      },
      mountpath      => $lv[mount],
    }
  }
}
```

Ceph

Dentro de la primera etapa del despliegue de la infraestructura, también se despliega un clúster Ceph en alta disponibilidad. Este clúster cuenta con tres monitores, 2 *managers*, 5 OSD, 4 MDS y 1 RGW. Los monitores son los primeros en ser desplegados, para luego permitir la incorporación de managers, OSDs y MDS. Posteriormente, se puede crear el entorno de alta disponibilidad escogido para el Marketplace con, al menos, un gateway de RADOS en cada zona. La elección de contenedores para los servicios se discute en el Anexo D.1.

Este manifiesto instala un clúster Ceph sobre la infraestructura concreta del prototipo. Primero se instala la herramienta *cephadm* que ayuda a administrar el clúster. Primero, se despliega un solo monitor que instale su configuración y después otros dos monitores que reciben la configuración del primero. Se habilita la segunda versión del protocolo de comunicación con los monitores *msgr2*. Siguen los *managers* y los OSD. Por último se crean los MDS y la arquitectura Multi-zona.

```
  # storage/manifests/ceph.pp
define storage::ceph (
  String $id
) {
  #
  # Inlcude ceph vars
  #
  include storage::ceph::vars
  $cluster_name = $storage::ceph::vars::cluster_name
  $fsid = $storage::ceph::vars::fsid
  $network = $storage::ceph::vars::network
  $ceph_release = $storage::ceph::vars::ceph_release

  #
  # Install & Configure cephadm
  #
  storage::ceph::cephadm { "cephadm-$id":
    config_path => "/etc/${cluster_name[$id]}",
    fsid => $fsid[$id],
    monitors => $network[$id][mon]
  }

  #
  # Connect cephadm to podman network
  # (the gateway container is provided with an iptables entry that allows traffic
into the tagged
  # network that is useful now)
  #
    exec { "/usr/bin/ip route replace 192.168.$id.0/24 via ${network[$id]
[network_gateway]}":
    require => Storage::Ceph::Cephadm["cephadm-$id"]
  }

  $hostnames = $network[$id][mon].map |$m| { $m[hostname] }
  $batch_hostnames = $hostnames[1,-1]
```

```
  #
  # Bootstrap monitor
  #
  storage::ceph::monitor { "${hostnames[0]}":
    require => Storage::Ceph::Cephadm["cephadm-$id"],
    bootstrap => true,
    fsid => $fsid[$id],
    monitors => $network[$id][mon],
    pub_net => $network[$id][ceph_public_net],
    cluster_net => $network[$id][ceph_cluster_net],
    cluster_name => $cluster_name[$id],
    id => $id
  } ->


  #
  # Add 2 monitors to cluster
  #
  storage::ceph::monitor { $batch_hostnames:
    fsid => $fsid[$id],
    monitors => $network[$id][mon],
    pub_net => $network[$id][ceph_public_net],
    cluster_net => $network[$id][ceph_cluster_net],
    cluster_name => $cluster_name[$id],
    id => $id
  }->

  #
  # Enable the msgr2 version protocol
  #
  exec { "/usr/sbin/cephadm shell -m /etc/${cluster_name[$id]}:/etc/ceph -- ceph
mon enable-msgr2":
  }

  #
  # Add managers to cluster
  #
  $network[$id][mgr].each |$mgr| {
    storage::ceph::manager { $mgr[hostname]:
                             require     =>     $network[$id][mon].map     |$mon|
{ Storage::Ceph::Monitor["${mon[hostname]}"] },
      ipaddress => $mgr[ipaddress],
      mon => $network[$id][mon][0],
      cluster_name => $cluster_name[$id],
      id => $id
    }
  }

  #
  # Add osds
  #
```

```
  $network[$id][osds].each |$osd| {
    storage::ceph::osd { $osd[hostname]:
      require => Storage::Ceph::Manager["${network[$id][mgr][0][hostname]}"],
      mon => $network[$id][mon][0],
      ipaddress => $osd[ipaddress],
      cluster_ipaddress => $osd[cluster_ipaddress],
      cluster_name => $cluster_name[$id],
      id => $id
    }
  }

  #
  # Add CephFS
  #
  storage::ceph::cephfs { "myfs-$id":
    require => Storage::Ceph::Osd["${network[$id][osds][0][hostname]}"],
    cluster_name => $cluster_name[$id],
    id => $id
  }

    #
    # Add RGW
    #
    storage::ceph::rgw { "rgw-$id":
        require => Storage::Ceph::Cephfs["myfs-$id"],
        cluster_name => $cluster_name[$id],
        fsid => $fsid[$id],
        id => $id,
        mon => $network[$id][mon][0],
        ipaddress => $network[$id][rgw][0][ipaddress],
        role => $id ? {
            '10' => 'master',
            default => 'slave'
        },
        rgw_options => {
            'realm_name' => 'opennebula',
            'zonegroup_name' => 'marketplace',
            'zonegroup_address' => '192.168.10.95:7480',
            'zone_name' => "one-$id",
        }
    }


}
```

Cephadm

Manifiesto que instala la herramienta *cephadm*. Tiene una serie de dependencias y requiere un fichero de configuración *ceph.conf* que leer y poder comunicarse con los monitores. Por último se crea una clave de administrador para poder ejecutar acciones que requieran ese privilegio.

```
  # storage/manifests/ceph/cephadm.pp
define storage::ceph::cephadm (
  String $config_path = "/etc/ceph",
  String $fsid = "",
  Array[Hash] $monitors = []
) {

  #
  # Requirements
  #
  $requirements = [{ binary => 'python3', version => '>3.8'}, 'podman']
  $requirements.each |$req| {
    $real_req = $req =~ Hash ? { false => $req, default => $req[binary] }
    unless $facts["installed_binaries"][$real_req] {
      err("${real_req} not installed")
    }
              common::compare_versions($facts["installed_binaries"][$real_req]
["version"], $req =~ Hash ? { true => $req[version], default => "any" })
  }
   $required_services = ['chrony'] # lvm2 is already installed so not needed
to check
  $required_services.each |$service| {
    ensure_resource('package', $service, {ensure => present})
  }

  #
  # Install cephadm
  #
  ensure_resource('package', "cephadm", {
    ensure  => 'installed',
    provider => 'apt'
  })

  file { $config_path:
    ensure => 'directory',
    recurse => true
  }

  #
  # Configure cephadm
  #
  file { "$config_path/ceph.conf":
    ensure => 'present',
    require => [Package["cephadm"], File["$config_path"]],
    content => epp('storage/ceph_conf', {
```

```
      'config' => {
        'global' => {
          'fsid' => $fsid,
          'mon_initial_members' => $monitors.map |$m| { $m[hostname] },
          'mon_host' => $monitors.map |$m| { $m[ipaddress] },
          'container_image' => 'quay.io/ceph/ceph:v18'
        },
        'mon' => {
          'auth_allow_insecure_global_id_reclaim' => false
        }
      }
    })
  }

  #
  # Generate admin keyring
  #
  exec { "/usr/sbin/cephadm shell -m $config_path:$config_path -- ceph-authtool \
         -C -g $config_path/ceph.client.admin.keyring \
         -n client.admin \
         --cap mon 'allow *' \
         --cap osd 'allow *' \
         --cap mds 'allow *' \
         --cap mgr 'allow *'":
    require => [Package["cephadm"], File[$config_path]],
  }
}
```

Variables del módulo

Manifiesto que contiene las variables utilizadas en el módulo de almacenamiento y Ceph.

```
 class storage::ceph::vars {
 $ceph_release = "18.2.7"
 $network = {
   '10' => {
     'osds' => [
               {'hostname'  =>  'osd-0-10',  'ipaddress'  =>  '192.168.10.80',
'cluster_ipaddress' => '192.168.30.80'},
               {'hostname'  =>  'osd-1-10',  'ipaddress'  =>  '192.168.10.81',
'cluster_ipaddress' => '192.168.30.81'},
               {'hostname'  =>  'osd-2-10',  'ipaddress'  =>  '192.168.10.82',
'cluster_ipaddress' => '192.168.30.82'},
               {'hostname'  =>  'osd-3-10',  'ipaddress'  =>  '192.168.10.83',
'cluster_ipaddress' => '192.168.30.83'},
               {'hostname'  =>  'osd-4-10',  'ipaddress'  =>  '192.168.10.84',
'cluster_ipaddress' => '192.168.30.84'},
     ],
     'mon' => [
       {'hostname' => 'mon-0-10', 'ipaddress' => '192.168.10.90'},
       {'hostname' => 'mon-1-10', 'ipaddress' => '192.168.10.91'},
       {'hostname' => 'mon-2-10', 'ipaddress' => '192.168.10.92'}
     ],
     'mgr' => [
       # active
       {'hostname' => 'mgr-0-10', 'ipaddress' => '192.168.10.93'},
       # standby
       {'hostname' => 'mgr-1-10', 'ipaddress' => '192.168.10.94'}
     ],
     'mds' => [
       # active
       {'hostname' => 'mds-0-10', 'ipaddress' => '192.168.10.100'},
       # standby
       {'hostname' => 'mds-1-10', 'ipaddress' => '192.168.10.101'},
       {'hostname' => 'mds-2-10', 'ipaddress' => '192.168.10.102'},
       {'hostname' => 'mds-3-10', 'ipaddress' => '192.168.10.103'},
     ],
           'rgw' => [
       {'hostname' => 'rgw-0-10', 'ipaddress' => '192.168.10.95'},
           ],
     'network_gateway' => '10.88.0.144',
     'ceph_cluster_net' => '192.168.30.0/24',
     'ceph_public_net' => '192.168.10.0/24'
   },
   '20' => {
     'osds' => [
               {'hostname'  =>  'osd-0-20',  'ipaddress'  =>  '192.168.20.80',
'cluster_ipaddress' => '192.168.30.80'},
               {'hostname'  =>  'osd-1-20',  'ipaddress'  =>  '192.168.20.81',
'cluster_ipaddress' => '192.168.30.81'},
```

```
                {'hostname'  =>  'osd-2-20',  'ipaddress'  =>  '192.168.20.82',
'cluster_ipaddress' => '192.168.30.82'},
                {'hostname'  =>  'osd-3-20',  'ipaddress'  =>  '192.168.20.83',
'cluster_ipaddress' => '192.168.30.83'},
                {'hostname'  =>  'osd-4-20',  'ipaddress'  =>  '192.168.20.84',
'cluster_ipaddress' => '192.168.30.84'},
      ],
      'mon' => [
        {'hostname' => 'mon-0-20', 'ipaddress' => '192.168.20.90'},
        {'hostname' => 'mon-1-20', 'ipaddress' => '192.168.20.91'},
        {'hostname' => 'mon-2-20', 'ipaddress' => '192.168.20.92'}
      ],
      'mgr' => [
        # active
        {'hostname' => 'mgr-0-20', 'ipaddress' => '192.168.20.93'},
        # standby
        {'hostname' => 'mgr-1-20', 'ipaddress' => '192.168.20.94'}
      ],
      'mds' => [
        # active
        {'hostname' => 'mds-0-20', 'ipaddress' => '192.168.20.100'},
        # standby
        {'hostname' => 'mds-1-20', 'ipaddress' => '192.168.20.101'},
        {'hostname' => 'mds-2-20', 'ipaddress' => '192.168.20.102'},
        {'hostname' => 'mds-3-20', 'ipaddress' => '192.168.20.103'},
      ],
            'rgw' => [
        {'hostname' => 'rgw-0-20', 'ipaddress' => '192.168.20.95'},
            ],
      'network_gateway' => '10.88.0.144',
      'ceph_cluster_net' => '192.168.30.0/24',
      'ceph_public_net' => '192.168.20.0/24'
    }
  }

  $fsid = {
     '10' => storage::uuid(),
     '20' => storage::uuid()
  }
  $cluster_name = {
    '10' => 'ceph-10',
    '20' => 'ceph-20'
  }
}
```

Monitores

Manifiesto que despliega un monitor de RADOS. Define la imagen de contenedor que varios servicios de Ceph utilizan y define una serie de acciones a llevar a cabo si es el primer monitor del clúster en ser desplegado. Se sigue por desplegar el contenedor y aprovisionarlo para que inicie el clúster o se comunique con los demás monitores.

```
  # storage/manifests/ceph/monitor.pp
define storage::ceph::monitor (
  $mon = $title,
  String $fsid,
  String $pub_net,
  String $cluster_net,
  Array[Hash] $monitors = [],
  Optional[String] $cluster_name = undef,
  Boolean $bootstrap = false,
  String $id
) {
    #
    # Variables locales
    #
                $cluster_name_        =        common::unwrap_or($cluster_name,
$storage::ceph::vars::cluster_name['10'])
  $mon_ipaddress = $monitors.filter |$v| { $v[hostname] == $mon }[0][ipaddress]
  $imageName = "netfull-ceph"

  if $id == '10' {
    $guest_vlan = $virt::containers::guest_vlan_ceph_10
  } else {
    $guest_vlan = $virt::containers::guest_vlan_ceph_20
  }

    #
    # Crea imagen de contenedor base para varios componentes Ceph
    #
  ensure_resource('Virt::Container_file', $imageName, {
    from => 'docker.io/library/debian:12',
    run => [
      "apt-get update && apt install -y net-tools iproute2 iputils-ping procps
wget gnupg software-properties-common",
     "wget -q -O- 'https://download.ceph.com/keys/release.asc' | apt-key add -",
      "apt-add-repository 'deb https://download.ceph.com/debian-reef/ bookworm
main'",
      "apt-get install -y ceph ceph-mon ceph-mgr"
    ]
  })

    #
    # Si es el primer monitor, crea el directorio de almacenamiento
    #
  if $bootstrap {
    $bootstrap_aware_instructions = [
```

```
          "ceph-authtool --create-keyring /etc/ceph/ceph.mon.keyring --gen-key
-n mon. --cap mon 'allow *'",
             "ceph-authtool /etc/ceph/ceph.mon.keyring --import-keyring /etc/
ceph/ceph.client.admin.keyring",
          "mkdir -p /var/lib/ceph/bootstrap-osd",
       "ceph-authtool --create-keyring /var/lib/ceph/bootstrap-osd/ceph.keyring
--gen-key -n client.bootstrap-osd --cap mon 'profile bootstrap-osd' --cap mgr
'allow r'",
          "ceph-authtool /etc/ceph/ceph.mon.keyring --import-keyring /var/lib/
ceph/bootstrap-osd/ceph.keyring",
            "monmaptool --create --add ${monitors[0][hostname]} ${monitors[0]
[ipaddress]} --fsid $fsid /tmp/monmap",
    ]
    $initial_members = $mon
         $mon_host  =  ["[v2:${monitors[0][ipaddress]}:3300/0,v1:${monitors[0]
[ipaddress]}:6789/0]"]
    $leader_mon_keyring = []
  } else {
             $bootstrap_aware_instructions   =   ["monmaptool   --create   --
add  ${monitors[0][hostname]}  ${monitors[0][ipaddress]}  --add  ${monitors[1]
[hostname]}    ${monitors[1][ipaddress]}    --add    ${monitors[2][hostname]}
${monitors[2][ipaddress]} --fsid $fsid /tmp/monmap"]
    $initial_members = $monitors.map |$m| { $m[hostname] }
         $mon_host  =  $monitors.map  |$m|  {  "[v2:${m[ipaddress]}:3300/0,v1:
${m[ipaddress]}:6789/0]" }
       $leader_mon_keyring  =  ["podman  cp  ${monitors[0][hostname]}:/etc/ceph/
ceph.mon.keyring $mon:/etc/ceph"]

  }


    #
    # Definición del contenedor
    #
  virt::podman_unit { "$mon.container":
    require => Virt::Container_file[$imageName],
    args => {
      unit_entry => {
        'Description' => "Ceph $id monitor $mon container"
      },
      container_entry => {
        'AddCapability' => 'NET_ADMIN NET_RAW IPC_LOCK',
        'ContainerName' => $mon,
        'Exec' => 'sleep infinity',
        'Image' => $imageName,
        'Network' => "pod_ceph_$id",
        'IP' => $mon_ipaddress,
        'Label' => '\"ceph\"',
        'HostName' => $mon
      },
      install_entry => {
        'WantedBy' => 'multi-user.target'
```

```
      },
      service_restart => true,

    }
  } ->
    #
    # Aprovisionamiento y configuración del contenedor
    #
  virt::container_provision { "${mon}_provision":
    container_name => $mon,
    plan => [
      {
                # ceph.conf
        "type" => "guest_file",
        "actions" => [
          {
            'path' => "/etc/ceph/${cluster_name_}.conf",
            'content' => epp('storage/ceph_conf', {
              'config' => {
                'global' => {
                  'fsid' => $fsid,
                  'mon_initial_members' => $initial_members,
                  'mon_host' => join($mon_host, ' '),
                  'public_network' => $pub_net,
                  'cluster_network' => $cluster_net,
                  'auth_cluster_required' => 'cephx',
                  'auth_service_required' => 'cephx',
                  'auth_client_required' => 'cephx',
                  'osd_pool_default_size' => 3
                }
              }
            })
          }
        ]
      },
      {
                # distribución de keyrings
        "type" => "container",
        "actions" => ["podman cp /etc/$cluster_name_/ceph.client.admin.keyring
$mon:/etc/ceph"] + $leader_mon_keyring
      },
      {
                # configuración y bootstrap del monitor
        "type" => "guest",
        "actions" => $guest_vlan +  [
          "ip route add default via 192.168.$id.254",
          ]
          + $bootstrap_aware_instructions +
          [ "chown ceph:ceph /etc/ceph/ceph.mon.keyring",
          "mkdir /var/lib/ceph/mon/${cluster_name_}-$mon",
          "chown ceph:ceph /var/lib/ceph/mon/${cluster_name_}-$mon",
          "ceph-mon -c /etc/ceph/$cluster_name_.conf -n mon.$mon --setuser ceph
```

```
--setgroup ceph --cluster $cluster_name_ --mkfs -i $mon --monmap /tmp/monmap --
keyring /etc/ceph/ceph.mon.keyring",
            "cp /etc/ceph/ceph.mon.keyring /var/lib/ceph/mon/${cluster_name_}-
${mon}/keyring",
             "ceph-mon -c /etc/ceph/$cluster_name_.conf -n mon.$mon --cluster
$cluster_name_ -i $mon --mon-data /var/lib/ceph/mon/${cluster_name_}-${mon} --
debug_ms 5"
        ]
      }

    ]
  }
}
```

Managers

Manifiesto que despliega un *manager* de RADOS. Primero se generan las claves con las *capabilities* de acceso a los distintos servicios de RADOS para poder monitorizar su actividad. Posteriormente se despliega el contenedor y se aprovisiona con su fichero de configuración específico, la distribución de sus claves de CephX y la puesta en marcha del demonio.

```
  # storage/manifests/ceph/manager.pp
define storage::ceph::manager(
  String $ipaddress,
  Hash $mon,
  Optional[String] $cluster_name = undef,
  Optional[String] $fsid = undef,
  String $id
) {

    #
    # Variables locales
    #
                $cluster_name_        =        common::unwrap_or($cluster_name,
$storage::ceph::vars::cluster_name[$id])
  $fsid_ = common::unwrap_or($fsid, $storage::ceph::vars::fsid[$id])

  if $id == '10' {
    $guest_vlan = $virt::containers::guest_vlan_ceph_10
  } else {
    $guest_vlan = $virt::containers::guest_vlan_ceph_20
  }

  #
  # Create manager keyring & caps
  #
  exec { "${name}-creation":
    command => "/usr/sbin/cephadm shell \\
  -m /etc/${cluster_name_}:/etc/ceph -- \\
   ceph auth get-or-create mgr.$name mon 'allow profile mgr' osd 'allow *' mds
'allow *' > /etc/$cluster_name_/ceph.mgr.$name.keyring"
  }

  #
  # Create manager container
  #
  virt::podman_unit { "$name.container":
    args => {
      unit_entry => {
        'Description' => "Ceph $id manager $name container"
      },
      container_entry => {
        'AddCapability' => 'NET_ADMIN NET_RAW IPC_LOCK',
        'ContainerName' => $name,
        'Exec' => 'sleep infinity',
        'Image' => 'netfull-ceph',
```

```
        'Network' => "pod_ceph_$id",
        'IP' => $ipaddress,
        'Label' => '\"ceph\"',
        'HostName' => $name,
      },
      install_entry => {
        'WantedBy' => 'multi-user.target'
      },
      service_restart => true,
    }
  }->

  #
  # Add keyring and start manager
  #
  virt::container_provision { "${name}_provision":
    require => Exec["${name}-creation"],
    container_name => $name,
    plan => [
      {
            # ceph.conf
        "type" => "guest_file",
        "actions" => [
          {
            'path' => "/etc/ceph/${cluster_name_}.conf",
            'content' => epp('storage/ceph_conf', {
              'config' => {
                'global' => {
                  'fsid' => $fsid_,
                  'mon_host' => $mon[ipaddress],
                  'mon_initial_members' => $mon[hostname]
                }
              }
            })
          }
        ]
      },
      {
            # directorio de keyring
        "type" => "guest",
        "actions" => ["mkdir -p /var/lib/ceph/mgr/${cluster_name_}-$name"]
      },
      {
            # distribución de claves
        "type" => "container",
        "actions" => [
              "podman cp /etc/${cluster_name_}/ceph.mgr.$name.keyring $name:/
etc/ceph",
           "podman cp /etc/${cluster_name_}/ceph.mgr.$name.keyring $name:/var/
lib/ceph/mgr/${cluster_name_}-$name/keyring"
        ]
      },
```

```
      {
            # configuración de red y puesta en marcha
        "type" => "guest",
        "actions" => $guest_vlan + [
          "ip route add default via 192.168.$id.254",
           "ceph-mgr -i $name -c /etc/ceph/$cluster_name_.conf -n mgr.$name --
cluster $cluster_name_ --setuser ceph --setgroup ceph"
        ]
      }

    ]
  }
}
```

OSD

Manifiesto que despliega un OSD de RADOS. Primero se carga la utilidad de formateo de dispositivos para la instalación del sistema XFS en dispositivos locales. Se sigue por preparar el particionamiento lógico, documentado más adelante. Al desplegar el contenedor, se especifica el dispositivo que debe ver a través de un volumen. El aprovisionamiento consiste en especificar el *ceph.conf* específico y la instalación de los ficheros internos y el despliegue del demonio.

```
  # storage/manifests/ceph/osd.pp
define storage::ceph::osd(
  Optional[String] $fsid = undef,
  Optional[String] $cluster_name = undef,
  Hash $mon,
  String $ipaddress,
  String $cluster_ipaddress,
  String $id
) {
    #
    # Variables locales
    #
                $cluster_name_        =        common::unwrap_or($cluster_name,
$storage::ceph::vars::cluster_name[$id])
  $fsid_ = common::unwrap_or($fsid, $storage::ceph::vars::fsid[$id])
  $osd_fsid = storage::uuid()

  if $id == '10' {
    $guest_vlan = $virt::containers::guest_vlan_ceph_10
    $cluster_vlan = $virt::containers::guest_vlan_stor_10
  } else {
    $guest_vlan = $virt::containers::guest_vlan_ceph_20
    $cluster_vlan = $virt::containers::guest_vlan_stor_20
  }

    #
    # Instalación de paquetes para FS XFS
    #
  ensure_resource ('package', 'xfsprogs', {
    ensure => 'installed'
  })
    #
    # Cargar módulo XFS del kernel
    #
  ensure_resource ('exec', 'load-xfs', {
    require => Package['xfsprogs'],
    command => '/usr/sbin/modprobe xfs'
  })

  #
  # Ensure lvm is created & clean
  # Needs to be checked at run time because the lvm might be created on the same
manifest run and in compile time would
```

```
  # fails to the eyes of facter
  #
  $pwd = common::pwd()
  $scripts_path = "$pwd/deploy/$module_name/scripts"
    exec  {  "/usr/bin/ruby  $scripts_path/prepare_lvm.rb  $name  $cluster_name_
$osd_fsid ${mon[hostname]}":
    require => Exec['load-xfs'],
    logoutput => true,
    refreshonly => true
  } ->

  #
  # Bootstrap OSD
  #
  virt::podman_unit { "$name.container":
    args => {
      unit_entry => {
        'Description' => "Ceph $id osd $name container"
      },
      container_entry => {
     'AddCapability' => 'NET_ADMIN NET_BIND_SERVICE SYS_ADMIN NET_RAW IPC_LOCK',
        'ContainerName' => $name,
        'Exec' => 'sleep infinity',
        'Image' => 'netfull-ceph',
        'Network' => "pod_ceph_$id",
        'IP' => $ipaddress,
        'HostName' => $name,
      'Volume' => "/etc/$cluster_name_/osd/$name:/var/lib/ceph/osd/ceph-$name",
        'EnvironmentFile' => "/tmp/env-$name",
        'Label' => '\"ceph\"',
        'Ulimit' => 'nofile=1048576:1048576'
      },
      install_entry => {
        'WantedBy' => 'multi-user.target'
      },
      service_restart => true,
    }
  }->

    #
    # Aprovisionamiento y configuración del OSD
    #
  virt::container_provision { "${name}_provision":
    container_name => $name,
    plan => [
      {
                # ceph.conf
        "type" => "guest_file",
        "actions" => [
          {
            'path' => "/etc/ceph/${cluster_name_}.conf",
            'content' => epp('storage/ceph_conf', {
```

```
              'config' => {
                'global' => {
                  'fsid' => $fsid_,
                  'mon_host' => $mon[ipaddress],
                  'mon_initial_members' => $mon[hostname],
                       'public_network' => $storage::ceph::vars::network[$id]
[ceph_public_net],
                     'cluster_network' => $storage::ceph::vars::network[$id]
[ceph_cluster_net]
                }
              }
            })
          },

        ]
      },
      {
                # red de replicación
        "type" => "container",
        "actions" => [
          "podman network connect --ip $cluster_ipaddress pod_stor_$id $name"
        ]
      },
      {
                # puesta en marcha
        "type" => "guest",
        "actions" => $guest_vlan + $cluster_vlan + [
          "ip route add default via 192.168.$id.254",
          "apt install udev", # needed by ceph-volume
         join(['bash -c "ceph-osd -c /etc/ceph/', $cluster_name_, '.conf -i \$ID
--mkfs --osd-data /var/lib/ceph/osd/ceph-', $name, ' --osd-uuid ', $osd_fsid,
'"'], ''),
          "chown -R ceph:ceph /var/lib/ceph/osd/ceph-$name",
           join(['bash -c "ceph-osd -c /etc/ceph/', $cluster_name_, '.conf -i
\$ID -n osd.\$ID --cluster ', $cluster_name_, ' --setuser ceph --setgroup ceph
--debug_ms 10 --osd-data /var/lib/ceph/osd/ceph-', $name, '"'], '')
        ]
      }

    ]
  }
}
```

MDS

Manifiesto que despliega un MDS de RADOS. Con la puesta en marcha del clúster ya se puede interactuar directamente con los monitores. Se crean las claves del servicio y se despliega el contenedor. Se aprovisiona con el fichero de configuración de Ceph, la distribución de claves y la puesta en marcha del servicio.

```
  # storage/manifests/ceph/mds.pp
define storage::ceph::mds(
  Optional[String] $fsid = undef,
  Optional[String] $cluster_name = undef,
  Hash $mon,
  String $ipaddress,
  String $id
) {
  #
  # Variables locales
  #
                $cluster_name_        =        common::unwrap_or($cluster_name,
$storage::ceph::vars::cluster_name[$id])
  $fsid_ = common::unwrap_or($fsid, $storage::ceph::vars::fsid[$id])

  if $id == '10' {
    $guest_vlan = $virt::containers::guest_vlan_ceph_10
  } else {
    $guest_vlan = $virt::containers::guest_vlan_ceph_20
  }
  #
  # Create local mds folder
  #
  exec { "/usr/bin/mkdir -p /etc/$cluster_name_/mds/$name":
  }->

  #
  # Create keyring
  #
   exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/etc/ceph -- ceph-
authtool --create-keyring /etc/ceph/mds/$name/keyring --gen-key -n mds.$name":
  } ->

  #
  # Add it to the auth registry
  #
  exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/etc/ceph -- ceph auth
add mds.$name osd 'allow rwx' mds 'allow *' mon 'allow profile mds' -i /etc/
ceph/mds/$name/keyring":
  }->

  #
  # Definición del contenedor
  #
  virt::podman_unit { "$name.container":
```

```
  args => {
    unit_entry => {
      'Description' => "Ceph $id mds $name container"
    },
    container_entry => {
      'AddCapability' => 'NET_ADMIN NET_RAW IPC_LOCK',
      'ContainerName' => $name,
      'Exec' => 'sleep infinity',
      'Image' => 'netfull-ceph',
      'Network' => "pod_ceph_$id",
      'IP' => $ipaddress,
      'Label' => '\"ceph\"',
      'HostName' => $name,
    },
    install_entry => {
      'WantedBy' => 'multi-user.target'
    },
    service_restart => true,
  }
}->

#
# Aprovisionamiento y configuración del MDS
#
virt::container_provision { "${name}_provision":
  container_name => $name,
  plan => [
    {
          # ceph.conf
      "type" => "guest_file",
      "actions" => [
        {
          'path' => "/etc/ceph/${cluster_name_}.conf",
          'content' => epp('storage/ceph_conf', {
            'config' => {
              'global' => {
                'fsid' => $fsid_,
                'mon_host' => $mon[ipaddress],
                'mon_initial_members' => $mon[hostname],
              },
              "mds.$name" => {
                'host' => "$name"
              }
            }
          })
        },

      ]
    },
    {
          # Directorio de claves
      "type" => "guest",
```

```
        "actions" => [
          "mkdir -p /var/lib/ceph/mds/$cluster_name_-$name",
        ]
      },
      {
            #  compartición de credenciales
        "type" => "container",
        "actions" => [
          "podman cp /etc/$cluster_name_/mds/$name/keyring $name:/var/lib/ceph/
mds/$cluster_name_-$name/keyring"
        ]
      },
      {
            # Puesta en marcha
        "type" => "guest",
        "actions" => $guest_vlan + [
          "ip route add default via 192.168.$id.254",
          "ceph-mds --cluster $cluster_name_ -i $name"
        ]
      }

    ]
  }
}
```

RGW

Manifiesto que despliega un Rados Gateway de RADOS, en una arquitectura Multi-zona. Se identifica el rol que juega la zona en la que se va a crear, se crea el usario administrador para la replicación y se despliega el contenedor. El aprovisionamiento consta del fichero de configuración y la distribución de credenciales. Por último se incorpora el RGW a la zona concreta.

```
  # storage/manifests/ceph/rgw.pp
define storage::ceph::rgw (
    String $id,
    String $cluster_name,
    String $fsid,
    Hash $mon,
    String $ipaddress,
    Enum['master', 'slave'] $role = 'slave',
    Hash $rgw_options
) {

    #
    # Variables locales
    #
    notice("Bootstraping RGW: $name")
  $fsid_ = common::unwrap_or($fsid, $storage::ceph::vars::fsid[$id])
  $pwd = common::pwd()
  $scripts_path = "$pwd/deploy/$module_name/scripts"
    if $id == '10' {
        $guest_vlan = $virt::containers::guest_vlan_ceph_10
    } else {
        $guest_vlan = $virt::containers::guest_vlan_ceph_20
    }

    #
    # Create realm and zonegroup if role is set to master
    #
    if $role == 'master' {
        $zone_creation_role = 'master'
        notice("Creating new realm")
          exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph --
radosgw-admin realm create --rgw-realm=${rgw_options[realm_name]} --default > /
etc/$cluster_name/rgw.realm.${rgw_options[realm_name]}":
            refreshonly => true
        }->
          exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph --
radosgw-admin zonegroup create --rgw-zonegroup=${rgw_options[zonegroup_name]}
--endpoints=${rgw_options[zonegroup_address]} --master --default > /etc/
$cluster_name/rgw.zonegroup.${rgw_options[zonegroup_name]}":
            refreshonly => true
        }->
        file { "/tmp/dummy-$name-$id":
            ensure => 'absent'
        }
```

```
    } else {
        $zone_creation_role = ""
        file { "/tmp/dummy-$name-$id":
            ensure => 'absent'
        }
    }

  #
  # Create `rgw-$id` ceph user
  #
    $user_keyring = "ceph.client.rgw-$id.keyring"
  storage::ceph::user { "rgw-$id":
        require => File["/tmp/dummy-$name-$id"],
    type => 'client',
    cluster_name => $cluster_name,
    caps => "mon 'allow rw' osd 'allow rwx'",
        output => $user_keyring
  }

  virt::podman_unit { "$name.container":
        require => Storage::Ceph::User["rgw-$id"],
    args => {
      unit_entry => {
        'Description' => "Ceph $id rgw $name container"
      },
      container_entry => {
     'AddCapability' => 'NET_ADMIN NET_BIND_SERVICE SYS_ADMIN NET_RAW IPC_LOCK',
        'ContainerName' => $name,
        'Exec' => 'sleep infinity',
        'Image' => 'netfull-ceph',
        'Network' => "pod_ceph_$id",
        'IP' => $ipaddress,
        'HostName' => $name,
                  'Volume' => "/etc/$cluster_name/$user_keyring:/var/lib/ceph/
radosgw/ceph-rgw-$id/keyring"
      },
      install_entry => {
        'WantedBy' => 'multi-user.target'
      },
      service_restart => true,
    }
  }->
    #
    # Aprovisionamiento y configuración de RGW
    #
  virt::container_provision { "${name}_provision":
    container_name => $name,
    plan => [
      {
            # ceph.conf
        "type" => "guest_file",
        "actions" => [
```

```
          {
            'path' => "/etc/ceph/$cluster_name.conf",
            'content' => epp('storage/ceph_conf', {
              'config' => {
                'global' => {
                  'fsid' => $fsid_,
                  'mon_host' => $mon[ipaddress],
                  'mon_initial_members' => $mon[hostname],
                       'public_network' => $storage::ceph::vars::network[$id]
[ceph_public_net],
                }
              }
            })
          },

        ]
      },
            {
            # distribución de credenciales
                "type" => "container",
           "actions" => ["podman cp /etc/$cluster_name/ceph.client.admin.keyring
$name:/etc/ceph"]
            },
      {
            # puesta en marcha
        "type" => "guest",
        "actions" => $guest_vlan + [
          "ip route add default via 192.168.$id.254",
                    "apt install -y radosgw",
                    "chown -R ceph:ceph /var/lib/ceph"
        ]
      }

    ]
  }->
  exec { "/usr/bin/ruby $scripts_path/rgw_zone_creation.rb $name $cluster_name
$id     $role     ${rgw_options[zonegroup_name]}     ${rgw_options[zone_name]}
$ipaddress:7480":
    logoutput => true,
    refreshonly => true
  }

}
```

CephFS

Manifiesto que despliega varios MDS de Ceph.

```
  # storage/manifests/ceph/cephfs.pp
define storage::ceph::cephfs(
  String $id,
  Optional[String] $cluster_name = undef
) {
    #
    # Creación de MDS especificados en la entrada
    #
  $storage::ceph::vars::network[$id][mds].each |$mds| {
    storage::ceph::mds { $mds[hostname]:
      cluster_name => $cluster_name,
      mon => $storage::ceph::vars::network[$id][mon][0],
      ipaddress => $mds[ipaddress],
      id => $id
    }
  }
}
```

Crear CephFS

Manifiesto que crea un sistema de ficheros CephFS. Se crean los pooles de datos y metadatos, sobre el que añadir capacidad de réplica mediante *erasure code*, y se crea el sistema de ficheros. Por último se crea un usuario autorizado para operar sobre el sistema de ficheros.

```
  # storage/manifests/ceph/newfs.pp
define storage::ceph::newfs (
    String $cluster_name = 'ceph',
    String $user_name = 'newfs'
) {

    #
    # Create EC 3-1 profile Ceph pool
    #
    storage::ceph::pool { "${name}_data-$cluster_name":
        type => 'erasure',
        cluster_name => $cluster_name,
        pg_autoscale => 'on',
        rbd => false,
        ecpool_opts => {
            'k' => 3,
            'm' => 1
        }
    }

    storage::ceph::pool { "${name}_meta-$cluster_name":
        type => 'replicated',
        cluster_name => $cluster_name,
        pg_autoscale => 'on',
        rbd => false,
    }

    #
    # Define crush rule for the least occupied HDD drives to form the EC pool
    #

    #
    # Define a CephFS over the EC pool
    #
    exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph -- ceph osd
pool set ${name}_data-$cluster_name allow_ec_overwrites true":
                            require    =>    [Storage::Ceph::Pool["${name}_meta-
$cluster_name"],Storage::Ceph::Pool["${name}_data-$cluster_name"]],
        refreshonly => true
    }->
    exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph -- ceph fs
new ${name} ${name}_meta-$cluster_name ${name}_data-$cluster_name --force":
        refreshonly => true
    }->

    #
```

```
    # Create user for pool
    #
     exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph -- ceph
fs authorize ${name} client.$user_name / rw > /etc/$cluster_name/ceph.client.
$user_name.keyring":
        refreshonly => true
    }


}
```

User

Manifiesto que crea un usuario en el registro de Ceph. Toma una serie de *capabilities* y permite guardar el *keyring* generado en un fichero.

```
  # storage/manifests/ceph/user.pp
define storage::ceph::user (
  String $type = 'client',
  Optional[String] $cluster_name = undef,
  Optional[String] $caps = undef,
    Optional[String] $output = undef
) {
    #
    # Variables locales
    #
                $cluster_name_        =        common::unwrap_or($cluster_name,
$storage::ceph::vars::cluster_name["10"])
    if $output {
        $redirection = "> /etc/$cluster_name_/$output"
    } else {
        $redirection = ""
    }
  #
  # Create user if doesnt exists
  #
  exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/etc/ceph -- ceph auth
get-or-create $type.$name $caps $redirection":
  }
}
```

Pool

Manifiesto que crea un pool Ceph. Identifica el modo de replicación, réplica *n* o *erasure code* y si se va a utilizar para almacenar imágenes RBD, lo que requiere acciones adicionales.

```
  # storage/manifests/ceph/pool.pp
define storage::ceph::pool (
  Enum['replicated', 'erasure'] $type = 'replicated',
  Enum['on', 'off', 'warn'] $pg_autoscale = 'off',
  Optional[String] $cluster_name = undef,
  Optional[String] $pg_num = undef,
    Boolean $rbd = true,
    Optional[Hash] $ecpool_opts = undef
) {

    #
    # Variables locales
    #
                $cluster_name_        =        common::unwrap_or($cluster_name,
$storage::ceph::vars::cluster_name["10"])
    $pg_num_ = common::unwrap_or($pg_num, "")

  notice("Defined ceph pool $name")

    #
    # Set ecpool options if needed
    #
    if $type == 'erasure' {
        $profile_values = join($ecpool_opts.map  |$k, $v| { "$k=$v" }, " ")
        exec { "erasure-profile-$cluster_name-$name":
           command => "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/etc/ceph
-- ceph osd erasure-code-profile set ecprofile $profile_values"
        }->
        exec { "create-$cluster_name-$name-pool":
                 command => "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/
etc/ceph -- ceph osd pool create $name $pg_num_ $type ecprofile --autoscale-
mode=$pg_autoscale"
        }
    } else {
        #
        # Create the pool
        #
        exec { "create-$cluster_name-$name-pool":
           command => "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/etc/ceph
-- ceph osd pool create $name $pg_num_ $type --autoscale-mode=$pg_autoscale"
        }
    }


    if $rbd {
        #
        # Enable the pool
```

```
        #
        exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/etc/ceph -- ceph
osd pool application enable $name rbd":
            require => Exec["create-$cluster_name-$name-pool"]
        }->

        #
        # Init the pool
        #
        exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name_:/etc/ceph -- rbd
pool init -p $name":
        }
    }

}
```

RBD

Manifiesto que crea un dispositivo de bloques RBD. Se asegura de que el pool en el que se va a crear exista y crea una imagen dentro. Al usuario especificado se le otorgan capacidades de explotación y se monta (*map*) el dispositivo en el punto de montaje especificado.

```
  # storage/manifests/ceph/rbd.pp
define storage::ceph::rbd (
  String $cluster_name = 'ceph',
  String $pool,
  String $image,
  String $size,
  String $mountpoint,
) {
  #
  # Variables locales
  #
  $pwd = common::pwd()
  $scripts_path = "$pwd/deploy/storage/scripts"

  #
  # Define pool & image
  #
  ensure_resource('storage::ceph::pool', $pool, {
    pg_autoscale => 'on',
    pg_num => "128",
    cluster_name => $cluster_name
  })
  exec { "create-rbd-image-$cluster_name-$image":
    require => Storage::Ceph::Pool[$pool],
    refreshonly => true,
    command => "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph -- rbd
create --size $size --pool $pool $image"
  }

  #
  # Add cephx authz
  #
  exec { "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph -- ceph auth
get-or-create client.$name mon 'allow r' osd 'allow rwx pool=$pool' > /etc/
$cluster_name/ceph.client.$name.keyring":
  }

  #
  # Map rbd device on the host
  #
  exec { "/usr/bin/ruby $scripts_path/rbdmap.rb $cluster_name $name $pool $image
$mountpoint":
    require => Exec["create-rbd-image-$cluster_name-$image"]
  }
```

```
}
```

S3 User

Manifiesto que crea un usuario de s3.

```
  # storage/manifests/ceph/s3_user.pp
define storage::ceph::s3_user (
    String $display_name = "User-$name"
) {
    notice("Creating Ceph S3 user: $name")

    # Creación de usuario de s3
    exec { "/usr/sbin/cephadm shell -m /etc/ceph-10:/etc/ceph -- radosgw-admin
user create --uid='$name' --display-name='$display_name'":
        refreshonly => true
    }
}
```

F.1.2 Scripts

rbdmap

Script que instala el sistema de ficheros XFS en un dispositivo y lo monta en un punto de montaje.

```
  #!/usr/bin/env ruby

require "open3"

cluster_name = ARGV[0]
client_name = ARGV[1]
pool = ARGV[2]
image = ARGV[3]
mountpoint = ARGV[4]

# Muestra un error en caso de fallo
# El programa termina si hay fallo
def show_error(output, status)
  if output != "" && status && status != 0
    puts "RBD mount went wrong: #{status}: #{output}"
    exit 1
  end
end

# Ejecuta un comando de sistema arbitrario y devuelve
# la salida del comando
def run_command(cmd)
  puts "RUNNING COMMAND: #{cmd}"
  out, stderr, status = Open3.capture3(cmd)
  show_error(stderr, status)
  return out.strip
end

#
# If there is an rbd already on that mountpoint do nothing
#
exit 0 unless run_command("/usr/bin/findmnt #{mountpoint}") == ""

#
# Ensure rbd kernel module is loaded
#
_ = run_command("/usr/sbin/modprobe rbd")

#
# Map the device
#
device      =      run_command("/usr/bin/rbd      -n      client.#{client_name}
-c        /etc/#{cluster_name}/ceph.conf        -k        /etc/#{cluster_name}/
ceph.client.#{client_name}.keyring  --id  #{client_name}  map  --pool  #{pool}
#{image}")
```

```
#
# Wipe device
#
_ = run_command("/usr/sbin/wipefs -a #{device}")

#
# Format device with xfs
#
_ = run_command("/usr/sbin/mkfs.xfs #{device}")

#
# Mount device into `mountpoint`
#
_ = run_command("/usr/bin/mount -t xfs #{device} #{mountpoint}")
```

RGW zone creation

Script que crea la arquitectura Multi-zona de Ceph según el rol especificado en el parámetro de entrada.

```
  #!/usr/bin/env ruby

require "open3"
require "json"
require "fileutils"

$cluster_name = ARGV[1]
id = ARGV[2]
role = ARGV[3]
zonegroup_name = ARGV[4]
zone_name = ARGV[5]
address = ARGV[6]

# Muestra un error en caso de fallo
# El programa termina si hay fallo
def show_error(cmd, output, status)
  if output != "" && status && status != 0
    puts "RGW admin command went wrong"
    puts cmd
    puts "OUTPUT #{status}: #{output}"
    exit 1
  end
end

# Ejecuta comando podman arbitrario
def run_command(cmd)
  container_name = ARGV[0]
  prefix = "podman exec #{container_name} "
  stdout, stderr, status = Open3.capture3(prefix + cmd)
  show_error(cmd, stderr, status)
  stdout
end

# Ejecuta comando de radosgw-admin y devuelve su resultado
def rgw_admin(cmd)
  rgw_prefix = "radosgw-admin -c /etc/ceph/#{$cluster_name}.conf -k /etc/ceph/
ceph.client.admin.keyring "
  run_command(rgw_prefix + cmd)
end

# Crea usuario en la arquitectura multi-zona
def create_user(zone_name, user_id)
  creds_file = "/tmp/radosgw-#{user_id}-creds.json"
  user_info = rgw_admin("user create --uid=#{user_id} --display-name=#{user_id}
--system --rgw-zone=#{zone_name}")
  File.write(creds_file, user_info)
  JSON.parse(user_info)
```

```
end

# Obtiene un usuario de radosgw-admin
def get_user(user_id)
  creds_file = "/tmp/radosgw-#{user_id}-creds.json"
  file = File.read creds_file
  JSON.load file
end

zonegroup_file = "/tmp/radosgw-zonegroup-#{zonegroup_name}.json"
sync_user = "synchronization-user"

if role == "master"
  #
  # Create zone
  #
        rgw_admin("zone    create    --rgw-zonegroup=#{zonegroup_name}    --rgw-
zone=#{zone_name} --master --endpoints=http://#{address}")

  #
  # Store zonegroup info for slaves
  #
          File.write(zonegroup_file,     rgw_admin("zonegroup     get     --rgw-
zonegroup=#{zonegroup_name}"))

  #
  # Create zone admin user
  #
  user_info = create_user(zone_name, sync_user)

  #
  # Add keys to zone
  #
             rgw_admin("zone       modify       --rgw-zone=#{zone_name}       --
access-key=#{user_info["keys"][0]["access_key"]}   --secret=#{user_info["keys"]
[0]["secret_key"]}")

  #
  # Update period
  #
  rgw_admin("period update --commit --rgw-zone=#{zone_name} ")
else

  #
  # Get admin user info & zone url
  #
  user_info = get_user(sync_user)
  zg_file = File.read zonegroup_file
  zonegroup_info = JSON.load(zg_file)
     master_zone   =   zonegroup_info["zones"].find   {   |zone|   zone["id"]   ==
zonegroup_info["master_zone"] }
```

```
  access_key = user_info["keys"][0]["access_key"]
  secret = user_info["keys"][0]["secret_key"]

  #
  # Pull realm config
  #
  realm_config = rgw_admin("realm pull --url=#{master_zone["endpoints"][0]} --
access-key=#{access_key} --secret=#{secret}")
  realm_config = JSON.parse(realm_config)

  # Make realm the default one
  rgw_admin("realm default --rgw-realm=#{realm_config["name"]}")

  #
  # Create secondary zone
  #
        rgw_admin("zone    create    --rgw-zonegroup=#{zonegroup_name}    --
rgw-zone=#{zone_name}    --access-key=#{access_key}    --secret=#{secret}    --
endpoints=http://#{address}")

  #
  # Update period
  #
   rgw_admin("period update --commit --rgw-zonegroup=#{zonegroup_name} --rgw-
zone=#{zone_name} ")
end

#
# Start rgw process
#
run_command("radosgw   -c   /etc/ceph/#{$cluster_name}.conf   --keyring   /var/
lib/ceph/radosgw/ceph-rgw-#{id}/keyring   --name   client.rgw-#{id}   --cluster
#{$cluster_name} --setuser ceph --setgroup ceph --rgw-zone=#{zone_name}")
```

LVM preparation

Script que perpara un LVM para ser utilizado por un OSD.

```
#!/usr/bin/env ruby

require 'open3'
require 'fileutils'

osd_name = ARGV[0]
cluster_name = ARGV[1]
fsid = ARGV[2]
monitor = ARGV[3]

def show_error(output, status)
  if output != "" && status && status != 0
    puts "OSD creation went wrong: #{status}: #{output}"
    exit 1
  end
end


stdout, stderr, status = Open3.capture3("lvdisplay -m main/#{osd_name}")

if stderr != ""
  puts "Lvm doesn't exist for osd #{osd_name}, create one with the same name"
  exit 1
end

lv_path = stdout.lines[1].strip.split[-1]
# Zap & clean mounted dir if it already exists
FileUtils.rm_rf("/etc/#{cluster_name}/osd/#{osd_name}")
Open3.capture3("umount /etc/#{cluster_name}/osd/#{osd_name}")
Open3.capture3("cephadm shell -m /etc/#{cluster_name}:/etc/ceph -- ceph-volume
lvm zap #{lv_path}")
puts "Device #{lv_path} zapped"

#
# Create new osd from cephadm shell
#
osd_secret, sdterr, status = Open3.capture3("cephadm shell -- ceph-authtool --
gen-print-key")
show_error(stderr, status)
osd_secret = osd_secret.strip
Open3.capture3("podman cp #{monitor}:/var/lib/ceph/bootstrap-osd/ceph.keyring /
etc/#{cluster_name}")
osd_id,   stderr,   status   =   Open3.capture3("echo   {   \"cephx_secret\":
\"#{osd_secret}\" } | cephadm shell -m /etc/#{cluster_name}:/etc/ceph -- ceph
osd new #{fsid} -i - -n client.bootstrap-osd -k /etc/ceph/ceph.keyring")
show_error(stderr, status)

osd_id = osd_id.strip
```

```
puts "New osd #{osd_name} created with fsid #{fsid}, osd_secret #{osd_secret}
and id #{osd_id}"
#
# Handle device preparation
#
File.open("/tmp/env-#{osd_name}",    File::CREAT|File::WRONLY)    {|f|    f.puts
"ID=#{osd_id}\nOSD_SECRET=#{osd_secret}"}
FileUtils.mkdir_p("/etc/#{cluster_name}/osd/#{osd_name}")
Open3.capture3("mkfs.xfs /dev/main/#{osd_name}")
Open3.capture3("mount      /dev/main/#{osd_name}      /etc/#{cluster_name}/osd/
#{osd_name}")
puts "Formatted device /dev/main/#{osd_name} and mounted"

#
# Create keyring with secret for osd
#
keyring_cmd = "cephadm shell -m /etc/#{cluster_name}:/etc/ceph -- ceph-authtool
--create-keyring /etc/ceph/osd/#{osd_name}/keyring --name osd.#{osd_id} --add-
key #{osd_secret}"
puts "Running command #{keyring_cmd}"
_, stderr, status = Open3.capture3(keyring_cmd)
show_error(stderr, status)
puts "Created keyring for osd"

#
# Add osd keyring to auth registry
#
_, stderr, status = Open3.capture3("cephadm shell -m /etc/#{cluster_name}:/etc/
ceph -- ceph auth add osd.#{osd_id} osd 'allow *' mon 'allow profile osd' -i /
etc/ceph/osd/#{osd_name}/keyring")
show_error(stderr, status)
puts "Added keyring to auth registry"
```

F.1.3 Funciones

CephX Key

```
  require 'securerandom'
require 'base64'
Puppet::Functions.create_function(:'storage::cephx_key') do
  dispatch :_cephx_key do
  end

  def _cephx_key
    rb = SecureRandom.random_bytes(16)
    Base64.strict_encode64(rb)
  end
end
```

UUID

```
  require 'securerandom'
Puppet::Functions.create_function(:'storage::uuid') do
  dispatch :_uuid do
  end

  def _uuid
    SecureRandom.uuid
  end
end
```

F.1.4 Plantillas

Ceph Conf

```
  <%- | Hash $config | -%>
# Config generated with Puppet
<% $config.each |$field, $field_config| { -%>
[<%= $field %>]
<% $field_config.each |$atr, $atr_v| { -%>
<% if $atr_v =~ Array { -%>
<%= $atr %> = <%= join($atr_v, ",") %>
<%- } else { -%>
<%= $atr %> = <%= $atr_v %>
<% } -%>
<% } -%>

<% } -%>
```

F.2 Módulo de virtualización

En este módulo se documentan los aspectos relativos a los servicios de cómputo de la infraestructura, el despliegue de los núcleos, *datastores* y nodo de virtualización de *OpenNebula* y *gateway* de la red frontal. Por cada contenedor que ejecuta el software del núcleo de *OpenNebula*, hay otro contenedor corriendo una base de datos *MariaDB*. También se aborda aquí el proceso de federación de las dos entidades federadas. El contenedor de la puerta de enlace es el primero en ser desplegado, seguido de las bases de datos y los núcleos de *OpenNebula*. Posteriormente se establece la comunicación entre cada núcleo, formando un único clúster en alta disponibilidad. Se termina ejecutando los demonios encargados de exportar métricas y desplegando el contenedor Prometheus.

Los *scripts* empleados en este módulo son los de aprovisionamiento de contenedores, uno de los más extensos, y el que simula el comportamiento requerido del demonio principal del sistema *systemd*. El primero ofrece una interfaz con la que acceder al contexto local de cada contenedor. Pese a sonar contraintuitivo (los contenedores vienen aprovisionados ya antes de ser desplegados), el procedimiento por el que se establece alta disponibilidad entre los tres núcleos de *OpenNebula* requiere acceso a los contenedores en tiempo de ejecución, siendo capaz de modificar ficheros de configuración y ejecutar comandos, esperando su respuesta y sincronizando el estado de otros contenedores. El recurso definido permite esta operativa y ofrece una interfaz por la que acceder a recursos del contenedor a través de variables personalizadas.

En concreto, primero se despliegan tres núcleos en alta disponibilidad que forman la zona maestra de la federación, seguido viene el núcleo maestro de la segunda zona con el que se establece la federación. Una vez establecida, se incorporan dos núcleos más a la zona esclava.

Se han empleado varias plantillas *epp* entre las que destacan las de creación de ficheros de contenedor (*Containerfile*) y la configuración de los *datastore*.

F.2.1 Manifiestos

Unidad Systemd Podman

Manifiesto que instala una unidad systemd de podman.

```
define virt::podman_unit (
  Hash $args,
){
    #
    # Variables locales
    #
  notice("Defining podman unit $name")
  $unit_name = "puppet-podman-$name"
  $mid = regsubst($unit_name, /\.container/, '', 'G')
  $quadlet_name = regsubst($mid, /\./, '-', 'G')

    #
```

```
    # Creación de fichero unit
    #
  systemd::manage_unit { $unit_name:
    path => '/etc/containers/systemd',
    * => $args,
  } ->
    #
    # Transforma systemd unit contenedor a systemd nativo
    #
  exec { "generate quadlet $unit_name":
    command => 'podman-system-generator /etc/systemd/system',
    path => '/usr/lib/systemd/system-generators',
  } ->

    #
    # Incorporar unidad al estado de la máquina
    #
  exec { "systemctl daemon-reload && systemctl start $quadlet_name":
    path => '/usr/bin',
  }

}
```

Red Podman

Manifiesto que define redes virtuales entre contenedores podman. Estas redes están descritas en la implementación.

```
 define virt::podman_network (
 String $id
){
    #
    # Red OpenNebula
    #
 virt::podman_unit { "pod_one_$id.network":
   args => {
     podman_network_entry => {
       'Driver' => 'macvlan',
       'Gateway' => "192.168.$id.254",
       'IPRange' => "192.168.$id.0/25",
       'NetworkName' => "pod_one_$id",
       'Options' => 'parent=br_one',
       'Subnet' => "192.168.$id.0/24",
     },
   }
 }

    #
    # Red Ceph
    #
 virt::podman_unit { "pod_ceph_$id.network":
   args => {
   podman_network_entry => {
       'Driver' => 'macvlan',
       'Gateway' => "192.168.$id.254",
       'IPRange' => "192.168.$id.0/25",
       'NetworkName' => "pod_ceph_$id",
       'Options' => 'parent=br_ceph',
       'Subnet' => "192.168.$id.0/24",

     }
   }
 }

    #
    # Red Replicación
    #
 virt::podman_unit { "pod_stor_$id.network":
   args => {
     podman_network_entry => {
       'Driver' => 'macvlan',
       'Gateway' => "192.168.30.254",
       'IPRange' => "192.168.30.0/25",
       'NetworkName' => "pod_stor_$id",
       'Options' => 'parent=br_stor',
```

```
        'Subnet' => '192.168.30.0/24',
      },
    }
  }
}
```

Aprovisionamiento de contenedores

Manifiesto que ejecuta un *script* ruby para aprovisionar un contenedor haciendo que cumpla con las relaciones establecidas en el plan de despliegue. El recurso crea un fichero *JSON* que emplea el *script* como entrada del plan de ejecución a seguir.

```
define virt::container_provision (
  String $container_name = 'test',
  String $common_name = "$name-$container_name",
  Array[Hash] $plan = []
) {
    #
    # Variables locales
    #
  $pwd = common::pwd()
  $scripts_path = "$pwd/deploy/$module_name/scripts"

    #
    # Fichero de entrada
    #
  file { "$scripts_path/$common_name.json":
    ensure => 'file',
    content => epp('virt/container_provision', {
      'container_name' => $container_name,
      'plan'     => $plan,
      'magic'          => $name
    })
  }

    #
    # Ejecutar script de aprovisionamiento
    #
  ensure_resource ('exec', $common_name, {
    require => File["$scripts_path/$common_name.json"],
    path => "/usr/bin",
       command  =>  "ruby  $scripts_path/container_provision.rb  $scripts_path/
$common_name.json $name",
    logoutput => true,
        timeout => 0
  })
}
```

Archivo de contenedor

Manifiesto que crea un fichero tipo contenedor, *Containerfile*, y construye la imagen (*build*).

```
  define virt::container_file (
  String $from = '',
  Array[String] $run = [],
  Array[String] $cmd = ['/bin/sh'],
  Array[String] $cp = []
    String $file_templates_path = lookup('deployment_paths::file_templates'),
) {
  $image_path = "$file_templates_path/Containerfile.$name"
  # ensure file with content
  file { $image_path:
    ensure => present,
    content => epp('virt/containerfile', {
      'from' => $from,
      'cp' => $cp,
      'run' => $run,
      'cmd' => $cmd
    })
  }->

  # build image
  exec { "build $name container image":
    path => '/usr/bin',
    command => "podman build -f $image_path -t $name"
  }
}
```

Gateway

El tráfico hacia el exterior es procesado por una serie de reglas de red. En este caso, se han establecido reglas de retransmisión de paquetes entre VLAN como puede verse en la Listado 9. Este router es un contenedor *Alpine Linux* con las reglas establecidas ya que no hay más necesidades de enrutamiento que las mencionadas. Este contenedor se acopla a los bridge que simulan las redes públicas internas de cada entidad.

```
iptables -t nat -A POSTROUTING -o %network[:podman][:interface]% -j MASQUERADE
iptables -t nat -A POSTROUTING -o $virt::containers::iface_one_10 -j
MASQUERADE
iptables -t nat -A POSTROUTING -o $virt::containers::iface_one_20 -j
MASQUERADE
iptables -A FORWARD -i $virt::containers::iface_one_10 -o
$virt::containers::iface_one_20 -j ACCEPT
iptables -A FORWARD -i $virt::containers::iface_one_20 -o
$virt::containers::iface_one_10 -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Listado 9: Reglas de IPTables para retransmisión de paquetes.

```
class virt::services::gateway
 {
    #
    # Variables locales
    #
  $nat_service = [
    "apk add iptables",
      "iptables -t nat -A POSTROUTING -o %network[:podman][:interface]% -j
MASQUERADE",
      "iptables -t nat -A POSTROUTING -o $virt::containers::iface_one_10 -j
MASQUERADE", # for cephadm
      "iptables -t nat -A POSTROUTING -o $virt::containers::iface_one_20 -j
MASQUERADE", # for cephadm
         # For the federation, vlans 10 and 20 need to be able to communicate
between each other
               "iptables -A FORWARD -i $virt::containers::iface_one_10 -o
$virt::containers::iface_one_20 -j ACCEPT",
               "iptables -A FORWARD -i $virt::containers::iface_one_20 -o
$virt::containers::iface_one_10 -j ACCEPT",
        "iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT"
  ]

    #
    # Definición del contenedor
    #
  virt::podman_unit { "gateway.container":
    args => {
      unit_entry => {
        'Description' => "gateway container "
      },
      container_entry => {
```

```
        'AddCapability' => 'NET_ADMIN NET_RAW',
        'ContainerName' => 'gateway',
        'Exec' => 'sleep infinity',
        'Image' => 'docker.io/library/alpine:3.22',
        'Network' => 'podman',
        'IP' => '10.88.0.144',
        'Sysctl' => 'net.ipv4.ip_forward=1'
      },
      install_entry => {
        'WantedBy' => 'multi-user.target'
      },
      service_restart => true,

    }
  }->
    #
    # Aprovisionamiento y puesta en marcha
    #
  virt::container_provision { "gateway_provision":
    container_name => "gateway",
    plan => [
      {
            # Configuración de red
        "type" => "container",
        "actions" => [
          "podman network connect --ip 192.168.10.254 pod_one_10 gateway",
          "podman network connect --ip 192.168.20.254 pod_one_20 gateway"
        ]
      },
      {
            # Puesta en marcha
        "type" => "guest",
                    "actions"   =>   $virt::containers::guest_vlan_one_20   +
$virt::containers::guest_vlan_one_10 + $nat_service
      }
    ]
  }
}
```

OpenNebula

Manifiesto que despliega el software de *OpenNebula* en un modo concreto de entre los disponibles: *ha*, para desplegar tres núcleos formando un clúster de alta disponibilidad; *slave_leader*, para desplegar un solo núcleo y crear una zona esclava cuya maestra es la especificada en *zone_id*; y *slave_followers*, para desplegar dos núcleos en la zona *zone_id*.

Los datastores solo se configuran cuando se configura la zona en modo de alta disponibilidad.

```
  define virt::services::opennebula(
  String $id,
    Enum['ha', 'slave_leader', 'slave_followers'] $mode = 'ha',
    String $zone_id,
    Optional[Hash] $leader = undef,
    Optional[String] $master = undef
){
  ensure_resource('virt::services::opennebula::kvm_node', "kvm_node", {})

    # Deploy all ha nodes
    if $mode == 'ha' {
        $virt::containers::oned_services[$id].each |$oned| {
            virt::services::opennebula::db { "one-db-${oned[hostname]}-$id":
                fed_id => $id,
                server_name => $oned[hostname]
            }->
            virt::services::opennebula::oned { "${oned[hostname]}":
                require => Virt::Services::Opennebula::Kvm_node["kvm_node"],
                leader => $virt::containers::oned_services[$id][0],
                oned => $oned,
                zone_id => $zone_id,
                fed_id => $id
            }
        }
    }
    # Deploy single service in ha
    elsif $mode == 'slave_leader' {
        $oned = $virt::containers::oned_services[$id][0]
        virt::services::opennebula::db { "one-db-${oned[hostname]}-$id":
            fed_id => $id,
            server_name => $oned[hostname]
        }->
        virt::services::opennebula::oned { "${oned[hostname]}":
            require => Virt::Services::Opennebula::Kvm_node["kvm_node"],
            leader => $oned,
            oned => $oned,
            zone_id => $zone_id,
            fed_id => $id,
            ha => false
        }
    }
    # Deploy nodes to attach to HA cluster
```

```
    else {
        $virt::containers::oned_services[$id].each |$oned| {
            if $oned[role] == 'follower' {
               virt::services::opennebula::db { "one-db-${oned[hostname]}-$id":
                    fed_id => $id,
                    server_name => $oned[hostname]
                }->
                virt::services::opennebula::oned { "${oned[hostname]}":
                   require => Virt::Services::Opennebula::Kvm_node["kvm_node"],
                    leader => $leader,
                    oned => $oned,
                    fed_id => $id,
                    zone_id => $zone_id,
                    master_ip => $virt::containers::one_vip[$master]
                }
            }
        }

    }

    #
    # Configure Ceph datastore when all nodes have attached to the cluster
    #
    if $mode == 'ha' or $mode == 'slave_followers' {
        virt::services::opennebula::ceph { "nebula-ceph-$id":
                 require => $virt::containers::oned_services[$id].map |$oned|
{ Virt::Services::Opennebula::Oned[$oned[hostname]] },
                frontend_nodes => $virt::containers::oned_services[$id].map |
$oned| { $oned[hostname] },
            fed_id => $id
        }
    }
}
```

Base de datos

Despliega un contenedor que ejecuta MariaDB. Crea la base de datos para *OpenNebula* y guarda sus ficheros en el directorio donde se ha montado el volumen de contenedor que lo conecta con la imagen de RBD de Ceph.

```
  define virt::services::opennebula::db (
    String $fed_id,
  String $server_name
) {
  # Container declaration

  $env = $facts['env_vars']
    $id = split($server_name, '_')[1]
  $container_name = "one-db-$id-$fed_id"
  $mountpoint = "/opt/$container_name"


  $sql_query_base = "mariadb -u root -p${env['MARIADB_ROOT_PASSWD']} -e \""
  $sql_query_end = ";\""
    $db_pass = $env['MARIADB_ONE_PASSWD']

    if $fed_id == '10' {
        $guest_vlan = $virt::containers::guest_vlan_one_10
    } else {
        $guest_vlan = $virt::containers::guest_vlan_one_20
    }

  $db_setup = [
    "$sql_query_base DROP DATABASE IF EXISTS opennebula$sql_query_end",

     "$sql_query_base CREATE USER IF NOT EXISTS oneadmin@'192.168.$fed_id.%'
IDENTIFIED BY '$db_pass'$sql_query_end",

    "$sql_query_base CREATE DATABASE opennebula$sql_query_end",

   "$sql_query_base GRANT ALL PRIVILEGES ON opennebula.* TO 'oneadmin'@'192.168.
$fed_id.%'$sql_query_end",

         "$sql_query_base SET GLOBAL TRANSACTION ISOLATION LEVEL READ
COMMITTED$sql_query_end"
  ]


  #
  # Prepare rbd device on the host for attaching opennebula db
  #
  file { $mountpoint:
    ensure => 'directory',
    mode => "0755",
    owner => "dnsmasq",
    group => "systemd-journal"
```

```
  } ->
  storage::ceph::rbd { $container_name:
    cluster_name => "ceph-$fed_id",
    pool => "one-db-$fed_id",
    image => "db-$id",
    size => "2G",
    mountpoint => $mountpoint,
  }->
  virt::podman_unit { "$container_name.container":
    args => {
      unit_entry => {
        'Description' => "one_db $id-$fed_id container"
      },
      container_entry => {
        'AddCapability' => 'NET_ADMIN',
        'ContainerName' => $container_name,
        'Image' => 'docker.io/library/mariadb:latest',
        'Network' => "pod_one_$fed_id",
        'IP' => $virt::containers::one_db_ip[$fed_id][$id],
      'Environment' => ["MARIADB_ROOT_PASSWORD=${env['MARIADB_ROOT_PASSWD']}"],
        'Volume' => "$mountpoint:/var/lib/mysql"
      },
      install_entry => {
        'WantedBy' => 'multi-user.target'
      },
      service_restart => true
    }
  }->

  # Container provisioning
  virt::container_provision { "${container_name}-provision":
    container_name => $container_name,
    plan => [
      {
        "type" => "container",
        "actions" => [
           # wait for mariadb to be ready
          "%expect[0.0.0.0:3306]% podman exec $container_name ss -tulpn",
        ]
      },
      {
        "type" => "guest",
        "actions" => $guest_vlan + $db_setup
      }
    ]
  }

}
```

Frontal *OpenNebula*

Manifiesto que despliega un núcleo de *OpenNebula*. Se conecta a la base de datos que le corresponda. En la parte de aprovisionamiento es donde juega un papel crucial el recurso de aprovisionamiento. De no ser por este, se hubiese ralentizado el despliegue al tener que ejecutar constantemente recursos tipo *exec* de Puppet. De este modo, con la flexibilidad que ofrece un *script*, se puede ejecutar el plan de acciones pudiendo acceder a *APIs* o librerías internas que agilizan el despliegue.

```
  define virt::services::opennebula::oned (
     Array[Hash]  $kvm_nodes  =  [{'hostname'  =>  'node1',  'ipaddress'  =>
'10.0.13.71'}],
  Hash $leader,
  Hash $oned,
  String $fed_id,
    String $zone_id,
    Boolean $ha = true,
    Optional[String] $master_ip = undef,
){
  # Container declaration

  # quitar los session de pam
  # pasar a root y oneadmin la clave de /var/lib/one/.ssh/id_rsa.pub como root
  $env = $facts['env_vars']
  $imageName = "almalinux_oned"
    $deploy_id = split($oned[hostname], '_')[1]
    $db_passwd = $env['MARIADB_ONE_PASSWD']
  if $fed_id == "10" {
    $connect_to_internet = [
                    "ip   route   add   default   via   192.168.10.254   dev
$virt::containers::vlan_iface_one_10"
    ]
    $guest_vlan = $virt::containers::guest_vlan_one_10
  } else {
    $connect_to_internet = [
                    "ip   route   add   default   via   192.168.20.254   dev
$virt::containers::vlan_iface_one_20"
    ]
    $guest_vlan = $virt::containers::guest_vlan_one_20
  }

  $db_config = @(END)
DB = [  BACKEND = "mysql",
    SERVER  = <%= $ipaddress %>,
    PORT    = 0,
    USER    = "oneadmin",
    PASSWD  = "<%= $passwd %>",
    DB_NAME = "opennebula",
    CONNECTIONS = 25,
    COMPARE_BINARY = "no" ]
  | - END
```

```
  ensure_resource('Virt::Container_file', $imageName, {
    from => 'docker.io/library/almalinux:9',
    cp => [
      "./opennebula.repo /etc/yum.repos.d/opennebula.repo"
    ],
    run => [
    #
    # Opennebula installation
    #
     "dnf update -y && dnf install -y sudo net-tools iproute 'dnf-command(config-
manager)'",
      "dnf config-manager --set-enabled crb",
      "dnf clean all",
      "dnf makecache",
      "dnf -y install epel-release",
      "dnf makecache",
      "groupadd sudo",
      "dnf -y install mariadb opennebula",
      "usermod -a -G sudo oneadmin",
    #
    # Ceph tools installation
    #
         "curl  --silent  --remote-name  https://download.ceph.com/rpm-reef/el9/
noarch/cephadm",
      "chmod +x cephadm",
      "./cephadm add-repo --version 19.2.2", # se fija la versión por problema
openssl v3.4
      "./cephadm install ceph-common"
    ]
  })
    #
    # Definición del contenedor
    #
  virt::podman_unit { "${oned[hostname]}.container":
    require => Virt::Container_file[$imageName],
    args => {
      unit_entry => {
        'Description' => '${oned[hostname]} container'
      },
      container_entry => {
        'AddCapability' => 'NET_ADMIN NET_RAW IPC_LOCK',
        'ContainerName' => $oned[hostname],
        'Image' => $imageName,
        'Network' => "pod_one_$fed_id",
        'Exec' => 'sleep infinity',
        'IP' => $oned[ipaddress],
        'Label' => '\"one\"',
        'HostName' => $oned[hostname]
      },
      install_entry => {
        'WantedBy' => 'multi-user.target'
```

```
      },
      service_restart => true
    }
  }->

  # Container provisioning
  virt::container_provision { "${oned[hostname]}_provision":
    container_name => $oned[hostname],
    plan => [
      {
            # Parche de systemd en contenedor
        "type" => "container",
        "actions" => [
                    "podman  cp  $virt::containers::scripts_path/mock_service.sh
${oned[hostname]}:/bin/service"
        ]
      },
      {
            # Configuración de red
        "type" => "guest",
        "actions" => $guest_vlan + $connect_to_internet
      },
      {
            # oned.conf
        "type" => "guest_file",
        "actions" => [
          {
            'path' => '/etc/one/oned.conf',
            'replace' => {
              'src' => 'DB = [ BACKEND = "sqlite",
           TIMEOUT = 2500 ]',
            'dest' => inline_epp($db_config, {'passwd' => $db_passwd, 'ipaddress'
=> $virt::containers::one_db_ip[$fed_id][$deploy_id]})
            }
          },
          {
            'path' => '/etc/one/oned.conf',
            'replace' => {
              'src' => 'VM_RESTRICTED_ATTR = "RAW/DATA"',
              'dest' => '#VM_RESTRICTED_ATTR = "RAW/DATA"',
            }
          },
          {
            'path' => '/var/lib/one/.one/one_auth',
            'content' => "oneadmin:oneadmin",
            'owner' => 'oneadmin:oneadmin'
          },
          {
            'path' => '/etc/sudoers',
            'append' => '%sudo ALL=(ALL) NOPASSWD: ALL'
          }
        ]
```

```
    },

    #
    # Leader initial setup
    #

    #
    # Start oned daemon and add server in standalone mode
    #
    {
      "type" => "guest",
      "actions" => [
        "%reload% chmod +x /bin/service",
        "su oneadmin - -c \"one start\"",
      ]
    },
    {
      "type" => "container",
      "actions" =>
        [
        "%expect[0.0.0.0:2633]% podman exec ${oned[hostname]} ss -tulpn",
        ]
    },
    {
      "type" => "guest",
      "onlyif" => $ha and $oned[role] == 'leader',
      "actions" => [
         "onezone server-add $zone_id --name ${oned[hostname]} --rpc http://
${oned[ipaddress]}:2633/RPC2",
        "one stop"
      ]
    },

    #
    # Follower setup
    #
    {
      "type" => "container",
      "onlyif" => $ha and $oned[role] == 'follower',
      "actions" => [
                  "podman exec ${oned[hostname]} one stop",
        "%expect[XML-RPC server stopped]% podman exec ${oned[hostname]} cat /
var/log/one/oned.log", # ensure it is stopped
                      "podman exec ${leader[hostname]} onedb backup -S
${virt::containers::one_db_ip[$fed_id][$deploy_id]} -t mysql -u oneadmin -p
$db_passwd -d opennebula -f /var/lib/one/nebula_backup.sql",
                "podman cp ${leader[hostname]}:/var/lib/one/nebula_backup.sql
${oned[hostname]}:/tmp",
        "podman exec ${oned[hostname]} rm -rf /var/lib/one/.one",
         "podman cp ${leader[hostname]}:/var/lib/one/.one ${oned[hostname]}:/
var/lib/one",
            "podman exec ${oned[hostname]} chown -R oneadmin:oneadmin /var/
```

```
lib/one/.one",
        "podman exec ${oned[hostname]} onedb restore -f -u oneadmin -p $db_passwd
-d opennebula -S ${virt::containers::one_db_ip[$fed_id][$deploy_id]} -t mysql /
tmp/nebula_backup.sql",
           "%expect[<STATE>3</STATE>]% podman exec ${leader[hostname]} onezone
show -x $zone_id", # if no leader is found the command below fails
           "podman exec ${leader[hostname]} onezone server-add $zone_id --name
${oned[hostname]} --rpc http://${oned[ipaddress]}:2633/RPC2"
        ]
      },

      #
      # HA configuration
      #
      {
        "type" => "guest_file",
                "onlyif" => $ha,
        "actions" => [
          {
            'path' => '/etc/one/oned.conf',
            'replace' => {
              'src' => 'SERVER_ID     = -1',
              'dest' => "SERVER_ID     = ${oned[id]}"
            }
          },
          {
            'path' => '/etc/one/monitord.conf',
            'replace' => {
              'src' => 'MONITOR_ADDRESS = "auto"',
                                              'dest'    =>    "MONITOR_ADDRESS    =
\"${virt::containers::one_vip[$fed_id]}\""
            }
          },
          {
            'path' => '/etc/one/oned.conf',
                 'append'  =>  inline_epp($virt::containers::raft_leader_hooks,
{'virtual_ip' => $virt::containers::one_vip[$fed_id], 'id' => $fed_id})
          }
        ]
      },

            #
            # Add additional config if in slave zone
            #
            {
        "type" => "guest_file",
                "onlyif" => $zone_id != undef and $master_ip != undef,
        "actions" => [
          {
            'path' => '/etc/one/oned.conf',
            'replace' => {
                           'src' => "\"STANDALONE\"",
```

```
                        'dest' => "\"SLAVE\""
            }
          },
                    {
                        'path' => '/etc/one/oned.conf',
                        'replace' => {
                            'src' => "ZONE_ID       = 0",
                            'dest' => "ZONE_ID       = $zone_id"
                        }
                    },
                    {
                        'path' => '/etc/one/oned.conf',
                        'replace' => {
                            'src' => "MASTER_ONED   = \"\"",
                           'dest' => "MASTER_ONED   = \"http://$master_ip:2633/
RPC2\""
                        }
                    },
                ]
      },
      {
        "type" => "guest",
                "onlyif" => $ha,
        "actions" => ["su oneadmin - -c \"one start\""]
      },

      # wait for server to become available & there is a leader
      {
        "type" => "container",
                "onlyif" => $ha,
        "actions" =>
          [
          "%expect[0.0.0.0:2633]% podman exec ${leader[hostname]} ss -tulpn",
           "%expect[<STATE>3</STATE>]% podman exec ${leader[hostname]} onezone
show -x $zone_id"
          ]
      },



      #
      # Distribute ssh key to nodes
      #
      {
        "type" => "container",
        "actions" => [
           "podman cp ${oned[hostname]}:/var/lib/one/.ssh/id_rsa.pub /var/lib/
one/id_rsa.pub",
          "cat /var/lib/one/id_rsa.pub >> /var/lib/one/.ssh/authorized_keys",
          "rm /var/lib/one/id_rsa.pub"
        ],
      },
```

```
      #
      # Distribute ssh key to frontal
      #
      {
        "type" => "container",
        "actions" => [
          "podman cp ${oned[hostname]}:/var/lib/one/.ssh/authorized_keys /var/
lib/one/authorized_keys",
          "cat /var/lib/one/.ssh/id_rsa.pub >> /var/lib/one/authorized_keys",
           "podman cp /var/lib/one/authorized_keys ${oned[hostname]}:/var/lib/
one/.ssh/authorized_keys ",
            "podman exec ${oned[hostname]} chown oneadmin:oneadmin /var/lib/
one/.ssh/authorized_keys",
          "rm /var/lib/one/authorized_keys"
        ],
      },

      #
      # Add KVM Hosts
      #
      {
        "type" => "guest",
        "onlyif" => $oned[role] == 'leader',
       "actions" => $kvm_nodes.map |$node| { "onehost create ${node[ipaddress]}
-i kvm -v kvm" }
      },

      #
      # Ceph definition
      #

      {
        "type" => "guest_file",
        "actions" => [
          {
                                                    'path'    =>    "/etc/ceph/
${storage::ceph::vars::cluster_name[$fed_id]}.conf",
            'content' => epp('storage/ceph_conf', {
              'config' => {
                'global' => {
                  'fsid' => $storage::ceph::vars::fsid[$fed_id],
                     'mon_host' => $storage::ceph::vars::network[$fed_id][mon]
[0][ipaddress],
                  'rbd_default_format' => 2
                }
              }
            })
          }
        ]
      }
    ]
```

```
    }
}
```

Datastores

Manifiesto que crea un datastore en *OpenNebula*. En *OpenTofu* también se crean datastores pero este manifiesto era necesario para poder desplegar primero los núcleos.

```
  define virt::services::opennebula::datastore (
  String $one_frontend,
  Enum['ceph'] $mad,
  Enum['system', 'image'] $type,
  String $bridge_list,
  Optional[Hash] $options = undef
) {
  $ds_file = "/tmp/one-ds-$name.txt"
  if $type == 'system' {
    $type_ = 'SYSTEM_DS'
  } else {
    $type_ = 'IMAGE_DS'
  }
  #
  # Create ds file
  #
  file { $ds_file:
    ensure => 'present',
    content => epp('virt/one_ds', {
      'name' => $name,
      'mad' => $mad,
      'type' => $type_,
      'bridge_list' => $bridge_list,
      'options' => $options
    })
  }->

  #
  # Copy file to destination (podman cmd since ssh might fail)
  #
  exec { $ds_file:
    command => "/usr/bin/podman cp $ds_file $one_frontend:$ds_file"
  }->

  #
  # Apply ds manifest
  #
  exec { "/usr/bin/podman exec $one_frontend onedatastore create $ds_file":
    require => Exec[$ds_file],
    refreshonly => true
  }
}
```

Soporte para Ceph Datastore

Crea los pool en Ceph necesarios para los datastores de imagen y sistema. Despliega también los datastores y propaga la clave Ceph al registro de libvirt.

```
  define virt::services::opennebula::ceph(
  Array[String] $frontend_nodes,
  String $fed_id
){

  #
  # Cluster setup for OpenNebula
  #
  $cluster_name = $storage::ceph::vars::cluster_name[$fed_id]

  #
  # Create `one` pool
  #
  storage::ceph::pool { "one-$fed_id":
    type => 'replicated',
    pg_num => "128",
    pg_autoscale => 'on',
    cluster_name => $cluster_name
  }

  #
  # Define `libvirt` ceph user
  #
  storage::ceph::user { "libvirt-$fed_id":
    type => 'client',
    cluster_name => $cluster_name,
    caps => "mon 'profile rbd' osd 'profile rbd pool=one-$fed_id'"
  }

  # One Nodes need `qemu-img` cli to work with ceph
  # (For now this is a one node deployment)
  #
  ensure_resource('package', "qemu-utils", {
    ensure => 'installed'
  })

  #
  # Place keyring and keyfile to frontend node
  #
  exec { "libvirt-key-$fed_id":
    command => "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph -- ceph
auth get-key client.libvirt-$fed_id -o /etc/ceph/client.libvirt-$fed_id.key",
    require => Storage::Ceph::User["libvirt-$fed_id"]
  }
  exec { "libvirt-keyring-$fed_id":
       command => "/usr/sbin/cephadm shell -m /etc/$cluster_name:/etc/ceph
--  ceph  auth  get  client.libvirt-$fed_id  -o  /etc/ceph/ceph.client.libvirt-
```

```
$fed_id.keyring",
    require => Storage::Ceph::User["libvirt-$fed_id"]
  }
  $frontend_nodes.each |$fnode| {
    exec { "$fnode-libvirt-key-$fed_id":
      require => Exec["libvirt-key-$fed_id"],
          command  =>  "/usr/bin/podman  cp  /etc/$cluster_name/client.libvirt-
$fed_id.key $fnode:/var/lib/one"
    }->
     exec { "/usr/bin/podman exec $fnode chown oneadmin:oneadmin /var/lib/one/
client.libvirt-$fed_id.key":
    }
    exec { "$fnode-libvirt-keyring-$fed_id":
      require => Exec["libvirt-keyring-$fed_id"],
       command => "/usr/bin/podman cp /etc/$cluster_name/ceph.client.libvirt-
$fed_id.keyring $fnode:/etc/ceph"
    }->
      exec { "/usr/bin/podman exec $fnode chown oneadmin:oneadmin /etc/ceph/
ceph.client.libvirt-$fed_id.keyring":
    }
  }

  $libvirt_secret = storage::uuid()
  #
  # Datastores setup
  #

  #
  # System Datastore
  #
  virt::services::opennebula::datastore { "system-$fed_id":
    require => Package["qemu-utils"],
    one_frontend => $virt::containers::oned_services[$fed_id][0][hostname],
    mad => 'ceph',
    type => 'system',
    bridge_list => '10.0.13.71',
    options => {
      'pool_name' => "one-$fed_id",
     'ceph_host' => join(['"'] + $storage::ceph::vars::network[$fed_id][mon].map
|$mon| { $mon[ipaddress] } + ['"'], ' '),
      'ceph_user' => "libvirt-$fed_id",
      'ceph_secret' => $libvirt_secret,
      'ceph_conf' => "/etc/$cluster_name/ceph.conf",
      'ceph_key' => "/etc/$cluster_name/client.libvirt-$fed_id.key"
    }
  }

  #
  # Image Datastore
  #
  virt::services::opennebula::datastore { "image-$fed_id":
    require => Package["qemu-utils"],
```

```
    one_frontend => $virt::containers::oned_services[$fed_id][0][hostname],
    mad => 'ceph',
    type => 'image',
    bridge_list => '10.0.13.71',
    options => {
      'pool_name' => "one-$fed_id",
     'ceph_host' => join(['"'] + $storage::ceph::vars::network[$fed_id][mon].map
|$mon| { $mon[ipaddress] } + ['"'], ' '),
      'ceph_user' => "libvirt-$fed_id",
      'ceph_secret' => $libvirt_secret,
      'ceph_conf' => "/etc/$cluster_name/ceph.conf",
      'ceph_key' => "/etc/$cluster_name/client.libvirt-$fed_id.key"
    }
  }

  #
  # Hosts setup
  #

  #
  # Install ceph utils
  #
  ensure_resource('exec', "ceph-utils host install", {
    command => "/usr/sbin/cephadm install ceph-common"
  })

  #
  # Define libvirt secret in qemu
  #
  $secret_xml = @(END)
  <secret ephemeral='no' private='no'>
    <uuid><%= $secret %></uuid>
    <usage type='ceph'>
      <name>client.libvirt-<%= $fed_id %> secret</name>
    </usage>
  </secret>
  | - END
  file { "/tmp/libvirt_secret_$fed_id.xml":
     content => inline_epp($secret_xml, {'secret' => $libvirt_secret, 'fed_id'
=> $fed_id}),
    ensure => 'present'
  }
  exec { "remove-secrets-$fed_id":
         command  =>  "/usr/bin/virsh  -c  qemu:///system  secret-undefine  --
secret \"\$(/usr/bin/virsh -c qemu:///system secret-list | /usr/bin/awk '/ceph
client.libvirt-$fed_id secret/ {print \$1}')\"",
    require => File["/tmp/libvirt_secret_$fed_id.xml"],
    refreshonly => true,
      onlyif  =>  "/usr/bin/virsh  -c  qemu:///system  secret-list  |  grep  \"ceph
client.libvirt-$fed_id secret\""
  }
      exec   {   "/usr/bin/virsh   -c   qemu:///system   secret-define   /tmp/
```

```
libvirt_secret_$fed_id.xml":
     require => [Exec["remove-secrets-$fed_id"], Exec["libvirt-key-$fed_id"],
Exec["libvirt-keyring-$fed_id"]],
    refreshonly => true,
  }->
    exec { "/usr/bin/virsh -c qemu:///system secret-set-value --secret
$libvirt_secret --base64 \$(/usr/bin/cat /etc/$cluster_name/client.libvirt-
$fed_id.key)":
    refreshonly => true,
  }
}
```

Nodo KVM

Configura e instala paquetes necesarios para desplegar el demonio que conecta el nodo KVM con los núcleos de *OpenNebula*.

```
define virt::services::opennebula::kvm_node {
  #
  # Add opennebula-node-kvm package
  #
  package { "opennebula-node-kvm":
    ensure => 'installed'
  }->
  file { "/var/lib/one":
    ensure => 'directory',
    owner => 'oneadmin',
    group => 'oneadmin'
  }->
  file { "/var/lib/one/.ssh":
    ensure => 'directory',
    recurse => true,
    purge => true,
    force => true
  }
  exec { "/usr/bin/ssh-keygen -t rsa -f /var/lib/one/.ssh/id_rsa -b 4096 -N ''
-q":
    refreshonly => true,
    subscribe => File["/var/lib/one/.ssh"]
  }->
  exec { "/usr/bin/chown oneadmin:oneadmin /var/lib/one/.ssh":
  }->
  exec {"/usr/bin/systemctl restart libvirtd":
  }

  #
  # Set appropriate apparmor profiles
  #
  file_line { "one node apparmor profile":
    path => '/etc/apparmor.d/abstractions/libvirt-qemu',
    line => '/var/lib/one/datastores/** rwk,',
    ensure => 'present'
  }

  #
  # Networking is bridged and over the kvm bridge already defined
  #
}
```

Despliegue Zona maestra de la federacion en HA

```
  define virt::services::opennebula::fed_masters_setup (
    String $masters_vip,
    String $master_container,
    String $masters_id
) {

    #
    # Configure zone endpoint
    #
    # Change each master node from standalone to master
    #
  $virt::containers::oned_services[$masters_id].each |$master_node| {
    virt::container_provision { "${master_node[hostname]}_second_provision":
      container_name => $master_node[hostname],
      plan => [
        {
                # Inicio federación
          "type" => "guest_file",
          "actions" => [
            {
              'path' => '/etc/one/oned.conf',
              'replace' => {
                'src' => "\"STANDALONE\"",
                'dest' => "\"MASTER\""
              }
            }
          ]
        },
                {
                #
                    "type" => "guest_file",
                    "onlyif" => $master_node[hostname] == $master_container,
                    "actions" => [
                        {
                            'path' => '/tmp/update_zone.one',
                            'content' => "ENDPOINT=\"http://$masters_vip:2633/
RPC2\""
                        }
                    ]
                },
                {
                    "type" => "guest",
                    "onlyif" => $master_node[hostname] == $master_container,
                    "actions" => ["onezone update 0 /tmp/update_zone.one"]
                },
        {
          "type" => "guest",
          "actions" => ["su oneadmin - -c \"one restart\""]
        },
                {
```

```
                  "type" => "container",
                  "actions" =>
                ["%expect[0.0.0.0:2633]% podman exec ${master_node[hostname]}
ss -tulpn"]
              }
      ]
    }
  }
}
```

Despliegue de HA

Manifiesto que modifica un núcleo en una zona formando un clúster de alta disponibilidad entre los núcleos de esa zona, eventualmente. Toma como parámetros la información del núcleo, el identificador de la zona y la IP flotante.

```
  define virt::services::opennebula::ha (
    Hash $oned,
    String $zone,
    String $vip
) {
    virt::container_provision { "${oned[hostname]}-ha":
        container_name => $oned[hostname],
        plan => [
            {
        "type" => "guest",
        "onlyif" => $oned[role] == 'leader',
        "actions" => [
             "onezone server-add $zone --name ${oned[hostname]} --rpc http://
${oned[ipaddress]}:2633/RPC2",
          "one stop"
        ]
      },
            {
                    "type" => "guest_file",
                    "actions" => [
                        {
                            'path' => '/etc/one/oned.conf',
                            'replace' => {
                                'src' => 'SERVER_ID     = -1',
                                'dest' => "SERVER_ID     = ${oned[id]}"
                            }
                        },
                        {
                            'path' => '/etc/one/monitord.conf',
                            'replace' => {
                                'src' => 'MONITOR_ADDRESS = "auto"',
                                'dest' => "MONITOR_ADDRESS = \"$vip\""
                            }
                        },
                        {
                            'path' => '/etc/one/oned.conf',
                    'append' => inline_epp($virt::containers::raft_leader_hooks,
{'virtual_ip' => $vip, 'id' => '20'})
                        },
                        {
                            'path' => '/tmp/update_zone.one',
                            'content' => "ENDPOINT=\"http://$vip:2633/RPC2\""
                        }
                    ]
                },
                {
```

```
                    "type" => "guest",
                    "actions" => ["su oneadmin - -c \"one start\""]
                },

                # wait for server to become available & there is a leader
                {
                    "type" => "container",
                    "actions" =>
                        [
                         "%expect[0.0.0.0:2633]% podman exec ${oned[hostname]}
ss -tulpn",
                       "%expect[<STATE>3</STATE>]% podman exec ${oned[hostname]}
onezone show -x $zone"
                        ]
                },
                {
                    "type" => "guest",
                    "actions" => ["onezone update $zone /tmp/update_zone.one"]
                }
        ]

    }
}
```

Zona

Manifiesto que crea una nueva zona en un despliegue de *OpenNebula*.

```
  define virt::services::opennebula::zone (
    String $slave_id,
    String $slave_endpoint,
    String $master
) {
    file { "/tmp/${slave_id}-zone.tmpl":
      ensure => 'present',
      content => "NAME = zone-$slave_id
      ENDPOINT = http://$slave_endpoint:2633/RPC2"
    }->
        exec  {"/usr/bin/podman  cp  /tmp/${slave_id}-zone.tmpl  $master:/tmp/
${slave_id}-zone.tmpl":
            refreshonly => true
    }->
   exec { "/usr/bin/podman exec $master onezone create /tmp/${slave_id}-zone.tmpl
> /tmp/${slave_id}-zone-id":
            refreshonly => true
    }
}
```

Control del estado de zona esclava

```
  define virt::services::opennebula::stop_slaves (
    Array[String] $slaves
) {
  #
  # Stop slaves One in HA
  #
  $slaves.each |$slave| {
        exec { "/usr/bin/podman exec ${virt::containers::oned_services[$slave]
[0][hostname]} one stop":
            refreshonly => true
        }
  }
}
```

Monitorización

Manifiesto que despliega los exportadores de métricas en los núcleos de una zona y sus nodos de virtualización.

```
  define virt::services::opennebula::monitoring (
    String $zone_id = "10"
) {

    #
    # KVM Node installation
    #
    ensure_resource('Package', "opennebula-prometheus-kvm", {
        ensure => "installed"
    })

   ensure_resource('Exec', "/usr/bin/systemctl enable --now opennebula-libvirt-
exporter.service", {
       require => Package["opennebula-prometheus-kvm"],
       refreshonly => true
    })

    #
    # Frontal Node installation
    #
    $virt::containers::oned_services[$zone_id].each |$oned| {
       exec { "/usr/bin/podman exec ${oned[hostname]} dnf install -y opennebula-
prometheus opennebula-prometheus-kvm":
        }->
         exec { "/usr/bin/podman exec ${oned[hostname]} /usr/bin/ruby /usr/lib/
one/opennebula_exporter/opennebula_exporter.rb &":
        }
    }

    if $zone_id == '10' {
        #
        # Generate config
        #
        $fst_oned = $virt::containers::oned_services[$zone_id][0][hostname]
        $prometheus_folder = "/etc/one-$zone_id/prometheus"
        file { ["/etc/one-$zone_id", $prometheus_folder]:
            ensure => 'directory'
        }->
           exec { "/usr/bin/podman exec $fst_oned /usr/share/one/prometheus/
patch_datasources.rb":
            refreshonly => true
        }->
        exec { "Copy-prometheus-config-$zone_id":
                command => "/usr/bin/podman cp $fst_oned:/etc/one/prometheus/
prometheus.yml $prometheus_folder/prometheus.yml",
            refreshonly => true
        }
```

```
        #
        # Start prometheus container
        #
        $container_name = "prometheus-$zone_id"
        $guest_vlan = $virt::containers::guest_vlan_one_10
        virt::podman_unit { "$container_name.container":
            require => Exec["Copy-prometheus-config-$zone_id"],
            args => {
                unit_entry => {
                    'Description' => "$container_name container"
                },
                container_entry => {
                    'AddCapability' => 'NET_ADMIN NET_RAW IPC_LOCK',
                    'ContainerName' => $container_name,
                    'Image' => "docker.io/prom/prometheus",
                    'Network' => "pod_one_$zone_id",
                    'IP' => $virt::containers::one_prometheus_ip[$zone_id],
                         'Volume' => "$prometheus_folder/prometheus.yml:/etc/
prometheus/prometheus.yml",
                    'User' => 'root',
                    'HostName' => $container_name
                },
                install_entry => {
                    'WantedBy' => 'multi-user.target'
                },
                service_restart => true
            }
        }->
        # Container provisioning
        virt::container_provision { "$container_name-provision":
            container_name => $container_name,
            plan => [
                {
                    "type" => "guest",
                           "actions" => $guest_vlan + ["ip route add default
via 192.168.10.254"]
                }
            ]
        }
    }
}
```

F.2.2 Scripts

Emulador systemd en contenedores

```
  #!/bin/sh

if [[ "$@" == "opennebula-hem start" ]]; then
  su oneadmin - -c "ruby /usr/lib/one/onehem/onehem-server.rb &"
elif [[ "$@" == "opennebula-hem stop" ]]; then
  pkill -u oneadmin -f onehem-server.rb
else
  exit 1
fi
```

Aprovisionamiento de contenedores

Script de aprovisionamiento de contenedores. Este ofrece una interfaz por la que poder acceder al estado del contenedor en cada momento. Se puede acceder a aspectos de red y almacenamiento, lo cual resulta útil tras configurar interfaces de red.

```
  #!/usr/bin/env ruby

require "json"
require "open3"
#require 'net_http_unix'

args_path = ARGV[0]
magic = ARGV[1]
command_entries = ["container", "guest"]

# Load json
exit 0 unless File.exist?(args_path)

file = open(args_path)
parsed = file.read
params = JSON.parse(parsed)

exit 0 if `podman exec #{params["container_name"]} ls -1 /var/#{magic} 2>/dev/
null` == "/var/#{magic}\n"
`podman exec #{params["container_name"]} touch /var/#{magic}`

#
# Función que obtiene el nombre de la interfaz que tiene asignada la dirección
IP _address_ en el contenedor _cname_.
#
def get_container_addressed_interface(cname, address)
  output = `podman exec #{cname} ip a show to #{address} | cut -d ':' -f2 | head
-n1 | cut -d '@' -f1`
  output.gsub("\n", "").strip!
end

#
# Función que devuelve un diccionario con las direcciones IP asigandas a cada
interfaz en el contenedor _container_name_
#
def network_layout(container_name)
              network_inspection_cmd       =       `podman       inspect       --
format='{{json .NetworkSettings.Networks }}' #{container_name}`.gsub("\n", "")

  if network_inspection_cmd == ""
    Puppet.notice("No container with provided name exists")
  end

  data = JSON.parse(network_inspection_cmd, { symbolize_names: true })
   result = data.map { |k, v| [k, { :ipaddress => v[:IPAddress], :interface =>
get_container_addressed_interface(container_name, v[:IPAddress]) }] }.to_h
```

```
  result
end

#
# Función que obtiene la información de la interfaz de red local del contenedor
# _container_name_ que está conectada a la red de contenedores _network_name_.
#
def network_layout_iface(container_name, network_name)
  data = network_layout(container_name)
  data.find { |k, v| k == network_name }
end

#
# Función que devuelve la información relativa a la red de todos los contenedores
desplegados en un host.
#
def scan_podman_services
  network_scan = `podman ps --format '{{.Names}}'`.lines.map { |container_name|
    container_name_ = container_name.gsub("\n", "")
    [container_name_, { :network => network_layout(container_name_) }]
  }.to_h
end

#
#  Función  que  devuelve  la  información  relativa  a  todos  los  contenedores
desplegados en un host.
#
def deployment_layout
  {
    :virt => {
      :services => scan_podman_services,
    },
  }
end

#
# Función que sustituye todos las variables de la API ofrecida por sus valores
reales.
#
def var_subs(src, ns)
  src.gsub(/%(\w+)(\[[^%]+\])%/) do |match|
    begin
      eval("ns['#{$1}']#{$2}")
    rescue NoMethodError
      puts ns
      puts "EVALUATING ns['#{$1}']#{$2}"
    end
  end
end

def fe_mode(file_desc)
  if file_desc[:replace]
```

```
    :fe_replace
  elsif file_desc[:content]
    :fe_content
  elsif file_desc[:append]
    :fe_append
  else
    :none
  end
end

#
# Función de utilidad para modificar, añadir o crear un fichero en el contenedor
con el contenido deseado
#
def produce_file_content(cname, file_desc)
  case fe_mode(file_desc)
  when :fe_replace
    file_content = `podman exec #{cname} cat #{file_desc[:path]}`
              file_content.gsub!    file_desc[:replace][:src].gsub(/\t/,    ""),
file_desc[:replace][:dest]
    file_content
  when :fe_content
    file_desc[:content]
  when :fe_append
   file_content = `podman exec #{cname} cat #{file_desc[:path]}`.gsub('\n', "")
    file_content + file_desc[:append]
  else
    ""
  end
end

#
# Función intermedia para modificar un fichero en un contenedor
#
def process_file_entry(cname, file_desc)
  file_content = produce_file_content(cname, file_desc)
  file_dest = "#{cname}:#{file_desc[:path]}"
  File.write("/tmp/tmp_file_entry", file_content)
  `podman cp /tmp/tmp_file_entry #{file_dest}`
   `podman  exec  #{cname}  chown  #{file_desc[:owner]}  #{file_desc[:path]}`  if
file_desc[:owner]
  File.delete("/tmp/tmp_file_entry")
end

#
# Función que sustituye todas las variables encontradas en todo el plan de
despliegue.
#
def map_var_subs(entry_map, container_namespace)
  return var_subs(entry_map, container_namespace) unless entry_map.respond_to?
(:map)
 entry_map.map { |k, v| [k.to_sym, map_var_subs(v, container_namespace)] }.to_h
```

```
end

#
# Función que devuelve el estado de los dispositivos que ve un contenedor.
#
def devices_layout(container_name, inspect)
  container_args = inspect["Config"]["CreateCommand"]
  mapped_devices = []
  (0..container_args.length - 1).each do |i|
     mapped_devices.append(container_args[i + 1]) if container_args[i] == "--
device"
  end
  # Handles full dereference for symlinks (lvm dev to device mapper dev)
     host_mapped   =   mapped_devices.map   {   |device|   [File.realpath(device),
device] }.to_h
  devices = inspect["HostConfig"]["Devices"]
          devices.map     {     |device|     [host_mapped[device["PathOnHost"]],
device["PathInContainer"]] }.to_h
end

#
# Función que genera un diccionario con el estado de red, almacenamiento y
despliegue.
#
def generate_namespace(cname)
  container_inspection = JSON.parse(`sudo podman inspect --format json #{cname}
`.gsub("\n", ""))[0]
  {
    "network" => network_layout(cname),
    "deployment" => deployment_layout,
    "devices" => devices_layout(cname, container_inspection),
  }
end

container_namespace = generate_namespace(params["container_name"])

#
# Lógica que ejecuta el plan especificado
#
params["plan"].each { |action_type|
  action_type["actions"].each do |action|
    #
    # Comprueba si debe ejecutar la siguiente acción
    #
    next unless action_type["onlyif"]

    #
    # Comprueba si tiene que recalcular el _container_namespace_.
    #
    if command_entries.include? action_type["type"]
      reloadable = false
      expecting = false
```

```
      expected_output = ""
      case action.split.first
      when "%reload%"
        reloadable = true
        action = action.split[1..-1].join(" ")
        container_namespace = generate_namespace(params["container_name"])
      when /%expect\[.*/
        expecting = true
        _, raw_expecting, action = action.split(/(%expect\[.*\]%)/)
        expected_output = raw_expecting[/\[.*\]/].delete("[]")
      end

      #
      # Ejecuta un mismo comando hasta que se complete un estado esperado o sea
de una sola ejecución.
      #
      loop do
        subs_entry = var_subs(action, container_namespace)
        stdout, stderr, status = Open3.capture3(subs_entry)
        if stderr != "" && status && status != 0
          puts "COMMAND #{subs_entry} for container #{params["container_name"]}
failed with status #{status}: #{stderr}"
          exit 1
        end

        if expecting && !stdout.include?(expected_output)
          puts "WAITING ON COMMAND: #{action} to meet #{expected_output}"
          sleep 1
        else
          break
        end
      end
    else
      # Its a file description
      next unless action != {}
      subs_entry = map_var_subs(action, container_namespace)
      process_file_entry(params["container_name"], subs_entry)
    end
  end
}

File.delete(args_path)
```

F.2.3 Funciones

Estado del despliegue

```
  require 'json'

Puppet::Functions.create_function(:'virt::network_layout') do
  dispatch :_network_layout do
    param 'String', :container_name
  end

  dispatch :_network_layout_iface do
    param 'String', :container_name
    param 'String', :network_name
  end

  def _network_layout(container_name)
                network_inspection_cmd    =    `sudo    podman    inspect    --
format='{{json .NetworkSettings.Networks }}' #{container_name}`.gsub("\n", "")
    if network_inspection_cmd == ""
      Puppet.notice("No container with provided name exists")
    end

    data = JSON.parse(network_inspection_cmd, {symbolize_names:true})
    result = data.map {| k, v | [k, {:ipaddress => v[:IPAddress], :interface =>
get_container_addressed_interface(container_name, v[:IPAddress])}] }.to_h
    result
  end

  def _network_layout_iface(container_name, network_name)
    data = _network_layout(container_name)
    data.find { |k, v| k == network_name }
  end

  def get_container_addressed_interface(cname, address)
    output = `sudo podman exec #{cname} ip a show to #{address} | cut -d ':' -
f2 | head -n1 | cut -d '@' -f1`
    output.gsub("\n", "").strip!
  end
end
```

F.2.4 Plantillas

Archivo de contenedor

```
  <%- |   String $from,
        Array[String] $run,
  Array[String] $cmd,
  Array[String] $cp
| -%>

FROM <%= $from %>

<% $cp.each |$copier| { -%>

COPY <%= $copier %>

<% } -%>
<% $run.each |$runner| { -%>

RUN <%= $runner %>

<% } -%>

CMD [<%= $cmd.map |$cmd_entry| { "\"#{$cmd_entry}\"" }.join(',')  %>]
```

```
  <%- |   String $container_name,
        Array[Hash] $plan,
      String $magic
| -%>

{
  "container_name": "<%= $container_name %>",
  "plan": [
    <% $plan.each |$action_type| { -%>
    {
      "type": "<%= $action_type["type"] %>",
        "onlyif": <%= $action_type["onlyif"] ? { undef => true, default =>
$action_type["onlyif"] } %>,
      "actions":  <% if $action_type["type"] =~ /(container|guest_file)/ { -%>

        <%= stdlib::to_json($action_type["actions"]) %>

      <% } -%>
      <%- elsif $action_type["type"] == "guest" { -%>
        <%- $transformed_actions = $action_type["actions"].map |$action| {
          $first_word = split($action, " ")[0]
          $tail = join(split($action, " ")[1,-1], " ")
          if $first_word =~ /%.+%/ {
            "$first_word podman exec $container_name $tail"
          } else {
            "podman exec $container_name $action"
          }
        } -%>
        <%= stdlib::to_json($transformed_actions) %>
      <% } -%>
    },
    <% } -%>
    { "type": "<%= $magic %>", "actions": [] }
    ]
}
```

Configuración de Datastore

```
  <%- | String $name,
       String $mad,
       String $type,
       String $bridge_list,
       Hash $options
       | -%>
NAME = <%= $name %>
TYPE = <%= $type %>
TM_MAD = <%= $mad %>
<% if $mad == 'ceph' { -%>
<% if $type == 'IMAGE_DS' { -%>
DS_MAD = ceph
<%- } -%>
DISK_TYPE = RBD

<% $options.each |$opt, $val| { -%>
<%= $opt.upcase() %> = <%= $val %>
<%- } %>
<%- } %>

BRIDGE_LIST = <%= $bridge_list %>
```

Anexo G

Planes de despliegue

Se ha empleado la herramienta Puppet Bolt para controlar el despliegue distribuido de los manifiestos definidos previamente. Para ello se han definido una serie de planes de despliegue donde se especifican qué acciones se han de ejecutar, en este caso, se compilan y envian, a través de una conexión SSH y el protocolo Puppet, los manifiestos previamente definidos; los objetivos, es decir, las máquinas en las que existe un agente Puppet que pueda ejecutar la acción enviada; y una serie de opciones adicionales, como usuario de ejecución o distintas pruebas.

Se ha empleado Hiera para definir las variables y parámetros de las clases Puppet. Así se consigue separar la información que corresponde cada entidad individualmente y agruparla en otros casos.

Plan principal

Manifiesto, referenciado en el Listado 7, que despliega los servicios virtualizados de una entidad, según se ha especificado en el diseño e implementación. Primero se comprueba que exista el directorio donde crear las unidades *systemd* de los contenedores. Se crea la red de contenedores y el clúster Ceph. Lo siguiente es crear el usuario s3 que emplea el *Marketplace*. De *OpenNebula*, lo pirmero que se despliega es la zona maestra con 3 núcleos forman el clúster en alta disponibilidad. Esto se consigue con el recurso *Puppet* personalizado *virt::services::opennebula* y el modo *ha*. Lo siguiente es desplegar un único núcleo en una zona distinta en modo *slave_leader*. Este último evento y la zona maestra son eventos concurrentes. El parámetro *zone_id* corresponde al identificador de la zona maestra. Cuando se han desplegado la zona maestra y la esclava, se despliega el recurso que federa ambas zonas. Se termina formando el clúster de alta disponibilidad en la zona esclava y desplegando los recursos de Ceph para los servicios de backup, *OPA* y monitorización. Además se corrigen algunos de los problemas encontrados y descritos en el anexo correspondiente.

```
plan cloud_fed::deploy_federation(){
  $targets = get_targets('fed_deploy')
  $scripts_path = lookup('deployment_paths::scripts')
  $templates_path = lookup('deployment_paths::file_templates')


  upload_file('virt/scripts',
      "$scripts_path/virt",
      $targets,
      _run_as => 'root'
  )
  upload_file('storage/scripts',
      "$scripts_path/storage",
      $targets,
      _run_as => 'root'
```

```
    )
    upload_file('virt/file_templates/opennebula.repo',
        "$templates_path",
        $targets,
        _run_as => 'root'
    )
    run_plan('facts', 'targets' => $targets)
    $plan_result = catch_errors() || {
        run_plan('cloud_fed::net_ifaces', 'targets' => $targets)
        run_plan('cloud_fed::storage_ceph', 'targets' => $targets)

        run_plan('cloud_fed::nebula_instances')

        run_plan('cloud_fed::nebula_federation',
            'master' => lookup('master_id'),
            'slaves' => lookup('slaves_ids')
        )

        run_plan('cloud_fed::slaves_ha', 'targets' => get_targets('slaves'))

        run_plan('cloud_fed::monitoring', 'targets' => $targets)

    }

    return {
        result => $plan_result
    }
}
```

Plan despliegue de red

```
  plan cloud_fed::net_ifaces(TargetSpec $targets) {
   $result = apply($targets, _run_as => 'root') {
        file { "/etc/containers/systemd":
            ensure => 'directory',
        }

        #
        # Deploy Podman network
        #
        virt::podman_network { "podman_network_${lookup('id')}":
            require => [
                File["/etc/containers/systemd"]
            ],
            id => lookup('id')
        }
    }
    if $result.ok() {
        run_command('sudo ip link add dummy0 type dummy', $targets)
        run_command('sudo ip link set dummy0 up', $targets)
        run_command('sudo ip link set dummy0 master br_stor', $targets)
        run_command('sudo ip link set br_stor up', $targets)

        out::message("Network interfaces deployed successfully")
    } else {
        fail_plan(
            'Network interfaces deployment was unsuccessful',
            'net_ifaces error',
            {'result' => $result.error()}
        )
    }

    return $result
}
```

Plan despliegue de Ceph

Este plan despliega un cluster Ceph y dos sitemas de ficheros CephFS en los objetivos especificados. Lo hace de forma distribuida y como el usuario *root*.

```
plan cloud_fed::storage_ceph(TargetSpec $targets) {
  $result = apply($targets, _run_as => 'root') {

        #
        # Deploy ceph clusters
        #
        storage::ceph {lookup('ceph', Hash, 'deep')[cluster_name]:
            id => lookup('id')
        }

        #
        # Deploy backup cephfs
        #
        storage::ceph::newfs { "backup-${lookup('id')}":
            require => Storage::Ceph[
                lookup('ceph', Hash, 'deep')[cluster_name]
            ],
            cluster_name => lookup('ceph', Hash, 'deep')[cluster_name],
            user_name    => lookup('ceph', Hash, 'deep')[backup][username]
        }

        #
        # Deploy policies cephfs
        #
        storage::ceph::newfs { "policies-${lookup('id')}":
            require => Storage::Ceph[
                lookup('ceph', Hash, 'deep')[cluster_name]
            ],
            cluster_name => lookup('ceph', Hash, 'deep')[cluster_name],
            user_name    => lookup('ceph', Hash, 'deep')[policies][username]
        }
    }
    if $result.ok() {
        out::message("Ceph storage cluster deployed successfully")
    } else {
        fail_plan(
            'Ceph storage cluster deployment was unsuccessful',
            'storage_ceph error',
            {'result' => $result.error()}
        )
    }

    return $result
}
```

Plan despliegue de OpenNebula

Plan que crea el usuario s3 para el *Marketplace* y despliega una primera zona maestra en alta disponibilidad. Posteriormente, en la máquina destinada a la zona esclava se despliega el líder de una nueva zona.

```
plan cloud_fed::nebula_instances() {
       $master_deploy = apply('vm0-cert', _run_as => 'root') {
           #
           # Create s3 user
           #
           storage::ceph::s3_user { "nebula":
               display_name => "Nebula S3",
               cluster_name => "ceph-${lookup('id')}"
           }

           #
           # Deploy master zone in HA
           #
           virt::services::opennebula { "master-zone":
               id => lookup('id'),
               mode => lookup('opennebula', Hash, 'deep')[ha_mode],
               zone_id => lookup('opennebula', Hash, 'deep')[zone_id]
           }
       }

       if $master_deploy.ok() {
           out::message("Master zone deployed successfully")
       } else {
           fail_plan(
               'Master zone deployment was unsuccessful',
               'nebula_instances error',
               {'result' => $master_deploy.error()}
           )
       }

       $slave_deploy = apply('vm1-cert', _run_as => 'root') {
           virt::services::opennebula { "slave-leader":
               id => lookup('id'),
               mode => lookup('opennebula', Hash, 'deep')[ha_mode][0],
               zone_id => lookup('opennebula', Hash, 'deep')[master_zone_id]
           }
       }

       if $slave_deploy.ok() {
           out::message("Slave zone deploy successfully")
       } else {
           fail_plan(
               'Slave zone deployment was unsuccessful',
               'nebula_instances error',
               {'result' => $slave_deploy.error()}
           )
```

```
        }
        return {
            master_zone => $master_deploy,
            slave_zone => $slave_deploy,
        }
}
```

Plan despliegue de federación

Manifiesto que establece la federación entre dos zonas de *OpenNebula*. Modifica la configuración de ambas zonas para iniciar el proceso de replicación maestro-esclavo.

```
  plan cloud_fed::nebula_federation(
    String $master,
    Array[String] $slaves
){
    $master_target = get_target('master')
    $slaves_targets = get_targets('slaves')
    $slave_host = get_target('vm1-cert')

    $master_fed_result = apply($master_target, _run_as => 'root') {
        $master_container = lookup('opennebula')[services][0][hostname]

        #
        # Configure master
        #
        virt::services::opennebula::fed_masters_setup { "fed_masters_setup":
            master_container => $master_container,
            masters_vip => lookup('opennebula')[vip],
            masters_id => lookup('id')
        }

        #
        # Create zones from master
        #
        $slaves.each |$slave| {
            virt::services::opennebula::zone { "zone-$slave":
                                                                    require    =>
Virt::Services::Opennebula::Fed_masters_setup["fed_masters_setup"],
                slave_id => $slave,
                        slave_endpoint => lookup('opennebula', Hash, 'deep')
[slave_endpoints][$slave],
                master => $master_container
            }
        }

        #
        # Make master db snapshot
        #
        exec { "federated-snap":
            require => Virt::Services::Opennebula::Zone["zone-20"],
                    command => "/usr/bin/podman exec $master_container onedb
backup --federated -S ${lookup('opennebula')[db]['0']} -t mysql -u oneadmin -p
${lookup('opennebula', Hash, 'deep')[db_config][one_passwd]} -d opennebula -f /
var/lib/one/one.db"
        }


        #
```

```
        # Copy files from master to slaves
        #
        $slaves.each |$slave| {
            exec { "/usr/bin/podman cp $master_container:/var/lib/one/one.db /
tmp/one.db":
                require => Exec["federated-snap"],
            }->
              exec { "/usr/bin/podman cp $master_container:/var/lib/one/.one /
tmp/one_auths":
                require => Exec["federated-snap"],
            }->
                exec { "/usr/bin/scp -o 'StrictHostKeyChecking=no' -i /home/
vagrant/.ssh/id_ecdsa -r /tmp/one_auths vagrant@${slave_host.uri()}:/tmp":
            }->
                exec { "/usr/bin/scp -o 'StrictHostKeyChecking=no' -i /home/
vagrant/.ssh/id_ecdsa /tmp/one.db vagrant@${slave_host.uri()}:/tmp":
            }
        }
    }

    $slave_fed_result = apply($slaves_targets, _run_as => 'root') {
        #
        # Stop slave zones
        #
        virt::services::opennebula::stop_slave { "stop-one-slaves":
            slave => lookup('id')
        }->
       exec { "/usr/bin/podman cp /tmp/one.db ${lookup('opennebula')[services]
[0][hostname]}:/tmp/one.db":
        } ->
        exec { "slave-${lookup('id')}-db-restore":
                    command  =>  "/usr/bin/podman  exec  ${lookup('opennebula')
[services][0][hostname]} onedb restore --federated -S ${lookup('opennebula')[db]
['0']} -t mysql -u oneadmin -p ${lookup('opennebula', Hash, 'deep')[db_config]
[one_passwd]} -d opennebula /tmp/one.db"
        }->
      exec { "/usr/bin/podman cp /tmp/one_auths ${lookup('opennebula')[services]
[0][hostname]}:/tmp/one_auths":
        }->
            exec  {  "/usr/bin/podman  exec  ${lookup('opennebula')[services][0]
[hostname]} mv -f /tmp/one_auths/one_auth /var/lib/one/.one":
        }->
            exec  {  "/usr/bin/podman  exec  ${lookup('opennebula')[services][0]
[hostname]} mv -f /tmp/one_auths/oneflow_auth /var/lib/one/.one":
        }->
            exec  {  "/usr/bin/podman  exec  ${lookup('opennebula')[services][0]
[hostname]} mv -f /tmp/one_auths/onegate_auth /var/lib/one/.one":
        }->
            exec  {  "/usr/bin/podman  exec  ${lookup('opennebula')[services][0]
[hostname]} mv -f /tmp/one_auths/sunstone_auth /var/lib/one/.one":
        }->
             virt::container_provision  {  "${lookup('opennebula')[services][0]
```

```
[hostname]}_fed_provision":
            require => Exec["slave-${lookup('id')}-db-restore"],
            container_name => lookup('opennebula')[services][0][hostname],
            plan => [
                {
                    "type" => "guest_file",
                    "actions" => [
                        {
                            'path' => '/etc/one/oned.conf',
                            'replace' => {
                                'src' => "ZONE_ID       = 0",
                                'dest' => "ZONE_ID       = 100"
                            }
                        },
                        {
                            'path' => '/etc/one/oned.conf',
                            'replace' => {
                                'src' => "MASTER_ONED   = \"\"",
                                        'dest' => "MASTER_ONED    = \"http://
${lookup('opennebula')[master_vip]}:2633/RPC2\""
                            }
                        },
                        {
                            'path' => '/etc/one/oned.conf',
                            'replace' => {
                                'src' => "\"STANDALONE\"",
                                'dest' => "\"SLAVE\""
                            }
                        },

                    ]
                },
                {
                    "type" => "guest",
                    "actions" => [
                        "chown -R oneadmin:oneadmin /var/lib/one/.one",
                        "su oneadmin - -c \"one start\""
                    ]
                },
                {
                    "type" => "container",
                    "actions" =>
                    ["%expect[0.0.0.0:2633]% podman exec ${lookup('opennebula')
[services][0][hostname]} ss -tulpn"]
                }
            ]
        }

    }

    return {
        master => $master_fed_result,
```

```
            slaves => $slave_fed_result,
        }

    }
```

Plan despliegue de zona esclava

Plan de despliegue que incorpora dos núcleos de *OpenNebula* a la zona esclava.

```
plan cloud_fed::slaves_ha(TargetSpec $targets) {
       $ha_deploy = apply($targets, _run_as => 'root') {
           #
           # Start HA cluster in slave zone
           #
           virt::services::opennebula::ha { "slave-zone-ha":
               oned => lookup('opennebula')[services][0],
               zone => lookup('opennebula')[zone_id],
               vip => lookup('opennebula')[vip]
           }

           #
           # Attach slave follower nodes to HA cluster
           #
           virt::services::opennebula { "slave-followers":
               require => Virt::Services::Opennebula::Ha["slave-zone-ha"],
               id => lookup('id'),
               mode => lookup('opennebula')[ha_mode][1],
               leader => lookup('opennebula')[services][0],
               zone_id => lookup('opennebula')[zone_id],
               master => lookup('opennebula')[master_id]
           }
       }

       if $ha_deploy.ok() {
           out::message("Slave HA zone deployment was successful")
       } else {
           fail_plan(
               'Slave HA zone deployment was unsuccessful',
               'slaves_ha error',
               {'result' => $ha_deploy.error()}
           )
       }
       return $ha_deploy
}
```

Plan despliegue de monitorización

Plan que despliega en paralelo el servicio de monitorización en ambas entidades.

```
plan cloud_fed::monitoring(TargetSpec $targets) {
       $monitoring = apply($targets, _run_as => 'root') {
           #
           # Start monitoring service
           #
           virt::services::opennebula::monitoring { "monitoring-$id":
               zone_id => lookup('id')
           }
       }

       if $monitoring.ok() {
           out::message("Monitoring deployment successfully")
       } else {
           fail_plan(
               'Monitoring deployment was unsuccessful',
               'monitoring error',
               {'result' => $monitoring.error()}
           )
       }
       return $monitoring
}
```

Anexo H

Aprovisionamiento del entorno

Para desplegar la red del entorno de despliegue, las máquinas virtuales y su aprovisionamiento se ha empleado la herramienta Vagrant. Ya que las máquinas virtuales son gestionadas por libvirt, se ha instalado el plugin de Vagrant que permite incorporar esta herramienta.

A continuación se presentan los scripts de aprovisionamiento y el fichero *Vagrantfile* definidos que automatizan el despliegue del entorno de virtualización partiendo de un sistema limpio. También se presenta la configuración del servicio HAProxy que ejecutan el *gateway*, para balancear la carga de las zonas Ceph, y los núcleos de *OpenNebula*, para proveer de terminación cifrada a los endpoint RPC.

Script de inicialización

Script que instala las dependencias básicas para poder desplegar las máquinas virtuales y la red e instala el maestro Puppet.

```
  #!/usr/bin/env bash
PUPPET_MODULEPATH="$HOME/cloud-fed/deploy:$HOME/cloud-fed/modules"
PUPPET_SETUP_PATH="$HOME/cloud-fed/init/setup.pp"
PUPPET_SSL_PATH="/etc/puppetlabs/puppet/ssl"

BLUE='\033[0;34m'
YELLOW='\033[1;33m'
GREEN='\033[0;32m'

# Prerequisitos

echo "--------------------------"
echo "Instalando dependencias..."
echo "--------------------------"
sudo apt -qq update
sudo apt -qq install -y curl rsync wget gnupg fuse3 fuse-overlayfs \
    nfs-kernel-server rpcbind jq yq

#
# Instalar dependencias del despliegue
# kvm, libvirtd, puppet, vagrant y podman
#

echo "--------------------"
echo "Instalando Puppet..."
echo "--------------------"
# Añadir repositorios de puppet
if [ !$(dpkg -l puppet8-release) ]; then
    echo "Puppet Forge api key: "
    read api_key
```

```
    echo "Installing puppet repositories"

    wget -q --content-disposition \
        "https://apt-puppetcore.puppet.com/public/puppet8-release-$(lsb_release
-cs).deb"
    sudo dpkg -i "puppet8-release-$(lsb_release -cs).deb"

    sudo bash -c "cat << EOT > /etc/apt/auth.conf.d/apt-puppetcore-puppet.conf
machine apt-puppetcore.puppet.com
login forge-key
password $api_key
EOT"
else
    echo "${YELLOW}---------------------------------"
    echo "${YELLOW}WARNING: Puppet installation found"
    echo "${YELLOW}---------------------------------"
fi

# Añadir repositorios de vagrant
if [ !$(dpkg -l vagrant) ]; then
    echo "Installing vagrant repositories"

    wget -q -O - https://apt.releases.hashicorp.com/gpg \
        | gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

    sudo bash -c "cat << EOT > /etc/apt/sources.list.d/hashicorp.list
deb [arch=$(dpkg --print-architecture) \
signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main
EOT"
else
    echo "${YELLOW}---------------------------------"
    echo "${YELLOW}WARNING: Vagrant installation found"
    echo "${YELLOW}---------------------------------"
fi

# Añadir repositorios alvistack para podman >=v4.4
alvistack_repokey="/etc/apt/trusted.gpg.d/alvistack.gpg"
if [ ! -f $alvistack_repokey ]; then
       wget  -q  -O  -  http://downloadcontent.opensuse.org/repositories/home:/
alvistack/Debian_12/Release.key \
        | sudo gpg --dearmor -o $alvistack_repokey

    sudo bash -c "cat << EOT > /etc/apt/sources.list.d/alvistack.list
deb http://downloadcontent.opensuse.org/repositories/home:\
/alvistack/Debian_12/ /
EOT"
else
    echo "${YELLOW}-----------------------------------------------------------"
    echo "${YELLOW}WARNING: Possible podman alvistack installation detected"
    echo "${YELLOW}-----------------------------------------------------------"
fi
```

```
echo "Instalando dependencias"
echo "qemu-system libvirt-daemon-system vagrant openjdk-17-jdk openjdk-17-jre \
    puppetserver puppet-agent podman"
sudo apt -qq update
sudo apt -qq install -y \
    qemu-system libvirt-daemon-system \
    vagrant \
    openjdk-17-jdk openjdk-17-jre puppetserver puppet-agent hiera puppet-bolt\
    python3 python3-pip \
    podman podman-netavark crun &>/dev/null

sudo   /opt/puppetlabs/bin/puppet   resource   service   puppet   ensure=running
enable=true
sudo systemctl start puppetserver
sudo systemctl enable puppetserver
sudo /opt/puppetlabs/bin/puppet \
    config set server cephgateway03.intra.unizar.es \
    --section main

# Instalación específica de ruby
gpg --keyserver keyserver.ubuntu.com \
    --recv-keys \
    409B6B1796C275462A1703113804BB82D39DC0E3 \
    7D2BAF1CF37B13E2069D6956105BD0E739499BDB

curl -sSL https://get.rvm.io | bash -s stable
source /home/cephadm/.rvm/scripts/rvm
rvm install ruby-3.1.7

gem install hiera fog-libvirt
echo "export PATH=$HOME/.rvm/gems/ruby-3.1.7/bin:$PATH" >> ~/.bashrc


#
# Despliegue de los bridge para las vm y gateway
#
sudo /opt/puppetlabs/bin/puppet config set codedir \
    $HOME/cloud-fed/deploy/puppet \
    --section server

sudo /opt/puppetlabs/bin/puppet config set environmentpath \
    $HOME/cloud-fed/deploy/puppet/environments/production \
    --section server

sudo /opt/puppetlabs/bin/puppet config set basemodulepath \
    $HOME/cloud-fed/deploy/puppet/environments/production/modules \
    --section server

sudo /opt/puppetlabs/bin/puppet config set hiera_config \
    $HOME/cloud-fed/deploy/puppet/environments/production/hiera.yaml \
    --section server
```

```
sudo sed -i \
    "s/server-code-dir/\/.*$/\/home\/cephadm\/cloud-fed\/deploy\/puppet/" \
    /etc/puppetlabs/puppetserver/conf.d/puppetserver.conf
sudo groupadd deploy
sudo usermod -aG deploy cephadm
sudo usermod -aG deploy puppet
sudo chown -R cephadm:deploy $HOME/cloud-fed
sudo chmod -R 775 $HOME/cloud-fed
sudo chmod -R 755 $HOME

sudo /opt/puppetlabs/bin/puppet apply \
    $PUPPET_SETUP_PATH \
    --modulepath=$PUPPET_MODULEPATH

sudo /opt/puppetlabs/bin/puppetserver ca generate \
    --certname vm0-cert,vm1-cert

sudo cp $PUPPET_SSL_PATH/certs/ca.pem /tmp
sudo cp $PUPPET_SSL_PATH/certs/vm0-cert.pem /tmp
sudo cp $PUPPET_SSL_PATH/certs/vm1-cert.pem /tmp
sudo cp $PUPPET_SSL_PATH/private_keys/vm0-cert.pem /tmp/pk-vm0-cert.pem
sudo cp $PUPPET_SSL_PATH/private_keys/vm1-cert.pem /tmp/pk-vm1-cert.pem

echo "${GREEN}-------------------------------------------------"
echo "${GREEN}Certificados de agentes Puppet firmados y en sitio"
echo "${GREEN}-------------------------------------------------"

sudo chmod o+r /tmp/*.pem

#
# Despliegue de las VM con vagrant
#

# Definición de pool de almacenamiento para VM
echo "${BLUE}-----------------------------------------"
echo "${BLUE}Creación de pooles de almacenamiento para VM"
echo "${BLUE}-----------------------------------------"

sudo mkdir -p /var/lib/virt/images
sudo mkfs.ext4 /dev/sda
sudo mount /dev/sda /var/lib/virt/images
virsh -c qemu:///system \
    pool-define-as \
    --name virtimages \
    --type dir \
    --target /var/lib/virt/images

virsh -c qemu:///system \
    pool-start virtimages
```

```
virsh -c qemu:///system \
    pool-autostart virtimages

# Instalación del plugin vagrant-libvirt
echo "${BLUE}---------------------------------"
echo "${BLUE}Instalando plugin vagrant-libvirt"
echo "${BLUE}---------------------------------"

vagrant plugin install vagrant-libvirt

ssh-keygen -q \
    -t ecdsa \
    -f /tmp/id_ecdsa.k1 \
    -N ""

ssh-keygen -q \
    -t ecdsa \
    -f /tmp/id_ecdsa.k2 \
    -N ""

# Rootless virsh
sudo usermod -aG libvirt $USER

echo "${GREEN}----------------"
echo "${GREEN}Plugin instalado"
echo "${GREEN}----------------"

echo "${BLUE}----------------------------------------"
echo "${BLUE}Desplegando máquinas para infraestructura"
echo "${BLUE}----------------------------------------"

vagrant up

echo "${GREEN}--------------------"
echo "${GREEN}Máquinas desplegadas"
echo "${GREEN}--------------------"


sudo ip r add 192.168.0.0/19 via 10.88.0.144

# Modificar ips de administración en el inventorio

vm0_ip=$(vagrant ssh vm0 -- ip -j a show eth0 \
    | jq -r '.[] | .addr_info[0].local')
vm1_ip=$(vagrant ssh vm1 -- ip -j a show eth0 \
    | jq -r '.[] | .addr_info[0].local')

yq -y \
    -i "(.targets[] | select(.name == 'vm0-cert')).uri = '$vm0_ip'" \
    inventory.yaml

yq -y \
```

```
    -i "(.targets[] | select(.name == 'vm1-cert')).uri = '$vm1_ip'" \
    inventory.yaml

#
# Ofrecer comando para desplegar la infraestructura
#
echo "${BLUE}--------------------------"
echo "${BLUE}Desplegando infraestructura"
echo "${BLUE}--------------------------"
bolt plan run \
    --project $HOME/cloud_fed/deploy/puppet/environments/production \
    cloud_fed::deploy_federation

echo "${GREEN}--------------------------"
echo "${GREEN}Infraestructura desplegada"
echo "${GREEN}--------------------------"
```

Vagrantfile

Manifiesto que define dos máquinas virtuales, aprovisionadas por el mismo script Bash y pasando los certificados para los agentes Puppet.

```
  require "yaml"
require "base64"

VM_COUNT = 2
DISK_SIZE = "20G"
DISK_PER_VM = 5

# Cargar configuración de hiera
config_file = File.join(File.dirname(__FILE__), "hiera.yaml")
if File.exist?(config_file)
  settings = YAML.load_file(config_file)
else
  raise "Configuration file not found: #{config_file}"
end

BRIDGES = settings["net::bridges"]
LVMS = settings["storage::lvms"]
VG = settings["storage::vgs"]

key1_pub = File.read("/tmp/id_ecdsa.k1.pub")
key1_priv = File.read("/tmp/id_ecdsa.k1")
key2_pub = File.read("/tmp/id_ecdsa.k2.pub")
key2_priv = File.read("/tmp/id_ecdsa.k2")

master_vagrant_keys = [
  Base64.strict_encode64(key1_priv),
  Base64.strict_encode64(key2_pub),
]
slave_vagrant_keys = [
  Base64.strict_encode64(key2_priv),
  Base64.strict_encode64(key1_pub),
]

vagrant_keys = [
  master_vagrant_keys,
  slave_vagrant_keys,
]

certs_script = <<-SCRIPT
echo "Moving certs..."
sudo mkdir -p /etc/puppetlabs/puppet/ssl/{certs,private_keys}
sudo mv /tmp/ca.pem /etc/puppetlabs/puppet/ssl/certs
sudo mv /tmp/$1.pem /etc/puppetlabs/puppet/ssl/certs
sudo mv /tmp/pk-$1.pem /etc/puppetlabs/puppet/ssl/private_keys/$1.pem
SCRIPT

Vagrant.configure("2") do |config|
```

```
# Use Debian 12 box for libvirt
config.vm.box = "debian/bookworm64"
config.vm.synced_folder ".", "/vagrant", disbled: true

# Libvirt provider settings
config.vm.provider :libvirt do |libvirt|
  libvirt.driver = "kvm"
  libvirt.storage_pool_name = "virtimages"
  libvirt.memory = 131072
  libvirt.cpus = 2
end
# Create multiple VMs
(0..VM_COUNT - 1).each do |i|
  config.vm.define "vm#{i}" do |node|
    node.vm.hostname = "vm#{i}"
    cert_name = "vm#{i}-cert"

    # Configure network - bridge to your VLAN bridge
    node.vm.network :public_network,
      :dev => BRIDGES[i],
      :mode => "bridge",
      :type => "bridge",
      :mac => "52:54:00:12:34:#{sprintf("%02x", i + 10)}",
      :ip => "192.168.#{i + 1}0.250"

    node.vm.provider :libvirt do |libvirt|
      (0..DISK_PER_VM - 1).each do |idx|
        libvirt.storage :file,
          :size => DISK_SIZE,
          :device => "vd#{("b".ord + idx).chr}",
          :cache => "none"
      end
    end
    # Provision with shell script to install Puppet agent
    node.vm.provision "shell",
      path: "vagrant_provision.sh",
      args: "#{i} #{vagrant_keys[i].join(" ")}"

    # Provision certificate files
    node.vm.provision "file",
                      source: "/tmp/ca.pem",
                      destination: "/tmp/ca.pem"
    node.vm.provision "file",
                      source: "/tmp/#{cert_name}.pem",
                      destination: "/tmp/#{cert_name}.pem"
    node.vm.provision "file",
                      source: "/tmp/pk-#{cert_name}.pem",
                      destination: "/tmp/pk-#{cert_name}.pem"
    node.vm.provision "shell",
      inline: certs_script,
      args: "#{cert_name}"
  end
```

```
  end
end
```

Aprovisionamiento de máquinas virtuales

Script Bash que instala Podman, y otras dependencias, y configura la red de las máquinas virtuales.

```
 #!/usr/bin/env bash
PUPPETSERVER="cephgateway03.intra.unizar.es"
VMID=$1
PRIV_KEY=$2
PUB_KEY=$3

export DEBIAN_FRONTEND=noninteractive

sudo apt -qq update
sudo apt -qq install -y curl rsync wget gnupg \
    fuse3 fuse-overlayfs python3 lvm2

# Install Puppet agent
wget https://apt.puppet.com/puppet8-release-bookworm.deb
dpkg -i puppet8-release-bookworm.deb
apt-get -qq update
apt-get -qq install -y puppet-agent

echo "10.0.13.71     $PUPPETSERVER" >> /etc/hosts

# Configure Puppet
cat > /etc/puppetlabs/puppet/puppet.conf <<EOF
[main]
certname = vm${VMID}-cert
server = $PUPPETSERVER
EOF

# Enable and start Puppet service
systemctl enable puppet
systemctl start puppet

sudo ln -s /opt/puppetlabs/puppet/bin/ruby /usr/bin/ruby

# Añadir repositorios alvistack para podman >=v4.4
alvistack_repokey="/etc/apt/trusted.gpg.d/alvistack.gpg"
if [ ! -f $alvistack_repokey ]; then
       wget  -q  -O  -  http://downloadcontent.opensuse.org/repositories/home:/
alvistack/Debian_12/Release.key \
        | sudo gpg --dearmor -o $alvistack_repokey

    sudo bash -c "cat << EOT > /etc/apt/sources.list.d/alvistack.list
deb http://downloadcontent.opensuse.org/repositories/home:\
/alvistack/Debian_12/ /
EOT"
else
    echo "-------------------------------------------------------"
    echo "WARNING: Possible podman alvistack installation detected"
```

```
    echo "------------------------------------------------------"
fi

sudo apt -qq update
sudo apt -qq install -y \
    podman podman-netavark crun \
    ceph-common

echo $PRIV_KEY | base64 --decode > /home/vagrant/.ssh/id_ecdsa
echo $PUB_KEY | base64 --decode >> /home/vagrant/.ssh/authorized_keys

sudo chmod 0600 /home/vagrant/.ssh/id_ecdsa

# Directorios para ficheros temporales
mkdir /tmp/templates
mkdir /tmp/scripts

# para conseguir libruby de los repos oficiales
sudo rm /etc/apt/sources.list.d/alvistack.list

# configuracion de red
NETID="$((VMID + 1))0"
echo "Setting up network..."

sudo ip link add name frontend type bridge
sudo ip link add name br_stor type bridge
sudo ip link set eth1 master frontend
sudo ip a flush dev eth1
sudo ip link set eth1 up
sudo ip link set frontend up

echo "Frontend bridge configured"

# conexión macvlan

echo "Setting up macvlan connection"
sudo ip link add link frontend name host-bridge type macvlan mode bridge
sudo ip a add 192.168.$NETID.250/24 dev host-bridge
sudo ip link set host-bridge up
echo "macvlan has ip address"
sudo ip r add 192.168.$NETID.0/24 dev host-bridge
echo "Macvlan connection configured"
```

Configuración HAProxy del *gateway*

Se define un *frontend* que acepta las peticiones de acceso a recursos del *Marketplace* y las redirige al *backend s3-balancer*. El *backend* aplica una política de *roundrobin* sobre los RGW.

```
  frontend http-in
        bind 192.168.10.254:80
        mode http
       option forwardfor      # except 127.0.0.0/8 quitada para SSL termination
        reqadd X-Forwarded-Proto:\ http # puesta para SSL termination

        default_backend s3-balancer


backend s3-balancer
        balance roundrobin
        mode http
        server rgw-10 192.168.10.95:7480
        server rgw-20 192.168.20.95:7480
```

Configuración HAProxy de los núcleos

Terminación TLS para el endpoint RPC de los endpoint de *OpenNebula*. Se aplica a comunicaciones entre zonas distintas.

```
  frontend http-in
        bind 192.168.10.10:80
        mode http
        option http-server-close
       option forwardfor      # except 127.0.0.0/8 quitada para SSL termination
        reqadd X-Forwarded-Proto:\ http # puesta para SSL termination
        redirect scheme https

frontend https-in
        bind 192.168.10.10:443 ssl no-sslv3 no-tls-tickets crt /etc/cert/
        reqadd X-Forwarded-Proto:\ https
       option forwardfor      # except 127.0.0.0/8 quitada para SSL termination
        #option httplog

        default_backend secure-rpc2


backend secure-rpc2
        mode http
        server nebulafrontal 127.0.0.1:2633
```

Anexo I

Manifiestos OpenTofu

En este anexo se presentan los manifiestos de *OpenTofu* empleados para describir la máquina virtual del validador de políticas, la del *backup*, junto a su datastore específico, la imagen de disco que emplean ambas máquinas y el *Marketplace*. Primero se ha de desplegar el manifiesto *00_providers*, especificando que solo se van a desplegar los usuarios y grupos con los que se deben desplegar el resto de recursos. Posteriormente, se pueden invocar el resto de manifiestos en un único plan de despliegue, que *OpenTofu* se encarga de establecer las relaciones de dependencias necesarias.

Definición del Driver de *OpenNebula*

```
terraform {
  required_providers {
    opennebula = {
      source = "opentofu/opennebula"
      version = "1.4.1"
    }
  }
}
```

Proveedores

Manifiesto que contacta con *OpenNebula* usando el usuario administrador *oneadmin* y crea los usuarios, ACL y grupos necesarios para las siguientes fases del despliegue. Los proveedores con usuario recién creados se ejecutan en un plan de despliegue distinto, pero esto permite agrupar todos los proveedores en el mismo fichero.

```
  #
# Oneadmin providers
#
provider "opennebula" {
    endpoint = "http://192.168.10.1:2633/RPC2"
    alias = "one-10"
    username = "oneadmin"
    password = "oneadmin"
}

provider "opennebula" {
    endpoint = "http://192.168.20.1:2633/RPC2"
    alias = "one-20"
    username = "oneadmin"
    password = "oneadmin"
}


#
# Define local variables
#
locals {
   services = ["opa", "backup"]
   zone1 = {
     policies_password = "policies"
     nebula = {
        federation_index = 10
        ipaddress = "192.168.10.55"
        password = "oneadmin"
     ssh_keys = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCDJtU18jEl0tQcqZNKmAt54GiGmb2qUK5cb1C
Z1rPJZy7CpULW+Zi1Ctg/RKKsZYpn/drVPuwj3R3JbuQBh24zPG78jeWZc7PuqDuRTMkSQlxuSeBJqZeisiI2BJv2SY
OtQ8YYgqay58f4IxD86DHTjiXTALvTWBtaV3nxXT3XoOpCwgNxecUNdee5RdrX1ncH5LOsq3YNPtKyN92FkXBPrmI2p
cYcM8SObM8lnmsqDvU0G7M7jJROoeJvuBwCt9MR5BIM+cnAFmrOu4sF8YS/
XS80A53RMZtl9cHipclCkLKQQxnI/ieGjeUeN6xRaJUE2wRdkW0sgicMRPuC4MVNM="
     }
     ceph = {
        fsid = "4d006b2c-a99d-4410-9b01-152944055b7b"
        cluster_name = "ceph-10"
        monitor = "192.168.10.90"
        users = {
            backup = {
                fs_name = "backup-10"
                key = "AQDXJPpoOhWpJhAAuFUHLyKQuf6UTXSbHd1Tgw=="
            }
            policies = {
```

```
                fs_name = "policies-10"
                key = "AQD3JPpoXpSNJRAA/r32NxgIFBjlfSrmWdltIw=="
            }
        }
    }
  }
  zone2 = {
    policies_password = "policies"
    nebula = {
       federation_index = 20
       ipaddress = "192.168.20.55"
       password = "oneadmin"
     ssh_keys = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCDJtU18jEl0tQcqZNKmAt54GiGmb2qUK5cb1C
Z1rPJZy7CpULW+Zi1Ctg/RKKsZYpn/drVPuwj3R3JbuQBh24zPG78jeWZc7PuqDuRTMkSQlxuSeBJqZeisiI2BJv2SY
OtQ8YYgqay58f4IxD86DHTjiXTALvTWBtaV3nxXT3XoOpCwgNxecUNdee5RdrX1ncH5LOsq3YNPtKyN92FkXBPrmI2p
cYcM8SObM8lnmsqDvU0G7M7jJROoeJvuBwCt9MR5BIM+cnAFmrOu4sF8YS/
XS80A53RMZtl9cHipclCkLKQQxnI/ieGjeUeN6xRaJUE2wRdkW0sgicMRPuC4MVNM="
    }
    ceph = {
       fsid = "420e41dd-a068-43fe-9cbc-13b0e9c66bf0"
       cluster_name = "ceph-20"
       monitor = "192.168.20.90"
       users = {
           backup = {
               fs_name = "backup-20"
               key = "AQAYJfpoDjTFIxAAFoRCQAjSt0IdwTNSn1Q5Lg=="
           }
           policies = {
               fs_name = "policies-20"
               key = "AQA5Jfpodod1GBAADh2L9bjpXR4dHZ3m+uQQIg=="
           }
       }
    }
  }
}

#
# Services federation group
#
resource "opennebula_group" "services" {
    provider = opennebula.one-10
    name = "services"
}

resource "opennebula_acl" "services_acl" {
    provider = opennebula.one-10
    user = "@${opennebula_group.services.id}"
    resource = "VM+NET+IMAGE+TEMPLATE/*"
    rights = "CREATE+USE+MANAGE+ADMIN"
}

#
```

```
# Policies federation group
#
resource "opennebula_group" "policies" {
    provider = opennebula.one-10
    name = "policies"
}

resource "opennebula_acl" "policies_acl" {
    provider = opennebula.one-10
    user = "@${opennebula_group.policies.id}"
    resource = "VM+NET+IMAGE+TEMPLATE/*"
    rights = "CREATE+USE+MANAGE+ADMIN"
}
#
# Policies zone user
#
resource "opennebula_user" "policies_10" {
    provider = opennebula.one-10
    name = "policies_10"
    password = local.zone1.policies_password
    auth_driver = "core"
    primary_group = opennebula_group.policies.id
    groups = [opennebula_group.services.id]
}

resource "opennebula_user" "policies_20" {
    provider = opennebula.one-20
    name = "policies_20"
    password = local.zone2.policies_password
    auth_driver = "core"
    primary_group = opennebula_group.policies.id
    groups = [opennebula_group.services.id]
}


#
# Policies user provider
#
provider "opennebula" {
    endpoint = "http://192.168.10.1:2633/RPC2"
    alias = "policies-10"
    username = "policies_10"
    password = local.zone1.policies_password
}

provider "opennebula" {
    endpoint = "http://192.168.20.1:2633/RPC2"
    alias = "policies-20"
    username = "policies_20"
    password = local.zone2.policies_password
}
```

Red

Manifiesto que crea las redes virtuales de *OpenNebula* a las que se conectarán las máquinas virtuales de las siguientes fases del despliegue. Aunque se desplieguen todos los recursos en el mismo plan, *OpenTofu* entiende las dependencias entre recursos y desplegará las máquinas virtuales una vez que las redes virtuales estén definidas.

```
  #
# Define vnet
#

# Red VLAN de servicio en la entidad 10
resource "opennebula_virtual_network" "services-10" {
  provider = opennebula.one-10
  name            = "services-10"
  type            = "802.1Q"
  physical_device = "br_kvm"
  vlan_id         = "10"
  gateway         = "192.168.10.254"
  dns             = "155.210.3.12 155.210.12.9"
  mtu             = 1500
}

# Red VLAN de servicio en la entidad 20
resource "opennebula_virtual_network" "services-20" {
  provider = opennebula.one-20
  name            = "services-20"
  type            = "802.1Q"
  physical_device = "br_kvm"
  vlan_id         = "20"
  gateway         = "192.168.10.254"
  dns             = "155.210.3.12 155.210.12.9"
  mtu             = 1500
}

# Rango de direcciones para la VLAN de servicios en la entidad 10
resource "opennebula_virtual_network_address_range" "services_ar-10" {
  provider = opennebula.one-10
  virtual_network_id = opennebula_virtual_network.services-10.id
  ar_type            = "IP4"
  size               = length(local.services)
  ip4                = local.zone1.nebula.ipaddress
}

# Rango de direcciones para la VLAN de servicios en la entidad 20
resource "opennebula_virtual_network_address_range" "services_ar-20" {
  provider = opennebula.one-20
  virtual_network_id = opennebula_virtual_network.services-20.id
  ar_type            = "IP4"
  size               = length(local.services)
  ip4                = local.zone2.nebula.ipaddress
}
```

Imágenes

Manifiesto que crea las imágenes locales a cada zona, no persistentes, sobre las que desplegar *OPA* y la máquina virtual del *backup*.

```
  #
# Define image
#

#
# Imagen no persistente con software mínimo preinstalado, cloud init,
# permisos de explotación y gestión para el propietario y usada como base para
# los servicios mínimos de la federación
#
resource "opennebula_image" "services-10" {
  provider = opennebula.one-10
  name         = "services"
  description  = "Debian limpio"
  datastore_id = 101
  persistent   = false
  lock         = "UNLOCK"
   path          = "https://marketplace.opennebula.systems/appliance/f4cc1890-f
430-013c-b669-7875a4a4f528/download"
  dev_prefix   = "vd"
  driver       = "qcow2"
  type         = "OS"
  group        = "services"
  size         = "8196"
  permissions  = "660"
}

# Homóloga de la entidad 10
resource "opennebula_image" "services-20" {
  provider = opennebula.one-20
  name         = "services"
  description  = "Debian limpio"
  datastore_id = 101
  persistent   = false
  lock         = "UNLOCK"
   path          = "https://marketplace.opennebula.systems/appliance/f4cc1890-f
430-013c-b669-7875a4a4f528/download"
  dev_prefix   = "vd"
  driver       = "qcow2"
  type         = "OS"
  group        = "services"
  size         = "8196"
  permissions  = "660"
}
```

Backup

Manifiesto que despliega la máquina virtual para el *backup*. En el contexto se especifican el usuario, su contraseña y las claves de acceso *SSH*. También se concreta en un *script* de inicio los paquetes a instalar.

```
  #
# Define backup VM
#

# Máquina virtual para el backup
resource "opennebula_virtual_machine" "backup-10" {
  provider = opennebula.one-10

  name        = "one-10-backup-backup"
  description = "backup vm"
  cpu         = 1
  vcpu        = 1
  memory      = 4096

  context = {
    NETWORK      = "YES"
    HOSTNAME     = "$NAME"
    USERNAME     = "oneadmin"
    PASSWORD     = local.zone1.nebula.password
    SSH_PUBLIC_KEY = local.zone1.nebula.ssh_keys

    # Instalación de paquetes y configuración de CephFS
    START_SCRIPT = <<-EOT
    #!/bin/bash
    apt update && apt install -y rsync qemu-utils
    [ -d /var/lib/one/datastores ] || mkdir -p /var/lib/one/datastores
    chown -R oneadmin:oneadmin /var/lib/one/datastores
    if [ ! -d /etc/ceph ]; then
        mkdir -p -m 755 /etc/ceph
                            mount    -t    ceph    backup@${local.zone1.ceph.fsid}.
${local.zone1.ceph.users.backup.fs_name}=/ /var/lib/one/datastores -o mon_addr=${local.zone1
    fi
    EOT
  }

  graphics {
    type   = "VNC"
    listen = "0.0.0.0"
    keymap = "es"
  }

  os {
    arch = "x86_64"
    boot = "disk0"
  }
```

```
    # Imagen de servicios no persistente
  disk {
    image_id = opennebula_image.services-10.id
    target   = "hda"
    driver   = "qcow2"
  }

  nic {
    model           = "virtio"
    network_id      = opennebula_virtual_network.services-10.id
  }

}

# Homóloga de la entidad 10
resource "opennebula_virtual_machine" "backup-20" {
  provider = opennebula.one-20

  name        = "one-20-backup-backup"
  description = "backup vm"
  cpu         = 1
  vcpu        = 1
  memory      = 4096

  context = {
    NETWORK      = "YES"
    HOSTNAME     = "$NAME"
    USERNAME     = "oneadmin"
    PASSWORD     = local.zone2.nebula.password
    SSH_PUBLIC_KEY = local.zone2.nebula.ssh_keys
    START_SCRIPT = <<-EOT
    #!/bin/bash
    apt update && apt install -y rsync qemu-utils
    [ -d /var/lib/one/datastores ] || mkdir -p /var/lib/one/datastores
    if [ ! -d /etc/ceph ]; then
        mkdir -p -m 755 /etc/ceph
                            mount   -t   ceph   backup@${local.zone2.ceph.fsid}.
${local.zone2.ceph.users.backup.fs_name}=/ /var/lib/one/datastores -o mon_addr=${local.zone2
    fi
    chown -R oneadmin:oneadmin /var/lib/one/datastores
    EOT
  }

  graphics {
    type   = "VNC"
    listen = "0.0.0.0"
    keymap = "es"
  }

  os {
    arch = "x86_64"
    boot = "disk0"
```

```
  }

  disk {
    image_id = opennebula_image.services-20.id
    target   = "hda"
    driver   = "qcow2"
  }

  nic {
    model           = "virtio"
    network_id      = opennebula_virtual_network.services-20.id
  }

}
```

Datastore de backup

Manifiesto que crea el datastore de tipo *BACKUP_DS* y conecta con la máquina de *backup*.

```
  #
# Datastore definition
#

# Datastore backup con driver Restic entidad 10
resource "opennebula_datastore" "backup-10" {
 provider = opennebula.one-10
 name = "backup"

 custom {
  datastore = "restic"
  transfer = "shared"
 }

 tags = {
  restic_password = "opennebula"
  restic_sftp_server = "${local.zone1.nebula.ipaddress}"
  type = "BACKUP_DS"
 }
 type = "IMAGE"
}

# Datastore backup con driver Restic entidad 20
resource "opennebula_datastore" "backup-20" {
 provider = opennebula.one-20
 name = "backup"

 custom {
  datastore = "restic"
  transfer = "shared"
 }

 tags = {
  restic_password = "opennebula"
  restic_sftp_server = local.zone2.nebula.ipaddress
  type = "BACKUP_DS"
  tm_mad = "-"
 }
 type = "IMAGE"
}
```

OPA

Manifiesto que despliega la máquian virtual sobre la que corre el contenedor *Docker* de *OPA*.

```
  #
# OPA engine VM
#
resource "opennebula_virtual_machine" "opa-10" {
  provider = opennebula.policies-10

  name        = "one-10-policies_10-opa"
  description = "opa engine vm"
  cpu         = 1
  vcpu        = 1
  memory      = 4096
  group       = "policies"
  permissions = "740" # use and manage by zone user and only use by policies group

  context = {
    NETWORK      = "YES"
    HOSTNAME     = "$NAME"
    USERNAME     = "oneadmin"
    PASSWORD     = local.zone1.nebula.password
    SSH_PUBLIC_KEY = local.zone1.nebula.ssh_keys
    START_SCRIPT = <<-EOT
    #!/bin/bash
    apt update && apt install -y podman
    [ -d /var/lib/policies ] || mkdir -p /var/lib/policies
    chown -R oneadmin:oneadmin /var/lib/policies
    if [ ! -d /etc/ceph ]; then
        mkdir -p -m 755 /etc/ceph
                        mount   -t   ceph   policies@${local.zone1.ceph.fsid}.
${local.zone1.ceph.users.policies.fs_name}=/ /var/lib/policies -o mon_addr=${local.zone1.cep
    fi
      sudo podman run -v /var/lib/policies:/policies -p 2345:2345 docker.io/
openpolicyagent/opa      run --server --log-level debug --addr=0.0.0.0:2345 /
policies
    EOT
  }

  # requerido para ejecutar contenedores
  raw {
    type = "kvm"
    data = "<cpu mode='host-passthrough'/>"
  }

  graphics {
    type   = "VNC"
    listen = "0.0.0.0"
    keymap = "es"
  }
```

```
  os {
    arch = "x86_64"
    boot = "disk0"
  }

  disk {
    image_id = opennebula_image.services-10.id
    target   = "hda"
    driver   = "qcow2"
  }

  nic {
    model           = "virtio"
    network_id      = opennebula_virtual_network.services-10.id
  }

}

resource "opennebula_virtual_machine" "opa-20" {
  provider = opennebula.policies-20

  name        = "one-20-policies_20-opa"
  description = "opa engine vm"
  cpu         = 1
  vcpu        = 1
  memory      = 4096
  group       = "policies"
  permissions = "740" # use and manage by zone user and only use by policies group

  context = {
    NETWORK      = "YES"
    HOSTNAME     = "$NAME"
    USERNAME     = "oneadmin"
    PASSWORD     = local.zone1.nebula.password
    SSH_PUBLIC_KEY = local.zone1.nebula.ssh_keys
    START_SCRIPT = <<-EOT
    #!/bin/bash
    apt update && apt install -y podman
    [ -d /var/lib/policies ] || mkdir -p /var/lib/policies
    chown -R oneadmin:oneadmin /var/lib/policies
    if [ ! -d /etc/ceph ]; then
        mkdir -p -m 755 /etc/ceph
                        mount   -t   ceph   policies@${local.zone2.ceph.fsid}.
${local.zone2.ceph.users.policies.fs_name}=/ /var/lib/policies -o mon_addr=${local.zone2.cep
    fi
      sudo  podman  run  -v  /var/lib/policies:/policies  -p  2345:2345  docker.io/
openpolicyagent/opa      run  --server  --log-level  debug  --addr=0.0.0.0:2345  /
policies
    EOT
  }

  # requerido para ejecutar contenedores
```

```
  raw {
    type = "kvm"
    data = "<cpu mode='host-passthrough'/>"
  }

  graphics {
    type   = "VNC"
    listen = "0.0.0.0"
    keymap = "es"
  }

  os {
    arch = "x86_64"
    boot = "disk0"
  }

  disk {
    image_id = opennebula_image.services-20.id
    target   = "hda"
    driver   = "qcow2"
  }

  nic {
    model           = "virtio"
    network_id      = opennebula_virtual_network.services-20.id
  }

}
```

Marketplace

Manifiesto que despliega el *Marketplace* privado de la federación. Se especifica el bucket, usuario y clave de acceso para conectar en el endpoint del balanceador.

```
  resource "opennebula_marketplace" "federation-marketplace" {
  name = "federation-marketplace"

  s3 {
    type = "ceph"
    access_key_id = "xxxxxxxxxxxxx"
    secret_access_key = "xxxxxxxxxxxxx"
    region = "marketplace"
    endpoint_url = "http://192.168.10.254"
    bucket_id= "xxxxxxxxxxxxx"
  }
}
```

Anexo J

Infraestructura de red

La asignación de direcciones a cada uno de los servicios y componentes del sistema se expresa en la Tabla 1. La cadena *vid* representa el número de VLAN utilizado en cada caso, pudiendo valer 10 o 20.

Nombre	IP Servicio	Puerto Servicio	IP Pública	IP Interna
Red cluster ceph	192.168.<vid>.0/24	-	-	-
Red replicación ceph	-	-	-	192.168.30.0/24
Gateway	192.168.<vid>.254	-	10.88.0.144	-
Ceph monitor	192.168.<vid>.90-2	3300, 6789	-	-
Ceph manager	192.168.<vid>.93-4	8443, 8080	-	-
Ceph OSD	192.168.<vid>.80-2	6800-7300	-	192.168.30.80-2
Ceph MDS	192.168.<vid>.100-3	6800-7300	-	-
Ceph RGW	192.168.<vid>.95	7480	-	-
Frontal *OpenNebula*	192.168.<vid>.0-2	2633, 9090	-	-
IP Virtual *OpenNebula*	192.168.<vid>.10	2633, 9090	-	-
MariaDB *OpenNebula*	192.168.<vid>.20-2	3306	-	-
MV Backup	192.168.<vid>.55	2431 (SFTP)	-	-
MV OPA	192.168.<vid>.56	2345	-	-
Prometheus	192.168.<vid>.30	9090	-	-

Tabla 1: Asignación de direcciones IP.