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

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 diseñado está dividida en 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 están virtualizados 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 se puedan compartir usuarios y recursos de virtualización, como imágenes de disco para máquinas virtuales. Este modelo permite a cada entidad mantener una gestión local de la 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* cuyos servicios están virtualizados en *OpenNebula* y almacena las copias en *Ceph*. Se ha valorado positivamente que la herramienta de *backup* cuente con integración nativa con *OpenNebula*, así la implementación de las estrategias y trabajos de *backup* diseñados ha sido lo más directa y efectiva posible.

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.

Glosario

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…).

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

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.

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.

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

Trunk En el contexto de interfaces de red que dejan pasar tráfico de VLAN, capacidad de la interfaz para transmitir más de una VLAN.

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*.

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

Device class Etiqueta asignada a un OSD.

Regla CRUSH Política que define cómo se colocan y replican los objetos en el clúster Ceph. Rige el dominio de fallo y jerarquía en el clúster. Un ejemplo básico es la asignación de un *device class* a un pool.

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

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*.

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 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 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 se destruye.

Índice

Capítulo 1.

Introducción

1.1. Contexto

El SICUZ es el Servicio de Informática y Comunicaciones de la Universidad de Zaragoza; encargado de mantener la operativa informática 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] en base a una arquitectura distribuida. Distintas universidades aportan parte de su infraestructura tecnológica para dar soporte a este despliegue. 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* privados 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* permite tener una gestión local de la infraestructura y compartida de la información de usuarios. Así, el nivel de confianza entre las entidades federadas rige el control de acceso a los recursos presentados.

A nivel de mantenimiento y hardware de la infraestructura, se tendrán unos costes fijos; el tráfico podrá entrar y salir sin restricciones ajenas a la organización (*egress fees*) y la escalabilidad de los recursos no tendrá 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 se tiene 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.

Se pretende establecer una federación en *OpenNebula* [1], respaldada por el sistema de almacenamiento, Ceph [2], con servicios de catálogo y confianza. Además, para tener la madurez de un sistema real, se ha diseñado una solución de recuperación ante desastres que permita recuperar configuraciones, estado del sistema y de máquinas virtuales y el catálogo de la federación.

[1] https://boira.es/

[2] https://docs.opennebula.io/6.10/installation_and_configuration/data_center_federation/overview.html

La migración entre zonas queda fuera de este proyecto ya que la federación en *OpenNebula*[2] se ha diseñado como un sistema de compartición de recursos estáticos. Los recursos esenciales de *OpenNebula* (scheduler, OneFlow y OneGate para despliegue de servicios)[3] se gestionan localmente.

La solución de recuperación ante desastres se centra en presentar una arquitectura funcional que permita guardar la información de la federación. Esta llega hasta el almacenamiento principal, Ceph. Una gestión más tradicional del almacenamiento, en este ámbito, queda fuera de este proyecto.

1.3. Objetivos

El objetivo principal del proyecto es definir, implementar y validar un modelo de federación entre dos instancias de *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 dará 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 intervendrán un sistema de monitorización y otro de control de accesos que validará las políticas definidas.

1.4. Metodología

El proyecto se ha desarrollado 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 han definido la arquitectura, la infraestructura base y las políticas iniciales. Posteriormente, han implementado 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 han servido 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

[3] https://docs.opennebula.io/6.10/installation_and_configuration/opennebula_services/overview.html

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 arbitrario 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* privada viene ya dada para el proyecto, *OpenNebula*, que permite federar varias instancias en lo que se denominan zonas. 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 el sobreaprovisionamiento, las listas de control de accesos y la alta disponibilidad, de la infraestructura y las máquinas virtuales. El sobreaprovisionamiento es la técnica por la que se asignan más recursos de los disponibles a máquinas virtuales, basándose en el hecho de que no todas las máquinas van a consumir todos sus recursos en un momento concreto. 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 o frontal (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, se ha trabajado sobre 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, se ha trabajado con 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 interfaz de dispositivo de bloque 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 la importancia de que cubran todos los componentes de la infraestructura, usuarios y servicios de la federación. Esta última, 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 empleadas estudiadas para el desarrollo de la solución se encuentra la Infraestructura como Código (*IaC* en inglés). Esta se define como el proceso de gestión y aprovisionamiento de recursos informáticos en una infraestructura *cloud*, a través de ficheros de definición interpretables por una máquina. 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. 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.

Capítulo 3.

Análisis y Diseño del sistema

3.1. Análisis del problema

Se ha desarrollado la infraestructura que da soporte a las instancias *cloud*. El diseño de la infraestructura tiene en consideración la **tolerancia a fallos y alta disponibilidad**.

Al estar desplegado en alta disponibilidad, el entorno *cloud* local requiere bajas latencias entre frontales 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 definidas **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 es 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 ha considerado 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.

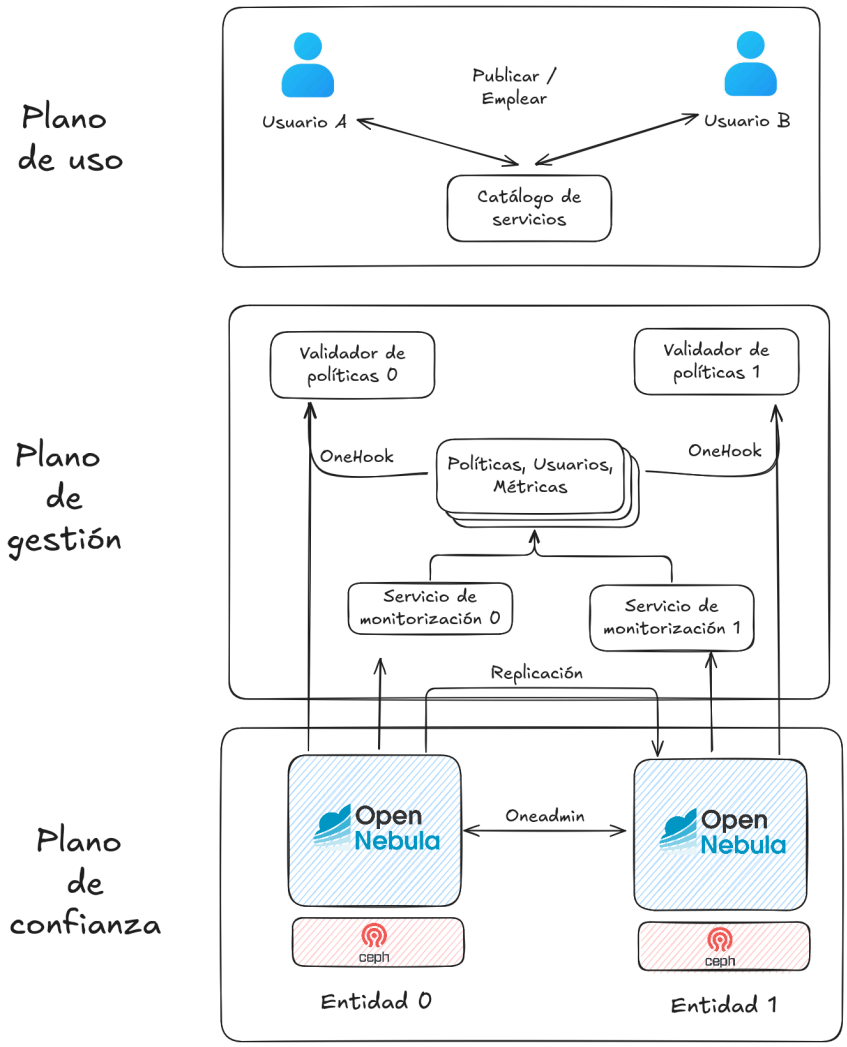

Figura 1: Arquitectura del prototipo de federación.

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 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 instancias de *OpenNebula*, cada una de ellas está desplegada en alta disponibilidad. A este nivel, y con la arquitectura de federación de *OpenNebula*, se ha establecido

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 ha establecido 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 una política de nombrado, que cubren máquinas virtuales, imágenes y *hooks*; estas son de gran utilidad para administrar estos y más recursos. Se han definido 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 E.

Hay un sistema de validación de políticas en cada entidad federada, dotando de alta disponibilidad a este servicio. Se ha planteado este servicio en un modo pasivo, es decir, que este sistema dicta si un recurso presentado cumple con las políticas, 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. Cada entidad federada cuenta con este sistema, dotando no solo de disponibilidad si no completitud de las métricas. La infraestructura y los servicios que está ofreciendo a la federación son el objeto de este sistema. 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 sistema de gestión de eventos que ofrece *OpenNebula* permite ejecutar la lógica personalizada que implementa este flujo de trabajo.

```
// 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.

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* y compartidos en la federación. Están sujetos a una serie de *ACL* y 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"[4], el cual es accesible desde las zonas de una federación, donde publicar los recursos mencionados ligados a una serie de permisos. Este modelo es el más adecuado dada su integración nativa con *OpenNebula*. Cada entidad cuenta con una interfaz que permite la conservación de los recursos alojados en el *Marketplace*.

Un usuario se crea en una entidad y se le asigna al grupo por defecto de esa entidad, extendiendo sus permisos mediante 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 de los disponibles en la federación.

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

3.3. Diseño de la Infraestructura

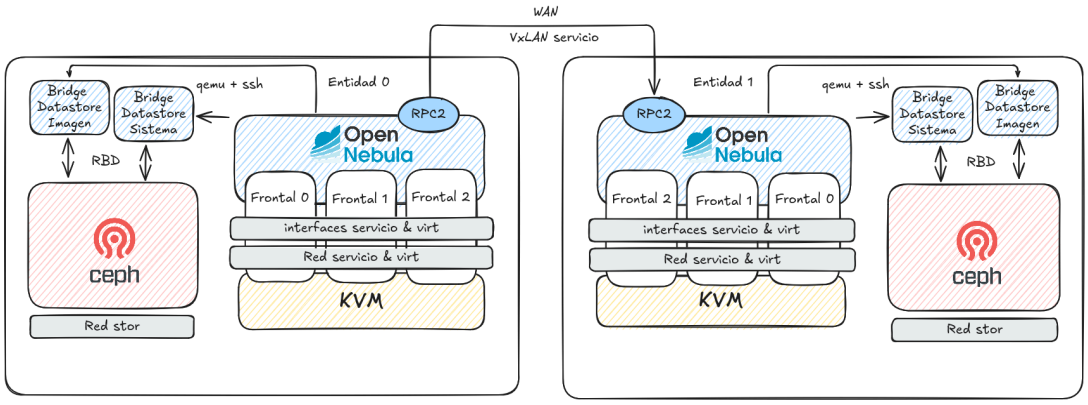

Figura 2: Diseño de la infraestructura multi sede.

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 frontales, 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 frontales como "máquinas salvajes", se consigue tener una infraestructura autocontenida. Cada frontal se coloca en un servidor distinto, asumiendo que cada entidad cuenta con, al menos, tres servidores de virtualización. Los datastores de *OpenNebula* se conectan con el driver Ceph y tienen como servidor puente los servidores de virtualización.

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

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 han definido 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 ha sido 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 han configurado 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 G.2 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 ha tenido que adoptar 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 han diferenciado 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[5] 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 que implementadas.

En Ceph se almacenan el estado de los frontales 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 G.1 se trata en detalle el módulo de almacenamiento implementado.

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

3.3.3. Despliegue

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.

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.

Durante la primera etapa del despliegue, se han declarado varias fases diferenciadas 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 cuenta en el párrafo siguiente. Con esto, ya se pueden desplegar los frontales 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 emplearán directamente los servicios de la federación. Esta primera etapa se comenta más en detalle en la Sección 4.1.3.

Dentro de esta primera etapa, para el caso de Ceph, 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 con, al menos, un *gateway* de RADOS en cada zona. Este despliegue se refleja en el Anexo G.1.

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.

En la siguiente etapa de despliegue están definidos los usuarios, grupos y permisos, creados en una primera fase, con los que desplegar el backup, validador de políticas y catálogo de servicios, cada uno en una fase distinta.

Existen recursos que tienen dependencias que no se pueden expresar mediante las herramientas de despliegue automático utilizadas. Por lo general, estas herramientas 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.

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 ha creado desde *OpenNebula*. En caso contrario, se suspende momentáneamente 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.

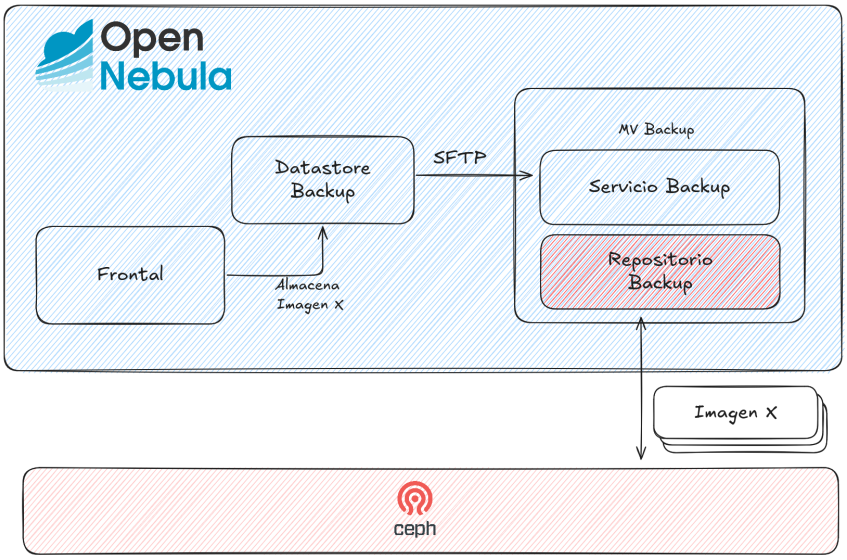

Figura 3: Arquitectura del servicio de backup para cada entidad.

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 frontal de *OpenNebula* y el servicio de backup escogido. Este protocolo es una restricción de *OpenNebula*[6] 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 ha configurado 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.

La implementación de la infraestructura de red que soporta al backup en cada entidad es libre, pero está sujeta a los SLA acordados para backup. Para ello se propone una red basada en la tecnología fibre

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

channel, con posibilidad de utilizar FCoE si se cuenta con mecanismos de DCB en la infraestructura interna de la entidad.

Capítulo 4.

Implementación

Para implementar el prototipo de federación diseñado se han empleado 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 ha empleado 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, se ha simulado el entorno real mencionado en el diseño. Este entorno consiste en un único servidor de virtualización, los problemas que conlleva se mencionan en el Anexo C.1.

El servidor en el que se ha simulado la infraestructura, tiene dos unidades de procesamiento de 32 hilos en modo *hyper-threading* (64 en total), tres discos de 15 mil revoluciones de 1 TiB de capacidad y dispuestos en RAID5.

4.1. Implementación de la Infraestructura

El despliegue está basado en contenedores *Podman*. Sabiendo que el backup y el sistema de control de accesos se desplegarían en máquinas virtuales, se ha hecho la infraestructura lo más ligera posible. Cada contenedor se ejecuta en modo no privilegiado aunque se le han asignado las *capabilities* *NET_RAW* y *NET_ADMIN* que permitan modificar la red. Cualquier acceso a dispositivos físicos, en el caso de OSDs o asegurar la persistencia de las bases de datos, se realizado a través de volúmenes con los dispositivos previamente configurados y montados en una ruta específica en el host.

En el Listado 3 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 frontales en alta disponibilidad de la zona maestra, incorporando después la zona esclava, salvando las relaciones de orden entre ellas.

4.1.1. Red

La Figura 4 pretende simular la topología de red presentada en la fase de diseño, teniendo únicamente un servidor. La distinción geográfica entre sedes se consigue mediante *VLANs*, haciendo que los componentes de cada entidad tengan que pasar por el router más cercano para poder comunicarse. Así, las interfaces de cada contenedor han sido configuradas correspondientemente con la *VLAN* a la que pertenece su servicio.

Para simular la comunicación interna de cada entidad, se han empleado interfaces tipo bridge huérfano que permiten simular el enrutado de capa 2 de un switch. Hay cuatro de estas interfaces: para la red pública Ceph, para la privada de Ceph, la red de replicación para OSDs; la pública de *OpenNebula*, para frontales, base de datos y acceso a la federación; y a la que se acoplan las máquinas virtuales. La interconexión entre los bridge se ha conseguido mediante interfaces de pares VETH. Así, la figura del router queda relegada para el acceso a internet o comunicación entre VLAN diferentes. Cabe destacar

```
class virt {
    # Despliegue de la red Podman
    [...]
    # Despliegue del gateway
  class {'virt::services::gateway':
              require    =>    $virt::containers::deployment_units.map    |$du|
{ Virt::Podman_network["network-$du"] },
  }
    # Despliegue de clusters ceph
  $virt::containers::deployment_units.each |$id| {
    storage::ceph {"ceph-$id":
      require => Class["virt::services::gateway"],
      id => $id
    }
  }
    # Creación cliente s3
    [...]
    # Despliegue de zona maestra en HA
    virt::services::opennebula { "master-zone":
        require => Storage::Ceph["ceph-10"],
        id => '10',
        mode => 'ha',
        zone_id => '0'
    }

    # Despliegue de un frontal en zona esclava
    [...]

    # Despliegue de la federación
  virt::services::opennebula::federation { "nebula-federation":
    require => Virt::Services::Opennebula["slave-leader"],
    master => '10',
    slaves => ['20']
  }

    # HA de la zona esclava y adhesión de nuevos frontales
    virt::services::opennebula::ha { "slave-zone-ha":
        require => Virt::Services::Opennebula::Federation["nebula-federation"],
        oned => $virt::containers::oned_services['20'][0],
        zone => '100',
        vip => $virt::containers::one_vip['20']
    }
    [...]
    # Creación de Cephfs para OPA y backup
    # Corrección de problemas por el entorno simulado
    [...]

    # Servicio de monitorización
    $virt::containers::deployment_units.each |$id| {
        virt::services::opennebula::monitoring { "monitoring-$id":
            require => Virt::Services::Opennebula["slave-followers"],
            zone_id => $id
        }
    }
}
```

Listado 3: Despliegue, en Puppet, *de la federación* OpenNebula, *zonas en HA y pooles de Ceph.*

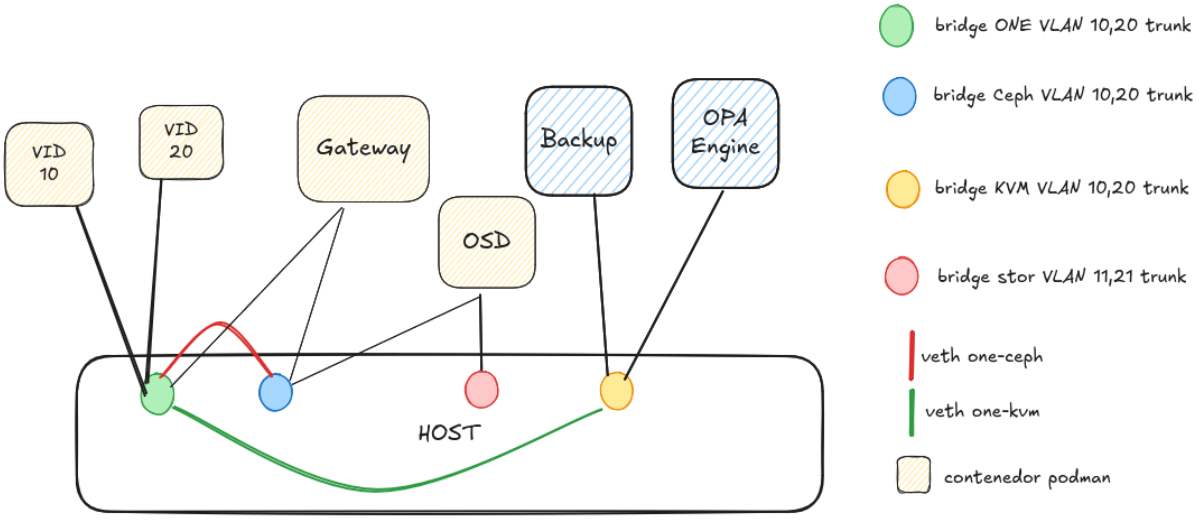

Figura 4: Infraestructura de red simulada

que ambas interfaces mencionadas cuentan con la capacidad de *trunk* para poder transmitir paquetes de varias VLAN.

Se ha definido una red frontal de todo el sistema que controle el acceso a Internet. Así, 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 4. 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. Esta configuración plantea un problema documentado en la Sección C.2.

```
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 4: Reglas de IPTables para retransmisión de VLAN.

La red de contenedores se implementado mediante *macvlan*, redes virtuales que se acoplan a una interfaz física, sobre las interfaces VLAN en cada bridge.

4.1.2. Almacenamiento

Cada *host* en el que se pretenden desplegar OSDs, tiene definido un grupo volumen. Este cuenta con un volumen físico para los discos físicos y tantos volúmenes lógicos como necesite cada componente de la infraestructura, y arquitectura. Así, para los OSDs, que en total son seis, se han instanciado seis volúmenes lógicos, permitiendo el uso de forma más granular del almacenamiento físico en cada *host*. 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 emplean 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 frontales se han guardado como imágenes en un pool de bases de datos.

Prometheus almacena sus métricas en una base de datos 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 ha empleado un volumen de contenedor cuyo soporte subyacente es un disco RBD de Ceph. Se ha creado una nueva imagen en el pool empleado para las bases de datos.

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*.

4.1.3. Despliegue de la infraestructura

Se ha escogido 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 han definido los módulos *net*, *storage* y *virt* para el despliegue de cada parte de la infraestructura por separado.

En el módulo *net* están definidos los recursos de red definidos en la Sección 4.1.1 con dependencias de despliegue internas: primero se despliegan los *bridge* y después, sobre estas, las interfaces VETH.

El módulo *storage* 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.

El módulo *virt* contiene la definición de los componentes relacionados con la infraestructura de virtualización. Esta consiste en el *gateway* y los componentes de *OpenNebula*. El contenedor de la puerta de enlace es el primero en ser desplegado, seguido de las bases de datos y los frontales de *OpenNebula*. Posteriormente se establece la comunicación entre cada frontal, formando un único clúster en alta disponibilidad. Se termina ejecutando los demonios encargados de exportar métricas y desplegando el contenedor Prometheus.

Entre los varios recursos personalizados que se han definido en Puppet, cabe destacar el de aprovisionamiento de contenedores. 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 frontales 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 frontales en alta disponibilidad que forman la zona maestra de la federación, seguido viene el frontal maestro de la segunda zona con el que se establece la federación. Una vez establecida, se incorporan dos frontales más a la zona esclava.

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 y comunicación mediante el uso de *hooks* de *OpenNebula*.

4.2.1. Sistema de control de accesos y validación de políticas

Para implementar este sistema se ha escogido *OPA* (*Open Policy Agent*) [10], que es un validador de políticas expresadas en el lenguaje de dominio específico *Rego*[7]. 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*[8]. Para ello, se ha creado una máquina virtual cuya imagen de disco no persistente, que tiene instalado *Debian*[9], usan el backup y esta. 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 ha instalado *Docker* y se ha desplegado *OPA*. En el anexo Sección C.3 se detalla un problema encontrado con este despliegue.

Se ha definido 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 ha desplegado localmente con exposición en el entorno físico-virtual de la máquina desde el puerto *2345*.

```
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 5: 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 servicios; usuarios de la federación, almacenamiento, compatibilidad técnica, backup y despliegue de aplicaciones finalistas. Estas se encuentran especificadas en el anexo Sección B.

Las políticas de nomenclatura y sobrecarga se validan mediante el lenguaje *Rego*, visto en el ejemplo del Listado 5, mientras que las demás están implementadas como código del despliegue en Puppet.

[7] https://www.openpolicyagent.org/docs/policy-language
[8] https://www.docker.com/
[9] https://www.debian.org/

```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 6: Lógica implementada en Ruby como Hook OpenNebula *para control de despliegue tras validación de políticas.*

En el Listado 6 se resalta una parte de un *hook* implementado, visto en el anexo Sección F.1, que se ejecuta tras la creación de un recurso en *OpenNebula*, que tenga asociado este script ruby. 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 ha escogido *Prometheus*[10] 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.

Para la monitorización de la infraestructura, se ha desplegado un contenedor separado que ejecuta el software de Prometheus. Los exportadores de métricas se distribuyen como paquetes de software desde el repositorio oficial de *OpenNebula*. Hay desplegados agentes de exportación de métricas en los nodos de virtualización y en los frontales y datastores de *OpenNebula*. Así, se cubre el estado de los servidores de cómputo, el de *OpenNebula*, como servicio, y el almacenamiento de la federación, respectivamente. Las métricas son accesibles desde un mismo *endpoint*.

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

```
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 7: Grupo de reglas Prometheus que detectan las políticas de uptime y downtime.

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.

En el Listado 7 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.

4.2.3. Comunicación entre servicios

Para ligar los eventos de creación y eliminación de máquinas virtuales, imágenes y trabajos de backup con el sistema de validación se han definido dos hooks de *OpenNebula*, ambos son un script Ruby. Uno de ellos realiza una petición HTTP, con una consulta en PromQL, al endpoint de Prometheus. El otro, interpreta la plantilla en formato XML del recurso en cuestión, lo valida mediante una petición HTTP a la API REST de OPA y actúa consecuentemente con el resultado obtenido. En caso de no cumplir la política, el recurso se elimina a través de la interfaz de Ruby que ofrece *OpenNebula*.

La otra parte de la definición de un hook es una plantilla que interpreta *OpenNebula*. En ella se especifica la ubicación del script, el evento que inicie su ejecución y los argumentos para el programa. *OpenNebula* ofrece dos tipos de eventos a los que suscribir un hook: cambios de estado (*state hooks*) o peticiones API (*API hooks*). Se han suscrito 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.

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* ha sido la "Multi-zona"[11]. 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 usuario, 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 de un *bucket*, sobre el que se almacenan los objetos del *Marketplace* privado de la federación.

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 ha empleado el software *HAProxy* para este servicio y se ha definido 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*.

El *Marketplace* de la federación se ha definido en un manifiesto *OpenTofu* con 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.

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 ha escogido *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 ha tenido 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 ha desplegado una máquina virtual en la que se ha montado 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 han instalado los paquetes *rsync* y *qemu-img*, requeridos por el driver *Restic*; se ha seguido la política de nomenclatura de máquinas virtuales y etiquetado; y se han distribuido las claves públicas *ssh* de los tres frontales a la máquina virtual. La definición de la máquina virtual, usuario y *datastore* pueden encontrarse en el manifiesto *OpenTofu* de la Sección H.

Las tareas definidas, tienen en cuenta las políticas de backup. Para cumplir con ellas, se han especificado 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 8 se ve la aplicación de una tarea para los servicios básicos de *OpenNebula* (tres frontales en HA con sus bases de datos).

Para la incorporación de nuevos servicios al sistema de backup, se ha definido un *hook*. Este leerá la etiqueta de la máquina virtual para determinar su prioridad y asignará su identificador a la tarea que

[11] 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 8: Definición de la tarea de backup one-10-90 en OpenNebula

le corresponda según la prioridad. Habrá igualmente otro *hook* definido para el borrado de máquinas donde se eliminará su identificador de la tarea.

Capítulo 5.

Validación y Pruebas realizadas

5.1. Validación

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 ha validado 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 ha servido para detectar errores en la configuración o fallos en el despliegue. Creando usuarios en *OpenNebula*, en cualquiera de las zonas, se ha comprobado que la replicación interna funciona si se pueden listar los usuarios desde ambas zonas. Para Ceph, se ha controlado 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.

5.2. Pruebas

5.2.1. Prueba de sobrecarga

Esta prueba involucra recolectar 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 que se escapen del alcance del servicio de validación de políticas.

Se ha desplegado 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 ha probado que se impida el despliegue de una nueva máquina virtual.

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

Se ha visto como la herramienta *cpuburn*[12] 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.

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

5.2.2. Prueba de backup

Para probar el despliegue del backup se va ha lanzado una tarea en *OpenNebula* que almacena el estado de una de las bases de datos de uno de los frontales. 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 ha desplegado una nueva máquina virtual que actuará como el cuarto frontal 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.

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

Capítulo 6.

Conclusiones y Trabajo futuro

Se ha conseguido el objetivo principal del proyecto: establecer una federación entre dos instancias de *OpenNebula*, que sirva como prototipo para un posterior despliegue real con dos entidades en zonas geográficamente separadas. Se ha probado la validez de la infraestructura real a través del simulador. La conversión de componentes del entorno simulado al real será casi directa mediante el uso de los manifiestos de despliegue automático.

Pese a venir impuesta por el proyecto, *OpenNebula* ha resultado ser una solución de gestión de recursos virtuales efectiva y flexible, permitiendo desarrollar el modelo de federación planteado y acomodándose al contexto del proyecto. La capacidad que tiene de desplegarse de forma autocontenida se ha mostrado muy útil a la hora de plantear el backup y gestionar sus componentes. No obstante, la gestión de la confianza de serie podría ser mejorada, ya que requiere un acuerdo personal entre las entidades.

Ceph ha mostrado ser la tecnología de almacenamiento idónea para un entorno *cloud*, permitiendo gestionar la información en un espacio de nombres global cuando su naturaleza lo requiere (Marketplace), a la vez que cada entidad puede ser soberana de su información en los casos que así lo precisan (datastores locales). La definición del modelo de almacenamiento (replicación o erasure code) y del tipo (bloque para datastores, objeto para Marketplace y backup y fichero para políticas) utilizado según el uso que se le vaya a dar al dato almacenado, es otra característica que ha facilitado el diseño del sistema, empleando una misma herramienta.

Las pruebas han demostrado que los servicios principales funcionan como se esperaba, que la infraestructura soporta la traslación de recursos 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] «Parámetro host-passthrough». [En línea]. Disponible en: https://forum.opennebula.io/t/cpu-host-passthrough-and-scheduling-vms/3349

[15] «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

Anexos

Capítulo A

Diagrama de Gantt

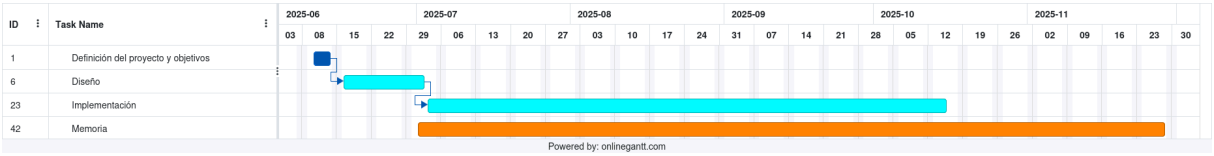

Figura 5: Diagrama de gantt.

Capítulo 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 5 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*[13] como driver del datastore de backup. El resto de servicios pueden tener una implementación diferente a la presentada en los siguientes apartados.

Capítulo C

Problemas encontrados

C.1 Múltiples instancias One en único servidor

Cada instancia de *OpenNebula* tendrá como único host de virtualización el mismo servidor donde se ha desplegado. Siendo que *OpenNebula* define dominios de *libvirt* al crear máquinas virtuales, y hay dos instancias de *OpenNebula* corriendo, existe la posibilidad de que se creen dominios con identificadores duplicados. Por ello, ha habido que modificar el estado de *OpenNebula*, a nivel de base de datos, para que una de las instancias comience con un identificador de máquinas virtuales superior al otro. La entidad esclava empieza por el identificador 100.

C.2 Bucles de red

La implementación del esquema de red del entorno de simulación es relativamente compleja. En este escenario, se ha detectado la presencia de bucles de red entre el router, los *bridge* y la interfaz VETH que

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

une los *bridge*. Para solucionar esta situación se ha activado el protocolo STP en los *bridge*, otorgándoles máxima prioridad para que la interfaz de VETH no quede en desuso. Por otro lado, la topología de red presentada es tolerante al fallo de la interfaz que une los bridges, a expensas de incrementar las latencias entre *OpenNebula* y Ceph, lo cual no supone problemas graves.

C.3 Contenedores en máquina virtual

Se ha de habilitar el modo de CPU *host-passthrough* [14], [15] 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"*

Capítulo D

Asignación de direcciones

La asignación de direcciones a cada uno de los servicios y componentes del sistema se expresa en la Tabla 1.

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.

Capítulo E

Implementación de políticas

E.1 Política de nombrado

```
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])
}
```

E.2 Política de sobrecarga

```
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
}
```

Capítulo F

Hooks *OpenNebula*

Todos los *hooks* definidos en *OpenNebula* empiezan 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"
```

F.1 Hook creación de recursos

```
#!/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
```

F.2 Hook eliminación de recursos

```
#!/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"
```

F.3 Librerías para los hooks

F.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

  # 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
```

F.3.2 *OpenNebula*

```
require "opennebula"
```

```ruby
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
```

F.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
```

Capítulo G

Manifiestos Puppet

Manifiesto principal

```
  # Despliegue del almacenamiento local
class { 'storage::lvm_stor':
  devices => ['/dev/sda'],
  lvs     => [
    { name => 'osd-0-10', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-1-10', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-2-10', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-3-10', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-4-10', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-0-20', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-1-20', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-2-20', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-3-20', size => '20G', mount => '/dev/none', fs_type => 'raw' },
    { name => 'osd-4-20', size => '20G', mount => '/dev/none', fs_type => 'raw' },
  ],
} ~>
# Despliegue de la red local
class { 'net::ifaces':
} ~>
# Despliegue de los servicios virtualizados
class { 'virt': }
```

G.1 Módulo de almacenamiento

G.2 Módulo de red

Interfaces

```
  # Author | Lucas Cauhé Viñao
# Contact | lcauhe@gmail.com
# Description | Host interfaces definition
# Path | net/manifests/ifaces.pp
class net::ifaces (
  ) {


  # Install debian routes provider
  # Docs -> https://github.com/voxpupuli/puppet-network/tree/14d7f2f7d225ef93b0bf5be39
b34506066132245?tab=readme-ov-file#dependencies
  if $facts['os']['name'] == 'Debian' {
    include 'network'
  }

  network_config { 'lo':
    ensure => 'present',
    onboot => 'true',
    family => 'inet',
    method => 'loopback',
    ipaddress => '127.0.0.1',
    netmask => '255.0.0.0',
  }
  network_config { 'eno3':
    ensure => 'present',
    onboot => 'true',
    method => 'manual',
    options => {
      'bridge_ports' => ['frontend']
    }
  }

  $bridge_ifaces = ['br_one', 'br_ceph', 'br_kvm', 'br_stor']

  $bridge_ifaces.each |$iface| {
    systemd::manage_unit { "${iface}.netdev":
      path => '/etc/systemd/network',
      netdev_entry => {
        'Name' => "${iface}",
        'Kind' => 'bridge',
      },

      bridge_entry => {
        'STP' => 'yes',
        'VLANFiltering' => 'yes',
      },
      service_restart => true
    }

    systemd::manage_unit { "${iface}.network":
      path => '/etc/systemd/network',
      match_entry => {
        'Name' => $iface,
      },
      bridge_vlan_entry => {
        'VLAN' => [10, 20],
```

```
    }
  }
}

systemd::manage_unit { "frontend.netdev":
  path => '/etc/systemd/network',
  netdev_entry => {
    'Name' => "frontend",
    'Kind' => 'bridge',
  },

  bridge_entry => {
    'VLANFiltering' => 'yes',
  },
  service_restart => true
} ->
systemd::manage_unit { "frontend.network":
  path => '/etc/systemd/network',
  match_entry => {
    'Name' => 'frontend',
  },
  network_entry => {
    'Address' => "10.0.13.71/24",
    'Gateway' => "10.0.13.254"
  },
} ->
systemd::manage_unit { "eno3.network":
  path => '/etc/systemd/network',
  match_entry => {
    'Name' => 'eno3',
  },
  network_entry => {
    'Bridge' => 'frontend'
  }
} ->
network_route { 'default':
  ensure => 'present',
  network => 'default',
  gateway => '10.0.13.254',
  interface => 'frontend',
  netmask => '0.0.0.0',
} ->
# VETH interfaces for interconnecting bridges as required
net::veth { "veth_one_ceph":
  br_src => 'br_one',
  br_dst => 'br_ceph',
  if_name => 'one_ceph',
  rev_if_name => 'ceph_one',
  allowed_vlans => [10, 20],
} ->
net::veth { "veth_one_kvm":
  br_src => 'br_one',
  br_dst => 'br_kvm',
  if_name => 'one_kvm',
  rev_if_name => 'kvm_one',
  allowed_vlans => [10, 20],
```

```
  } ->
  exec { "systemctl daemon-reload && systemctl restart systemd-networkd ":
    path => '/usr/bin'
  }

}
```

Interfaz VETH

```
  # net/manifests/veth.pp
define net::veth (
  String  $br_src = 'br0',
  String  $br_dst = 'br1',
  String  $if_name = 'br0_br1',
  String  $rev_if_name = 'br1_br0',
  Array[Integer] $allowed_vlans = [],
) {

  systemd::manage_unit { "${if_name}.netdev":
    path => '/etc/systemd/network',
    netdev_entry => {
      'Name' => $if_name,
      'Kind' => 'veth',
    },
    peer_entry => {
      'Name' => $rev_if_name,
    }
  }

  systemd::manage_unit { "${if_name}.network":
    path => '/etc/systemd/network',
    match_entry => {
      'Name' => $if_name,
    },
    network_entry => {
      'Bridge' => $br_src,
    },
    bridge_vlan_entry => {
      'VLAN' => $allowed_vlans
    }
  }


  systemd::manage_unit { "${rev_if_name}.network":
    path => '/etc/systemd/network',
    match_entry => {
      'Name' => $rev_if_name,
    },
    network_entry => {
      'Bridge' => $br_dst,
    },
    bridge_vlan_entry => {
      'VLAN' => $allowed_vlans
    }
  }
}
```

Capítulo H

Manifiestos OpenTofu