Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño de un cloud privado flexible

Design of a flexible private cloud

Autor

Jorge Senso Juanas

Director

Unai Arronategui Arribalzaga

Universidad de Zaragoza

Escuela de Ingeniería y Arquitectura

Febrero 2025

RESUMEN

El trabajo aborda la necesidad de crear entornos flexibles para prácticas académicas en asignaturas de administración de sistemas distribuidos, dado que su configuración y mantenimiento representan un desafío significativo. Se busca enriquecer la experiencia de aprendizaje de administración de estos entornos mediante la posibilidad de puesta en funcionamiento de diferentes tipos de cloud privado.

El objetivo principal es desarrollar una aplicación capaz de modelar y gestionar de forma básica los principales componentes de un cloud privado, como máquinas físicas, virtuales, contenedores, plataformas distribuidas, sistemas de almacenamiento y elementos de red, así como los servicios auxiliares necesarios para su funcionamiento.

A partir de un análisis exhaustivo del estado del arte, se refinan los requisitos para el diseño de la aplicación, determinando los elementos tecnológicos (Libvirt, Openstack y Kubernetes) y de despliegue para la gestión de máquinas físicas, máquinas virtuales y contenedores. Esta sigue la arquitectura Clean, para garantizar modularidad y mantenibilidad, mediante el diseño del modelo de entidades de la aplicación, una lógica de negocio que define su comportamiento y una serie de adaptadores que median la utilización de las diferentes tecnologías externas utilizadas. Se validó la operativa de aprovisionamiento de máquinas físicas, gestión de máquinas virtuales y despliegue de plataformas distribuidas mediante pruebas graduales en entornos virtuales y físicos.

En conclusión, el sistema satisface los objetivos, proporcionando una solución robusta y flexible. Proponiéndose como trabajo futuro el soporte a modelos híbridos, mejoras en la interacción y profundizar en cuestiones de seguridad.

ABSTRACT

The work addresses the need to create flexible environments for academic practices in courses on distributed systems administration, given that their configuration and maintenance pose a significant challenge. The aim is to enhance the learning experience in managing these environments by enabling the deployment of different types of private clouds.

The main objective is to develop an application capable of modeling and managing, at a basic level, the main components of a private cloud, such as physical machines, virtual machines, containers, distributed platforms, storage systems, and network elements, as well as the auxiliary services necessary for their operation.

Based on a comprehensive analysis of the state of the art, the requirements for the application design are refined, determining the technological elements (Libvirt, Openstack, and Kubernetes) and deployment methods for managing physical machines, virtual machines, and containers. The application follows the Clean Architecture to ensure modularity and maintainability, through the design of the application's entity model, a business logic layer that defines its behavior, and a series of adapters that mediate the use of various external technologies. The provisioning of physical machines, management of virtual machines, and deployment of distributed platforms were validated through gradual testing in both virtual and physical environments.

In conclusion, the system meets the objectives, providing a robust and flexible solution. Future work proposes supporting hybrid models, improving interaction, and delving deeper into security issues.

Índice

AGRADECIMIENTOS

A mi padre, Óscar Blasco, Paul Hodggets, Unai Arronategui y Eduardo Fiat por enseñarme y confirmarme mi vocación.

A mi abuela Ana y mi hermano Marcos, ojalá estuvierais aquí.

A mi madre, por su apoyo incondicional e interminable.

A Irene, por ayudarme a entender este mundo, guiarme cuando me pierdo en mí y siempre empujarme hacia adelante.

A Alba García, Rubén París, Felipe Esteban, mi abuelos Felipe y Mª Carmen, y mis tías Vicky y Reme, por vuestro cariño y estar siempre presentes.

A Dani Lafuente, Dani Luna, Andrés Ciria, Mario Álvarez, Jorge Calvo, Alberto Hernando, Carlos Mayo, Simón Bosque, Paula Gironés, Antorio Herrero, Óscar Segarra y a todos los me hicieron sentir en casa lejos de ella.

A todos mis compañeros, y ahora amigos, que me acogieron con los brazos abiertos en Zaragoza.

Capítulo 1

Introducción y objetivos

1.1. Motivación

En el Grado de Informática existen múltiples asignaturas cuyas prácticas presentan unos requisitos complejos a nivel de infraestructura. Un claro ejemplo de esta situación es ”Administración de Sistemas 2”. La cual, centrada en la administración de sistemas distribuidos (*cloud*), requiere de entornos de prácticas complejos y heterogéneos, que permitan interactuar con diversos sistemas de almacenamiento y redes, así como con plataformas distribuidas.

La gestión de estos entornos, englobando todo el ciclo de vida, desde su configuración y puesta en marcha, hasta su mantenimiento, se trata de una tarea altamente demandante. Así mismo, esta asignatura presenta un matiz especial, puesto que los escenarios planteados en las prácticas expresan similitudes con la infraestructura que las subyace.

Por tanto, este trabajo se plantea como una aplicación de apoyo para la puesta en marcha de la infraestructura computacional de prácticas, permitiendo definir, modificar y desplegar de forma flexible diversas arquitecturas de cloud privado, incluyendo gestión de máquinas físicas, virtuales y plataformas distribuidas.

1.2. Objetivos y alcance

El principal objetivo del trabajo consiste en el diseño y desarrollo de una aplicación capaz de modelar y gestionar los diversos componentes de un cloud privado, incluyendo máquinas físicas, máquinas virtuales, contenedores, plataformas distribuidas, elementos de red y sistemas de almacenamiento que lo constituyen.

Dicha gestión engloba el control del ciclo de vida de las diversas máquinas y el despliegue de plataformas sobre ellas, abarcando aquellos servicios necesarios para el correcto funcionamiento de cualquier sistema distribuido, englobando servicios de nombres, tiempo, gestión de direcciones IP, identidad y monitorización.

1.3. Metodología

La metodología planteada parte de una fase inicial de estudio del estado del arte, análisis del problema y recogida de requisitos, tras la cual, tiene lugar un desarrollo iterativo e incremental de la aplicación, alternando fases de diseño, desarrollo y pruebas. El resumen de la planificación se muestra en el Anexo A.

Capítulo 2

Estado del arte

2.1. Conceptos

Esta sección desarrolla los conceptos fundamentales sobre los que se desarrolla el trabajo.

Cloud privado

Un cloud privado está compuesto, en esencia, por un conjunto de servicios distribuidos, sustentados sobre un soporte físico o virtual y que tiene como finalidad ofrecer un servicio.

Los soportes necesarios para el funcionamiento de un cloud privado son las máquinas, elementos de red y sistemas de almacenamiento, así como componentes eléctricos y de climatización. Siguiendo esta definición, un sistema distribuido capaz de ofrecer alguno de estos elementos puede servir de soporte para otro.

Elementos de red

Se entiende como elementos de red a aquellos que, independientemente del medio y tecnología empleados, permiten la comunicación e interacción entre sistemas.

Sistemas de almacenamiento

Se definen como sistemas de almacenamiento a aquellos que permiten el almacenamiento persistente y recuperación de información, de forma independiente del medio subyacente y tecnologías empleadas.

Monitorización y observabilidad

La monitorización consiste en el proceso sistemático de recuperación de información de un sistema informático, para su posterior análisis y entendimiento, pudiendo incluir visualizaciones con el fin de facilitarlo. [1][2]

La observabilidad se puede establecer como una parte del estudio de la monitorización, enfocado principalmente a la obtención de información para conocer el estado o condición interna de un determinado sistema. [1]

Gestión de direcciones IP

La gestión de direcciones IP, también denominada IPAM, engloba la gestión de nombres y direcciones IP, generalmente provistas por servicios DNS y DHCP respectivamente.

Identidad

La identidad se refiere al conjunto de atributos o características que permiten identificar a una entidad, independientemente si se trata de un individuo, organización, aplicación o dispositivo. [3]

Además, se define como autenticación al proceso de verificación de la identidad de un individuo, y se define autorización a la asignación de permisos a una entidad autenticada.

Por tanto, se define servicio de identidad a aquel que permite la gestión de identidades, permitiendo que sólo las entidades autenticadas

y autorizadas tengan acceso sobre los recursos establecidos.

2.2. Tecnologías

Existe una serie de tecnologías que abordan diferentes partes de la problemática abordada por este trabajo.

Gestión de máquinas virtuales y contenedores

En primer lugar, plataformas de virtualización distribuidas como Openstack, Opennebula, Proxmox, LXD o Vsphere se centran principalmente en la gestión de máquinas virtuales, aunque algunas presentan ciertas posibilidades de gestión de contenedores. En particular, enfatizar que Openstack es una plataforma de sistema de código abierto utilizado en la mayoría de las empresas de cloud público, salvo los tres principales proveedores (Amazon AWS, Microsoft Azure y Google GCP).

Así mismo, cabe destacar Libvirt, plataforma de máquina única en red que permite la gestión de máquinas virtuales de forma más básica con selección directa de nodos remotos sobre los que operar y abstrayendo el hipervisor empleado.

Por otro lado, la gestión de contenedores gira principalmente en torno a Kubernetes y sus distribuciones, como Openshift, Mirantis, Rancher o Fury. Si bien es cierto que existen alternativas como Nomad, su relevancia es considerablemente más reducida.

En relación a Kubernetes, cabe destacar la existencia de tecnologías como Kubevirt, que lo extiende y permite la ejecución de máquinas virtuales. Así como Metal3, que aporta una extensión para gestionar máquinas físicas, aunque fuertemente ligadas al entorno de Kubernetes.

Además, se presenta una curiosa dicotomía, al existir la posibilidad de desplegar Kubernetes sobre Openstack, mediante Magnum, como

el proceso inverso, desplegando la infraestructura de Openstack sobre Kubernetes, mediante Helm. [4]

Gestión de máquinas físicas

Por su parte, la problemática de gestión de máquinas físicas es el aspecto con menor número de oferta tecnológica con diferencia, siendo Canonical MAAS y Openstack Ironic las principales. Además, cabe destacar que Ironic ofrece en términos generales una mayor flexibilidad, puesto que MAAS está diseñado con el ecosistema de Canonical en mente.

También merece atención la diversidad de protocolos y herramientas de control remoto que presentan las máquinas físicas. Fuera de los estándares IPMI y Redfish, encontramos un sinfín de opciones, generalmente ligadas al fabricante, de entre las que se destacan iLO (HP), iDRAC (Dell), AMT (Intel) o IMC (Cisco). Esta diversidad dificulta en cierta medida la gestión de máquinas en entornos heterogéneos.

Modelización de recursos

En el ámbito de modelización, encontramos distintas alternativas, según su alcance podemos distinguir entre modelización de infraestructura, sistemas locales y sistemas distribuidos.

Como herramientas de modelización de infraestructura, agnósticos, encontramos enfoques declarativos como el de Terraform u orientados a librería, como Pulumi. También encontramos herramientas ligadas a proveedores, como CloudFormation u Openstack Heat.

En el ámbito de la modelización de sistemas locales, los principales actores son Ansible, Puppet y SaltStack, los cuales plantean arquitecturas y enfoques dispares. Tanto Ansible como SaltStack cuentan con un lenguaje procedural, mientras que Puppet opta por uno declarativo. Así mismo, la operativa de Puppet y Chef es agente-servidor, mientras que

la planteada por Ansible o Puppet Bolt, es únicamente cliente, realizando la operativa mediante SSH. [5][6]

Finalmente, dentro de la modelización de sistemas distribuidos sólo podemos encontrar a Juju como una opción, abarcando todos los aspectos a excepción de las máquinas físicas. [7]

Sistemas de almacenamiento

En relación a los sistemas de almacenamiento, cabe destacar NFS por su sencillez operativa, así como otras opciones más sofisticadas como GlusterFS o Ceph, siendo el último un caso bastante interesante por sus capacidades de tolerancia a fallos y permitir el uso, por parte de los clientes, de dispositivos de bloque, almacenamiento de objetos y sistema de ficheros distribuido.

Gestión de direcciones IP y descubrimiento de servicios

Para lograr esta tarea, se puede hacer uso de servidores DNS como NSD, bind o PowerDNS, junto a servidores DHCP como ISC o Dnsmasq. Dentro de las herramientas encargadas de realizar la gestión de asignaciones podemos encontrar Netbox o phpIPAM.

Identidad y secretos

Como soluciones de identidad, se dispone en primera instancia, de sistemas LDAP tradicionales, como 389 Directory Server, OpenLDAP o soluciones más completas como FreeIPA. Así como herramientas con capacidades de control de acceso y gestión de identidad más avanzadas, como Keycloak, Authelia, Ory o Authentik.

Además, como soluciones de gestión de secretos, claves e información sensible, podemos encontrar productos como Vault, OpenBao o Conjur.

Monitorización

Finalmente en el ámbito de la monitorización, encontramos un amplio abanico de herramientas, como son agregadores de logs, como Fluentd o Logstash, herramientas de recolección y procesado, como Prometheus, Loki o Graphite, de visualización como Grafana o Kibana, e incluso soluciones integrales como Zabbix, ELK Stack o Graylog.

Capítulo 3

Análisis y diseño

3.1. Análisis

3.1.1. Evaluación del estado del arte

Se puede observar que no existe en la actualidad una herramienta que ofrezca una solución completa a la problemática planteada. La elección tecnológica concreta se trata de una tarea complicada, puesto que se trata de un escenario complejo en el que intervienen una cantidad extensa de servicios. Además, la tendencia actual muestra que ciertos productos tratan de abarcar todo, incluso solapándose entre ellos, agravando la dificultad a la hora de tomar una decisión. En este contexto, dada la ausencia de deuda tecnológica de este trabajo, se decide optar por una metodología que selecciona y combina aquellos conceptos y tecnologías más adecuados para cada aspecto planteado en los objetivos. En esta línea, se concretan una serie de requisitos más detallados que precisan los objetivos iniciales.

Con respecto a plataformas de cloud privado, se ha optado por abordar el despliegue y gestión de máquinas físicas mediante Bifrost, tecnología derivada de Openstack, máquinas virtuales mediante tecnologías Libvirt y Openstack, y contenedores mediante tecnología Kubernetes.

3.1.2. Requisitos

La tabla 3.1 define formalmente los requisitos obtenidos a partir de objetivos del trabajo.

RF1	El sistema ha de permitir la creación y destrucción de máquinas virtuales
RF2	El sistema ha de permitir el aprovisionamiento de máquinas físicas (*baremetal*) previamente registradas
RF3	El sistema ha de permitir el despliegue de plataformas distribuidas. Incluyendo como mínimo Libvirt (RF3.1), Openstack sin alta disponibilidad (RF3.2) y con alta disponibilidad (RF3.3), Kubernetes sin alta disponibilidad (RF3.4) y con alta disponibilidad (RF3.5).
RF4	El sistema ha de permitir el control del encendido y apagado de máquinas
RF5	El sistema ha de guardar registro de las diversas máquinas y las plataformas desplegadas sobre ellas
RF6	El sistema ha de ofrecer servicios de trazabilidad y monitorización
RF7	El sistema ha de ofrecer servicios de gestión de identidades y autorización, así como de gestión de secretos
RF8	El sistema ha de ofrecer servicios de tiempo, nombres (DNS), gestión de direcciones IP y descubrimiento de servicios
RF9	El sistema ha de permitir emplear la plataforma de almacenamiento distribuido NFS

Tabla 3.1: Requisitos funcionales del trabajo

3.1.3. Casos de uso

A partir de los requisitos previamente descritos, se definen los siguientes casos de uso. Se considera únicamente al administrador como actor, puesto que no se plantea la interacción de usuarios con la herramienta.

Gestión de máquinas físicas (*baremetal*)

- **Registrar una máquina física** en la aplicación.
- **Aprovisionar una máquina** a partir de una imagen de disco.
- **Eliminar una máquina física** de la aplicación.

Gestión de máquinas virtuales

- **Desplegar una máquina virtual** sobre una máquina física remota.
- **Destruir una máquina virtual**, eliminándola del sistema.

Gestión de plataformas

- **Desplegar una plataforma** partiendo desde el aprovisionamiento de las máquinas físicas, hasta la configuración completa de la misma.

Gestión de imágenes

- **Añadir una imagen**
- **Recuperar una imagen**
- **Eliminar una imagen**
- **Modificar una imagen**

3.2. Diseño

3.2.1. Arquitectura del sistema

A continuación se describen los principales elementos del sistema.

Servicios de apoyo, aquellos que proveen los servicios básicos para la correcta operativa del sistema, comprendiendo servicios de tiempo, identidad, secretos, gestión de direcciones IP, servicio de nombres y monitorización.

Aplicación de despliegues, engloba la aplicación en sí, junto a los servicios y otros elementos auxiliares para permitir su correcta ejecución.

Máquinas disponibles, refiriéndose a aquellas máquinas pertenecientes al sistema y que se encuentran disponibles para su gestión mediante la aplicación.

Estos elementos han de tener conectividad de red entre sí, pero no presentan limitaciones particulares a nivel de topología de red.

La Figura 3.1 esquematiza la arquitectura.

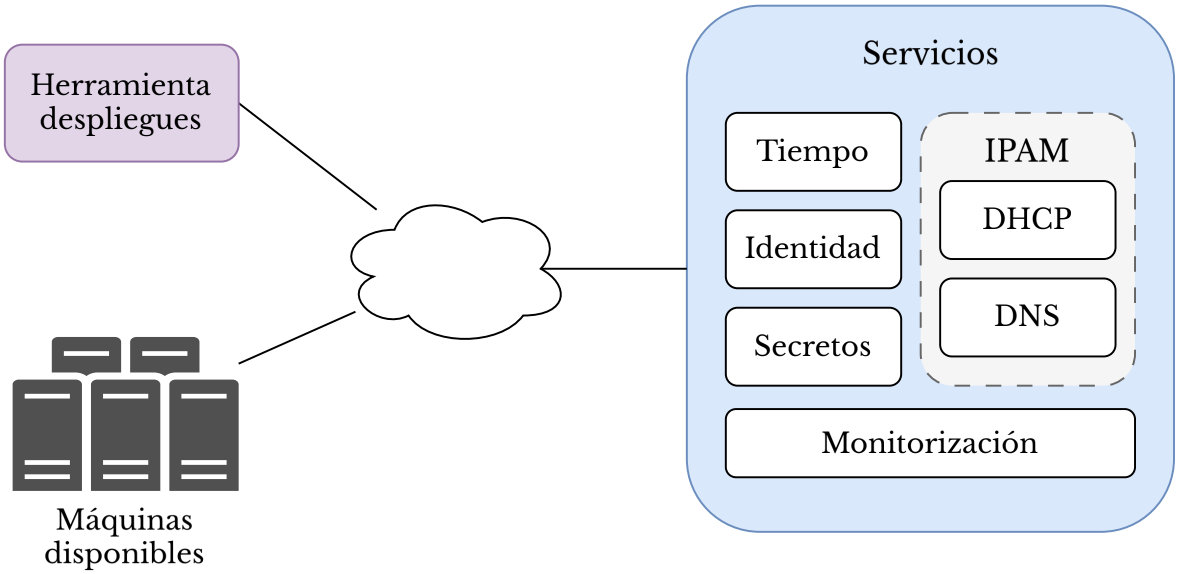

Figura 3.1: Arquitectura del sistema

3.2.2. Arquitectura software

En base a los requisitos y casos de uso previamente definidos, se plantea que la aplicación se acogerá a la arquitectura Clean, con el fin de lograr un diseño fácilmente mantenible y extensible. [8] Dicha arquitectura se detalla en el Anexo B. Las siguientes secciones detallan las capas definidas, las cuales se pueden observar en Figura 3.3.

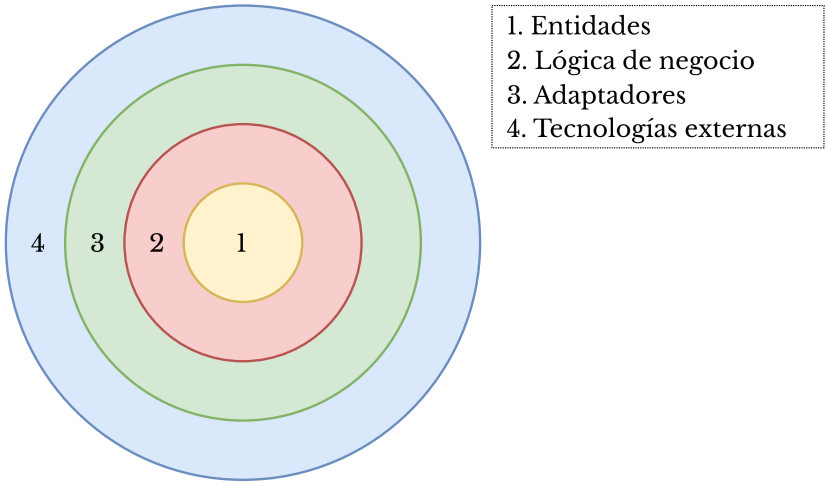

Figura 3.2: Capas definidas

3.2.3. Modelización de Entidades

Para modelar los distintos elementos del sistema, se han definido las siguientes entidades y sus relaciones:

Máquina, englobando máquinas físicas y virtuales. Contiene la información necesaria para su control remoto, incluyendo administración (IPMI, Redfish, etc.) y ejecución remota (SSH). Cada máquina guarda relaciones con sus sistemas de almacenamiento, redes, así como las plataformas desplegadas sobre ella.

Almacenamiento, representando los sistemas de almacenamiento que pueden ser empleados por las máquinas. Incluyendo almacenamientos locales como discos físicos, RAID o volúmenes virtuales, y distribuidos,

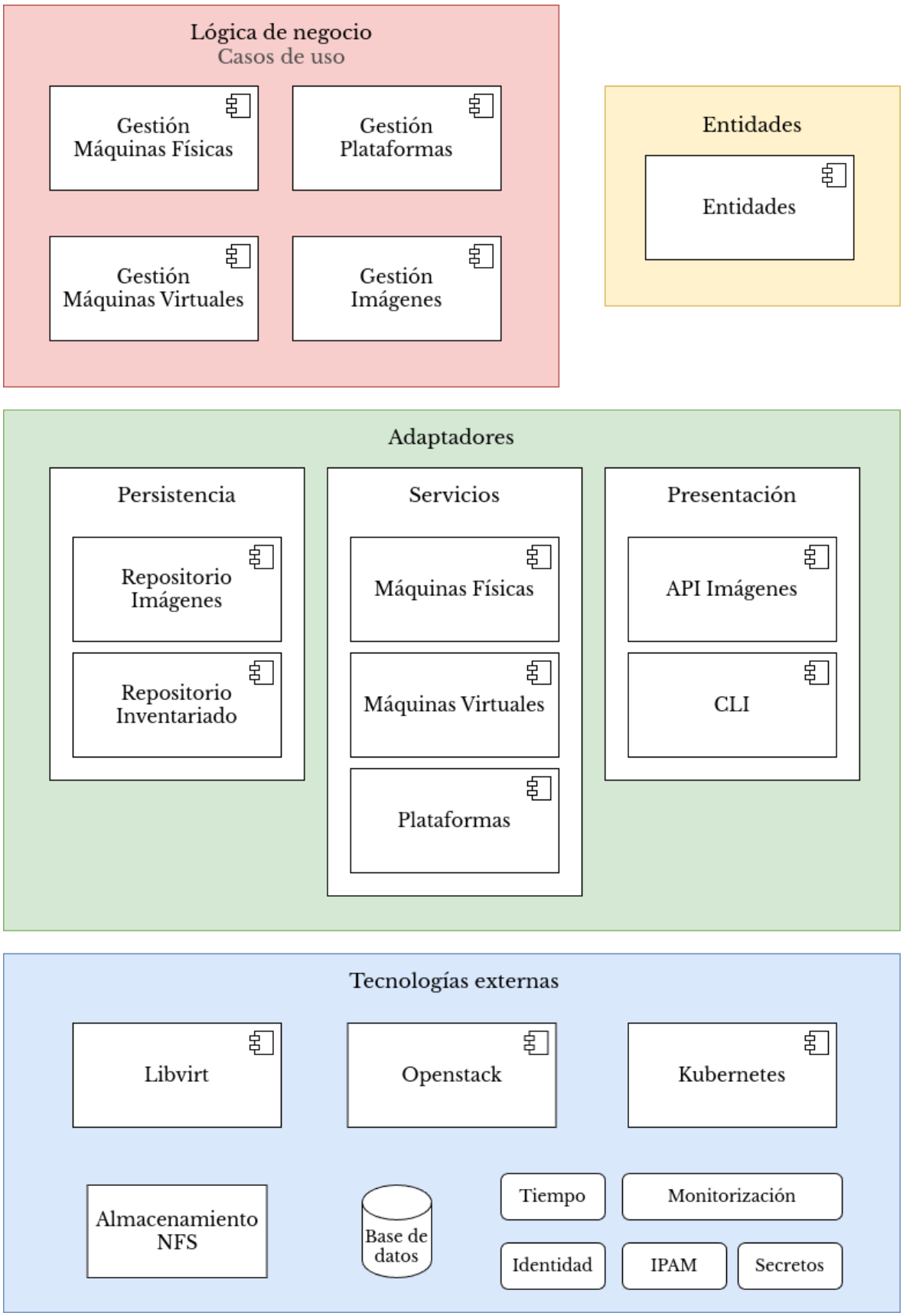

Figura 3.3: Componentes de las capas definidas

como sistemas de ficheros distribuidos o almacenes de objetos.

Red, modelando las distintas redes físicas y virtuales, e **interfaz**, indicando la conexión de una máquina.

Plataforma, refiriéndose a aquellos servicios distribuidos desplegados sobre las máquinas y englobando la especificación de los recursos necesarios para su despliegue.

Imagen, definiendo una imagen de disco, pudiendo incluir detalles del sistema operativo.

3.2.4. Lógica de Negocio

En la lógica de negocio se desarrollan las funcionalidades planteadas en los requisitos.

En primer lugar, se detalla la gestión de máquinas físicas, con la operativa de registro de una máquina física, conllevando su inclusión en el inventariado del sistema junto a su preparación para permitir control remoto y aprovisionamiento.

Por otra parte, se desarrolla la funcionalidad de aprovisionamiento de una máquina, lo cual implica la realización de operaciones necesarias para la instalación de un sistema sobre la máquina física.

Así como la posibilidad de eliminar una máquina física, que supone la eliminación de dicha máquina del inventariado de la aplicación.

En cuanto a la gestión de máquinas virtuales, se introduce la operación de despliegue de una máquina virtual, a partir de la selección de una imagen básica de entre las disponibles. El sistema se encargará de crear una imagen diferencial, configurar la máquina, ponerla en funcionamiento y registrarla en el inventariado.

De la misma forma, se permite la destrucción de una máquina virtual, lo que conlleva la detención, destrucción y eliminación del inventariado de la máquina virtual.

Respecto a la gestión de plataformas, se concibe la funcionalidad de despliegue de una, pudiendo ser Libvirt, Openstack con o sin alta disponibilidad o Kubernetes con o sin alta disponibilidad. Esta operativa comprende el aprovisionamiento de las máquinas implicadas, y la puesta en marcha y configuración de la misma, así como su registro en el inventariado.

En cuanto a la gestión de imágenes, se plantea, por una parte, la operación de añadido de imagen, que conlleva su descarga y el cálculo de su *checksum*. Por otra parte la recuperación de una imagen, que implica la recuperación de la imagen de disco en el formato provisto originalmente. Así como la eliminación de una imagen, que supone su borrado del inventariado. Y finalizando con la modificación de una imagen, que permite el cambio de las propiedades de la imagen.

3.2.5. Adaptadores

Dentro de la capa de adaptadores, se plantea la organización del código en tres módulos, según el objetivo de los sistemas externos con los que interactúan:

Presentación, muestran la información al usuario y controlan la interacción de este. Se plantea la interacción del caso de uso *gestión de imágenes* a través de un API Rest, así mismo, se esboza en otro módulo una CLI para la interacción del administrador.

Persistencia, gestiona el almacenamiento y recuperación de datos. Realiza las conversiones necesarias entre las entidades y los formatos específicos. Se definen dos módulos, uno relacionado con la persistencia

de imágenes, y otro con el inventariado.

Servicios, englobando el resto de interacciones. Se definen tres módulos de servicios, siendo virtualización, gestión de máquinas físicas y despliegue de plataformas.

3.2.6. Interacción entre las capas

A continuación se describe la interacción de los distintos casos de uso definidos en la capa de lógica de negocio, con el resto de módulos del sistema.

En primer lugar, los casos de uso de gestión de máquinas físicas se apoyan sobre el servicio homónimo para realizar su tarea. Además, los casos de uso de registro y borrado de máquinas interactúan con el repositorio de inventariado.

La gestión de máquinas virtuales, por su parte, gira principalmente en torno al servicio de virtualización. Más en detalle, los casos de despliegue y destrucción de máquinas hacen uso del repositorio de inventariado.

A continuación, la de gestión de plataformas se apoya sobre el caso de uso de gestión de máquinas físicas y sobre el servicio de despliegue de plataformas.

Finalmente, el caso de uso de gestión de imágenes hace uso del módulo de persistencia de imágenes, así como recibe la interacción a través del API Rest definido en el módulo de presentación.

Capítulo 4

Implementación

4.1. Tecnologías empleadas

A continuación se desarrollan las tecnologías seleccionadas para la implementación y su justificación.

Lenguajes y tecnologías software para la aplicación

La aplicación se ha implementado mayoritariamente en el lenguaje Go, haciendo uso de scripts en lenguaje Bash para tareas concretas. Se elige Go motivado mayoritariamente por su reducido uso de recursos y la sencillez de la compilación estática, en aras a la reducción de dependencias, simplificando el despliegue de la misma. Así mismo, el extenso soporte y simplicidad del lenguaje se han valorado positivamente.

Por otro lado, para implementar la arquitectura planteada, se hace un uso extensivo del patrón de inyección de dependencias. Si bien es posible realizar esto de forma manual, se ha decidido hacer uso del framework Fx para llevar a cabo esta tarea, puesto que reduce el tamaño y complejidad del código, resultando en código más legible y mantenible.

Así mismo, las conexiones a base de datos se han realizado mediante un

ORM, con el fin de aportar flexibilidad proporcionando independencia del sistema gestor subyacente. La elección tecnológica en este caso es Gorm, por su extenso soporte y facilidad de uso.

Gestión de máquinas y plataformas

En primer lugar, la gestión de máquinas físicas se realiza mediante Bifrost, por su facilidad de uso y flexibilidad operativa. Así mismo, la gestión de máquinas virtuales se realiza mediante libvirt, por su extenso soporte, facilidad de su operativa remota y puesta en marcha. Dicha gestión se realiza mediante el protocolo estándar IPMI.

Por otro lado, la elección tecnológica para el despliegue de las plataformas distribuidas Openstack y Kubernetes no es trivial. No solo la oferta tecnológica que automatiza y gestiona el despliegue de ambos es ciertamente extensa, sino que también se presentan posibilidades de despliegues combinados, desplegando la infraestructura de uno sobre el otro.

En este trabajo, se ha priorizado la facilidades de gestión de las plataformas ofrecidas por las tecnologías, teniendo en cuenta el mantenimiento más allá de su puesta en marcha. Así como se ha descartado el uso de despliegues combinados con el fin de preservar la independencia de las plataformas y mantener la posibilidad de desplegarlas de forma independiente.

Con esto en mente, la tecnología seleccionada para el despliegue y gestión de Openstack es Kolla-Ansible y Kubespray para Kubernetes. Ambas escogidas por las facilidades que ofrecen para las configuraciones específicas de las plataformas, incluyendo configuraciones de alta disponibilidad, así como por la flexibilidad que ofrecen a la hora de determinar las máquinas donde se desplegará.

Gestión de direcciones IP y descubrimiento de servicios

La gestión de direcciones IP se ha realizado mediante NSD como servidor DNS, Unbound como *resolver* e ISC como servidor DHCP. La extensión temporal del trabajo ha obligado a prescindir de la implementación de un IPAM más sofisticado. Las configuraciones planteadas se detallan en el Anexo C.

Gestión de identidad y secretos

En relación a la gestión de identidad, se opta por Authentik como IAM, así como OpenBao como servicio de gestión de secretos. Se detallan en el Anexo F.

Monitorización y observabilidad

Como solución de monitorización y observabilidad, se opta por el stack Prometheus-Grafana, motivado por las facilidades de integración con las plataformas distribuidas planteadas.

En primer lugar, la integración de Kubernetes con Prometheus es posiblemente el caso de uso más común de la herramienta. Así como el despliegue de Openstack mediante Kolla agrega todos los logs de los diversos servicios mediante Fluentd, trivializando su integración con Prometheus. Los detalles de la operativa se muestran en el Anexo E.

Tiempo

Se ha decidido el uso de NTP como protocolo de tiempo, puesto que satisface la precisión requerida por los diversos elementos del sistema. Como servidor NTP concreto se ha elegido Chrony.

4.2. Entidades

Se desarrolla la implementación, de los modelos de entidades definidos en el diseño, en lenguaje Go, siguiendo una estructura que puede verse en el Anexo M.

4.3. Lógica de Negocio

Se han desarrollado los diferentes módulos que implementan las funcionalidades del diseño, en una serie de módulos y paquetes Go, que pueden verse referenciados en el Anexo M.

En cuanto a la gestión de máquinas físicas, englobando registro, aprovisionamiento y supresión, la aplicación se apoya en Bifrost, la cual se invoca con el contexto adecuado, a través de su API, tal y como haya sido escogido por el administrador.

Para la gestión de las máquinas virtuales, se hace uso de la librería *libvirt-go* que permite la interacción con una máquina Libvirt remota a través de su API. De esta forma se realiza la implementación de las operaciones de despliegue y destrucción de máquinas virtuales.

Respecto a las plataformas, se implementa el despliegue de Libvirt como plataforma para usuario, mediante imágenes de disco junto a un script shell que configura de forma adaptada la plataforma.

Por su parte, Openstack es desplegado mediante imágenes de disco junto a la invocación externa de la herramienta Kolla-Ansible, acompañada de la configuración correspondiente, sea en alta disponibilidad o sin ella, e incluyendo el número de máquinas implicadas, diferenciados según su rol e incluyendo número de replicas para tolerancia a fallos.

Igualmente en el caso de Kubernetes, el despliegue es similar, pero

interactuando con la herramienta Kubespray, permitiendo también la configuración en alta disponibilidad, lo cual supone definir máquinas global y número de maestros.

Finalmente toda la operativa de la gestión de imágenes se implementa en Go, junto a llamadas de sistema para manipulación de ficheros. Las imágenes de disco se han preparado manualmente teniendo en consideración las diferentes configuraciones hardware de las máquinas físicas (HP Proliant Microserver Gen8, Dell Poweredge M610 y M620). Finalmente, las imágenes se ofrecen mediante un API Rest, implementado mediante la librería *chi* de Go, puede consultarse en el Anexo L.

4.4. Adaptadores e interacción con Tecnologías Externas

4.4.1. Módulo de servicios

Dentro del módulo de servicios se distinguen los relativos a gestión de máquinas físicas, virtualización y despliegue de plataformas. A continuación se desarrollan los detalles de implementación de los adaptadores, así como las tecnologías externas empleadas.

Gestión de máquinas físicas

Tal y como se ha explicado, la tecnología empleada para esta tarea es Bifrost, cuyo despliegue y configuración han de preceder a la ejecución de la aplicación, cuya operativa se describe en detalle en el Anexo G.

Se plantea la interacción de la aplicación mediante el uso del API Rest ofrecido por Bifrost, para solicitar el registro y aprovisionamiento. El adaptador realizará las conversiones necesarias entre las entidades y los parámetros HTTP pertinentes. Así mismo, las imágenes empleadas para

el aprovisionamiento se obtendrán de forma remota desde el repositorio de imágenes.

Virtualización

Se plantea la necesidad de una máquina capaz de manejar máquinas virtuales para la operativa de este servicio. En el trabajo, se plantea la existencia de una máquina física con libvirt, sobre la que se desplegarán y gestionarán las máquinas virtuales especificadas.

El adaptador se implementa haciendo uso de la librería `libvirt-go`, la cual consiste en un enlazado con la librería de interacción con la plataforma, escrita en C.

Las máquinas virtuales solicitadas a través de este servicio hacen uso de imágenes diferenciales. Para la creación de estas imágenes, se hace uso de la librería *go-qcow*, así como se interactúa con el repositorio de imágenes.

Despliegue de plataformas

Cada una de las plataformas a desplegar presenta una serie de requisitos particulares. Se plantea la posibilidad de desplegar Libvirt, Openstack sin alta disponibilidad, con alta disponibilidad, y Kubernetes sin alta disponibilidad, y con alta disponibilidad.

En primer lugar, el despliegue de Openstack depende de una máquina configurada para ejecutar Kolla-Ansible cuyos operativa se detalle en el Anexo H. De la misma forma, el despliegue de Kubernetes depende de una máquina configurada con Kubespray, y su operativa se detalla en el Anexo I. Finalmente, el despliegue de Libvirt, por su parte, no depende de ninguna máquina auxiliar para su instalación y configuración.

4.4.2. Módulo de persistencia

En este módulo se dispone tanto del repositorio de imágenes, como del repositorio de inventariado, cuya operativa es bastante pareja. Ambos hacen uso del ORM Gorm con el fin de abstraer y desacoplar la interacción con el almacenamiento subyacente.

Cabe destacar que el repositorio de inventariado permite el almacenamiento y recuperación de las distintas entidades, para lo cual se serializan y deserializan mediante `gob`, incluido en la librería estándar de Go.

4.4.3. Módulo de presentación

Este módulo es el que presenta una menor madurez, se plantea su extensión como trabajo futuro. Cabe destacar la presencia de un API Rest implementado para permitir la interacción remota con el caso de uso de gestión de imágenes. La documentación del API se puede consultar en el Anexo L.

Además, se plantea un CLI básico, implementado mediante la librería `cobra`, que ilustraría una posible forma de interacción del administrador con la aplicación.

4.5. Despliegue de la aplicación

Tal y como se ha mencionado el servicio de máquinas físicas depende de una instalación de Bifrost, la cual ha de residir en el mismo segmento físico de red que las máquinas a gestionar, puesto que su operativa se basa en PXE. El Anexo G detalla su operativa. Así como el servicio de virtualización depende de al menos una máquina capaz de virtualizar.

Para el trabajo se plantea una máquina con libvirt permitiendo operativa remota.

Por su parte, el servicio de despliegue de plataformas presenta requisitos específicos propios de cada plataforma a desplegar. Siendo una máquina con Kolla-Ansible para el despliegue de Openstack en cualquiera de sus configuraciones, y una máquina con Kubespray para el despliegue de Kubernetes.

Así mismo, todas las funcionalidades dependen del servicio de inventariado y el repositorio de imágenes. En el trabajo, el repositorio de imágenes se despliega de forma independiente, pero por sencillez el servicio de inventariado permanece ligado a la aplicación.

Se plantea una fase inicial de puesta en marcha, que configure el entorno, satisfaciendo las dependencias de la herramienta, mediante la puesta en marcha de los servicios requeridos y el aprovisionamiento de las máquinas que los albergarán.

Para llevar a cabo esta tarea, se plantea preparar una serie de imágenes de disco preconfiguradas con los servicios descritos. El administrador podrá decidir sobre qué maquinas despliega estas imágenes, pudiendo ser todas sobre la misma, o separarlos según su criterio. Para la inyección inicial, se plantea el uso de una instalación local de Bifrost, residente en una máquina del administrador.

Las figura 4.1 ilustra el proceso de despliegue de las máquinas auxiliares para la operativa de la herramienta. El despliegue de los servicios básicos se describiría con una visualización equivalente.

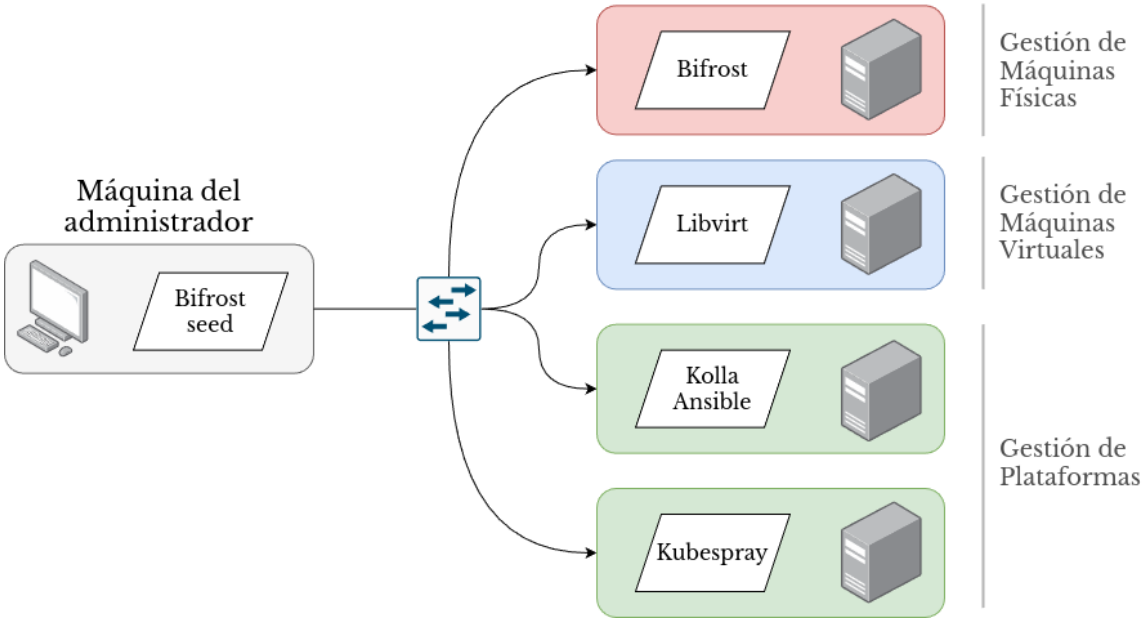

Figura 4.1: Puesta en marcha de plataformas

Capítulo 5

Validación y pruebas

5.1. Metodología

Todas las pruebas han seguido la misma aproximación, descrita a continuación.

Para cada despliegue, se parte de un entorno de pruebas realizado sobre el mayor número posible de elementos virtualizados, con el fin de evitar, en los pasos iniciales, problemas específicos del entorno físico.

Sobre este entorno inicial, se realiza un primer despliegue manual para validar las configuraciones, tras lo cual se van sustituyendo gradualmente elementos virtuales por sus equivalentes físicos.

Para las pruebas en máquinas físicas, se han empleado tres nodos HP con gestión remota mediante iLO o IPMI, un *blade system* Dell con 12 nodos con gestión remota mediante iDRAC o IMPI, así como cuatro torres con configuraciones más humildes. El inventario se detalla en el Anexo K.

La automatización del proceso se va realizando de forma paralela, pudiendo validar las partes automatizadas al ir incorporando elementos físicos.

Despliegues complejos además pueden requerir su división en tareas más sencillas, para su posterior integración, con el fin de simplificar las validaciones.

5.2. Aprovisionamiento de máquinas físicas

Dada su importancia por servir de apoyo para el resto de funcionalidades de la aplicación, el proceso de aprovisionamiento de máquinas físicas fue el primero en ser validado.

Pruebas

Las pruebas realizadas validaban el correcto control remoto de las máquinas, incluyendo control energético, posibilidad de detectar el hardware, eliminar los datos del almacenamiento y aprovisionar la máquina con la imagen deseada, incluyendo la posibilidad de descargar la imagen desde un origen remoto.

Escenario virtual

Para las pruebas iniciales, se parte de un entorno virtualizado en su totalidad. Por un lado, se dispone de una máquina con una instalación de Bifrost, herramienta que permitirá el control remoto y aprovisionamiento de máquinas físicas.

Por otro lado, se dispone de tres máquinas virtuales que simularán ser máquinas físicas. Dado que la operativa de control remoto de máquinas físicas se basa en la interacción con sus BMC, ha sido necesario incluir una versión virtualizada de estos mediante VirtualBMC. Una explicación en detalle de esta tecnología se puede encontrar en el Anexo D.

Primer escenario físico

En la siguiente prueba, se sustituyen las máquinas virtuales por una

máquina física, la cual es, deliberadamente, aquella que presenta la configuración hardware más sencilla de las disponibles. Se trata de una máquina con un sólo disco duro, un interfaz de red administrativo y dos interfaces de red para uso general.

Segundo escenario físico

Tras validar la operativa anterior, el siguiente escenario incluye un conjunto heterogéneo de máquinas físicas, incluyendo aquellas cuya configuración hardware no es trivial, como nodos pertenecientes a un blade system, con múltiples discos e interfaces de red por nodo y asignación de direcciones MAC controlada.

5.3. Despliegue de plataformas distribuidas

5.3.1. Libvirt

Las pruebas de la plataforma básica Libvirt han supuesto el despliegue automatizado de la misma sobre dos máquinas remotas. Para validar la operativa remota se ha probado la definición, configuración, puesta en marcha y destrucción de una máquina virtual. Así mismo, aprovechando el soporte del almacenamiento en red NFS, se ha validado la capacidad de migración entre nodos remotos Libvirt.

5.3.2. Openstack

El despliegue de Openstack, tal y como se ha planteado previamente, se realiza sobre contenedores, haciendo uso de Kolla-Ansible.

Pruebas planteadas

Se plantean tres pruebas, consistiendo la primera en un despliegue

mononodo con los servicios mínimos para funcionar, seguida del mismo conjunto de servicios en configuración multinodo y finalizando con un despliegue multinodo con prestaciones de alta disponibilidad. Tanto en las pruebas con entorno virtual, como con físico, se plantea la existencia de una máquina virtual en la que reside Kolla-Ansible, controlando la operativa.

Desarrollo

Las pruebas iniciales conllevan la creación de tantas máquinas virtuales como nodos se simulen. Kolla-Ansible operara de forma remota sobre estas, poniendo en marcha la infraestructura planteada. Las pruebas en entorno físico se realizaron sobre *blades*.

5.3.3. Kubernetes

Por su parte, el despliegue de Kubernetes se realiza mediante Kubespray, tal y como se ha mencionado previamente.

Pruebas planteadas

Se plantean dos pruebas, la primera consistiendo en un despliegue con un único maestro y la segunda contemplando una configuración de maestro tolerante a fallos.

Desarrollo

El desarrollo de las pruebas es casi idéntico al de Openstack, se parte de la configuración mínima sobre máquinas virtuales y se finaliza con el despliegue sobre *blades*.

Capítulo 6

Conclusiones

La aplicación desarrollada durante el trabajo ha logrado satisfacer los objetivos fijados, habiéndose logrado el diseño e implementación de una aplicación capaz de modelar los diversos elementos que componen un cloud privado, entendiéndose como tales a las diversas máquinas físicas, virtuales, contenedores, plataformas distribuidas, elementos de red, sistemas de almacenamiento y servicios auxiliares.

Para la realización de esta tarea, ardua por la complejidad de los conceptos y tecnologías implicadas, se ha requerido un análisis exhaustivo del estado del arte, con el fin de determinar los elementos adecuados para este trabajo. Así mismo, se han determinado y puesto en marcha los servicios de apoyo necesarios para el correcto funcionamiento del sistema, englobando servicios de tiempo, identidad, secretos, gestión de direcciones IP, descubrimiento de servicios, nombres y monitorización.

6.1. Experiencia personal

Este trabajo me ha permitido explorar y profundizar en aspectos fundamentales presentes en cualquier cloud. Si bien la mayoría han sido

mencionados durante mi formación en el grado, debido a su amplitud, la mayoría se abordan de una forma bastante superficial.

Considero que este trabajo ha nutrido considerablemente mi entendimiento de la materia, no solo por brindarme la oportunidad de profundizar en ciertas tecnologías, sino por poder interactuar directamente con todos los elementos de un sistema real y ver de cerca la cohesión que presentan entre sí. Así mismo, el haber podido realizar pruebas de gestión de máquinas físicas con hardware real ha sido una experiencia realmente enriquecedora.

Finalmente destacar que el desarrollo del trabajo me ha permitido poner en práctica diversas técnicas y metodologías que he aprendido durante el grado, las cuales han facilitado de forma notable todo el proceso.

6.2. Trabajo futuro

Se plantean, como posibles mejoras a este trabajo, pudiendo constituir otros Trabajos de Fin de Grado o Máster:

- El diseño y desarrollo de una herramienta CLI o gráfica para facilitar la interacción del administrador con el sistema.
- La extensión del sistema para poder realizar despliegues sobre cloud público, pudiendo extender la operativa a modelos de cloud híbrido.
- La implementación de sistemas de almacenamiento distribuidos más sofisticados, como Ceph
- La profundización en aspectos de seguridad del sistema.
- El diseño e implementación de un conjunto extenso de pruebas que garanticen el correcto funcionamiento del sistema, englobando no sólo pruebas unitarias, de integración o extremo a extremo, sino también *chaos engineering* o *fuzzing*.
- La extensión de las capacidades de control energético, enfocado en *green computing*.

Bibliografía

[1] P. Hodgetts Isarría and U. Arronategui Arribalzaga, "Diseño, Despliegue y Monitorización de un Simulador Distribuido de Eventos Discretos." *Trabajo Fin de Grado, Universidad de Zaragoza*, 2022.

[2] IBM, "¿Qué es la observabilidad?" Dec. 2024. [Online]. Available: https://www.ibm.com/es-es/topics/observability

[3] J. Camp, "Digital identity," *IEEE Technology and Society Magazine*, vol. 23, no. 3, pp. 34–41, 2004.

[4] Openstack Contributors, "Containers Whitepaper: Leveraging Containers and Openstack - OpenStack Open Source Cloud Computing Software." [Online]. Available: https://www.openstack.org/use-cases/containers/leveraging-containers-and-openstack/

[5] Y. Brikman, *Terraform: Up and Running, 3rd Edition.* O'Reilly, 2022, iSBN: 9781098116743. [Online]. Available: https://www.oreilly.com/library/view/terraform-up-and/9781098116736/

[6] S. Wågbrant and V. Dahlén Radic, *Automated Network Configuration : A Comparison Between Ansible, Puppet, and SaltStack for Network Configuration*, 2022. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:mdh:diva-58886

[7] Canonical, "Canonical Juju | Why Juju?" [Online]. Available: https:

//juju.is/why-juju

[8] R. C. Martin, “Clean Coder Blog.” [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[9] FreeIPMI. What is IPMI? [Online]. Available: https://www.gnu.org/software/freeipmi/freeipmi-faq.html#What-is-IPMI_003f

[10] Prometheus Contributors, “prometheus/node_exporter,” Jan. 2025, original-date: 2013-04-18T14:44:52Z. [Online]. Available: https://github.com/prometheus/node_exporter

[11] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, p. 612–613, Nov. 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[12] Bifrost Contributors, “Bifrost.” [Online]. Available: https://docs.openstack.org/bifrost/2024.2/index.html

[13] Openstack Contributors, “Kolla ansible docs | Neutron - Networking Service.” [Online]. Available: https://docs.openstack.org/kolla-ansible/2024.2/reference/networking/neutron.html

[14] Kubespray Contributors, “Kubespray docs | operations/ha-mode.” [Online]. Available: https://github.com/kubernetes-sigs/kubespray/blob/master/docs/operations/ha-mode.md

Lista de Figuras

Lista de Tablas

Anexos

Anexo A

Planificación

A.1. Resumen de esfuerzos dedicados

Tarea	Horas
Estudio del estado del arte	55h
Diseño	42h
Comprensión, exploración y prototipado de los despliegues	51h
Desarrollo de la aplicación	117h
Validación y pruebas de la operativa	56h
Reuniones	37h
Memoria	48h
Total	**406h**

Tabla A.1: Resumen de horas dedicadas

Anexo B

Arquitectura Clean

B.1. Contexto

La arquitectura Clean es un enfoque de diseño de software propuesto por Robert C. Martin (Uncle Bob) que busca crear sistemas mantenibles, escalables y legibles. La aplicación desarrollada en este trabajo se acoge a esta arquitectura.

B.2. Descripción de la arquitectura

Esta arquitectura define una serie de capas, de tal forma que aquellas internas encapsulan las entidades y reglas de negocio y no han de depender de sistemas externos. Por contra, las capas externas son aquellas que se relacionan con sistemas externos y pueden ser dependientes de las tecnologías empleadas.

En este contexto, los cambios que se realicen en las capas exteriores no deben afectar a las interiores. Este desacoplamiento puede lograrse mediante la *regla de dependencia* y el *principio de inversión de dependencias*.

La regla de dependencia consiste en que sólo las capas externas pueden

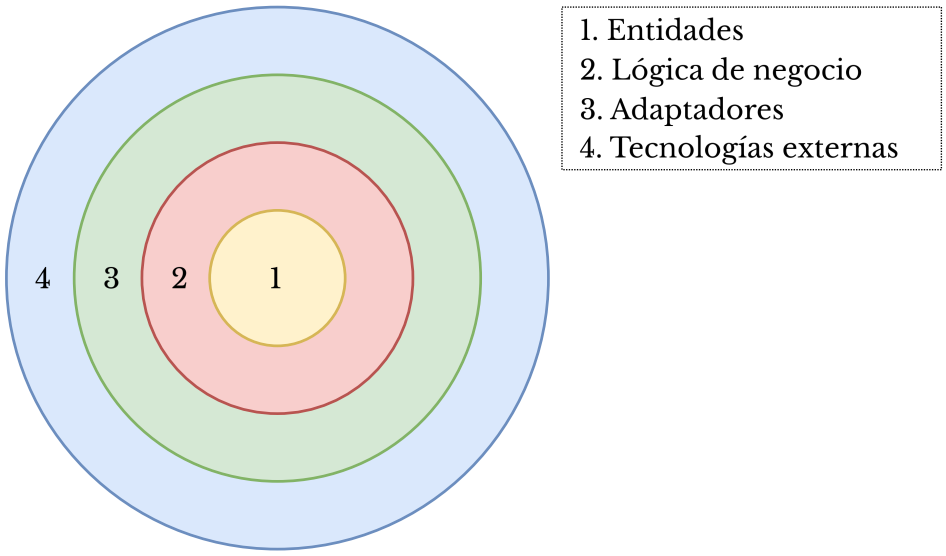

Figura B.1: Capas definidas

referenciar directamente a las internas, pero nunca al revés. Para lograr que una capa interna interactúe con una externa, se hace uso del principio de inversión de dependencias, por el que la capa interna define un interfaz que será implementado en las capas externas.

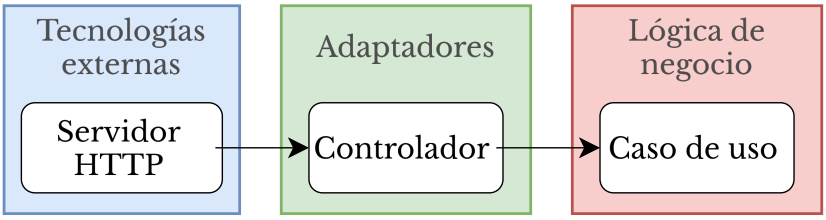

Figura B.2: Ejemplo de la *regla de dependencia*

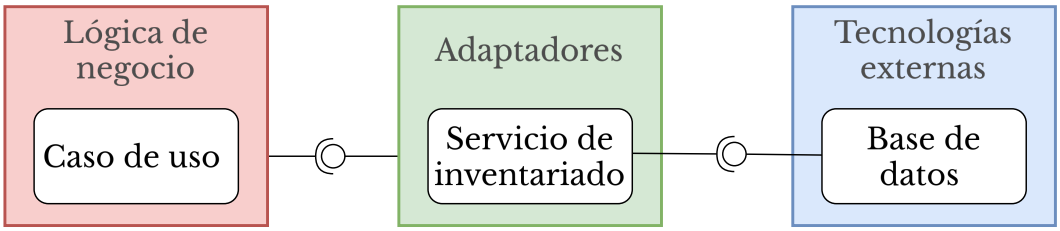

Figura B.3: Ejemplo del *principio de inversión de dependencias*

Anexo C

Gestión de direcciones IP y descubrimiento de servicios

C.1. Contexto

Este anexo desarrolla la arquitectura y configuración de los distintos servicios empleados para la asignación de nombres y direcciones IP en el entorno de la aplicación.

C.2. Diseño

Para la gestión de direcciones IP, se plantea únicamente el uso de un servidor DNS y un servidor DHCP. Estos residen dentro de la red local empleada y gestionan las direcciones de todas las máquinas físicas empleadas, así como de algunas máquinas virtuales de especial interés, como las que contienen servicios de tiempo, nombres u otras herramientas de apoyo como Kubespray o Kolla-Ansible.

C.3. Implementación

Las tecnologías escogidas han sido NSD junto a Unbound como servidor DNS y ISC-DHCP como servidor DHCP.

La configuración del servidor DNS se muestra en los siguientes fragmentos: el bloque de código C.1 muestra la inclusión de las zonas directa e inversa en `nsd.conf`, el bloque de código C.2 muestra un fragmento de la configuración de la zona directa y el bloque de código C.3, de la inversa. Así mismo, la configuración del DHCP se muestra en el bloque de código C.4

```
# Omitido por claridad

zone:
        name: "zygote.lan"
        zonefile: "zygote.zone"
zone:
        name: "6.0.10.in-addr.arpa."
        zonefile: "6.0.10.in-addr.arpa.zone"
```

Código C.1: Configuración NSD

```
$ORIGIN zygote.lan.     ; default zone domain
$TTL 86400              ; defalt time to live

@ IN SOA ns1.zygote.es. hostmaster.zygote.es. (
           2025011502  ; serial number
           28800       ; Refresh
           14400       ; Retry
           864000      ; Expire
           86400       ; Min TTL
           )

           NS      ns1.zygote.es.

; Blades
blade-620-1 IN        A   10.0.6.56
blade-620-2 IN        A   10.0.6.63
blade-620-3 IN        A   10.0.6.57
; Omitido por claridad

; Cubos
cubo-0      IN  A   10.0.6.75
cubo-1      IN  A   10.0.6.76
cubo-2      IN  A   10.0.6.77

; Servicios
ns1         IN  A   10.0.6.15
time        IN  A   10.0.6.15
bifrost     IN  A   10.0.6.22
kolla       IN  A   10.0.6.24
kubespray   IN  A   10.0.6.180
```

Código C.2: Definición zona directa

```
$ORIGIN 6.0.10.in-addr.arpa.    ; default zone domain
$TTL 86400              ; default time to live

@ IN SOA ns1.zygote.lan. hostmaster.zygote.lan. (
           2025011200  ; serial number
           28800       ; Refesh
           14400       ; Retry
           864000      ; Expire
           86400       ; Min TTL
          )

                  NS      ns1.zygote.lan.

; Blades
56                PTR        blade-620-1.zygote.lan.
63                PTR        blade-620-2.zygote.lan.
57                PTR        blade-620-3.zygote.lan.
; Omitido por claridad

; Cubos
75                PTR        cubo-0.zygote.lan.
76                PTR        cubo-1.zygote.lan.
77                PTR        cubo-2.zygote.lan.

; Servicios
15                PTR        ns1.zygote.lan.
15                PTR        time.zygote.lan.
22                PTR        bifrost.zygote.lan.
24                PTR        kolla.zygote.lan.
180                PTR        kubespray.zygote.lan.
```

Código C.3: Definición zona inversa

```
option routers 10.0.6.254;

subnet 10.0.6.0 netmask 255.255.255.0 {
  option routers 10.0.6.254;
}

host ns1 {
  hardware ethernet 52:54:00:98:d6:1d;
  fixed-address 10.0.6.15;
}

host cubo-0 {
  hardware ethernet 00:FD:45:FC:44:CC;
  fixed-address 10.0.6.175;
}

# ...

host blade-620-1 {
  hardware ethernet 24:B6:FD:6B:89:01;
  fixed-address 10.0.6.56;
}

host blade-620-2 {
  hardware ethernet 24:B6:FD:6B:89:0E;
  fixed-address 10.0.6.63;
}

host blade-620-3 {
  hardware ethernet 24:B6:FD:6B:89:1B;
  fixed-address 10.0.6.57;
}

# ...
```

Código C.4: Configuración DHCP

Anexo D

Soporte virtualizado de IPMI

D.1. Contexto

La especificación IPMI (Intelligent Platform Management Interface) define una abstracción estandarizada que permite hacer uso de las funcionalidades de monitorización y control ofrecidas por una plataforma hardware. [9]

Estas funcionalidades son generalmente provistas por un microcontrolador empotrado, denominado BMC (Baseboard Management Controller).

La virtualización de estos sistemas puede resultar de interés con el fin de componer entornos de prueba para sistemas de gestión de hardware, como Bifrost, o como herramienta de docencia.

D.2. VirtualBMC

La herramienta VirtualBMC permite la virtualización de BMC asignados a dominios libvirt. Una vez definido, el BMC virtual permanece a la escucha de peticiones IPMI y realiza las operaciones correspondientes

sobre el dominio asignado.

La herramienta se compone de un demonio (`vbmcd`) y un cliente (`vbmc`), que han de ejecutarse sobre la misma máquina, lo que podría suponerse como una limitación severa, pero la posibilidad de definir dominios libvirt remotos aporta suficiente flexibilidad para una operativa normal.

El bloque de código D.1 muestra un ejemplo de uso. Se parte de un dominio libvirt denominado `test-vm` previamente definido.

```
# Definicion del BMC virtual
vbmc add test-vm --port 6230 --username vbmcadmin --password
↪  adminpassword

# Puesta en marcha
vbmc start test-vm

# Verificacion de la operativa
ipmitool -I lanplus -H 127.0.0.1 -p 6230 -U vbmcadmin -P adminpassword
↪  power status
```

Código D.1: Ejemplo VirtualBMC

Anexo E

Sistemas de monitorización y observabilidad

E.1. Contexto

Este anexo describe los sistemas de monitorización y observabilidad implantados. Se ha optado por el stack Prometheus-Grafana motivado por la facilidad de integración con las plataformas distribuidas desplegadas mediante la herramienta (Kubernetes y Openstack).

E.2. Arquitectura

Se plantea la monitorización de todos los elementos que componen el entorno, desde las máquinas físicas, hasta las plataformas desplegadas. La recolección de las métricas se realiza mediante Prometheus, para su posterior consulta y visualización desde Grafana.

En el trabajo, limitado por la extensión temporal, se ha realizado una implementación parcial, que ilustra las posibilidades del sistema y es fácilmente extensible a una implementación completa.

E.3. Implementación

La implementación realizada consiste en la monitorización de las máquinas, sin incluir las plataformas desplegadas. Para llevar a cabo esta tarea, se hace uso de Prometheus Node Exporter para la recolección de métricas del sistema. [10]

La configuración del Exporter es sencilla, consiste en la descarga de un binario y su configuración como servicio. El bloque de código E.1 muestra la configuración como servicio en un sistema con SystemD.

Dichas métricas son recolectadas por un servidor de Prometheus, el cual ha de tener registradas las máquinas a las que interrogar, tal y como se muestra en el bloque de código E.2.

Finalmente, para lograr la integración con Grafana, tras su puesta en marcha, se ha de registrar una fuente de datos Prometheus, con la dirección del servidor. Un ejemplo de visualización con Grafana se muestra en la Figura E.1.

```
[Unit]
Description=Node exporter
After=network.target

[Service]
User=root
Group=root
Type=simple
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

Código E.1: Servicio para Prometheus Node Exporter

```
scrape_interval: 15s

scrape_configs:
- job_name: node-blade
  static_configs:
  - targets: [
    'blade-620-1.zygote.lan:9100',
    'blade-620-2.zygote.lan:9100',
    'blade-620-3.zygote.lan:9100',
    'blade-610-1.zygote.lan:9100',
    'blade-610-2.zygote.lan:9100',
    'blade-610-3.zygote.lan:9100',
    #...
]
```

Código E.2: Configuración servidor Prometheus

Figura E.1: Visualización con Grafana

Anexo F

Sistemas de identidad y secretos

F.1. Contexto

Se plantea el uso de Authentik como solución IAM, así como de OpenBao como gestor de secretos. Estas tecnologías son fácilmente integrables con las plataformas distribuidas desplegables por la herramienta, facilitando la unificación de identidades y secretos entre ellas.

F.2. Implementación

F.2.1. Servicio de identidad

El despliegue de Authentik propuesto por la documentación oficial se realiza mediante Docker Compose, proporcionando una instalación bastante directa. Las configuraciones deseadas se han de realizar mediante variables de entorno, de forma previa al despliegue.

Así mismo, la instalación por defecto requiere cierta interacción, pero es posible automatizar la operativa proporcionando las variables de entorno adecuadas.

F.2.2. Servicio de secretos

La puesta en marcha de OpenBao requiere ciertas consideraciones. Por defecto, el sistema se encuentra en un estado ”sellado” (*sealed*), en el que todo sus datos se encuentran cifrados. Para permitir la operativa normal de la herramienta, se ha de descrifrar, transicionando a un estado ”abierto” (*unsealed*).

Durante el proceso de instalación, se genera una clave de cifrado, empleada para el cifrado y descifrado de todos los datos. Dada la importancia de esta clave, se cifra mediante otra clave, denominada raíz, cuya distribución se plantea mediante *Shamir's Secret Sharing*. [11]

Por tanto, se ha de prestar atención a las claves generadas durante el proceso de instalación de la herramienta. El proceso de apertura se ilustra en el bloque de código F.1. Por defecto, la instalación genera cinco fragmentos, siendo al menos tres los necesarios para lograr la apertura.

```
# Provee un fragmento de clave
bao operator unseal
# output:
# Key (will be hidden):
# Sealed: false
# Key Shares: 1
# Key Threshold: 3
# Unseal Progress: 0

# Tras lograr unseal
bao login
```

Código F.1: Proceso de apertura

F.2.3. Integración de ambos

La integración de OpenBao con Authentik se realiza mediante OpenID Connect, cuya configuración se describe a continuación. En primer lugar,

se ha de definir en Authentik un proveedor OAuth2 / OIDC, así como una aplicación asociada a este. La configuración del proveedor ha de incluir las direcciones que se permiten como *callback*, para permitir la operativa de OIDC.

Una vez realizado esto, se ha de configurar OpenBao tal y como se muestra en el bloque de código F.2, haciendo uso del *client id* y *client secret* generados por Authentik.

```
bao auth enable oidc
bao write auth/oidc/config\
    oidc_discovery_url=\
      "http://iam.zygote.lan:9000/application/o/openbao-slug/"\
    oidc_client_id="$CLIENT_ID"\
    oidc_client_secret="$CLIENT_SECRET"\
    default_role="reader"


bao write auth/oidc/role/reader\
    bound_audiences="$CLIENT_ID"\
    allowed_redirect_uris=\
      "https://secrets.zygote.lan:8200/oidc/callback"\
    allowed_redirect_uris=\
      "http://secrets.zygote.lan:8250/oidc/callback"\
    allowed_redirect_uris=\
      "http://localhost:8250/oidc/callback"\
    user_claim="sub"\
    policies="reader"
```

Código F.2: Configuración de OpenBao para OIDC

Anexo G

Gestión de nodos físicos: Bifrost

G.1. Contexto

Dentro del ecosistema de Openstack se encuentra Ironic, servicio encargado de la gestión de nodos físicos, permitiendo su inspección, configuración y aprovisionamiento, mediante protocolos estándar como PXE e IPMI, así como con otros protocolos propietarios como iDRAC (DELL) o iLO (HP).

El proyecto Bifrost busca independizar esta operativa, con el fin de poder gestionar máquinas físicas sin necesidad de disponer de un despliegue Openstack completo. Además, este proyecto ofrece las herramientas para obtener una instalación automatizada. [12]

G.2. Descripción de la operativa

Para ilustrar la operativa de Bifrost, a continuación se describe un ejemplo de uso.

El proceso comienza registrando un nodo físico, añadiendo como mínimo el protocolo de control remoto a emplear, como por ejemplo

IPMI, y los detalles para establecer la conexión.

G.2.1. Inspección

A continuación, Bifrost es capaz de detectar de forma automática detalles sobre el hardware, para lo cual ofrece distintas posibilidades. En primer lugar, la opción menos exhaustiva, denominada *out of band*, consiste en obtener detalles interrogando a la máquina con el protocolo de control especificado.

Una detección más intensiva requiere el uso de un agente, la cual se denomina *in band* y su operativa se puede clasificar en gestionada (*managed*) o no gestionada (*unmanaged*). En la primera, el proceso completo de selección de dispositivo de arranque, incluyendo la selección del dispositivo y encendido, es controlado en remoto por Bifrost, mientras que en la segunda se ha de configurar el arranque por red de forma manual.

El agente se provee a la máquina mediante PXE, se carga en RAM y realiza pruebas específicas para detectar las características del hardware.

G.2.2. Gestión

El agente provee además funcionalidades de gestión de la máquina, permitiendo gestión del almacenamiento, aprovisionamiento y monitorización.

Si se solicita, el agente puede eliminar el contenido de los discos antes de continuar con el proceso, así como crear y configurar volúmenes. Así mismo, permite aprovisionar el nodo, solicitándolo desde Bifrost.

El proceso de aprovisionamiento consiste en la descarga de la imagen de

disco solicitada desde Bifrost, para su posterior volcado en la máquina. La imagen puede residir en la máquina con la instalación de Bifrost, o residir en un servidor remoto.

El proceso de aprovisionamiento no implica el fin del control, es posible volver a cargar el agente cuando se desee y repetir los procesos deseados.

G.3. Ejemplo de uso

El bloque de código G.1 muestra el registro de un nodo controlado vía IPMI, al que se le asocia una dirección MAC, con el fin de permitir su reconocimiento por el servidor DHCP. Así mismo, provee el agente para permitir su gestión.

```
baremetal node create
--name sample \
--driver ipmi \
--driver-info ipmi_username=$IPMIUSER \
--driver-info ipmi_password=$PASSWORD \
--driver-info ipmi_address=10.0.6.50 \
--property capabilities=boot_mode:bios

# Valida la configuración y conectividad
baremetal node validate sample

# Provee el agente
baremetal node manage
```

Código G.1: Registro de un nodo

A continuación, el bloque de código G.2 muestra la configuración de la imagen que se empleará para aprovisionar la máquina. En el ejemplo se emplea una imagen de disco completo, lo que se aconseja en la documentación de Bifrost. En este caso, la imagen reside en la máquina que el servicio, tal y como denota la directiva `file`.

Finalmente, se realiza el aprovisionamiento. En este paso, se permite incluir un parámetro con configuraciones específicas para la máquina.

```
baremetal node set --instance-info
↪  image_source=file:///opt/images/debian12.qcow2 --instance-info
↪  image_checksum=0700c61846af7df944eef4beafe28f31 sample
```

Código G.2: Configuración para el aprovisionamiento

Cabe destacar que se permite proveer una configuración *cloud-config*, en el caso de que la imagen seleccionada haga uso de *cloud-config*.

```
baremetal node deploy cubo-0 --config-drive \
'{"meta_data": {"hostname": "sample", "networks": [{ "id": "enp1s0",
↪  "link": "enp1s0", "network_id":
↪  "b2dbeb87-6842-410c-baed-aae61ba53a38", "type": "ipv4_dhcp"}],
↪  "services": [{ "address": "155.210.3.12", "type": "dns" }] }}'
```

Código G.3: Aprovisionamiento

Anexo H

Despliegue de Openstack sobre contenedores

H.1. Contexto

El proyecto Kolla logra desplegar los distintos componentes de Openstack sobre contenedores. Así mismo, el subproyecto Kolla-Ansible provee las herramientas para lograr un despliegue autónomo y fácilmente configurable.

Esta herramienta permite especificar de forma individual los servicios de Openstack presentes en el despliegue, así como ofrece diversas funcionalidades para dotarlos de alta disponibilidad y tolerancia a fallos.

Por otro lado, presenta gran flexibilidad para determinar qué servicios se ubicarán en qué nodos, mediante un inventario de Ansible. Incluso contempla la posibilidad de desplegar todos los servicios sobre un mismo nodo, lo que denominan *all-in-one*, lo que resulta de interés para aproximaciones iniciales, pruebas y docencia.

H.2. Configuración

Dado que la herramienta se basa en Ansible para su funcionamiento, se ha de proveer un inventario que especifique las máquinas sobre las que se operará. El bloque de código H.1 muestra un inventario para un despliegue mononodo, mientras que el bloque de código H.2 muestra una configuración multinodo sencilla, con un nodo de control y dos de cómputo.

Así mismo, se ha de proveer un fichero de configuración con parámetros específicos de Kolla, que determinarán los servicios y otras funcionalidades presentes en el despliegue Openstack resultante. Estos parámetros serán interpretados como variables de Ansible, con las consideraciones que conlleva. El bloque de código H.4 muestra un ejemplo de configuración mínima. Para más detalle se recomienda consultar detalladamente el fichero de ejemplo, así como la documentación relativa a Neutron (servicio de red) puede resultar de interés. [13]

```
[control]
localhost       ansible_connection=local

[network]
localhost       ansible_connection=local

[compute]
localhost       ansible_connection=local

[monitoring]
localhost       ansible_connection=local

[storage]
localhost       ansible_connection=local

# Omitido por claridad
```

Código H.1: Inventario para despliegue mononodo

```
[control]
blade-620-1

[network]
blade-620-1
blade-620-2
blade-620-3

[compute]
blade-620-2
blade-620-3

# Omitido por claridad
```

Código H.2: Inventario para despliegue mononodo

```
# IP virtual que será empleada por Keepalived
kolla_internal_vip_address: "10.0.6.80"

# Interfaz sobre la que se enlazarán las API de los servicios
network_interface: "eno1"

# Interfaz (o interfaces) consideradas como conectadas a la red
# externa.
neutron_external_interface: "eno2"

# Esta opción puede omitirse, pero se incluye por consistencia
kolla_base_distro: "debian"
```

Código H.3: Configuración mínima

H.3. Puesta en marcha

Para operar con Kolla-Ansible, se ha de hacer uso de su CLI, la cual gestiona las dependencias necesarias para la ejecución de los scripts de Ansbile, así como se encarga de ejecutarlos en el orden correcto y con los parámetros adecuados. El bloque de código muestra un ejemplo de despliegue.

```
# Instalación de dependencias
kolla-ansible install-deps

# Generación en /etc/kolla/passwords.yml las contraseñas
# empleadas para los servicios de Openstack
kolla-genpwd

# Preparación del despliegue
kolla-ansible bootstrap-servers -i ./multinode
kolla-ansible prechecks -i ./multinode

# Despliegue
kolla-ansible deploy -i ./multinode
```

Código H.4: Puesta en marcha y despliegue

Anexo I

Despliegue de Kubernetes mediante Kubespray

I.1. Contexto

El proyecto Kubespray consiste en una herramienta de automatización de despliegues de Kubernetes, sobre infraestructura de cloud público o privado. Se operativa se basa en Ansible. Cabe destacar que permite una configuración bastante extensa, permitiendo incluir características de alta disponibilidad. Así mismo, permite el despliegue de toda la operativa sobre un único nodo, lo cual puede ser de interés para docencia.

I.2. Configuración

Puesto que su operativa se basa en Ansible, se ha de proveer un inventario con las máquinas sobre las que se realizará el despliegue. El bloque de código I.1 muestra una configuración mínima y el bloque de código I.2 una para alta disponibilidad.

La configuración de las distintas funcionalidades, como pueden ser el CNI (Container Network Interface) empleado o las configuraciones de

alta disponibilidad, se realizan mediante variables de Ansible.

Se entiende por una configuración de alta disponibilidad a aquella que ofrece un clúster `etcd` tolerante a fallos y una forma de acceso resiliente al API de los maestros.

Kubespray facilita lograr esta configuración. En primer lugar, despliegues `etcd` son tolerantes a fallos por defecto. Por otro lado, en relación al acceso al API, se ofrece la posibilidad de hacer *localhost loadbalancing*, así como desplegar un balanceador de carga con el rol `kube-vip`, o incluso el emplear un balanceador externo. [14]

En el trabajo, con el fin de simplificar la operativa, se ha empleado *localhost loadbalancing*, el cual consiste en incluir un proxy Nginx en todos los nodos trabajadores, configurados con las direcciones de todos los nodos maestro. La figura I.1

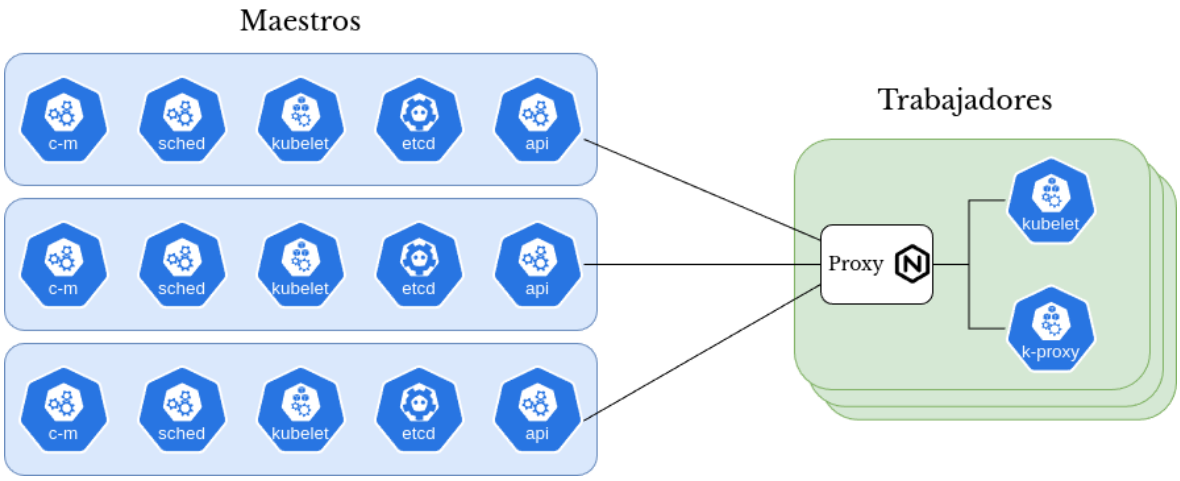

Figura I.1: Configuración en alta disponibilidad

```
[kube_control_plane]
kube-master-0.zygote.lan

[etcd:children]
kube_control_plane

[kube_node]
kube-worker-0.zygote.lan
```

Código I.1: Inventario mínimo

```
[kube_control_plane]
kube-master-0.zygote.lan
kube-master-1.zygote.lan
kube-master-2.zygote.lan

[etcd:children]
kube_control_plane

[kube_node]
kube-worker-0.zygote.lan
kube-worker-1.zygote.lan
kube-worker-2.zygote.lan
kube-worker-3.zygote.lan
kube-worker-4.zygote.lan
```

Código I.2: Inventario para alta disponibilidad

Anexo J

Configuración de instancias

J.1. Contexto

Se presenta una dualidad a la hora de realizar las configuraciones de las distintas máquinas. Por un lado, es posible la preparación de imágenes de disco específicas para cada una de las máquinas, de forma anticipada al despliegue. Y por el contrario, es posible el despliegue de una imagen genérica para su posterior configuración en la máquina destino.

Así mismo, ambos procesos pueden realizarse con scripts en su totalidad, o con herramientas de automatización, incluyendo Packer, Ansible o Puppet.

La decisión de la metodología a seguir no es trivial y, en la mayoría de los casos, resulta en un punto intermedio de ambas.

J.2. Metodologías

J.2.1. Anticipación al despliegue

En esta contexto, se plantea la creación de imágenes de disco que incluyan todo el software y configuraciones necesario para cada una de las máquinas, de forma anticipada al despliegue final.

Es común que este proceso se realice sobre máquinas virtuales, que serán configuradas manualmente o utilizando alguna automatización, tras lo cual, su imagen de disco podrá ser desplegada sobre la máquina destino.

J.2.2. Configuración completa en destino

En este caso, se plantea el despliegue de imágenes base, sin configurar o con configuraciones mínimas y comunes a todos los nodos, para su posterior configuración de forma remota, mediante la ejecución de scripts remotos de forma manual, o automatizada con herramientas como Ansible.

J.2.3. Configuración específica en destino

En la mayoría de los escenarios, es imposible preparar las imágenes completamente personalizadas para cada una de las máquinas de forma anticipada. Es por lo que se plantea una metodología intermedia, en la que las imágenes preparadas contienen todo el software y configuraciones comunes al mayor número de máquinas posible, así como herramientas como `cloud-init` que permitirán un ajuste específico para cada máquina, que será aplicado en destino.

Anexo K

Inventario de máquinas físicas

K.1. Máquinas utilizadas

Este anexo enumera y detalla las máquinas físicas que han sido utilizado durante la elaboración del trabajo, consistiendo en:

3 Servidores HP Proliant Microserver Gen 8

Los cuales cuentan con un procesador de 4 núcleos, 4GB de memoria principal y 500GB de almacenamiento. Presentan dos interfaces de red de propósito general y una tercera dedicada para su gestión remota, mediante iLO, la solución del fabricante.

1 Blade system

Se cuenta con un *blade system* Dell Poweredge M1000e, con 12 nodos, siendo 3 Poweredge M620 y 9 Poweredge M610. Los primeros cuentan con dos procesadores de 6 núcleos cada uno y los segundos con dos procesadores de 4 núcleos. Todos cuentan además con 192 GB de memoria principal.

Respecto al almacenamiento, los M620 cuentan con 2 SSD, uno de 2TB y otro de 256GB, mientras que los M610 cuentan cada uno con uno de

1TB y otro de 256GB.

El *blade system* contiene dos switches Dell con puertos GbE, así como otros dos switches con conexiones 10GbE. Cada nodo dispone de 4 puertos, cada uno conectado a un switch y la configuración MAC se preconfigura en el chasis, ligándose a la bahía en lugar de la tarjeta de red del nodo, facilitando el remplazo o intercambio de los mismos, en relación a la configuración de red del sistema.

Otras máquinas

- Una torre con 8 núcleos, 128GB de memoria principal, 1TB de almacenamiento, un interfaz de red GbE y dos 10GbE

- Una torre con 4 núcleos, 32GB de memoria principal, 500GB de almacenamiento y dos interfaces de red GbE.

- Dos torres con 4 núcleos, 24GB de memoria principal, 500GB de almacenamiento y dos interfaces GbE.

Anexo L

API Rest de Imágenes

L.1. Contexto

Se plantea un API Rest para permitir la interacción remota con el caso de uso de repositorio de imágenes, el cual permite la creación, recuperación, modificación y borrado de imágenes de disco.

L.2. Uso

Las siguientes páginas ilustran el funcionamiento de los endpoints implementados del API.

get **/image**

Obtiene el listado de las imagenes disponibles

Parameter

no parameter

Response application/json

200 ok


```
[{
  "name":"blade-base",
  "os": {
    "architecture":"amd64",
    "checksum":"2abc576b8dd68ad12a02950726fa2fca",
    "distribution":"Debian",
    "family":"Linux",
    "format":"qcow2",
    "release":"bookworm"
  },
  "source-uri":"file:///opt/imagerepo/blades5.qcow2",
  "uri":"/image/blade-base"
},
{
  "name":"libvirt-base",
  "os": {
    "architecture":"amd64",
    "checksum":"ecf2b332c09b7548c67972826c18ead3",
    "distribution":"Debian",
    "family":"Linux",
    "format":"qcow2",
    "release":"bookworm"
  },
  "source-uri":"http://example.com/libvirt.qcow2",
  "uri":"/image/libvirt-base"
}]
```


500 La lista de imágenes no se ha podido recuperar

get **/image/{name}**
Obtiene una imagen

Parameter

name Nombre de la imagen

Response application/x-qemu-disk

200 ok

```
Binario con la imagen
```

404 Not found: la imagen no está registrada

425 Too early: la imagen aún no se ha obtenido

500 Internal server error: La imagen no se ha podido recuperar

delete **/image/{name}**
Elimina una imagen

Parameter

name Nombre de la imagen

Response

204 No content

```
Binario con la imagen
```

500 Internal server error: Error genérico

post **/image/{name}**
Añade una imagen

Parameter

name Nombre de la imagen

Body application/json

```
{
  "name":"libvirt-base",
  "os": {
    "architecture":"amd64",
    "checksum":"ecf2b332c09b7548c67972826c18ead3",
    "distribution":"Debian",
    "family":"Linux",
    "format":"qcow2",
    "release":"bookworm"
  },
  "source-uri":"http://example.com/libvirt.qcow2",
  "uri":"/image/libvirt-base"
}
```

Response

201 Created

400 Bad request: el cuerpo de la solicitud presenta errores

500 Internal server error: La imagen no se ha podido añadir

put **/image/{name}**
Modifica una imagen

Parameter

name Nombre de la imagen

Body application/json

```
{
  "os": {
    "architecture":"amd64",
    "checksum":"ecf2b332c09b7548c67972826c18ead3",
    "distribution":"Debian",
    "family":"Linux",
    "format":"qcow2",
    "release":"bookworm"
  },
  "source-uri":"http://example.com/libvirt.qcow2",
  "uri":"/image/libvirt-base"
}
```

Response application/-disk

204 No content

400 Bad request: el cuerpo de la solicitud presenta errores

500 Internal server error: La imagen no se ha podido actualizar

Anexo M

Elementos relevantes del código

M.1. Organización del código y ejemplo de modelización de entidades

Los bloques de código M.3, M.2 y M.4 muestran la organización de código de los módulos de mayor relevancia. Se incluyen como ejemplo de la modelización los bloques de código M.5, M.6 y M.7.

```
internal/core/
|-- entities/
|   |-- image.go
|   |-- machine.go
|   |-- network.go
|   |-- plat/
|   |   |-- errors.go
|   |   |-- kubernetes.go
|   |   |-- libvirt.go
|   |   |-- openstack.go
|   |   `-- os.go
|   |-- platforms.go
|   `-- storage.go
|-- errors.go
|-- ports.go
`-- usecases/
    |-- baremetal/
    |   `-- baremetal.go
    |-- image/
    |   `-- image.go
    |-- platform/
    |   `-- platform.go
    `-- virt/
        `-- virt.go
```

Código M.1: Estructura de directorios capa de entidades y lógica de negocio

```
internal/delivery/
|-- cli
|   |-- commands
|   |   |-- baremetal
|   |   |   |-- deploy.go
|   |   |   |-- modules.go
|   |   |   |-- register.go
|   |   |   `-- root.go
|   |   |-- generic.go
|   |   `-- platforms
|   |       |-- libvirt
|   |       |   |-- deploy.go
|   |       |   |-- modules.go
|   |       |   `-- root.go
|   |       |-- modules.go
|   |       |-- openstack
|   |       |   |-- deploy.go
|   |       |   |-- modules.go
|   |       |   `-- root.go
|   |       `-- root.go
|   |-- helpers
|   |   `-- wrapper.go
|   |-- modules.go
|   `-- root.go
`-- image
    |-- controller.go
    `-- routes.go
```

Código M.2: Estructura de directorios módulo de presentación

```
internal/repositories/
|-- image
|   |-- dao.go
|   |-- errors.go
|   `-- repository.go
`-- inventory
    |-- dao.go
    |-- errors.go
    |-- machine.go
    |-- platform.go
    |-- repository.go
    |-- repository_test.go
    `-- serialize.go
```

Código M.3: Estructura de directorios módulo de persistencia

```
internal/services/
|-- baremetal
|   |-- baremetal.go
|   `-- http_types.go
|-- platforms
|   |-- kubernetes.go
|   |-- libvirt.go
|   `-- openstack.go
`-- virt
    |-- config_builders.go
    |-- errors.go
    |-- images.go
    |-- network.go
    |-- power.go
    `-- virt.go
```

Código M.4: Estructura de directorios módulo de servicios

```
// Abstract machine model.
//
// Defines common methods for all machine types, including those related to
// power and execution controls.
type Machine interface {
        GetName() string
        GetInterfaces() []NetworkInterface
        GetStorages() []Storage
        GetHost() string

        // Power controls
        PowerOn() error
        PowerOff() error
        GetPowerStatus() bool

        GetSshConfig() SshConfig
        GetStatus() string

        // If the machine is managed by a platfrom, returns it
        GetManagerPlatform() Plat

        // Execution controls
        ExecCommand(command string) (string, error)
        ExecScript(path string) (string, error)
        CopyFile(source string, destination string) error
}
```

Código M.5: Interfaz Máquina

```
// Physical Machine Entity.
//
// Models a given physical machine. Current version
// can only be controlled via ssh or ipmi.
// Although it could be easily extended to
// support other open protocols such as redfish, or
// vendor like iLO or iDRAC.
type PhysicalMachine struct {
        Name       string
        Status     string
        Host       string
        Ssh        SshConfig
        Interfaces []NetworkInterface
        Storages   []Storage
        Ipmi       IpmiConfig
        BootMode   string // bios or uefi
}

const (
        STATUS_BAREMETAL_MANAGED = "baremetal-managed"
        STATUS_FREE              = "free"
        STATUS_WITH_PLATFORMS    = "with-platforms"
        STATUS_NOT_MANAGED       = "not-managed"
)

type SshConfig struct {
        Username   string
        AdminHost  string
        PrivateKey string
}

type IpmiConfig struct {
        Address  string
        Username string
        Password string
        Port     *string
}
```

Código M.6: Entidad máquina física

```
// Calls exec controls to execute a command in the machine
func (m PhysicalMachine) ExecCommand(command string) (string, error) {
        out, err := ssh.RemoteRun(m.Ssh.Username, m.Ssh.AdminHost,
        ↪  m.Ssh.PrivateKey, command)
        return out, err
}

// Calls exec controls to execute a script in the machine
func (m PhysicalMachine) ExecScript(path string) (string, error) {
        out, err := ssh.RemoteRunFile(m.Ssh.Username, m.Ssh.AdminHost,
        ↪  m.Ssh.PrivateKey, path)
        return out, err
}

// Calls exec controls to copy a file in the machine
func (m PhysicalMachine) CopyFile(source string, destination string)
↪  error {
        return ssh.CopyFileToRemote(m.Ssh.Username, m.Ssh.AdminHost,
        ↪  m.Ssh.PrivateKey, source, destination)
}

// Other methods are ommitted for clarity
```

Código M.7: Fragmento de implementación del interfaz Máquina