ESCUELA POLITÉCNICA

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO EN INGENIERÍA INFORMÁTICA EN
INGENIERÍA DEL SOFTWARE

TRABAJO FIN DE GRADO

CML: Modelado, Validación y Generación de Aplicaciones Basadas en Contenedores

ESCUELA POLITÉCNICA

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL SOFTWARE

TRABAJO FIN DE GRADO

CML: Modelado, Validación y Generación de Aplicaciones Basadas en Contenedores

Autor: Lorenzo Gabriel Ceballos Bru

Tutora: Cristina Vicente Chicote

Co-Tutor: José Ramón Lozano Pinilla

Resumen

En los últimos años, se viene observando un creciente interés en el ámbito del desarrollo de software por el uso de tecnologías basadas en contenedores. Actualmente, son muchas las empresas que utilizan estas tecnologías en sus proyectos y su popularidad sigue en aumento.

Los contenedores permiten empaquetar, distribuir y administrar aplicaciones de forma independiente de la plataforma sobre las que éstas de despliegan y ejecutan, ofreciendo una alternativa muy atractiva (por su ligereza y menor demanda de recursos) a las soluciones basadas en el uso de máquinas virtuales. La creciente complejidad del software basado en contenedores hace cada vez más necesario el uso de mecanismos adicionales de orquestación que permitan gestionar la ejecución de los numerosos contenedores que pueden llegar a integrar los complejos sistemas actuales.

Entre las tecnologías basadas en contenedores, Docker es posiblemente la más extendida hoy en día. Por su parte, Docker-Compose es una de las herramientas más utilizadas para la orquestación de arquitecturas basadas en Docker. Sin embargo, a pesar de la creciente popularidad de estas tecnologías, la especificación y orquestación de arquitecturas basadas en Docker puede llegar a resultar muy compleja, entre otras cosas, debido a (1) la complejidad del estándar que define dichas especificaciones; y (2) la falta de herramientas que faciliten su definición y validación automática. Como resultado, la curva de aprendizaje de estas tecnologías resulta muy elevada y el proceso que es necesario llevar a cabo para definir y validar una arquitectura basada en Docker/Docker-Compose, muy lento y laborioso.

En este contexto, el Trabajo Fin de Grado que aquí se presenta, pretende desarrollar un conjunto de herramientas que faciliten la especificación, validación y visualización de arquitecturas basadas en contenedores y, más específicamente, en Docker/Docker-Compose. Estas herramientas no sólo persiguen reducir la curva de aprendizaje para aquellos desarrolladores que no estén familiarizados con el uso de estas tecnologías, sino que ofrecerán a los desarrolladores más expertos, nuevas características actualmente no soportadas por otras herramientas (validación automática de las especificaciones, representación dual sincronizada de las mismas en formato gráfico y textual, etc.).

Abstract

In recent years, there has been a growing interest for the use of container-based technologies within the software development community. Currently, many companies use these technologies in their projects and their popularity continues to grow.

Containers allow applications to be packaged, distributed and managed independently of the platform on which they are deployed and executed, offering a very attractive alternative to solutions based on virtual machines (heavier and more resource consuming). The growing complexity of container-based software makes it increasingly necessary to use additional orchestration mechanisms to manage the execution of the numerous containers that can integrate today's complex systems.

Among container-based technologies, Docker is possibly the most widespread in use today. Similarly, Docker-Compose is a widely adopted tool for orchestrating Docker-based architectures. However, despite the growing popularity of these technologies, the specification and orchestration of Docker-based architectures can become very complex, among other reasons due to: (1) the complexity of the standard defining the specifications; and (2) the lack of tools enabling their definition and automatic validation. As a result, the learning curve for these technologies is quite steep, and the process required to define and validate a Docker/Docker-Compose-based architecture quite slow and burdensome.

In this context, the Final Degree Project presented here develops a set of tools aimed at easing the specification, validation and visualization of Docker/Docker-Compose-based architectures. These tools not only seek to reduce

the learning curve for novel developers, but to provide more experienced ones with new features, currently not supported by existing tools (automatic validation of the specifications, dual and synchronized graphic-textual representation, etc.).

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Hoy en día, muchas de las aplicaciones software que se desarrollan siguen siendo monolíticas, esto es, se diseñan para ejecutarse en una única máquina física con determinadas características hardware. Estas aplicaciones, sobre todo las de cierta envergadura, se organizan como un conjunto de componentes acoplados entre sí que se desarrollan, despliegan y gestionan como una única entidad dentro de un *ecosistema software* compuesto, entre otros elementos, por un sistema operativo y una serie de bibliotecas. Los cambios en la versión o en la configuración de estos elementos (introducidos a veces como requisito para otras aplicaciones que se ejecutan en la misma máquina) o la necesidad de migrar a una nueva plataforma hardware puede afectar significativamente al rendimiento e incluso al correcto funcionamiento del software monolítico. Para evitar los conflictos asociados a las distintas configuraciones requeridas por varias aplicaciones lo más sencillo sería ejecutarlas en distintas máquinas, cada una de ellas convenientemente equipada con el ecosistema necesario para su ejecución. Sin embargo, esta solución puede resultar difícil de escalar y no siempre es económicamente viable, debido al alto coste que implica la adquisición de nuevas máquinas.

En respuesta a estos problemas, en los últimos años se observa un aumento en la popularidad de las denominadas *técnicas de virtualización*, gracias a las

cuales es posible ejecutar, en una única máquina física, varias “máquinas virtuales” concurrentes, cada una de ellas equipada con su propio ecosistema. Las máquinas virtuales permiten desacoplar completamente el entorno de ejecución de una aplicación del hardware subyacente. Como contrapartida, su uso suele implicar un mayor consumo de recursos, lo que hace poco aconsejable su uso en determinados casos.

De forma casi simultánea a las máquinas virtuales, aunque comenzando a ganar popularidad mucho más tarde que éstas, surgen las denominadas *tecnologías de contenedores*. Los contenedores ofrecen un mecanismo de empaquetado lógico que permite aislar entre sí las aplicaciones que se ejecutan en una misma máquina. Cada contenedor puede alojar el código de una aplicación y todas sus dependencias (entorno de ejecución, librerías, configuraciones, etc.). Todos estos elementos se aíslan permitiendo la ejecución del contenedor en cualquier máquina, independientemente de su configuración.

Tanto los contenedores como las máquinas virtuales permiten ejecutar, de forma aislada, varias aplicaciones en una misma máquina física. Sin embargo, los contenedores no necesitan tener su propio sistema operativo sino que se les asignan ciertos recursos del sistema operativo y del hardware del equipo anfitrión. Por ello, son una alternativa a las máquinas virtuales mucho más ligera.

Conviene señalar que los contenedores no han existido siempre como los conocemos hoy en día [27]. De hecho, no es hasta el año 1979 que aparece la primera herramienta que ofrece un cierto nivel de aislamiento de procesos. La llamada al sistema ’chroot’, introducida durante el desarrollo de Unix V7, permitía cambiar el directorio raíz de un proceso y de sus hijos a una nueva ubicación del sistema de ficheros. Aunque la funcionalidad de este comando es muy limitada, supuso el primer paso hacia las tecnologías de aislamiento de procesos. Dos décadas más tarde, en el año 2000, *FreeBSD* introdujo un mecanismo de jaulas (*jails*), que utilizaban una virtualización a nivel de sistema operativo para conseguir una clara separación entre sus servicios y los de sus clientes. Las *FreeBSD jails* permitían particionar un sistema FreeBSD en

varios sistemas independientes, a los que se les podían asignar sus propias direcciones IP e interfaces de red. Sólo un año más tarde, surgió otro mecanismo similar, *Linux VServer*, gracias al que era posible particionar determinados recursos del sistema (memoria, direcciones de red, sistema de ficheros, etc.). Sin embargo, a pesar de que algunos años más tarde Linux desarrolló otro sistema (OpenVZ) que permitía la virtualización, el aislamiento y la gestión de recursos, estas tecnologías todavía no eran lo suficientemente completas como para acercarse a las funcionalidades que ofrecían las máquinas virtuales.

En el año 2008, surge la tecnología LXC (*Linux Containers*), la primera y más completa implementación de un sistema de gestión de contenedores. Esta tecnología ofrecía un mecanismo para limitar y priorizar los recursos de distintas aplicaciones, consiguiendo además un aislamiento completo de sus procesos, sistemas de ficheros y redes. LXC se basa en el uso de *cgroups* y *namespaces*. Por un lado, los *Control Groups* (cgroups), presentados por Google, permiten aislar y contabilizar el uso de recursos de un grupo de procesos. Los *namespaces* de Linux, por otro lado, son un mecanismo de Linux cuyo objetivo es que cada proceso solo pueda ver los recursos de su sistema virtual, ocultándolos al resto de procesos. A pesar del gran avance que supuso LXC, su enorme complejidad limitó considerablemente su éxito, no logrando el grado de adopción esperado por parte de la industria del software.

La tecnología *Docker* [3] surge en 2013 y, a diferencia de LXC, abstrae la complejidad asociada al uso de determinadas características del kernel de Linux, haciéndolo mucho más sencillo de usar. Docker permite a los desarrolladores empaquetar, distribuir y administrar sus aplicaciones utilizando contenedores en los que se alojan todas sus dependencias, bibliotecas, configuraciones y, en general, todos los elementos necesarios para su correcta ejecución [8]. Este empaquetado hace posible que las aplicaciones se ejecuten en cualquier máquina Linux, independientemente de su configuración. En sus orígenes, Docker utilizó tecnología LXC hasta que ésta fue sustituida por una librería propia, *libcontainer* [24]. La simplicidad de Docker ha favorecido su adopción y ha hecho que el uso de contenedores se popularice

enormemente en los últimos años, hasta el punto de convertirse en la tecnología más utilizada para el despliegue de software en cualquier servidor [29]. De hecho, empresas como IBM, Microsoft, Google, Cisco o RedHat apostaron por esta tecnología y contribuyeron significativamente a su desarrollo [24].

A pesar de todas las ventajas que ofrece Docker, el aumento incesante en la complejidad del software hace que el uso de esta tecnología resulte insuficiente en proyectos de gran envergadura, en particular, cuando es necesario coordinar la ejecución de un número elevado de aplicaciones desplegadas en contenedores. Para abordar este problema, en los últimos años, han surgido diversas herramientas denominadas orquestadoras (*orchestators*), tales como *Kubernetes* [9], *Docker-Compose* [16], o *Docker Swarm* [6], éstas últimas basadas en Docker.

Conviene señalar que en la actualidad, casi la cuarta parte de las empresas de software hacen uso de Docker como herramienta para desplegar sus aplicaciones (23.4 % en 2018) [10] y, aproximadamente la mitad de ellas, utilizan además herramientas para facilitar la orquestación de sus sistemas.

1.1. Descripción del problema

A pesar de la gran evolución que han experimentado las tecnologías basadas en contenedores, actualmente su adopción está siendo más lenta y compleja de lo esperado. De hecho, son muchas las empresas que encuentran dificultades para migrar sus arquitecturas actuales a otras basadas en contenedores.

Aunque existen algunas herramientas para la especificación de arquitecturas basadas en contenedores y su orquestación, éstas presentan importantes limitaciones, sobre todo en lo relativo a su validación, requiriendo un proceso largo, tedioso y muy poco intuitivo de depuración. En el caso concreto de Docker-Compose, en el que se ha centrado este Trabajo Fin de Grado, hasta donde sabemos, no existe ninguna herramienta que facilite la edición y validación de dichas especificaciones conforme al

estándar [16]. Es más, dicho estándar establece únicamente qué estructura deben tener los fichero de texto necesarios para definir, configurar y orquestar una arquitectura basada en contenedores Docker (sintaxis textual del lenguaje de especificación de Docker-Compose). Sin embargo, en particular en aplicaciones complejas en las que aparecen involucrados un número elevado de contenedores, consideramos que sería de utilidad poder contar con una representación gráfica de las especificaciones con la que, abstrayendo muchos de los detalles del equivalente textual, fuera posible observar, de un vistazo, cómo está organizada una determinada arquitectura basada en contenedores.

En esta línea, a continuación se detallan los objetivos planteados en el presente Trabajo.

1.2. Objetivos

El principal objetivo de este Trabajo Fin de Grado consiste en desarrollar un conjunto de herramientas que faciliten a los desarrolladores de software la adopción de Docker y Docker-Compose como tecnologías para el diseño, despliegue y orquestación de arquitecturas basadas en contenedores. Asimismo, estas herramientas deberán permitir a los desarrolladores que ya utilizan estas tecnologías seguir trabajando con especificaciones previamente creadas, sin necesidad de realizar ninguna adaptación en ellas ni en su modo de trabajar.

Este objetivo general se concreta en los siguientes subobjetivos:

- Estudiar la estructura de los ficheros textuales definida en el estándar de Docker-Compose [16].
- A partir del estudio anterior, definir la gramática de un lenguaje textual de modelado que permita crear y validar especificaciones textuales conformes a dicho estándar.
- Implementar un editor textual a partir de la gramática anterior que ofrezca

facilidades de compleción de sintaxis, formateo automático del texto y validación de las especificaciones.

- Definir una representación gráfica asociada a los principales elementos de las especificaciones textuales.
- Implementar un editor gráfico que permita crear, mostrar y editar especificaciones de arquitecturas Docker-Compose tanto desde cero como a partir de una especificación textual previa.
- Lograr mantener correcta y automáticamente sincronizados los modelos gráficos y textuales.

1.3. Metodología

El Proyecto se ha llevado a cabo siguiendo una metodología ágil, con iteraciones semanales. Cada semana, en la reunión de seguimiento con los tutores del Trabajo, se analizaban las dificultades afrontadas y los avances realizados durante la semana anterior y se fijaban los objetivos y ajustes necesarios en la planificación para la siguiente iteración. Las reuniones de seguimiento se realizaron tanto de forma presencial como telemática. Lo discutido en cada reunión se fue reflejando en un documento compartido a modo de libro de actas. Adicionalmente, se utilizó la herramienta Click-Up para definir y hacer un seguimiento de las distintas tareas definidas en cada reunión y registrar las horas empleadas en cada una de ellas. Conviene señalar que se utilizó GitHub a lo largo de todo el Proyecto como sistema de control de versiones y que toda la documentación, tanto interna como externa, se fue realizado en paralelo a la implementación desde el primer momento, lo que ha facilitado considerablemente el desarrollo y el seguimiento del Trabajo.

1.4. Planificación de tareas

A continuación, se detallan las tareas desarrolladas a lo largo del proyecto, así como su duración.

- **Análisis de requisitos**. A partir de los objetivos del proyecto, en esta tarea se definieron los principales requisitos funcionales y no funcionales de las herramientas a desarrollar. A esta tarea se le dedicaron, aproximadamente, 15 horas.
- **Diseño de las herramientas de modelado**. Partiendo de los requisitos previamente identificados, se diseñaron las distintas herramientas del Proyecto, identificando las funcionalidades que debía implementar cada una de ellas. Esta tarea requirió de unas 20 horas.
- **Estudio de herramientas disponibles**. Como parte de esta tarea se realizó un estudio de las herramientas más utilizadas actualmente para crear/editar especificaciones Docker-Compose. Asimismo, se analizaron las tecnologías disponibles para poder implementar las herramientas previamente diseñadas. En esta tarea se invirtió aproximadamente 25 horas.
- **Estudio del estándar Docker-Compose**. En esta tarea se estudió en profundidad el estándar de especificación de los ficheros Docker-Compose y se realizaron varios ejemplos para asegurar su correcta y total comprensión. Esta tarea requirió unas 40 horas de trabajo.
- **Definición del meta-modelo de Docker-Compose**. A partir del estándar anterior, en esta tarea se implementó el meta-modelo asociado a la sintaxis abstracta del lenguaje de especificación de arquitecturas Docker-Compose. Este meta-modelo se enriqueció con una serie de restricciones adicionales, algunas directamente derivadas del estándar y otras de desarrollo propio. A esta tarea se le dedicó unas 30 horas.

- **Implementación del editor textual de Docker-Compose**. A partir del meta-modelo anterior se generó una gramática por defecto que, posteriormente, se extendió y enriqueció para, a partir de ella, implementar un editor textual que permitiera la creación y edición de especificaciones Docker-Compose, completas y conformes al estándar. En esta tarea se invirtieron unas 50 horas de trabajo.

- **Implementación del editor gráfico de Docker-Compose**. Como parte de esta tarea se implementó un editor para poder crear y editar especificaciones Docker-Compose de forma gráfica. Este editor se integró con el editor textual desarrollado en la tarea anterior y se extendió para permitir a los usuarios: (1) visualizar la representación textual asociada a cada uno de los elementos del modelo gráfico; y (2) mostrar u ocultar las distintas capas (tipos de elementos) del diagrama. A esta tarea se le dedicó unas 25 horas.

- **Validación de las herramientas**. Esta tarea permitió comprobar el correcto funcionamiento de las herramientas desarrolladas. Para ello, se especificaron varias arquitecturas (tanto desde cero como a partir de algunos de los ejemplos disponibles en Internet) utilizando ambos editores y comprobando que, en todos los casos, estos funcionaban correctamente y la sincronización entre los dos tipos de especificaciones era correcta. A esta tarea se le dedicó unas 30 horas.

- **Documentación**. Como parte de esta tarea se ha redactado toda la documentación, interna y externa, del Proyecto. Como ya se ha indicado previamente, la tarea de documentación se ha ido desarrollando a lo largo de toda la duración del Proyecto y, en total, ha supuesto unas 75 horas de trabajo.

Como se puede observar en el **diagrama de Gantt** incluido a continuación (ver Figura 1.1), el tiempo total invertido en el desarrollo del Proyecto ha sido de unas 310 horas.

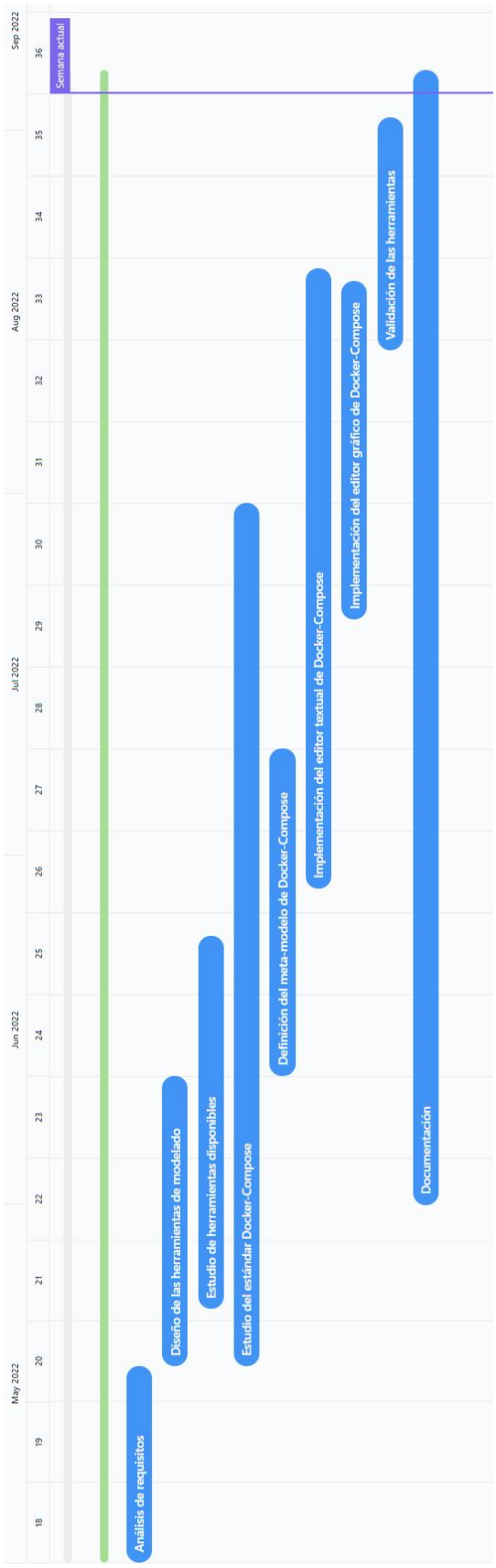

Figura 1.1: Diagrama de Gantt

1.5. Estructura del documento

El resto de la presente Memoria se organiza como sigue:

- **Capítulo 2.** Introduce algunos de los conceptos fundamentales necesarios para comprender las tecnologías utilizadas en el Proyecto y describe, de forma resumida, algunas de las herramientas disponibles en la actualidad para crear y editar ficheros de especificación de arquitecturas basadas en Docker/Docker-Compose;
- **Capítulo 3.** Recoge el resultado de las fases de análisis y diseño de las herramientas que se han desarrollado como parte del Proyecto. Incluye también el diseño de la sintaxis abstracta y concreta (tanto textual como gráfica) del lenguaje de especificación de arquitecturas Docker/Docker-Compose en el que se ha basado la implementación de dichas herramientas;
- **Capítulo 4.** Describe la implementación de la sintaxis abstracta, previamente diseñada, mediante un meta-modelo y un conjunto de restricciones adicionales. Detalla cómo se han implementado los editores gráfico y textual, así como los mecanismos utilizados para su sincronización;
- **Capítulo 5.** Presenta dos ejemplos de uso de las herramientas desarrolladas, uno propio y otro basado en la especificación de una arquitectura existente, disponible en Internet;
- **Capítulo 6.** Expone las principales dificultades encontradas a lo largo del Trabajo, así como las soluciones adoptadas en cada caso. Cierra la memoria presentando las conclusiones y los posibles trabajos que podría resultar de interés abordar en el futuro como continuación del Proyecto.

Capítulo 2

Conceptos previos y trabajos relacionados

En esta sección se introducen brevemente los principales conceptos relacionados con las tecnologías y herramientas utilizadas en el Proyecto. Además, se revisan brevemente las herramientas existentes similares a las desarrolladas en este Trabajo.

2.1. Conceptos previos

Como se ha indicado anteriormente, las principales tecnologías relacionadas con contenedores que se van a utilizar en este Trabajo son Docker y Docker-Compose. A continuación, se definen los principales elementos y conceptos relacionados con ambas tecnologías.

2.1.1. Docker

Docker es la tecnología principal en torno a la que se van a desarrollar las herramientas propuestas en este trabajo. Docker permite empaquetar aplicaciones en contenedores que albergan, tanto su código, como las bibliotecas, configuraciones y, en general,

cualquier dependencia necesaria para su correcto despliegue y ejecución. Idealmente, cada contenedor se ocupará de ejecutar un único proceso, por lo que una aplicación puede utilizar tantos contenedores, interoperando entre sí, como sean necesarios. Las principales ventajas que se obtienen al aislar estos procesos en contenedores son: (1) la posibilidad de ejecutar las aplicaciones en distintas plataformas, agilizando su despliegue, gestión y mantenimiento; y (2) el bajo coste que, en términos de los recursos del sistema, conlleva esta solución, comparada con otras basadas, por ejemplo, en virtualización.

El principal concepto que utiliza Docker es el de **imagen**. Una imagen es una plantilla de solo lectura que permite crear uno o varios contenedores idénticos. Estas imágenes no necesitan contar con su propio sistema operativo (a diferencia de las máquinas virtuales), ya que incluyen tanto el código de la aplicación como todas las dependencias y configuraciones necesarias para su correcto funcionamiento. La estructura de estas imágenes se define mediante un archivo de texto denominado **Dockerfile** [24].

Por otro lado, un **contenedor** es una instancia activa y en ejecución de una imagen. Docker permite la creación, el arranque, la detención y la destrucción de los distintos contenedores que integran cada aplicación y se encarga de mantenerlos aislados unos de otros, de forma que cada uno tendrá sus propios recursos y no podrá acceder a los de los demás. De este modo, Docker permite la definición de arquitecturas altamente cohesionadas y muy poco acopladas.

Docker también permite definir otra serie de elementos que interactúan entre sí y con los contenedores del ecosistema: volúmenes, redes, configuraciones y secretos.

Los **volúmenes** (`Volume`) almacenan los datos que utilizan los contenedores en una ubicación específica dentro del sistema de ficheros del equipo anfitrión. Los volumen pueden ser compartidos por varios contenedores y deben conservarse más allá de su vida útil.

Las **redes** (`Network`) permiten a los distintos contenedores comunicarse de manera sencilla, asignándoles una dirección IP a cada uno de ellos. De esta forma, es posible

definir y configurar de distinto modo las redes de cada arquitectura, estableciendo distintos protocolos de comunicación e incluso distintos tipos de redes.

Por último, las **configuraciones** (`Configs`) y los **secretos** (`secrets`) permiten adaptar el comportamiento de un contenedor sin necesidad de modificar su imagen. La principal diferencia entre ambos es que el uso de secretos suele limitarse a aquellos aspectos que involucran datos sensibles, como claves de APIs o contraseñas, permitiendo especificar restricciones para su acceso.

El funcionamiento de Docker se consigue gracias a la interconexión de distintos componentes software a través de la **API Docker Engine**, siendo el **Docker Engine** el motor de esta tecnología. Este motor, que puede ejecutarse en cualquier sistema local, se controla mediante un cliente de línea de comandos que permite la creación de imágenes y contenedores Docker. Además de este cliente, otro de los componentes principales del Docker Engine es el denominado **Docker Daemon**: un proceso que se encarga de responder a los comandos relacionados con la gestión de los contenedores y del resto de los objetos Docker. Este proceso se ejecuta en segundo plano, recibiendo las peticiones de la API Docker Engine.

2.1.2. Docker-Compose

Como ya se ha mencionado anteriormente, el proceso de despliegue y gestión de aplicaciones basadas en contenedores se complica a medida que éstas son cada vez de mayor envergadura y necesitan un mayor número de contenedores. Así, resulta frecuente tener que recurrir a herramientas de orquestación que faciliten la gestión y coordinación de estos contenedores. Entre ellas, en el ámbito de la tecnología Docker, *Docker-Compose* es posiblemente la más popular actualmente. Esta herramienta permite definir y configurar todos los servicios requeridos por una determinada aplicación por medio de un archivo *YAML*. A partir de este fichero, el despliegue de la aplicación (creación y ejecución de todos sus servicios) puede hacerse ejecutando un único comando “docker-compose”.

En el archivo “docker-compose.yml” deben definirse todos los contenedores que va utilizar la aplicación que se quiere ejecutar. Para cada contenedor, deberá especificarse la imagen que se va a utilizar para crearlo. Esta imagen puede obtenerse de Internet (especificando su nombre en la propiedad “image”) o de un Dockerfile almacenado en el sistema de ficheros del equipo anfitrión (especificando su ruta en la propiedad “build”). Además de los contenedores, el fichero “docker-compose.yml” deberá contener la especificación de los volúmenes, redes, configuraciones y secretos necesarios para ejecutar la aplicación.

2.2. Trabajos relacionados

Como ya se ha mencionado en la sección anterior, Docker-Compose permite ejecutar arquitecturas Docker cuya especificación se debe almacenar en un fichero de tipo YAML [28]. En la página web de Docker se encuentra disponible la especificación del estándar de Docker-Compose [16]. Este estándar define el formato que debe tener el fichero “docker-compose.yml”, indicando cómo deben especificarse los distintos elementos de la arquitectura y sus propiedades.

Existen numerosos validadores de ficheros YAML, como **JSON formatter** [1] o **CodeBeautify** [2]. Estas herramientas se limitan a comprobar si el contenido de un archivo cumple con el formato y la sintaxis de YAML. YAML permite definir valores sencillos o escalares (números, cadenas, booleanos o fechas) y estructuras de mapas y listas. Los mapas permiten asociar pares clave-valor (las clave deben ser únicas), mientras que las listas pueden incluir cualquier número de valores distintos, cada uno de ellos precedido de un guión. Herramientas como las dos mencionadas previamente pueden servir para comprobar la corrección de un fichero “docker-compose.yml” pero sólo en relación a la sintaxis de YAML. Ninguna de ellas permite comprobar si la especificación incluida en el fichero “docker-compose.yml” es o no conforme al estándar de Docker-Compose, esto es, si los distintos elementos de la arquitectura y sus propiedades están correctamente definidos o no.

Entre las herramientas disponibles para llevar a cabo la validación de especificaciones Docker-Compose, cabe destacar **Docker-Compose Config** [4]. Esta herramienta es capaz de identificar únicamente errores de formato, tipográficos (p.ej., palabras reservadas mal escritas) y valores duplicados. Una alternativa a esta herramienta, aún más limitada, es **Docker-Compose Validator** [7].

Sin embargo, ninguna de estas herramientas permite comprobar, de forma completa, la corrección de una especificación Docker-Compose. Además, el hecho de que el parseo que realizan de del fichero "docker-compose.yml" no se base en una gramática formal del lenguaje definido en el estándar, hace que sean poco robustas y confiables. De hecho, muchos de los desarrolladores que trabajan con esta tecnología reportan dificultades para depurar sus especificaciones e indican que suele ser necesario llevar a cabo numerosas iteraciones a base de prueba y error, haciendo que el proceso de validación resulte largo y tedioso.

Capítulo 3

Análisis y diseño

En este apartado se va a describir el análisis que se ha realizado previo al desarrollo del Proyecto, además del diseño de la sintaxis abstracta y concreta del lenguaje de modelado en el que se ha basado la implementación de las herramientas desarrolladas.

3.1. Análisis

A continuación, se indican los distintos requisitos, tanto funcionales como no funcionales que se han definido para el sistema. Además, se indicarán los distintos casos de uso y su diagrama.

3.1.1. Requisitos funcionales

Los requisitos funcionales del sistema se encuentran listados en la Tabla 3.1.

3.1.2. Requisitos no funcionales

Los principales requisitos no funcionales se indican en la Tabla 3.2

Nº	Descripción
RF1	Soporte a modelos Docker-compose
RF2	Coloración de sintaxis para modelos Docker-compose
RF3	Formato automático para modelos Docker-compose
RF4	Validación de modelos Docker-compose
RF5	Creación de modelos Docker-compose
RF6	Soporte a edición gráfica/textual de modelos Docker-compose

Tabla 3.1: Requisitos funcionales.

Nº	Descripción
RNF1	Facilidad de uso
RNF2	Gestión de errores
RNF3	Interfaz gráfica sencilla
RNF4	Basado en especificación Docker-compose
RNF5	Fácilmente escalable a otros modelos
RNF6	Basado en meta-modelos

Tabla 3.2: Requisitos no funcionales.

3.1.3. Descripción de los actores

Para el desarrollo del sistema propuesto, únicamente hemos considerado como actores a los arquitectos encargados de diseñar arquitecturas Docker-Compose, independientemente de su nivel de conocimientos y experiencia en uso de tecnologías basadas en contenedores.

Estos actores podrán definir especificaciones “docker-compose.yml” utilizando para ello un editor textual que permitirá su validación, mostrará sugerencias de auto-completado y ofrecerá coloración de sintaxis. Todas estas funcionalidades también se encuentran disponibles para aquellos archivos de definición del sistema creados previamente, que pueden ser abiertos directamente con el editor textual. Asimismo, podrán generar diagramas a partir de modelos textuales previamente definidos. Gracias al uso de un editor gráfico, podrán editar estos diagramas, añadiendo y configurando nuevos elementos de entre los disponibles en su paleta.

De esta forma, se facilitará el proceso de definición, validación y despliegue de

sistemas con la herramienta Docker-Compose, tanto si los usuarios se está iniciando en el uso de esta tecnología como si son diseñadores experimentados, ya que las herramientas proporcionadas resultan muy sencillas e intuitivas de utilizar.

3.1.4. Descripción y diagrama de casos de uso

En la Tabla 3.3 se definen los diferentes casos de uso del sistema, los cuales serán realizados por el único actor del mismo, el arquitecto de sistemas.

Nº	Descripción
CU1	Creación de un nuevo modelo Docker-Compose
CU2	Creación de un diagrama a partir de un modelo Docker-compose
CU3	Edición gráfica/textual de un modelo Docker-compose
CU4	Validación de un modelo Docker-Compose

Tabla 3.3: Casos de uso.

A continuación, se describen los casos de uso definidos para el sistema, cada uno de ellos representado en una tabla diferente (Tabla 3.4, Tabla 3.5, Tabla 3.6 y Tabla 3.7, respectivamente).

CU-01	**Creación de un nuevo modelo Docker-compose**	
Actores	Arquitecto de sistemas	
Descripción	El arquitecto de sistemas podrá crear un nuevo modelo Docker-Compose como un fichero de texto	
Secuencia Normal	**Paso**	**Acción**
	1	Crear un nuevo proyecto de modelado
	2	Seleccionar la opción “New” → “File” sobre el nuevo proyecto
	3	Especificar el nombre del archivo con extensión .yaml
	4	Definir el modelo con la sintaxis de Docker-Compose
	5	Guardar el modelo

Tabla 3.4: Descripción del caso de uso 01

CU-02	**Creación de un diagrama a partir de un modelo Docker-compose**	
Actores	Arquitecto de sistemas	
Descripción	El arquitecto de sistemas podrá generar una representación gráfica de un modelo Docker-Compose en formato textual	
Secuencia Normal	**Paso**	**Acción**
	1	Expandir el fichero de extensión .yaml correspondiente al modelo
	2	Seleccionar la opción “New Representation” sobre el elemento raíz del modelo
	3	Seleccionar la opción “new Docker-Compose Diagram”
	4	Especificar el nombre del diagrama
	5	Seleccionar la opción “OK” para crear el diagrama

Tabla 3.5: Descripción del caso de uso 02

CU-03	**Edición gráfica/textual de un modelo Docker-Compose**		
Actores	Arquitecto de sistemas		
Descripción	El arquitecto de sistemas podrá editar un modelo Docker-Compose tanto de manera textual como gráfica		
Secuencia Normal	**Paso**	**Acción**	
	1	Abrir un modelo existente	
		1.1	Para la edición textual, abrir el archivo de extensión .yaml correspondiente al modelo
		1.2	Para la edición gráfica, abrir el diagrama de tipo “Docker Compose Diagram” asociado al modelo
	2	Realizar los cambios correspondientes	
	3	Guardar el modelo	

Tabla 3.6: Descripción del caso de uso 03

CU-04	**Validación de un modelo Docker-Compose**	
Actores	Arquitecto de sistemas	
Descripción	El arquitecto de sistemas podrá validar un modelo Docker-Compose definido con cualquiera de los editores	
Secuencia Normal	**Paso**	**Acción**
	1	Abrir la repesentación gráfica de un modelo Docker-Compose
	2	Elegir la opción “Validate diagram” tras hacer click derecho en el fondo del diagrama
	3	Revisar los errores y advertencias en caso de que se hayan generado
Comentarios	Los pasos descritos en la secuencia no corresponden al caso de la gramática textual, ya que la validación es automática en ese caso	

Tabla 3.7: Descripción del caso de uso 04

Adicionalmente, en la Figura 3.1 se muestra el diagrama de casos de uso, que en este caso es muy sencillo, debido a que solo existe un único actor.

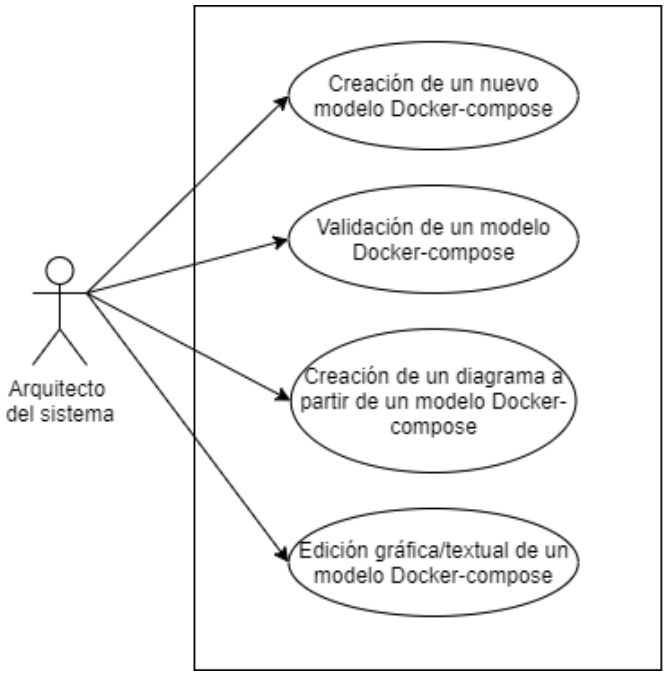

Figura 3.1: Diagrama de casos de uso

3.2. Diseño

En este apartado se va a describir cómo se ha diseñado tanto la sintaxis abstracta como las dos sintaxis concretas (textual y gráfica) del lenguaje de modelado definido para Docker-Compose.

Antes de comenzar con cada una de las sintaxis mencionadas, es necesario definir qué es un modelo y la forma en la que se puede representar. Un modelo es una descripción simplificada de un sistema software, que funciona como una abstracción para comprender mejor la realidad. Para representar modelos se utilizan **lenguajes específicos de dominio** (*DSL*, *domain-specific language*), que son lenguajes de modelado que pueden ser tanto gráficos como textuales.

Estos lenguajes deben incluir una sintaxis (tanto abstracta como concreta) y una semántica. La *sintaxis abstracta* es el conjunto de combinaciones lógicas de los elementos del lenguaje, mientras que la *sintaxis concreta*, que depende de la abstracta, define la notación del lenguaje de modelado [22].

3.2.1. Sintaxis abstracta

La sintaxis asbtracta de un lenguaje de modelado para Docker-Compose debe permitir modelar los conceptos descritos en su propia especificación, previamente enlazada. Según esta, en Docker-Compose existen cinco conceptos principales, los cuales son servicios (contenedores), redes, volúmenes, configuraciones y secretos.

Los **contenedores** representan el concepto más importante de la herramienta, puesto que son estos los que conforman las aplicaciones que permite Docker desarrollar. Estos tienen una serie de propiedades que permiten configurar su comportamiento. La única característica que debe especificarse de manera obligatoria en un contenedor es la imagen a partir de la cual debe ser creado, la cual puede especificarse mediante la propiedad *image*, que permite utilizar cualquier imagen disponible en Docker Hub [5],

o *build*, que permite especificar una imagen existente en el sistema de ficheros del equipo local en el que se ejecuta la aplicación.

Docker ofrece un número muy elevado de propiedades que pueden configurarse para cada contenedor, por lo que se ha decidido que la sintaxis abstracta contemple únicamente un número reducido de estas. Para ello, se han seleccionado algunas de las más importantes, concretamente, las propiedades *cpu_count*, *command*, *container_name*, *restart*, *init*, *read_only*, *devices*, *dns*, *ports* y *environment*. Los contenedores, además, pueden estar conectados entre sí mediante enlaces o depender de otros, propiedades que también se encuentran disponibles en las herramientas (*links* y *depends_on*, respectivamente) .

Otro de los conceptos que permite definir Docker-Compose son las **redes**, que pueden utilizarse para comunicar distintos componentes de un sistema y, al igual que los contenedores, tienen una serie de propiedades que permiten configurar su funcionamiento. Con estas puede especificarse el controlador que debe ser utilizado por la red (propiedad *driver*), una serie de opciones del controlador (*driver_opts*), un nombre personalizado para la red (*name*), una configuración IPAM para la misma (*ipam*) y algunas otras características. Adicionalmente, cabe destacar que los contenedores pueden relacionarse con distintas redes o incluso con ninguna si no requiere conexión a otros contenedores.

Los **volúmenes** se encargan de almacenar los datos que son utilizados por los contenedores, conservándolos más allá de la vida útil de estos, y algunas de las propiedades opcionales que se han definido para estos son el controlador que se debe utilizar (propiedad *driver*), una serie de opciones para el controlador (*driver_opts*), un nombre personalizado para el volumen (*name*), una serie de etiquetas (*labels*) o si se trata o no de un volumen externo (*external*). Al igual que con las redes, los vólumnes pueden relacionarse con uno o varios contenedores, permitiendo así que los datos de estos vólumenes sean accesibles desde los contenedores.

Por su lado, las **configuraciones** permiten adaptar el comportamiento de un contenedor

y sus propiedades permiten indicar si se trata de una configuración externa o no (propiedad *external*), así como especificar un fichero a partir del cual crearla (*file*) o el nombre de una configuración externa existente (*name*). Las configuraciones pueden relacionarse con contenedores.

Los **secretos** representan el último concepto de la sintaxis abstracta y funcionan de la misma manera que las configuraciones, aunque se encuentran más orientados a datos sensibles. Ambos conceptos tienen las mismas propiedades, y los secretos también pueden asociarse a distintos contenedores. Se exige que, tanto estos como las configuraciones, se definan como externos o bien se especifique un fichero a partir del cual crearlos.

Se han añadido, además, algunas restricciones a la sintaxis abstracta. Estas impiden, entre otras cosas, que se definan relaciones duplicadas, que un contenedor dependa o este enlazado consigo mismo o que haya elementos repetidos en listas de etiquetas u opciones.

3.2.2. Sintaxis concreta

La sintaxis concreta de un lenguaje de modelado se trata del conjunto de símbolos que se utilizan para la definición de modelos, estableciendo de esta forma la notación del propio lenguaje. La sintaxis concreta depende de la abstracta, ya que esta última define lo que puede ser expresado en un modelo válido. De hecho, una sintaxis abstracta es compatible con múltiples representaciones (sintaxis concretas), las cuales pueden ser gráficas o textuales [22].

Para el lenguaje de modelado de Docker-Compose se han definido dos sintaxis concretas: (1) una gramática textual, que coincide con el lenguaje original de la herramienta Docker-Compose; y (2) una sintaxis gráfica que permite definir modelos, aunque con menos detalle, a través del uso de diferentes tipos de símbolos. A continuación, se describirán ambas sintaxis concretas.

3.2.2.1. Gramática textual

La primera sintaxis concreta desarrollada para el lenguaje de modelado se trata de una gramática textual, que corresponde con la especificación de la herramienta Docker-Compose, que permite definir todos los elementos (contenedores, redes, volúmenes, etc.) que se deseen utilizar en el sistema, así como las propiedades de cada uno de ellos. Como se ha comentado previamente, en esta herramienta la descripción del sistema se define en un fichero de extensión ".yml", siguiendo así la sintaxis y formato de la especificación YAML.

Esta gramática tiene 5 etiquetas principales que separan las distintas secciones que pueden definirse en un fichero de Docker-Compose, sin que sea necesario seguir un orden específico a la hora de definirlas.

- "**version**" (obligatoria): tras la cual debe especificarse, entre comillas, la versión de la herramienta Docker-Compose que se desea utilizar (1, 2, 2.x, 3 o 3.x, siendo "x" cualquier número entero entre 1 y 9), siempre y cuando dicha versión exista y se encuentre soportada.

- "**services**" (obligatoria): comienza la sección en la que se definen todos los contenedores que se desean desplegar, en la que es necesario que al menos exista un contenedor. En esta sección pueden definirse tantos contenedores como sea necesario, para lo cual ha de especificarse el nombre de cada uno de ellos seguido por dos puntos.

  Los contenedores tienen una serie de propiedades cuyos valores pueden ser especificados para configurar el comportamiento de los mismos. Para ello, dentro de cada elemento, debe escribirse la etiqueta correspondiente a la propiedad seguida del valor que se le desee asignar, los cuales deben cumplir cierto formato según la propiedad de la que se trate. Todas estas propiedades, salvo una excepción que se comentará más adelante, son opcionales, y pueden definirse sin seguir ningún orden específico. En la especificación de Docker-Compose

se encuentran detallados todas las propiedades que la herramienta permite definir, aunque para la gramática desarrollada se ha definido únicamente un número limitado de ellas debido a la gran cantidad de propiedades existentes. Concretamente, las propiedades que pueden especificarse son las siguientes, junto con algunos ejemplos:

- *"image"*: permite especificar la imagen a partir de la cual crear el contenedor, que se obtendrá a partir del repositorio de Docker Hub.

```
services:
   frontend:
       image: awesome/webapp
```

- *"build"*: permite especificar la ruta local donde se encuentra la imagen a partir de la cual crear el contenedor.

```
services:
   frontend:
       build: app/data
   backend:
       build: .
```

  Cada contenedor debe tener asignado obligatoriamente un valor a su propiedad *build* o a la propiedad *image*, pero no podrán definirse ambas.

- *"cpu_count"*: define el número de CPUs que se asignan al contenedor. Debe ser un número entero.

```
cpu_count: 3
```

- *"command"*: permite definir una serie de comandos que se ejecutará en cuanto el contenedor se encuentre desplegado. Este conjunto de comandos se debe especificar entre comillas.

- *"container_name"*: permite especificar un nombre personalizado para el

contenedor.

```
services:
   web:
       container_name: my-web-container
```

- *"restart"*: permite especificar la política de reinicio del contenedor. Los posibles valores son: *"no"* (por defecto), el contenedor no se reiniciará en ningún caso; *always*, indica que el contenedor siempre debe reiniciarse hasta que se elimine; *on-failure*, el contenedor se reiniciará si su ejecución termina con un código de error; o *unless-stopped*, se reiniciará cuando su ejecución termine excepto cuando sea detenido o borrado.

- *"init"*: permite indicar, mediante un valor booleano (true o false), si se debe ejecutar un *init process* (PID 1) dentro del contenedor.

- *"read_only"*: permite indicar, mediante un valor booleano (true o false), si el sistema de ficheros del contenedor debe ser de solo lectura.

- *"devices"*: permite definir una lista de dispositivos asignados al contenedor. Cada elemento debe tener el fomato "ruta_host:ruta_contenedor[:permisos_cgroup]", donde tanto *ruta_host* como *ruta_contenedor* deben tener un formato de ruta correcto, mientras que en *permisos_cgroup* puede especificarse una cadena formada por uno o más de los caracteres 'r' (permiso de lectura), 'w' (permiso de escritura) o 'm' (mknod), según los permisos que se deseen otorgar.

```
devices:
   - "/dev/ttyUSB0:/dev/ttyUSB0"
   - "/dev/sda:/dev/xvda:rwm"
```

- *"dns"*: permite especificar una serie de servidores DNS personalizados para la interfaz de red del contenedor. Puede ser un valor único o una lista de elementos, cada uno de los cuales debe tener un formato de dirección

IPv4.

```
dns: 8.8.8.8
```

```
dns:
  - 8.8.8.8
  - 9.9.9.9
```

- *"environment"*: permite definir una serie de variables de entorno para el contenedor, en las que se debe especificar un nombre y, opcionalmente, un valor. El conjunto de variables de entorno pueden definirse con una sintaxis de tipo (*nombre: valor*), o de lista (- *nombre=valor*).

```
environment:
  RACK_ENV: development
  SHOW: "true"
  USER_INPUT:
```

```
environment:
  - RACK_ENV=development
  - SHOW=true
  - USER_INPUT
```

- *"ports"*: permite especificar una lista de puertos del contenedor que serán expuestos, pudiendo mapearlos con puertos del equipo anfitrión. El formato de cada elemento debe ser el siguiente: "[puertos_host:]puertos_contenedor[/protocolo]". Donde en "puertos_contenedor" debe indicarse el puerto o rango de puertos (puerto_inicial - puerto_final); en "puertos_host" (opcional), debe tener el formato "[direccion_ip:]puertos", donde, puede especificarse un puerto o rango de puertos en *puertos* y, opcionalmente, la dirección IP del equipo anfitrión en *direccion_ip*, que debe tener un formato correcto de dirección

IPv4; y por último en "protocolo" puede especificarse, opcionalmente, el protocolo a utilizar.

```
ports:
   - "3000"
   - "3000-3005"
   - "8000:8000"
   - "9090-9091:8080-8081"
   - "49100:22"
   - "127.0.0.1:8001:8001"
   - "127.0.0.1:5000-5010:5000-5010"
   - "6060:6060/udp"
```

- *"links"*: permite definir una serie de enlaces de red que conectan al contenedor con alguno del resto de servicios definidos. El conjunto de enlaces de define como una lista de elementos, cada uno de los cuales debe seguir el siguiente formato: "service[:alias]". En *service* debe especificarse el nombre del contendor que al que se desea conectar mediante el enlace, el cual tiene que estar definido en la sección de los servicios de la aplicación. No podrán definirse distintos enlaces que conecten con el mismo servicio. Opcionalmente, pueden especificarse, un alias para el enlace en *alias*.

```
services:
   web:
       links:
           - api
           - db:database
           - redis
```

- *"depends_on"*: permite definir una serie de dependencias de inicio y cierre entre el contenedor y otros de los servicios definidos. Estas dependencias pueden escribirse simplemente como una lista en la que, en cada elemento,

se especifica el nombre de uno de los servicios de los que depende el contenedor. No podrán definirse distintas dependencias que especifiquen el mismo servicio.

```
services:
   web:
       build: .
       depends_on:
           - db
           - redis
   redis:
       image: redis
   db:
       image: postgres
```

Las dependencias pueden definirse también con otra sintaxis que permiten especificar una condición que debe cumplirse para satisfacerla. Para cada dependencia, debe definirse un elemento con el nombre del servicio del que dependerá el contenedor seguido de dos puntos y dentro del elemento puede especificarse propiedad “condition”, a la que puede asignarse uno de los siguientes valores: *service_started* (por defecto), indica que el contenedor debe iniciarse una vez se haya iniciado el servicio del que depende; *service_healthy*, el contenedor debe iniciarse una vez el servicio del que depende se encuentre desplegado y en estado “sano” (como se indique en la propiedas *healthcheck*); o *service_completed_successfully*, que indica que debe iniciarse una vez se haya completado correctamente la ejecución del servicio del que depende.

```
services:
   web:
       build: .
       depends_on:
```

```
            db:
                condition: service_healthy
            redis:
                condition: service_started
    redis:
        image: redis
    db:
        image: postgres
```

- *"networks"*: permite definir una serie de redes a las que el contenedor debe estar conectado, mediante una lista en la que en cada elemento se especifique el nombre de una de las redes a las que se desea conectar el contenedor. Estas redes deben encontrarse definidas en la sección "networks", que se explicará más adelante.

```
services:
    frontend:
        image: awesome/webapp
        networks:
            - front-tier
            - back-tier
networks:
    front-tier:
    back-tier:
```

Además de esta, existe otra sintaxis para especificar una red a la que debe estar conectado cierto contenedor, la cual permite configurar algunos campos adicionales. Utilizando esta sintaxis, en cada elemento del conjunto de redes pueden asignarse valores a las propiedades explicadas a continuación, sin necesidad de seguir un orden concreto, para conseguir la configuración deseada. Además, en esta sintaxis ya no se definen las redes

como una lista de elementos, sino como un mapa.

- *"ipv4_address"*: permite especificar una dirección IPv4 estática para el contenedor.
- *"priority"*: permite especificar una prioridad mediante un número entero que indicará en qué orden debe conectarse el contenedor a la red en cuestión.
- *"aliases"*: permite especificar una lista de nombres alternativos para el contenedor en la red.
- *"link_local_ips"*: permite especificar una lista de direcciones IPv4 de encalce-local.

```
services:
   some-service:
       networks:
           front-tier:
           ipv4_address: 172.32.238.10
               aliases:
                   - alias1
                   - alias2
           back-tier:
               link_local_ips:
                   - 57.123.22.11
                   - 57.123.22.13
               priority: 100
networks:
   front-tier:
   back-tier:
```

- *"volumes"*: permite definir una serie de volúmenes que deben

ser accesibles desde el contenedor, a través de una lista en la que cada elemento debe seguir el siguiente formato: "volumen:ruta_contenedor[:modo_acceso]". Donde en "volumen" debe especificarse el nombre del volumen que debe ser accesible desde el contendor, el cual debe estar definido en la sección "volumes", descrita más adelante; en "ruta_contenedor" debe especificarse la ruta del sistema de ficheros del contenedor en la que se desea montar el volumen, por lo que debe tener un formato de ruta correcto; y en "modo_acceso" se especificará, opcionalmente, el modo de acceso que puede ser rw (lectura y escritura) por defecto; ro (solo lectura); z (compartido) o Z (no compartido, privado).

```
services:
   backend:
       image: awesome/backend
       volumes:
           - db-data:/database
           - metrics:/etc/data:rw
volumes:
   db-data:
   metrics:
```

Al igual que con las redes a las que debe conectarse una red, hay otra sintaxis para especificar que un volumen debe ser accesible desde cierto contenedor permitiendo editar algunos campos adicionales. Utilizándola, en cada elemento del listado de volúmenes pueden asignarse valores a las siguientes propiedades, sin necesidad de seguir un orden concreto, para conseguir la configuración deseada:

- *"source"* (obligatorio): permite especificar el nombre del volumen que debe ser accesible desde el contendor. Es necesario que el volumen

especificado se encuentre definido en la sección “volumes”.

- *“target”* (obligatorio): permite especificar la ruta del sistema de ficheros del contenedor en la que se desea montar el volumen, y debe tener un formato de ruta correcto.
- *“type”*: tipo de montura. Debido a cómo se ha implementado la gramática, el único tipo permitido es el de “volume”.
- *“read_only”*: permite especificar, mediante un valor booleano, si el volumen será de tipo sólo lectura o no.
- *“volume”*: permite, dentro de esta propiedad, configurar opciones adicionales si se trata de una montura de tipo volumen (el único tipo permitido en este caso). Para este tipo, la única propiedad que puede definirse dentro del elemento es la de “nocopy”, a la que se le debe asignar un valor booleano.

```
services:
   backend:
       image: awesome/backend
       volumes:
           - type: volume
             source: db-data
             target: /data
             volume:
               nocopy: true
           - type: volume
             source: metrics
             target: /var/run/postgres/postgres.sock
             read_only: true
volumes:
   db-data:
```

```
   metrics:
```

- *"configs"*: permite definir una serie de configuraciones asociadas al contenedor que permiten adaptar su comportamiento mediante una lista en la que en cada elemento se especifique el nombre de una de las configuraciones que se desean asociar al contenedor. Estas configuraciones deben encontrarse definidas en la sección "configs", que se explicará más adelante.

```
services:
   redis:
       image: redis:latest
       configs:
           - my_config
           - my_other_config
configs:
   my_config:
       file: ./my_config.txt
   my_other_config:
       external: true
```

De manera similar a los volúmenes asociados a un contenedor, existe otra sintaxis para especificar las configuraciones del contenedor, que permite asignar valores a algunas propiedades que la sintaxis anterior no permite configurar. Para ello, en cada elemento del listado pueden especificarse y asignar valores a las siguientes propiedades:

  - *"source"* (obligatorio): permite especificar el nombre de la configuracion que desea asociarse al contenedor, la cual debe encontrarse definida en la sección "configs".
  - *"target"*: permite especificar la ruta y nombre del fichero de

configuración que se montará en el sistema de ficheros del contenedor, el cual debe tener un formato de ruta correcto.

- *"uid"*: permite especificar el *UID* (*user identifier*) dueño del archivo de configuración y se le debe asignar un valor numérico entre comillas dobles.
- *"gid"*: permite especificar el *GID* (*group identifier*) dueño del archivo de configuración y se le debe asignar un valor numérico entre comillas dobles, al igual que el anterior.
- *"mode"*: permite especificar, mediante un número entero, los permisos para el archivo de configuración que será montado.

```
services:
   redis:
       image: redis:latest
       configs:
           - source: my_config
             target: /redis_config
             uid: "103"
             gid: "103"
             mode: 0440
configs:
   my_config:
       external: true
```

- *"secrets"*: permite definir una serie de secretos asociados al contenedor que permiten adaptar su comportamiento. Debido a que los secretos funcionan de la misma manera que las configuraciones y tienen las mismas propiedades, la sintaxis para especificar los secretos asociados a un contenedor es la misma que la que se usa para listar las configuraciones que utiliza un contenedor. La primera forma de especificar los secretos

asociados a un contenedor es, simplemente, mediante un listado en el que se especifica, en cada elemento, el nombre de uno de las secretos que se desean asociar, los cuales deben encontrarse definidos en la sección "secrets".

```
services:
   frontend:
       image: awesome/webapp
       secrets:
           - server-certificate
secrets:
   server-certificate:
       file: ./server.cert
```

De la misma manera que con las configuraciones de un contenedor, los secretos pueden expresarse mediante una sintaxis más completa que permite asignar valores a ciertas propiedades adicionales. Estas propiedades son las mismas que pueden definirse con la sintaxis de las configuraciones asociadas a un contenedor, por lo que no se van a describir de nuevo.

```
services:
   frontend:
       image: awesome/webapp
       secrets:
           - source: server-certificate
             target: server.cert
             uid: "103"
             gid: "103"
             mode: 0440
secrets:
   server-certificate:
```

```
        external: true
```

- "**networks**": comienza la sección en la cual pueden definirse las redes que se deseen utilizar desde los distintos contenedores. De la misma manera que en la sección de los servicios, en esta pueden definirse tantas redes como sean necesarias, para lo cual debe especificarse el nombre de cada una de ellas seguida de dos puntos.

```
services:
   frontend:
       image: awesome/webapp
       networks:
           - front-tier
           - back-tier
networks:
   front-tier:
   back-tier:
```

Aunque especificar los nombres de las redes de esta forma es suficiente para definirlas, pueden asignarse valores, opcionalmente, a una serie de propiedades de cada red para configurar el comportamiento de estas. De igual manera que con los servicios, dentro de cada elemento que define una red puede especificarse el nombre de cualquiera de las propiedades posibles, seguido de dos puntos, y, tras este, el valor que se le desee asignar. La definición de las propiedades no debe seguir ningún orden concreto, y todas las que pueden definirse con las siguientes:

- *"driver"*: permite especificar el controlador (*driver*) que debe ser utilizado para esta red. Algunos valores son "overlay", "default" o "bridge", entre otros.

```
driver: overlay
```

- *"attachable"*: permite especificar, mediante un valor booleano, que contenedores independientes puedan conectarse a ella
- *"enable_ipv6"*: permite especificar, mediante un valor booleano, si se permite utilizar direcciones IPv6 en la red.
- *"internal"*: permite especificar, mediante un valor booleano, si la red debe estar aislada externamente.
- *"external"*: permite especificar, mediante un valor booleano, si el ciclo de vida de la red debe mantenerse fuera del ciclo del sistema.
- *"name"*: permite especificar un nombre personalizado para la red.

```
networks:
   network1:
       name: my-app-net
```

- *"labels"*: permite definir una serie de etiquetas para la red. Para cada una se especifica un nombre y, opcionalmente, un valor. Dentro de la propiedad, debe especficarse el nombre de cada etiqueta, seguida de dos puntos, y el valor de la misma.

```
labels:
   com.example.description: "Financial transaction network"
   com.example.department: "Finance"
   com.example.label-with-empty-value: ""
```

- *"driver_opts"*: permite especificar una lista de opciones del controlador definidas como pares clave-valor. Dentro de la propiedad, debe especficarse la clave de cada opción, seguida de dos puntos, y el valor de la misma.

```
driver_opts:
   foo: "bar"
   baz: 1
```

- *"ipam"*: permite especificar una configuración IPAM personalizada para la red. Dentro del elemento se pueden asignar valores a una serie de propiedades para configurar la red como se desee, donde ninguna de ellas es obligatoria, aunque se debe especificar al menos una de ellas si se define la propiedad *ipam*. Las propiedades que pueden editarse son las siguientes:
  - *"driver"*: permite definir un controlador IPAM personalizado para utilizar en lugar del predeterminado.
  - *"config"*: permite especificar una lista de elementos de configuración. Dentro de cada uno de ellos, pueden asignarse valores a las siguientes propiedades:
    - "subnet": permite especificar la dirección de la subred, que debe indicarse en formato *CIDR* (*direccion_ipv4/prefijo*).
    - "ip_range": permite especificar un rango de direcciones IP para asignar las direcciones de los contenedores. También debe indicarse en formato *CIDR*.
    - "gateway": permite especificar la puerta de enlace para la subred, que debe tener un formato correcto de dirección IPv4.
    - "aux_addresses": permite especificar una lista de direcciones IPv4 auxiliares que utilizará el controlador de red, que se indicarán como pares clave-valor, especificando un nombre como clave de cada dirección (*nombre: direccion*).
  - *"options"*: permite especificar una lista de opciones del controlador definidas como pares clave-valor.

```
ipam:
   driver: default
   config:
       - subnet: 172.28.0.0/8
```

```
        ip_range: 172.28.5.0/24
        gateway: 172.28.5.254
        aux_addresses:
          host1: 172.28.1.3
          host2: 172.28.1.4
          host3: 172.28.1.5
    options:
        foo: bar
        baz: "0"
```

- "**volumes**": indica el inicio de la sección en la que pueden definirse todos los volúmenes que se van a utilizar en el sistema.

```
services:
   backend:
       image: awesome/database
       volumes:
           - db-data:/etc/data
volumes:
   db-data:
   metrics:
```

Los volúmenes se definen de la misma manera que las redes ya que debe especificarse el nombre de cada uno, seguido de dos puntos, y dentro del elemento podrán asignarse diferentes valores, de manera opcional y sin necesidad de seguir ningún orden específico, a las siguientes propiedades para configurar el volumen como se desee:

- *"driver"*: permite especificar el controlador que debe utilizarse para el volumen.

```
driver: foobar
```

- *"external"*: permite especificar, mediante un valor booleano, si se trata o no de un volumen externo. En caso de serlo, significa que ya existe en el sistema, y su ciclo de vida es independiente de la aplicación o contenedor que lo use.
- *"name"*: permite especificar un nombre personalizado para el volumen.

```
volumes:
   data:
       name: "my-app-data"
```

- *"labels"*: permite definir una serie de etiquetas para el volumen, especificando un nombre y, opcionalmente, un valor

```
labels:
   com.example.description: "Database volume"
   com.example.department: "IT/Ops"
   com.example.label-with-empty-value: ""
```

- *"driver_opts"*: permite especificar una lista de opciones del controlador definidas como pares clave-valor.

```
volumes:
   example:
       driver_opts:
           foo: "nfs"
           o: "addr=10.40.0.199,nolock,soft,rw"
           device: ":/docker/example"
```

- "**configs**": permite definir las configuraciones para adaptar el comportamiento de los contenedores de la aplicación. Las configuraciones se definen de la misma manera que el resto de elementos (contenedores, redes, volúmenes, etc.) en sus respectivas secciones, el nombre de cada una de ellas debe ser especificado seguido de dos puntos dentro de la sección *configs*.

```
services:
   frontend:
       image: awesome/webapp
       configs:
           - httpd-config
configs:
   httpd-config:
       external: true
```

Al igual que con el resto de elementos, dentro de las configuraciones definidas pueden editarse una serie de propiedades de manera opcional y sin necesidad de seguir ningún orden concreto a la hora de especificarlas. Las propiedades a las que se les pueden asignar valores dentro de las configuraciones son las siguientes:

- *"file"*: permite especificar la ruta del fichero a partir del cual debe ser creada la configuración, por lo que su formato debe ser el de una ruta. Las configuraciones pueden ser externas o ser creadas a partir de un archivo, por lo que no puede especificarse una ruta en esta propiedad e indicar a su vez que se trata de una configuración externa (siguiente propiedad).

  ```
  configs:
     http_config:
         file: ./httpd.conf
  ```

- *"external"*: permite especificar, mediante un valor booleano, si la configuración es externa o no. En caso de asignar el valor "true" a la propiedad, se está indicando que se trata de una configuración externa, es decir, que ya está creada, por lo que la implementación de Docker-Compose no intentará crearla. En este caso, además, no puede especificarse ningún valor para la propiedad *file*.

  ```
  configs:
  ```

```
   http_config:
       external: true
```

- *"name"*: permite especificar el nombre de la configuración existente en la plataforma. El valor de esta propiedad solo puede ser especificado si se ha indicado que la configuración es externa.

```
configs:
   http_config:
       external: true
       name: "${HTTP_CONFIG_KEY}"
```

- "**secrets**": en esta se definen los secretos que, como ya se ha comentado, funcionan de manera muy similar a las configuraciones, con la diferencia de que los secretos están más centrados en los datos sensibles. Por ello, estos no solo se definen de la misma manera que las configuraciones, sino que, además, las propiedades a las que se les pueden asignar valores dentro del elemento son las mismas, por lo que no se vuelven a describir.

```
services:
   frontend:
       image: awesome/webapp
       secrets:
           - server-certificate
secrets:
   server-certificate:
       external: true
```

3.2.2.2. Editor gráfico

La segunda de las sintaxis concretas se trata de una notación gráfica que permite definir modelos para Docker-Compose contruyendo diagramas con una serie de símbolos

gráficos. Estos diagramas no permiten definir modelos con tanto detalle como la gramática textual explicada en la sección anterior, por lo que sería necesario combinar esta notación con la sintaxis concreta textual para conseguir un modelo completo.

Tanto la notación gráfica como el editor que permite crear diagramas de acuerdo a la misma se han desarrollado utilizando *Sirius*. Estos diagramas están formados por nodos y por aristas o relaciones que interconectan dos nodos distintos.

La notación permite definir cualquiera de los principales objetos Docker, los cuales son representados por nodos con forma de caja. Cada nodo mostrará un icono diferente dependiendo del tipo de objeto del que se trate y el nombre que se le haya asignado al mismo (una de las propiedades que pueden editarse en esta sintaxis).

En la Figura 4.10 se muestra el nodo que representaría a un contenedor definido con el nombre "web". En el interior del nodo se muestra, además del nombre, el icono de la herramienta Docker, que ha sido asignado para representar a los servicios definidos. De igual manera, una red definida con el nombre "net", quedaría representada en el diagrama por un nodo como el que se muestra en la Figura 3.3. La figura 3.4 muestra como se representaría un volumen definido con el nombre "db", en este caso, con un icono de una carpeta dentro del nodo. Una configuración definida con el nombre "http" se representaría con un nodo como el mostrado en la Figura 3.5, que incluye el icono correspondiente a las configuraciones (icono de engranaje). Por último, la Figura 3.6 muestra la representación de un secreto definido con el nombre "key", usando un icono de un candado en este caso.

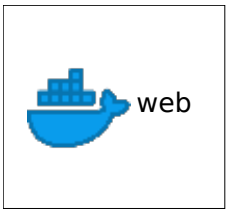

Figura 3.2: Nodo servicio

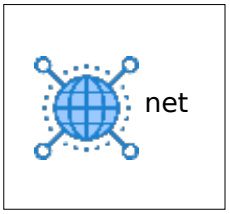

Figura 3.3: Nodo red

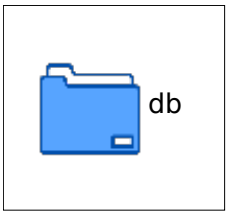

Figura 3.4: Nodo volumen

Figura 3.5: Nodo configuración

Figura 3.6: Nodo secreto

Además de los cinco tipos de nodos que se han comentado, pueden definirse relaciones entre algunos de ellos. Todas ellas son representadas por flechas que van desde un nodo a otro, cuyos colores y formas varían según el tipo de relación que representen.

Docker-Compose permite definir dos tipos de relaciones entre servicios, de enlace y de dependencia. En los diagramas de notación gráfica, estas dependencias se representan con una flecha azul formada por una línea continua, la cual va desde un nodo contenedor hasta otro del que depende. En la Figura 3.7 se muestra la representación de una relación que define que un servicio “redis” depende de otro llamado “web”.

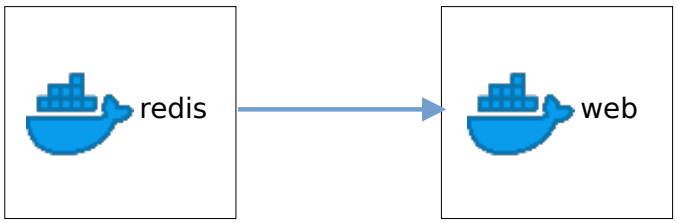

Figura 3.7: Relación de dependencia entre dos nodos de servicios

Además de la relación de dependencia, pueden definirse enlaces de red para conectar

dos servicios entre sí. Esta relación se representa mediante una flecha discontinua azul que va desde un contenedor a otro con el que se desea conectar. Un enlace de red definido entre un contenedor "redis" y otro "web" se representaría con una flecha como la de la Figura 3.8.

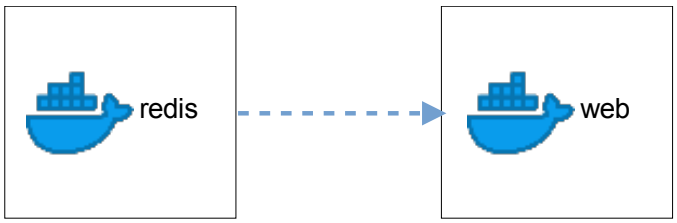

Figura 3.8: Relación de enlace de red entre dos nodos de servicios

Los servicios también pueden conectarse con cualquier otro tipo de objeto Docker. Las relaciones definidas entre un servicio y una red especifican que el servicio estará conectado a esa red. Estas se representan mediante una flecha verde discontinua que va desde el nodo contenedor hasta uno de red. La Figura 3.9 muestra la representación de una relación entre un servicio "web" y una red "net".

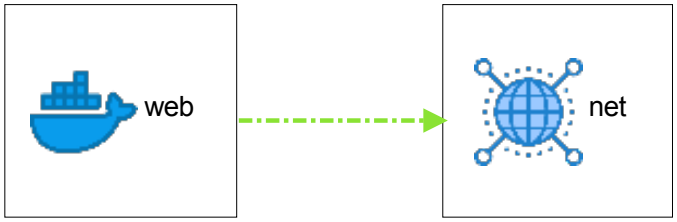

Figura 3.9: Relación entre un nodos de servicio y un nodo de red

Por otro lado, las relaciones entre un servicio y un volumen especifican que el volumen debe ser accesible desde el servicio en cuestión. Estas se representan con una flecha continua lila que va desde un nodo servicio hasta otro de tipo volumen. Una relación que defina que un volumen llamado "db" debe ser accesible desde un contenedor "web" se representaría como se muestra en la Figura 3.10.

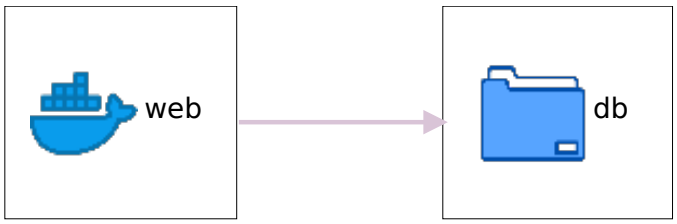

Figura 3.10: Relación entre un nodos de servicio y un nodo de volumen

Por último, tanto las configuraciones como los secretos permiten adaptar el comportamiento de ciertos servicios. Para asociar cualquier contenedor a una configuración o a un secreto debe definirse una relación que se representa mediante una flecha amarilla que va desde el nodo servicio hasta el nodo de configuración o secreto. Las flechas que conectan servicios con configuraciones son discontinuas, mientras que las que los asocian con secretos son punteadas. En la Figura 3.11 se muestra como se representaría una asociación entre un servicio "web" y una configuración "http". Una relación definida entre un servicio "web" y un secreto "key" se representaría como se muestra en la Figura 3.12.

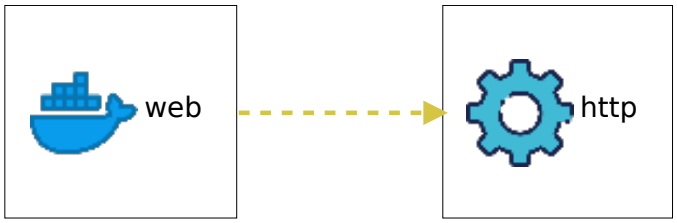

Figura 3.11: Relación entre un nodo de servicio y un nodo de configuración

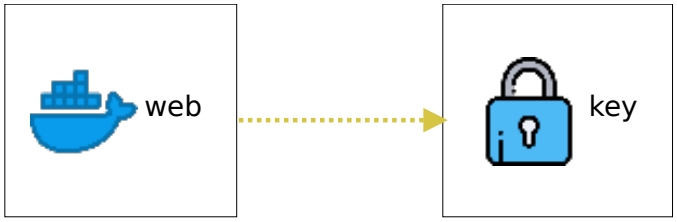

Figura 3.12: Relación entre un nodo de servicio y un nodo de secreto

Capítulo 4

Implementación

En este apartado se va a describir el proceso de implementación del lenguaje de modelado desarrollado para el proyecto. Esto incluye el desarrollo tanto del metamodelo asociado a la sintaxis abstracta del lenguaje como de la gramática textual y el editor gráfico.

4.1. Metamodelo

Eclipse Modeling Framework (*EMF* [17]) es un framework de modelado basado en *Eclipse* que permite construir herramientas y aplicaciones basadas en modelos.

EMF permite definir sintaxis abstractas utilizando el estándar *XMI* (*XML Metadata Interchange*). Estas se almacenan en archivos de texto con extensión *.ecore*, que tendrán un formato XMI. El fichero *ecore* permite utilizar los siguientes elementos para la definición de un metamodelo [30]:

- *EClass*: representa las metaclases del metamodelo. Se identifican con un nombre y se les pueden añadir, opcionalmente, atributos y referencias. Pueden, además, referenciar supertipos.
- *EAttribute*: representa meta-atributos, atributos de las metaclases definidas.

Estos tendrán un nombre y un tipo.

- *EReference*: representa asociaciones entre dos metaclases. Estas tienen un nombre y un tipo, que corresponde con la metaclase destino de la asociación. Contiene un atributo booleano llamado *containment* que indica si se trata de una relación de composición o de una referencia. Los atributos *lowerBound* y *upperBound* especifican la multiplicidad de la asociación.
- *EDataType*: representa los tipo de los atributos que son tipos primitivos.

EMF ofrece una manera de definir y editar la sintaxis abstracta gráficamente, a través de un metamodelo. Un *metamodelo* es una especificación formal de la sintaxis abstracta de un lenguaje. Se trata de una forma gráfica de expresar el conjunto de combinaciones lógicas de los elementos del lenguaje. Un metamodelo es un modelo que representa modelos, es decir, define lo que puede ser expresado en un modelo válido de cierto lenguaje de modelado. Cada modelo del lenguaje puede ser considerado como una instancia del metamodelo.

La opción "Initialize Ecore Diagram" del menú contextual de un archivo ecore permite generar una representación gráfica del fichero. De esta forma, EMF ofrece la posibilidad de editar un metamodelo como un diagrama, definiendo todos los elementos que lo forman.

Utilizando los conceptos de orientación a objetos, los conceptos del lenguaje se modelan como clases en el metamodelo (metaclases). Los atributos de una metaclase representan las propiedades del concepto correspondiente del lenguaje. Las relaciones entre conceptos se modelan como referencias (asociaciones dirigidas) entre metaclases. En caso de tratarse de una relación de agregación parte-de (dependencia existencial), se debe modelar como una composición. Una de las metaclases debe representar el elemento raíz del metamodelo, a partir del cual se podrá navegar a lo largo de todo el metamodelo.

Así, se ha creado el fichero "DockerCompose.ecore" y generado su representación

gráfica, en la que se modelan todos los conceptos de la sintaxis abstracta. La metaclase *DockerCompose* representa el elemento raíz del metamodelo definido. A partir de este, puede navegarse a lo largo de todo el metamodelo. Es se debe a que, los conceptos principales de la gramática (los objetos Docker que pueden definirse utilizando Docker-Compose) parten de este mediante relaciones de agregación. Las metaclases *Service*, *Network*, *Volume*, *Config* y *Secret* representan a los diferentes conceptos del lenguaje (servicios, redes, volúmenes, configuraciones y secretos).

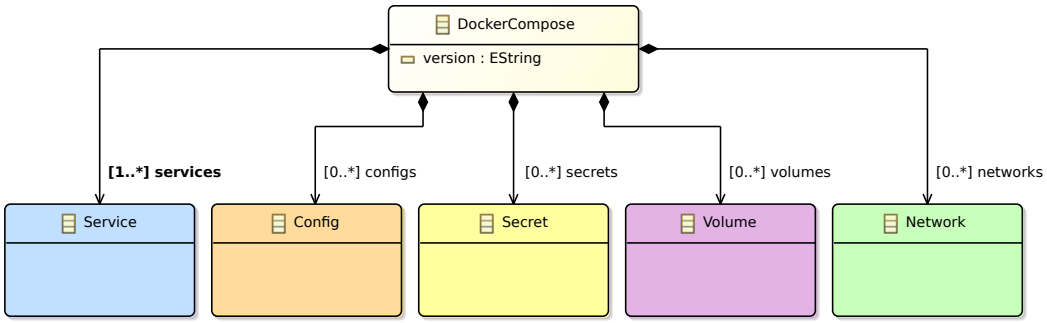

Figura 4.1: Conceptos principales de la gramática en el metamodelo

Todos los conceptos cuentan con distintas propiedades que pueden editarse en el lenguaje de Docker-Compose. Las propiedades que guardan valores simples (como cadenas, números enteros o valores booleanos) se modelan como atributos de la metaclase correspondiente. Por otro lado, en caso de tratarse de propiedades complejas con más de un valor, estas se representan como nuevas metaclases que parten del elemento (relación de composición). En estas nuevas metaclases se defininen los atributos y composiciones necesarios para guardar la información de la propiedad. Lo mismo ocurre cuando una propiedad almacena una lista de valores o elementos. En esos casos, los elementos se modelan como nuevas metaclases que parten de la metaclase que representa al objeto Docker en cuestión. La cardinalidad de las composiciones modeladas indica si la propiedad se trata de una lista de valores o si, por el contrario, almacena un único elemento complejo.

La metaclase *Service* representa a los contenedores/servicios que forman la aplicación. El elemento raíz (*DockerCompose*) se compone de uno o más servicios, por lo que es

obligatorio que un modelo válido contenga, al menos, un contenedor. Los atributos de la metaclase representan propiedades del servicio con valores simples, como cadenas (*name*, *build*, *command*, *container_name*, *image* y *restart*), números enteros (*cpu_count*) y valores booleanos (*init*, *read_only*).

Otras propiedades de los servicios son listados de valores o elementos complejos (con más de un valor). Cada una de estas se modelan como nuevas metaclases (*EnvironmentVariable*, *Port*, *Device*, *DNS*), las cuales parten de *Service* con relaciones de composición. Al ser propiedades opcionales del servicio, este puede componerse de cero o más elementos de estos tipos.

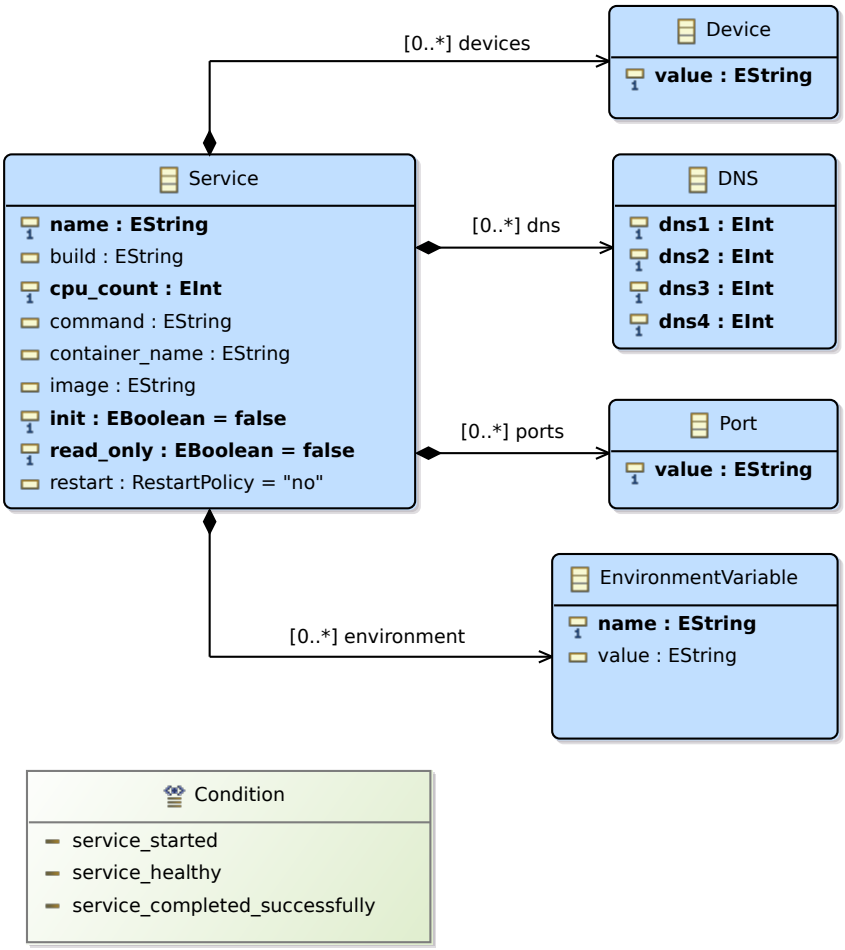

Figura 4.2: Concepto servicio con todas sus propiedades en el metamodelo

Por último, los servicios pueden estar relacionados con otros objetos Docker (redes,

volúmenes, configuraciones y secretos) o, incluso, tener enlaces o dependencias con otros servicios. Estas relaciones, además, pueden tener algunas propiedades específicas, por lo que deben modelarse como nuevas metaclases. Todas ellas componen a la metaclase *Service*, y cada una tiene una asociación con la metaclase que relaciona con el servicio, además de una serie de atributos que representan las propiedades de cada relación. Las relaciones de enlaces y dependencias entre servicios se modelan con las metaclases *Link* y *Dependency*, respectivamente. Por otro lado, las metaclases *NetworkConnector*, *VolumeConnector*, *ConfigConnector* y *SecretConnector* representan las relaciones entre servicios y, respectivamente, redes, volúmenes, configuraciones y secretos.

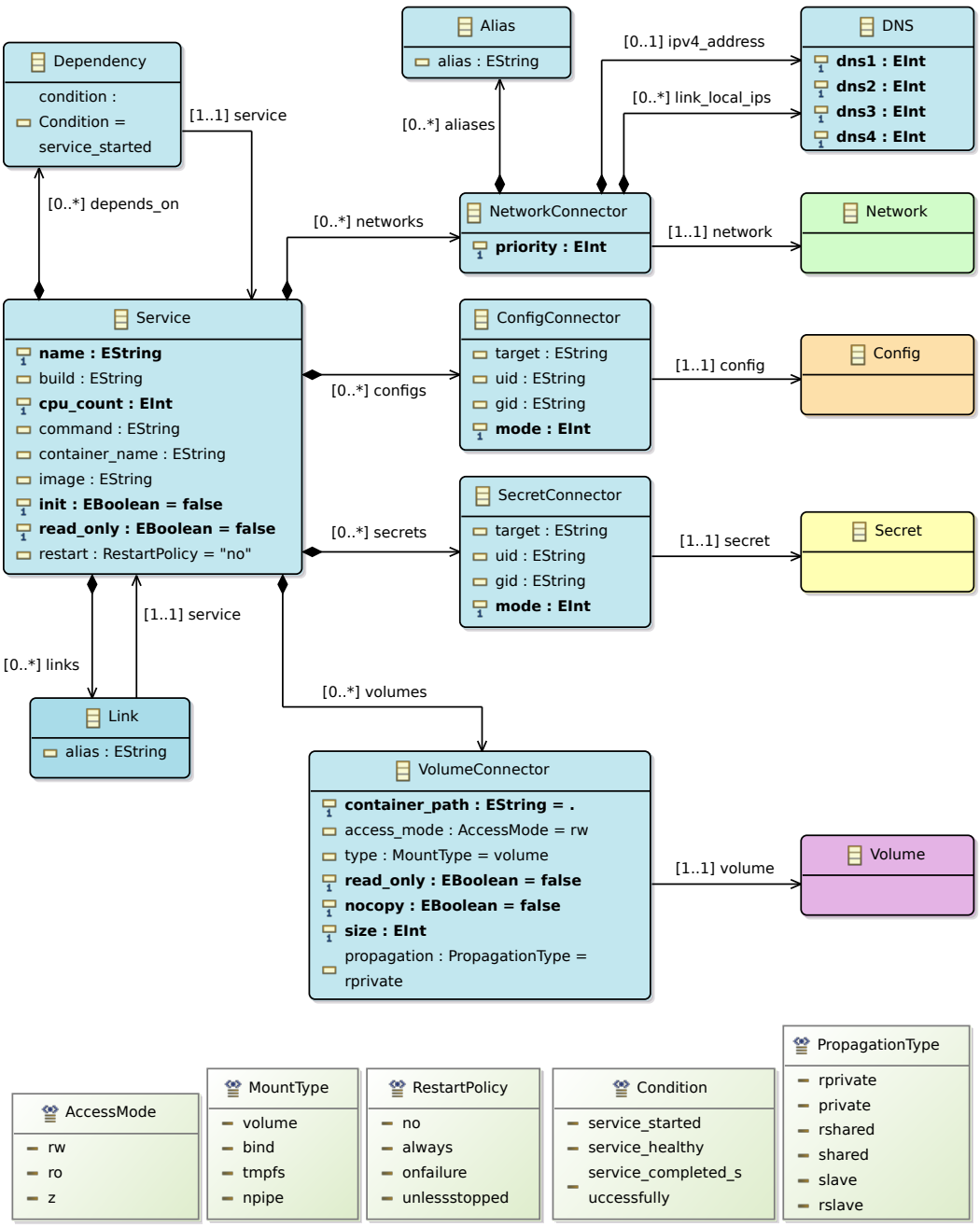

Figura 4.3: Concepto servicio con todas sus asociaciones en el metamodelo

El resto de objetos Docker se modelan de la misma manera que los servicios, con metaclases que parten del elemento raíz, de tal manera que la metaclase *DockerCompose* estará compuesta de cero o varios elementos estos tipos.

Las configuraciones se modelan con la metaclase *Config*, con los atributos *name*, *file* y

config_name de tipo cadena y el atributo booleano *external*. La metaclase *Secret*, que representa a los secretos, es muy similar. Esta almacena los atributos de tipo cadena *name*, *file* y *secret_name* y el atributo booleano *external*.

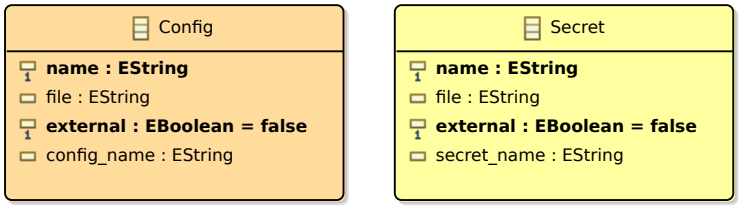

Figura 4.4: Conceptos configuración y secreto con sus propiedades en el metamodelo

Los volúmenes se modelan con la metaclase *Volume*, que tiene los atributos *name*, *driver* y *volume_name* de tipo cadena y el atributo booleano *external*. Además, los volúmenes tienen dos propiedades que guardan un listado de elementos, por lo que se han modelado como dos metaclases diferentes (*VolumeLabel* y *VolumeDriverOpt*). Así, *Volume* se comopone de cero a varios elementos de estas metaclases.

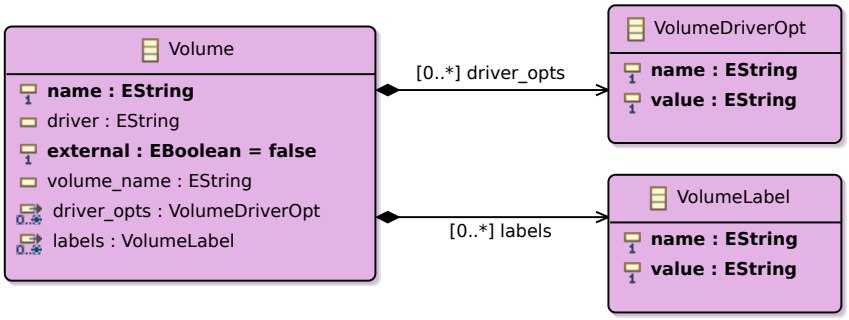

Figura 4.5: Concepto volumen con todas sus propiedades en el metamodelo

De la misma manera, la metaclase *Network* representa las redes de la aplicación, con los atributos booleanos *attachable*, *enable_ipv6*, *internal* y *external* y los atributos *name*, *network_name* y *driver* de tipo cadena. Además, tiene dos propiedades que almacenan listas de elementos que se modelan con las metaclases *NetworkLabel* y *NetworkDriverOpt* (modeladas de la misma manera que las metaclases *VolumeLabel* y *VolumeDriverOpt*). Por último, *Network* tiene una propiedad compleja que se modela

con la metaclase *IPAM*. Esta, a su vez, tiene una serie de propiedades que se modelan con el atributo *driver* y las metaclases *IPAMOption* e *IPAMConfig*, de las que *IPAM* se compone.

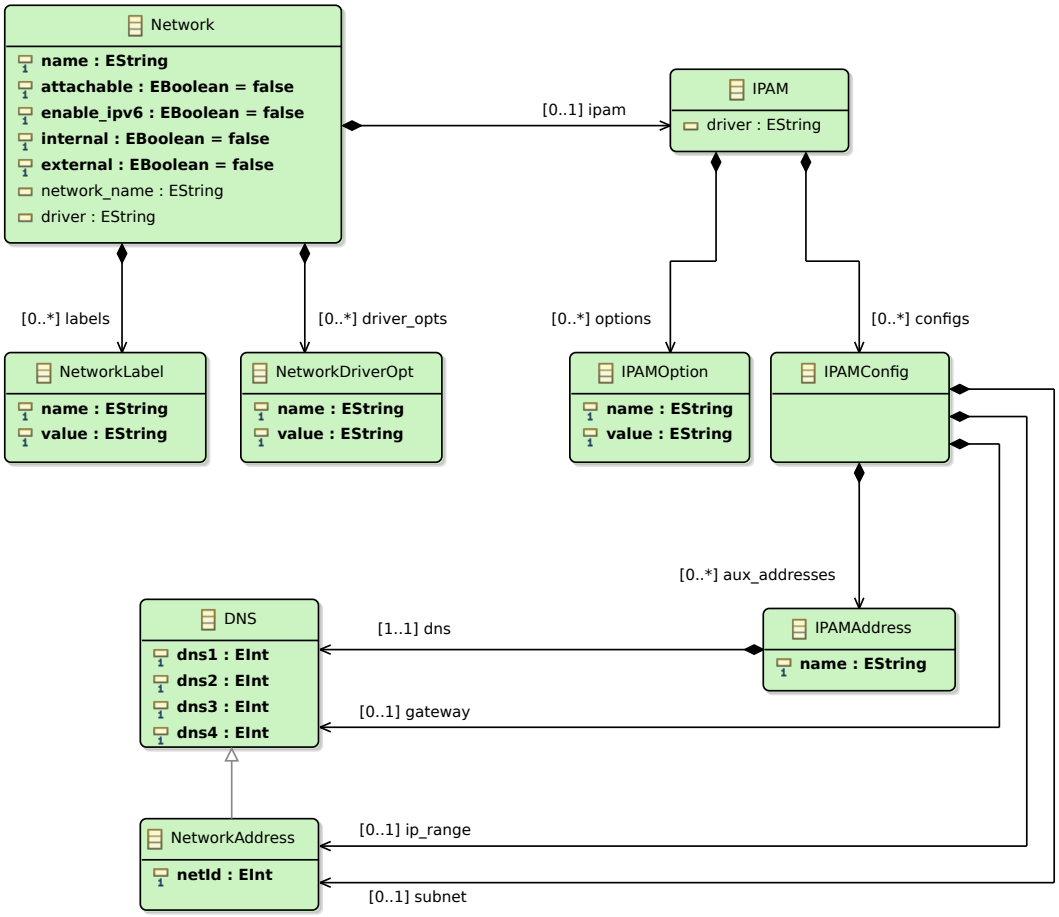

Figura 4.6: Concepto red con todas sus propiedades en el metamodelo

Ecore permite, además, definir los atributos como identificadores, así como el número mínimo y máximo de instancias de cada atributo. Así, todos los atributos *name* de las metaclases que representan a los objetos Docker principales se declaran como identificadores únicos. El resto de propiedades de estos objetos son opcionales, por lo que a los demás atributos de estas metaclases se les especifica un valor mínimo de cero instancias.

La definición de un metamodelo suele no ser suficiente para especificar una sintaxis

abstracta completa. Estos deben acompañarse con una serie de reglas semánticas que determinan cuándo un modelo está bien formato. Estas reglas, conocidas como *well-formedness rules*, permiten expresar restricciones que no pueden expresarse únicamente con la definición de un metamodelo.

Las reglas semánticas de corrección para esta sintaxis abstracta se han definido utilizando *OCL* [23] (*Object Constraints Language*), un lenguaje formal para definir restricciones y consultas sobre modelos UML. OCL ofrece mecanismos para recuperar los valores de un objeto, navegar por un conjunto de objetos relacionados e iterar sobre colecciones de objetos [12].

Para definir estas reglas se utiliza *OCLinEcore* [19], otra de las herramientas con las que puede editarse un fichero ecore, la cual tranforma la definición del metamodelo al formato textual *oclinecore*. Este, además de permitir editar metaclases, atributos y referencias de un metamodelo, es ideal para añadir reglas semánticas (well-formedness rules) a la sintaxis, ya que otros editores no señalan errores sintácticos o semánticos en las relgas ni permiten visualizarlas correctamente [18].

Las sentencias OCL deben estar asignadas a un contexto, que será el punto de partida para evaluar la expresión. Para ello, en el fichero ecore en formato OCLinEcore, las reglas deben especificarse dentro del elemento que define a la metaclase que quiere asignarse como contexto de la regla. Para cada una debe especificarse un nombre precedido por la palabra reservada “invariant”. Tras esto, se define, utilizando el lenguaje OCL, la expresión que debe ser verdadera para que la restricción se cumpla.

Para la metaclase *Service* se han definido las siguientes reglas:

- *image_or_build*: especifica que todo servicio debe definir un valor para la propiedad *build* o la propiedad *image*, pero nunca para ambos.
- *no_self_dependencies*: impide que se definan dependencias de un servicio con sí mismo.
- *no_self_links*: impide que se definan enlaces entre un servicio y sí mismo.

- *different_dependencies*: impide que se duplique una dependencia entre dos servicios.
- *different_links*: impide que se duplique un enlace entre dos servicios.
- *different_networks*: impide que se duplique una conexión entre un servicio y una red.
- *different_volumes*: impide que se duplique una relación entre un servicio y un volumen.
- *different_configs*: impide que se duplique una relación entre un servicio y una configuración.
- *different_secrets*: impide que se duplique una relación entre un servicio y un secreto.
- *different_environment_variables*: impide definir variables de entorno con nombres duplicados para un mismo servicio.

Las siguientes relgas son las que se han definido en el conexto de la metaclase *Network*:

- *different_labels*: impide definir etiquetas con nombres duplicados para una misma red.
- *different_driver_opts*: impide definir opciones del controlador con nombres duplicados para una misma red.

En el contexto de la metaclase *Volume* se han definido las reglas que se mencionan a continuación:

- *different_labels*: impide definir etiquetas con nombres duplicados para un mismo volumen.
- *different_driver_opts*: impide definir opciones del controlador con nombres duplicados para un mismo volumen.

Las siguientes reglas se han definido dentro del contexto de la metaclase *Config*:

- *file_or_external*: especifica que las configuraciones deben tener un valor asignado a la propiedad *file* o bien un valor "true" asignado a la propiedad *external*.
- *external_name*: indica que la propiedad *config_name* de una configuración solo puede definirse en caso de que la configuración sea externa, es decir, que tenga un valor "true" asignado a la propiedad *external*.

En el contexto de la metaclase *Secret* se han definido las reglas listadas a continuación:

- *file_or_external*: especifica que los secretos deben tener un valor asignado a la propiedad *file* o bien un valor "true" asignado a la propiedad *external*.
- *external_name*: indica que la propiedad *secret_name* de un secreto solo puede definirse en caso de que el secreto sea externo, es decir, que tenga un valor "true" asignado a la propiedad *external*.

La siguiente regla ha sido definida en el contexto de la metaclase *DNS*:

- *correct_ip_format*: restringe el rango de valores que puede tomar cada atributo de la clase *DNS*, que representa a las direcciones IPv4 en el metamodelo, para asegurarse de que se cumple correctamente el formato de IPv4.

La metaclase *IPAM* se ha especificado como contexto de las siguientes reglas:

- *any_property*: asegura que todos los elementos de tipo IPAM tengan especifiquen especificados para, al menos, una de sus propiedades, ya que todas son opcionales.
- *different_options*: impide definir opciones con nombres duplicados para una misma configuración IPAM.

La siguiente regla ha sido definida en el contexto de la metaclase *NetworkConnector*:

- *different_aliases*: impide definir aliases con nombres duplicados para un mismo elemento de tipo *NetworkConnector*.

Por último, las reglas explicadas a continuación se han definido en el contexto de la

metaclase *IPAMConfig*:

- *any_property*: asegura que todos los elementos de tipo IPAMConfig tengan valores especificados para, al menos, una de sus propiedades, ya que todas son opcionales.
- *different_addresses*: impide definir direcciones IPv4 auxiliares con nombres duplicados para un mismo elemento IPAMConfig.

El fichero "DockerCompose.ecore" completo en formato OCLinEcore, en el que se encuentran definidas todas las restricciones OCL mencionadas, se ha incluido en el anexo A.

4.2. Gramática textual

Como se describe en la sección 3.2.2.1, la primera de las sintaxis concretas desarrolladas para el la sintaxis abstracta definida se trata de una gramática textual, que conicide con la sintaxis utilizada en la herramienta Docker-Compose. Esta se ha desarrollado utilizando *Xtext* [21], un framework que permite crear lenguajes de modelado textuales, asociando una representación textual a los conceptos de la sintaxis abstracta.

EMF permite crear una gramática con Xtext para un un metamodelo definido, generando un proyecto Xtext a partir del archivo que define el metamodelo. Este incluirá un fichero de texto con extensión *.xtext*, llamado "DockerCompose.xtext" en este caso, en el cual debe definirse, utilizando el lenguaje *Xtext*, la gramática textual completa. Este se genera con una gramática por defecto generada a partir del metamodelo, la cual puede usarse a modo de plantilla.

En el fichero "DockerCompose.xtext" deben especificarse una serie de reglas que definan la gramática textual. Debe definirse una de las llamadas *parser rules* por cada metaclase del metamodelo, que especifican cómo se define cada tipo de elemento en la

gramática.

La regla *DockerCompose* define como escribir elementos de tipo *DockerCompose* en la gramática. Al tratarse del elemento raíz del metamodelo, esta regla especifica cómo se define un modelo completo. Esta se compone de varios fragmentos separados por el carácter '&', lo que indica que pueden definirse sin seguir un orden concreto. El carácter '?' final de algunos de ellos indica que son opcionales. Cada fragmento comienza por una cadena entre comillas que representa una palabra reservada del lenguaje o literal seguida una asignación de uno de los atributos de *DockerCompose*. La asignación "version=Version" significa que el atributo *version* debe inicializarse usando otra regla llamada *Version*, por lo que para definir este fragmento en un modelo debe escribirse la palabra reservada correspondiente y darle un valor al atributo *version* de la manera en que lo inidica la regla *Version*.

Los demás fragmentos están definidos de la misma manera, aunque con un carácter '+' al final de las asignaciones que indica que pueden hacerse una o varias veces, ya que se tratan de propiedades que almacenan más de un valor o elemento. Por ejemplo, "(services+=Service)+" indica que el atributo *service* de la clase *DockerCompose* se inicializa con uno o más servicios definidos con la regla *Service*.

```
DockerCompose returns DockerCompose:
 (
          ('version:'   version=Version)?
        & ('services:'  (services+=Service)+ )
        & ('volumes:'   (volumes+=Volume)+ )?
        & ('configs:'   (configs+=Config)+ )?
        & ('secrets:'   (secrets+=Secret)+ )?
        & ('networks:'  (networks+=Network)+ )?
 );
```

Cada una de las reglas correspondientes a los conceptos principales del lenguaje se define de la misma manera. Tras especificar un nombre para el elemento seguido

del carácter ':', hay una serie de fragmentos, cada uno describiendo una propiedad del elemento, que pueden escribirse sin seguir un orden concreto. Los atributos se inicializan con las reglas *EString*, *EInt* o *EBoolean*, que definen, respectivamente, como deben especificarse las cadenas, números enteros o valores booleanos. Por otra parte, las composiciones o referencias deben inicializarse utilizando reglas que definan la metaclase correspondiente en cada caso.

Algunas propiedades pueden inicializarse de distinta forma. Para ello, se utiliza el carácter '|', que separa dos elementos indicando que pueden definirse bien uno de ellos o bien el otro. En la regla *Service* que indica como definir servicios, el fragmento "(('-'dns+=DNS)+ | dns+=DNS))?" indica que la propiedad *dns* puede inicializarse especificando varios elementos con la regla *DNS*, cada uno precedido por el carácter '-', o bien con un único elemento definido por la regla *DNS*. También pueden definirse distintas reglas para definir un mismo tipo de elemento de formas diferentes. El fragmento "((networks+=NetworkConnector_long)+ | (networks+=NetworkConnector_short)+))?" indica que la propiedad *networks* de un contenedor puede inicializarse con varias redes definidas según indica la regla *NetworkConnector_long* o bien utilizando la regla *NetworkConnector_short*. De forma similar a la regla *Service* se definen el resto de reglas correspondientes a los conceptos principales (reglas *Network*, *Volume*, *Config* y *Secret*).

```
Service returns Service:
 {Service}
 name=ID ':'
 (
         ('build:'           build=PATH)?
  & ('image:'            image=Image)
       & ('cpu_count:'       cpu_count=EInt)?
       & ('command:'         command=ID)?
       & ('container_name:'  container_name=EString)?
       & ('restart:'         restart=RestartPolicy)?
```

```
        & ('init:'              init=EBoolean)?
        & ('read_only:'         read_only=EBoolean)?

        & ('links:'             ('-'links+=Link)+ )?
        & ('depends_on:'        (depends_on+=Dependency)+)?
        & ('networks:'          ((networks+=NetworkConnector_long)+
                    | (networks+=NetworkConnector_short)+ ))?
        & ('volumes:'           ('-'volumes+=VolumeConnector)+)?
        & ('configs:'           ('-'configs+=ConfigConnector)+)?
        & ('secrets:'           ('-'secrets+=SecretConnector)+)?

        & ('environment:' ((environment+=EnvironmentVariableMap)+
                  | (environment+=EnvironmentVariableList)+ ))?
        & ('devices:'           ('-'devices+=Device)+ )?
        & ('dns:'               (('-'dns+=DNS)+ | dns+=DNS) )?
        & ('ports:'             ('-'ports+=Port)+ )?
 );
```

Las metaclases que representan conectores entre un servicio y algún otro objeto Docker tienen reglas similares. La regla *VolumeConnector*, que define la metaclase *VolumeConnector*, puede especificarse de dos formas distintas, separadas por el carácter '|' en la definición de la regla. Una de ellas permite especificar los valores de más atributos que la otra. Ambas incluyen una asignación a la propiedad *volume*, y debido a que se trata de una referencia a un volumen, debe inicializarse con el identificador (nombre) de uno ya definido. Esto se expresa de la manera "[Volume|EString]". Las demás reglas que representan conectores (*Dependency*, *Link*, *NetworkConnector*, *ConfigConnector* y *SecretConnector*) se definen de manera similar, y todas ellas contienen una referencia a otro elemento definido.

```
VolumeConnector returns VolumeConnector:
 {VolumeConnector}
```

```
((volume=[Volume|ID]':'container_path=PATH
                        (':'access_mode=AccessMode)?) |
(
      ('source:'        volume=[Volume|ID])
    & ('type:'          type=MountType)?
    & ('target:'        container_path=PATH)
    & ('read_only:'     read_only=EBoolean)?
    & ('bind:'
        'propagation:' propagation=PropagationType)?
    & ('volume:'
        'nocopy:'       nocopy=EBoolean)?
    & ('tmpfs:'
        'size:'         size=EInt)?
));
```

El resto de metaclases también necesitan reglas que definan cómo deben describirse sus elementos. Estas serán más o menos complejas dependiendo del número de atributos que tenga la clase en cuestión, y de la sintaxis asociada al elemento, aunque todas más serán simples que las reglas correspondientes a los conceptos principales del lenguaje y a las clases conectoras, similares a la que se muestra a continuación.

```
VolumeLabel returns VolumeLabel:
   {VolumeLabel}
   name=EString':' value=EString;
```

El resto de metaclases necesitan reglas que definan una manera de escribir. También deben crearse reglas para cada enumeración definida en el metamodelo. Estas van precedidas por la palabra reservada "enum", y deben definir la forma en la que se escribe cada uno de sus posibles valores, separados por el carácter '|'.

```
enum RestartPolicy returns RestartPolicy:
   no = '"no"' | always = 'always' | onfailure = 'on-failure'
```

```
                | unlessstopped = 'unless-stopped';
```

Como se puede comprobar en algunas de las reglas mostradas anteriormente, no todos los atributos de tipo cadena se inicializan con la regla *EString*. Algunos necesitan cumplir cierto formato, por lo que deben definirse reglas que especifiquen esos formatos. Los atributos *build* de la clase *Service* y *container_path* de *VolumeConnector* deben inicializarse con una cadena que tenga formato de ruta de un sistema de ficheros. Para estos casos se definen reglas como la mostrada a continuación, que define el formato que deben tener los atributos que almacenen rutas.

```
Path returns ecore::EString:
   ((ID('.'ID)*|'.'|'..') ('/'(ID('.'ID)*|'.'|'..'))* '/'?)
   | ('/' ((ID('.'ID)*|'.'|'..')'/')* (ID('.'ID)*|'.'|'..')?);
```

A diferencia del ejemplo anterior, para los atributos a los que deban asignarse cadenas entre comillas con un formato específico deben definirse reglas terminales. Para estas se utiliza la palabra reservada "terminal" y se les asignan nombres escritos con letras mayúsculas.

```
terminal DEVICE_DEF:
   '"'(((ID('.'ID)*|'.'|'..') ('/'(ID('.'ID)*|'.'|'..'))* '/'?)
   | ('/' ((ID('.'ID)*|'.'|'..')'/')* (ID('.'ID)*|'.'|'..')?))
   ':'(((ID('.'ID)*|'.'|'..') ('/'(ID('.'ID)*|'.'|'..'))* '/'?)
   | ('/' ((ID('.'ID)*|'.'|'..')'/')* (ID('.'ID)*|'.'|'..')?))
   (':'PERMISSION)?'"';
```

A la hora de definir reglas terminales hay que tener en cuenta que pueden no interpretarse como se desea. Por ejemplo, los atributos *gid* y *uid* de las clases *ConfigConnector* y *SecretConnector* deben ser inicializados con una cadena formada por un número entero entre comillas, para lo cual se define la regla terminal *QUOTED_INT*. A su vez, al atributo *version* de *DockerCompose* también se le pueden asignar números enteros entre comillas, aunque únicamente los números 1, 2 y 3

(terminal *VERSION_INT*).

```
terminal VERSION_INT:
    '"'('1'|'2'|'3')'"';
terminal QUOTED_INT:
    '"'INT'"';
```

En este caso, no habrá problemas a la hora de asignar cadenas con el formato de *VERSION_INT* al atributo *version*. Sin embargo, debido a que la definición del terminal *QUOTED_INT* es posterior a la de la otra regla, un valor que quiera ser asignado a los atributos *gid* o *uid* puede ser interpretado como terminal *VERSION_INT* en lugar de *QUOTED_INT*, siempre que la cadena cumpla el formato de la primera regla.

Para evitar que se produzca un problema en un caso como el mencionado anteriormente, se define una regla *QuotedInt* que incluye a ambos terminales, y que será la que se ha de cumplir a la hora de asignar valores a los atributos *gid* y *uid*.

```
QuotedInt returns ecore::EString:
    QUOTED_INT | VERSION_INT;
```

De manera similar, se ha creado una regla terminal llamada *PORT_DEF* que define como especificar un elemento de tipo *Port*, pero para que la definición de un puerto no sea interpretada como terminal *VERSION_INT* o *QUOTED_INT* se crea una nueva regla *Ports* que incluye a todos las terminales.

```
Ports returns ecore::EString:
    PORT_DEF | VERSION_INT | QUOTED_INT;
```

Además, a la regla *EString*, que especifica todas las formas en las que pueden escribirse cadenas, se le deben añadir todas estas reglas terminales para poder escribir cadenas sin que, necesariamente, se interpreten como alguno de estos terminales.

```
EString returns ecore::EString:
    STRING | ID | '"''"' | PORT_DEF | VERSION_INT
    | QUOTED_INT | VERSION_FLOAT | DEVICE_DEF;
```

4.2.1. Formatter

Otra de posibilides que ofrece Xtext es la de implementar un *formatter* personalizado para una gramática, es decir, una manera de reorganizar los elementos de un fichero texto, mediante el uso de espacios en blanco, tabulaciones y saltos de línea, para mejorar su legibilidad sin cambiar el valor semántico del documento. Para ello, únicamente se debe especificar en un archivo *xtend* la forma en la que se desea formatear el fichero, indicando los espacios y saltos de línea que debe haber entre cada elemento [21].

Al generar el código del editor de texto para la gramática, se crea un fichero "DockerComposeFormatter.xtend", con un código generado por defecto que puede usarse como plantilla para describir cómo debe funcionar el fomatter. El código incluye una clase *DockerComposeFormatter* con un método *format* que toma el documento y un objeto de tipo *DockerCompose* como parámetros. Este se trata del encargado de formatear todos los elementos del documento correspondientes al elemento raíz de la gramática.

El código de *format* debe llamar a otros métodos que se encarguen de formatear los objetos que parten del elemento raíz. Para ello, se recorren los servicios, redes, volúmenes, configuraciones y secretos del objeto mediante bucles *for* y se llama al método *format* de cada uno de ellos. Para que esto funcione correctamente, deben implementarse más métodos como este, cambiando el parámetro de tipo *DockerCompose* por otro de la clase que se desee formatear [25].

Por último, se deben colocar los espacios correspondientes antes o después de las palabras reservadas de la gramática correspondientes al elemento *DockerCompose*. En este caso, deben añadirse saltos de línea antes de cada una de las etiquetas principales del lenguaje. El código "dockerCompose.regionFor.keyword("services:")" sirve para localizar la cadena que se introduzca como parámetro ("services:" en este caso) dentro del elemento. Una vez localizada, se utiliza el método *prepend* para añadir algo justo

antes del elemento. Como lo que se quiere añadir en este caso es un salto de línea, se especifica “newLine”. Esto se repite para todas las distintas etiquetas del elemento.

```
def dispatch void format(DockerCompose dockerCompose,
                         extension IFormattableDocument document){
   for (service : dockerCompose.services) {
     service.format
   }
   for (network : dockerCompose.networks) {
     network.format
   }
   for (volume : dockerCompose.volumes) {
     volume.format
   }
   for (config : dockerCompose.configs) {
     config.format
   }
   for (secret : dockerCompose.secrets) {
     secret.format
   }
   dockerCompose.regionFor.keyword("services:").prepend[newLine];
   dockerCompose.regionFor.keyword("networks:").prepend[newLine];
   dockerCompose.regionFor.keyword("volumes:").prepend[newLine];
   dockerCompose.regionFor.keyword("configs:").prepend[newLine];
   dockerCompose.regionFor.keyword("secrets:").prepend[newLine];
}
```

Como ya se ha mencionado, para que un objeto sea formateado correctamente debe tener su correspondiente método *format*, con un parámetro de la clase en cuestión. Por ejemplo, para que la llamada “volume.format” funcione correctamente, se debe implementar un método como el mostrado a continuación. El código

“volume.prepend[space=“\n\t”]” añade un salto de línea y una tabulación antes del objeto *service*, es decir, antes del nombre del mismo.

Tras esto, se deben localizar las palabras reservadas de cada propiedad que pueda tener el servicio, añadiendo un salto de línea y dos tabulaciones para que tengan una indentación más profunda. Con el código “service.regionFor.keyword(“driver:”).prepend[space=“\n\t\t”]” se localiza la etiqueta “driver:” y se añaden los espacios mencionados antes de la misma. Además, se añade la llamada “append[space=“ ”]” al final de la línea, que indica que tras la etiqueta, y antes del siguiente elemento, debe haber un único espacio en blanco. Lo mismo se repite con cada etiqueta. Por último, al igual que en el método anterior, se deben llamar a los métodos *format* de todos los objetos que parten del volumen.

```
def dispatch void format(Volume volume,
                         extension IFormattableDocument document){
   volume.prepend[space="\n\t"]
   volume.regionFor.keyword(":").prepend[noSpace];
   volume.regionFor.keyword("external:")
                    .prepend[space="\n\t\t"].append[space=" "];
   volume.regionFor.keyword("driver:")
                    .prepend[space="\n\t\t"].append[space=" "];
   volume.regionFor.keyword("name:")
                    .prepend[space="\n\t\t"].append[space=" "];
   volume.regionFor.keyword("labels:").prepend[space="\n\t\t"];
   volume.regionFor.keyword("driver_opts:")
                    .prepend[space="\n\t\t"];
   for (label : volume.labels) {
     label.format
   }
   for (driver_opt : volume.driver_opts) {
     driver_opt.format
```

```
    }
}
```

Lo mismo ha de hacerse con el resto de objetos que deban ser formateados, implementando un método por cada clase en el que se indiquen los espacios, saltos de línea y tabulaciones que se deban añadir al documento. Han habido ciertos casos en los que, las tabulaciones que se debían añadir antes de cierta palabra dependían de si esta se trataba o no de la primera cadena en la definición del elemento.

El siguiente método es un ejemplo de estos casos. En él se compara el offset de, por ejemplo, la cadena "subnet:" (código "config.regionFor.keyword("subnet:").offset") con el del elemento completo (código "config.allSemanticRegions.get(0).offset"). En este caso, deben añadirse tabulaciones antes de la cadena únicamente si los offsets son diferentes, es decir, si la cadena no es la primera en la descripción del elemento.

```
def dispatch void format(IPAMConfig config,
                         extension IFormattableDocument document){
   if (config.regionFor.keyword("subnet:") !== null) {
     if (config.regionFor.keyword("subnet:").offset !=
         config.allSemanticRegions.get(0).offset) {
                  config.regionFor.keyword("subnet:")
                  .prepend[space="\n\t\t\t\t "];
     }
   }
   if (config.regionFor.keyword("ip_range:") !== null) {
     if (config.regionFor.keyword("ip_range:").offset !=
         config.allSemanticRegions.get(0).offset) {
                  config.regionFor.keyword("ip_range:")
                  .prepend[space="\n\t\t\t\t "];
     }
   }
```

```
    if (config.regionFor.keyword("gateway:") !== null) {
      if (config.regionFor.keyword("gateway:").offset !=
          config.allSemanticRegions.get(0).offset) {
                    config.regionFor.keyword("gateway:")
                    .prepend[space="\n\t\t\t\t "];
      }
    }
    if (config.regionFor.keyword("aux_addresses:") !== null) {
      if (config.regionFor.keyword("aux_addresses:").offset !=
          config.allSemanticRegions.get(0).offset) {
                    config.regionFor.keyword("aux_addresses:")
                    .prepend[space="\n\t\t\t\t "];
      }
    }
    for (address : config.aux_addresses) {
      address.format
    }
}
```

Tras definir todos los métodos necesarios en el archivo "DockerComposeFormatter.xtend", bastará con pulsar la combinación de teclas *Ctrl* + *Shift* + *F* para que el documento se formatee de la manera descrita.

El fichero "DockerComposeFormatter.xtend" completo, que define el funcionamiento del formatter desarrollado para el editor de texto, ha sido incluido en el anexo C.

4.3. Editor gráfico

La segunda sintaxis concreta desarrollada se trata de la notación gráfica definida en la sección 3.2.2.2, que permite definir y editar un modelo Docker-Compose utilizando un

diagrama, aunque pudiendo editar únicamente algunos aspectos de la sintaxis abstracta del lenguaje.

Tanto la notación como el editor gráfico han sido desarrolladas con *Sirius* [20], un proyecto de Eclipse que permite crear herramientas gráficas de modelado, aprovechando las tecnologías *EMF* y *GMF* de Eclipse. Está adaptado particularmente para casos en los que se necesita desarrollar representaciones gráficas para un DSL ya definido.

Sirius se distribuye como parte del entorno de desarrollo *Obeo Designer* [13] [14], un proyecto de Eclipse ofrecido como su "solución profesional" para el desarrollo de herramientas de modelado personalizadas. Este programa es el que se ha utilizado para desarrollar tanto el proyecto Sirius que define el editor gráfico como el resto de proyectos necesarios para definir la sintaxis abstracta y la gramática textual junto con su editor.

Un proyecto Sirius contiene un archivo de extensión *.odesign*, en el que deben describirse todos los aspectos de la notación gráfica asociada al editor. Este fichero, llamado "project.odesign" en este caso, muestra y permite editar sus propiedades como nodos en formato de árbol. Dentro del nodo "DockerComposeViewPoint" se crea un nodo de tipo *Diagram Description*, asociado a la representación gráfica, con el nombre "Docker Compose Diagram". Dentro de este deberán crearse todos los elementos necesarios para describir la sintaxis gráfica.

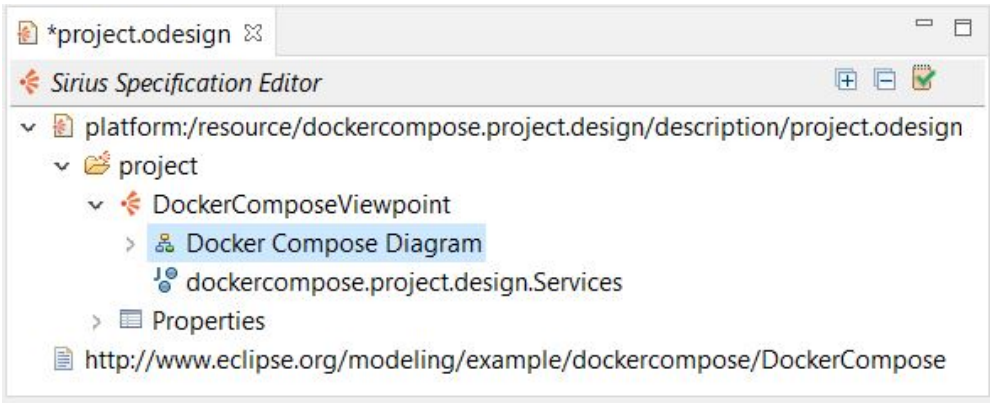

Figura 4.7: Fichero "project.odesign"

Deben crearse es una serie de capas (nodos de tipo *Layer*), cada una conteniendo ciertos elementos del diagrama. Esto permite elegir qué capas mostrar y cuáles ocultar, pudiendo mostrar solo algunos elementos deseados en el diagrama. En la representación gráfica de los modelos Docker-Compose habrá una capa por cada uno de los conceptos principales de la gramática, de manera que cada una de ellas contendrá únicamente los elementos relacionados con su concepto correspondiente. Así, se crean las capas *ServiceLayer*, *NetworksLayer*, *VolumesLayer*, *ConfigsLayer* y *SecretsLayer*.

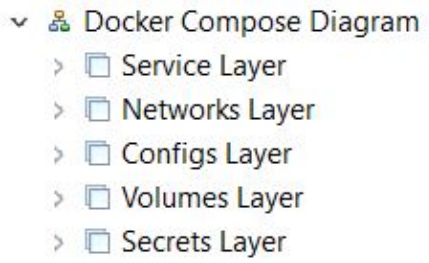

Figura 4.8: Capas definidas en el diagrama

En la notación gráfica existe un tipo de nodo por cada concepto principal (servicios, redes, volúmenes, configutaciones). Para asignarlos a su capa correspondiente, cada uno de ellos se crea dentro de la capa a la que pertenece. Creando, por ejemplo, el nodo *ServiceNode* dentro de la capa *ServiceLayer*, todos los nodos de tipo servicio en un diagrama podrán ocultarse si se oculta la capa mencionada. Así, se crean, además del ya mencionado, los nodos *NetworkNode*, *VolumeNode*, *ConfigNode* y *SecretNode*.

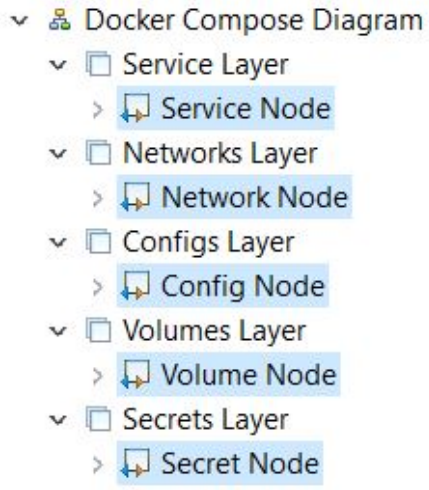

Figura 4.9: Nodos definidos en las distintas capas

Por cada nodo creado debe especificarse la clase del metamodelo que representan y el nombre de la relación de composición que une al elemento raíz (*DockerCompose*) con la metaclase en cuestión. Además de esto, se especifica el atributo *name* como etiqueta de todos los nodos, así como un icono identificativo para cada tipo de ellos.

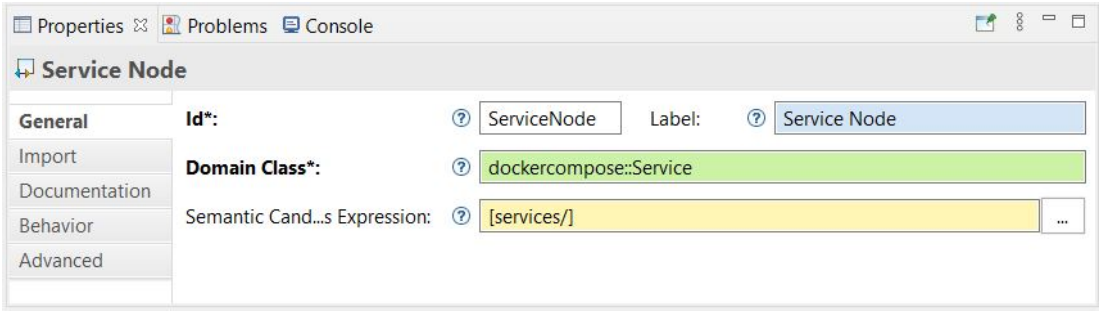

Figura 4.10: Propiedades del nodo ServiceNode

Con los nodos ya definidos, deben crearse también herramientas para poder crear nodos de cada tipo en los diagramas. Para ello, dentro de cada capa se crea una sección de herramientas llamada *NodePalette*, una paleta que incluirá todas las herramientas de creación de cada nodo. Esta sección de herramientas debe crearse de manera duplicada en cada una de las capas, aunque cada sección incluirá solo la herramienta de creación del tipo de nodo correspondiente a la capa en la que se encuentra.

En la sección *NodePalette* de la capa de servicios debe crearse la herramienta de creación de nodos servicios, un elemento de tipo *Node Creation* llamado *ServiceTool*. Ha de especificarse el tipo de objeto que debe ser creado con la herramienta, definiendo dos operaciones dentro de esta. La primera de ellas se trata de un cambio de contexto a la variable *container*, el elemento sobre el que debe utilizarse la herramienta. En este caso, se trata del elemento raíz *DockerCompose*, es decir, el diagrama en sí. Por último, debe crearse una operación de creación de una instancia, especificando su tipo (*Service*) y el nombre de la referencia que lo une con el elemento raíz (*services*).

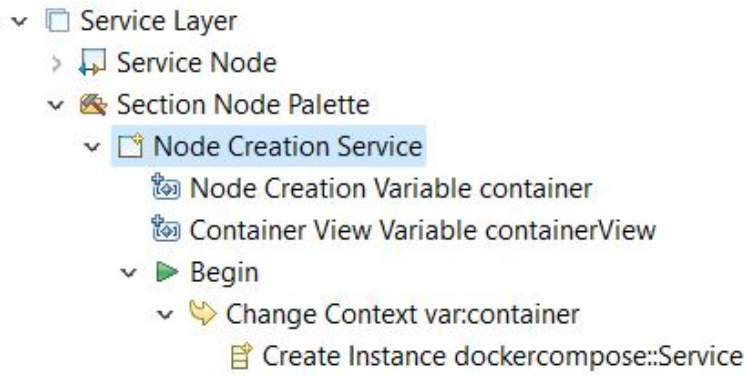

Figura 4.11: Herramienta ServiceTool

En la paleta de herramientas de cada sección debe definirse su correspondiente herramienta de creación de nodo tal y como se describe en el ejemplo anterior.

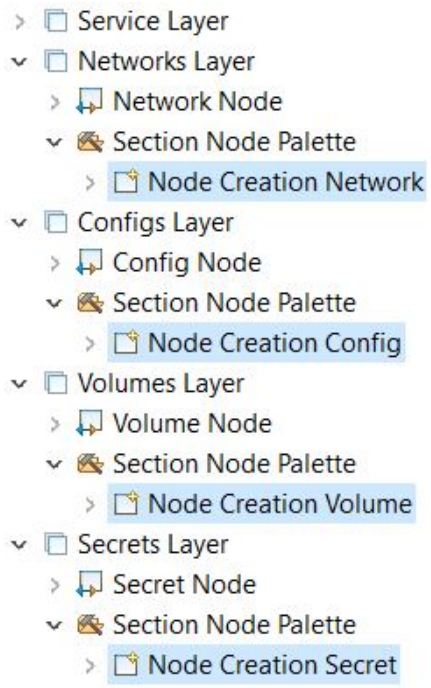

Figura 4.12: Herramientas de creación de nodos de las distintas capas

Al igual que con los nodos, deben crearse las aristas que representan relaciones entre distintos objetos en el diagrama. Dado que todas ellas parten de servicios, estas se crean dentro de la capa *ServiceLayer*, especificando en cada una las clases del metamodelo que representan tanto la arista como los elementos origen y destino.

Por ejemplo, para las relaciones entre servicios y redes se crea la arista *NetworksEdge*, especificando que representará a la clase *NetworkConnector*, que su orgien será de tipo *Service* y su destino de tipo *Network*. Además, deben especificarse las propiedades de la clase conectora que corresponden al origen y destino. El elemento origen, de tipo *Service*, se trata del contenedor de *NetworkConnector*, mientras que el destino, de tipo *Network*, está relacionado con el elemento *NetworkConnector* mediante la asociación *network*.

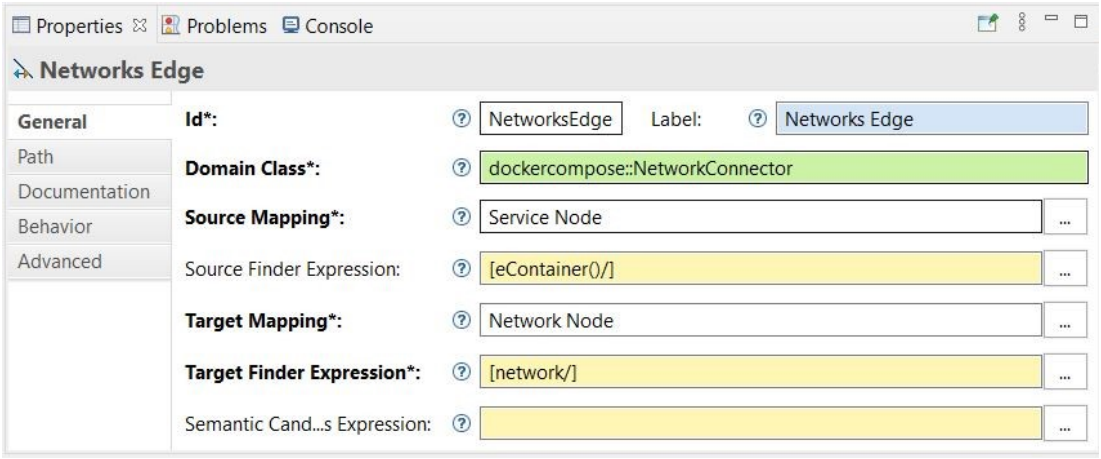

Figura 4.13: Propiedades de la arista NetworksEdge

Además de la mencionada anteriormente, se crean las aristas *DependsOnEdge*, *LinksEdge*, *VolumesEdge*, *ConfigsEdge* y *SecretsEdge* especificando la misma información.

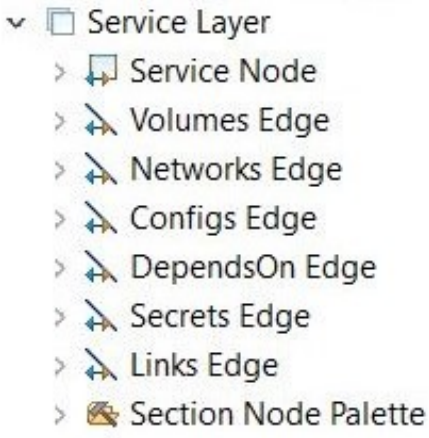

Figura 4.14: Aristas definidas en la capa ServiceLayer

Para finalizar, deben definirse herramientas para la creación de cada una de estas aristas. Estas serán creadas en una única sección de herramientas llamada *RelationsPalette*, ubicada dentro de la capa *ServiceLayer*. En cada una se debe especificar el tipo de arista que se crea con la herramienta, además de las operaciones que deben realizarse.

Para el ejemplo de la herramienta de creación de la arista *NetworksEdge*, llamada

NetworksTool, debe definirse una operación de cambio de contexto a la variable *source*, es decir, el origen de la arista. Tras esta, se crea una operación de creación de una instancia, especificando que debe ser de tipo *NetworkConnector* y que parte del origen mediante la referencia llamada *networks*. Para finalizar, debe especificarse una operación *Set* para asignar el elemento origen a la propiedad *network*.

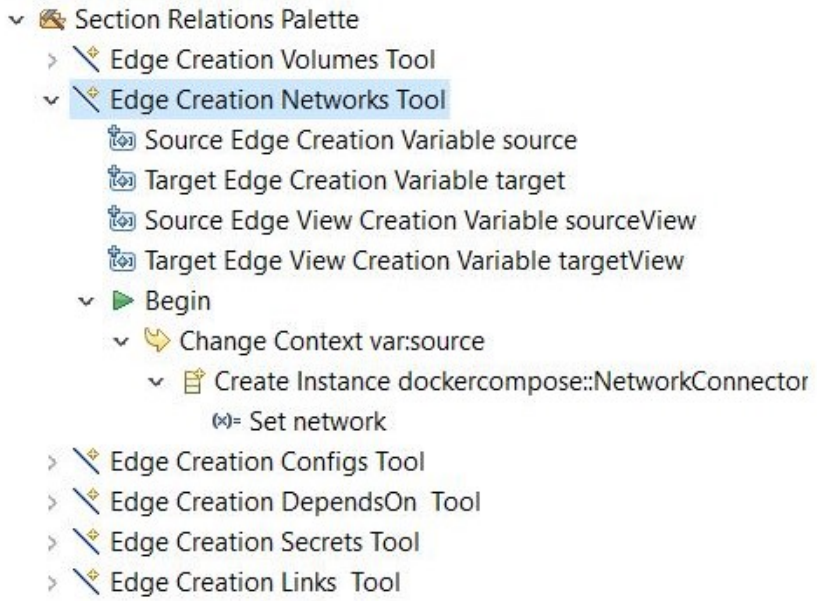

Figura 4.15: Herramientas de creación de aristas

4.3.1. Integración Sirius y Xtext

Uno de los objetivos que se pretendían conseguir al desarrollar una notación gráfica para el lenguaje es el de integrarla junto con la gramática textual de Docker-Compose, de tal manera que, desde un elemento del diagrama, puediera accederse a su fragmento correspondiente de la representación textual del modelo.

Esto fue posible gracias a la utilización de Obeo Designer con Sirius. En el fichero "project.odesign" pueden definirse propiedades personalizadas que aparecerán en la pestaña de propiedades del editor gráfico. Para ello, debe crearse un nodo de tipo *Properties View*, dentro del cual podrán las propiedades deseadas. Por cada tipo de nodo que puede definirse en un diagrama, se crea un elemento de tipo *Page* y uno de

tipo *Group*.

La página, cuyo identificado debe ser “XtextPage”, representa la nueva sección o página de propiedades en la que se mostrará la información de la representación gráfica deseada. Por otro lado, al grupo se le debe asignar la cadena “XtextGroup” como identificador, y representa la parte de la página en la que se mostrará el fragmento Xtext. En ambos tipos debe especificarse la clase representada por el nodo correspondiente en cada caso. Además, dentro de cada grupo debe crearse un elemento de tipo *Custom Widget*, en el que únicamente debe especificarse la cadena “org.eclipse.sirius.example.xtextwidget.XtextPartialViewer” como identificador. Esto indica el código que debe ejecutarse para mostrar el fragmento Xtext deseado.

Para que esto último funcione correctamente, ha de crearse un proyecto, llamado “org.eclipse.sirius.example.xtextwidget” en este caso, que almacene las clases “XtextPartialViewerController”, “XtextPartialViewerLifecycleManager”, “XtextPartialViewerLifecycleManagerProvider” y “XtextPartialViewerWidget” incluidas en el repositorio *Xtext-Sirius-integration* de *GitHub* [26].

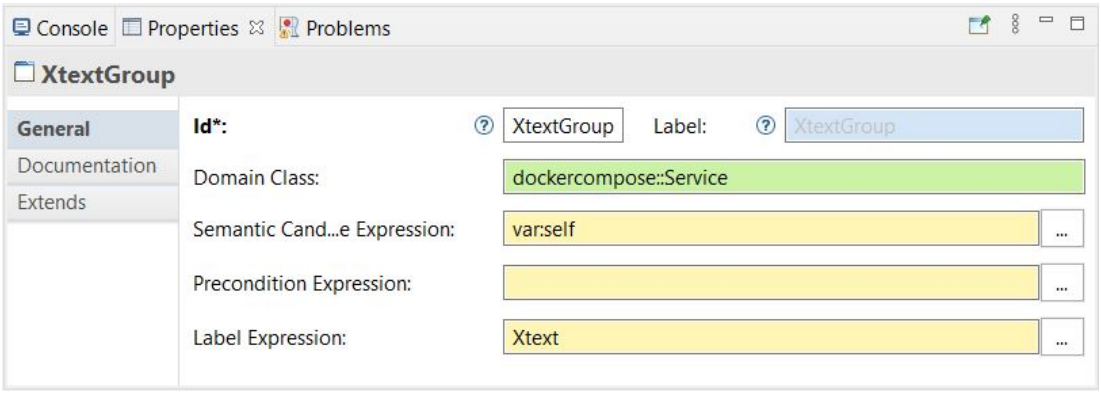

Figura 4.16: Propiedades del elemento XtextGroup correspondiente al nodo Service

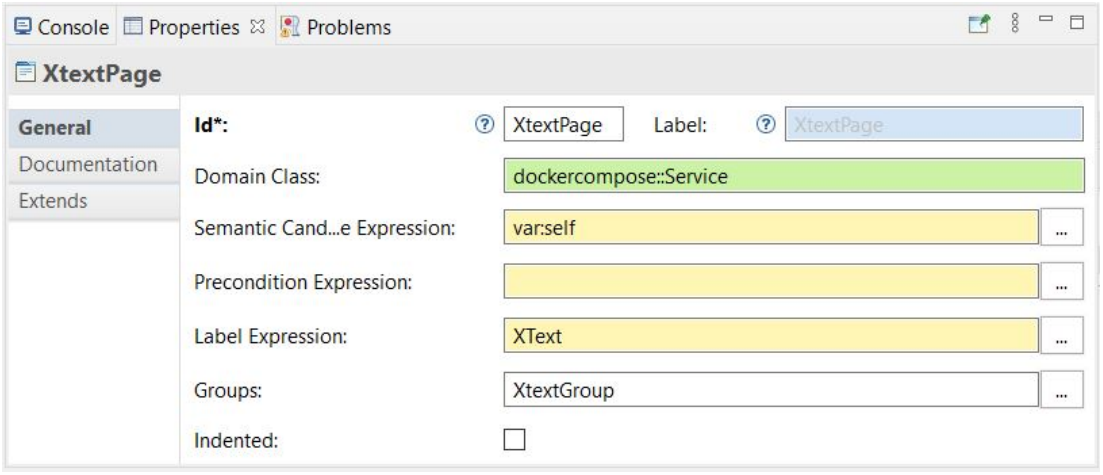

Figura 4.17: Propiedades del elemento XtextPage correspondiente al nodo Service

Capítulo 5

Ejemplos de uso

En este apartado se muestran dos ejemplos de uso del sistema desarrollado. Uno de ellos consiste en la creación de un modelo Docker-Compose desde cero, haciendo uso tanto del editor gráfico como del textual para especificar todas las propiedades necesarias. Por el contrario, en el otro ejemplo se importará un fichero de texto Docker-Compose ya existente, a partir del cual podrá generarse un diagrama utilizando la notación gráfica definida.

5.1. Ejemplo 1

Como se ha comentado previamente, este primer ejemplo de uso consiste en la creación de un modelo Docker-Compose que defina la arquitectura de una aplicación a modo de ejemplo. Para ello, se ha hecho uso del editor gráfico para construir un diagrama con la estructura principal del modelo, definiendo los objetos Docker de la arquitectura y las relaciones entre los mismos.

El diagrama creado se muestra en la figura 5.1. El modelo consta de dos contenedores. Uno de ellos, llamado “agrologistics_db”, será el encargado de ejecutar la base de datos, mientras que el servicio “agrologistics_api” ejecutará una API para la

aplicación. Para lograr una comunicación entre ambos, estos deben estar conectados a una misma red, para lo que se define la red "net". Además, se especifica que "agrologistics_api" depende del otro contenedor, de manera que se inicie posteriormente. Por último, el contenedor correspondiente a la base de datos ha de tener una serie de volúmenes ("mysql", "mysql_model", "mysql_data" y "mysql_user") para conseguir una persistencia de los datos de la aplicación.

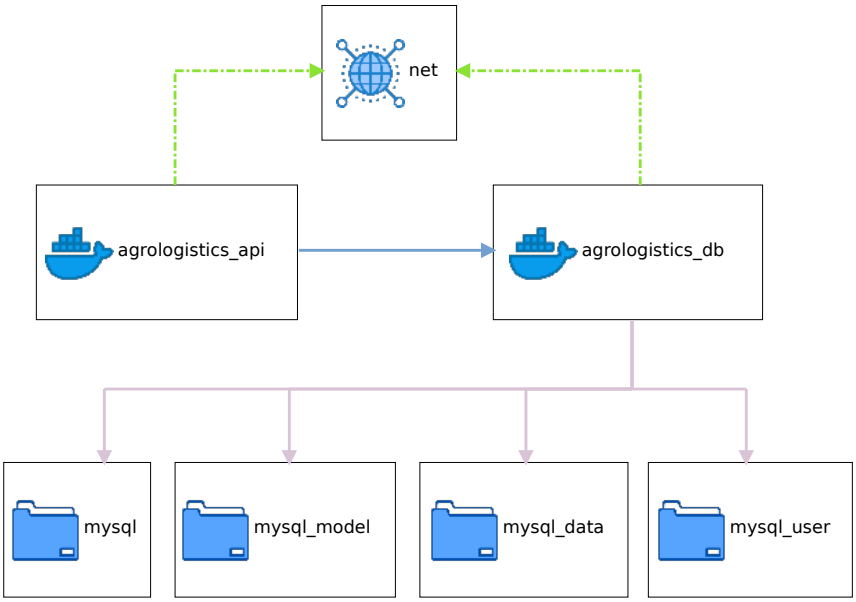

Figura 5.1: Arquitectura de ejemplo modelada con el editor gráfico

Definiendo el modelo con el diagrama de la figura 5.1 en el editor gráfico, su correspondiente representación textual con la gramática de Docker-Compose queda de la siguiente manera:

```
services:
  agrologistics_db:
    networks:
      - net
    volumes:
      - mysql_model:.
      - mysql_data:.
```

```
        - mysql_user:.
        - mysql:.
  agrologistics_api:
     depends_on:
        - agrologistics_db
     networks:
        - net
networks:
  net:
volumes:
  mysql_model:
  mysql_data:
  mysql_user:
  mysql:
```

Con esta modelo todavía no se ha conseguido la definición de una aplicación completa, ya que para especificar cómo deben comportarse tanto los contenedores de la misma como el resto de sus objetos se necesitan asignar valores a algunas propiedades que no pueden editarse de manera gráfica. Además de esto, hace falta incluso inicializar algunos atributos obligatorios para que el modelo sea válido, específicamente, el atributo *image* o *build* de los contenedores.

Para completar la definición del modelo de ejemplo se hace uso del editor textual, especificando la versión del elemento raíz, añadiendo los atributos obligatorios necesarios y algunos otros que configuran el comportamiento de la aplicación. Entre otros, se asignan direcciones IPv4 estáticas para los servicios en la red, se define un controlador y una configuración IPAM para la red y se especifican variables de entorno, políticas de reinicio y puertos expuestos de los contenedores.

```
version: "3.3"
services:
```

```
agrologistics_db:
    image: mysql:5.7
    restart: always
    environment:
        MYSQL_DATABASE: 'agrologistics'
        MYSQL_USER: 'user'
        MYSQL_PASSWORD: 'root'
        MYSQL_ROOT_PASSWORD: 'root'
        MYSQL_ROOT_HOST: '%'
    ports:
        - "3310:3306"
    networks:
        net:
            ipv4_address: 172.30.0.2
    volumes:
        - mysql_model:/docker-entrypoint-initdb.d/model.sql
        - mysql_data:/docker-entrypoint-initdb.d/data.sql
        - mysql_user:/docker-entrypoint-initdb.d/user.sql
        - mysql:/etc/mysql/my.cnf
agrologistics_api:
    build: ./api/
    restart: on-failure
    ports:
        - "8080:8080"
    depends_on:
        - agrologistics_db
    command: sh -c "node index.js"
    networks:
        net:
```

```
            ipv4_address: 172.30.0.3
networks:
   net:
       driver: bridge
       ipam:
           driver: default
           config:
               - subnet: 172.30.0.0/8
volumes:
   mysql_model:
   mysql_data:
   mysql_user:
   mysql:
```

Al especificar los valores de todas estas propiedades se está configurando el funcionamiento de cada uno de los objetos principales del modelo. Sin embargo, no se modifica la estructura principal de este (objetos Docker definidos y sus relaciones), por lo que el diagrama no sufre modificaciones al completar la especificación textual.

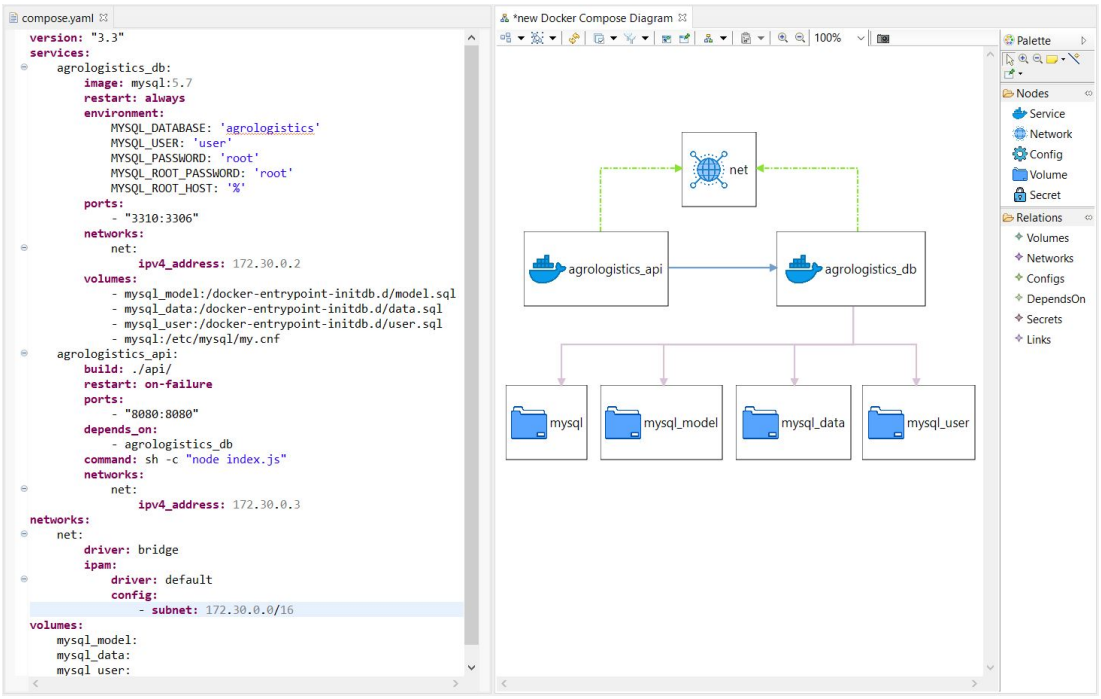

Figura 5.2: Representación textual y gráfica del primer modelo de ejemplo en los editores

5.2. Ejemplo 2

Como segundo ejemplo de uso, se ha creado un modelo importando un archivo Docker-Compose ya existente, extraído del repositorio de aplicaciones de muestra Awesome-compose de Docker [15]. El fichero de texto, cuyo contenido se muestra a continuación, define dos contenedores que utilizan una red, un volumen y un secreto.

```
services:
    backend:
        build: backend
        ports:
            - "8080:8080"
        environment:
            - POSTGRES_DB=example
        networks:
```

```
            - spring-postgres
    db:
        image: postgres
        restart: always
        secrets:
            - db-password
        volumes:
            - db-data:/var/lib/postgresql/data
        networks:
            - spring-postgres
        environment:
            - POSTGRES_DB=example
            - POSTGRES_PASSWORD_FILE=/run/secrets/db-password
volumes:
    db-data:
secrets:
    db-password:
        file: db/password.txt
networks:
    spring-postgres:
```

Con la notación gráfica desarrollada utilizando Sirius, puede generarse fácilmente el diagrama asociado al modelo creado a partir del fichero de texto. La representación gráfica en cuestión se muestra en la figura 5.3.

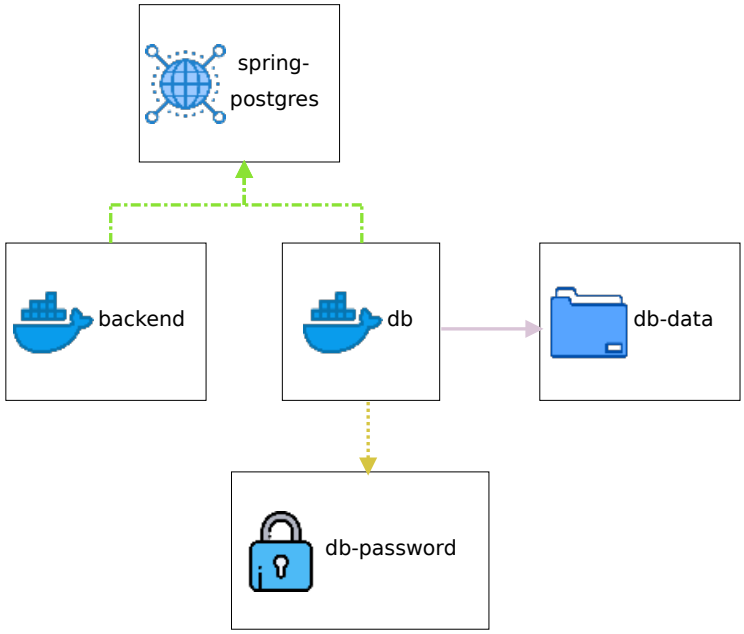

Figura 5.3: Diagrama asociado al fichero Docker-Compose de ejemplo

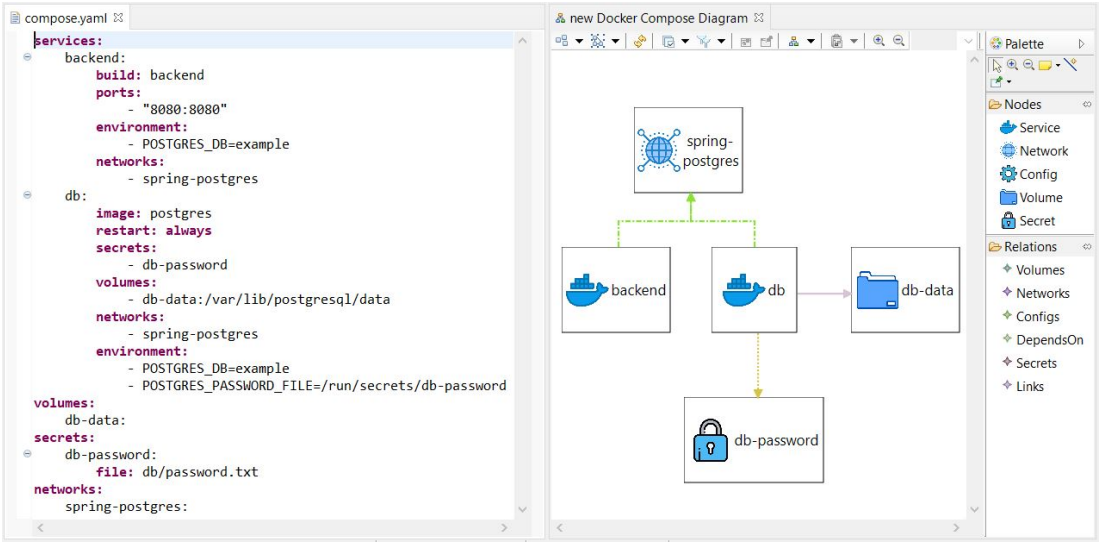

Figura 5.4: Representación textual y gráfica del segundo modelo de ejemplo en los editores

Capítulo 6

Conclusiones y trabajos futuros

Este capítulo recoge las principales dificultades afrontadas durante el Proyecto, presenta las principales conclusiones a las que hemos llegado tras su desarrollo y propone algunas posibles extensiones que podría resultar interesante abordar en el futuro como continuación de este Trabajo.

6.1. Principales dificultades encontradas

A continuación, se describen algunas de las principales dificultades encontradas a lo largo del Proyecto, así como las soluciones y decisiones adoptadas para afrontar cada una de ellas.

6.1.1. Definición de la gramática de Docker-Compose

Una vez estudiada y comprendida la estructura y el funcionamiento de la especificación Docker-Compose, debíamos definir la gramática del lenguaje de modelado que íbamos a implementar de forma que ésta recogiera todos los elementos contemplados en el estándar. Para ello se consideraron dos posibilidades: (1) definir directamente la gramática del lenguaje con Xtext y, a partir de ella, derivar el meta-modelo EMF

necesario para obtener la implementación de nuestros editores de modelos (tanto el textual como el gráfico), o bien (2) partir de un meta-modelo EMF que especificara la sintaxis abstracta del lenguaje (común a los dos editores) y, a partir de él, obtener la gramática Xtext equivalente. Esta gramática debería posteriormente ajustarse para hacer coincidir la sintaxis concreta textual del lenguaje con la del estándar.

Inicialmente, optamos por la primera opción ya que nos pareció la más sencilla: básicamente consistía en codificar en Xtext la gramática descrita en el estándar. Sin embargo, el meta-modelo EMF generado a partir de nuestra gramática resultó ser muy complejo y difícil de comprender y modificar (por ejemplo, para añadir las restricciones OCL identificadas como necesarias). Además, nos resultó imposible implementar un editor gráfico operativo (ni con GMF/EuGENia ni con Sirius) basado en el meta-modelo generado. Así las cosas, finalmente optamos por desarrollar un meta-modelo EMF que recogiera los elementos principales del estándar y, a partir de él, generar y ajustar la gramática Xtext.

6.1.2. Implementación y ajuste del editor textual

El ajuste de la gramática Xtext generada a partir de nuestro meta-modelo supuso un importante reto. Por ejemplo, resultó necesario prestar especial atención a la hora de definir las reglas terminales, a fin de evitar que determinadas cadenas pudieran interpretarse con más de una de ellas. También resultó especialmente complejo dar soporte al hecho de que determinadas palabras reservadas puedan interpretarse de distinto modo según el contexto en el que aparezcan, o que el estándar permita especificar determinados elementos de distintas formas (representaciones alternativas). Resulta curioso sin embargo que, aunque para algunos elementos el estándar sólo define una posible representación, es posible encontrar en Internet diversos ejemplos de especificaciones válidas en las que el formato utilizado para definir dichos elementos difiere del recogido en el estándar. La identificación de estos aspectos ambiguos o insuficientemente detallados en el estándar sólo ha sido posible gracias a la revisión de

muchos de los ejemplos disponibles tanto en el estándar como en los muchos ejemplos de proyectos Docker-Compose disponibles en Internet.

6.1.3. Implementación y ajuste del editor gráfico

La principal dificultad que hemos encontrado a la hora de implementar nuestro editor gráfico ha sido la de dar con las herramientas adecuadas para lograr su sincronización e integración con el nuestro textual basado en Xtext. Inicialmente, optamos por utilizar GMF/EuGENia para especificar la notación gráfica de nuestro lenguaje, ya que conocíamos estas herramientas y habíamos trabajado previamente con ellas. Sin embargo, su integración con Xtext está poco documentada y, sinceramente, tras probar numerosas soluciones (las propuestas en la documentación de las propias herramientas, las ofrecidas por diversos usuarios a través de varios foros de Internet y nuestro propios intentos a base de indagar en la estructura de los ficheros generados tanto por GMF/EuGENia como por Xtext), dudamos de que esta integración sea realmente factible, al menos, con las últimas versiones de Eclipse Modeling Tools, con las que hemos probado.

Finalmente, optamos por migrar todo nuestro proyecto a Obeo Designer [13] v. 11.6: un entorno de modelado basado en Eclipse que incluye las principales herramientas típicamente incluidas en Eclipse Modeling Tools (EMF, OCL, Epsilon, Xtext, EuGENia, Sirius, Acceleo, etc.), además de facilitar la integración con diversas herramientas externas (UML, GraphML, Simulink, etc.) y dar soporte, de forma nativa, a la sincronización de los modelos gráficos y textuales obtenidos a partir de editores desarrollados con Sirius y Xtext, respectivamente.

A pesar de tener que aprender desde cero el manejo de Sirius (como alternativa a GMF/EuGENia) para construir nuestro editor gráfico, una vez comprendido su funcionamiento básico, resultó relativamente sencillo obtener una primera versión operativa de la herramienta. Esta primera versión nos permitió comprobar cómo los modelos gráficos y los textuales se sincronizaban correcta y automáticamente (al

guardar cualquiera de ellos), incluso estando los dos abiertos simultáneamente y sin tener que persistir ninguno de ellos en formato XMI. A partir de esta primera versión, se realizaron numerosas mejoras, entre las que cabe destacar (1) la integración de Xtext en Sirius para poder mostrar, en la vista de propiedades, la especificación textual asociada al elemento seleccionado en el modelo gráfico; y (2) la definición en los modelos gráficos de múltiples capas que pueden mostrarse u ocultarse de forma independiente para así obtener distintas vistas de de las arquitecturas Docker-Compose que se modelan.

6.2. Conclusiones

Una vez finalizado el Proyecto, a continuación se describen las principales conclusiones obtenidas del mismo en cuanto a los resultados obtenidos y lo aprendido en el proceso.

Como principal resultado de este Proyecto se han desarrollado varias herramientas orientadas a facilitar la creación, edición, visualización y validación de arquitecturas Docker-Compose. En este sentido, se ha cumplido con los objetivos inicialmente marcados al principio del Trabajo (ver Sección 1.2).

La realización de este Proyecto me ha permitido poner en práctica mucho de lo aprendido a lo largo de mis estudios de Grado, por ejemplo, en relación con el diseño e implementación de herramientas de modelado, la teoría de lenguajes, las buenas prácticas de programación o la planificación y gestión de proyectos software, entre muchos otros. Además, me ha ofrecido la oportunidad de aprender el manejo de nuevas tecnologías y herramientas: desde Docker/Docker-Compose hasta Sirius, pasado por Latex/Overleaf (herramientas utilizadas para preparar la presente memoria).

Personalmente, haber sido capaz de conjugar todo lo aprendido en el Grado y de afrontar los numerosos retos y dificultades que se han ido planteando a lo largo del Proyecto, logrando finalmente alcanzar los objetivos marcados, ha supuesto para

mí una gran satisfacción. El desarrollo de lenguajes específicos de dominio ha sido uno de los aspectos que más me ha gustado en el Grado, por lo que el trabajo realizado me ha parecido bastante interesante, a pesar de las muchas dificultades (la mayoría de carácter técnico) que se han ido planteando.

Por último, me gustaría destacar que tanto la forma de trabajar con los tutores del Proyecto como su trato para conmigo me han sorprendido bastante y muy gratamente. Ambos han intentado ayudarme en todo lo posible y siempre han estado disponibles para intentar resolver mis dudas o pensar, junto a mí, posibles soluciones a los problemas que iban surgiendo.

6.3. Trabajos futuros

Para terminar, en esta sección se enumeran una serie de posibles trabajos futuros que podrían desarrollarse como continuación del presente Trabajo.

- Como ya se ha comentado, el estándar de Docker-Compose es enormemente complejo y extenso, en gran parte por la cantidad de propiedades que pueden especificarse para cada uno sus elementos. La sintaxis abstracta desarrollada en el Proyecto contempla todos los posibles atributos de los elementos de tipo red (`Network`), volumen (`Volume`), configuración (`Config`) y secreto (`Secret`), pero no cubre la totalidad de las propiedades que pueden definirse para los contenedores (`Service`). Por lo tanto, una posible extensión sería completar la especificación tanto del meta-modelo como de la gramática desarrolladas en el Proyecto para dar soporte a la totalidad del estándar.

- Además, la gramática también podría extenderse (sin necesidad de modificar el meta-modelo) para dar soporte tanto a las distintas formas de especificar determinados elementos, tal y como permite el estándar, como para permitir pequeñas variantes válidas de formato (por ejemplo, atributos que pueden definirse entre comillas o sin ellas). Conviene señalar que la implementación

de estas extensiones para dotar al editor textual de toda la flexibilidad que ofrece el estándar, no es en absoluto trivial.

- Por último, también podría resultar muy interesante desarrollar herramientas similares a las implementadas en este Proyecto pero para otras tecnologías de contenedores como, por ejemplo, *Kubernetes*. De hecho, esto abriría la puerta a poder implementar transformaciones de modelos que permitieran traducir automáticamente especificaciones Docker-Compose a *Kubernetes*, y viceversa.

Anexos

Apéndice A

Anexo 1: Metamodelo en formato OCLinEcore

En este anexo se incluye el fichero “DockerCompose.ecore” en formato OCLinEcore.

```
1  import ecore : 'http://www.eclipse.org/emf/2002/Ecore';
2
3  package dockercompose : dockercompose = 'http://www.eclipse.org/
4  modeling/example/dockercompose/DockerCompose'
5  {
6      class DockerCompose
7      {
8          attribute version : String[?];
9          property services : Service[+|1] { ordered composes };
10         property networks : Network[*|1] { ordered composes };
11         property volumes : Volume[*|1] { ordered composes };
12         property configs : Config[*|1] { ordered composes };
13         property secrets : Secret[*|1] { ordered composes };
14     }
15
```

```
16      class Service
17      {
18          attribute name : String[1] { id };
19          attribute build : String[?];
20          attribute cpu_count : ecore::EInt[1];
21          attribute command : String[?];
22          attribute container_name : String[?];
23          property devices : Device[*|1] { ordered composes };
24          property dns : DNS[*|1] { ordered composes };
25          attribute image : String[?];
26          attribute init : Boolean[1];
27          attribute read_only : Boolean[1];
28          attribute restart : RestartPolicy[?];
29          property depends_on :
30                  Dependency[*|1] { ordered composes };
31          property volumes :
32                  VolumeConnector[*|1] { ordered composes };
33          property configs :
34                  ConfigConnector[*|1] { ordered composes };
35          property secrets :
36                  SecretConnector[*|1] { ordered composes };
37          property networks :
38                  NetworkConnector[*|1] { ordered composes };
39          property ports : Port[*|1] { ordered composes };
40          property links : Link[*|1] { ordered composes };
41          property environment :
42                  EnvironmentVariable[*|1] { ordered composes };
43          invariant image_or_build:
44                   (image->excluding('')->notEmpty() and
```

```
45                  build->excluding('')->isEmpty()) or
46                  (image->excluding('')->isEmpty() and
47                  build->excluding('')->notEmpty());
48          invariant no_self_dependencies:
49                  self.depends_on.service->excludes(self);
50          invariant no_self_links:
51                  self.links.service->excludes(self);
52          invariant different_networks:
53                  self.networks.network->isUnique(name);
54          invariant different_volumes:
55                  self.volumes.volume->isUnique(name);
56          invariant different_configs:
57                  self.configs.config->isUnique(name);
58          invariant different_secrets:
59                  self.secrets.secret->isUnique(name);
60          invariant different_dependencies:
61                  self.depends_on.service->isUnique(name);
62          invariant different_links:
63                  self.links.service->isUnique(name);
64          invariant different_environment_variables:
65                  self.environment->isUnique(name);
66      }
67      class Network
68      {
69          attribute name : String[1] { id };
70          attribute attachable : Boolean[1];
71          attribute enable_ipv6 : Boolean[1];
72          attribute internal : Boolean[1];
73          attribute external : Boolean[1];
```

```
74          attribute network_name : String[?];
75          property labels : NetworkLabel[*|1] { ordered composes };
76          property driver_opts :
77                  NetworkDriverOpt[*|1] { ordered composes };
78          attribute driver : String[?];
79          property ipam : IPAM[?] { composes };
80          invariant different_labels: self.labels->isUnique(name);
81          invariant different_driver_opts:
82                   self.driver_opts->isUnique(name);
83      }
84      class Volume
85      {
86          attribute name : String[1] { id };
87          attribute driver : String[?];
88          attribute external : Boolean[1];
89          attribute volume_name : String[?];
90          property driver_opts :
91                  VolumeDriverOpt[*|1] { ordered composes };
92          property labels : VolumeLabel[*|1] { ordered composes };
93          invariant different_labels: self.labels->isUnique(name);
94          invariant different_driver_opts:
95                   self.driver_opts->isUnique(name);
96      }
97      class Config
98      {
99          attribute name : String[1] { id };
100         attribute file : String[?];
101         attribute external : Boolean[1];
102         attribute config_name : String[?];
```

```
103         invariant file_or_external:
104                   (file->excluding('')->notEmpty()
105                   and external=false) or
106                   (file->excluding('')->isEmpty()
107                   and external=true);
108         invariant external_name:
109                   config_name->excluding('')->notEmpty()
110                   implies external=true;
111     }
112     class Secret
113     {
114         attribute name : String[1] { id };
115         attribute file : String[?];
116         attribute environment : String[?];
117         attribute external : Boolean[1];
118         attribute secret_name : String[?];
119         invariant file_or_external:
120                   (file->excluding('')->notEmpty()
121                   and external=false) or
122                   (file->excluding('')->isEmpty() and external=true);
123         invariant external_name:
124                   secret_name->excluding('')->notEmpty()
125                   implies external=true;
126     }
127     class Device
128     {
129         attribute value : String[1];
130     }
131
```

```
132
133     class DNS
134     {
135         attribute dns1 : ecore::EInt[1];
136         attribute dns2 : ecore::EInt[1];
137         attribute dns3 : ecore::EInt[1];
138         attribute dns4 : ecore::EInt[1];
139         invariant correctIpFormat: dns1>=0 and dns2>=0 and
140                   dns3>=0 and dns4>=0 and dns1<=255 and
141                   dns2<=255 and dns3<=255 and dns4<=255;
142     }
143     enum RestartPolicy { serializable }
144     {
145         literal no : '"no"';
146         literal always = 1;
147         literal onfailure : 'on-failure' = 2;
148         literal unlessstopped : 'unless-stopped' = 3;
149     }
150     class VolumeDriverOpt
151     {
152         attribute name : String[1];
153         attribute value : String[1];
154     }
155     class VolumeLabel
156     {
157         attribute name : String[1];
158         attribute value : String[1];
159     }
160
```

```
161
162     class NetworkLabel
163     {
164         attribute name : String[1];
165         attribute value : String[1];
166     }
167
168     class NetworkDriverOpt
169     {
170         attribute name : String[1];
171         attribute value : String[1];
172     }
173
174     class IPAM
175     {
176         attribute driver : String[?];
177         property options : IPAMOption[*|1] { ordered composes };
178         property configs : IPAMConfig[*|1] { ordered composes };
179         invariant any_property:
180                   (driver->excluding('')->notEmpty()) or
181                   (options->size()>0) or (configs->size()>0);
182         invariant different_options: self.options->isUnique(name);
183     }
184
185     class IPAMOption
186     {
187         attribute name : String[1];
188         attribute value : String[1];
189     }
```

```
190     class VolumeConnector
191     {
192         property volume : Volume[1];
193         attribute container_path : String[1] = '.';
194         attribute access_mode : AccessMode[?];
195         attribute type : MountType[?];
196         attribute read_only : Boolean[1];
197         attribute nocopy : Boolean[1];
198         attribute size : ecore::EInt[1];
199         attribute propagation : PropagationType[?];
200     }
201     class Dependency
202     {
203         property service : Service[1];
204         attribute condition : Condition[?];
205     }
206     class ConfigConnector
207     {
208         property config : Config[1];
209         attribute target : String[?];
210         attribute uid : String[?];
211         attribute gid : String[?];
212         attribute mode : ecore::EInt[1];
213     }
214     class SecretConnector
215     {
216         property secret : Secret[1];
217         attribute target : String[?];
218         attribute uid : String[?];
```

```
219
220         attribute gid : String[?];
221         attribute mode : ecore::EInt[1];
222     }
223     class NetworkConnector
224     {
225         property network : Network[1];
226         attribute priority : ecore::EInt[1];
227         property ipv4_address : DNS[?] { composes };
228         property link_local_ips : DNS[*|1] { ordered composes };
229         property aliases : Alias[*|1] { ordered composes };
230         invariant different_options:
231                   self.aliases->isUnique(alias);
232     }
233     enum AccessMode { serializable }
234     {
235         literal rw;
236         literal ro = 1;
237         literal z = 2;
238     }
239     class NetworkAddress extends DNS
240     {
241         attribute netId : ecore::EInt[1];
242     }
243     class Alias
244     {
245         attribute alias : String[?];
246     }
247
```

```
248     class IPAMAddress
249     {
250         attribute name : String[1];
251         property dns : DNS[1] { composes };
252     }
253     class IPAMConfig
254     {
255         property gateway : DNS[?] { composes };
256         property subnet : NetworkAddress[?] { composes };
257         property ip_range : NetworkAddress[?] { composes };
258         property aux_addresses :
259                   IPAMAddress[*|1] { ordered composes };
260         invariant any_property: (gateway->notEmpty()) or
261                   (subnet->size()>0) or (ip_range->size()>0)
262                   or (aux_addresses->size()>0);
263         invariant different_addresses:
264                   self.aux_addresses->isUnique(name);
265     }
266     class Port
267     {
268         attribute value : String[1];
269     }
270     enum MountType { serializable }
271     {
272         literal volume;
273         literal bind = 1;
274         literal tmpfs = 2;
275         literal npipe = 3;
276     }
```

```
277     enum PropagationType { serializable }
278     {
279         literal rprivate;
280         literal private = 1;
281         literal rshared = 2;
282         literal shared = 3;
283         literal slave = 4;
284         literal rslave = 5;
285     }
286     enum Condition { serializable }
287     {
288         literal service_started;
289         literal service_healthy = 1;
290         literal service_completed_successfully = 2;
291     }
292     class Link
293     {
294         property service : Service[1];
295         attribute alias : String[?];
296     }
297     class EnvironmentVariable
298     {
299         attribute name : String[1];
300         attribute value : String[?];
301     }
302 }
```

Apéndice B

Anexo 2: Gramática Xtext

En este anexo se incluye el fichero "DockerCompose.xtext", en el que se define la gramática del lenguaje de modelado.

```
grammar org.xtext.example.dockercompose.DockerCompose
with org.eclipse.xtext.common.Terminals

import "http://www.eclipse.org/modeling/example/dockercompose
       /DockerCompose"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

DockerCompose returns DockerCompose:
 (
        ('version:'    version=Version)?
    &   ('services:'  (services+=Service)+ )
    &   ('volumes:'   (volumes+=Volume)+ )?
    &   ('configs:'   (configs+=Config)+ )?
    &   ('secrets:'   (secrets+=Secret)+ )?
    &   ('networks:'  (networks+=Network)+ )?
 );
```

```
17
18 Version returns ecore::EString:
19  VERSION_INT | VERSION_FLOAT;
20
21 terminal VERSION_INT:
22  '"'('1'|'2'|'3')'"';
23
24 terminal VERSION_FLOAT:
25  '"'('2.'('1'..'9')|'3.'('1'..'9'))'"';
26
27 terminal QUOTED_INT:
28  '"'INT'"';
29
30 QuotedInt returns ecore::EString:
31  QUOTED_INT | VERSION_INT;
32
33 EDouble returns ecore::EDouble:
34  INT '.' INT;
35
36 Service returns Service:
37  {Service}
38  name=ID ':'
39  (
40         ('build:'         build=PATH)?
41     &   ('image:'         image=Image)?
42     &   ('cpu_count:'       cpu_count=EInt)?
43     &   ('command:'       command=Command)?
44     &   ('container_name:'    container_name=EString)?
45     &   ('restart:'       restart=RestartPolicy)?
```

```
46      &   ('init:'         init=EBoolean)?
47      &   ('read_only:'     read_only=EBoolean)?
48
49      &   ('links:'         ('-'links+=Link)+ )?
50      &   ('depends_on:'       (depends_on+=Dependency)+)?
51      &   ('networks:'      ((networks+=NetworkConnector_long)+ |
52                           (networks+=NetworkConnector_short)+ ))?
53      &   ('volumes:'       ('-'volumes+=VolumeConnector)+)?
54      &   ('configs:'       ('-'configs+=ConfigConnector)+)?
55      &   ('secrets:'       ('-'secrets+=SecretConnector)+)?
56
57      &   ('environment:'   ((environment+=EnvironmentVariableMap)+
58                         | (environment+=EnvironmentVariableList)+ ))?
59      &   ('devices:'       ('-'devices+=Device)+ )?
60      &   ('dns:'           (('-'dns+=DNS)+ | dns+=DNS) )?
61      &   ('ports:'         ('-'ports+=Port)+ )?
62   );
63
64  EnvironmentVariableMap returns EnvironmentVariable:
65   {EnvironmentVariable}
66   name=ID':' (value=EString|value=PATH)?;
67
68 EnvironmentVariableList returns EnvironmentVariable:
69   {EnvironmentVariable}
70   '-' name=ID('='(value=EString|value=PATH))?;
71
72  Port returns Port:
73   {Port}
74   value=Ports;
```

```
75
76  Device returns Device:
77   {Device}
78   value=DEVICE_DEF;
79
80  Image returns ecore::EString:
81   (dotID(':'(EInt|dotID)+)*'/')? (dotID(':'(EInt|dotID)+)*'/')?
82   ID (':'(EInt|EDouble|'-'?dotID)+|'@'dotID(':'(EInt|dotID)+)*)?;
83
84  Link returns Link:
85   {Link}
86   service=[Service|ID](':'alias=EString)?;
87
88  Dependency returns Dependency:
89   {Dependency}
90   (('-'service=[Service|ID]) |
91   (service=[Service|ID]':'
92   'condition:' condition=Condition));
93
94  NetworkConnector_long returns NetworkConnector:
95   {NetworkConnector}
96   network=[Network|ID]':'
97   (
98          ('ipv4_address:'   ipv4_address=DNS)?
99      &   ('priority:'     priority=EInt)?
100
101     &   ('aliases:'      ('-'aliases+=Alias)+ )?
102     &   ('link_local_ips:'   ('-'link_local_ips+=DNS)+ )?
103  );
```

```
104
105 NetworkConnector_short returns NetworkConnector:
106  {NetworkConnector}
107  '-'network=[Network|ID];
108
109 Alias returns Alias:
110  {Alias}
111  alias=EString;
112
113 VolumeConnector returns VolumeConnector:
114  {VolumeConnector}
115  ((volume=[Volume|ID]':'container_path=PATH(':'access_mode=AccessMode)?) |
116  (
117         ('source:'        volume=[Volume|ID])
118     &   ('type:'         type=MountType)?
119     &   ('target:'         container_path=PATH)
120     &   ('read_only:'       read_only=EBoolean)?
121     &   ('bind:'
122          'propagation:'   propagation=PropagationType)?
123     &   ('volume:'
124          'nocopy:'      nocopy=EBoolean)?
125     &   ('tmpfs:'
126          'size'':'      size=EInt)?
127  ));
128
129 ConfigConnector returns ConfigConnector:
130  {ConfigConnector}
131  ((config=[Config|ID]) |
132  (
```

```
133         ('source:'   config=[Config|ID])
134     &   ('target:'   target=PATH)?
135     &   ('uid:'   uid=QuotedInt)?
136     &   ('gid:'   gid=QuotedInt)?
137     &   ('mode:'   mode=EInt)?
138  ));
139
140 SecretConnector returns SecretConnector:
141  {SecretConnector}
142  ((secret=[Secret|ID]) |
143  (
144         ('source:'   secret=[Secret|ID])
145     &   ('target:'   target=PATH)?
146     &   ('uid:'   uid=QuotedInt)?
147     &   ('gid:'   gid=QuotedInt)?
148     &   ('mode:'   mode=EInt)?
149  ));
150
151 Network returns Network:
152  {Network}
153  name=ID ':'
154  (
155     '{''}' |
156         (('driver:'     driver=EString)?
157     &   ('attachable:'   attachable=EBoolean)?
158     &   ('enable_ipv6:'   enable_ipv6=EBoolean)?
159     &   ('internal:'   internal=EBoolean)?
160     &   ('external:'   external=EBoolean)?
161
```

```
162
163     &   ('name:'       network_name=EString)?
164     &   ('labels:'       (labels+=NetworkLabel)+ )?
165     &   ('driver_opts:' (driver_opts+=NetworkDriverOpt)+ )?
166     &   ('ipam:'       ipam=IPAM)?)
167  );
168
169 Volume returns Volume:
170  {Volume}
171  name=ID':'
172  (
173         ('external:'   external?=EBoolean)?
174     &   ('driver:'      driver=EString)?
175     &   ('name:'       volume_name=EString)?
176     &   ('labels:'       (labels+=VolumeLabel)+ )?
177     &   ('driver_opts:' (driver_opts+=VolumeDriverOpt)+ )?
178  );
179
180 Config returns Config:
181  {Config}
182  name=ID':'
183  (
184         ('external:'   external=EBoolean)?
185     &   ('file:'       file=PATH)?
186     &   ('name:'       config_name=EString)?
187  );
188
189
190
```

```
191
192 Secret returns Secret:
193  {Secret}
194  name=ID':'
195  (
196         ('external:'   external=EBoolean)?
197     &   ('file:'       file=PATH)?
198     &   ('environment:' environment=EString)?
199     &   ('name:'       secret_name=EString)?
200  );
201
202 VolumeLabel returns VolumeLabel:
203  {VolumeLabel}
204  name=EString':' value=EString;
205
206
207 VolumeDriverOpt returns VolumeDriverOpt:
208  {VolumeDriverOpt}
209  name=EString':' value=EString;
210
211
212 NetworkLabel returns NetworkLabel:
213  {NetworkLabel}
214  name=LabelID':' value=EString;
215
216
217 NetworkDriverOpt returns NetworkDriverOpt:
218  {NetworkDriverOpt}
219  name=EString':' value=EString;
```

```
220
221 IPAM returns IPAM:
222  {IPAM}
223  (
224         ('driver:'   driver=EString)?
225     &   ('config:'   ('-'configs+=IPAMConfig)+ )?
226     &   ('options:'   (options+=IPAMOption)+ )?
227  );
228
229 IPAMConfig returns IPAMConfig:
230  {IPAMConfig}
231  (
232         ('subnet:'         subnet=NetworkAddress)?
233     &   ('ip_range:'       ip_range=NetworkAddress)?
234     &   ('gateway:'       gateway=DNS)?
235     &   ('aux_addresses:'    (aux_addresses+=IPAMAddress)+ )?
236  );
237
238 IPAMOption returns IPAMOption:
239  {IPAMOption}
240  name=EString':' value=EString;
241
242 IPAMAddress returns IPAMAddress:
243  {IPAMAddress}
244  name=EString':' dns=DNS;
245
246 Ports returns ecore::EString:
247  PORT_DEF | VERSION_INT | QUOTED_INT;
248
```

```
249 terminal PORT_DEF:
250  ('"'((((((('0'..'9')|('0'..'9')('0'..'9')|'1'('0'..'9')('0'..'9')|
251  '2'('0'..'5')('0'..'5'))'.'(('0'..'9')|('0'..'9')('0'..'9')|
252  '1'('0'..'9')('0'..'9')|'2'('0'..'5')('0'..'5'))'.'(('0'..'9')|
253  ('0'..'9')('0'..'9')|'1'('0'..'9')('0'..'9')|
254  '2'('0'..'5')('0'..'5'))'.'(('0'..'9')|('0'..'9')('0'..'9')|
255  '1'('0'..'9')('0'..'9')|'2'('0'..'5')('0'..'5'))':
256  ')?INT'-'INT':')?INT'-'INT('/'ID)?)|(((((('0'..'9')|
257  ('0'..'9')('0'..'9')|'1'('0'..'9')('0'..'9')|
258  '2'('0'..'5')('0'..'5'))'.'(('0'..'9')|
259  ('0'..'9')('0'..'9')|'1'('0'..'9')('0'..'9')|
260  '2'('0'..'5')('0'..'5'))'.'(('0'..'9')|
261  ('0'..'9')('0'..'9')|'1'('0'..'9')('0'..'9')|
262  '2'('0'..'5')('0'..'5'))'.'(('0'..'9')|
263  ('0'..'9')('0'..'9')|'1'('0'..'9')('0'..'9')|
264  '2'('0'..'5')('0'..'5'))':')?INT':')?INT('/'ID)?))'"');
265
266 terminal DEVICE_DEF:
267  '"'(((ID('.'ID)*|'.'|'..') ('/'(ID('.'ID)*|'.'|'..'))* '/'?) |
268  ('/' ((ID('.'ID)*|'.'|'..')'/')* (ID('.'ID)*|'.'|'..')?))':
269  '(((ID('.'ID)*|'.'|'..') ('/'(ID('.'ID)*|'.'|'..'))* '/'?) |
270  ('/' ((ID('.'ID)*|'.'|'..')'/')* (ID('.'ID)*|'.'|'..')?))
271  (':'PERMISSION)?'"';
272
273 terminal PERMISSION:
274  'r'|'w'|'m'|'rw'|'wr'|'rm'|'mr'|'mw'|'wm'|
275  'rwm'|'rmw'|'wrm'|'wmr'|'mrw'|'mwr';
276
277
```

```
278 PATH returns ecore::EString:
279  ((ID('.'ID)*|'.'|'..') ('/'(ID('.'ID)*|'.'|'..'))* '/'?) |
280  ('/' ((ID('.'ID)*|'.'|'..')'/')* (ID('.'ID)*|'.'|'..')?);
281
282 EString returns ecore::EString:
283  STRING | ID | '"''"' | PORT_DEF | VERSION_INT | QUOTED_INT |
284  VERSION_FLOAT | DEVICE_DEF;
285
286 LabelID returns ecore::EString:
287  ID('.'ID)*;
288
289 EInt returns ecore::EInt:
290  INT;
291
292 Command returns ecore::EString:
293  (EString|ANY_OTHER|PATH|EInt|'-')+;
294
295 terminal ID:
296  ('^')?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'-'|
297  '0'..'9')*;
298
299 dotID returns ecore::EString:
300  ID('.'ID)*;
301
302 DNS returns DNS:
303  dns1=EInt'.'dns2=EInt'.'dns3=EInt'.'dns4=EInt;
304
305 NetworkAddress returns NetworkAddress:
306  dns1=EInt'.'dns2=EInt'.'dns3=EInt'.'dns4=EInt'/'netId=EInt;
```

```
307
308 EBoolean returns ecore::EBoolean:
309     'true' | 'false';
310
311 enum RestartPolicy returns RestartPolicy:
312     no = '"no"' | always = 'always' | onfailure = 'on-failure' |
313     unlessstopped = 'unless-stopped';
314
315 enum AccessMode returns AccessMode:
316     rw = 'rw' | ro = 'ro' | z = 'z' ;
317
318 enum MountType returns MountType:
319     volume = 'volume' | bind = 'bind' |
320     tmpfs = 'tmpfs' | npipe = 'npipe';
321
322 enum PropagationType returns PropagationType:
323     rprivate = 'rprivate' | private = 'private' |
324     rshared = 'rshared' | shared = 'shared' |
325     rslave = 'rslave' | slave = 'slave';
326
327 enum Condition returns Condition:
328     service_started = 'service_started' |
329     service_healthy = 'service_healthy' |
330     service_completed_successfully =
331     'service_completed_successfully';
```

Apéndice C

Anexo 3: Formatter

En este anexo se incluye el fichero "DockerComposeFormatter.xtend", en el que se define el funcionamiento del *formatter* del editor textual.

```
1  package org.xtext.example.dockercompose.formatting2
2
3  import com.google.inject.Inject
4  import dockercompose.*
5  import org.eclipse.xtext.formatting2.AbstractFormatter2
6  import org.eclipse.xtext.formatting2.IFormattableDocument
7  import org.xtext.example.dockercompose.services
8                                          .DockerComposeGrammarAccess
9  class DockerComposeFormatter extends AbstractFormatter2 {
10  @Inject extension DockerComposeGrammarAccess
11
12  def dispatch void format(DockerCompose dockerCompose,
13                      extension IFormattableDocument document) {
14     for (service : dockerCompose.services) {
15       service.format
16     }
```

```
17      for (network : dockerCompose.networks) {
18        network.format
19      }
20      for (volume : dockerCompose.volumes) {
21        volume.format
22      }
23      for (config : dockerCompose.configs) {
24        config.format
25      }
26      for (secret : dockerCompose.secrets) {
27        secret.format
28      }
29      dockerCompose.regionFor.keyword("services:").prepend[newLine];
30      dockerCompose.regionFor.keyword("networks:").prepend[newLine];
31      dockerCompose.regionFor.keyword("volumes:").prepend[newLine];
32      dockerCompose.regionFor.keyword("configs:").prepend[newLine];
33      dockerCompose.regionFor.keyword("secrets:").prepend[newLine];
34  }
35
36  def dispatch void format(Service service,
37                           extension IFormattableDocument document) {
38      for (device : service.devices) {
39        device.format
40      }
41      for (dNS : service.dns) {
42        dNS.format
43      }
44      for (dependency : service.depends_on) {
45        dependency.format
```

```
46      }
47      for (volumeConnector : service.volumes) {
48        volumeConnector.format
49      }
50      for (configConnector : service.configs) {
51        configConnector.format
52      }
53      for (secretConnector : service.secrets) {
54        secretConnector.format
55      }
56      for (networkConnector : service.networks) {
57        networkConnector.format
58      }
59      for (port : service.ports) {
60        port.format
61      }
62      for (env : service.environment) {
63        env.format
64      }
65
66      service.prepend[space="\n\t"]
67      service.regionFor.keyword(":").prepend[noSpace];
68      service.regionFor.keyword("image:")
69                       .prepend[space="\n\t\t"].append[space=" "];
70      service.regionFor.keyword("build:")
71                       .prepend[space="\n\t\t"].append[space=" "];
72      service.regionFor.keyword("cpu_count:")
73                       .prepend[space="\n\t\t"].append[space=" "];
74      service.regionFor.keyword("command:")
```

```
75                      .prepend[space="\n\t\t"].append[space=" "];
76      service.regionFor.keyword("container_name:")
77                      .prepend[space="\n\t\t"].append[space=" "];
78      service.regionFor.keyword("restart:")
79                      .prepend[space="\n\t\t"].append[space=" "];
80      service.regionFor.keyword("init:")
81                      .prepend[space="\n\t\t"].append[space=" "];
82      service.regionFor.keyword("read_only:")
83                      .prepend[space="\n\t\t"].append[space=" "];
84      service.regionFor.keyword("environment:")
85                      .prepend[space="\n\t\t"];
86      service.regionFor.keyword("devices:").prepend[space="\n\t\t"];
87      service.regionFor.keyword("dns:").prepend[space="\n\t\t"];
88      service.regionFor.keyword("ports:").prepend[space="\n\t\t"];
89      service.regionFor.keyword("links:").prepend[space="\n\t\t"];
90      service.regionFor.keyword("depends_on:")
91                      .prepend[space="\n\t\t"];
92      service.regionFor.keyword("networks:")
93                      .prepend[space="\n\t\t"];
94      service.regionFor.keyword("volumes:").prepend[space="\n\t\t"];
95      service.regionFor.keyword("configs:").prepend[space="\n\t\t"];
96      service.regionFor.keyword("secrets:").prepend[space="\n\t\t"];
97
98      for (k : service.regionFor.keywords("-")) {
99        k.prepend[space="\n\t\t\t"].append[space=" "];
100     }
101 }
102
103 def dispatch void format(EnvironmentVariable env,
```

```
104                     extension IFormattableDocument document) {
105     env.prepend[space="\n\t\t\t"];
106     if (env.regionFor.keyword("-") !== null) {
107       if (env.regionFor.keyword("-").offset ==
108           env.allSemanticRegions.get(0).offset) {
109         env.regionFor.keyword("=")
110                      .prepend[noSpace].append[noSpace];
111         env.regionFor.keyword("-").append[space=" "];
112       }
113       else {
114         env.regionFor.keyword(":")
115                      .prepend[noSpace].append[space=" "];
116       }
117     }
118     else {
119       env.regionFor.keyword(":")
120                    .prepend[noSpace].append[space=" "];
121     }
122 }
123
124 def dispatch void format(Dependency dependency,
125                     extension IFormattableDocument document) {
126     dependency.prepend[space="\n\t\t\t"]
127     dependency.regionFor.keyword("-").append[space=" "];
128     dependency.regionFor.keyword("condition:")
129               .prepend[space="\n\t\t\t\t"].append[space=" "];
130 }
131
132 def dispatch void format(NetworkConnector connector,
```

```
133                     extension IFormattableDocument document) {
134     connector.prepend[space="\n\t\t\t"]
135     connector.regionFor.keyword(":").prepend[noSpace];
136     connector.regionFor.keyword("ipv4_address:")
137              .prepend[space="\n\t\t\t\t"].append[space=" "];
138     connector.regionFor.keyword("priority:")
139              .prepend[space="\n\t\t\t\t"].append[space=" "];
140     connector.regionFor.keyword("aliases:")
141                        .prepend[space="\n\t\t\t\t"];
142     connector.regionFor.keyword("link_local_ips:")
143                        .prepend[space="\n\t\t\t\t"];
144
145     for (k : connector.regionFor.keywords("-")) {
146       if (k.offset != connector.allSemanticRegions.get(0).offset){
147          k.prepend[space="\n\t\t\t\t\t"].append[space=" "];
148       }
149       else
150          k.append[space=" "]
151     }
152 }
153
154 def dispatch void format(VolumeConnector connector,
155                     extension IFormattableDocument document) {
156     if (connector.regionFor.keyword("source:") !== null) {
157       if (connector.regionFor.keyword("source:").offset !=
158           connector.allSemanticRegions.get(0).offset) {
159          connector.regionFor.keyword("source:")
160                               .prepend[space="\n\t\t\t "];
161       }
```

```
162     }
163     if (connector.regionFor.keyword("target:") !== null) {
164       if (connector.regionFor.keyword("target:").offset !=
165           connector.allSemanticRegions.get(0).offset) {
166         connector.regionFor.keyword("target:")
167                              .prepend[space="\n\t\t\t "];
168       }
169     }
170     if (connector.regionFor.keyword("type:") !== null) {
171       if (connector.regionFor.keyword("type:").offset !=
172           connector.allSemanticRegions.get(0).offset) {
173         connector.regionFor.keyword("type:")
174                              .prepend[space="\n\t\t\t "];
175       }
176     }
177     if (connector.regionFor.keyword("read_only:") !== null) {
178       if (connector.regionFor.keyword("read_only:").offset !=
179           connector.allSemanticRegions.get(0).offset) {
180         connector.regionFor.keyword("read_only:")
181                              .prepend[space="\n\t\t\t "];
182       }
183     }
184     if (connector.regionFor.keyword("bind:") !== null) {
185       if (connector.regionFor.keyword("bind:").offset !=
186           connector.allSemanticRegions.get(0).offset) {
187         connector.regionFor.keyword("bind:")
188                              .prepend[space=" "];
189       }
190       else
```

```
191         connector.regionFor.keyword("bind:")
192                             .prepend[space="\n\t\t\t "];
193     }
194     if (connector.regionFor.keyword("volume:") !== null) {
195       if (connector.regionFor.keyword("volume:").offset !=
196           connector.allSemanticRegions.get(0).offset) {
197         connector.regionFor.keyword("volume:")
198                             .prepend[space="\n\t\t\t "];
199       }
200     }
201     if (connector.regionFor.keyword("tmpfs:") !== null) {
202       if (connector.regionFor.keyword("tmpfs:").offset !=
203           connector.allSemanticRegions.get(0).offset) {
204         connector.regionFor.keyword("tmpfs:")
205                             .prepend[space="\n\t\t\t "];
206       }
207     }
208     connector.regionFor.keyword("source:").append[space=" "];
209     connector.regionFor.keyword("target:").append[space=" "];
210     connector.regionFor.keyword("type:").append[space=" "];
211     connector.regionFor.keyword("read_only:").append[space=" "];
212     connector.regionFor.keyword("propagation:")
213             .prepend[space="\n\t\t\t\t"].append[space=" "];
214     connector.regionFor.keyword("nocopy:")
215             .prepend[space="\n\t\t\t\t"].append[space=" "];
216     connector.regionFor.keyword("size:")
217             .prepend[space="\n\t\t\t\t"].append[space=" "];
218
219     for (k : connector.regionFor.keywords(":")) {
```

```
220       k.prepend[noSpace].append[noSpace];
221     }
222   }
223
224   def dispatch void format(ConfigConnector connector,
225                       extension IFormattableDocument document) {
226     if (connector.regionFor.keyword("source:") !== null) {
227       if (connector.regionFor.keyword("source:").offset !=
228           connector.allSemanticRegions.get(0).offset) {
229         connector.regionFor.keyword("source:")
230                               .prepend[space="\n\t\t\t "];
231       }
232     }
233     if (connector.regionFor.keyword("target:") !== null) {
234       if (connector.regionFor.keyword("target:").offset !=
235           connector.allSemanticRegions.get(0).offset) {
236         connector.regionFor.keyword("target:")
237                               .prepend[space="\n\t\t\t "];
238       }
239     }
240     if (connector.regionFor.keyword("uid:") !== null) {
241       if (connector.regionFor.keyword("uid:").offset !=
242           connector.allSemanticRegions.get(0).offset) {
243         connector.regionFor.keyword("uid:")
244                               .prepend[space="\n\t\t\t "];
245       }
246     }
247     if (connector.regionFor.keyword("gid:") !== null) {
248       if (connector.regionFor.keyword("gid:").offset !=
```

```
249             connector.allSemanticRegions.get(0).offset) {
250           connector.regionFor.keyword("gid:")
251                                .prepend[space="\n\t\t\t "];
252         }
253       }
254       if (connector.regionFor.keyword("mode:") !== null) {
255         if (connector.regionFor.keyword("mode:").offset !=
256             connector.allSemanticRegions.get(0).offset) {
257           connector.regionFor.keyword("mode:")
258                                .prepend[space="\n\t\t\t "];
259         }
260       }
261       connector.regionFor.keyword("source:").append[space=" "];
262       connector.regionFor.keyword("target:").append[space=" "];
263       connector.regionFor.keyword("uid:").append[space=" "];
264       connector.regionFor.keyword("gid:").append[space=" "];
265       connector.regionFor.keyword("mode:").append[space=" "];
266   }
267
268   def dispatch void format(SecretConnector connector,
269                            extension IFormattableDocument document) {
270
271       if (connector.regionFor.keyword("source:") !== null) {
272         if (connector.regionFor.keyword("source:").offset !=
273             connector.allSemanticRegions.get(0).offset) {
274           connector.regionFor.keyword("source:")
275                                .prepend[space="\n\t\t\t "];
276         }
277       }
```

```
278
279     if (connector.regionFor.keyword("target:") !== null) {
280       if (connector.regionFor.keyword("target:").offset !=
281           connector.allSemanticRegions.get(0).offset) {
282         connector.regionFor.keyword("target:")
283                             .prepend[space="\n\t\t\t "];
284       }
285     }
286
287     if (connector.regionFor.keyword("uid:") !== null) {
288       if (connector.regionFor.keyword("uid:").offset !=
289           connector.allSemanticRegions.get(0).offset) {
290         connector.regionFor.keyword("uid:")
291                             .prepend[space="\n\t\t\t "];
292       }
293     }
294
295     if (connector.regionFor.keyword("gid:") !== null) {
296       if (connector.regionFor.keyword("gid:").offset !=
297           connector.allSemanticRegions.get(0).offset) {
298         connector.regionFor.keyword("gid:")
299                             .prepend[space="\n\t\t\t "];
300       }
301     }
302
303     if (connector.regionFor.keyword("mode:") !== null) {
304       if (connector.regionFor.keyword("mode:").offset !=
305           connector.allSemanticRegions.get(0).offset) {
306         connector.regionFor.keyword("mode:")
```

```
307                                 .prepend[space="\n\t\t\t "];
308         }
309       }
310
311       connector.regionFor.keyword("source:").append[space=" "];
312       connector.regionFor.keyword("target:").append[space=" "];
313       connector.regionFor.keyword("uid:").append[space=" "];
314       connector.regionFor.keyword("gid:").append[space=" "];
315       connector.regionFor.keyword("mode:").append[space=" "];
316   }
317
318   def dispatch void format(Network network,
319                            extension IFormattableDocument document) {
320       network.prepend[space="\n\t"]
321       network.regionFor.keyword(":").prepend[noSpace];
322       network.regionFor.keyword("driver:")
323                            .prepend[space="\n\t\t"].append[space=" "];
324       network.regionFor.keyword("attachable:")
325                            .prepend[space="\n\t\t"].append[space=" "];
326       network.regionFor.keyword("enable_ipv6:")
327                            .prepend[space="\n\t\t"].append[space=" "];
328       network.regionFor.keyword("internal:")
329                            .prepend[space="\n\t\t"].append[space=" "];
330       network.regionFor.keyword("external:")
331                            .prepend[space="\n\t\t"].append[space=" "];
332       network.regionFor.keyword("name:")
333                            .prepend[space="\n\t\t"].append[space=" "];
334       network.regionFor.keyword("labels:").prepend[space="\n\t\t"];
335       network.regionFor.keyword("driver_opts:")
```

```
336                     .prepend[space="\n\t\t"];
337     network.regionFor.keyword("ipam:").prepend[space="\n\t\t"];
338     for (label : network.labels) {
339       label.format
340     }
341     for (driver_opt : network.driver_opts) {
342       driver_opt.format
343     }
344     network.ipam.format
345   }
346
347   def dispatch void format(NetworkLabel label,
348                       extension IFormattableDocument document) {
349     label.prepend[space="\n\t\t\t"]
350     label.regionFor.keyword(":")
351                    .prepend[noSpace].append[space=" "];
352   }
353
354   def dispatch void format(NetworkDriverOpt opt,
355                       extension IFormattableDocument document) {
356     opt.prepend[space="\n\t\t\t"]
357     opt.regionFor.keyword(":").prepend[noSpace].append[space=" "];
358   }
359
360   def dispatch void format(IPAM ipam,
361                       extension IFormattableDocument document) {
362     ipam.regionFor.keyword("driver:")
363                   .prepend[space="\n\t\t\t"].append[space=" "];
364     ipam.regionFor.keyword("config:").prepend[space="\n\t\t\t"];
```

```
365     ipam.regionFor.keyword("options:").prepend[space="\n\t\t\t"];
366
367     for (k : ipam.regionFor.keywords("-")) {
368       k.prepend[space="\n\t\t\t\t"].append[space=" "];
369     }
370     for (configs : ipam.configs) {
371       configs.format
372     }
373     for (opt : ipam.options) {
374       opt.format
375     }
376   }
377
378   def dispatch void format(IPAMOption opt,
379                       extension IFormattableDocument document) {
380     opt.prepend[space="\n\t\t\t\t"]
381     opt.regionFor.keyword(":").prepend[noSpace].append[space=" "];
382   }
383
384   def dispatch void format(IPAMConfig config,
385                       extension IFormattableDocument document) {
386
387     if (config.regionFor.keyword("subnet:") !== null) {
388       if (config.regionFor.keyword("subnet:").offset !=
389           config.allSemanticRegions.get(0).offset) {
390         config.regionFor.keyword("subnet:")
391                           .prepend[space="\n\t\t\t\t "];
392       }
393     }
```

```
394
395     if (config.regionFor.keyword("ip_range:") !== null) {
396       if (config.regionFor.keyword("ip_range:").offset !=
397           config.allSemanticRegions.get(0).offset) {
398         config.regionFor.keyword("ip_range:")
399                         .prepend[space="\n\t\t\t\t "];
400       }
401     }
402
403     if (config.regionFor.keyword("gateway:") !== null) {
404       if (config.regionFor.keyword("gateway:").offset !=
405           config.allSemanticRegions.get(0).offset) {
406         config.regionFor.keyword("gateway:")
407                         .prepend[space="\n\t\t\t\t "];
408       }
409     }
410
411     if (config.regionFor.keyword("aux_addresses:") !== null) {
412       if (config.regionFor.keyword("aux_addresses:").offset !=
413           config.allSemanticRegions.get(0).offset) {
414         config.regionFor.keyword("aux_addresses:")
415                         .prepend[space="\n\t\t\t\t "];
416       }
417     }
418     for (address : config.aux_addresses) {
419       address.format
420     }
421   }
422
```

```
423  def dispatch void format(IPAMAddress address,
424                           extension IFormattableDocument document) {
425     address.prepend[space="\n\t\t\t\t\t"]
426     address.regionFor.keyword(":")
427                           .prepend[noSpace].append[space=" "];
428  }
429
430  def dispatch void format(Volume volume,
431                           extension IFormattableDocument document) {
432     volume.prepend[space="\n\t"]
433     volume.regionFor.keyword(":").prepend[noSpace];
434     volume.regionFor.keyword("external:")
435                          .prepend[space="\n\t\t"].append[space=" "];
436     volume.regionFor.keyword("driver:")
437                          .prepend[space="\n\t\t"].append[space=" "];
438     volume.regionFor.keyword("name:")
439                          .prepend[space="\n\t\t"].append[space=" "];
440     volume.regionFor.keyword("labels:").prepend[space="\n\t\t"];
441     volume.regionFor.keyword("driver_opts:")
442                          .prepend[space="\n\t\t"];
443     for (label : volume.labels) {
444       label.format
445     }
446     for (driver_opt : volume.driver_opts) {
447       driver_opt.format
448     }
449  }
450
451  def dispatch void format(VolumeLabel label,
```

```
452                     extension IFormattableDocument document) {
453     label.prepend[space="\n\t\t\t"]
454     label.regionFor.keyword(":")
455                    .prepend[noSpace].append[space=" "];
456 }
457
458 def dispatch void format(VolumeDriverOpt opt,
459                     extension IFormattableDocument document) {
460     opt.prepend[space="\n\t\t\t"]
461     opt.regionFor.keyword(":").prepend[noSpace].append[space=" "];
462 }
463
464 def dispatch void format(Config config,
465                     extension IFormattableDocument document) {
466     config.prepend[space="\n\t"]
467     config.regionFor.keyword(":").prepend[noSpace];
468     config.regionFor.keyword("external:")
469                    .prepend[space="\n\t\t"].append[space=" "];
470     config.regionFor.keyword("file:")
471                    .prepend[space="\n\t\t"].append[space=" "];
472     config.regionFor.keyword("name:")
473                    .prepend[space="\n\t\t"].append[space=" "];
474 }
475
476 def dispatch void format(Secret secret,
477                     extension IFormattableDocument document) {
478     secret.prepend[space="\n\t"]
479     secret.regionFor.keyword(":").prepend[noSpace];
480     secret.regionFor.keyword("external:")
```

```
481                     .prepend[space="\n\t\t"].append[space=" "];
482     secret.regionFor.keyword("file:")
483                     .prepend[space="\n\t\t"].append[space=" "];
484     secret.regionFor.keyword("name:")
485                     .prepend[space="\n\t\t"].append[space=" "];
486     secret.regionFor.keyword("environment:")
487                     .prepend[space="\n\t\t"].append[space=" "];
488  }
489 }
```

Apéndice D

Anexo 4: Manual de usuario

En este anexo se indican, a modo de manual, las principales funcinalidades que ofrece el sistema y los pasos que el usuario debe seguir para poder utilizarlas correctamente.

El código desarrollado se encuentra. en su totalidad, en el repositorio CML de *GitHub* [11]. Este debe descargarse como fichero comprimido con la opción "Download Zip".

Una vez hecho esto, el código debe importarse en el programa Obeo Designer. Para ello, se utiliza la opción "File" $\rightarrow$ "Import..." $\rightarrow$ "Existing Projects into Workspace", que abrirá una ventana en la que se deben seleccionar los proyectos a importar. En esta, se ha de especificar, en la opción "Select archive file", la ruta del fichero comprimido descargado. Posteriormente, se mostrará un listado con los proyectos encontrados en el archivo. Para importarlos se deben seleccionar todos ellos y pulsar el botón "Finish".

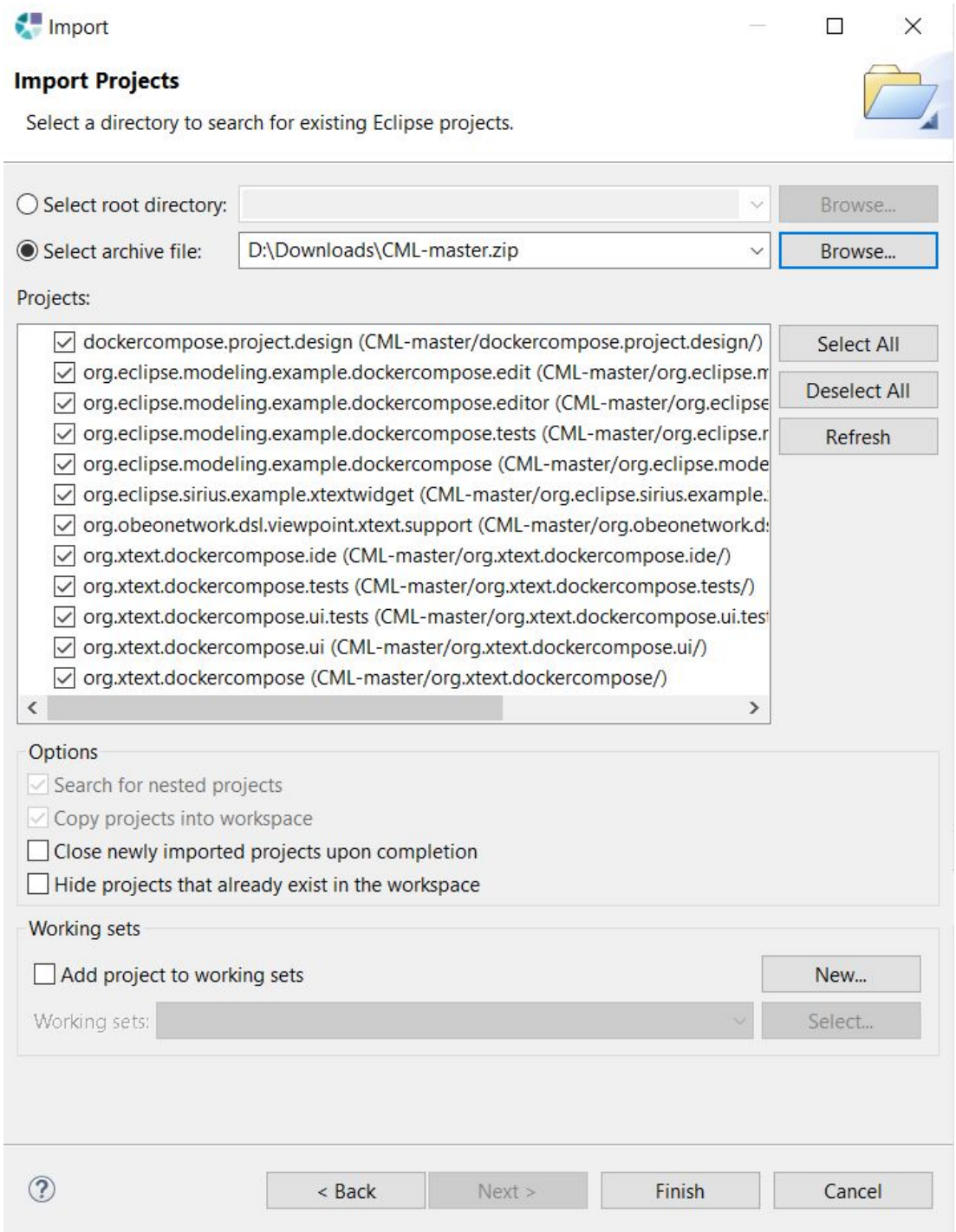

Figura D.1: Proyectos a importar en Obeo Designer

Tras importar el código, todos los proyectos aparecerán en la pestaña “Model Explorer”. Para poder utilizar el tanto editor gráfico desarrollado como el de texto , debe arrancarse otra instancia de Obeo Designer. Para ello, se selecciona

la opción “Run As” → “Eclipse Application” del menú contextual del proyecto “org.xtext.dockercompose”. Así, se abrirá una nueva ventana de Obeo Designer, con un espacio de trabajo en el que estarán cargados todos los plugins necesarios para el funcionamiento de los editores.

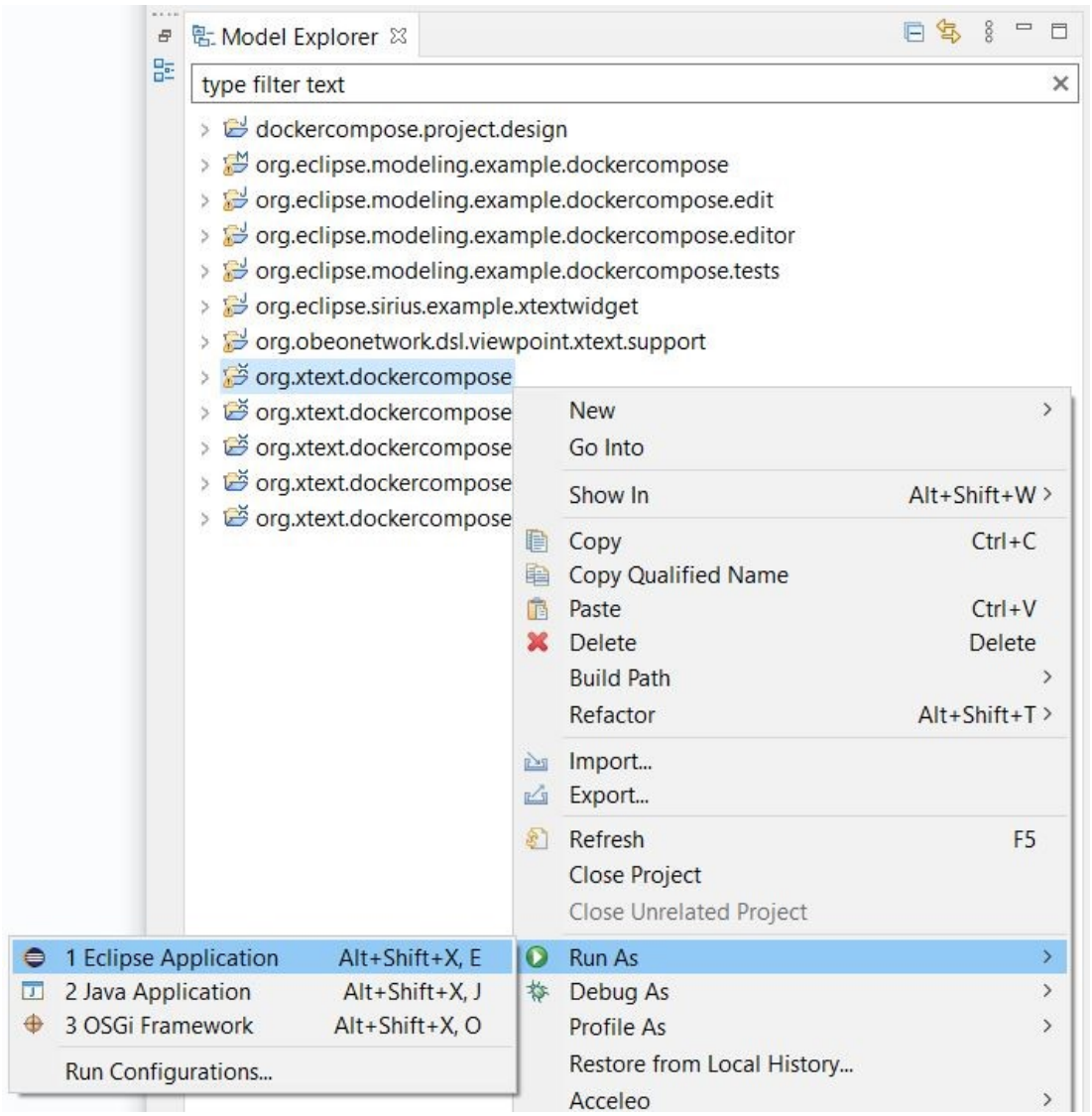

Figura D.2: Designer

En la segunda ventana de Obeo Designer debe crearse un proyecto de modelado con la opción “New” → “Mdeling Project”. Esto abrirá una ventana en la que, únicamente, ha de especificarse el nombre del nuevo proyecto. Para poder utilizar el editor de la gramática textual desarrollada, debe incluirse un fichero de extensión .yaml en el

proyecto, para lo cual existen dos posibilidades.

Si se desea crear un archivo desde cero y comenzar a definir los elementos necesarios para una aplicación que se quiere desarrollar, debe utilizarse la opción “New” → “File” del menú contextual del proyecto y especificar un nombre para el fichero que va a crearse. Basta con que el archivo tenga extensión .yaml para que este se abra por defecto con el editor de texto desarrollado, en el que podrá definirse un modelo utilizando la gramática definida en el apartado 3.2.2.1. Además, las palabras reservadas del lenguaje se resaltarán, se marcarán errores siempre que no se sigan las reglas de la gramática y se ofrecerá la posibilidad de formatear el contenido del archivo de texto con la combinación de teclas *Ctrl* + *Shift* + *F*.

Si, por el contrario, se desea utilizar el editor con un fichero Docker-Compose ya definido, este puede simplemente copiarse y pegarse dentro del proyecto, lo que permitirá editarlo utilizando el editor de texto desarrollado siempre y cuando su extensión sea .yaml.

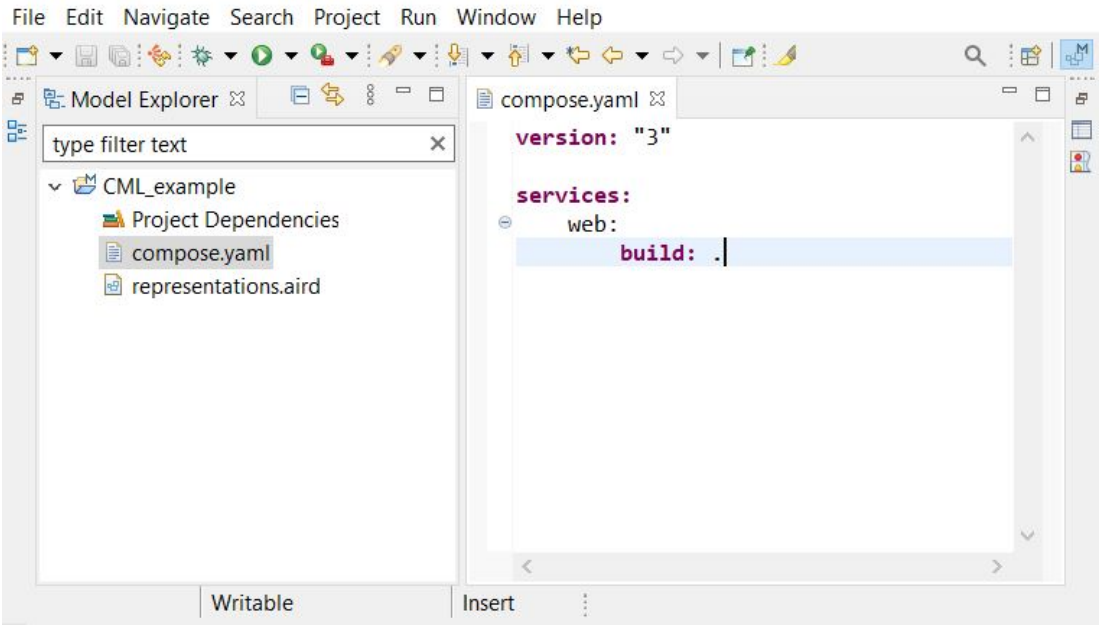

Figura D.3: Editor de texto de Docker-Compose

Para utilizar poder editar el modelo mediante un diagrama, ha de generarse la

representación gráfica de un modelo Docker-Compose definido en un fichero de texto. Para ello, debe utilizarse la opción “New Representation” → “Other” del menú contextual del elemento raíz del archivo de extensión .yaml, al que puede accederse expandiendo el fichero de texto desde la pestaña “Model Explorer”. Tras esto, se seleccionará la opción “Docker-Compose Diagram” y se especificará un nombre para la representación gráfica.

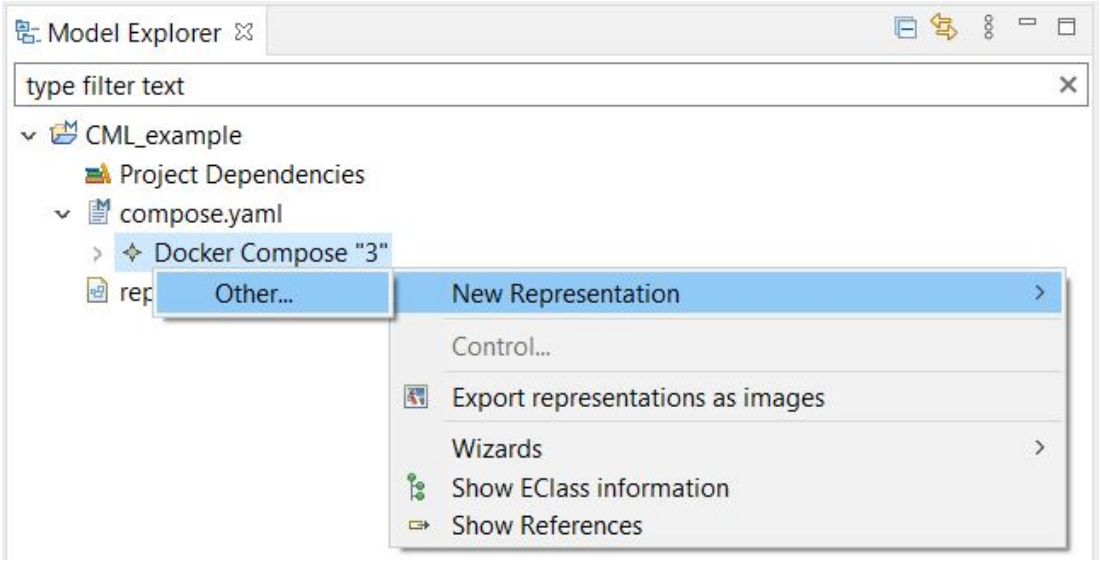

Figura D.4: Generación de un diagrama a partir de un modelo textual

Posteriormente, el nuevo diagrama se abrirá con el editor gráfico desarrollado, en el que podrá editarse utilizando la notación gráfica definida en el apartado 3.2.2.2 y las herramientas definidas. Tanto la representación gráfica como la textual se mantendrán sincronizadas, pudiendo utilizar los dos editores para configurar el modelo como se desee.

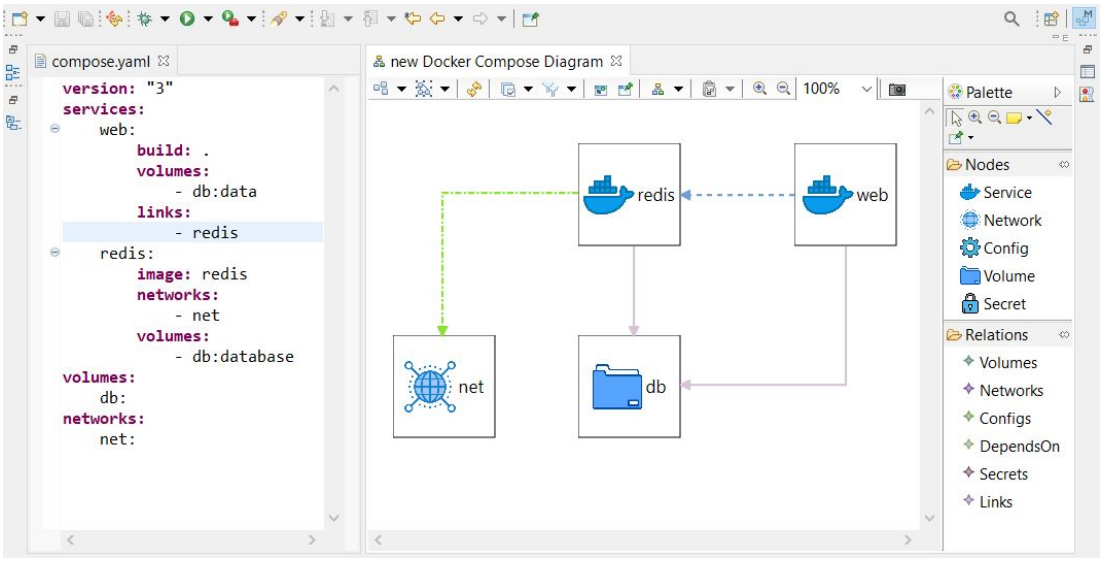

Figura D.5: Representación textual y gráfica de un modelo Docker-Compose

Una opción adicional que permite el editor gráfico es la de mostrar únicamente las capas deseadas de un diagrama. Existe una capa por cada concepto principal del lenguaje de Docker-Compose, las cuales han sido definidas durante la implementación de la notación gráfica. La opción mostrada en la figura D.6 muestra un listado con las capas que pueden seleccionarse o deseleccionarse en función de lo que se quiera ocultar en el diagrama.

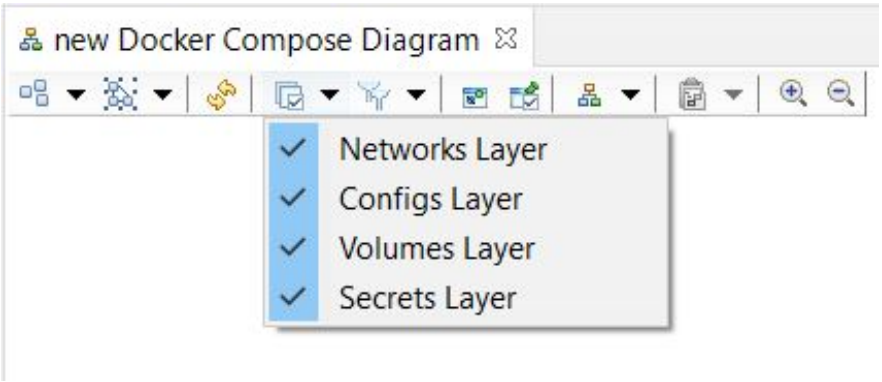

Figura D.6: Listado de capas que pueden ocultarse o mostrarse en un diagrama

Por ejemplo, si se decide deseleccionar la capa *Volumes*, el diagrama generado con el modelo mostrado en la figura D.5 quedará como el que se muestra en la figura D.7,

ocultando tanto los nodos de tipo *Volume* como las aristas que relacionan algún nodo de este tipo con cualquier otro. Dado que únicamente se están ocultando esos elementos, estos no son eliminados de la representación textual del modelo.

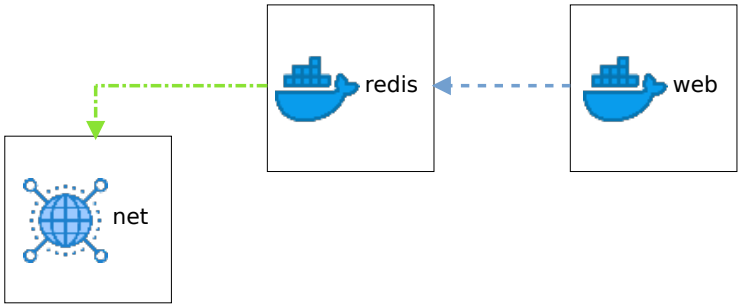

Figura D.7: Diagrama generado a partir de un modelo ocultando la capa Volumes

Bibliografía

[1] Best YAML Validator Online. URL: https://jsonformatter.org/yaml-validator/cd1cfb.

[2] CodeBeautify - docker-compose. URL: https://codebeautify.org/yaml-validator/cbccd63a.

[3] Docker. URL: https://www.docker.com/.

[4] docker compose convert — Docker Documentation. URL: https://docs.docker.com/engine/reference/commandline/compose_convert/.

[5] Docker Hub Container Image Library. URL: https://hub.docker.com/.

[6] Docker Swarm. URL: https://www.sumologic.com/glossary/docker-swarm/.

[7] GitHub - serviceprototypinglab/dcvalidator: Validation and quality check of Docker Compose files. URL: https://github.com/serviceprototypinglab/dcvalidator.

[8] Historia de la evolución de la tecnología de contenedor de Linux - programador clic. URL: https://programmerclick.com/article/73172079548/.

[9] Kubernetes. URL: https://kubernetes.io/.

[10] 8 surprising facts about real Docker adoption — Datadog, 2018. URL: https://www.datadoghq.com/docker-adoption/.

[11] Lorenzo Gabriel Ceballos Bru. CML GitHub repository. URL: https://github.com/elpiter15/CML.

[12] Jordi Cabot. The Ultimate Object Constraint Language (OCL) tutorial. URL: https://modeling-languages.com/ocl-tutorial/.

[13] Obeo Designer. Define your own Modeling Tools. URL: https://www.obeodesigner.com/en/product.

[14] Obeo Designer. Eclipse Sirius. URL: https://www.obeodesigner.com/en/product/sirius.

[15] Docker. Awesome-compose sample apps. URL: https://github.com/docker/awesome-compose/blob/master/spring-postgres/compose.yaml.

[16] Docker. Docker-Compose Documentation. URL: https://docs.docker.com/compose/compose-file/.

[17] Eclipse. Eclipse Modeling Framework (EMF). URL: https://www.eclipse.org/modeling/emf/.

[18] Eclipse. OCLinEcore tutorial. URL: https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.ocl.doc%2Fhelp%2FTutorials.html.

[19] Eclipse. OCL/OCLinEcore. URL: https://wiki.eclipse.org/OCL/OCLinEcore#OCLinEcore_Editor.

[20] Eclipse. What is Sirius? URL: eclipse.org/community/eclipse_newsletter/2013/november/article1.php.

[21] Eclipse. Xtext. URL: https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html.

[22] J. García, F.O. García, V. Pelechano, A. Vallecillo, J.M. Vara, and C. Vicente-Chicote. *Desarrollo de Software Dirigido por Modelos*. 2013. URL: http://www.lcc.uma.es/~av/Publicaciones/12/LibroDSDM.pdf.

[23] OMG (Object Management Group). Object Constraint Language. URL: https://www.omg.org/spec/OCL/2.4/PDF.

[24] IONOS. Docker, la conocida tecnología de virtualización de contenedores - IONOS, 2022. URL: https://www.ionos.es/digitalguide/servidores/know-how/que-es-docker/.

[25] Dominik Jetzen. The new Formatter API for Xtext 2.8. URL: https://blogs.itemis.com/en/tabular-formatting-with-the-new-formatter-api.

[26] ObeoNetwork. Xtext-Sirius-integration GitHub repository. URL: https://github.com/ObeoNetwork/Xtext-Sirius-integration/tree/master/xtext-support-parent/examples/org.eclipse.sirius.example.fowlerdsl.xtextwidget/src/org/eclipse/sirius/example/fowlerdsl/xtextwidget.

[27] Rani Osnat. A Brief History of Containers From the 1970s Till Now, 2020. URL: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016.

[28] RedHat. ¿Qué es YAML? YAML y su uso en la automatización. URL: https://www.redhat.com/es/topics/automation/what-is-yaml.

[29] TXEMA RODRÍGUEZ. De Docker a Kubernetes: entendiendo qué son los contenedores y por qué es una de las mayores revoluciones de la industria del desarrollo., 2019. URL: https://www.xataka.com/otros/docker-a-kubernetes-entendiendo-que-contenedores-que-mayores-//revoluciones-industria-desarrollo.

[30] Lars Vogel. Eclipse Modeling Framework (EMF) - Tutorial. URL: https://www.vogella.com/tutorials/EclipseEMF/article.html.