Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación
Mención en Telemática

Entorno de simulación de redes distribuido basado en ns-3 y computación en nube con virtualización basada en contenedores

Autor
D. Pablo García Zarza

Tutor
Dr. D. Manuel Rodríguez Cayetano
D. Sergio Serrano Iglesias

Valladolid, 12 de julio de 2019

Título:	**Entorno de simulación de redes distribuido basado en ns-3 y computación en nube con virtualización basada en contenedores**
Autor:	**D. Pablo García Zarza**
Tutor:	**Dr. D. Manuel Rodríguez Cayetano** **D. Sergio Serrano Iglesias**
Departamento:	**Tratamiento de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

Presidente:	**Dr. D. Juan I. Asensio Pérez**
Vocal:	**Dr. D. Miguel L. Bote Lorenzo**
Secretario:	**Dr. D. Federico Simmross Wattenberg**

Fecha:	**16 de julio de 2019**
Calificación:	

Resumen

La simulación de redes en entornos académicos es útil para que los alumnos comprueben los conocimientos adquiridos. Sin embargo, el tiempo necesario para llevar a cabo las simulaciones puede llegar a ser elevado, por lo que es necesario desarrollar un sistema que permita reducir el tiempo de ejecución. En un trabajo anterior, denominado DNSE3, se había desarrollado un sistema que permitía distribuir las simulaciones en varias máquinas virtuales sobre la arquitectura de OpenStack. Para adecuar los recursos utilizados a la demanda actual de trabajos, el sistema determinaba el número de máquinas virtuales activas que necesitaba en cada instante, proceso que repercute en el rendimiento global de la aplicación debido a la activación y desactivación de dichas máquinas virtuales. El objetivo principal de este Trabajo Fin de Grado fue reformar DNSE3 para conseguir mejorar el rendimiento de dicha versión. Para ello, las simulaciones pasan a ejecutarse de máquinas virtuales a contenedores. Un contenedor es una unidad estándar de software que contiene todo lo que necesita un programa y, al contrario que una máquina virtual, se ejecuta directamente sobre el sistema operativo de la máquina anfitriona, por lo que la sobrecarga de arrancar o detener nuevos contenedores es menor que la de las máquinas virtuales. De esta manera, y debido a las limitaciones existentes en las herramientas proporcionadas por OpenStack para la gestión de contenedores, se diseña una arquitectura basada en un clúster formado por un número variable de máquinas virtuales donde se ejecutan un número variable de contenedores. Para conseguir un clúster de estas características, el servicio encargado de escalar el número de máquinas virtuales y contenedores se realiza a medida para que exista un equilibrio entre rendimiento y utilización

de los recursos disponibles. Una vez obtenidos los resultados con esta nueva arquitectura se comparan con los que se obtuvieron con la arquitectura basada en máquinas virtuales y se comprueba que el rendimiento mejora cuando el número de simulaciones a completar es bajo o medio. Esta mejora es debida a que el tiempo necesario para escalar el número de réplicas que ejecutan las simulaciones pasa de minutos a segundos. Este hecho favorece que el alumno pueda obtener los resultados de las simulaciones en un tiempo menor, lo que es beneficioso para el mismo debido a que podrá realizar un mayor número de simulaciones de diferente configuración en un tiempo más bajo.

Palabras clave

Simulación, contenedor, clúster, máquina virtual, nube computacional.

Abstract

Network simulation in academic environments is useful for students to test their knowledge. However, the time required to complete simulations can be high, so it is necessary to develop a system that allows shorter execution times. A previous work, called DNSE3, parallelized simulations in several virtual machines over the OpenStack platform. This system adapted the number of virtual machines to the number of unfinished jobs, improving the performance of the application. The main objective of this Graduate Thesis is to redesign the current version of DNSE3 to improve the performance, using containers instead of virtual machines. A container is a standard software unit that has everything an application needs to run, and that is executed directly over the kernel of the host. Due to limitations of OpenStack projects for container management, a cluster-based architecture is developed consisting of several virtual machines where it is executed a variable number of containers. For this reason, a service is developed that allows to scale the number of virtual machines and containers. This way, the service meets a trade off between performance and use of resources of the private computing cloud. With the new version, DNSE3 gets better results when the system must do a low or medium number of simulations. This improvement is due to the time that the system needs to scale containers is lower than what was needed with the older version of the application. This level of performance would allow students to get the results of the simulations in a shorter time, which means that they will be able to carry out a greater number of simulations of different configuration in the same time.

Keywords

Simulation, container, cluster, virtual machine, computational cloud.

Agradecimientos

Son muchas personas a las que tengo que agradecer el apoyo que me han proporcionado para conseguir llegar hasta aquí. En primer lugar, gracias Manuel por ofrecerme la oportunidad de continuar con este proyecto y gracias Sergio por echarme una mano con el mismo cada vez que me quedaba bloqueado.

Gracias a Eduardo, Miguel y Juan Ignacio por guiarme en el desarrollo de este proyecto. Sin vuestros consejos no habría sido posible conseguir los resultados obtenidos.

Gracias a todos mis compañeros de carrera. Sin el apoyo que nos proporcionamos, además de los buenos ratos que pasamos, el camino que he tenido que recorrer hasta llegar aquí habría sido mucho más duro.

Para finalizar quiero agradecer el apoyo de mi familia. Sin vuestro apoyo y compresión habría sido imposible llegar hasta aquí. Simplemente gracias.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Las simulaciones se basan en modelos matemáticos que imitan el comportamiento esperado de un sistema real [1]. Con las simulaciones se consigue evitar los costes derivados de la construcción y mantenimiento de los sistemas reales. Además, los modelos en los que se basan las simulaciones son descritos mediante programas informáticos, por lo que se pueden modificar y adaptar a nuevas situaciones. Más concretamente, la simulación de redes de comunicaciones en entornos académicos es útil para que los alumnos afiancen los conocimientos que se les han explicado durante las distintas clases teóricas y puedan comprobar, y entender, cómo afecta la variación de los distintos parámetros que intervienen en el funcionamiento de las comunicaciones al rendimiento global de las mismas [2]. Algunos de los simuladores utilizados en entornos académicos son *ns-3*[1] y *Riverded Modeler*[2].

La utilización de simuladores de redes en entornos académicos acarrea dos problemas: el primero de ellos es que el alumno necesita un tiempo de adaptación para aprender a utilizar el simulador; el segundo problema es que el propio simulador necesita un tiempo para llevar a cabo las operaciones oportunas con el fin de imitar el comportamiento del modelo simulado, por lo que los usuarios tienen que esperar un tiempo no despreciable hasta que obtienen los resultados [3]. Además, si los alumnos desean comprender cómo afecta la variación del valor de uno de los parámetros de la red simulada al funcionamiento de esta, deberán ejecutar la simulación varias veces con los posibles valores que dicho parámetro pueda tener, por lo que el tiempo necesario para obtener todos los resultados y comenzar a trabajar con ellos se incrementa aún más.

Dentro de los planes de estudio actuales del Grado en Ingeniería de Tecnologías Específicas de Telecomunicación[3] y del Grado en Ingeniería de Tecnologías de Telecomunicación[4] de la Universidad de Valladolid se imparten, respectivamente, las asignaturas *Teletráfico* e *Ingeniería de Tráfico en Redes Telemáticas*. Estas asignaturas utilizan el simulador de eventos discretos *ns-3* como herramienta para que los alumnos comprueben si han adquirido los conocimientos teóricos impartidos en dichas asignaturas. Para solventar los problemas previamente mencionados en la utilización de simuladores, se diseñó la aplicación DNSE3 (*Distributed Network Simulation Environment 3*) [4], una aplicación de simulación que hace uso de la nube computacional para procesar de forma distribuida las simulaciones con el ob-

[1] https://www.nsnam.org/
[2] https://www.riverbed.com/gb/products/steelcentral/steelcentral-riverbed-modeler.html
[3] https://www.tel.uva.es/docencia/grados/itesp.htm
[4] https://www.tel.uva.es/docencia/grados/itec.htm

jetivo de reducir los tiempos de ejecución. Dicho entorno proporciona una interfaz Web en la que se puede elegir el modelo que se va a ejecutar, el valor de los parámetros con los que se va a ejecutar la simulación y el número de repeticiones que se desea ejecutar. Con esta interfaz Web el alumno utiliza de manera transparente el simulador *ns-3* sin necesidad de destinar parte de su tiempo a aprender a utilizarlo, lo que solventa el primero de los problemas citados en el párrafo previo.

Para abordar el segundo de los problemas mencionados, el referente al tiempo que necesita el propio simulador en completar la ejecución del modelo simulado, distintos autores han buscado distribuir dichas ejecuciones sobre una nube computacional [2, 5, 6], debido a que, aunque las distintas simulaciones de un mismo trabajo provengan de un mismo modelo de ejecución, los resultados obtenidos son independientes entre sí. La computación en la nube se refiere tanto al *software* como servicio accesible mediante Internet, como al *hardware* ubicado en *data centers* que permite proveer dicho servicio. En función de quién es el propietario de los recursos *hardware* se puede clasificar las distintas nubes de computación en nubes privadas, nubes públicas y nubes híbridas. Para aprovechar las ventajas de paralelizar las simulaciones, la implementación de DNSE3 se ejecuta sobre una nube privada cuyos recursos son gestionados por OpenStack[5], una plataforma de código abierto que permite controlar un número elevando de recursos como un servicio de manera remota. OpenStack está dividido en servicios que permiten agregar nuevas funcionalidades a la nube de cómputo [7]. En [4] se comprobó que al distribuir las simulaciones sobre dicha nube de computo, el tiempo necesario para obtener los resultados se reducía significativamente.

Como se ha comentado, la aplicación DNSE3 hace uso de una arquitectura formada por un grupo de máquinas virtuales para ejecutar las simulaciones que le soliciten los usuarios. El número de máquinas virtuales es variable para evitar desaprovechar recursos de la nube computacional privada sobre la que se está ejecutando. En [4] se propuso que este número esté directamente relacionado con la cantidad de simulaciones que los usuarios han solicitado y el sistema aún no ha ejecutado. Por ello, el cálculo del número de máquinas activas se define mediante una política de escalado. La diferencia entre las distintas políticas de escalado radica en la cantidad de máquinas virtuales que puede llegar a activar el sistema cada vez que se evalúa la política. Se comprobó que, cuanto mayor fuera el número de máquinas que el sistema podía activar, el tiempo que se necesitaba para finalizar un número de simulaciones fijo disminuía [4]. Sin embargo, también se comprobó que existía una diferencia sustancial en los tiempos de finalización un trabajo cuando las máquinas virtuales ya estaban lanzadas, frente a cuando se tenían que iniciar para completar dichos trabajos. Esta diferencia de tiempos es debida al coste computacional que supone iniciar una nueva máquina virtual (desde aprovisionar los recursos *hardware*, hasta iniciar el propio sistema operativo). También se comprobó que al disminuir el número de máquinas virtuales cuando el número de simulaciones desciende, existe un tiempo en el que los recursos están aprovisionados, pero no se utilizan, por lo que se desaprovechan. Por estos motivos, tanto cuando se incrementa como cuando se decrementa el número de máquinas virtuales, es necesario tener en cuenta los retardos provocados por el escalado de las máquinas virtuales en el rendimiento global de la aplicación, por lo que puede llegar a ser necesario utilizar políticas muy agresivas para obtener tiempos de simulación razonables, como el caso de iniciar una nueva máquina virtual aun cuando no se vaya a utilizar en ese instante, desaprovechando los recursos que tiene reservado, puesto que si los recursos han sido aprovisionados por el gestor de la nube de computo, no se pueden utilizar para iniciar otra instancia hasta que no se liberen.

Una vez que se ha determinado que el tiempo necesario para iniciar o finalizar máquinas virtuales tiene un impacto significativo en el rendimiento global del sistema, y que este

[5] https://www.openstack.org/

impacto afecta directamente a la política de escalado utilizada, es necesario explorar nuevos caminos para aprovechar, de manera más eficiente, los beneficios que ofrece la nube computacional. Así pues, este Trabajo Fin de Grado (TFG) se inicia con la voluntad de mejorar el rendimiento que ofrece la versión actual de la aplicación DNSE3 [2]. Por ello, se busca una alternativa al uso de máquinas virtuales como base de ejecución del servicio encargado de llevar a cabo las distintas simulaciones que soliciten los alumnos. La alternativa que se va a utilizar son los contenedores, unidad mínima de software que incluye todas las dependencias que un programa necesita para su ejecución [8]. En teoría, los contenedores se inician y finalizan de manera más rápida que las máquinas virtuales debido a que los primeros se ejecutan sobre los segundos, con lo que no tiene que iniciarse un sistema completo por contenedor. Gracias a ello, se solventaría el problema de retardos provocados por el escalado del servicio encargado de ejecutar las simulaciones. Así pues, se portará dicho servicio de una arquitectura basada en máquinas virtuales a otra mixta compuesta por un número pequeño de máquinas virtuales donde se ejecutarán los contenedores. Al reducir el número total de máquinas virtuales que puede llegar a tener el sistema, se consigue que los retardos provocados por su activación y destrucción disminuyan.

1.2. Objetivos

Con el objetivo de reducir los retardos derivados de la activación de cada una de las máquinas virtuales sobre las que se ejecutan las distintas simulaciones y de liberar de manera eficiente los recursos *hardware* que están aprovisionados pero no se están utilizando, se decidió buscar una alternativa que mitigara este problema. La solución que se decidió utilizar es la tecnología de contenedores. Un contenedor es una unidad mínima de software que incluye todas las dependencias que un programa necesita para su ejecución [8]. Los contenedores se ejecutan sobre el sistema operativo de la máquina anfitriona, por lo que su tiempo de despliegue debería ser inferior que el que necesita una máquina virtual para estar disponible para ejecutar el servicio de simulación, debido a que esta segunda tecnología tiene que *reservar* los recursos *hardware* que la máquina virtual vaya a utilizar, cargar la imagen del sistema operativo elegido e iniciarlo. Por este motivo, el objetivo principal de este Trabajo Fin de Grado será **determinar si una arquitectura basada en contenedores**, sobre la que se ejecutarían las diferentes simulaciones, **mejoraría el rendimiento global de la aplicación**. Para ello se deberá modificar la arquitectura actual [2] para aprovechar los beneficios de esta nueva tecnología. Una vez que se haya adecuado la aplicación a esta nueva arquitectura, se deberá comprobar el rendimiento global de la aplicación y comparar los nuevos resultados con los que se obtuvieron con la versión anterior de la aplicación [4] para determinar si el uso de la tecnología de contenedores está justificado para la aplicación DNSE3 y otras aplicaciones que presente problemas similares a esta.

Otro de los objetivos de este TFG será **mejorar el rendimiento del servicio encargado de generar el informe con los resultados obtenidos durante las distintas simulaciones**. Estos informes se generan una vez que han finalizado todas las simulaciones de un trabajo con el objetivo de facilitar a los alumnos el trabajo de análisis de los resultados obtenidos. Se ha comprobado que, en la versión actual de la aplicación DNSE3 [2], este servicio está ralentizando toda la aplicación debido al tiempo que precisa para generar cada uno de los informes por el elevado número de peticiones que tiene realizar al servicio donde se almacenan los resultados de las distintas simulaciones.

Para finalizar, se deberá intentar **solventar los problemas que se detecten durante la utilización de la versión actual del DNSE3**. También se deberán solucionar los

posibles problemas derivados de la aplicación de las nuevas tecnologías que se van a utilizar para realizar el resto de objetivos de este Trabajo Fin de Grado.

1.3. Metodología

Para completar los distintos objetivos de este TFG se ha seguido la línea de trabajo del método de la ingeniería. Este método se compone de cuatro fases: analizar las posibles soluciones a un problema; intentar desarrollar un sistema que mejore las soluciones existentes; implementar dicho sistema y analizar su funcionamiento; por último, comparar los resultados obtenidos con el nuevo sistema frente al antiguo. Estas fases hay que repetirlas hasta que no sea posible realizar más mejoras al sistema [9]. Así pues, para resolver el problema de los retardos asociados a la creación de máquinas virtuales se siguió el siguiente esquema: en la primer lugar se buscó una alternativa a las máquinas virtuales que fue la tecnología de contenedores; durante la segunda fase de trabajo estudió el funcionamiento de la tecnología de contenedores y de cómo aplicar esta tecnología al sistema actual; una vez que se comprendió el funcionamiento de la tecnología de contenedores se comenzó la segunda tercera fase del método de la ingeniería, realizando las modificaciones oportunas a la aplicación para que utilizara la tecnología de contenedores y se realizaron unas pruebas de rendimiento para comprobar las prestaciones que ofrecía esta nueva versión; por último, se comparan los resultados obtenidos en la nueva versión con los que se obtuvieron en [4] para decidir si se continúa trabajando con la tecnología de contenedores.

Para que este sistema de fases funcione de manera correcta, se han realizado reuniones con los tutores del Trabajo Fin de Grado cada quince días o tres semanas. En estas reuniones se explicaba el trabajo realizado entre ese instante y la reunión previa y se exponían los resultados que se habían conseguido hasta ese instante. A partir de estos resultados, se decidían cuáles serían los próximos pasos que se iban a seguir. Con estas reuniones se ha conseguido que el trabajo se realizada de manera ágil y se ha podido alcanzar los objetivos impuestos en este TFG gracias a la realimentación que se ha obtenido en las mismas.

1.4. Estructura del documento

A partir de este punto del presente documento, se explicará tanto los fundamentos teóricos de las tecnologías utilizadas, como su aplicación al DNSE3 y los resultados que ofrece dicha aplicación con los cambios realizados. En el capítulo 2 se indicarán las tecnologías utilizadas en el desarrollo de este TFG, centrándose en las tecnologías existentes más representativas para la creación y gestión de contenedores, así como sus principales características. Este capítulo se dividirá en varias secciones: en la sección 2.1 se analizará los tipos de contenedores que existen en el mercado actual; en la sección 2.2 se comparará los diferentes tipos de motores de orquestación que se utilizan para controlar los contenedores; en la 2.3 se describirá las soluciones que ofrece OpenStack para trabajar con la tecnología de contenedores dentro de su arquitectura; por último, en la sección 2.4 se indicará qué tecnologías de las descritas en las secciones previas se van a utilizar en la aplicación DNSE3 y los motivos por los que se ha llegado a esa elección.

En el capítulo 3 se describirá tanto la arquitectura de la aplicación actual presentada en [2], como los cambios realizados en cada uno de los servicios para conseguir adaptar la aplicación a las nuevas tecnologías. En la sección 3.1 se indicarán los trabajos previos hasta llegar a la versión actual de la aplicación y se hará una descripción más específica de la

última versión funcional. En la sección 3.2 se describirá la nueva arquitectura de la aplicación, profundizando en los servicios en los que se ha llevado a cabo cambios más profundos. En la sección 3.4 se describirá cómo se puede desplegar la aplicación junto con parámetros que se deben introducir para conseguir dicho despliegue. Para finalizar, en 3.5 se indicarán los aspectos más relevantes de este capítulo.

En el capítulo 4 se expondrá el rendimiento de la aplicación DNSE3. Se comparará los resultados obtenidos con los cambios que se han realizado durante la elaboración de este Trabajo Fin de Grado, con los resultados obtenidos con la versión anterior de la misma aplicación reflejados en [4]. Por último, en el capítulo 5 se realizará un comentario para indicar si se ha conseguido cumplir los objetivos que se impusieron al principio de dicho trabajo. Además, se indicará cuáles podrían ser los próximos pasos para conseguir que la aplicación DNSE3 tenga un rendimiento óptimo, con el objetivo de que el tiempo necesario para realizar un número elevado de simulaciones no sea un inconveniente para los alumnos.

Capítulo 2

Tecnologías relacionadas

En el desarrollo de la última versión funcional de la aplicación DNSE3 presentada en [4] se utilizaron múltiples tecnologías para lograr el objetivo de proporcionar los resultados de las simulaciones que soliciten los alumnos en un tiempo razonable.

- El núcleo de la aplicación DNSE3 está desarrollado en Java. Este lenguaje de programación es multiplataforma, lo cual permite su ejecución sobre distintos sistemas operativos y distintas configuraciones de *hardware*. Además, que el código de la aplicación esté escrito en Java facilitará su comprensión a futuros desarrolladores, debido a la gran popularidad con la que cuenta dicho lenguaje de programación. Todos los años la plataforma *GitHub* presenta un informe[1] con, entre otras cosas, los lenguajes de programación más utilizados, y en dicho informe se puede observar que Java ocupa la segunda posición entre los lenguajes más utilizados.
- Los distintos servicios que forman la aplicación hacen uso de la arquitectura REST (*REpresentational State Transfer*) [10]. De este modo, las comunicaciones entre los distintos servicios son *stateless*, lo que, por un lado, facilita su implementación y, por otro, permite descargar a la parte servidora de la aplicación de carga de trabajo puesto que el cliente es el responsable de mantener el estado global de la aplicación, siendo el servidor responsable de mantener el estado de los *recursos* [11]. Además, al implementar una arquitectura REST se consigue una interfaz uniforme puesto que se consigue una manera estándar de gestionar los *recursos* a través de peticiones y respuestas [11], lo que facilita la realización de futuras modificaciones.
- Para la ejecución de las distintas simulaciones que le son solicitadas por los usuarios, utiliza el simulador `ns-3`. Está licenciado bajo GNU GPLv2 y es un simulador de red de eventos discretos, destinado para investigación y uso educativo, cuyos módulos están escritos en `C++` [12].
- La versión *actual* de la aplicación DNSE3 se ejecuta sobre una nube de computación privada, gestionada por OpenStack, por lo que hace uso de servicios propios de dicha plataforma. Por ejemplo, para almacenar los resultados de las distintas simulaciones utiliza el servicio de almacenamiento de objetos *Swift*[2]. Otro ejemplo de la utilización de tecnologías propias de OpenStack en la aplicación DNSE3, es el proyecto *Heat*[3] usado para adecuar el número de máquinas virtuales que ejecutan las simulaciones, a la demanda de trabajos actuales por parte de los alumnos.

[1] https://octoverse.github.com/projects#languages
[2] https://wiki.openstack.org/wiki/Swift
[3] https://wiki.openstack.org/wiki/Heat

Gracias a la utilización de las tecnologías previamente citadas, la versión *actual* de la aplicación DNSE3 es una versión funcional. Sin embargo, en dicha versión se detectaron problemas de rendimiento, asociados a retardos provocados por la creación y destrucción de las máquinas virtuales sobre las que se ejecutan las simulaciones. Por este motivo, se buscó una tecnología alternativa que mitigara estos retardos. La tecnología que se decidió utilizar fueron los contenedores.

Un *contenedor* es una unidad estándar de software que contiene todas las dependencias que necesita un programa para poder ejecutarse [8]. Los contenedores no necesitan iniciar un sistema operativo completo como tendría que realizar una máquina virtual ya que se ejecutan sobre el *kernel* del *host*[4]. Al aprovechar el *kernel* del sistema operativo anfitrión, la inicialización y destrucción de contenedores se realiza en menos tiempo que el que se necesitaría una máquina virtual para conseguir el mismo resultado. Por esta cualidad, si se utiliza la tecnología de contenedores en el servicio encargado de la ejecución de las simulaciones, se podrían solventar los problemas de retardos encontrados en la versión actual de la aplicación DNSE3.

Al ejecutar varios contenedores sobre un mismo sistema operativo, cabría esperar que los contenedores fueran una tecnología más insegura que las máquinas virtuales, debido a que estas últimas no comparten el *kernel* ni con el *host* sobre el que se ejecutan ni con el resto de las posibles máquinas virtuales que se estén ejecutando sobre dicho *host*. Esto no es necesariamente cierto, puesto que para garantizar el aislamiento entre los contenedores y el sistema operativo anfitrión, el ciclo de vida de los contenedores es gestionado por un motor de contenedor propio del tipo utilizado [13]. Los motores de contenedor se encargan de que los contenedores solamente tengan acceso a la cantidad de recursos (utilización de CPU, de memoria RAM, etc.) que se les permita. Si no se limitan los recursos que un contenedor puede utilizar, podrá usar, si los necesita, la totalidad de los recursos que tenga disponible el *host*. Para los despliegues de contenedores más complejos, en los cuales se necesita utilizar los recursos de varios *hosts*, es necesario utilizar motores de orquestación de contenedores (*Container Orchestration Engines, COE*). Estas tecnologías permiten escalar el número de contenedores que se ejecutan en un servicio, encargándose de distribuirlos por los diferentes *hosts* para hacer uso de sus recursos.

2.1. Tipos de contenedores

Un contenedor es una unidad estándar de software que incluye todas las dependencias que necesita un programa para su correcto funcionamiento [8]. Los contenedores parten de una imagen de contenedor, en la que estarán tanto la aplicación que se desea ejecutar como las dependencias que ésta necesite. Todos los contenedores que parten de la misma imagen de contenedor se iniciarán siempre en el estado con el que fueron creados. Los datos generados por cada uno de los contenedores se perderán cuando dicho contenedor se destruya, a no ser que se utilice una tecnología auxiliar que almacene estos datos de manera persistente.

La principal diferencia entre una máquina virtual y un contenedor es que la primera, necesita cargar un sistema operativo completo para iniciarse, mientras que la segunda tecnología aprovecha el *kernel* del sistema operativo anfitrión [13]. Por este motivo, los contenedores son una tecnología más liviana que las máquinas virtuales, con lo que se consigue que su inicialización y parada se realice de manera más rápida.

[4]Con la palabra *host* definiremos tanto a una máquina virtual gestionada por un *hypervisor* como a una máquina física que tenga instalado un sistema operativo específico

La primera aproximación de una tecnología similar a lo que en la actualidad se denomina contenedor, fue la utilización del comando `chroot` en sistemas basados en UNIX. Utilizando este comando, se consigue *separar* el sistema de ficheros del sistema operativo que se esté utilizando del que utiliza el proceso afectado y sus subprocesos hijos [13]. A partir de esta idea, la tecnología fue evolucionando hasta llegar a los contenedores actuales. En esta subsección se describirán tres de las tecnologías más utilizadas para empaquetar programas dentro de contenedores que se plantearon como alternativa para su utilización en la nueva versión de la aplicación DNSE3.

2.1.1. Docker

Docker[5] fue lanzado en el año 2013 y se ha convertido en la plataforma líder para empaquetar *software* a nivel mundial. Fue diseñada para facilitar la creación, el desarrollo y la ejecución de programas. Es multiplataforma, puesto que su motor de contenedor se puede ejecutar tanto en GNU/Linux como en Windows[6] y Mac [13, 14].

Esta plataforma se basa en *Containerd*[7], proyecto de código abierto cedido por Docker a la *Cloud Native Computing Foundation*[8] para que se consiguiera estandarizar y extender su tecnología [8]. El motor de contenedor de esta plataforma se denomina *Docker Engine*[9] y existen dos versiones del mismo, una versión empresarial (*Docker Engine Enterprise*) y otra gratuita (*Docker Engine Community*). Las diferencias más relevantes entre estas dos versiones son el tiempo de mantenimiento (se pasa de 24 meses de la versión de pago a 7 en la gratuita) y que las nuevas características implementadas por *Docker Inc.* llegan primero a la versión *Enterprise* que a la *Community*.

La plataforma Docker proporciona un repositorio, denominado *Docker Hub*[10] donde poder almacenar las imágenes de contenedor que se desarrollen. De esta forma, siempre que se tenga acceso a Internet, se podrá acceder a dichas imágenes de contenedor, lo que facilita la portabilidad para poder trabajar en múltiples *hosts*. Para la creación de la imagen de contenedor que contendrá el programa a ejecutar y todas sus dependencias, se suele partir de una imagen base de contenedor para simplificar el proceso. Dentro del repositorio *Docker Hub* se pueden encontrar imágenes oficiales para este propósito, como es el caso de Ubuntu[11].

2.1.2. rkt

rkt[12] es un motor de contenedores desarrollado por CoreOS[13] para entornos de producción nativos de la nube. Nació como alternativa a las primeras versiones de Docker, puesto que en estas existían problemas de seguridad, que posteriormente se fueron subsanando [13]. Además, la ejecución de los contenedores Docker se realiza mediante el uso de un demonio al que se le conceden permisos de superusuario, mientras que la ejecución de contenedores rkt se puede realizar desde un usuario sin privilegios [13, 15].

[5] https://www.docker.com/

[6] Para utilizar Docker en Windows es necesario usar la versión `Pro` (https://www.microsoft.com/es-es/p/windows-10-pro/df77x4d43rkt/48DN) ya que necesita utilizar características de virtualización que solamente están incluidas en dicha versión.

[7] https://containerd.io/

[8] https://www.cncf.io/

[9] https://www.docker.com/products/docker-engine

[10] https://hub.docker.com

[11] https://hub.docker.com/_/ubuntu

[12] https://coreos.com/rkt/

[13] https://coreos.com/

Esta tecnología cuenta con un enfoque de "*pod* nativo", siendo un *pod* la unidad de trabajo del sistema de orquestación de contenedores Kubernetes[14]. rkt implementa un formato de contenedor abierto y estandarizado, la especificación *App Container*[15], que define el formato de la imagen de contenedor y el entorno de ejecución. Además, permite ejecutar otras imágenes de contenedor, como son las creadas para la plataforma Docker [15].

2.1.3. LXC

LXC[16] (*Linux Containers*) son los contenedores específicamente diseñados para trabajar con Linux. Al igual que el resto de tecnologías, proporciona unas herramientas para la creación y gestión de contenedores. Se les considera el paso intermedio entre la utilización del comando `chroot` y la creación de una máquina virtual completa [16].

El objetivo principal de este tipo de contenedores es la creación de un entorno lo más cercano posible a una instalación estándar de Linux, pero sin la necesidad de tener un núcleo completo para cada uno de los contenedores que se ejecuten [16]. El requisito principal que se necesita para utilizar este tipo de tecnología es disponer de un núcleo de Linux igual o superior a la versión 2.6.32.

2.1.4. Tipo de contenedor escogido para DNSE3

Una vez que se han visto las posibles alternativas de contenedores que pueden ser válidas para empaquetar el servicio encargado de realizar las simulaciones de la aplicación DNSE3, se tuvo que elegir una de ellas. Las tres alternativas comparten la idea de qué es un contenedor y de cómo aislar los procesos que se ejecutan dentro de estos del resto del sistema. Todas ellas son gratuitas, o tienen una versión gratuita, y la documentación aportada por los desarrolladores es completa y accesible en sus respectivas páginas web, por lo que, a nivel técnico, cualquiera de ellas se podría utilizar en la aplicación DNSE3.

Una vez visto que las tres opciones podrían haberse usado para crear un contenedor donde ejecutar el servicio encargado de realizar las simulaciones, se decide utilizar contenedores tipo **Docker**, ya que se han convertido en el estándar *de facto* de este tipo de tecnologías. Por este motivo, gran parte de los motores de orquestación de contenedores son compatibles con Docker. Además, existen alternativas en la nube de cómputo pública para ejecutar contenedores Docker, destacando *Amazon Web Services* (AWS) con su servicio *AWS Fargate*[17] que permite ejecutar contenedores sin tener que gestionar clústeres manualmente. Este último dato es de gran importancia para la aplicación DNSE3 puesto que, en la actualidad, dicha aplicación se está ejecutando sobre una nube privada, pero no se descarta que en un futuro pueda ser portada a una pública. Este es el motivo por el que los servicios de DNSE3 deben utilizar las tecnologías más populares para que se pueda cambiar de un tipo a otro de nube de cómputo con el menor número de modificaciones posibles.

2.2. Motores de orquestación de contenedores

Los motores de orquestación de contenedores (*Container Orchestration Engines, COE*) se utilizan para desplegar servicios basados en contenedores, servicios que pueden estar formados

[14] https://kubernetes.io/
[15] https://coreos.com/rkt/docs/latest/app-container.html
[16] https://linuxcontainers.org/
[17] https://aws.amazon.com/es/fargate/

por diferentes contenedores de la misma tecnología [13]. Los COE también son necesarios cuando se necesitan desplegar contenedores por distintos *hosts*. Esta cualidad otorga a los programas que se ejecutan dentro de contenedores un alto grado de escalabilidad, puesto que, si se diseñan de una manera apropiada, podrán llegar a utilizar un número *ilimitado* de recursos.

Una vez que se ha decidido utilizar contenedores de tipo Docker, se buscaron las diferentes alternativas que de motores de orquestación que fueran compatibles con los contenedores tipo Docker. Esta elección quedó limitada por las tecnologías soportadas por el proyecto *Magnum*[18] de OpenStack del que se hablará en la subsección 2.3.2, al igual que se describirá el proyecto *Zun*[19] en la sección 2.3.1, proyecto que no necesita de la creación de un clúster de máquinas virtuales para aprovechar los recursos de la nube de cómputo. Las alternativas que permiten la doble compatibilidad con Docker y *Magnum* son Kubernetes, Apache Mesos[20] y Docker Swarm[21]. Para aprovechar los recursos de varios *hosts*, este tipo de tecnologías crean un clúster formado por dichos *hosts*, en el que, unos nodos se encargan de la gestión propia del clúster, y otros, de la ejecución de los contenedores de los servicios que estén activos en el mismo [14, 17]. Aunque comparten la arquitectura, a continuación se destacan algunas peculiaridades tanto de Kubernetes como de Apache Mesos, puesto que Docker Swarm se describirá más profundamente en 2.2.1.

- **Kubernetes** es una plataforma para la gestión de la carga de trabajo y los servicios sobre contenedores, facilitando la orquestación de estos [17, 18]. La unidad de trabajo de Kubernetes es el *Pod*, que puede llegar a contener uno o más contenedores que compartirán un mismo almacenamiento local accesible para todos ellos. Admite tanto contenedores tipo Docker como *rkt*. Al *host* o *hosts* que gestione o gestionen el clúster se denomina *Kubernete Master*, mientras que el nombre que se les da a los encargados de ejecutar los *pods* es *Kubernete Node* [17].

- **Apache Mesos** se desarrolla siguiendo los mismos principios que el núcleo de Linux, con la diferencia del nivel de abstracción. El *kernel* de esta tecnología se ejecuta sobre cada *host* proporcionando un *framework* para la gestión de los recursos. En este caso, al gestor del clúster se le denomina *Master*, y a los encargados de ejecutar los contenedores *Slaves*, que pueden ser del tipo Docker puesto que son compatibles con esta tecnología [13].

Debido a la elección de contenedores tipo Docker para empaquetar el *software* encargado de ejecutar las simulaciones dentro de la aplicación DNSE3 y, como se querían utilizar proyectos propios de OpenStack para la gestión de contenedores sobre la versión instalada en la nube de cómputo privada en la que se va a ejecutar DNSE3, la elección del motor de orquestación de contenedores se simplificó, resultando únicamente como opción viable el uso de Docker Swarm. La elección del COE utilizado quedó reducida a Docker Swarm puesto que, con la versión actualmente instalada de OpenStack e intentando hacer uso del proyecto *Magnum* para orquestar el clúster de máquinas virtuales sobre el que se ejecutarían los contenedores, no se consiguió que funcionara ninguna de las otras alternativas. Aunque la nueva versión resultante de la realización de este Trabajo Fin de Grado se ha implementado sobre dicho COE, los cambios necesarios para adaptarse a otro motor de orquestación de contenedores no serían muy laboriosos, puesto que, como se acaba de ver, Docker se ha convertido en el estándar de la industria, por lo que los motores de orquestación están preparados para

[18] https://wiki.openstack.org/wiki/Magnum
[19] https://wiki.openstack.org/wiki/Zun
[20] http://mesos.apache.org/
[21] https://github.com/docker/swarm

su gestión. Del mismo modo, si en un futuro se decide prescindir de la utilización de este COE para utilizar tecnologías propias de las nubes de cómputo públicas, la elección de los contenedores Docker simplificará completar dicho cambio.

2.2.1. Docker Swarm

Docker Swarm es el COE oficial de Docker. Con él se podrán gestionar los clústeres formados por varios *hosts* sobre los que se van a ejecutar servicios basados en contenedores Docker, con el objetivo de utilizar los recursos de cada *host*, si fuese necesario. En estos clústeres nos encontramos dos tipos de hosts, los que actúan como gestor del clúster (*Swarm manager*) y los que ejecutan los contenedores pero no pueden controlar ni saber lo que sucede en el clúster (*Swarm worker*) [14].

- ***Swarm manager***. Este tipo de nodos está encargado de la gestión del clúster. Desde ellos se indicarán los servicios a ejecutar. También se gestionarán los nodos que forman el clúster. Estos nodos son los únicos que conocen el *hardware* del resto de *hosts* que forman el clúster. Es necesario que exista, al menos, un *Swarm manager*, pero Docker recomienda incluir más de uno por clúster, hasta un máximo de 7, con el objetivo de aumentar la tolerancia frente a fallos, calculada como $\frac{N-1}{2}$, siendo N el número de nodos *manager*. Esta tolerancia a fallos es debida a que en los nodos de este tipo se encuentra el punto de acceso, o *endpoint*, para la creación o destrucción de servicios basados en contenedor que se desea ejecutar dentro del clúster. Así que, si no se tiene más de un *Swarm manager* en el clúster y, por algún motivo, se pierde, los *workers* se quedarán ejecutando el número de contenedores que se les hubiera indicado en la última comunicación con el *manager*. Por este motivo habría que disolver el clúster y volver a formarlo si se desea volver a tener control sobre el mismo. Aumentar el número de nodos *manager* no supone una mejora en el rendimiento, ya que todos ellos necesitan conocer el estado global del clúster en cada instante, por lo que aumentar el número de nodos puede reducir el rendimiento del clúster, debido a que se tiene que aumentar el número de comunicaciones dentro del mismo [19]. Un *Swarm manager* también puede ejecutar contenedores como si fuera un nodo *worker* con el objetivo de aprovechar el mayor número de recursos posible. Es posible *degradar* a un *manager* para que solamente actúe como nodo *worker*.

- ***Swarm worker***. Este tipo de nodos está encargado de la ejecución de los distintos contenedores que les indique el *Swarm manager*. Dentro del clúster pueden existir cero o más nodos de este tipo. Los nodos *worker* no son conscientes de los nodos que están incluidos en el clúster. Tampoco saben qué tipo de contenedores ni en qué cantidad se están ejecutando dentro del clúster. Un nodo *worker* solo conoce lo que sucede dentro de él. A los nodos *worker* se les puede *promocionar* a *Swarm manager* desde un nodo *manager*.

La distribución de los contenedores por los distintos *hosts* que formen un clúster gestionado por uno o más *managers* puede seguir tres políticas [20]: ***spread***, ***binpack*** y ***random***. Con la política *random* la elección del lugar donde ejecutar el nuevo contenedor a ejecutar en el clúster se toma aleatoriamente. Utilizando la estrategia *spread*, política utilizada por defecto si no se indica otra, Docker Swarm inicia el nuevo contenedor sobre el *host* con más recursos disponibles. Si existen dos nodos en el clúster con el mismo número de recursos disponibles, Docker Swarm despliega el nuevo contenedor sobre el *host* con menos contenedores activos. Con la estrategia *binpack*, Docker Swarm despliega el nuevo contenedor sobre el *host*

con menos recursos disponibles. Si dos nodos tienen el mismo número de recursos disponibles, con la estrategia *binpack*, el nuevo contenedor se iniciará en el nodo con más contenedores activos. La ventaja de utilizar la estrategia *spread* frente a *binpack* es que, con la primera, se distribuyen los contenedores por todos los *hosts* que forman el clúster, por lo que, si un nodo falla, el número de contenedores que se pierde es más bajo que si se utiliza la política *binpack* y se pierde el *host* más congestionado. En contraposición, la ventaja de utilizar *binpack* frente a *spread* es que con esta política se consigue concentrar la mayor parte de contenedores en un menor número de *hosts*.

2.3. Uso de contenedores sobre OpenStack

OpenStack es una plataforma de gestión de nube computacional gratuita y abierta. Permite realizar la gestión de grandes grupos de recursos de cómputo, almacenamiento y redes desde un único lugar. Fue fundado en 2010 por Rackspace[22] y NASA[23], pero actualmente es gestionada por una organización sin ánimo de lucro, *The OpenStack Foundation*[24] [7, 13]. La arquitectura de OpenStack está distribuida en proyectos, entre los que hay que destacar los considerados proyectos núcleos de OpenStack [13]: ***Nova***[25], servicio de cómputo, responsable de administrar el ciclo de vida de las máquinas virtuales, siendo la interfaz del *hypervisor* que se esté utilizando; ***Neutron***[26], servicio de administración de redes, encargado de la gestión de las redes de comunicación; ***Swift***, servicio de almacenamiento de objetos, que utiliza una arquitectura distribuida para almacenar los datos; ***Glance***[27], servicio que proporciona las imágenes de sistema; ***Keystone***[28], servicio de gestión de identidades, encargado de autenticar y autorizar las acciones de los usuarios y los servicios; y por último, ***Cinder***[29], servicio de almacenamiento de bloques, encargado de proporcionar persistencia a los datos generados por las distintas máquinas virtuales.

La versión actual de la aplicación DNSE3 [2] utiliza los proyectos previamente citados, además de los proyectos ***Heat***, el servicio de orquestación de los recursos que gestiona OpenStack, y ***Gnocchi***[30], servicio de métricas. Con el uso combinado de todos estos servicios se consiguió automatizar que, mediante la publicación periódica del número de simulaciones pendientes de realizar, se adecuara el número de máquinas virtuales activas sobre las que se ejecutaban dichas simulaciones para hacer un uso responsable de los recursos existentes en la nube de cómputo privada. Como ya es conocido, el escalado del número de máquinas virtuales acarrea unos retardos asociados a su inicialización y destrucción, por lo que se decidió utilizar contenedores de tipo Docker para mitigar este problema. Finalmente, la aplicación DNSE3 se va a ejecutar sobre una nube computacional gestionada por OpenStack debido a que es la plataforma que se está utilizando para gestionar de los recursos de la nube de cómputo privada del grupo de trabajo donde se está realizando este TFG. Por ello se decidió buscar alternativas propias de esta plataforma que estuvieran preparadas para trabajar con contenedores. Dentro de OpenStack existen dos proyectos con ese propósito, ***Zun*** y ***Magnum***. El resto de la sección se utilizará para describir dichos proyectos.

[22] https://www.rackspace.com
[23] https://www.nasa.gov/
[24] https://www.openstack.org/foundation/
[25] https://docs.openstack.org/nova/latest/
[26] https://docs.openstack.org/neutron/latest/
[27] https://docs.openstack.org/glance/latest/
[28] https://docs.openstack.org/keystone/latest/
[29] https://docs.openstack.org/cinder/latest/
[30] https://gnocchi.xyz/

2.3.1. Zun

Zun, cuya primera versión fue lanzada durante el ciclo de la versión *Mitaka*[31], es un servicio de OpenStack que permite la gestión de aplicaciones empaquetadas en contenedores [13]. Para gestionar dichos contenedores hace uso de los proyectos núcleo *Keystone*, *Neutron* y *Glance*, además de ser compatible con *Heat*, para llevar a cabo la automatización del escalado del número de contenedores y *Kuryr*[32], proyecto que, en teoría, permitiría la interacción entre los contenedores creados por *Zun* y las máquinas virtuales creadas gracias al proyecto *Nova*.

El proyecto *Zun* es compatible con contenedores tipo Docker. Con este proyecto, se elimina la necesidad de utilizar máquinas virtuales para la ejecución de contenedores, es decir, se elimina la necesidad de uso de un clúster, reduciendo el número de *middlewares* entre los contenedores y el *hardware* de la nube de cómputo, lo que, en principio, proporcionará un mayor rendimiento. La equivalencia de este proyecto para la nube de cómputo AWS es *Amazon Elastic Container Service*[33] (ECS).

En referencia a la aplicación DNSE3, si se hubiera utilizado este proyecto para la gestión de los contenedores sobre los que se ejecutan las simulaciones, no habría hecho falta realizar modificaciones a la arquitectura de la aplicación, debido a que se podría seguir publicando el número de simulaciones por completar gracias a la métrica que gestiona el escalado en DNSE3 publicada en *Gnocchi*. Con ese valor, y atendiendo a una serie de reglas fijadas previamente, *Heat* orquestaría, a través de *Zun*, el número de contenedores activos que estuvieran realizando las simulaciones solicitadas por los alumnos. El motivo principal por el que se descartó la utilización de *Zun* para la aplicación DNSE3 fue la falta de compatibilidad que existe, en la actualidad, entre dicho proyecto y el proyecto *Nova*. La nube de computo privada sobre la que se desea ejecutar el DNSE3 alberga otros proyectos basados en máquinas virtuales, y en ningún caso, las modificaciones realizadas para actualizar dicha aplicación pueden afectar al resto de proyectos. Además, la guía de instalación de *Zun* para la versión *Pike*[34] de OpenStack, versión que actualmente se están utilizando en la nube de computo privada, es inexistente [21].

2.3.2. Magnum

Magnum, cuya primera versión fue lanzada el año 2014, es el proyecto de OpenStack encargado de gestionar los motores de orquestación de contenedores, por lo que su objetivo es proveer de una infraestructura sobre la que ejecutar los contenedores. La base de esta infraestructura son las máquinas virtuales, así que *Magnum* necesita valerse de los proyectos *Nova*, *Neutron* y *Cinder*, entre otros, para su creación [13]. Soporta la gestión de los COE Docker Swarm, Kubernetes y Apache Mesos, que como se vio en la sección 2.2, comparten una arquitectura de clúster similar.

Las máquinas virtuales que utiliza el proyecto *Magnum* son creadas a partir de una plantilla. En dicha plantilla se especifican los recursos que pueden utilizar las máquinas que ejerzan de gestores del clúster y las que ejecuten los contenedores (dichos valores pueden ser diferentes). También se tiene que indicar qué imagen de sistema operativo se va a ejecutar en cada una de las máquinas. *Magnum* indica que para utilizar Kubernetes o Docker Swarm

[31] https://www.openstack.org/software/mitaka/
[32] https://wiki.openstack.org/wiki/Kuryr
[33] https://aws.amazon.com/es/ecs/
[34] https://www.openstack.org/software/pike/

hay que utilizar una versión específica de *Fedora*[35] llamada *Fedora Atomic*[36], y para usar Apache Mesos una de *Ubuntu*[37]. Dichas imágenes están modificadas para incluir los motores de orquestación y motores de contenedor de cada tecnología [22]. Cuando se incrementa el número de máquinas virtuales del clúster para disponer de más recursos en este, *Magnum* realizará las tareas oportunas para que la nueva máquina virtual se incluya dentro del clúster.

A partir de lo anterior, se determinó que *Magnum* podría ser una tecnología viable para gestionar el clúster de máquinas virtuales sobre las que ejecutar el servicio encargado de realizar las simulaciones de la aplicación DNSE3. Por este motivo, se decidió instalar en la versión de OpenStack que gestiona los recursos de la nube de cómputo privada, comprobando que la instalación de dicho proyecto no afectaba al resto de proyectos que estaban instalados previamente. En las primeras pruebas realizadas para comprobar el funcionamiento del clúster gestionado por *Magnum* se detectó que cuando se utilizaba el *Swarm manager* del clúster como nodo híbrido, en el que ejecutan contenedores además de encargarse de las tareas de gestión propias de este tipo de nodos, en los procesos de escalado del número de máquinas que formaban el clúster, los contenedores se ejecutaban de manera errónea, dejando de funcionar durante el tiempo necesitaba el clúster para aumentar en una unidad el número de máquinas virtuales activas, que llegaba a los tres minutos. El problema de que los contenedores dejaran de funcionar se solventó utilizando la máquina *Swarm manager* solamente para gestionar el clúster, y, por ello, se indicó en la creación de la plantilla que los recursos del nodo *manager* y los recursos de los nodos *workers* fueran diferentes, creando así un clúster con dos configuraciones de máquinas virtuales, lo que permitía reducir el número de recursos desperdiciados. Una vez que se solucionó este problema, quedaban por resolver otros dos: el tiempo que necesitaba el clúster para incrementar en una unidad el número de máquinas activas en el clúster, que como se ha indicado, podía llegar a necesitar tres minutos; cómo se iba a realizar el escalado del número de contenedores activos en el clúster, puesto que el proyecto *Magnum* permite automatizar el escalado de las máquinas virtuales que se ejecutan en ellas mediante el uso de otros proyectos propios de OpenStack, pero no proporciona ninguna herramienta para automatizar la creación y destrucción de los contenedores ejecutados sobre estas.

Finalmente, para solventar estos problemas se decidió descartar la utilización *Magnum* y crear un servicio propio de la aplicación que automatizara el escalado en el plano de máquinas virtuales y en el de contenedores para adecuar los recursos utilizados al número de trabajos pendientes de realizar. Este servicio propio de la aplicación hará uso del proyecto *Heat* para la creación y destrucción de las máquinas virtuales que formen el clúster, además de la API proporcionada por Docker para indicar el número de contenedores activos dentro de este. La descripción de este nuevo servicio de la aplicación se realiza en la sección **??**.

2.4. Conclusiones

Con el objetivo de mitigar el problema de retardos relacionados con la creación y destrucción de las máquinas virtuales sobre las que se ejecutan las simulaciones en la versión *actual* de la aplicación DNSE3 [2], se decide utilizar la tecnología de contenedores, que permiten iniciar aplicaciones en un tiempo menor que el que se necesitaría para iniciar la misma aplicación creando una máquina virtual para ello. Una vez vistas varias tecnologías de contenedores se decide utilizar Docker para empaquetar el *software* encargado de ejecutar las simulaciones y el motor de orquestación de contenedores Docker Swarm para gestionar dichos contenedores.

[35] https://getfedora.org/es/
[36] https://www.projectatomic.io/
[37] https://ubuntu.com/

Para adecuar el número de máquinas virtuales y contenedores a la demanda de trabajos en la aplicación DNSE3 se decide crear un servicio propio de la aplicación que usará tanto la API de Docker para gestionar el número de contenedores activos en el clúster como el proyecto *Heat* para adecuar el número de máquinas virtuales dentro de este.

Capítulo 3

Rediseño y nueva implementación de la aplicación

La versión actual de la aplicación DNSE3 [4] consigue distribuir las simulaciones de red por un número variable de máquinas virtuales. Esto le sirve, en primer lugar, para proporcionar a los alumnos los resultados de los modelos simulados en un tiempo razonable y, en segundo lugar, para utilizar de manera responsable los recursos disponibles en la nube de cómputo privada sobre la que se ejecuta. Aun así, debido a que el proceso de activación y desactivación de las máquinas virtuales conlleva unos retardos no despreciables, se decide ejecutar las simulaciones sobre contenedores, que, como se ha visto en el capítulo 2, se activan y desactivan de forma más rápida que las máquinas virtuales.

La tecnología utilizada para empaquetar el *software* necesario para ejecutar las simulaciones que soliciten los usuarios va a ser Docker, el tipo de contenedor más popular que se ha convertido en el estándar de la industria. Por los problemas que se encontraron en los proyectos propios de OpenStack para la gestión de contenedores, se decidió crear una nueva arquitectura para ejecutar las simulaciones. Esta nueva arquitectura estaría basada en máquinas virtuales sobre las que se ejecutarían los contenedores Docker que tendrían todo lo necesario para realizar las simulaciones. Dichas máquinas virtuales estarían provistas de una capacidad de cómputo suficiente para albergar varios contenedores Docker con lo que se evitará tener que activar y desactivar un número alto de máquinas virtuales.

3.1. Trabajos previos

El trabajo realizado en este TFG parte de la versión presentada en [2], que a su vez estaba basada en DNSE (*Distributed Network Simulation Environment*) [3], por lo que se comenzará describiendo el proyecto DNSE. Este proyecto utilizaba el simulador ns-2[1], versión previa al ns-3, con el fin de realizar las simulaciones de red que le solicitaran los usuarios. Para ello, distribuía las simulaciones sobre un *grid* computacional, consiguiendo que los alumnos obtuvieran los resultados de los modelos simulados en un tiempo razonable. Un *grid* computacional es una infraestructura que permite compartir, a través de la red, recursos de *hardware* y *software* entre organizaciones [23]. La aplicación fue dividida en servicios para que se encargaran de realizar unas tareas concretas, lo que permitía que dichos servicios pudieran ser reutilizados en otras aplicaciones. Con DNSE, además de permitir realizar simulaciones de tipo individual y de barrido de un parámetro, se podían analizar los resultados obtenidos de

[1] https://www.isi.edu/nsnam/ns/

las distintas simulaciones, ya que daba la posibilidad de generar gráficos del tipo *x-y*. También se podía genera animaciones de la red simulada, gracias al uso de la herramienta *Nam* (*Network Animator*) propia del simulador ns-2.

La razón que motivó el cambio de la aplicación DNSE a DNSE3 fue la aparición del paradigma de la nube computacional, con el que se consigue utilizar los recursos disponibles de forma más simple que en un *grid*, puesto que todos ellos se gestionan desde un único punto. Se comenzó a trabajar con la idea del DNSE3 en [6] y se presentó una aplicación funcional en [2]. DNSE3 utiliza el simulador ns-3 y la nube computacional para procesar las simulaciones con el mismo objetivo que el DNSE: proporcionar a los alumnos los resultados de las simulaciones en un tiempo razonable. Igual que su predecesora, DNSE3 está dividida en servicios, cada uno de los cuales tiene asignado una serie de tareas a realizar. Los servicios fueron diseñados siguiendo una arquitectura REST con el objetivo de acceder a sus funciones como recursos. Con ello se conseguía proporcionar, por un lado, una interfaz uniforme que facilitaría futuras modificaciones y, por otro, que la parte servidora no tenga que mantener el estado de la comunicación consiguiendo con ello reducir la carga de trabajo de esta parte. Dispone de un cliente web diseñado para facilitar a los alumnos el uso de la aplicación. Desde este cliente se podrá solicitar la realización de simulaciones, tanto de tipo individual como de barrido de parámetros, dando información acerca del número de simulaciones realizadas y restantes.

3.1.1. Arquitectura de la aplicación basada en máquinas virtuales

La aplicación DNSE3 está compuesta por una serie de servicios, cada uno encargado de un número limitado de tareas concretas. En [2] se dividen estos servicios en tres tipos: **servicios genéricos de la aplicación**, representados en la Fig. 3.1 por una circunferencia con el borde continuo, son servicios que pueden ser portados a otro programa debido al poco nivel de acoplamiento que tienen con el resto de servicios de la aplicación; **servicios propios de la aplicación**, representados por un rectángulo de trazo continuo, son servicios que realizan tareas muy específicas de la aplicación, por lo que su utilización en otras aplicaciones se antoja complicado; y, por último, **servicios propios de la infraestructura**, representados en Fig. 3.1 por un rectángulo con el borde discontinuo, que están orientados a la gestión de los recursos que ofrece la infraestructura, por lo que habrá que modificar dichos servicios si se pasa a utilizar otra plataforma. Finalmente, se representa con un círculo con trazo discontinuo a la interfaz gráfica desde la que los usuarios podrán gestionar los modelos que se desean simular. Los servicios que componen esta versión de la aplicación son los siguientes:

- **Servicio de Orquestación**. Es el encargado de controlar al resto de servicios que forman la aplicación, además de ser el punto de acceso al sistema desde el cliente Web, por lo que es uno de los servicios propios de la aplicación. Entre sus tareas más relevantes encontramos: compilación de cada uno de los modelos a simular que los usuarios cargan en el sistema, creando para ello un proyecto por modelo compilado y guardando el modelo compilado en el Servicio de Almacenamiento; publicación de tareas en el Servicio de Colas cuando los alumnos solicitan la simulación de un proyecto; obtención de los resultados generados por el Servicio de Informes para entregárselos a los usuarios finales en un comprimido que se crea incluyendo todos los ficheros.
- **Servicio de Colas**. Es el servicio encargado de mantener y ordenar las tareas que le envía un primer servicio y completa un tercero. Tiene un nivel de acoplamiento muy bajo, ya que en DNSE3 se encarga de gestionar las simulaciones pendientes de realizar, pero si se utiliza en otro programa se encargaría de gestionar otra tarea. Realiza una

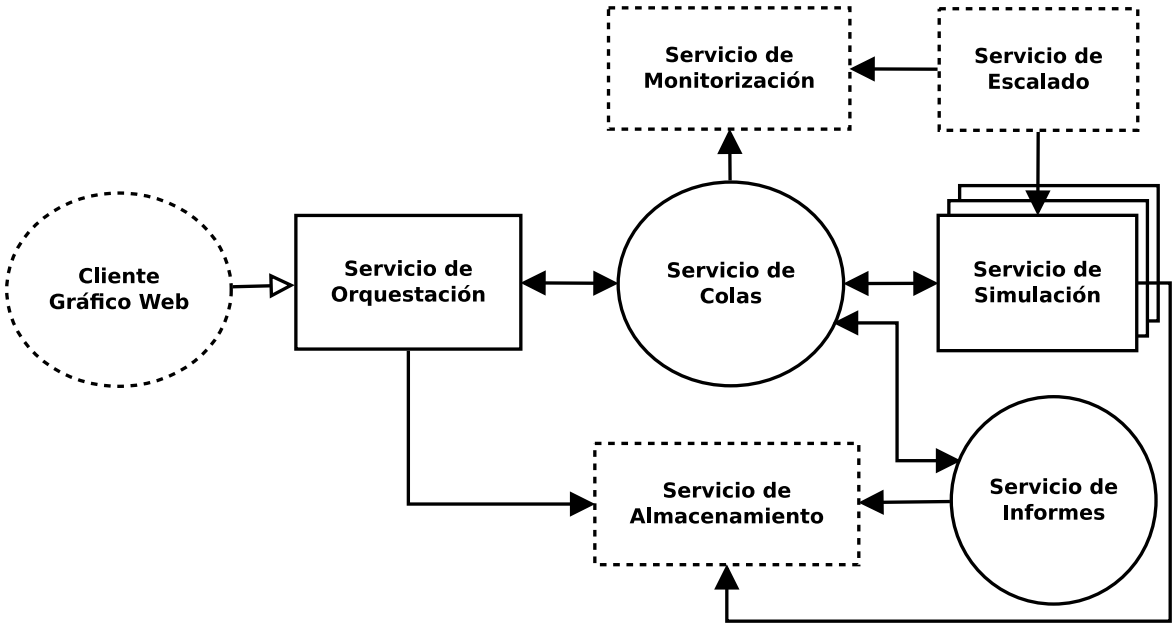


Fig. 3.1: Arquitectura orientada a servicios de la aplicación presentada en [2].

rotación de dos niveles, para que por un lado, ningún usuario acapare los recursos del sistema con la solicitud de un número elevado de tareas, y por otro, que el servicio encargado de consumir las tareas no solicite más de una vez, de forma seguida, la misma tarea. Para la aplicación DNSE3, aparte de gestionar las simulaciones que publica el Servicio de Orquestación, también se encarga de gestionar el orden en el que se realizan los informes.

- **Servicio de Simulación**. Encargado de ejecutar las simulaciones que solicitan los usuarios, utilizando para ello el modelo compilado por el Servicio de Orquestación y el simulador de red de eventos discretos ns-3. Los modelos compilados los recoge del Servicio de Almacenamiento. Al llevar a cabo la tarea computacionalmente más costosa de todo DNSE3, se distribuye en varias réplicas, cada una de ellas se está ejecutando sobre una máquina virtual a la que se le aprovisiona con una CPU virtual y 2GB de memoria RAM [4]. El número de réplicas activas de este servicio se determina mediante el número de tareas pendientes de realizar y la política de escalado elegida. Su ciclo de ejecución consiste en solicitar nuevas tareas de forma periódica al Servicio de Colas, en los espacios de tiempo que no esté realizando ninguna simulación. La comunicación es iniciada desde el Servicio de Simulación puesto que el Servicio de Colas no es consciente del número de réplicas en el que el Servicio de Simulación está dividido. Una vez que finaliza la ejecución de la simulación, guarda el resultado de esta en el Servicio de Almacenamiento.

- **Servicio de Informes**. Encargado de recoger cada uno de los resultados que se han obtenido en cada simulación perteneciente a un trabajo, con el objetivo de formatearlos de una manera adecuada para facilitar el trabajo a los usuarios finales de la aplicación. Para cumplir con su tarea, realiza peticiones de forma periódica al Servicio de Colas solicitando nuevos trabajos para realizar, del mismo modo que hacía el Servicio de Simulación. Cuando obtiene una nueva tarea, tiene que realizar dos consultas por simulación al Servicio de Almacenamiento: una para obtener el resultado de la simulación y otra para recuperar los parámetros con los que se ha obtenido dicho resultado a través de la ejecución del modelo. Para finalizar, una vez que tiene formateados los

resultados, los guarda utilizando el Servicio de Almacenamiento.

- **Servicio de Almacenamiento**. Encargado de la persistencia de los datos de la aplicación, entre los que se encuentran los modelos cargados por los usuarios, los modelos compilados por el Servicio de Orquestación, los resultados de cada simulación, etc. Está totalmente ligado a la plataforma que se esté utilizando para gestionar la nube de cómputo. DNSE3 se está ejecutando sobre una nube gestionada por OpenStack, por lo que para este servicio se está utilizando el proyecto de almacenamiento de objetos *Swift*. Con este servicio se comunican todas las partes de la aplicación que necesiten almacenar información de forma persistente.

- **Servicio de Monitorización**. Se encarga de almacenar y distribuir la métrica del número de tareas pendientes de completar. Este servicio vuelve a estar ligado a la plataforma utilizada para gestionar los recursos de la nube de cómputo, que en el caso de OpenStack esta tarea recae sobre *Gnocchi*. La métrica se consigue mediante la publicación periódica por parte del Servicio de Colas del número de trabajos pendientes de realizar.

- **Servicio de Escalado**. Servicio encargado de adecuar el número de máquinas virtuales a la demanda de trabajos que exista en el sistema mediante la aplicación de un conjunto de reglas de escalado. Este servicio vuelve a depender totalmente de la plataforma utilizada para gestionar la nube de cómputo, que para la aplicación DNSE3 es *Heat*.

3.2. Rediseño de la arquitectura de la aplicación

El servicio computacionalmente más pesado de la aplicación DNSE3 es el Servicio de Simulación, servicio encargado de ejecutar las simulaciones que solicitan los usuarios utilizando para ello el simulador ns-3. La nueva versión de la aplicación DNSE3 ejecuta este servicio sobre un número variable de contenedores tipo Docker, con lo que se consigue reducir el tiempo necesario para aumentar o disminuir dicho número de contenedores activos en el sistema. Por las limitaciones descritas en 2.3, los contenedores que ejecutan el Servicio de Simulación se tienen que desplegar sobre un clúster formado por un número variable de máquinas virtuales, por lo que los retardos asociados a la creación y destrucción de máquinas virtuales seguirán existiendo.

En la Fig. 3.2 se puede observar los servicios que forman la aplicación DNSE3 en su nueva versión. De la misma forma que sucedía en [2] se pueden clasificar dichos servicios en varios tipos: **servicios específicos de la aplicación**; **servicios genéricos de la aplicación**; y por último, **servicios independientes de la aplicación**. Para representar gráficamente cada uno de estos servios se está utilizando la misma representación gráfica que se utilizaba en Fig. 3.1. Los servicios que forman la nueva versión de DNSE3 son los siguientes:

- **Servicios genéricos de la aplicación**. Son servicios desarrollados para realizar alguna tarea específica de DNSE3, pero por su diseño se pueden utilizar para realizar una tarea similar en otra aplicación.

  - **Servicio de Colas**. Realiza la misma función que en la versión anterior de la aplicación, con la salvedad de que la métrica del número de trabajos pendientes de realizar la publica sobre el Servicio de Monitorización y Escalado.
  - **Servicio de Informes**. Este servicio se ha actualizado para que utilice de forma eficiente las nuevas características del Servicio de Almacenamiento. Gracias a esta

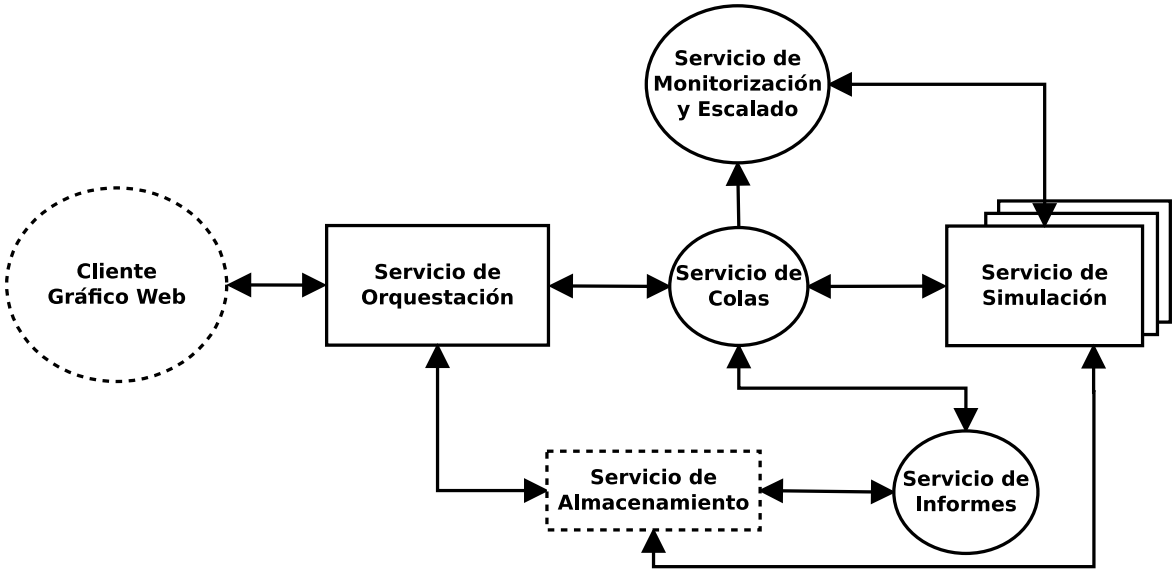


Fig. 3.2: Arquitectura orientada a servicios de la nueva versión de la aplicación DNSE3 basada en la arquitectura presentada en [2].

modificación recoge todos los resultados de un mismo trabajo de una sola vez. Para ello solicita la descarga del directorio donde se han almacenado todos los resultados del trabajo.

- **Servicio de Monitorización y Escalado**. Siendo consciente del valor de una métrica y aplicando una política específica es capaz de gestionar el número de máquinas virtuales que forman un clúster y los contenedores que se ejecutan sobre dicho clúster. En la aplicación DNSE3 se encarga de recoger la publicación periódica del valor de las tareas pendientes de completar que realiza el Servicio de Colas y, a partir de dicho valor y del número de máquinas virtuales y de contenedores que estén activos en el clúster, escalar los recursos consumidos para hacer un uso eficiente de los mismos. Este servicio se ha desarrollado desde cero para esta nueva versión y como se acaba de indicar desempeña la tarea que antes realizaba *Gnocchi* además de utilizar *Heat* para llevar a cabo la creación y destrucción de máquinas virtuales.

- **Servicios específicos de la aplicación**. Son servicios especialmente diseñados para esta aplicación por lo que su uso en otra se antoja complicado por el coste que supondría adaptarlos.

  - **Servicio de Orquestación**. Se encarga de las mismas tareas que tenía que realizar en la versión anterior de la aplicación. Se aprovecha de las características del nuevo Servicio de Almacenamiento para no tener que crear el comprimido de los resultados formateados por el Servicio de Informes.
  - **Servicio de Simulación**. Lleva a cabo la ejecución de las simulaciones publicadas en el Servicio de Colas de la misma forma que lo estaba haciendo en la versión anterior de DNSE3. Este servicio realiza la tarea computacionalmente más costosa, por lo que van a realizarse las modificaciones necesarias para que se ejecute dentro de contenedores Docker que faciliten su escalado.

- **Servicios independientes de la aplicación**. Son servicios de terceras partes que se utilizan dentro de la aplicación para cubrir una tarea específica. La ventaja de estos

servicios es que no es necesario destinar recursos para desarrollarlos.

- **Servicio de Almacenamiento**. Se encarga de mantener la persistencia de los datos de la aplicación. En la nueva versión de la aplicación DNSE3 se pasa de utilizar un servicio propio de la plataforma de gestión de la nube de cómputo a usar ownCloud[2].

Para escalar el número de réplicas que ejecutan el Servicio de Simulación se ha pasado de utilizar servicios propios de la plataforma, a desarrollar un servicio propio de la aplicación. Dicho cambio estuvo motivado en un principio por las limitaciones existentes en los proyectos propios de OpenStack destinados a trabajar con contenedores. Con el desarrollo del Servicio de Monitorización y Escalado se consigue obtener más flexibilidad a la hora de aplicar la política de escalado deseada, puesto que se desarrolla a medida para DNSE3. Con este cambio se consigue aumentar el número de veces que se evalúa la cantidad de trabajos pendientes de realizar que el que permitía el Servicio de Métricas utilizado en la versión anterior de la aplicación.

El Servicio de Almacenamiento de esta nueva versión sufre un cambio importante puesto que pasa de utilizar un servicio propio de la plataforma de gestión de la nube computacional como era *Swift*, a utilizar un servicio desarrollado por un tercero, ownCloud. Se realiza este cambio para aprovechar las características que proporciona ownCloud, como la descarga de un directorio, junto con todo su contenido, en un fichero con formato ZIP[3]. Esta característica será utilizada en Servicio de Informes para intentar mejorar su rendimiento y en el Servicio de Orquestación para reducir su carga de trabajo.

3.2.1. Servicio de Simulación

Internamente el Servicio de Simulación no ha sufrido grandes cambios en lo referente a las funciones que desarrolla. Para cumplir con el objetivo de llevar a cabo las simulaciones que solicitan los usuarios realiza una serie de pasos: en primer lugar, el Servicio de Simulación solicita periódicamente al Servicio de Colas nuevos trabajos para realizar; cuando consigue un trabajo comprueba que es válido y descarga el ejecutable de la simulación previamente compilado por el Servicio de Orquestación desde el Servicio de Almacenamiento; cuando obtiene el modelo compilado ejecuta la simulación utilizando para ello ns-3; una vez que completa la simulación se guarda en el Servicio de Almacenamiento tanto el resultado de la simulación como los parámetros que se han utilizado para conseguir ese resultado; cuando finaliza el proceso para almacenar los resultado conseguidos se vuelve a reiniciar el ciclo del Servicio de Informes volviendo a solicitar al Servicio de Colas un nuevo trabajo que realizar.

3.2.2. Servicio de Monitorización y Escalado

Una vez que se determinó que el Servicio de Simulación se iba a ejecutar dentro de contenedores Docker, se necesita determinar la forma de desplegar un número variable de este tipo de contenedores para que se adecuen los recursos consumidos a las simulaciones que están pendientes de realizarse en el Servicio de Colas. En la versión presentada de este TFG este escalado se realiza a través del Servicio de Monitorización y Escalado. Este servicio podrá controlar tanto el número de contenedores que empaquetan el Servicio de Simulación como las máquinas virtuales que forman el clúster sobre el que se ejecutan los contenedores.

[2] https://owncloud.org/

[3] http://www.iana.org/assignments/media-types/application/zip

El desarrollo del Servicio de Monitorización y Escalado ha venido motivado porque los proyectos de escalado de contenedores propios de la plataforma de OpenStack, o bien se quedaban cortos de funciones (*Magnum* puede adecuar el número de máquinas virtuales activas en el clúster a partir de una métrica pero no tiene ninguna opción para controlar el número de contenedores), o directamente no se podían instalar en la nube de cómputo privada por posibles problemas de compatibilidad con los proyectos que actualmente se están ejecutando en dicha nube (como sucede con el proyecto *Zun*, que no especifica su compatibilidad con las máquinas virtuales que creadas por *Nova*).

Al recoger el valor de las tareas pendientes de realizar directamente de las publicaciones periódicas (cada dos segundos) del Servicio de Colas se permite ajustar la cantidad de recursos utilizados por el Servicio de Simulación de forma más rápida a como se estaba realizando en la versión anterior. En dicha versión se utilizaba el servicio de métricas propio *Gnocchi*, que permitía recoger el valor del número de simulaciones esperando en el Servicio de Colas cada sesenta segundos. La cantidad de recursos utilizados vendrá determinada por el número de máquinas virtuales activas en el clúster cuyo número variará, a su vez, de los contenedores que estén activos.

En Fig. 3.3 se puede ver el número de comunicaciones existentes alrededor del Servicio de Monitorización y Escalado. En la misma figura se representan las máquinas virtuales con un rectángulo violeta (cuyo número varía entre 1 y $K+1$ puesto que la máquina virtual *manager* siempre está activa), el enjambre que se extiende por todas las máquinas virtuales con un rectángulo azul y cada uno de los contenedores que se despliegan en el enjambre donde se está ejecutan el Servicio de Simulación con un rectángulo rojo. Los contenedores se reparten por las distintas máquinas del clúster según la política que se le indique a Docker Swarm. Además, se puede ver que por un lado, y como se acaba de indicar, el Servicio de Colas publica el número de tareas esperando a ser realizadas. Con este valor el Servicio de Monitorización y Escalado será capaz de decidir si es necesario aumentar o disminuir el número de contenedores activos en el clúster. La acción de escalado del número de contenedores activos la realiza comunicándose con el *Swarm Manager* del clúster. El escalado del número de máquinas virtuales activas en el clúster se realiza mediante *Heat*. Este proyecto permite orquestar la creación y destrucción de máquinas virtuales a partir de una plantilla donde se indica la cantidad de recursos que puede utilizar, así como la imagen de sistema operativo a utilizar. Por ello, se hace uso del proyecto *Heat* en la nueva versión de DNSE3 para incrementa o reducir el número de máquinas virtuales con las que se esté reservando recursos para la ejecución del Servicio de Simulación. Además, *Heat* permite ejecutar un *script* al finalizar la creación de la máquina virtual. Esta cualidad la aprovechará DNSE3 para que las nuevas máquinas se introduzcan dentro del enjambre sobre el que se ejecutan las distintas réplicas del Servicio de Simulación. La plantilla con toda la información que necesita *Heat* para inicializar las máquinas virtuales se le proporciona en la nueva versión de DNSE3 mediante un fichero en formato JSON[4].

Política aplicada en el Servicio de Monitorización y Escalado

Como es necesario seguir utilizando máquinas virtuales en la nueva versión de la aplicación, con el objetivo de reducir al mínimo los retardos asociados a la creación y destrucción de las máquinas virtuales, se decide utilizar un número bajo de máquinas virtuales sobre las que se pueda lanzar un número elevado de contenedores. De esta forma, las máquinas virtuales utilizadas para ejecutar los contenedores tendrán reservados 10 CPUs virtuales y 8GB de memoria RAM.

[4]https://tools.ietf.org/html/rfc8259

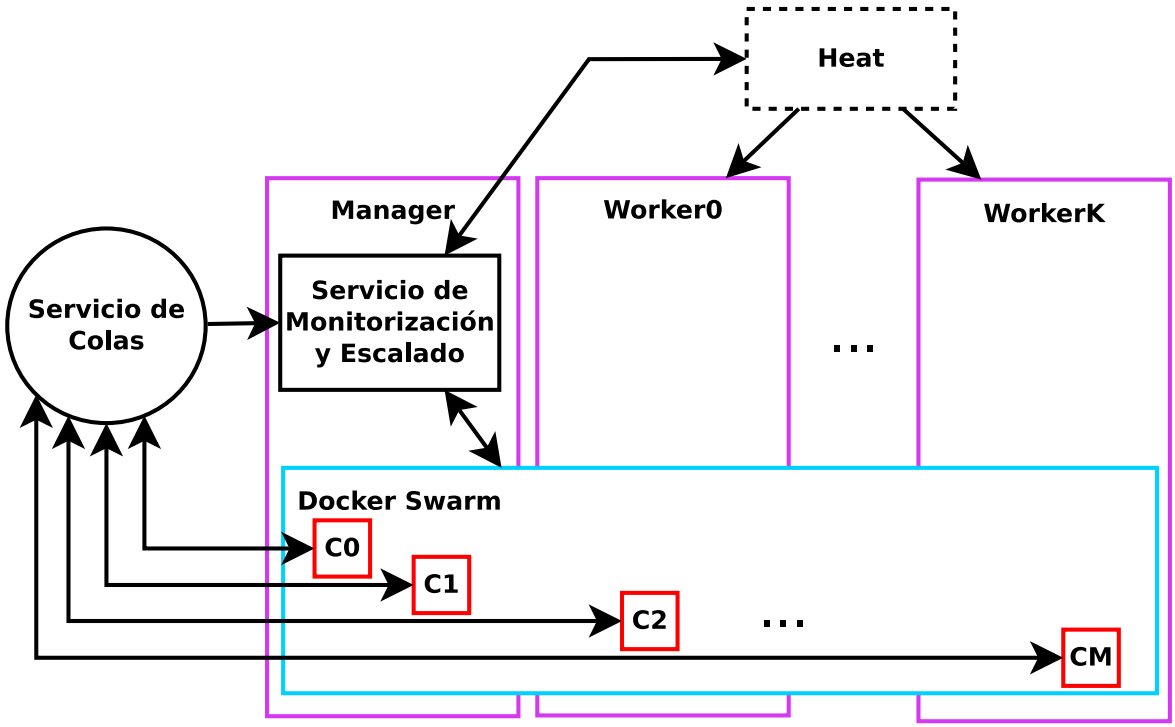


Fig. 3.3: Comunicaciones que existen alrededor del Servicio de Monitorización y Escalado.

Para evaluar si el número de contenedores ejecutando el Servicio de Simulación es el adecuado, teniendo en consideración el valor de tareas esperando en el Servicio de Colas, el Servicio de Monitorización y Escalado necesita que se le proporcione una política que le indique qué hacer. Esta política se podrá ajustar para que se consuman más o menos recursos. La política utilizada en la nueva versión de la aplicación DNSE3 será similar a la presentada como P_{cons} en [4], que incrementaba el número de máquinas virtuales ejecutando el Servicio de Simulación de una en una. Se toma esta decisión ya que no es una política que utilice agresivamente los recursos de la nube de cómputo, garantiza un tiempo necesario para conseguir los resultados en simulaciones de distinto número de repeticiones razonable si se compara con los tiempos conseguidos con políticas más agresivas utilizando para ello un menor uso de recursos, y, de todas las políticas de escalado presentadas en [4] era la que suponía unos costes económicos más bajos. Por todo ello, se desarrolló la política de escalado del número de contenedores siguiendo las siguientes reglas:

- El número de contenedores activos en el sistema se aumenta una unidad cuando se detecte que el valor del número de tareas pendientes de realizarse (sim_{pend}) entre el número de contenedores activos en el sistema ($Docker_{act}$) es superior o igual a un umbral establecido, T:

$$If\left(\frac{sim_{pend}}{Docker_{act}} \geq T\right) \rightarrow Docker_{act} = ++Docker_{act}$$

  Se realizará esta acción siempre y cuando el número de contenedores activos en el clúster no supere al valor de multiplicar el número de máquinas activas en el clúster, MV_{act}, por el número de núcleos que dispone cada una de ellas, Num_{CPU}.

- La cantidad de contenedores activos en el clúster se reducirá a la mitad, o al entero superior en el caso de que la división de como resultado un número decimal, cuando se

cumpla:

$$If\left(\frac{sim_{pend}}{Docker_{act}} < \frac{T}{2}\right) \rightarrow Docker_{act} = ceil\left(\frac{Docker_{act}}{2}\right)$$

Esta reducción se realizará hasta llegar al número mínimo de contenedores activos en el clúster que se ha establecido en uno.

Una vez que se ha determinado la política que controlará el escalado de los contenedores faltaría buscar otra para determinar el número de máquinas virtuales activas en el clúster. Como se ha indicado previamente, el número de máquinas virtuales está directamente relacionado con la cantidad de contenedores activos en el sistema. La política que se utilizará en el Servicio de Monitorización y Escalado para la nueva versión de DNSE3 es la siguiente:

- Se inicia una nueva máquina virtual cuando se cumpla que el número de simulaciones pendientes entre el número de contenedores activos en el clúster más la mitad de los contenedores que admite cada *host* sea superior o igual al umbral:

  $$If\left(\frac{sim_{pend}}{Docker_{act} + \frac{Num_{CPU}}{2}} \geq T\right) \rightarrow MV_{act} = MV_{act} + 1$$

  Por ello, solamente se realiza un incremento del número de máquinas virtuales cuando con el valor actual del número de simulaciones esperando nos permita desplegar la mitad de los contenedores que se podrían ejecutar dentro de cada máquina virtual. Por lo que se acaba de ver, la política de iniciar una nueva máquina es poco agresiva con lo que se espera conseguir desperdiciar el menor número de recursos posibles de la nube de computación donde está actualmente activo DNSE3.

- Se reduce en una unidad el número de máquinas virtuales del clúster cuando se cumpla:

  $$If\left(Docker_{act} < (MV_{act} - 1)\, Num_{CPU}\right) \rightarrow MV_{act} = MV_{act} - 1$$

  Con esta condición se consigue liberar recursos de la nube de computación que no se estén utilizando, con lo que el número de recursos desperdiciados se reduce.

En la política que se acaba de explicar se utiliza el mismo valor del umbral T para escalar tanto máquinas virtuales como contenedores. Dicho valor, para el caso de contenedores, determinará el número máximo de simulaciones que se desean asignar a cada contenedor, por lo que determinará la velocidad de escalado de clúster. La búsqueda del valor apropiado para este umbral se realizará en el capítulo 4. Además, aunque para esta versión de la aplicación de DNSE3 se haya utilizado esta política concreta se podría diseñar cualquier otra si se desea que haga un uso más o menos agresivo de los recursos que puede utilizar el Servicio de Simulación de la aplicación.

3.2.3. Servicio de Almacenamiento

El Servicio de Almacenamiento de la nueva versión de DNSE3 pasa de utilizar *Swift* a ownCloud. Con este cambio se pretende agregar nuevas funcionalidades que permitan mejorar el rendimiento ofrecido por el resto de servicios de la aplicación. Para ello habrá que diseñar una serie de métodos que permitan al resto de servicios que forman la aplicación comunicarse con el Servicio de Almacenamiento.

En primer lugar, se deberá diseñar un método que permita el almacenamiento tanto de los resultados que se generan en el Servicio de Simulación, como el ejecutable que compila el

Servicio de Orquestación a partir del *script* proporcionado por los usuarios y los resultados formateados por el Servicio de Informes. Este método deberá comprobar la existencia de la ruta de directorios donde se desea almacenar el objeto y si no existe dicha ruta crear los directorios que falten para poder realizar el salvado de los datos.

Directamente relacionado con el método de subida de datos se necesita un método que cree directorios. Almacenar los datos en una estructura de directorios permitirá descargar todos los ficheros incluidos en una carpeta con una sola operación.

Otro de los métodos básicos que el Servicio de Almacenamiento deberá tener será el de descarga de un fichero. Este fichero puede ser tanto el ejecutable que necesita el Servicio de Simulación para completar las distintas simulaciones como los resultados generados con cada simulación. Además de este tipo de descarga se deberá crear un método que permita obtener un directorio completo.

Cuando el usuario elimine el proyecto desde la interfaz Web de la aplicación se deberán borrar los datos de ese proyecto. Para ello, se diseñará un método que permita la eliminación de directorios. Este método también podrá se utilizado para borrar un fichero concreto dentro del sistema.

Además de diseñar los métodos destinados a la manipulación de ficheros y directorios se crearán otros métodos para la manipulación de usuarios dentro de ownCloud. Estos métodos permitirán la creación, modificación y eliminación de usuarios. Aunque en la versión de DNSE3 desarrollada en este TFG no se van a utilizar, se abre la posibilidad de que cada usuario de la aplicación DNSE3 tenga disponible un usuario propio en ownCloud.

3.2.4. Resto de servicios de la nueva versión

El **Servicio de Orquestación** de la nueva versión de la aplicación DNSE3 se encarga de las mismas funciones que tenía en la versión antigua, ya que sigue gestionando al resto de servicios. Continúa comunicándose con el Servicio de Almacenamiento para guardar los *scripts* que cargan los usuarios en el sistema, además de compilar dichos *scripts* y almacenar el modelo compilado. Para el Servicio de Orquestación el cambio del Servicio de Almacenamiento es transparente cuando realiza salvado de datos puesto que las llamadas a los métodos que utiliza para comunicarse con ownCloud son iguales a las que utilizaba para comunicarse con Swift. El cambio más relevante realizado en este servicio es la descarga de los ficheros de resultados formateados por el Servicio de Informes. En la versión anterior del Servicio de Orquestación necesitaba descargar uno a uno cada fichero de resultados y luego empaquetar todos los resultados en un ZIP para enviárselo a los usuarios de la aplicación. En la nueva versión, aprovechándose de la utilización de ownCloud, se descarga el directorio completo de resultados formateados en un comprimido. Una vez que obtiene el comprimido lo envía directamente al usuario final de la aplicación sin necesidad de realizar más tareas.

El **Servicio de Informes** se encarga de crear una serie de ficheros que contendrán los resultados de las simulaciones pertenecientes a un trabajo con el objetivo de facilitar la utilización de estos resultados a los usuarios. El ciclo de trabajo es el mismo en las dos versiones de la aplicación. La diferencia entre versiones es la descarga de los ficheros de resultados, ya que en la versión anterior se necesitaba realizar como mínimo dos peticiones por cada simulación y en la nueva versión con una petición se obtienen todos los ficheros de resultados empaquetados en un ZIP. Con esta modificación se pasa, en el caso más extremo para el *script* `P2-ns-QueueingTheory-m_m_1-m_g_1` sin modificar, de realizar 20.000 peticiones del Servicio de Informes al de Almacenamiento a una única petición, lo que reduce el tiempo necesario para obtener todos los resultados y generar los informes.

El **Servicio de Colas** de la aplicación DNSE3 se encarga de gestionar el orden en el que se realizan las simulaciones y los informes. Este servicio es el que menos cambios presenta entre las dos versiones de la aplicación, puesto que solamente se ha necesitado modificar el lugar donde publica la métrica, pasándose de utilizar *Gnocchi* a directamente el Servicio de Monitorización y Escalado.

3.3. Implementación de la nueva arquitectura

3.3.1. Servicio de Simulación

Una vez que se ha decido utilizar contenedores tipo Docker para empaquetar el *software* del Servicio de Simulación y todas sus dependencias, hay que crear la imagen base de contenedor de la que partirán todos los contenedores que ejecuten este servicio. Para ello, lo primero que hay que crear es un fichero `Dockerfile` donde se describen los pasos que debe ejecutar el motor Docker para la creación de la imagen. El primero de estos pasos será elegir la imagen oficial en la que se va a basar. Como la versión anterior del Servicio de Simulación se ejecutaba sobre máquinas virtuales que utilizaban el sistema operativo Ubuntu `18.04`, se ha mantenido esta elección de sistema operativo y versión, ya que se comprobó que la ejecución sobre ese sistema no generaba incompatibilidades. Una vez escogida la imagen base a utilizar, se tiene que determinar qué dependencias necesita utilizar el programa que se va a ejecutar dentro del contenedor. El Servicio de Simulación depende de ns-3 y JRE (*Java Runtime Environment*) a través de OpenJDK[5]. La versión utilizada del simulador de red ns-3 es la `3.18.2`, compatible con los *scripts* utilizados en las asignaturas que hacen uso de la aplicación DNSE3. Para la utilización de ns-3 por parte del Servicio de Simulación, hay que instalar las dependencias que se indican en [12], entre las que encontramos compiladores de `C` y `C++`. JRE es necesario ya que el Servicio de Simulación está desarrollado en Java, por lo que necesita de una máquina virtual Java para su ejecución. Una vez instaladas todas las dependencias se copia al interior de la imagen de contenedor un JAR[6] que tendrá empaquetado el Servicio de Simulación y todas sus dependencias (como es la biblioteca de Restlet[7]). Por seguridad, y aunque no es estrictamente necesario puesto que los contenedores no pueden acceder a recursos que no se les permita acceder, se crea un usuario sin privilegios administrativos para ejecutar el Servicio de Simulación. El último paso que se tiene que dar para finalizar el fichero `Dockerfile` es indicar la acción que tiene que realizar el contenedor al ser iniciado, que en el caso que nos ocupa será la ejecución del Servicio de Simulación con los parámetros que se necesiten. Los parámetros se establecen mediante variables de entorno propias del contenedor para que se puedan modificar en su despliegue. Estos parámetros son la dirección y puerto que utiliza el Servicio de Colas y la cantidad de tareas que puede recoger cada Servicio de Simulación. Todos los pasos seguidos para la construcción de la imagen de contenedor están indicados en el Anexo A.

Generado el fichero `Dockerfile` ya se puede crear la imagen base de contenedor con el Servicio de Simulación. Para ello se utiliza la aplicación propia de Docker `docker build`. Esta herramienta crea la imagen base de contenedor siguiendo los pasos establecidos en `Dockerfile`. Durante este proceso crea imágenes intermedias o puntos de retorno por si se produce algún error del paso actual al siguiente. Si se produce un error, la utilidad `docker build` aborta la creación de la imagen, pero en el momento que se vuelva a iniciar la construcción de la misma imagen base de contenedor, *docker build* utilizará una de las imágenes

[5] https://openjdk.java.net/
[6] https://docs.oracle.com/javase/1.5.0/docs/guide/jar/jar.html
[7] https://restlet.com/open-source/

intermedias para no tener que volver a realizar todos los pasos. Una vez que se ha completado la creación de la imagen de contenedor ya se puede utilizar para desplegar contenedores en la máquina donde se haya realizado todo el proceso, pero si se quiere que esta imagen base se pueda utilizar desde otros *hosts*, se tiene que publicar en un repositorio. Para almacenar la imagen base de contenedor en el repositorio Docker Hub se utiliza otra utilidad de Docker, `docker push`, necesitándose una cuenta de usuario para que permita realizar dicho almacenamiento.

3.3.2. Servicio de Monitorización y Escalado

Por uniformidad con el resto de los servicios de la aplicación DNSE3 el Servicio de Monitorización y Escalado fue desarrollado en Java. Este servicio cuenta con dos partes bien diferenciadas: por un lado, la parte servidora, que se encarga de atender las peticiones que le llegan del Servicio de Colas con el valor de las métricas; por otro lado, la evaluación de la métrica recibida desde el Servicio de Colas y utilizada para adecuar la cantidad de recursos utilizados por las distintas réplicas del Servicio de Simulación.

La parte servidora del servicio ha sido desarrollada siguiendo el estilo de arquitectura *software* REST, utilizando para ello el *framework* Restlet[8]. Por este motivo, el Servicio de Colas realiza una petición `HTTP PUT` al Servicio de Monitorización y Escalado cada vez que desea publicar el valor del número de simulaciones pendientes de realizar. Esta petición llegará al Servicio de Monitorización y Escalado a través del puerto `8084` y al recurso `/v0.1/{queue}` donde `{queue}` se sustituirá por `colaSimulaciones` para la aplicación DNSE3. En dicha petición se encontrará en formato JSON tanto el valor de las tareas pendientes como el instante temporal en el cual se realizó la medición. Una vez que la parte servidora obtiene este objeto, comprueba que su formato sea correcto y le envía los datos a la parte del servicio encargada del escalado de los recursos.

La parte del Servicio de Monitorización y Escalado que tiene como objetivo determinar la cantidad de recursos a utilizar tiene un método destacado, `run()`. Dicho método se ejecuta en bucle y realiza comprobaciones para determinar si el número de contenedores y máquinas virtuales es el apropiado, utilizando para ello una política concreta. La comprobación se realiza siguiendo una espera exponencial. El tiempo de espera vendrá determinado por la existencia de cambios en el número de recursos utilizados por el Servicio de Simulación. Si no se detecta ningún cambio, este tiempo de espera se irá incrementando hasta un máximo de 16 segundos entre iteración.

3.3.3. Servicio de Almacenamiento

El Servicio de Almacenamiento de la aplicación DNSE3 se encarga de la persistencia de los datos. Se entiende como datos de la aplicación los *scripts* cargados por los alumnos, el modelo compilado del *script* que se acaba de citar, los resultados de cada una de las simulaciones que forman parte de cada trabajo y los resultados formateados por el Servicio de Informes.

En la nueva versión de DNSE3 se pasa de utilizar un servicio propio de OpenStack, el proyecto de almacenamiento de objetos *Swift*, a una alternativa de propósito más general, como es el caso de ownCloud. ownCloud es un servicio para compartir ficheros, desarrollado en PHP y JavaScript, licenciado bajo AGPLv3 y se puede instalar en máquinas con una memoria RAM superior a 128MB, aunque la propia ownCloud recomienda utilizar un mínimo de 512MB [24]. El motivo principal por el que se porta el Servicio de Almacenamiento desde

[8] https://restlet.com/

un proyecto propio de OpenStack a otro de terceros es para intentar mejorar el rendimiento del Servicio de Informes.

El Servicio de Informes necesita recoger todos los resultados de las simulaciones que se hayan realizado dentro de cada trabajo para entregar a los usuarios estos resultados formateados. Por este motivo, en la versión de DNSE3 que utiliza Swift, el Servicio de Informes necesita realizar al menos dos peticiones HTTP por cada simulación: una para recoger los parámetros con los que se ha ejecutado la simulación y las siguientes para obtener los ficheros de salida que se hayan generado al llevar a cabo la simulación. En la versión que utiliza ownCloud se pasará a realizar una única petición, solicitando la descarga del directorio que contiene todos los ficheros de resultados del trabajo del cual se va a realizar el informe. ownCloud empaquetará dicho directorio, con todos sus subdirectorios y ficheros, en un ZIP y se lo enviará al Servicio de Informes, ahorrando un número elevado de comunicaciones entre Informes y Almacenamiento.

ownCloud permite utilizarse sobre contenedores de tipo Docker proporcionando, a través de Docker Hub, las imágenes base de contenedor, además del fichero[9] necesario para inicializar el sistema. En dicho fichero se podrá configurar las contraseñas que utilizan la base de datos y el usuario con privilegios administrativos, así como el puerto que va a utilizar ownCloud. Para conseguir la persistencia de datos utilizando contenedores se hace uso de *volúmenes Docker* [25]. Estos *volúmenes* se almacenan en una parte del sistema de ficheros de la máquina anfitriona[10] no siendo accesibles para otros procesos del sistema. De esta forma si el contenedor se detiene o se elimina, los datos no se pierden y podrán ser accesibles más adelante.

Los servicios de Orquestación, Simulación e Informes utilizan el Servicio de Almacenamiento. Por este motivo, en la versión anterior de la aplicación se desarrolló una serie de métodos para facilitar la comunicación con *Swift*. Para adaptarse a ownCloud se tuvo que modificar dichos métodos además de crear otros nuevos, como el de la creación de un directorio o la descarga de un directorio completo como un ZIP. La gestión de ficheros en ownCloud se realiza a través del estándar WebDAV, por lo que las comunicaciones no son puramente REST, ya que se necesitan utilizar métodos específicos de WebDAV como MKCOL (creación de un directorio) y PROPFIND (obtención de los recursos de un directorio). Aparte de los métodos destinados a la gestión de ficheros, se crearon métodos para la gestión de usuarios por si en un futuro se desea que cada alumno que utiliza la aplicación tenga un usuario propio en el Servicio de Almacenamiento. Para la creación del cliente con el que gestionar tanto ficheros como usuarios de ownCloud se hizo uso de *Apache HttpClient*[11] para los métodos que utilicen peticiones estándar de HTTP puesto que en el desarrollo de la versión anterior de DNSE3 que si se utilizaba el cliente de Restlet no se conseguían liberar correctamente los recursos utilizados. En los métodos en los que se necesitaba utilizar métodos específicos de WebDAV se hizo uso de *Jackrabbit WebDAV Library*[12].

3.4. Despliegue de los diferentes servicios

Como se acaba de ver, la nueva versión de DNSE3 está compuesta por seis servicios. En esta sección se va a explicar cómo se están distribuyendo estos servicios sobre la nube de cómputo privada gestionada por OpenStack del grupo de trabajo donde se está llevando a

[9] https://github.com/owncloud-docker/server/blob/master/docker-compose.yml
[10] Si se está utilizando un sistema GNU/Linux en la ruta `/var/lib/docker/volumes/`
[11] https://hc.apache.org/httpcomponents-client-ga/index.html
[12] https://jackrabbit.apache.org/jcr/components/jackrabbit-webdav-library.html

Servicio	Puerto	Parámetros de entrada
Orquestación	8080 (Web)	Dirección IP y puerto de Colas
	8081 (Notificación)	
Colas	8082	Dirección IP y puerto de Notif. Orques.
Simulación	-	Dirección IP y puerto de Colas
		Límite de simulaciones por contenedor
Monitorización y Escalado	8084	Puerto utilizado de escucha (8084)
		Máximo de contenedores por MV
		Máximo de MV en el clúster
		Nombre del servicio de contenedores
Informes	-	Dirección IP y puerto de Colas
		Límite de informes a realizar en paralelo
Almacenamiento	9090	-

Tabla 3.1: Resumen de la relación existente entre cada servicio y puertos que utiliza. También se indican los parámetros que se les tiene que pasar a cada servicio para inicializarlo.

cabo la ejecución de la aplicación. A continuación se describirá los recursos utilizados por cada servicio y las dependencias que necesitan para su inicialización, además del resumen que se realiza en la tabla 3.4 con la relación de puertos y parámetros de entrada de cada uno de los servicios. Además, en la tabla 3.4 se puede ver un resumen de los datos de despliegue de cada uno de los servicios.

- El Servicio de Orquestación de la aplicación DNSE3 se ejecuta sobre una máquina virtual que dispone de 2 vCPUs, 2GB de memoria RAM y 20GB de disco. El sistema operativo utilizado es Ubuntu 18.04. Este servicio necesita el simulador ns-3 para compilar los *scripts* que carguen en el sistema los usuarios, JRE para ejecutar el propio servicio y un sistema gestor de base de datos como es *MySQL*[13] para almacenar la relación entre usuarios y datos generados. El Servicio de Orquestación escucha peticiones entrantes en los puertos `8080` y `8082`, reservando el primero para la comunicación con el cliente Web y el segundo para el Servicio de Colas. Necesita los ficheros `Makefile` para la compilación de los modelos y `almacen.properties` para la comunicación con el Sistema de Almacenamiento.

- El Servicio de Colas se ejecuta sobre la misma cantidad de *hardware* virtual que el que utiliza Orquestación: 2 vCPUs, 2GB RAM y 20GB de disco. También utiliza Ubuntu 18.04 como sistema operativo. Necesita JRE para iniciarse ya que está desarrollado en Java. Escucha en el puerto `8081` las peticiones que le llegan del Servicio de Informes y de Simulación con el objetivo de recoger nuevas tareas.

- El Servicio de Informes utiliza una configuración de 1 vCPU, 2GB de memoria RAM y 10GB de disco además del sistema operativo Ubuntu 18.04. No precisa de más dependencias que JRE para su ejecución y el fichero `almacen.properties` para comunicarse con el Sistema de Almacenamiento. También necesita el fichero `almacen.properties` para obtener la dirección del Servicio de Monitorización y Escalado.

- El Servicio de Simulación y el Servicio de Monitorización y Escalado están siendo ejecutados sobre una misma máquina virtual, que dispone de 10 vCPUs, 8GB de RAM y 10GB de disco con Ubuntu 18.04. Esto sucede solamente en la máquina virtual que se

[13] https://www.mysql.com/

crea inicialmente. En las máquinas creadas bajo demanda para aumentar la cantidad de recursos del clúster, aunque la configuración *hardware* es la misma que la máquina creada inicialmente, solamente se ejecutan los contenedores con el Servicio de Simulación empaquetado.

- Por un lado, el Servicio de Simulación se ejecuta sobre contenedores Docker, por lo que se necesita instalar el cliente de Docker como dependencia, al que habrá que configurar[14] para que acepte peticiones HTTP a través del puerto `8085` y se pueda controlar desde el Servicio de Monitorización y Escalado. Cuando el Servicio de Simulación precisa de más recursos se crean máquinas virtuales con la misma configuración de *hardware* virtual que la que se acaba de describir, que contarán con el sistema operativo Ubuntu 18.04, el cliente de Docker y la imagen base de contenedor con el Servicio de Simulación. Las comunicaciones existentes entre las máquinas virtuales se observan en la figura 3.4. En dicha figura se puede ver que dentro de cada máquina virtual (rectángulo con el borde violeta) se ejecuta un número variable de contenedores (rectángulo con el borde rojo) y dentro de estos contenedores se ejecuta el Servicio de Simulación (rectángulo de borde negro) donde se llevará a cabo la ejecución de las simulaciones que soliciten los usuarios de la aplicación. Desde la máquina *manager* se controla el número de contenedores que se están ejecutando en el clúster.
- Por otro lado, el Servicio de Monitorización y Escalado necesita JRE para ejecutarse y el puerto `8084` para recibir las métricas del Servicio de Colas. También necesita el fichero `almacen.properties` puesto que en él está la dirección de la máquina mánager del clúster donde se ejecuta el Servicio de Simulación, que en el caso actual es la misma máquina en la que se está ejecutando el Servicio de Monitorización y Escalado.

- El Servicio de Almacenamiento de la aplicación DNSE3 se ejecuta en una máquina virtual con 4 vCPUs, 8GB de memoria RAM y 10GB de disco. Además, se le añade un volumen para garantizar la persistencia de los datos. El sistema operativo utilizado es Ubuntu 18.04 al que se le ha instalado el cliente de Docker para que pueda ejecutar ownCloud empaquetado en contenedores Docker. El puerto utilizado por este servicio es el `9090`.

3.5. Conclusiones

La nueva versión de la aplicación DNSE3 basada en contenedores se ha dividido en seis Servicios. De estos servicios el que desempeña la tarea computacionalmente más costosa es el Servicio de Simulación, por lo que se ha creado una imagen base de contenedor para que se pueda replicar en un número variable de contenedores, con lo que se conseguirá paralelizar la ejecución de las simulaciones y conseguir los resultados de estas en un tiempo razonable. Para conseguir automatizar el número de recursos que se utilizan para realizar las simulaciones, se ha desarrollado el Servicio de Monitorización y Escalado que permite gestionar el número de contenedores que se ejecutan sobre un clúster formado por un número variable de máquinas virtuales.

Otro de los servicios que ha sufrido un cambio relevante con la nueva versión es el Servicio de Almacenamiento. Este servicio pasa de utilizar un proyecto propio de OpenStack como es

[14] Se siguieron los pasos de [26] para que el cliente de Docker acepte peticiones HTTP a través de la red

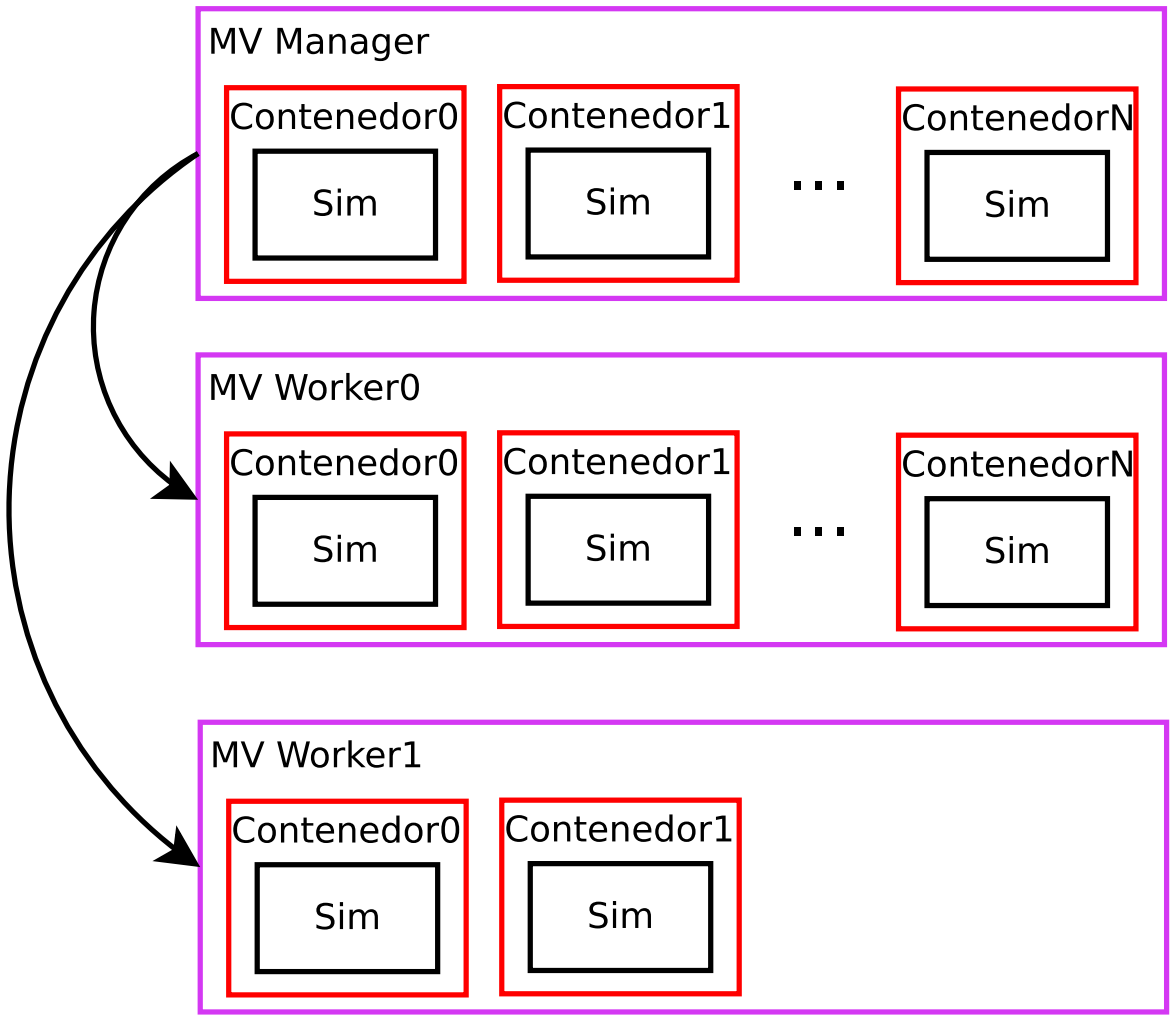


Fig. 3.4: Distribución del Servicio de Simulación usando una arquitectura mixta basada en máquinas virtuales y contenedores.

Swift, a utilizar ownCloud. Con este cambio se consigue que el Servicio de Informes pueda recoger con una única petición la totalidad de los resultados de las simulaciones realizadas en cada trabajo, lo que supone reducir el número de comunicaciones entre Informes y Almacenamiento, con la consecuente reducción del tiempo que se necesita para realizar cada uno de los informes.

Capítulo 4

Evaluación de rendimiento

Finalizado el desarrollo de la nueva versión de la aplicación DNSE3 en la que se van a utilizar contenedores en lugar de máquinas virtuales para llevar a cabo la ejecución de la tarea computacionalmente más costosa y en la que además, se pasa de utilizar Swift a ownCloud para conseguir el almacenamiento persistente de los datos, se realizan una serie de pruebas con el objetivo de evaluar las prestaciones que ofrecen los servicios de Informes y Simulación remodelados. Para evaluar dichas prestaciones se compararán los resultados obtenidos con los reportados en [4].

La nube de cómputo sobre la que se va a realizar las pruebas está compuesta por un procesador de 24 núcleos a 2,2GHz y 256GB de memoria RAM. Los recursos utilizados para la ejecución de estas pruebas serán similares a los que se utilizaron en [4]. Los Servicios de Orquestación y Colas se ejecutan en máquinas virtuales que disponen de 2 CPUs virtuales y 2GB de RAM. El Servicio de Informes se ejecuta sobre una máquina virtual con una vCPU y 2GB de memoria RAM. El Servicio de Simulación de la versión de la aplicación presentada en [4] podía hacer uso de hasta 30 máquinas virtuales que disponían de una vCPU y 2GB de RAM. En la versión basada en contenedores el Servicio de Simulación podrá usar hasta 3 máquinas virtuales de 10 CPUs virtuales y 8GB de memoria RAM, sobre las que se podrá lanzar hasta un máximo de 10 contenedores sobre cada una de ellas. El Servicio de Almacenamiento que utiliza ownCloud a través de contenedores Docker se ejecuta sobre una máquina virtual con 4 CPUs virtuales, 8GB de RAM y 10GB de disco al que se le añadirá un volumen para garantizar la persistencia de los datos.

4.1. Resultados del Servicio de Simulación

4.1.1. Marco experimental

Todas las pruebas realizadas se llevan a cabo en periodos en los que la nube de cómputo se encuentra con un volumen de trabajo bajo. El objetivo de hacerlas de este modo es que estas pruebas de rendimiento, que serán computacionalmente exigentes, no afecten al resto de proyectos que se encuentran alojados en la nube privada. Para la realización de las pruebas se utilizó el *script* `P2-ns-QueueingTheory-m_m_1-m_g_1` [27] puesto que es el mismo *script* que se usó en [4].

En [4] se hicieron pruebas de 50, 100, 500, 1.000, 5.000 y 10.000 simulaciones, por lo que en las pruebas de rendimiento de la nueva versión se realizarán trabajos del mismo volumen de simulaciones. Además, en dicho *paper* se diferenciaban dos tipos de simulaciones: simulaciones

cortas, que para llevarse a cabo se necesitaba alrededor de 1 segundo y simulaciones *largas*, que precisaban de 4 veces más tiempo que las *cortas* para completarse.

Por ello, el primer paso que se tiene que dar es comprobar el tiempo que se necesita para realizar este tipo de simulaciones tanto en una máquina virtual sobre la arquitectura de OpenStack que gestiona los recursos de la nube de cómputo privada sobre la que se va a ejecutar la aplicación DNSE3, como dentro de un contenedor que se ejecuta sobre una máquina virtual gestionada por OpenStack sobre la misma nube de cómputo. Además, se han replicado las mismas pruebas en un ordenador personal (CPU de 6 núcleos a 2,2GHz, 4,1GHz en modo *turbo*, con *Hyper-threading* que le permite ejecutar dos hilos por núcleo; 8GB de memoria RAM; tiene instalado el sistema operativo Ubuntu 19.04) con el objetivo de poner en perspectiva los resultados que se alcanzan en la nube de cómputo privada.

Tipo de simulación	**Lugar de ejecución**	**Tiempo medio (seg.)**	**Varianza**
Simulación *corta*	MV sobre OpenStack	1,3	$2,94 \times 10^{-3}$
	Docker sobre MV sobre OpenStack	1,4	$3,8 \times 10^{-3}$
	Ordenador personal	0,8	$3,83 \times 10^{-4}$
	Docker sobre ordenador personal	1,1	$1,59 \times 10^{-3}$
Simulación *larga*	MV sobre OpenStack	4,9	$1,46 \times 10^{-2}$
	Docker sobre MV sobre OpenStack	5,6	$2,57 \times 10^{-2}$
	Ordenador personal	4	$4,91 \times 10^{-3}$
	Docker sobre ordenador personal	4,5	$1,07 \times 10^{-2}$

Tabla 4.1: Tiempo medio, en segundos, necesario para llevar a cabo simulaciones de tipo *corta* y *larga* sobre distintas configuraciones de *hardware* y *software*.

Como se puede ver en la tabla 4.1, el tiempo necesario para ejecutar las simulaciones *largas* es aproximadamente 4 veces superior al tiempo que se necesita para realizar las *cortas*. Estos media temporal es calculada a partir de 10 experimentos utilizando para ello el comando `time`[1] de GNU/Linux. La relación entre el tiempo necesario para realizar las simulaciones *cortas* y *largas* es la misma que la que se obtenía en [4]. Por este motivo se usarán los resultados reflejados en ese documento para comparar el rendimiento de la nueva versión de la aplicación frente a los que se obtuvieron con la versión antigua. Los tiempos que arroja la ejecución del modelo simulado sobre el ordenador personal son mejores que los que se obtiene en la nube de cómputo privada. El motivo de que suceda esto es que en el ordenador personal se está ahorrando un nivel de virtualización, puesto que en las pruebas de la tabla 4.1 *Ordenador personal* las simulaciones se realizan directamente sobre el sistema operativo instalado y en el segundo tipo de pruebas que se realizan sobre el ordenador personal, los contenedores utilizan el *kernel* del sistema operativo instalado en la máquina anfitriona.

Para comprobar el correcto funcionamiento del Servicio de Simulación empaquetado en contenedores Docker y la cantidad de recursos que consumen estos contenedores se

[1] https://www.gnu.org/software/time/

han realizado una serie de pruebas. Estas pruebas consistían en la ejecución del *script* `P2-ns-QueueingTheory-m_m_1-m_g_1` [27], sobre máquinas virtuales con diferente configuración de *hardware*. Las configuraciones de *hardware* utilizadas fueron las configuraciones predefinidas en la plataforma OpenStack [28]: `m1.small`, 1 vCPU, 2GB de memoria RAM; `m1.medium`, 2 vCPU, 4GB de RAM; `m1.large`, 4 vCPU, 8GB de memoria RAM; `m1.xlarge`, 8 vCPU, 16GB de RAM. Los resultados obtenidos en las distintas pruebas se pueden observar en la Fig. 4.1. Con estas pruebas se comprobó que el número óptimo de contenedores activos con el Servicio de Simulación sobre un *host* estará directamente relacionado con el número de núcleos que disponga el procesador del *host*. Este resultado será utilizado para determinar el número máximo de contenedores por *host* en la versión de la aplicación DNSE3 resultante de este Trabajo Fin de Grado. En la misma figura se puede observar que para el caso de la configuración `m1.xlarge` el tiempo de realización de las simulaciones se reduce ligeramente cuando el número de contenedores sobrepasa al número de núcleos virtuales. En la versión final de la aplicación se aprovecha esta ligera mejoría aumentando el número de simulaciones que cada Servicio de Simulación puede ejecutar en paralelo. La ganancia que se obtiene al realizar más de una simulación al mismo tiempo en cada Servicio de Simulación se verá en 4.1.3.

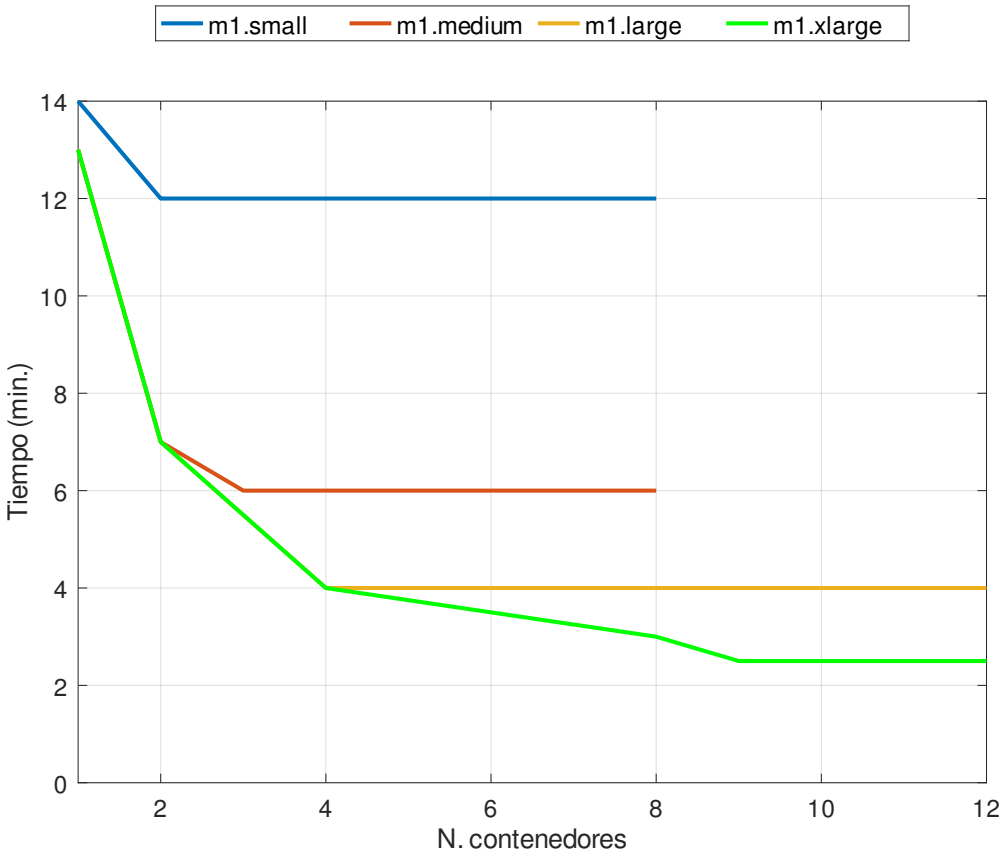


Fig. 4.1: Gráfica que representa el tiempo que fue necesario para realizar un trabajo de 500 simulaciones utilizando diferente número de contenedores sobre diferentes configuraciones de *hardware* virtual.

En lo que resta de sección, las pruebas realizadas para comprobar el tiempo que se necesita para completar un número determinado de simulaciones se realizan con el Servicio de Monitorización y Escalado activo. Gracias a los registros que genera este servicio se obtiene tanto el número de simulaciones por completarse (dato que es publicado por el Servicio de

Colas cada dos segundos), como el número de contenedores activos y el número de máquinas virtuales activas. Por como se implementó la espera exponencial del Servicio de Monitorización y Escalado, la evaluación del número de contenedores y máquinas virtuales se realiza como máximo cada 16 segundos.

4.1.2. Velocidad a la que se escalan los contenedores

En 3.2.2 se introdujo que el tiempo de adaptación de los recursos al número de trabajos pendientes de ejecución se realiza en función del valor del umbral T. El valor de T en la versión actual del Servicio de Monitorización y Escalado es un valor constante, no pudiéndose cambiar mediante ningún parámetro al iniciar el servicio. Para determinar cómo afecta el valor de esta constante a los tiempos de finalización de las simulaciones se realizan las mismas pruebas para los valores $T = 10$ y $T = 100$. En estas pruebas se configura al Servicio de Simulación ejecutado sobre contenedores para que su número máximo de tareas a realizar en paralelo sea de una simulación. Además, se usa ownCloud como Servicio de Almacenamiento.

En la tabla 4.2 se puede ver el tiempo medio, en segundos, que se ha necesitado para completar un número variable de simulaciones. En la misma tabla se puede observar que la versión con $T = 10$ ofrece mejores resultados que la que utiliza $T = 100$. Es debido a que con un valor bajo de T se consumen más rápido los recursos disponibles y se liberan más tarde. En la tabla 4.3 se puede ver el número de contenedores y máquinas virtuales que como máximo se utiliza en cada una de las pruebas realizadas. Se observa que la versión con $T = 100$ realiza un uso menos agresivo de los recursos disponibles. Además, con $T = 10$ en los casos de 1.000 simulaciones *cortas* y 500 simulaciones *largas* se observa que, aunque en el momento de iniciar una nueva máquina virtual se encontraron simulaciones pendientes para activar la mitad de los contenedores que puede soportar cada máquina virtual, una vez que se consigue activar la nueva máquina virtual no se llega a activar esa cantidad de contenedores. Esto es debido a que no se tiene en cuenta la velocidad a la que se están completando las simulaciones, solamente el número de simulaciones pendientes de realizar.

		50	**100**	**500**	**1.000**	**5.000**	**10.000**
Simulaciones *cortas*	$T = 10$	48	57	179	255	871	1.605
	$T = 100$	110	214	453	603	1.291	2.075
	$\frac{T=10}{T=100}$	0,436	0,266	0,395	0,423	0,675	0,774
Simulaciones *largas*	$T = 10$	134	174	398	585	1.861	3.612
	$T = 100$	315	618	1.409	1.839	3.371	5.016
	$\frac{T=10}{T=100}$	0,425	0,282	0,282	0,318	0,404	0,72

Tabla 4.2: Tiempo medio (segundos) que necesita la aplicación DNSE3 para realizar un número variable de simulaciones con valores diferentes del umbral T. En la tercera fila de cada tipo de simulación se expresa la ratio de $\frac{T=10}{T=100}$.

Con estas pruebas se toma la decisión de utilizar el valor del umbral $T = 10$. Esta decisión se toma ya que el principal objetivo de la aplicación DNSE3 es proporcionar a los usuarios de la aplicación los resultados de las simulaciones en el menor tiempo posible, utilizando de forma eficiente los recursos disponibles en la nube de cómputo. El umbral $T = 10$ cumple con estas dos condiciones, puesto que con este valor se consiguen mejores resultados que con $T = 100$ y también, se hace un uso responsable de los recursos disponibles en la nube de cómputo privada, puesto que para activar una nueva máquina virtual se necesita que haya una cantidad razonable de trabajos esperando a ser realizados. Además, una vez que se ha inicializado una nueva máquina virtual, esta no se destruye hasta que el número de trabajos

		50		**100**		**500**		**1.000**		**5.000**		**10.000**	
		D.	MV	D.	MV	D.	MV	D.	MV	D.	MV	D.	MV
Simulac. *cortas*	$T = 10$	5	1	8	1	17	2	23	3	30	3	30	3
	$T = 100$	1	1	2	1	5	1	10	1	30	3	30	3
Simulac. *largas*	$T = 10$	5	1	8	1	21	3	30	3	30	3	30	3
	$T = 100$	1	1	2	1	5	1	10	1	30	3	30	3

Tabla 4.3: Número máximo de contenedores (D.) y máquinas virtuales (MV) que se han llegado a utilizar para comprobar cómo afecta el valor del umbral T al rendimiento de la aplicación.

en el Servicio de Colas no sea muy bajo, por lo que DNSE3 se aprovecha de los recursos que reserva durante más tiempo. Esto es muy relevante, ya que para **iniciar una nueva máquina virtual** se necesitan **63 segundos de media**, por lo que una vez que se ha iniciado una nueva máquina virtual es conveniente mantenerla iniciada el mayor tiempo posible. Para completar con el análisis del tiempo que necesita una máquina virtual para cambiar de estado, hay que añadir que el tiempo medio que necesita el sistema para **destruir una máquina virtual** es de **14 segundos**. En la Fig. 4.2 se puede observar la evolución que sufre el número de trabajos pendientes de realizar frente a la evolución del número de contenedores y máquinas virtuales activas en el sistema. La transición que más tiempo necesita es la creación de las máquinas virtuales. Se puede ver gráficamente dicha transición en 4.2 ya que son los dos intervalos temporales en los que hay suficientes simulaciones esperando a ser realizadas pero el número de contenedores activos se mantiene constante (aproximadamente entre los instantes temporales 60 - 120 segundos y 190 - 250 segundos).

4.1.3. Número máximo de tareas por contenedor

Al iniciar el Servicio de Simulación, bien sea ejecutado directamente sobre un *host* o sobre un contenedor, se le tienen que pasar dos parámetros. En primer lugar, se le pasa como argumento la dirección IP y puerto que utiliza el Servicio de Colas para que pueda establecer comunicación con él. En último lugar, se le indica el número máximo de simulaciones que puede realizar el Servicio de Simulación en paralelo. En la tabla 4.4 se pueden ver los resultados que se obtiene cuando se modifica este valor y se realizan simulaciones de la misma longitud y número de repeticiones. Para realizar estas pruebas se ha fijado el valor del umbral $T = 10$.

		50	**100**	**500**	**1.000**	**5.000**	**10.000**
Simulaciones *cortas*	$sim = 4$	21	30	112	204	841	1.493
	$sim = 1$	48	57	179	255	871	1.605
	$\frac{sim=4}{sim=1}$	0,44	0,53	0,63	0,8	0,97	0,93
Simulaciones *largas*	$sim = 4$	52	124	314	490	1.794	3.453
	$sim = 1$	134	174	398	585	1.861	3.612
	$\frac{sim=4}{sim=1}$	0,39	0,71	0,79	0,84	0,96	0,96

Tabla 4.4: Comparativa del tiempo medio (segundos) que necesita la nueva versión del Servicio de Simulación de la aplicación DNSE3 cuando se le pasa como argumento que pueda realizar hasta un máximo de 4 simulaciones en paralelo ($sim = 4$) y cuando se le indica que solamente puede realizar una simulación cada vez ($sim = 1$).

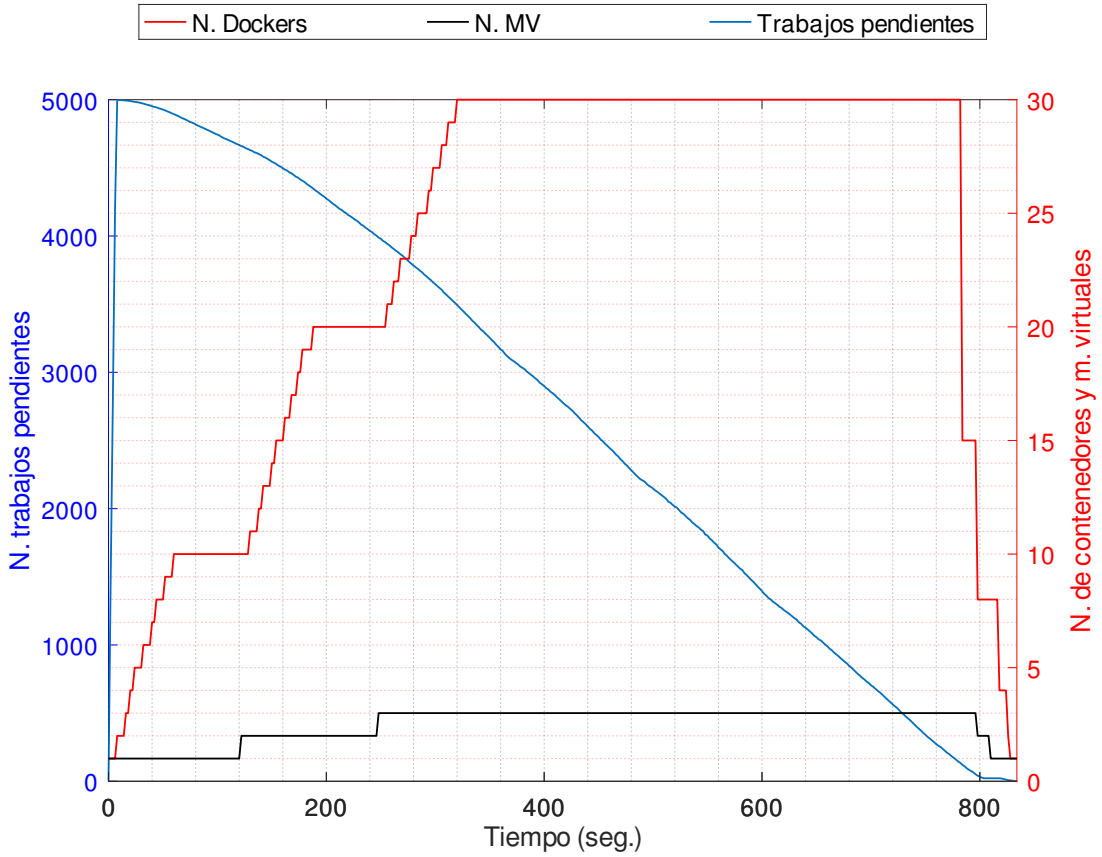


Fig. 4.2: Evolución del número de trabajos que están esperando a ser realizados en el Servicio de Colas (en azul) frente al número de contenedores (*N. Dockers*, en rojo) y máquinas virtuales (*N. MV*, en negro). El trabajo que se representa está compuesto por 5.000 simulaciones *cortas* y se realiza con el valor del umbral $T = 10$.

En los resultados obtenidos en la tabla 4.4 se observa que los tiempos que se obtienen cuando se fija a cuatro el número máximo de tareas que cada Servicio de Simulación puede realizar en paralelo son mejores que cuando se fija dicho valor a uno. Esto es debido a que cuando se realizan varias simulaciones en paralelo por cada contenedor, se consigue aprovechar mejor los tiempos de E/S puesto que cuando un proceso finaliza su ejecución otro entra en su lugar. Cuando se utiliza la opción de $sim = 1$ el núcleo virtual está más tiempo *desaprovechado* ya que cuando finaliza la ejecución de una simulación tiene que *esperar* al almacenamiento de los datos y la obtención del nuevo ejecutable para llevar a cabo la siguiente simulación.

Además, se observa que la ganancia al utilizar cuatro simulaciones por contenedor se va reduciendo a medida que el número de repeticiones que compone cada experimento es mayor. Una posible explicación de la obtención de estos resultados se explican por el ritmo al que se consigue completar las simulaciones. Tomando los experimentos de 10.000 simulaciones *cortas* y fijándose en los periodos en los que el número de contenedores se mantiene estable (incremento del número de máquinas virtuales y cuando se ha alcanzado el número máximo de máquinas virtuales en el clúster) se observa que el ritmo al que el Servicio de Simulación completa simulaciones de este tipo es de aproximadamente 5 simulaciones por segundo con 10 contenedores, 7 simulaciones por segundo para 20 contenedores y 8 simulaciones por segundo cuando están activos los 30 contenedores del clúster para el caso de $sim = 4$. Cuando se

utiliza $sim = 1$ el ritmo con el que se completan las simulaciones es de aproximadamente 4 simulaciones por segundo, 6 simulaciones por segundo y 7 simulaciones por segundo respectivamente. Esta pérdida de prestaciones al incrementar el número de contenedores puede ser debida a la presión que son sometidos el resto de los servicios que componen la aplicación. Se comprueba que el Servicio de Almacenamiento basado en ownCloud cuando tiene que resolver un número elevado de peticiones puede llegar a necesitar 5 segundos desde el momento que se inicia el guardado de los resultados de una simulación hasta que finalmente se consiguen almacenar.

4.1.4. Comparativa de resultados entre versiones

La versión anterior de DNSE3 se basaba en una arquitectura de máquinas virtuales para conseguir distribuir las réplicas del Servicio de Simulación con el objetivo de obtener los resultados de los distintos trabajos en un tiempo razonable. En [4] se expusieron los resultados obtenidos con el escalado del número de máquinas controlado por distintas políticas. Entre estas políticas destaca P_{cons} que es una política similar a la que se utiliza en la versión actual de la aplicación. En esa política el valor del umbral T rige la velocidad a la que se activan o desactivan las máquinas virtuales que ejecutan el Servicio de Simulación. Por la similitud entre las políticas utilizadas se realiza una comparativa entre las distintas versiones, cuyos resultados se pueden ver en las tablas 4.5 y 4.6. Las pruebas realizadas a la nueva versión de la aplicación se llevan a cabo tanto con el Servicio de Almacenamiento basado en Swift como con el que está basado en ownCloud, con el valor del umbral $T = 10$ y 4 simulaciones ejecutándose en paralelo por contenedor.

		50	**100**	**500**	**1.000**	**5.000**	**10.000**
Simulaciones *cortas*	$DNSE3_{Swift}$	20	33	122	182	520	914
	$DNSE3_{old}$	50	92	271	375	802	1.165
	$\frac{DNSE3_{Swift}}{DNSE3_{old}}$	0,4	0,36	0,45	0,49	0,65	0,79
Simulaciones *largas*	$DNSE3_{Swift}$	60	160	286	476	1.583	3.067
	$DNSE3_{old}$	83	157	364	505	1.060	1.522
	$\frac{DNSE3_{Swift}}{DNSE3_{old}}$	0,72	1,02	0,79	0,95	1,49	2,02

Tabla 4.5: Comparativa del tiempo medio (segundos) que necesita la nueva versión de DNSE3 cuando utiliza Swift como Servicio de Almacenamiento ($DNSE3_{Swift}$) distribuyendo las simulaciones en una arquitectura mixta de máquinas virtuales y contenedores y la antigua versión ($DNSE3_{old}$) que utiliza una arquitectura basada únicamente en máquinas virtuales. Los resultados de$DNSE3_{old}$ se han extraído de [4].

Los resultados obtenidos en las distintas pruebas indican que el Servicio de Simulación de la aplicación DNSE3 basado en una arquitectura mixta de contenedores y máquinas virtuales obtiene mejores resultados que la versión del mismo servicio que utiliza una arquitectura basada únicamente en máquinas virtuales cuando el número de simulaciones es bajo o medio. Se obtienen peores resultados con la nueva versión del Servicio de Simulación cuando el número de simulaciones a realizar es elevado. Se llega a estos resultados tanto para simulaciones de tipo *corto* como de tipo *largo*. Uno de los motivos de que se produzca este resultado es la pérdida de rendimiento al introducir un nuevo nivel de virtualización con los contenedores. En trabajos de un número bajo o medio de simulaciones, como los contenedores se activan más rápido que las máquinas virtuales el efecto de la perdida de rendimiento *pesa* menos que tener un número mayor de réplicas del Servicio de Simulación ejecutando simulaciones. En

		50	**100**	**500**	**1.000**	**5.000**	**10.000**
Simulaciones *cortas*	$DNSE3_{own}$	21	30	112	204	841	1.493
	$DNSE3_{old}$	50	92	271	375	802	1.165
	$\frac{DNSE3_{own}}{DNSE3_{old}}$	0,42	0,33	0,41	0,68	1,05	1,28
Simulaciones *largas*	$DNSE3_{own}$	52	124	314	490	1.794	3.453
	$DNSE3_{old}$	83	157	364	505	1.060	1.522
	$\frac{DNSE3_{own}}{DNSE3_{old}}$	0,63	0,80	0,86	0,97	1,69	2,27

Tabla 4.6: Comparativa del tiempo medio (segundos) que necesita la nueva versión de DNSE3 cuando utiliza ownCloud como Servicio de Almacenamiento ($DNSE3_{own}$) que distribuye las simulaciones en una arquitectura mixta de máquinas virtuales y contenedores y la antigua versión ($DNSE3_{old}$) que utiliza una arquitectura basada únicamente en máquinas virtuales. Los resultados de$DNSE3_{old}$ se han extraído de [4].

trabajos de muchas simulaciones, aunque se llegue más rápido al número máximo de contenedores activos en el clúster, la perdida por la doble virtualización es más relevante puesto que se está realizando una tarea computacionalmente muy exigente. Si se utilizase un sistema de contenedores nativo los resultados que se obtuviesen serían probablemente mejores a los que han obtenido con en las pruebas realizadas ya que, en primer lugar, no existiría la doble virtualización de máquinas virtuales y contenedores y, en segundo lugar, no se tendría que crear ninguna máquina virtual con lo que se ahorraría, de media, 120 segundos en los casos en los que se haya tenido que iniciar dos máquinas virtuales.

Otro de los aspectos relevantes que se extraen de estas pruebas es la pérdida de rendimiento que sufre el Servicio de Simulación cuando se utiliza ownCloud para el Servicio de Almacenamiento frente a las prestaciones que se obtiene cuando se usa Swift para el mismo propósito. En la tabla 4.7 se puede ver la comparativa de los resultados obtenidos cuando el Servicio de Simulación se le ha indicado que puede realizar un máximo de cuatro simulaciones en paralelo y el umbral T del Servicio de Monitorización y Escalado se fija a 10. En los resultados expuestos en dicha tabla se observa que para trabajos de un número bajo de repeticiones el tiempo necesario para completarlos con ambas versiones del Servicio de Almacenamiento es muy parecido. Si el número de simulaciones por trabajo es alto, la versión del Servicio de Almacenamiento basado en Swift consigue mejores resultados. Por lo visto durante el desarrollo de las pruebas realizadas se puede decir que Swift se está comportando de manera más estable que ownCloud.

		50	**100**	**500**	**1.000**	**5.000**	**10.000**
Simulaciones *cortas*	$DNSE3_{Swift}$	20	33	122	182	520	914
	$DNSE3_{own}$	21	30	112	204	841	1.493
	$\frac{DNSE3_{Swift}}{DNSE3_{own}}$	0,95	1,1	1,09	0,89	0,62	0,61
Simulaciones *largas*	$DNSE3_{Swift}$	60	160	286	476	1.583	3.067
	$DNSE3_{own}$	52	124	314	490	1.794	3.453
	$\frac{DNSE3_{Swift}}{DNSE3_{own}}$	1,15	1,29	0,91	0,97	0,88	0,89

Tabla 4.7: Comparativa del tiempo medio (segundos) que necesita la nueva versión de DNSE3 cuando utiliza Swift ($DNSE3_{Swift}$) o ownCloud ($DNSE3_{own}$) para el Servicio de Almacenamiento.

Para ver de forma gráfica la consistencia de los resultados obtenidos en las distintas pruebas realizadas para obtener los tiempos medios reflejados en las tablas de esta subsección, se van a representar dichos tiempos en figuras de tipo `boxplot` que permiten observar la simetría

que presenta cada grupo de pruebas y la posible existencia de valores atípicos (*outliers*). En el sistema de `boxplot` se representa mediante un rectángulo los percentiles primero, segundo (mediana) y tercero, siendo el segundo percentil representado con una línea horizontal roja. Además se representa con forma de T los valores máximo y mínimo de la serie representada cuando estos valores no estén a más de 1,5 veces la distancia entre los percentiles primero y tercero (distancia denominada IQR, *Interquartile Range*). Cuando algún valor de la serie tiene un valor mayor (o inferior) a $1,5 \times IQR$ se denomina *outlier* y se representa con el símbolo +. Si supera 3 veces el valor de IQR se representa con el símbolo *o* [29]. Estas representaciones se pueden ver en las figuras 4.3 y 4.4. En estas figuras se observan varios *outliers* que pueden estar causados por tres causas diferentes: el primer motivo es que se puede tardar en detectar un incremento en el número de simulaciones pendientes de realizar hasta un máximo de 16 segundos debido a cómo se programó la espera exponencial en el Servicio de Monitorización y Escalado; el segundo posible motivo de la aparición de estos *outliers* es que las máquinas virtuales han podido necesitar más tiempo del habitual para iniciarse; el último posible motivo es que se ha destruido un contenedor por parte del Servicio de Monitorización y Escalado, el cual tenía asignado una simulación en el momento de la destrucción, por lo que hasta que el Servicio de Colas no detecte que no se va a poder completar ese trabajo no se permite a otra réplica del Servicio de Simulación ejecutar dicha simulación. Esta última circunstancia se suele dar cuando quedan muy pocos trabajos por realizar, por lo que se destruye gran parte de los contenedores activos del clúster.

4.1.5. Uso de una nube de cómputo pública para la ejecución del servicio

La tecnología empleada para empaquetar el *software* del Servicio de Simulación y todas sus dependencias son los contenedores de tipo Docker. Estos contenedores se han convertido en el estándar de la industria, por lo que existen múltiples alternativas en las nubes de cómputo públicas para su ejecución. En esta subsección se va centrar en las tecnologías **AWS Fargate**[2] y **ACI**[3] (*Azure Container Instances*) que permiten el despliegue de servicios basados en contenedores sin la necesidad de añadir un motor de orquestación. Estas dos alternativas liberan al desarrollador de tener que planificar la infraestructura bajo la que se van a ejecutar los contenedores y permiten adecuar el número de contenedores activos en el servicio a través de la definición de una serie de reglas. Estos servicios cobran por la cantidad de recursos consumidos de la siguiente forma:

- AWS Fargate factura por segundos la cantidad de CPUs virtuales y memoria RAM que utiliza el servicio desde el momento en el que se inicia la descarga de la imagen base de contenedor. Si el servicio finaliza su tarea antes del primer minuto se paga ese minuto entero. Impone el requisito de que si se desea escoger una configuración de una vCPU (o una cantidad superior de CPUs) por contenedor obliga a contratar 2GB de memoria RAM para cada uno de los contenedores. A mayores de la cantidad de recursos *hardware* virtual utilizado hay que añadir el tráfico que sale de los contenedores hacia Internet.
  - Coste vCPU por hora: 0,04048 USD.
  - Coste de GB de memoria RAM por hora: 0,004445 USD.
  - Coste por GB de tráfico saliente desde los contenedores hacia Internet: 0,09 USD si no se sobrepasa la cantidad de 9,999TB al mes. Si no se alcanza el GB de tráfico saliente hacia Internet no se aplica ningún coste.

[2] https://aws.amazon.com/es/fargate/
[3] https://azure.microsoft.com/es-es/services/container-instances/

- En ACI se factura por segundos los recursos de vCPU y memoria RAM consumidos por los contenedores. No tiene ninguna limitación a la hora de escoger configuraciones de *hardware* virtual. Tampoco se cobra el primer minuto entero si el servicio finaliza su función en memos tiempo, en ese caso solamente se pagaría por los segundos que hubiera estado activo.
    - Coste vCPU por hora: 0,0486 USD.
    - Coste de GB de memoria RAM por hora: 0,0054 USD.
    - Coste por GB de tráfico saliente desde los contenedores hacia Internet: 0,087 USD si no se sobrepasa la cantidad de 10TB al mes. Si no se alcanzan los 5GB de tráfico saliente a Internet no se aplica ningún coste.

Los contenedores que ejecutan el Servicio de Simulación de la aplicación DNSE3 tienen que disponer de una vCPU, ya que realizan una tarea computacionalmente exigente. Además, se establece un mínimo de 2GB de memoria RAM para que este servicio pueda soportar cualquier tipo de simulaciones, dentro de lo razonable. Con estos requisitos de *hardware* virtual se realiza la estimación del coste medio que hubiera supuesto realizar las simulaciones llevadas a cabo con ownCloud funcionando como Servicio de Almacenamiento. Los resultados de esta estimación se pueden observar en la tabla 4.8. Para la realización de estos cálculos se tiene en cuenta el número máximo de contenedores que llegaron a activarse en cada prueba. No se tiene en consideración el coste que supone el tráfico desde los contenedores a Internet.

AWS Fargate	**50**	**100**	**500**	**1.000**	**5.000**	**10.000**
Simulaciones *cortas*	0,003	0,005	0,015	0,05	0,346	0,614
Simulaciones *largas*	0,004	0,014	0,086	0,202	0,738	1,421
ACI	**50**	**100**	**500**	**1.000**	**5.000**	**10.000**
Simulaciones *cortas*	0,001	0,003	0,018	0,061	0,416	0,739
Simulaciones *largas*	0,004	0,016	0,104	0,243	0,888	1,709

Tabla 4.8: Coste medio individual (en dólares estadounidenses) que supondría haber realizado las pruebas de rendimiento de la nueva versión de DNSE3 sobre distintas nubes de cómputo públicas. Para estas pruebas se fijó el valor del umbral $T = 10$, se indicó al Servicio de Simulación que podía realizar 4 simulaciones en paralelo y se utilizó ownCloud para el Servicio de Almacenamiento de DNSE3.

4.2. Resultados del Servicio de Informes

El Servicio de Informes de la aplicación DNSE3 se encarga de formatear el resultado de todos las simulaciones que formen un trabajo, con el objetivo de que a los alumnos que utilicen la aplicación les resulte sencillo analizar dichos datos con una herramienta externa como GNU Octave. El Servicio de Informes se sigue ejecutando sobre una máquina virtual como se ejecutaba en la versión anterior de la aplicación. Su funcionamiento es muy similar al que realizaba en la versión anterior, pudiéndose paralelizar la realización de los informes.

Las modificaciones que sufre este servicio vienen motivadas por el cambio realizado en el Servicio de Almacenamiento. Utilizando ownCloud se pude solicitar la descarga de un directorio completo, obteniéndose en la respuesta dicho directorio empaquetado en formato ZIP. Con ello se consigue reducir de forma significativa el número de comunicaciones que se tiene que establecer entre Informes y Almacenamiento cuando el trabajo se compone de un número elevado de simulaciones. En la tabla 4.9 se puede observar el tiempo medio que se

necesita para completar cada informe compuesto por un número fijo de simulaciones. Estos tiempos se obtienen directamente del Servicio de Informes puesto que se ha modificado dicho servicio para que indique cuanto tiempo necesita para completar cada uno de los informes que realice.

	50		**100**		**500**	
	Media	Varianza	Media	Varianza	Media	Varianza
$Report_{new}$	3	$6,86 \times 10^{-2}$	3	$5,56 \times 10^{-2}$	10	$6,47 \times 10^{-1}$
$Report_{old}$	6	9,24	5	$3,34 \times 10^{-2}$	66	$2,13 \times 10^{3}$
$\frac{Report_{new}}{Report_{old}}$	0,5		0,6		0,15	
	1.000		**5.000**		**10.000**	
	Media	Varianza	Media	Varianza	Media	Varianza
$Report_{new}$	18	2,7	82	$1,64 \times 10$	169	$7,37 \times 10$
$Report_{old}$	70	$2,1 \times 10^{2}$	388	$3{,}01 \times 10^{3}$	873	$6,38 \times 10$
$\frac{Report_{new}}{Report_{old}}$	0,26		0,21		0,19	

Tabla 4.9: Tiempo medio (en segundos) necesario para realizar un informe formado por un número de simulaciones. Se compara la versión del Servicio de Informes de DNSE3 que utiliza ownCloud como Servicio de Almacenamiento ($Report_{new}$) frente a la versión que utiliza Swift ($Report_{old}$).

Los resultados que arroja la tabla 4.9 indican que el Servicio de Informes de la nueva versión de DNSE3 finaliza todo tipo de trabajos en menor tiempo que el que necesitaba la versión anterior de la aplicación. Esto es debido a la reducción del número de comunicaciones que se tiene que establecer entre Informes y Almacenamiento para conseguir tanto los resultados de las simulaciones como los parámetros con los que se han conseguido dichos resultados. Una vez obtenido el ZIP se descomprime para poder acceder a todos los ficheros de resultados. En la tabla 4.9 se observa que el nuevo Servicio de Informes siempre ofrece mejores prestaciones que el antiguo y que además, esta mejora de prestaciones se acentúan cuando el número de simulaciones que forman el trabajo es elevado.

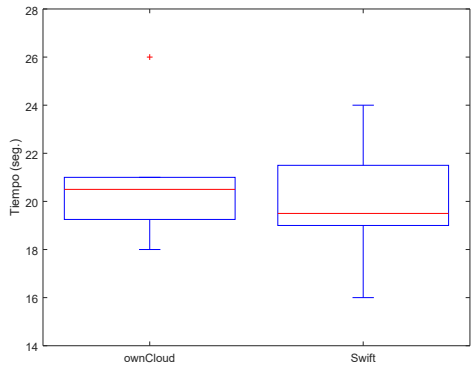
4.3. Conclusiones

Los cambios introducidos en el Servicio de Simulación para que pase a ejecutarse sobre contenedores, en vez de sobre máquinas virtuales, permiten que el número de réplicas que ejecutan este servicio se adapte de forma más rápida al número de trabajos esperando a ser realizados. Una vez completados estos cambios, si el número de simulaciones a ejecutar es bajo o medio se consigue mejorar las prestaciones que ofrecía la versión DNSE3 presentada en [4]. Si el número de trabajos a realizar es elevado, la nueva versión ofrece peor rendimiento que la versión anterior. Esta pérdida de rendimiento se produce, en parte, por la doble virtualización de ejecutar contenedores sobre máquinas virtuales. Además, tras las pruebas realizadas con Swift y ownCloud para el Servicio de Almacenamiento se llega a la conclusión de que ownCloud ofrece peores resultados que Swift cuando el número de simulaciones a realizar es elevado.

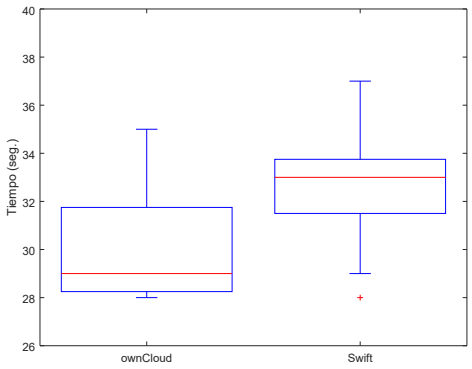
Aun así, con todos los experimentos realizados se comprueba que la aplicación DNSE3 con el Servicio de Simulación distribuido sobre Dockers es una versión funcional. Esta versión consigue adaptar la cantidad de recursos utilizados por el Servicio de Simulación a la demanda actual del sistema. Además, una vez que las simulaciones han finalizado, se ha comprobado que para el Servicio de Informes el cambio a favor de ownCloud en el Servicio de Almacenamiento

provoca una mejoría considerable, llegando a necesitar una quinta parte del tiempo que necesitaba la versión anterior de la aplicación para la realización de los informes compuestos por un número elevado de resultados.

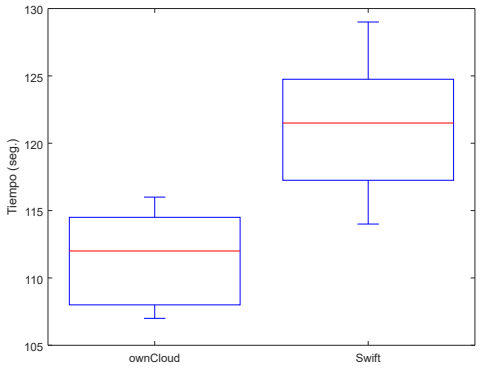
Combinando los resultados obtenidos en los Servicios de Simulación e Informes se llega a la conclusión de que, salvo para el caso de un elevado número de simulaciones *largas*, los alumnos obtendrán los resultados de las simulaciones en un tiempo menor que el tenían que esperar para conseguirlos con la versión anterior de DNSE3.



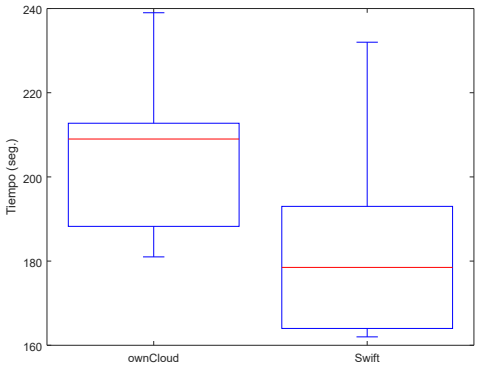
(a) Pruebas de 50 repeticiones



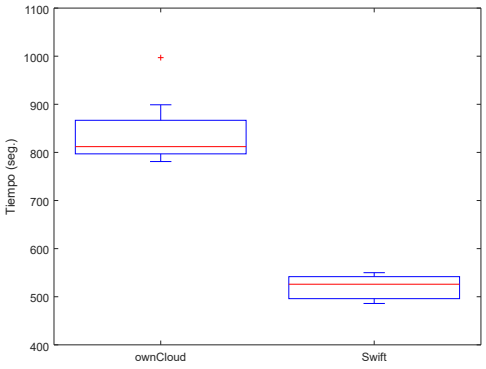
(b) Pruebas de 100 repeticiones



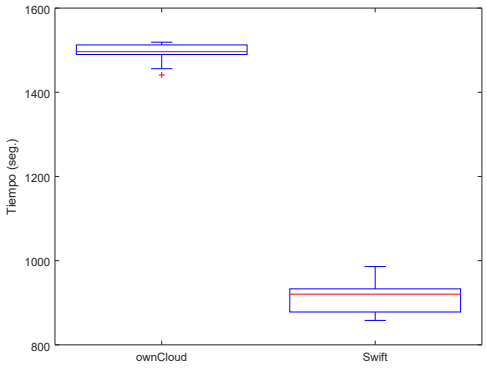
(c) Pruebas de 500 repeticiones



(d) Pruebas de 1.000 repeticiones

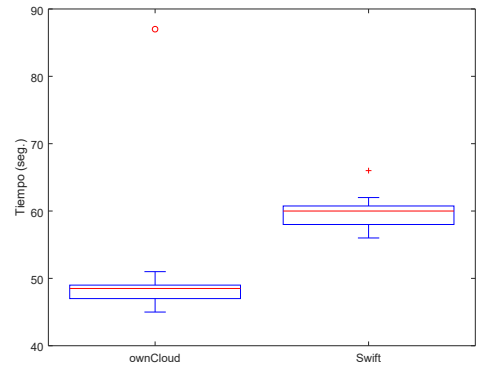


(e) Pruebas de 5.000 repeticiones

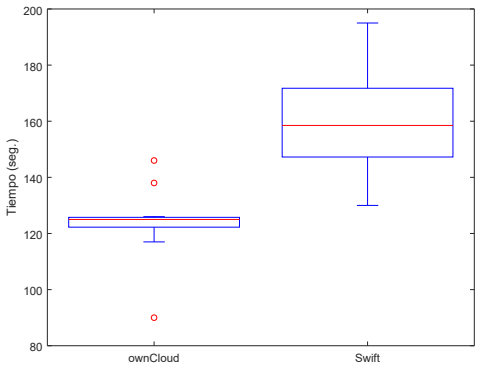


(f) Pruebas de 10.000 repeticiones

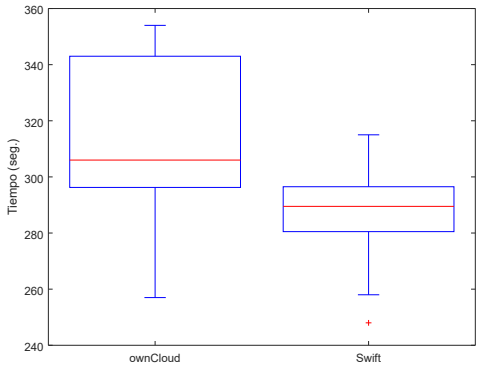
Fig. 4.3: Representación de la consistencia que ha ofrecido el sistema durante las distintas pruebas de simulaciones *cortas*. Se muestran los resultados tanto de ownCloud como de Swift cuando se usa cada uno de estos proyectos en el Servicio de Almacenamiento.



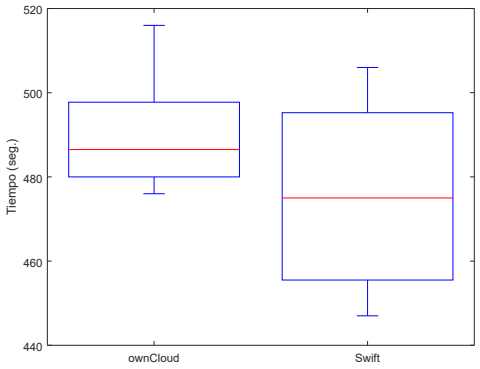
(a) Pruebas de 50 repeticiones



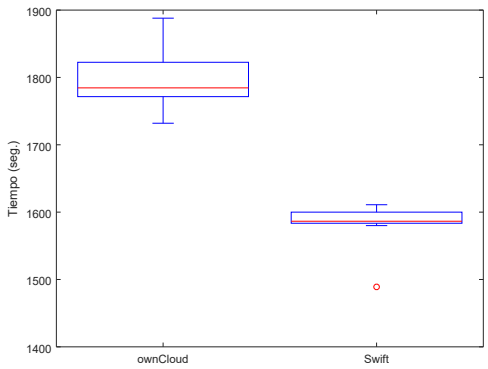
(b) Pruebas de 100 repeticiones



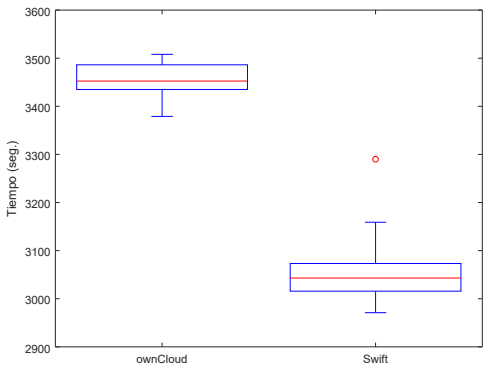
(c) Pruebas de 500 repeticiones



(d) Pruebas de 1.000 repeticiones



(e) Pruebas de 5.000 repeticiones



(f) Pruebas de 10.000 repeticiones

Fig. 4.4: Representación de la consistencia que ha ofrecido el sistema durante las distintas pruebas de simulaciones *largas*. Se muestran los resultados tanto de ownCloud como de Swift cuando se usa cada uno de estos proyectos en el Servicio de Almacenamiento.

Capítulo 5

Conclusiones y líneas de trabajo futuro

5.1. Recordatorio del objetivo del trabajo

El uso de simuladores de redes en ambientes educativos permite a los alumnos experimentar con distintas configuraciones de red para entender cómo afecta la variación de alguno de los parámetros que intervengan en la comunicación al rendimiento de la red. Para realizar este análisis, los alumnos *barrerán* el parámetro a estudiar para comprobar cómo afecta su valor a la red simulada. El principal problema que presentan las simulaciones es el tiempo que precisan para completarse. Para reducir dicho tiempo se desarrolló la aplicación DNSE3 que hace uso de la nube computacional para paralelizar la ejecución de las simulaciones.

En [4] se presentó una versión de DNSE3 completamente funcional. Esta versión utilizaba una combinación de servicios propios y proyectos de la plataforma gestora de los recursos de la nube de cómputo sobre la que se ejecuta. Con esta combinación de servicios propios y genéricos se conseguía adaptar el número de réplicas del servicio que realizaba las simulaciones a la demanda existente de simulaciones en el sistema. Estas réplicas se ejecutan sobre máquinas virtuales y se iniciaban y se destruían a medida que se necesitaban o dejaban de necesitarse con el objetivo consumir de forma responsable los recursos disponibles en la nube de cómputo.

La creación y destrucción de las máquinas virtuales donde se ejecutan las simulaciones trae asociado un retardo que afecta de forma negativa al rendimiento global de la aplicación. Por este motivo se decide utilizar la tecnología de contenedores, tecnología que permite empaquetar un programa con sus dependencias. Los contenedores se inician de manera más rápida que las máquinas virtuales puesto que aprovechan el *kernel* del sistema operativo donde se ejecutan, por lo que no es necesario cargar un sistema operativo completo.

5.2. Conclusiones del trabajo realizado

Con el objetivo de aprovechar la mayor velocidad de escalado de los contenedores, se adapta el servicio encargado de completar las simulaciones que solicitan los usuarios de la aplicación a esta tecnología. Se utiliza para ello contenedores de tipo Docker, que es el tipo de contenedor que se ha convertido en el estándar *de facto* de la industria. Su ejecución se realiza sobre un clúster formado por máquinas virtuales. El número de máquinas virtuales activas en el clúster es controlado por un servicio propio de la aplicación del mismo modo que este

servicio se encarga de controlar el número de contenedores. El motivo por el que se ejecutan los contenedores sobre un clúster de máquinas virtuales es que el proyecto disponible para la gestión de contenedores sobre OpenStack, que es la plataforma encargada de gestionar los recursos de la nube privada donde se ejecuta DNSE3, puede crear incompatibilidades con los proyectos que están actualmente ejecutándose en dicha nube. Esta incompatibilidad estaba fundamentada en que, en la versión actualmente instalada de OpenStack y con los proyectos destinados a la gestión de contenedores, no se permite desplegar conjuntamente proyectos basados exclusivamente en contenedores (sin máquinas virtuales) y otros basados únicamente en máquinas virtuales.

Con la imagen de contenedor del servicio encargado de ejecutar las simulaciones creada, se realiza una serie de pruebas en las que se simula el mismo modelo sobre una máquina virtual lanzada sobre la nube privada y sobre un contenedor, que a su vez se despliega sobre una máquina virtual creada en la misma nube. Con estas pruebas se detecta que se produce una pérdida de rendimiento al llevar a cabo la simulación dentro del contenedor. Esta pérdida es de aproximadamente el 7 % en simulaciones *cortas* y del 12 % en simulaciones *largas*. Por ello, cuando el número de simulaciones es elevado, la versión de DNSE3 que se basa en máquinas virtuales obtiene mejores resultados que la que usa contenedores. Si el número de simulaciones a realizar es bajo o medio, la pérdida de rendimiento al ejecutar las simulaciones en contenedores tiene menos *peso* que el que tiene el tiempo que necesita una máquina virtual en activarse, por lo que se obtienen los resultados de los trabajos en un tiempo menor que el que necesitaba la versión anterior de DNSE3. Esto puede ser debido a las pérdidas de rendimiento sufridas por la doble virtualización de máquinas virtuales y contenedores. Si se hiciera un despliegue *nativo* del Servicio de Simulación sobre contenedores se conseguirían mejorar los tiempos que se han obtenido, ya que por un lado se eliminaría el problema de la doble virtualización y por otro, no se necesitaría esperar a la creación de una nueva máquina virtual para aumentar el número de contenedores activos que están ejecutando las simulaciones.

Otro punto de interés de los cambios realizados en la nueva versión de la aplicación DNSE3 es el que ha sufrido el servicio encargado de mantener la persistencia de los datos. En esta versión se pasa de utilizar Swift a ownCloud. La elección de ownCloud se basó en dos motivos: permite desplegarse sobre contenedores Docker, lo que facilitará el trabajo a realizar si en un futuro se desea que todo DNSE3 se ejecute sobre contenedores y permite la descarga de un directorio completo en formato ZIP realizando una única petición. Esta segunda característica se utiliza para mejorar el rendimiento del servicio encargada de formatear los resultados de las distintas simulaciones para que los usuarios de la aplicación puedan trabajar con estos datos de manera sencilla. La parte negativa de la utilización de ownCloud para la persistencia de datos de DNSE3 se encontró en el servicio encargado de realizar las simulaciones, puesto que se detectó que en trabajos de muchas repeticiones con ownCloud se conseguía un comportamiento menos estable que el se obtenía al utilizar Swift.

Por todas estas razones se concluye que la nueva versión de DNSE3 que distribuye la ejecución de las simulaciones sobre contenedores y que realiza la persistencia de datos utilizando ownCloud es una versión funcional y que presenta mejores prestaciones en muchos casos comparando las prestaciones obtenidas con la versión previa. Esta versión obtiene mejores resultados que la versión presentada en [4] cuando se completan trabajos de un número bajo o medio de simulaciones, pero son peores cuando el número de simulaciones a completar es muy elevado. Aún así, al pasar a utilizar ownCloud para almacenar los datos de la aplicación permite que la realización de los informes de resultados que se les entregan a los alumnos se realice de forma más rápida que con la versión anterior, lo que compensa el aumento de tiempo que se necesita para realizar los trabajos compuestos por un número elevado de si-

mulaciones. Por este motivo, el tiempo que percibe el usuario para que el sistema complete trabajos de distinta longitud será más bajo que el percibía con la versión anterior.

5.3. Líneas de trabajo futuro

Durante las pruebas realizadas para comprobar el rendimiento de la nueva versión de la aplicación se detectaron distintos problemas entre los que destacan la perdida de rendimiento al ejecutar las simulaciones sobre contenedores y la falta de estabilidad al utilizar ownCloud para almacenar los datos de la aplicación. Si se sigue utilizando la misma configuración de la nube de cómputo privada del grupo de trabajo sobre la que se está ejecutando DNSE3 no se puede solventar la pérdida de rendimiento al ejecutar las simulaciones dentro de contenedores, que viene provocada por la doble virtualización al utilizar contenedores sobre máquinas virtuales.

En cambio, sí que se puede seguir continuando la evolución de la aplicación para que solvente los problemas encontrados por el uso de ownCloud. Para ello se propone realizar los siguientes cambios:

- Para reducir el número de comunicaciones entre el Servicio de Almacenamiento y el de Simulación se guardaría los ejecutables de los modelos simulados descargados durante un tiempo. En la versión actual del Servicio de Simulación se está descargando el modelo simulado cada vez que se inicia una nueva simulación, sin tener en consideración las simulaciones previas. La descarga del ejecutable se realiza a través de una URI, por lo que si se replica localmente esta estructura, antes de realizar la descarga del modelo se comprobará si el modelo está descargado localmente. Si lo está, lo utilizará directamente, y si no lo está, se descargará del Sistema de Almacenamiento. Con este sistema de caché se conseguirá reducir la carga de trabajo del Sistema de Almacenamiento. Además, como al eliminar los contenedores que ejecutan réplicas del Servicio de Simulación se borran los datos que han ido generado, no habría problemas de liberación de recursos.

- Otra manera de reducir la carga de trabajo que soporta el Servicio de Almacenamiento basado en ownCloud, es que el Servicio de Simulación almacene de forma persistente los ficheros de resultados que obtenga al completar las simulaciones, pero que además envíe estos ficheros también al Servicio de Informes. De esta forma Informes iría obteniendo los resultados de las simulaciones a medida que se fueran completando, lo que le permitiría comenzar a trabajar en el formateo de dichos resultados antes que el trabajo se finalice. Con ello se consigue liberar de carga de trabajo al Servicio de Almacenamiento ya que Informes no tiene que recoger los resultados de dicho servicio, y los tiempos de realización de los informes deberían mejorar puesto que Informes no tiene que esperar a que se finalice el trabajo completo para que se comiencen a formatear los resultados.

Otra mejora que se podrá realizar a la versión actual de la aplicación es incluir el ritmo al que se están completando las simulaciones en la decisión de escalado del número de réplicas que ejecuten el Servicio de Simulación. En las pruebas realizadas para la realización del capítulo 4 se comprobó que en ocasiones se comienza a reservar recursos para el Servicio de Simulación mediante la creación de una nueva máquina virtual, pero cuando esta finalizaba su inicialización ya no era necesaria debido a que en ese tiempo se habían consumido un número elevado de simulaciones. Este problema ocurre porque el valor del umbral T del Servicio de Monitorización y Escalado es fijo. Si se consiguiera adaptar el valor de dicho umbral al tiempo medio que se está necesitando para ejecutar las últimas simulaciones se podría estimar

la cantidad de simulaciones que se van a realizar en el tiempo que necesita el sistema para activar una nueva máquina virtual, y con ello, decidir si la creación de la nueva máquina virtual estaría justificada.

Para finalizar, aunque se ha comprobado en las pruebas realizadas que la versión actual es completamente funcional se tiene que comprobar su funcionamiento en producción. El estrés al que será sometido DNSE3 durante las pruebas que se realicen con alumnos de las asignaturas a las que va destinada esta aplicación determinará su robustez.

Referencias

[1] Averill M. Law y W. David Kelton. *Simulation Modeling and Analysis*. Ed. por McGraw-Hill Education. 2nd edition. McGraw-Hill, Inc., 1991.

[2] Sergio Serrano Iglesias. «Implementación y evaluación de un entorno de simulación de redes distribuido basado en ns-3 y computación en nube». Trabajo Fin de Master. E.T.S.I. de Telecomunicación, Universidad de Valladolid, septiembre de 2017. URL: http://uvadoc.uva.es/handle/10324/27614.

[3] Miguel L. Bote-Lorenzo y col. «A grid service-based distributed network simulation environment for computer networks education». En: *Computer Applications in Engineering Education* 20.4 (diciembre de 2012), págs. 654-665. DOI: 10.1002/cae.20435.

[4] Sergio Serrano Iglesias y col. «A self-scalable distributed network simulation environment based on cloud computing». En: *Cluster Computing* (2018). DOI: 10.1007/s10586-018-2816-5.

[5] Michael Armbrust y col. «A view of cloud computing». En: *Communications of the ACM* 53.4 (abril de 2010), pág. 50. DOI: 10.1145/1721654.1721672.

[6] Rafael Cano Parra. «Entorno de simulación de redes TCP/IP usando servicios REST basados en la nube computacional». Trabajo Fin de Master. E.T.S.I. de Telecomunicación, Universidad de Valladolid, 2012. URL: http://uvadoc.uva.es/handle/10324/2681.

[7] *What is OpenStack?* URL: https://www.openstack.org/software/ (visitado el 14 de mayo de 2019).

[8] *What is a Container - Docker*. URL: https://www.docker.com/resources/what-container (visitado el 14 de mayo de 2019).

[9] W. Richards Adrion. *Research Methodology in Software Engineering*. Ed. por Tichy, Habermann y Prechelt. Vol. 18. 1. ACM Software Engineering Notes, SIGSoft, 1993, págs. 36-37.

[10] Roy Thomas Fielding. «Architectural Styles and the Design of Network-based Software Architectures». Tesis doct. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (visitado el 30 de junio de 2019).

[11] Jérôme Louvel, Thierry Templier y Thierry Boileau. *Restlet in Action. Developing RESTful web APIs in Java.* Ed. por Mannig Publications. Mannig Publications Co., 8 de octubre de 2012. 432 págs. ISBN: 9781935182344.

[12] ns-3 Network Simulator. *ns-3 Tutorial, Release ns-3.18.1*. Noviembre de 2013. URL: https://www.nsnam.org/docs/release/3.18/tutorial/ns-3-tutorial.pdf (visitado el 25 de junio de 2019).

[13] Pradeep Kumar Singh y Madhuri Kumari. *Containers in OpenStack. Leverage OpenStack services to make the most of Docker, Kubernetes and Mesos*. Pack Publishing Ltd., 22 de diciembre de 2017. 176 págs. ISBN: 978-1-78839-438-3.

[14] Russ Mckendrick y Scott Gallagher. *Mastering Docker - Second Edition*. Packt Publishing Ltd., 21 de julio de 2017. 392 págs. ISBN: 1787280241.

[15] *rkt - rkt Container Engine with CoreOS*. URL: `https://coreos.com/rkt/docs/latest/` (visitado el 11 de junio de 2019).

[16] *Linux Containers - LXC - Introduction*. URL: `https://linuxcontainers.org/lxc/introduction/` (visitado el 11 de junio de 2019).

[17] Gigi Sayfan. *Mastering Kubernetes*. Packt Publishing Ltd., 24 de mayo de 2017. 426 págs. ISBN: 1786461005.

[18] *What is Kubernetes? - Kubernetes*. URL: `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/` (visitado el 12 de junio de 2019).

[19] *How nodes work - Docker Documentation*. URL: `https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/` (visitado el 11 de junio de 2019).

[20] *Docker Swarm strategies - Docker Documentation*. URL: `https://docs.docker.com/swarm/scheduler/strategy/` (visitado el 11 de junio de 2019).

[21] *OpenStack Docs: Zun Installation Guide*. URL: `https://docs.openstack.org/zun/pike/install/index.html` (visitado el 15 de junio de 2019).

[22] *OpenStack Docs: Magnum User Guide*. URL: `https://docs.openstack.org/magnum/latest/user/index.html` (visitado el 16 de junio de 2019).

[23] *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2 de diciembre de 2003. 748 págs. ISBN: 1558609334.

[24] *ownCloud Documentation Overview :: ownCloud Documentation*. URL: `https://doc.owncloud.org/server/index.html` (visitado el 28 de junio de 2019).

[25] *Manage data in Docker - Docker Documentation*. URL: `https://docs.docker.com/storage/` (visitado el 28 de junio de 2019).

[26] *Docker - How do I enable the remote API for dockerd*. URL: `https://success.docker.com/article/how-do-i-enable-the-remote-api-for-dockerd` (visitado el 29 de junio de 2019).

[27] Juan I. Asensio Pérez. «Teletráfico 2017-2018, Práctica II - Teoría de Colas y Simulación con ns-3. Parte 1.» En: *E.T.S.I. de Telecomunicación, Universidad de Valladolid* (marzo de 2018).

[28] *OpenStack Docs: Manage flavors*. URL: `https://docs.openstack.org/horizon/latest/admin/manage-flavors.html` (visitado el 29 de junio de 2019).

[29] *Function Reference: boxplot*. URL: `https://octave.sourceforge.io/statistics/function/boxplot.html` (visitado el 2 de julio de 2019).

Anexo A

Fichero Dockerfile

Con este fichero se crea la imagen base de contenedor de la que parten todas las réplicas de los contenedores que ejecutan el Servicio de Simulación. Para su creación se necesita los ficheros de instalación del simulador ns-3 en su versión `3.18.2`[1], el JAR que contiene el propio Servicio de Simulación y el fichero `almacen.properties` con información del Servicio de Almacenamiento.

```
1  FROM ubuntu:18.04
2
3  #Instalación de ns-3 y sus dependencias
4  RUN apt update && apt upgrade -y && apt install -y gcc-6 g++-6
       python && update-alternatives --install /usr/bin/gcc gcc /usr
       /bin/gcc-6 10 && update-alternatives --install /usr/bin/g++ g
       ++ /usr/bin/g++-6 10
5  ADD ns-allinone-3.18.2.tar.bz2 /opt
6  RUN cd /opt/ns-allinone-3.18.2/ns-3.18.1 && CXXFLAGS="-Wall" ./
       waf -d debug --enable-examples --enable-tests configure && ./
       waf
7  RUN echo "/opt/ns-allinone-3.18.2/ns-3.18.1/build" >> /etc/ld.
       so.conf.d/ns3.conf && ldconfig
8
9  #Instalación JRE
10 RUN apt install -y default-jre
11
12 #Creación del usuario sin privilegios y copia de los ficheros
       necesarios
13 RUN useradd -md /home/simulacion -s /bin/bash -U simulacion
14 USER simulacion:simulacion
15 WORKDIR /home/simulacion
16 COPY --chown=simulacion:simulacion simulation.jar .
17 COPY --chown=simulacion:simulacion almacen.properties .
18
19 #Se establecen las variables de entorno y el punto de entrada
20 ENV queueAddress=10.0.104.238:8081
21 ENV limit=1
22 ENTRYPOINT java -jar simulation.jar $queueAddress $limit
```

[1] https://www.nsnam.org/releases/ns-allinone-3.18.2.tar.bz2