



ULPGC
Universidad de
Las Palmas de
Gran Canaria

Escuela de
Ingeniería Informática



AUTOMATIZACIÓN DEL DESPLIEGUE DE MÁQUINAS VIRTUALES PARA LOS LABORATORIOS DE LA ESCUELA DE INGENIERÍA INFORMÁTICA 2.0

Trabajo de Fin de Título

Grado en Ingeniería Informática (Tecnologías de la Información)

Autor

Héctor Acosta Valido

Tutores

Alexis Quesada Arencibia

Carmelo Rubén García Rodríguez

Resumen

Este proyecto se basa en el trabajo realizado en el TFT con título "Automatización del despliegue de máquinas virtuales para los laboratorios de la Escuela de Ingeniería Informática", con el objetivo de ampliar su funcionalidad. Específicamente, este proyecto busca aumentar las operaciones que los usuarios pueden llevar a cabo sobre las máquinas virtuales.

Por otro lado, una parte importante del trabajo comprende el uso de distintas tecnologías para mejorar tanto el modo que tiene la herramienta de gestionar las tareas de larga duración, como la comunicación entre los distintos componentes del proyecto.

Por último, se implementa un sistema que permite incrustar un escritorio remoto en la propia herramienta web, facilitando el flujo de trabajo de los usuarios.

Contenido

Resumen	3
Índice de Tablas	7
Índice de Ilustraciones	8
Introducción.....	9
1. Estado actual y objetivos	10
1.1. Estado actual.....	10
1.2. Objetivos	11
2. Competencias cubiertas	12
2.1. Competencias Comunes a la Ingeniería Informática (CII).....	12
2.1.1. CII01	12
2.1.2. CII02	12
2.1.3. CII05	12
2.1.4. CII011	12
2.1.5. CII013	13
2.1.6. CII016	13
2.1.7. CII018	13
2.2. Competencias de las Tecnologías de la Información (TI)	13
2.2.1. TI01	13
2.2.2. TI02	13
2.2.3. TI03	14
2.2.4. TI06	14
3. Aportaciones.....	15
3.1. Entorno técnico.....	15
3.2. Entorno personal	15
4. Normativa y legislación	16
4.1. MIT	16
4.2. GNU GPL.....	16
4.3. GNU LGPL.....	16
4.4. BSD	16
4.5. PSFL	17
4.6. MPL	17
5. Metodología de trabajo, planificación inicial y ajustes	18

5.1.	Metodología de trabajo	18
5.2.	Planificación inicial.....	18
5.3.	Ajustes en la planificación.....	20
6.	Tecnologías, herramientas y entorno de trabajo	21
6.1.	Tecnologías usadas	21
6.1.1.	Cliente (DIS vLab Client)	21
6.1.2.	Servidor (DIS vLab Server)	22
6.1.3.	Virtualización	24
6.1.4.	Otros	26
6.2.	Herramientas	26
6.2.1.	JetBrains WebStorm	26
6.2.2.	JetBrains PyCharm	27
6.2.3.	Postman.....	27
6.2.4.	Mozilla Firefox	27
6.3.	Entorno de trabajo.....	27
7.	Análisis previo.....	29
7.1.	Análisis de los casos de uso	29
8.	Análisis del diseño original	30
8.1.	Cliente (DVLC)	30
8.2.	Servidor (DVLS)	33
9.	Desarrollo	35
9.1.	Mejoras en las historias de usuario	35
9.1.1.	Listar los hosts de un laboratorio	35
9.1.2.	Cliente VNC.....	36
9.1.3.	Mostrar información de discos duros	37
9.1.4.	Permitir editar RAM y vCPUs.....	37
9.1.5.	Mostrar parámetros de configuración	38
9.1.6.	Editar laboratorios.....	39
9.1.7.	Implementación de Celery	40
9.1.8.	Implementación de <i>WebSockets</i>	42
10.	Resultados y conclusiones	44
11.	Trabajos futuros.....	45
11.1.	Prueba en un entorno real	45

11.2.	Configuración de clientes.....	45
11.3.	Mejoras en el despliegue	45
Anexo I: Instalación de la herramienta.....		46
	Paquetes necesarios	46
	Configuración de <i>Celery</i>	47
	Configuración del sistema	48
	Servicio DVLS	49
	Configuración de Nginx	50
	SSH	51
	Problemas	51
Anexo II: Configuración de los clientes.....		52
Anexo III: Manual de usuario.....		53
	Conexión VNC	53
	Editar información de un dominio.....	55
	Ver <i>hosts</i> de un laboratorio	56
	Edición de un laboratorio	57
Bibliografía.....		58

Índice de Tablas

Tabla 1: Planificación inicial (Elaboración propia)	18
Tabla 2: Planificación ajustada (Elaboración propia)	20
Tabla 3: Tareas a implementar en Celery	40

Índice de Ilustraciones

Ilustración 1: Diagrama de casos de uso (elaboración propia)	29
Ilustración 2: Organización general del cliente	30
Ilustración 3: Estructura del directorio "src"	31
Ilustración 4: Estructura del directorio "app"	31
Ilustración 5: Diseño del autor original	33
Ilustración 6: Organización general del servidor	33
Ilustración 7: Directorio "app" del servidor	34
Ilustración 8: Interior del directorio "api"	34
Ilustración 9: Contenido de la carpeta "endpoints"	34
Ilustración 10: Modal de listado de hosts	35
Ilustración 11: Modal del escritorio remoto	36
Ilustración 12: Sección de dominios con información sobre los discos duros	37
Ilustración 13: Modal de edición de vCPUs y RAM	38
Ilustración 14: Parámetros de configuración de la herramienta	38
Ilustración 15: Modal de edición de laboratorios	39
Ilustración 16: Cambios en las carpetas al añadir Celery	41
Ilustración 17: Selección de un dominio en ejecución	53
Ilustración 18: Opción de "Escritorio Remoto"	53
Ilustración 19: Modal del escritorio remoto solicitando la contraseña	54
Ilustración 20: Escritorio remoto en funcionamiento	54
Ilustración 21: Selección de la opción "Editar" de un dominio	55
Ilustración 22: Modal de edición de RAM y vCPUs	55
Ilustración 23: Selección de la opción "Ver hosts" de un laboratorio	56
Ilustración 24: Modal de información de hosts de un laboratorio	56
Ilustración 25: Selección de la opción "Editar" de un laboratorio	57
Ilustración 26: Modal de edición de laboratorios	57

Introducción

Al inicio de cada curso, o cuatrimestre, y para satisfacer las necesidades docentes de las asignaturas impartidas en los distintos laboratorios de la Escuela de Ingeniería Informática (EII) y el Departamento de Informática y Sistemas (DIS), estos deben ser configurados de una forma específica y personalizada para cada uno de ellos, ya que las necesidades en cuanto a software cambian en función de la materia.

Hasta ahora, el proceso para la configuración de estos laboratorios, llevado a cabo por el Centro de Cálculo del Departamento de Informática y Sistemas (CCDIS), presentaba ciertos problemas.

En primer lugar, al ser los laboratorios compartidos entre distintas asignaturas, si un usuario corrompiera su partición, o peor aún, el equipo en sí, esto afectaría a la docencia del resto de asignaturas que hacían uso de dicha estación.

Además, un cambio en los requisitos software de una asignatura implicaría tener que volver a configurar todos los equipos del laboratorio, lo cual aumentaría el tiempo necesario para la puesta en marcha de los laboratorios.

Para ofrecer una solución a este problema, se desarrolló un Trabajo de Fin de Título (TFT) previo que proponía “la virtualización de las infraestructuras de los laboratorios, de tal forma que esas particiones base sean sustituidas por máquinas virtuales” [1].

Si bien la configuración de los laboratorios se sigue llevando a cabo de la manera tradicional, el proyecto citado ofrece amplias posibilidades de automatizar las tareas de configuración, como son la creación de máquinas virtuales y plantillas de equipos y el despliegue de estas en los laboratorios.

Con este proyecto, se busca tomar el testigo del anterior autor, ampliando las funcionalidades y capacidades del anterior trabajo, con el objetivo de mejorar su usabilidad, y haciéndolo más eficiente.

Gracias a la naturaleza modular de la herramienta, y a la documentación presente, las tareas de mejora se pueden llevar a cabo siguiendo unos procesos establecidos, que permiten facilitar la detección de errores, y facilitan la solución de estos.

Así, las mejoras a implementar van desde pequeños arreglos que armen a la herramienta de funcionalidades menores, y que permitan la familiarización con el entorno de trabajo, hasta el uso de distintas tecnologías que permitan que las tareas más largas se ejecuten de forma asíncrona, mejorando de esta forma la comunicación entre los distintos componentes.

1. Estado actual y objetivos

1.1. Estado actual

La instalación y configuración de los laboratorios de la EII y el DIS es llevada a cabo por el personal del CCDIS, siguiendo una serie de pasos, como se puede observar en [1, p. 4]

Para solventar esta situación, y agilizar este proceso, se propuso la virtualización de dicha infraestructura, permitiendo así una gestión centralizada.

La virtualización es “la creación a través de software de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de hardware, un sistema operativo, un dispositivo de almacenamiento o cualquier otro recurso de red” [2]. Esto permite a un usuario crear una o varias “máquinas virtuales”, es decir, todo un sistema operativo que se ejecuta de forma simulada, pues los recursos que usa “no existen”.

Para esto, se desarrolló una herramienta de gestión hecha a medida, teniendo en cuenta los requisitos del personal.

Esta herramienta tomó la forma de una aplicación web [3], que se puede dividir en dos componentes. Un cliente, llamado “DIS vLab Client (DVLS)”, y un servidor, llamado “DIS vLab Server (DVLC)”. Ambos componentes se pueden identificar, respectivamente, con el *Front end* y el *Back end* de una aplicación web [4].

Siguiendo este modelo, el cliente DVLC, engloba la interfaz de la aplicación, permitiendo al usuario llevar a cabo las distintas acciones de gestión, mientras se comunica con el *Back end*, (DVLS).

Por su lado, el servidor DVLS recibirá las órdenes del usuario, transformadas en peticiones por el cliente, y las llevará a cabo.

A esto se le conoce como estilo arquitectónico REST [5]. Siguiendo este estilo, un servicio web llamado RESTful (DVLS), permite el acceso a una serie de recursos, mediante unos métodos predefinidos. Así, un cliente que quiera acceder a estos (DVLC) envía una petición con la dirección de un recurso (URI) [6], haciendo uso de uno de los métodos, definidos en el protocolo HTTP [7].

Así, la aplicación creada permitía llevar a cabo las siguientes tareas:

- Creación de dominios: Los dominios son máquinas virtuales. A través del cliente web, un usuario es capaz de crear un dominio a medida, especificando varias características como pueden ser la memoria RAM, el número de núcleos virtuales de CPU, o el espacio de disco duro.
- Creación de plantillas: Una vez los dominios han sido configurados, y se ha instalado el software necesario, es necesario crear una plantilla de estos. Una

plantilla es una versión “descontextualizada” de los mismos. Este paso es necesario para poder distribuir las máquinas virtuales en los laboratorios.

- Definir laboratorios: Un usuario es capaz de definir un laboratorio, dando para él un nombre y un rango de direcciones IP que representan los equipos en este.
- Desplegar plantillas: Por último, el usuario podrá desplegar una o varias plantillas en uno o varios laboratorios.

1.2. Objetivos

Las funcionalidades definidas permiten llevar a cabo la tarea para la que fue diseñada la herramienta. Sin embargo, tras analizar la misma se descubrieron varias posibilidades de mejora que permitirían hacerla más completa, e incluso el autor de esta definió posibles líneas de trabajo [1, pp. 54-57].

El objetivo de este trabajo es identificar, diseñar e implementar posibles mejoras al trabajo original.

Al tratarse de un trabajo de expansión de otro ya existente, se explotan otras facetas necesarias en el desarrollo de aplicaciones, pues será necesario adaptarse a un modelo ya establecido, una situación con la que un desarrollador se encontrará con frecuencia durante su trayectoria profesional.

Desde el punto de vista de las Tecnologías de la Información, tanto el despliegue de la herramienta, como la análisis e implementación de posibles mejoras permiten desarrollar aptitudes muy solicitadas en este ámbito, por lo que servirá de preparación a la hora de aprender a enfocar este tipo de retos.

Por último, el tener que adaptarse a las tecnologías elegidas por el autor original, obliga al alumno a ser autodidacta y a permanecer en constante formación, algo muy importante para todo profesional de la informática.

2. Competencias cubiertas

Las competencias cubiertas por este trabajo se pueden dividir en dos categorías, Las Comunes a la Ingeniería Informática (CII), y las específicas a las Tecnologías de la Información (TI) [8]. A continuación, se pasa a desglosarlas.

2.1. Competencias Comunes a la Ingeniería Informática (CII)

2.1.1. CII01

“Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente”.

Esta competencia puede verse cubierta en todos los puntos del desarrollo de este trabajo, desde el análisis inicial a la implementación y la validación.

2.1.2. CII02

“Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social”.

La puesta en funcionamiento, en un entorno de explotación, de este proyecto cuadra, perfectamente, en la definición de esta competencia.

2.1.3. CII05

“Conocimiento, administración y mantenimiento de sistemas, servicios y aplicaciones informáticas”.

La necesidad de poner en funcionamiento tanto un entorno de trabajo, como un laboratorio de pruebas local pone de manifiesto la presencia de esta competencia.

2.1.4. CII011

“Conocimiento y aplicación de las características, funcionalidades y estructura de los Sistemas Operativos y diseñar e Implementar aplicaciones basadas en sus servicios”.

La naturaleza web de la aplicación, además de su estructura interna y el despliegue en equipos conectados en red coincide con la descripción de esta competencia.

2.1.5. CII013

“Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los Sistemas de información, incluidos los basados en web”.

Los modelos de comunicación entre los distintos componentes de la herramienta hacen posible que esta se identifique con la competencia expuesta.

2.1.6. CII016

“Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería del software”.

Dada la metodología de trabajo seleccionada, que aplica los conocimientos adquiridos durante la carrera, no solo para el desarrollo sino para la mejora de aplicaciones informáticas, parece razonable decir que esta competencia se desarrolla.

2.1.7. CII018

“Conocimiento de la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional”.

El estudio de licencias y normativas de las distintas tecnologías, y los componentes que forman parte de este proyecto obligan al desarrollo de esta competencia.

2.2. Competencias de las Tecnologías de la Información (TI)

2.2.1. TI01

“Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la organización y las comunicaciones”.

La necesidad de ajustarse a los requisitos del personal de la ULPGC hace necesario aplicar esta competencia.

2.2.2. TI02

“Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados”.

Tener presente las posibilidades, en cuanto a medios e infraestructura, con las que cuenta este proyecto es fundamental para el éxito del mismo.

2.2.3. TI03

“Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de las aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomías y usabilidad de los sistemas”.

La herramienta ha sido construida sobre las especificaciones y peticiones expuestas por el personal del DIS y el autor del trabajo original, asegurando el desarrollo de esta competencia.

2.2.4. TI06

“Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil”.

El hecho de que esta herramienta esté construida sobre un entorno web, además de la posibilidad de un despliegue centralizado de máquinas virtuales, a través de una red local, pone de manifiesto que esta competencia ha sido cubierta durante el desarrollo de este proyecto.

3. Aportaciones

3.1. Entorno técnico

La facilitación del despliegue y puesta en funcionamiento de los equipos en los laboratorios de la Escuela de Ingeniería Informática tendría un impacto beneficioso en las labores de mantenimiento de la infraestructura tecnológica, ahorrando tiempo al personal del DIS, pues el método tradicional es especialmente costoso en cuanto a inversión de tiempo.

Por otro lado, el uso de las tecnologías de virtualización está cada vez más extendido, y aunque llevar a cabo una virtualización total de las instalaciones de la escuela no sea posible, este proyecto intenta beneficiarse de lo que esta tecnología ofrece, teniendo en cuenta las limitaciones en cuanto a infraestructura presentes.

3.2. Entorno personal

El hecho de tener que adaptarse a las condiciones de trabajo puestas por un tercero, en este caso el autor original del proyecto, ha permitido desarrollar una actitud autodidacta y analítica a la hora de abordar un trabajo de este tipo.

Además, el desarrollo de una aplicación de este tipo, que incorpora conocimientos de varios campos de la informática, permite al autor obtener una visión general de estos, un rasgo importante a la hora llevar a cabo proyectos futuros.

Por otro lado, la situación global actual, en la que el movimiento se ha limitado, y no se tenía acceso a los entornos de trabajo de la Escuela de Ingeniería Informática, además de la casi obligatoriedad de que toda comunicación fuera por medios telemáticos, ha favorecido la capacidad de toma de decisiones, además de la autonomía, otros dos aspectos fundamentales a la hora de desarrollar una carrera profesional plena.

4. Normativa y legislación

Las distintas herramientas y tecnologías usadas se acogen a varias licencias que se pasa a explicar a continuación.

4.1. MIT

La licencia MIT es originada en el Instituto Tecnológico de Massachusetts. Se considera una “licencia software permisiva”, lo cual quiere decir que permite que una obra sea redistribuida tanto de forma libre como privada [9]

Esta licencia es compatible con otras muchas licencias *copyleft*, que es una práctica legal mediante la que un autor propicia el libre uso y distribución de su obra, siempre que estas libertades se preserven al distribuir sus copias [10] [11].

En este proyecto, esta licencia es usada por varios componentes: Socket.IO, SQLAlchemy, PyJWT, Python pam, Eventlet, Angular, Clarity Desing System, y algunos elementos de noVNC [12] [13] [14] [15] [16] [17] [18].

4.2. GNU GPL

La licencia GNU GPL (GNU *General Public License*) es una licencia muy usada en el mundo del software y el código abierto, y garantiza la libertad de usar, estudiar, compartir y modificar el software. Además de declarar que el software bajo esta licencia es libre, se busca protegerlo de cualquier intento de apropiación que infrinja las libertades citadas anteriormente. Fue creada originalmente para el proyecto GNU [19].

Bajo esta licencia, se encuentran CentOS y uWSGI [20] [21].

4.3. GNU LGPL

La licencia GNU LGPL (GNU *Lesser General Public License*) se basa en la GNU GPL, pero su principal diferencia es que no exige que las obras creadas a partir de una original sean GPL. En definitiva, esto quiere decir que el software creado a partir de otro software LGPL puede ser libre, o no [22].

Se encuentran bajo esta licencia los siguientes componentes del proyecto: Libvirt, Paramiko y PolicyKit [23] [24] [25].

4.4. BSD

La licencia BSD, en origen, era otorgada a los sistemas *Berkeley Software Distribution*. Es una licencia permisiva, como la licencia MIT, con la que comparte muchas similitudes.

Se la puede encontrar en los siguientes elementos: *Flask*, *Celery*, *Werkzeug*, *Redis* y *Nginx* [26] [27] [28] [29] [30].

Werkzeug y Redis usan una versión modificada, conocida como BSD modificada, e impide que el receptor del software pueda tener derecho a usar el nombre del autor original sin su permiso.

Por otro lado, Nginx usa una versión llamada BSD simplificada.

4.5. PSFL

La PSFL (*Python Software Foundation License*) es una licencia de software libre permisiva, parecida a la BSD, y compatible con la licencia GPL. Al igual que la licencia BSD, permite modificaciones al código fuente, además de trabajos derivados, sin que los resultados de estos tengan que ser de código abierto [31].

Bajo esta licencia se encuentran las versiones de *Python*, desde la 2.1, en 2001 [32].

4.6. MPL

La MPL (*Mozilla Public License*) es una licencia para software libre, de código abierto y mantenida por la Fundación *Mozilla*. Es un híbrido entre la BSD y la GNU GPL, buscando encontrar un equilibrio entre los intereses de los desarrolladores de código abierto y los propietarios [33].

Bajo esta licencia se encuentra noVNC [34].

5. Metodología de trabajo, planificación inicial y ajustes

5.1. Metodología de trabajo

Para el desarrollo de este proyecto se ha optado por seguir una metodología incremental [35], pues se está familiarizado con esta, al haber sido usada con anterioridad, y ser enseñada en varias asignaturas de la carrera. Además, se adecúa al tipo de proyecto que se va a llevar a cabo.

Este modelo divide el desarrollo en una serie de etapas (iteraciones), de cuya conclusión se obtiene un “incremento”, que aporta funcionalidad al producto final. Cada uno de estos incrementos cuenta con su propia fase de análisis, diseño e implementación.

Al dividir el desarrollo en pequeñas iteraciones, se gana en flexibilidad, y es posible adaptarse a las distintas situaciones imprevistas o cambios en los requisitos que aparezcan.

5.2. Planificación inicial

La planificación inicial del proyecto queda plasmada en la Tabla 1:

Fases	Tareas	Duración Estimada (Horas)
Estudio previo / Análisis	Tarea 1.1: Análisis del trabajo original y familiarización con su funcionamiento	35
	Tarea 1.2: Estudio de las tecnologías necesarias para continuar el desarrollo	
	Tarea 1.3: Preparación de un entorno de pruebas compatible con el trabajo original	
Diseño / Desarrollo / Implementación	Tarea 2.1: Diseño de implementación de las mejoras	185
	Tarea 2.2: Desarrollo / ampliación de las funcionalidades propuestas	
Evaluación / Validación / Prueba	Tarea 3.1: Generación de plantillas de distinto tipo	40
	Tarea 3.2: Despliegue de máquinas virtuales	
	Tarea 3.3: Análisis de resultados	
Documentación / Presentación	Tarea 4.1: Desarrollo de la memoria del trabajo	40
	Tarea 4.2: Presentación del TFT	

Tabla 1: Planificación inicial (Elaboración propia)

Las primeras sesiones de trabajo se destinaron a analizar y probar la herramienta ya existente, y a abstraer el diseño general de la misma. Esto tiene dos objetivos:

- Por un lado, detectar las áreas donde se pueden encontrar mejoras.
- Además, comprender el diseño a nivel de programación de la herramienta permitiría agilizar cualquier implementación futura.

Una vez identificadas las mejoras que se van a implementar, estas se dividieron en dos grupos:

- En el primer grupo, se encontraban pequeñas mejoras, principalmente en el *Front end*. Estas serían las primeras en llevarse a cabo, lo cual permitiría familiarizarse con la estructura software de la herramienta.
- En el segundo grupo se encuentran tareas más grandes, que modificaban la estructura general del proyecto y que abarcaban tanto el *Front end* como el *Back end*. Para llevar a cabo estas tareas es necesario conocer el funcionamiento de la herramienta, por lo que su implementación se dejó para el final.

Habiendo identificado las distintas líneas de trabajo a llevar a cabo, esto permitió llevar a cabo un diseño preliminar de los distintos “incrementos”, sabiendo cuál debía ser la salida de cada uno de ellos.

Debido a la selección de tecnologías usadas por el autor original, se tuvo que dedicar una parte del desarrollo a la formación en el uso de estas.

Conociendo ya los distintos objetivos, y con los conocimientos necesarios para enfrentarse a la implementación de las distintas mejoras, solo queda preparar un entorno de desarrollo que sea compatible con el original. El objetivo de esta fase es obtener un entorno de desarrollo que permita probar los cambios hechos de una forma sencilla y ágil.

Tras esta fase de Análisis, se pudo pasar a la implementación, la fase más larga de este proyecto. Como se expuso antes, esta se dividió en distintos “incrementos” que otorgaban alguna funcionalidad a la herramienta, y que se categorizan en dos grandes grupos.

Para cada incremento, se debía llevar a cabo un análisis de lo que se quería obtener y del entorno, tras lo que se pasaba a un diseño preliminar. Esta tarea se vio facilitada debido a que la herramienta original se diseñó de forma que añadir componentes nuevos se podía llevar a cabo siguiendo un proceso establecido y relativamente sencillo. Incluso las últimas tareas, que requerían llevar a cabo ciertos cambios en la estructura del proyecto, pudieron ser diseñadas con relativa facilidad.

Aunque en la Tabla 1 se reconoce la validación como otra fase, lo cierto es que el desarrollo incremental implica que en cada “iteración” se debe llevar a cabo una evaluación de lo implementado.

5.3. Ajustes en la planificación

Durante la ejecución de la planificación propuesta, se ponen de manifiesto varios errores a la hora de estimar el esfuerzo, en horas, necesario. Esto se expone más claramente en la Tabla 2.

Fases	Tareas	Duración Estimada (Horas)	Duración Ajustada (Horas)
Estudio previo / Análisis	Tarea 1.1: Análisis del trabajo original y familiarización con su funcionamiento	35	50
	Tarea 1.2: Estudio de las tecnologías necesarias para continuar el desarrollo		
	Tarea 1.3: Preparación de un entorno de pruebas compatible con el trabajo original		
Diseño / Desarrollo / Implementación	Tarea 2.1: Diseño de implementación de las mejoras	185	230
	Tarea 2.2: Desarrollo / ampliación de las funcionalidades propuestas		
Evaluación / Validación / Prueba	Tarea 3.1: Generación de plantillas de distinto tipo	40	60
	Tarea 3.2: Despliegue de máquinas virtuales		
	Tarea 3.3: Análisis de resultados		
Documentación / Presentación	Tarea 4.1: Desarrollo de la memoria del trabajo	40	40
	Tarea 4.2: Presentación del TFT		

Tabla 2: Planificación ajustada (Elaboración propia)

En primer lugar, las tareas de formación llevaron más tiempo del previsto. Esto se debe a que las tecnologías usadas están en continua evolución, lo que causa que las versiones usadas para el desarrollo original ya hayan quedado obsoletas. Esto obliga a tener especial cuidado a la hora de consultar la documentación.

En la fase de implementación, el error de estimación se debe a que el cálculo original era muy optimista a la hora de calcular el esfuerzo necesario.

La tarea de validación, que incluye tanto las pruebas llevada a cabo sobre cada una de las mejoras implementadas, como las pruebas hechas para asegurar que la herramienta funciona en un entorno de explotación consumieron más tiempo del calculado inicialmente, debido a la necesidad de llevar a cabo pruebas en distintos entornos, con distintas versiones de las tecnologías, para comprobar con cuáles el funcionamiento era correcto.

6. Tecnologías, herramientas y entorno de trabajo

6.1. Tecnologías usadas

Para el uso de las tecnologías no ha habido opciones entre las que elegir. Este trabajo debía adaptarse a las tecnologías usadas por su antecesor, las cuales se describirán en los siguientes apartados.

6.1.1. Cliente (DIS vLab Client)

Para el desarrollo del cliente fueron necesarias las siguientes tecnologías, principalmente ligadas al *Front end*.

6.1.1.1. Angular

“Angular” es un “*Framework* de diseño de aplicaciones y una plataforma de desarrollo para crear *Single-page application (SPA)* eficientes y sofisticadas” [36]. Para entender esto, es necesario definir ciertos conceptos:

- *Framework*: En lo que a software se refiere, un *Framework* es un software que da una funcionalidad básica que un desarrollador puede modificar para darle una funcionalidad específica [37].
- *Single-page application (SPA)*: Una SPA, o aplicación de página única es un tipo de aplicación o sitio web en el que la interacción con el usuario se lleva a cabo reescribiendo una sola página, al contrario del modelo tradicional en el que el buscador recarga páginas nuevas completas [38].

Así, “Angular” ofrece las herramientas básicas necesarias para el desarrollo de la aplicación cliente.

Su organización en “componentes”, permite confeccionar sitios web de una forma relativamente sencilla y eficiente.

6.1.1.2. Clarity Design System

“*Clarity Design System*”, o “*Project Clarity*”, es un “sistema de diseño *open source* que combina unas pautas de diseño de experiencia de usuario (UX), un *Framework* HTML/CSS, y los componentes de Angular” [39]. “*Project Clarity*” permite editar el diseño de un sitio de una forma relativamente sencilla, abstrayendo gran parte del código necesario para esto, y creando una serie de componentes que se pueden combinar y personalizar. Además, cuenta con una integración en Angular que permite que ambas tecnologías funcionen de forma coherente y eficiente.

6.1.1.3. noVNC

“noVNC es tanto un cliente VNC, como una librería JavaScript, y una aplicación escrita sobre dicha librería, que funciona en cualquier explorador moderno” [34]. Para comprender esto, conllevaría explicar qué es un cliente VNC.

VNC, o *Virtual Network Computing* (Computación Virtual en Red) es un programa que permite observar las acciones de un servidor, como la salida del monitor, de forma remota [40]. Siguiendo esta lógica, un cliente VNC es una aplicación capaz de conectarse a un servidor y observar las acciones que ocurren en este.

Así, noVNC permite incrustar un cliente VNC en cualquier sitio a través de su aplicación y su librería JavaScript.

6.1.2. Servidor (DIS vLab Server)

Para el desarrollo del servidor fue necesario el uso de las siguientes tecnologías:

6.1.2.1. Python

“*Python* es un lenguaje de programación interpretado, interactivo y orientado a objetos” [41]. *Python* soporta el uso de módulos, excepciones, escritura dinámica, tipos de datos dinámicos de alto nivel y clases, además de ser extensible mediante el uso de C y C++. Se puede considerar un lenguaje multiparadigma, pues soporta programación orientada a objetos, imperativa y funcional, y se distribuye por la “*Python Software Foundation*”, bajo una licencia de código abierto [42].

Python es un lenguaje que destaca por ser altamente accesible, rasgo por el que es hoy en día uno de los lenguajes de programación más usados. Aunque es especialmente famoso en los campos de la inteligencia artificial y el *machine learning*, es un lenguaje muy adaptable [43].

6.1.2.2. Flask

“*Flask* es un *framework* WSGI ligero para el desarrollo de aplicaciones web. Está diseñado para que empezar sea fácil y rápido” [44]. Para comprender esta definición, conviene explicar el siguiente término:

- WSGI: Estas son las siglas de *Web Server Gateway Interface*, y es una convención que permite a los servidores web enviar peticiones a aplicaciones web o *frameworks* escritos en *Python* [45]. Su origen se debe a que los desarrolladores de aplicaciones web querían poder hacer sus programas usando el lenguaje *Python*, pero los servidores web de entonces no eran capaz de interpretarlo.

Conociendo esto, se puede definir *Flask* como un *framework* de desarrollo web que hace uso de la tecnología WSGI para permitir el desarrollo de estas con *Python*.

6.1.2.3. SQLite

“SQLite es una librería de C, que implementa un motor de bases de datos SQL pequeño, rápido, autónomo, muy fiable y completo. Se encuentra en todos los teléfonos móviles y en la mayoría de los ordenadores” [46]. Se definirán los siguientes términos, que ayudarán a comprender la descripción:

- Sistema de Gestión de Bases de Datos (SGBD): Es un conjunto de programas que permiten llevar a cabo operaciones sobre una base de datos. Se divide en una serie de subsistemas, o componentes [47].
- Motor de base de datos: Componente de los SGBD que acepta peticiones del resto de subsistemas o componentes, y permite el acceso a las bases de datos en los dispositivos de almacenamiento [47].
- SQL (*Structured Query Language*): Es un lenguaje de programación usado para administrar bases de datos. Permite hacer peticiones sobre un SGBD para obtener la información contenida en una base de datos [48].

Comprendiendo estos conceptos, se puede entender SQLite como una librería del lenguaje de programación C, que permite la gestión de un motor de bases de datos.

Ya que se puede usar como sistema de bases de datos, y no necesita la configuración de un servidor para funcionar, SQLite es usado por una amplia colección de aplicaciones, entre las que destacan los sistemas operativos *Android*, *iOS*, o *Windows Phone*, además de los navegadores *Google Chrome* o *Mozilla Firefox* [49].

Aunque está escrito en C, cuenta con interfaces que permiten usarlo en otros lenguajes, como *Python*.

6.1.2.4. uWSGI

“El proyecto uWSGI tiene como objetivo desarrollar un conjunto de soluciones que permita construir servicios de alojamiento” [50]. Se podría decir que el proyecto busca “centralizar” el desarrollo de servicios de alojamiento web, haciendo que todas las aplicaciones que componen un servicio de este tipo usen una misma API, y un estilo de configuración común.

Se puede definir una API, o *Application Programming Interfaces*, como “una especificación formal que establece cómo un módulo de software se comunica o interactúa con otro” [51]. A grandes rasgos, una API permite que dos programas se comuniquen.

En este proyecto, uWSGI permitirá que un servidor *Flask* funcione en un servidor web, como *Nginx*, en un entorno de producción.

6.1.2.5. NGINX

NGINX es un servidor web de código abierto que, además, es capaz de funcionar como proxy inverso [52].

Un proxy inverso, como su nombre indica, funciona de forma contraria a un proxy *forward*. Es decir, un proxy *forward* hace de intermediario para que sus clientes se pongan en contacto con cualquier servidor, y un proxy inverso permite a sus servidores asociados ser contactados por cualquier cliente [53].

6.1.2.6. Celery

Celery es una “cola de tareas distribuida”. Es decir, es un “mecanismo que permite distribuir el trabajo a lo largo de varios hilos o máquinas” [54]. La entrada que recibe es un a tarea, y un proceso llamado “trabajador”, monitoriza de forma continua las colas de tareas buscando una que ejecutar.

Para conseguir esto, hace uso de mensajes, a través de un “agente”, que hace de intermediario entre clientes y trabajadores.

Para este proyecto, Celery permite ejecutar tareas largas en segundo plano, permitiendo que la aplicación siga funcionando mientras estas se ejecutan.

6.1.2.7. WebSockets

WebSockets es una tecnología que permite establecer un canal de comunicación bidireccional, full-dúplex entre un cliente y un servidor sobre un socket TCP [55].

Al permitir comunicación bidireccional, y a través de eventos, el uso de esta tecnología permitirá mejorar la forma en que ambas aplicaciones, DVLS y DVLC, se comuniquen de una forma más eficiente, ya que ahora el servidor es capaz de iniciar comunicaciones con el cliente para indicarle que, por ejemplo, la ejecución de una tarea ya ha finalizado.

En este caso, se usará la librería Socket.IO, que cuenta con implementaciones para *Angular* y *Python* [56].

6.1.3. Virtualización

El funcionamiento básico de la herramienta se basa en el uso de la virtualización. Para esto, se hace uso de varias tecnologías. Antes de continuar con la descripción de las tecnologías, describir algunos conceptos:

- *Hypervisor*: O Monitor de Máquina Virtual (*Virtual Machine Monitor*), es una plataforma software, firmware o hardware que nos permite crear y ejecutar máquinas virtuales. Es el encargado de la ejecución de los sistemas operativos anfitriones [57].
- Virtualización hardware: Consiste en virtualizar computadores como plataformas hardware completas. Es decir, se virtualiza todo un computador, con su CPU, su memoria RAM y su espacio de almacenamiento, entre otros [58]. Este tipo de virtualización incluye varias vertientes y tecnologías que mejoran su

funcionamiento, aunque describirlas se encuentra fuera del objetivo de este trabajo. Este tipo de virtualización será la usada por la herramienta desarrollada.

- Virtualización de escritorio: Consiste en encapsular todos los datos que conlleva la ejecución de un escritorio completo (datos y programas en ejecución), y separarlo de la máquina física. Así, dicho escritorio se ejecuta y almacena desde un servidor remoto, mientras que otro equipo actúa como cliente [59].
- Containerización: La containerización, o virtualización a nivel de sistema operativo, permite la ejecución de múltiples instancias, llamadas contenedores, de espacios de usuario. Los programas que se ejecutan en estos contenedores los ven como computadores reales [60].

6.1.3.1. Libvirt

“Libvirt es una colección de software que provee de un modo conveniente de gestionar máquinas virtuales y otras funcionalidades de la virtualización, como el almacenamiento y las interfaces de red” [61]. En otras palabras, el conjunto de herramientas que componen libvirt nos permiten manejar máquinas virtuales. Estas herramientas son:

- API: La API de libvirt, escrita en C, son el conjunto de funciones que permiten la gestión de la virtualización. Aunque esté escrita en C, es accesible desde otros lenguajes de programación, como *Python*, o Java [23].
- libvirtd: libvirtd es un programa que se ejecuta como un demonio en los servidores que actúan como anfitriones de virtualización, y que se encarga de distintas tareas de gestión de los clientes, como pueden ser el inicio, parada y migración de máquinas virtuales entre servidores [62].
- virsh: Es un programa de línea de comandos que se puede usar para gestionar máquinas virtuales. Es la principal interfaz con la API de libvirt [63].

Así, libvirt da toda la infraestructura necesaria para poder montar un entorno de virtualización completo. Además, es un proyecto de código abierto, y su desarrollo continúa hoy en día.

6.1.3.2. KVM

KVM es un módulo del núcleo de Linux que hace las veces de hipervisor, y que está incluido por defecto en Linux desde el 5 de febrero de 2007. Aunque KVM permite la abstracción de dispositivos, no es capaz de emular procesadores. KVM es un software que sigue siendo libre en su totalidad.

6.1.4. Otros

Además de las tecnologías ya mencionadas, para facilitar el desarrollo del trabajo se han usado las siguientes tecnologías adicionales.

6.1.4.1. Git

Git es un sistema de control de versiones gratuito y de código abierto, que permite agilizar el desarrollo de proyectos de programación [64].

El control de versiones consiste en la gestión de los cambios o actualizaciones que se llevan a cabo sobre un producto o proyecto. Si bien, el control de versiones puede hacerse manualmente, los sistemas de control de versiones (VCS), permiten agilizar esta tarea y hacerlo de forma eficiente y rápida. Aunque el control de versiones puede llevarse a cabo en cualquier campo, es principalmente conocido en entornos informáticos [65].

6.1.4.2. CentOS

CentOS es una distribución de Linux derivada de *Red Hat Enterprise Linux* (RHEL). El proyecto CentOS es gestionado por la comunidad, y busca dar una buena plataforma sobre la que poder construir comunidades *open source*. El proyecto CentOS cuenta con una junta de gobierno que dirige varios grupos llamados “*Special Interest Groups*”, o SIGs, que son los encargados de proveer mejoras, añadidos o cambios para el núcleo de CentOS Linux [66].

CentOS es un sistema operativo de código abierto, que busca ofrecer a sus usuarios un software de clase empresarial, gratuito [20], y ya sido el usado para el desarrollo del proyecto.

6.2. Herramientas

Para el desarrollo del trabajo, se hizo necesario el uso de las siguientes herramientas:

6.2.1. JetBrains WebStorm

WebStorm es un IDE (*Integrated Development Environment* o Entorno de Desarrollo Integrado), centrado en el desarrollo web, JavaScript y TypeScript, desarrollado por *JetBrains* [67].

Por defecto, cuenta con soporte para Angular, permitiendo agilizar el desarrollo de proyectos que usen este *framework*. Aunque la licencia es de pago, la universidad cuenta con una licencia de estudiantes que se pudo usar durante el desarrollo.

6.2.2. JetBrains PyCharm

PyCharm es un IDE dirigido para el desarrollo con *Python*. Al igual que *WebStorm*, está desarrollado por *JetBrains*, aunque este último cuenta con una versión *open source*, llamada *PyCharm Community Edition*, que ha sido la usada en este proyecto [67].

6.2.3. Postman

“Postman es una plataforma colaborativa para el desarrollo de APIs” [68].

Es un software que permite comprobar el funcionamiento de una API. Ha sido de especial utilidad durante el desarrollo de este proyecto, tanto a la hora de entender cómo funcionaba el proyecto original, como para probar las mejoras antes de implementarlas en el cliente.

6.2.4. Mozilla Firefox

“Firefox es un navegador web, libre y de código abierto, desarrollado para Linux, Android, iOS, macOS y Microsoft Windows...” [69].

Al tratarse de una aplicación web, ese necesario usar un navegador para probar las características implementadas. Se ha elegido sobre otros navegadores, como Google Chrome por su alta compatibilidad con el entorno Linux, además de por ser de código abierto.

6.3. Entorno de trabajo

El desarrollo del proyecto se llevó a cabo sobre un ordenador portátil, Acer TravelMate P259-MG-549Q, con las siguientes características:

- Procesador Intel i5-6200U (Dual Core 2,3GHz).
- 16GB de memoria RAM DDR4 2133MHz.
- Disco de estado sólido de 250GB.
- Tarjeta gráfica GeForce GTX 940MX con 2GB de VRAM.
- El sistema operativo instalado es CentOS 7.

Sobre este mismo ordenador se llevaron a cabo algunas pruebas del despliegue de la herramienta.

Como ordenador cliente, esto es, el computador sobre el que se desplegarían las máquinas virtuales, es decir, que actuaría como una estación en un laboratorio, se contaba con un equipo con las siguientes características:

- Procesador AMD FX-8350 4,0GHz 8X Black Edition.
- 16GB de memoria RAM DDR3 1600MHz.
- Disco duro mecánico de 1TB.
- Placa base Gigabyte GA-970A-DS3P Rev 2.0.
- El sistema operativo instalado es CentOS 7.

Además de las pruebas de despliegue, sobre este computador también se llevaron a cabo pruebas del despliegue de la herramienta.

Los equipos se encontraban conectados a la misma red local.

7. Análisis previo

Antes de poder comenzar el desarrollo de las mejoras de la herramienta, es necesario llevar a cabo un análisis de esta, y descubrir en qué sentidos se puede mejorar.

Este análisis tiene dos grandes objetivos: En primer lugar, mediante varias pruebas de los distintos casos de uso, se busca encontrar qué funcionalidades se podrían implementar, para mejorar la usabilidad de la herramienta. Por otro lado, se busca comprender lo mejor posible cómo está organizada, de forma interna, la herramienta, cómo se comunican sus componentes, y cómo se añadirían funcionalidades a estos.

Así, comprendiendo qué se puede mejorar, y cómo encajan estas mejoras en la infraestructura general, se podrán establecer las distintas líneas de trabajo.

7.1. Análisis de los casos de uso

En la memoria del trabajo original [1, pp. 45,46], se definen una serie de casos de uso sobre los que se trabaja. Estos son: ver *Dashboard*, ver dominio, crear dominio, encender dominio, apagar dominio, eliminar dominio, clonar dominio a plantilla, seleccionar dominio, ver laboratorios, crear laboratorio, eliminar laboratorio, seleccionar laboratorio, ver plantillas, desplegar plantilla, eliminar plantilla y seleccionar plantilla.

Así, durante las primeras sesiones de trabajo, además de instalar y configurar la herramienta, se llevó a cabo un trabajo de pruebas de los distintos casos de uso.

Así, una vez acabadas estas pruebas, se confeccionó la siguiente lista de mejoras, que queda reflejada en la Ilustración 1:

- Se añadirían los siguientes casos de uso:
 - Conectarse mediante VNC a un dominio en ejecución.
 - Editar RAM y vCPUs de un dominio.
 - Editar la información de un laboratorio.
 - Listar los hosts de un laboratorio.
 - Ver los parámetros de configuración.

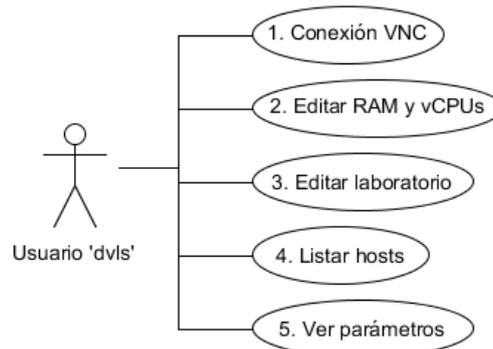


Ilustración 1: Diagrama de casos de uso (elaboración propia)

Además, para el caso de uso “Ver dominio”, se añadiría la funcionalidad de “Mostrar información de disco duro”.

8. Análisis del diseño original

Una vez detectadas cuáles pueden ser las posibles mejoras, se debe analizar cómo estas encajan en el diseño de la aplicación.

A continuación, para cada componente de la herramienta, se explicará cómo encajan las mejoras.

8.1. Cliente (DVLC)

Al tratarse de un proyecto de Angular, el cliente sigue la estructura general de este *framework*, mostrada en la Ilustración 2.

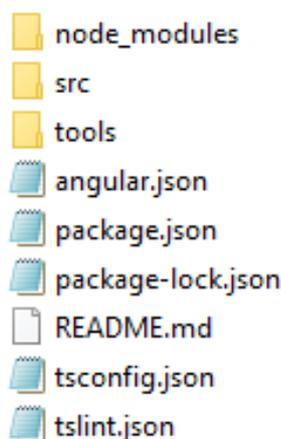


Ilustración 2: Organización general del cliente

Donde cada elemento tiene una utilidad [70]:

- Directorio “*node_modules*”: Aquí se almacenan los paquetes y dependencias usados por el proyecto.
- Directorio “*src*”: Aquí se almacenan los proyectos del código fuente del proyecto.
- Directorio “*tools*”: Este directorio no pertenece a la estructura general de un proyecto Angular. Ha sido añadido para almacenar las herramientas externas que serán usadas por este proyecto en particular.
- Fichero “*angular.json*”: Contiene la configuración general para el proyecto, como las opciones de ejecución.
- Fichero “*package.json*”: Configuración de los paquetes y dependencias del proyecto.
- Fichero “*package-lock.json*”: Información sobre la versión de los distintos paquetes y dependencias usados.
- Fichero “*README.md*”: Documentación general de la aplicación.
- Fichero “*tsconfig.json*”: Configuración de TypeScript.
- Fichero “*tslint.json*”: Configuración de TSLint.

De los directorios descritos, tiene especial interés la carpeta “src”, ilustrado en la Ilustración 3, así que se pasará a analizarla en profundidad, describiendo los componentes más significativos:

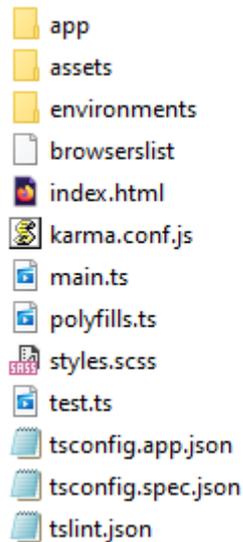


Ilustración 3: Estructura del directorio "src"

- Directorio “app”: Contiene los ficheros que se corresponden con los componentes de la aplicación.
- Directorio “assets”: Contiene imágenes y otros medios que usará la aplicación.
- Directorio “environments”: Contiene configuración de entorno para la aplicación, como variables.
- Fichero “index.html”: Página HTML base. Es la que se envía cuando alguien visita el sitio web.
- Fichero “main.ts”: Es el “punto de entrada” a la aplicación. La compila y carga el módulo base de esta.
- Fichero “styles.scss”: Es la lista de los ficheros CSS que usa el proyecto.

A continuación, se mostrará la estructura del fichero “app”, Ilustración 4, de principal importancia para el proyecto.

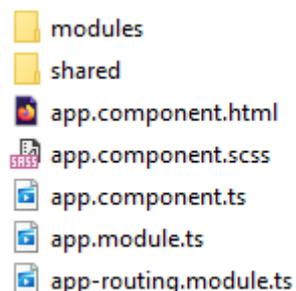


Ilustración 4: Estructura del directorio "app"

- Directorio “modules”: Aquí se almacenarán los distintos módulos que componen nuestra aplicación.

- Directorio “*shared*”: Aquí se almacenan todos los recursos que se comparten entre varios componentes. Un ejemplo sería el servicio *Restful*, que controla todo lo relacionado con la comunicación con el *back end*.
- Fichero “*app-routing.module.ts*”: Define la lógica de navegación de la app.
- Fichero “*app.module.ts*”: Define al módulo raíz.
- Fichero “*app.component.html*”: Este fichero define el *template* asociado al componente principal, o raíz, de la aplicación.
- Fichero “*app.component.scss*”: Define los estilos del componente raíz.
- Fichero “*app.component.ts*”: Define la lógica del componente raíz.

Antes de seguir profundizando en la organización de la aplicación, merece la pena pararse a analizar cómo estos ficheros muestran el funcionamiento del sistema de módulos y componentes de *Angular*.

En primer lugar, conviene definir qué es un módulo en *Angular*. Un módulo es la forma que tiene *Angular* de organizar partes de su código que se relacionan de alguna forma. Esto permite organizar la aplicación de una forma coherente. No se describen unas normas exactas de cuándo se debe agrupar los componentes dentro de un módulo, por lo que es responsabilidad del desarrollador encontrar el modelo que más se adecúe a la naturaleza de su proyecto [71].

Sabiendo un módulo agrupa varios componentes, conviene describir más detalladamente un componente de *Angular*. Un componente controla algo que vemos en pantalla, aunque un componente puede agrupar otros. En una misma página, un componente puede agrupar la vista general, mientras que otros podrían encargarse de controlar cada elemento que se vea. El nivel de complejidad de cada componente varía, aunque lo más correcto es intentar que estos sean lo más simples posible [72].

Un módulo está descrito por un solo fichero, cuyo nombre sigue el esquema “[nombre-del-módulo.module.ts]”, mientras que un componente se compone de un fichero “[nombre-del-componente.component.ts]”, otro llamado “[nombre-del-componente.component.html]”, y un último que se llama “[nombre-del-componente.component.scss]”.

Así, cuando se quiera crear un nuevo componente o módulo, entre otras acciones, se deberán crear los ficheros que los definen. Entender esto permitirá facilitar la implementación de las mejoras.

Antes de continuar, conviene analizar los contenidos del fichero “*modules*”, Ilustración 5, lo que permitirá comprender la estructura que el autor original diseñó para su herramienta [1, pp. 57,58].

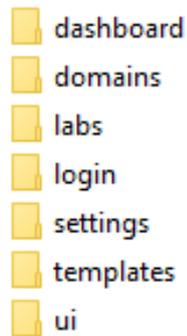


Ilustración 5: Diseño del autor original

En esta estructura de carpetas, cada una de ellas representa un módulo. El autor decidió dividir la aplicación en módulos que representarían cada una de las secciones del sitio web.

- *Dashboard*: Representa la página de inicio de la aplicación.
- *Domains*: Agrupa los componentes necesarios para la gestión de dominios.
- *Labs*: Módulo para la gestión de laboratorios.
- *Login*: Módulo que implementa la página de *login* y su lógica.
- *Settings*: Módulo encargado de administrar los parámetros de configuración que hacen que la aplicación funcione correctamente.
- *Templates*: Módulo encargado de la gestión y despliegue de plantillas.
- *UI*: Módulo que contiene la interfaz de usuario.

Estos conocimientos ya permiten averiguar dónde encajarían las mejoras propuestas.

8.2. Servidor (DVLS)

El servidor, DVLS, sigue la estructura mostrada en la Ilustración 6:

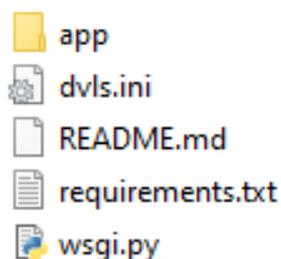


Ilustración 6: Organización general del servidor

- Directorio *“app”*: En esta carpeta se encuentra el código fuente de la aplicación.
- Fichero *“dvls.ini”*: Configuración general para el servidor uWSGI.
- Fichero *“README.md”*: Descripción general del proyecto.
- Fichero *“requirements.txt”*: Paquetes de *Python* requeridos por la aplicación.
- Fichero *“wsgi.py”*: *Script* que inicia la ejecución de la aplicación.

En la Ilustración 7, se mostrarán los ficheros internos de la aplicación, comenzando por el directorio “*app*”:

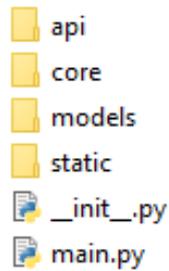


Ilustración 7: Directorio “*app*” del servidor

En este fichero, se puede apreciar la estructura general de la aplicación.

- El directorio “*api*”: Contiene todo lo referente a la *API RESTful*.
- El directorio “*core*”: Contiene los *scripts* que permiten poner en funcionamiento la aplicación.
- El directorio “*models*”: Almacena las conversiones a *Python* de los distintos modelos de datos necesarios, además del fichero de la base de datos.
- El directorio “*static*”: Contiene el código del *Front end*, compilado para poder ser incluido en el servidor de producción.

De los directorios encontrados, el más destacable es el de “*api*”, apreciable en la Ilustración 8.

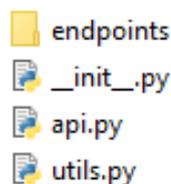


Ilustración 8: Interior del directorio “*api*”

De estos contenidos, lo más importante para comprender el funcionamiento es el directorio “*endpoints*”. Este contiene los recursos que ofrece la *API*, destinando un fichero a cada recurso, como se puede apreciar en la Ilustración 9.

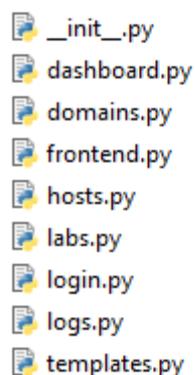


Ilustración 9: Contenido de la carpeta “*endpoints*”

9. Desarrollo

Conociendo la estructura general, llega el momento de plantearse dónde encajan las mejoras que se han propuesto.

9.1. Mejoras en las historias de usuario

En primer lugar, se describirán las historias de usuario implementadas, en el orden que fueron incluidas.

9.1.1. Listar los hosts de un laboratorio

Esta funcionalidad requiere editar tanto el *front end* como el *back end*.

En el cliente, DVLC, es necesario añadir un nuevo componente, un modal, donde se mostrarán los hosts de un laboratorio. Un modal es una ventana emergente que centra la acción del usuario, y le pide llevar a cabo una acción [73].

En este caso, cuando un usuario quiere ver los hosts de un laboratorio, se encontrará el modal expuesto en la Ilustración 10.



Código	Usuario de acceso	IP
LAB_01_0	root	192.168.1.43
1 host		

CERRAR

Ilustración 10: Modal de listado de hosts

En un entorno de explotación real, este modal mostraría cada uno de los equipos del laboratorio.

Además del trabajo del cliente, el servidor también requiere trabajo. El autor original ya había preparado un *endpoint* para esto, por lo que solo hizo falta editar la URL asignada, y llevar a cabo algunas correcciones en el código original para que la comunicación con el cliente fuera correcta.

La elección de realizar una tarea relativamente sencilla en primer lugar fue intencionada, pues permitió un mayor nivel de conocimiento del funcionamiento interno de la herramienta y sus componentes.

9.1.2. Cliente VNC

La siguiente tarea fue la implementación de un cliente VNC que formase parte de la propia herramienta, y que eliminara la necesidad de usar software externo, además de agilizar las conexiones.

Tras analizar las posibilidades de implementación de esta funcionalidad, se descubrió que, para conseguir una funcionalidad completa, era necesario más trabajo del planteado inicialmente, por lo que se decidió hacer una implementación parcial, que más adelante se puliría para conseguir una funcionalidad completa.

Llevar a cabo esta tarea requería llevar a cabo un trabajo, tanto en el *front end*, como en el *back end*, aunque la primera versión de la implementación solo tuvo en cuenta al primero.

En el cliente, y como parte de la primera iteración de esta funcionalidad, se llevaron a cabo varias tareas. En primer lugar, se creó un componente modal en el que se incrustaría, posteriormente, el cliente VNC.

A continuación, se hace uso del módulo “*@novnc*”, que incluye las funcionalidades requeridas para hacer funcionar un cliente VNC, para establecer la conexión con la máquina virtual, siempre que esta esté encendida. El resultado se puede comprobar en la Ilustración 11.

Escritorio remoto



Ilustración 11: Modal del escritorio remoto

Un detalle de esta versión, y el que hace que esté incompleta es la necesidad de hacer uso de la herramienta *websockify*, que nos permite “traducir tráfico normal a tráfico de sockets...” [74]. Esto es necesario para poder establecer la conexión con el cliente, por lo que el uso de la herramienta es obligatorio. Sin embargo, su puesta en funcionamiento requiere ejecutar un comando en el host de virtualización, y esto no es posible desde el propio cliente, por lo que se decidió parar el desarrollo aquí hasta dar con la solución.

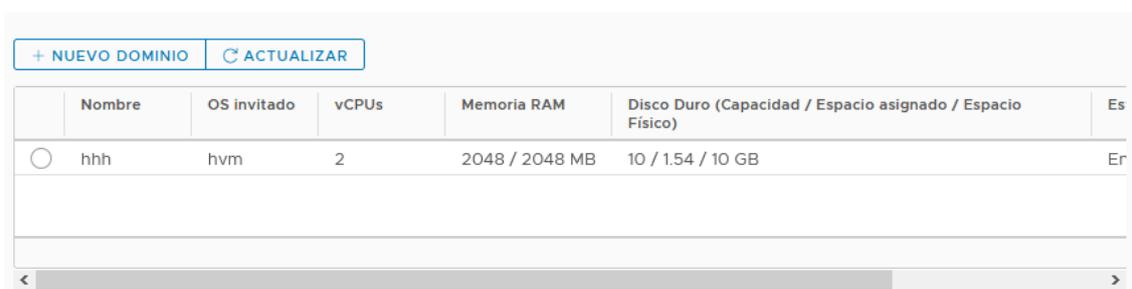
Más adelante, se llegó a una solución. El cliente enviaría una petición al servidor, solicitándole que ejecutar dicho comando, e indicándole ciertas opciones. El servidor entonces ejecutaría dicho comando, y guardaría el *PID*, o “*process ID*” es “un número usado por el *kernel* de la mayoría de los sistemas operativos para identificar de forma unívoca un proceso activo” [75]. Así, una vez se desee cerrar el cliente VNC, el *front end* enviará una petición al *back end*, indicándole esto, y este usará el *PID* para parar el proceso.

9.1.3. Mostrar información de discos duros

Esta tarea requiere editar el código ya existente, que permite al servidor obtener la información de todos los dominios de virtualización definidos en el equipo. En particular, además de la información ya obtenida hasta ahora, como el estado (activo/inactivo), la memoria RAM, o el número de vCPUs, se enviará información sobre el disco duro, siendo esta la siguiente [76]:

- Capacidad: Tamaño lógico en bytes del disco (la capacidad que percibirá el sistema operativo invitado).
- Asignación: Cuánto espacio de almacenamiento ocupa el disco.
- Físico: Espacio físico en bytes del fichero que representa el disco.

Esto se hizo a través de la API de *libvirt*, obteniendo en primer lugar la localización del disco asignado a una máquina virtual, y obteniendo esta información sobre dicho disco, haciendo uso de una función que lleva a cabo esta tarea. El resultado se puede comprobar en la Ilustración 12.



	Nombre	OS invitado	vCPUs	Memoria RAM	Disco Duro (Capacidad / Espacio asignado / Espacio Físico)	Es
<input type="radio"/>	hhh	hvm	2	2048 / 2048 MB	10 / 1.54 / 10 GB	Er

Ilustración 12: Sección de dominios con información sobre los discos duros

9.1.4. Permitir editar RAM y vCPUs

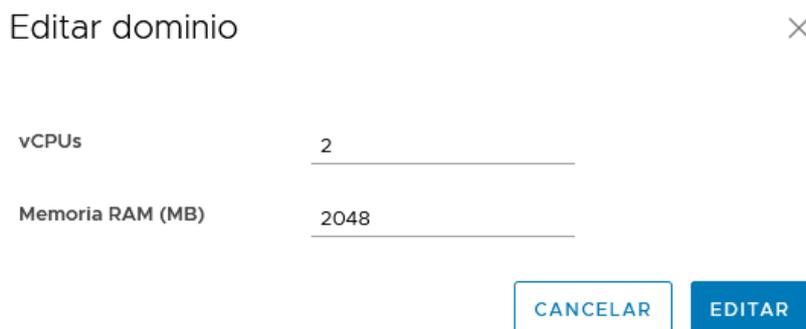
El siguiente paso, fue el de permitir editar la memoria RAM y el número de vCPUs asignados a un dominio de virtualización. Esta tarea requiere aplicar los conocimientos adquiridos hasta ahora para trabajar tanto con el *front end* como en el *back end*.

En este caso, se decidió que lo más eficiente sería empezar por el trabajo en el servidor.

Aquí, fue necesario añadir una nueva operación al *endpoint* de dominios. Esta tarea recibe desde el cliente la información a cambiar, es decir, la nueva cantidad de memoria RAM, además del nuevo número de vCPUs y el identificador del dominio que se desea editar. Con esta información, se hace uso de la API de *libvirt* para cambiar la

configuración de dicho dominio, y se envía una respuesta al cliente informando del éxito o fracaso de la tarea.

Habiendo hecho esto en el servidor, se puede editar el cliente para que haga uso de esta nueva operación. Para esto, se ha creado un modal que permita a un usuario introducir la nueva información que desea incluir, y que envía la petición al servidor. El modal resultante se puede observar en la Ilustración 13



Editar dominio ×

vCPUs	<input type="text" value="2"/>
Memoria RAM (MB)	<input type="text" value="2048"/>

Ilustración 13: Modal de edición de vCPUs y RAM

9.1.5. Mostrar parámetros de configuración

Incluir esta funcionalidad era necesario, pues el anterior autor había dejado ya el componente para ello creado en el cliente. Lo que se hizo fue incluir un nuevo *endpoint* en el servidor, llamado “*settings*”, o configuración. Este cuenta con una única operación que, a petición del cliente, le envía varios parámetros de configuración de la herramienta, como el directorio de definición de los dominios, o el de almacenamiento de los ficheros que representan los discos duros. A continuación, se editó el componente ya creado para que hiciera uso de esta operación, y presentar dicha información al usuario, como se observa en la Ilustración 14.

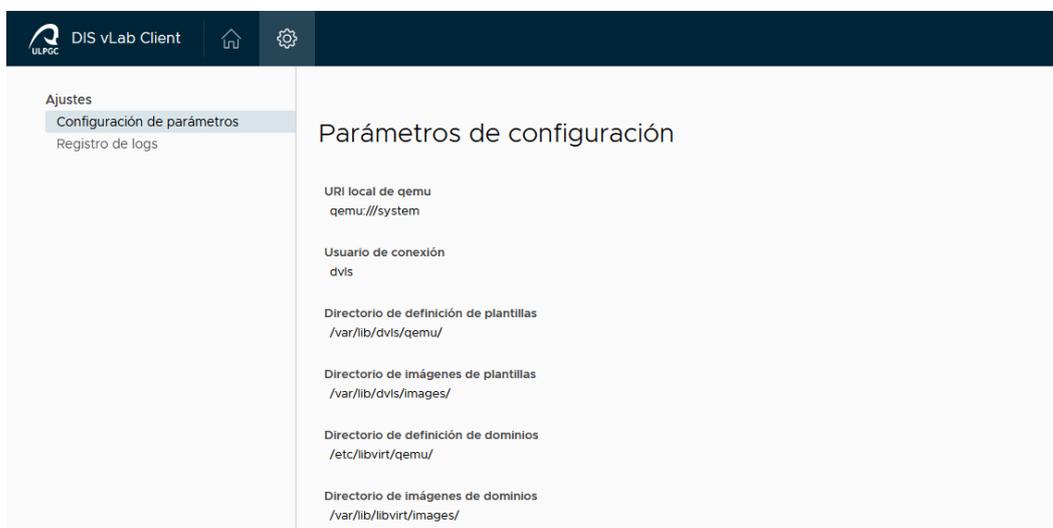


Ilustración 14: Parámetros de configuración de la herramienta

9.1.6. Editar laboratorios

La última historia de usuario implementada fue aquella que permitía editar la información que define un laboratorio.

Siguiendo lo aprendido a la hora de implementar la historia de usuario “Permitir editar RAM y vCPUs”, se decidió empezar a trabajar en el *back end*.

Así, se procedió a añadir una operación nueva, llamada “*update_lab*”, al *endpoint* que se encargaba de los laboratorios. Esta operación recibe, del cliente, toda la nueva información de un laboratorio, y se encarga de actualizar la base de datos con esta información.

A continuación, se pasó a trabajar con el *front end*. En primer lugar, se creó un componente modal, que permite al usuario editar cualquiera de los campos expuestos, como puede ser el nombre del laboratorio, el rango de direcciones IP que lo componen, o la descripción de este. Este componente es también el encargado de enviar dicha información al servidor, cuando el usuario confirma los cambios. Todo esto se puede visualizar en la Ilustración 15.

Editar laboratorio

×

Nuevo código del laboratorio:	LAB_01
Nueva descripción:	Lab de prueba
Nuevo inicio del rango IP:	192.168.1.43
Nuevo final del rango IP:	192.168.1.43
# vCPUs por host:	4 ▲▼
Memoria RAM por host (GB):	8196 ▲▼
Cantida de disco por host (GB):	200 ▲▼

CANCELAR

ACTUALIZAR

Ilustración 15: Modal de edición de laboratorios

9.1.7. Implementación de Celery

La implementación de *Celery*, al contrario que las historias de usuario, requiere editar la estructura básica de la herramienta, es decir, no se puede añadir *Celery* “simplemente” añadiendo nuevos componentes, sino que se debe trabajar a un nivel interno en la aplicación.

El trabajo que conlleva esta tarea se encuentra, principalmente, en el *back end*. Y antes de decidir implementarlos, conlleva refrescar qué se consigue con *Celery*.

Celery permite que cierto código se ejecute en otro hilo, e incluso en otro equipo. Así, la herramienta no se quedará en espera cuando se tenga que ejecutar tareas largas, como puede ser el despliegue de una plantilla en un laboratorio.

También se debe tener en cuenta que existen casos en los que no interesa usar esta herramienta. Un ejemplo serían las tareas que requieren de una respuesta para seguir funcionando, como la carga del *dashboard*, que debe esperar a que se obtenga la información necesaria para continuar su curso.

Además, la estructura cliente-servidor sigue una estructura que, generalmente, funciona de la siguiente forma:

1. El cliente envía una petición al servidor.
2. El servidor ejecuta el código asignado a dicha petición.
3. El servidor responde al cliente.

Es importante mantener esto presente, ya que este esquema se debe mantener.

Por otro lado, las tareas más rápidas pueden ver su ejecución ralentizada por el uso de *Celery*.

Así que se debe llevar a cabo un análisis preliminar sobre qué tareas interesa implementar con Celery. Estas tareas deben ser de larga duración, y la ejecución de otras tareas no debe depender del estado de estas, así que los mejores candidatos serían los referenciados en la

Tabla 3.

Nombre de tarea	Fichero de la carpeta “endpoints”
Generar un dominio	<i>Domains</i>
Clonar un dominio a plantilla	<i>Templates</i>
Desplegar una plantilla en un laboratorio	<i>Templates</i>

Tabla 3: Tareas a implementar en Celery

Sabiendo esto, toca estudiar cuál es la manera más eficiente de implementar *Celery* en el proyecto.

Como el servidor siempre debe responder a las peticiones del cliente con una respuesta, se ha decidido que la mejor forma de llevar a cabo la implementación es siguiendo este esquema:

1. El cliente envía una petición al servidor.
2. El servidor recibe dicha petición, y ordena crear la tarea de *Celery* asignada a dicha petición.
3. El servidor responderá al cliente, especificando el *ID* de la tarea, y *Celery* permanecerá ejecutando la tarea en segundo plano.

Para conseguir esto, se tuvieron que llevar a cabo los siguientes cambios.

En primer lugar, se creó un nuevo directorio llamado “*tasks*”, bajo la carpeta “*api*”, en el que se almacenará el código de las distintas tareas de *Celery*, Ilustración 16.

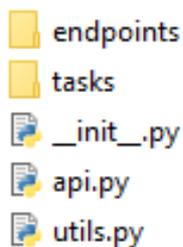


Ilustración 16: Cambios en las carpetas al añadir Celery

En la carpeta “*tasks*”, crearemos los ficheros necesarios para alojar el código de las tareas, siguiendo el esquema del fichero “*endpoints*”, por lo que se generarán dos archivos: “*domains*” y “*templates*”.

Una vez hecho esto, se copiará el código de las tareas especificadas en la

Tabla 3, a sus ficheros correspondientes en la carpeta “*tasks*”, y se editarán las funciones presentes en los ficheros originales para que sigan el comportamiento deseado, es decir, creen la tarea de *Celery* correspondiente y respondan al cliente con el *ID* de dicha tarea.

En principio, esta implementación básica debería bastar, aunque aún queda un tema por tratar: ¿Cómo sabe el cliente que una tarea ha acabado?

Hasta ahora, se sabía que una tarea había acabado porque el servidor respondía al suceder esto. Como esto ya no es así, se genera un problema.

Tras una ligera investigación, se descubrió que el uso de *WebSockets* es la solución perfecta, pues permite una comunicación bidireccional, es decir, tanto el cliente como el servidor pueden enviar mensajes, y no como ahora, que las peticiones siempre las envía el cliente, y el servidor solo puede enviar respuestas.

Sin embargo, como la implementación de *WebSockets* conlleva un esfuerzo considerable, y para poder tener una implementación de *Celery* funcional, se logró una solución alternativa. Cada cierto tiempo, el cliente preguntaría al servidor por el estado de la tarea de *Celery*, haciendo uso del *ID* que el servidor envió como respuesta.

Además de lo ya mencionado, *Celery* también requiere la ejecución, en segundo plano de un “trabajador”, que se encarga de tareas y llevarlas a cabo. En principio, este se ponía en funcionamiento mediante la ejecución de un comando en un terminal, aunque más adelante se consiguió hacer que este se pusiera en funcionamiento y se parase junto a la herramienta.

Aunque esta solución no sea la más eficiente, dado que se pretende implementar *WebSockets* a continuación, se decidió dar esta tarea por acabada.

9.1.8. Implementación de *WebSockets*

La última funcionalidad por añadir, el uso de *WebSockets*, permitirá a la herramienta mejorar, en gran medida, la comunicación entre sus componentes. Al igual que con *Celery*, esto requerirá editar partes de la estructura básica de la herramienta.

Antes de adentrarse más en la implementación, conviene recordar qué hace *WebSockets*.

En esencia, *WebSockets* permite establecer un canal de comunicación bidireccional a través de un puerto *TCP*. Esto permite que dos aplicaciones se conecten a dicho puerto, envíen y reciban mensajes.

En este caso, *WebSockets* permitirá que el servidor comunique al cliente cuándo ha acabado una tarea larga. Dependiendo de la página cargada en ese momento, el cliente podrá llevar a cabo distintas acciones.

La implementación de *WebSockets* usada en el servidor y en el cliente será *Socket.IO*, pues cuenta con implementaciones tanto para *Python*, como para *JavaScript*.

En primer lugar, se debe “encapsular” la ejecución de la herramienta, para que soporte el uso de comunicación *WebSockets*. Esto se puede hacer automáticamente por la propia librería.

Tras esto, ya se puede implementar el uso de esta herramienta en el proyecto. Esto se puede hacer mediante dos operaciones básicas.

La primera operación sería la de emitir un mensaje con una información, lo cual se hace ejecutando la función “*emit*”, e indicando la información que se quiere enviar en formato *JSON*.

Esta operación envía dicho mensaje a todas las aplicaciones conectadas al mismo puerto *TCP* que esta.

En el caso del servidor, todos los mensajes enviados tienen el nombre “*task-finished*”, y se envía al acabar la ejecución de las tareas largas. En el propio *JSON* se especifica el tipo de tarea cuya ejecución ya ha terminado.

La otra operación es la de recibir un mensaje. El modo en el que esto funciona es estableciendo un “evento” cuya ejecución se espera. Cuando otra aplicación envíe un

mensaje con el mismo nombre que el evento que hemos indicado, se ejecutará el código especificado.

En el caso del servidor, esto solo sucede cuando se produce una conexión o desconexión del puerto *TCP* usado por *WebSockets*. Este mensaje se envía de forma automática cuando un cliente se conecta y desconecta.

En el caso del cliente, no es necesario cambiar la configuración básica. Simplemente se debe instalar un paquete que incluya el cliente de *Socket.IO*, y ya se podrán crear las operaciones básicas mencionadas.

En el caso del cliente, no es necesario enviar ningún mensaje, pero sí que se recibirá uno, el de *"task-finished"*. Una vez se recibe, se analiza el tipo de tarea que ha terminado, y se actúa en función a esto.

Una vez acabada la implementación de *WebSockets*, se puede comprobar que la comunicación entre los componentes de la aplicación es bastante más eficiente, pues no es necesario enviar un mensaje cada cierto tiempo para saber el estado de las tareas largas, sino que el propio servidor es capaz de avisar al cliente cuando la ejecución de estas termina.

En este punto, se han completado los objetivos propuestos para este proyecto.

10. Resultados y conclusiones

Tras el desarrollo de este trabajo, se puede concluir que los objetivos propuestos al inicio han sido cumplidos. El usuario de la herramienta, además de las tareas originales, ahora es capaz de:

- Editar la información de los laboratorios.
- Ver el listado de *hosts* de un laboratorio.
- Editar tanto la RAM como las vCPUs de un dominio.
- Conectarse mediante un cliente VNC incrustado a un dominio encendido.
- Consultar los parámetros de configuración.

El funcionamiento de las historias de usuario implementadas puede comprobarse en el Anexo III.

Por otro lado, la implementación de *Celery* y *WebSockets* ha permitido mejorar la comunicación entre los componentes del proyecto, además de mejorar el rendimiento, ejecutando las tareas más largas en segundo plano.

Para asegurar el funcionamiento de la herramienta en un entorno de producción, se llevó a cabo un análisis de requisitos, para confirmar que esta funciona en el entorno para el que fue diseñada.

El desarrollo de un proyecto de este tipo, en el que se debe continuar el trabajo de otro desarrollador, ha sido especialmente útil para el autor de este trabajo, pues se han manifestado ciertos obstáculos que nacen en desarrollos de este tipo, como errores de versiones, o incompatibilidad de ciertos componentes que están en continua evolución.

Además, la puesta en funcionamiento del proyecto requiere llevar a cabo un trabajo de análisis, propio de las Tecnologías de la Información, que ha cimentado los conocimientos adquiridos en varias asignaturas del Grado.

Así, se puede concluir que la finalización satisfactoria de este trabajo no solo ha permitido aumentar las capacidades del original y, potencialmente, facilitar el trabajo del personal de CCDIS, sino que ha armado al alumno de las herramientas y conocimientos necesarios para poder enfrentarse, en un entorno laboral, a las dificultades inherentes al análisis, desarrollo y puesta en funcionamiento del software.

11. Trabajos futuros

Si bien este trabajo nace como ampliación de otro, y aunque se han cumplido los objetivos propuestos al inicio del desarrollo, existen varias posibilidades de expansión que pueden servir como líneas de trabajo en proyectos futuros.

11.1. Prueba en un entorno real

La puesta en funcionamiento de la herramienta en un servidor de la universidad, además del despliegue de máquinas virtuales en laboratorios reales es una de las posibilidades más obvias.

Comprobar el correcto funcionamiento de la herramienta en el entorno para el que está destinada permitiría dar validez a este trabajo y a su antecesor.

11.2. Configuración de clientes

Con el funcionamiento actual, el usuario de una máquina cliente debe iniciar sesión en CentOS, y luego desde dicho sistema operativo debe hacer uso de la herramienta “*virt-manager*”, para encender la máquina virtual y conectarse a esta.

Una línea de trabajo interesante podría ser la configuración de los clientes, tal vez de forma centralizada desde la propia herramienta, de forma que se automaticen estas tareas. Así, un usuario solo tendría que elegir al iniciar la máquina en qué sistema quiere iniciar sesión.

Existen varias maneras de llevar esto a cabo, por ejemplo, se podrían configurar usuarios específicos para cada máquina que, al iniciar sesión, la enciendan y se conecten al escritorio de esta. O tal vez se podría desarrollar un script que, al inicio, pregunte al usuario a qué máquina quiere conectarse.

11.3. Mejoras en el despliegue

Como se puede apreciar en el Anexo I, la instalación de la herramienta en el servidor requiere llevar a cabo una serie de tareas que podrían ser automatizadas. Un buen trabajo podría ser el desarrollo de, por ejemplo, una serie de scripts que permitan automatizar el despliegue.

Otra posibilidad es la de la “*containerización*”, lo cual permitiría generar “*contenedores*” con todo lo necesario para ejecutar la herramienta. Esto ya se contemplaba en la anterior iteración de este proyecto [1].

Anexo I: Instalación de la herramienta

Con los añadidos hechos a la herramienta, el proceso de instalación ha variado, por lo que se pasa a dar instrucciones para poner en funcionamiento la herramienta.

Paquetes necesarios

La instalación debe llevarse a cabo sobre un equipo con CentOS 7, actualizado.

En primer lugar, se deben instalar los repositorios EPEL e IUS, usando los siguientes comandos:

```
# yum install https://dl.fedoraproject.org/pub/epel/7/x86\_64/
# yum install \
https://repo.ius.io/ius-release-el7.rpm \
https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

Se deben instalar los siguientes paquetes:

- “Virtualization Platform” (grupo).
- “Virtualization Hypervisor” (grupo).
- “Virtualization Tools” (grupo).
- “Virtualization Client” (grupo).
- “Development” (grupo).
- “libvirt-devel.x86-64”.
- “libguestfs-tools”.
- “python36u”.
- “python36u-devel”.
- “python36u-pip”.
- “nginx”.
- “openssl”.
- “erlang”.
- “socat”.

Una vez instalados los paquetes, es necesario reiniciar el equipo.

El código de la aplicación debe encontrarse bajo el directorio “*/usr/lib/dvls*”, por lo que se debe descargar el código del repositorio ahí:

```
# cd /usr/lib
# git clone https://github.com/Acova/DIS-vLab-Server-2\_0
# mv DIS-vLab-Server-2_0 dvls
```

Es necesario el uso del paquete “*virtualenv*” de *Python* para usar la herramienta, por lo que se descargará y se preparará el entorno:

```
# pip3.6 install virtualenv
# cd dvls
# virtualenv venv
# source venv/bin/actíivate
# pip install -r requirements.txt
# deactivate
```

Configuración de *Celery*

En primer lugar, se debe instalar *RabbitMQ* e iniciar el servicio:

```
# wget https://www.rabbitmq.com/releases/rabbitmq-server/v3.6.10/rabbitmq-server-3.6.10-1.el7.noarch.rpm
# rpm --import https://www.rabbitmq.com/rabbitmq-release-signing-key.asc
# rpm -Uvh rabbitmq-server-3.6.10-1.el7.noarch.rpm
# service start rabbitmq-server
```

A continuación, se debe crear un servicio para *Celery*. Para esto, es necesario crear un fichero bajo el directorio “*/etc/systemd/system*”, por ejemplo, “*celery.service*”. En este fichero se debe escribir lo siguiente:

```
[Unit]
Description=Celery Service
After=network.target

[Service]
WorkingDirectory=/usr/lib/dvls
ExecStart=/bin/sh -c '/usr/lib/dvls/venv/bin/celery --app=app.core.celery worker -f /var/log/dvls-worker.log -l DEBUG -E'
```

Y ya se puede iniciar el servicio creado:

```
# systemctl start celery.service
```

Configuración del sistema

A continuación, se deben llevar a cabo varias configuraciones en el sistema.

En primer lugar, se debe crear un usuario, no-root, que será el encargado de acceder a la aplicación y con el que se ejecutarán las órdenes.

```
# useradd dvls
# passwd dvls
# usermod -a -G libvirt dvls
```

Además, se deben añadir varias normas al Firewall para permitir que todo funcione correctamente.

```
# firewall-cmd --add-service=http --permanent
# firewall-cmd --add-service=https --permanent
# firewall-cmd --add-port=5900-5910/tcp --permanent
# firewall-cmd --add-port=6080/tcp --permanent
# firewall-cmd --reload
```

Por otro lado, hay que configurar PolicyKit, para permitir que un usuario no root lleve a cabo ciertas acciones. En primer lugar, se debe crear el fichero `"/usr/share/polkit-1/rules.d/50-libvirt.rules"`. Este fichero debe contener lo siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policyconfig PUBLIC
"-//freedesktop//DTD PolicyKit Policy Configuration 1.0//EN"
"http://www.freedesktop.org/standards/PolicyKit/1.0/policyconfig.dtd">
<policyconfig>
  <vendor>DVLS</vendor>
  <vendor_url>https://www.dis.ulpgc.es/</vendor_url>

  <action id="es.ulpgc.dis.dvls.virt-sysprep">
    <description>Run 'virt-sysprep'</description>
    <message>Authentication is required to descontextualize domain disks as
non-root user</message>
    <defaults>
      <allow_inactive>no</allow_inactive>
      <allow_active>no</allow_active>
    </defaults>
    <annotate
      key="org.freedesktop.policykit.exec.path">/usr/bin/virtsysprep</annotate>
    </action>
</policyconfig>
```

También se debe crear el fichero `"/etc/polkit-1/rules.d/60-virt-sysprep.rules"`, con el siguiente texto:

```
polkit.addRule(function(action, subject){
    if (action.id == "es.ulpgc.dis.dvls.virt-sysprep" &&
        subject.isInGroup("libvirt")) {
        return polkit.Result.YES;
    }
});
```

Para la correcta autenticación del usuario en la aplicación, es necesario ejecutar el siguiente comando:

```
# echo "auth required pam_unix.so" > /etc/pam.d/dvls
```

Por último, será necesario crear los directorios `"/var/lib/dvls"`, `"/var/lib/dvls/images"`, y `"/var/lib/qemu"`. El directorio `"/var/lib/dvls"` debe ser compartido mediante NFS.

Servicio DVLS

Es necesario crear un servicio que se encargue de iniciar el servidor DVLS. Para esto, se debe crear un archivo en el directorio `"/etc/systemd/system"`, por ejemplo, `dvls.service`, con el siguiente contenido:

```
[Unit]
Description=uWSGI instance for DIS vLab Server
After=network.target

[Service]
Type=simple
WorkingDirectory=/usr/lib/dvls
ExecStart=/usr/lib/dvls/venv/bin/python3.6 /usr/lib/dvls/wsgi.py

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

Además, se debe cambiar el dueño de la carpeta `"/usr/lib/dvls"`:

```
# chown dvls:dvls /usr/lib/dvls
```

Configuración de Nginx

En primer lugar, es necesario crear un certificado autofirmado con OpenSSL, para poder hacer uso de HTTPS, con las mejoras en seguridad que eso conlleva. Si se cuenta con unos, este paso puede saltarse.

En primer lugar, es necesario asegurarse de que el directorio `"/etc/nginx/ssl"` existe.

A continuación, se puede generar un certificado, con su clave con el siguiente comando:

```
# openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -keyout /etc/nginx/ssl/nginx.key -  
out /etc/nginx/ssl/nginx.crt
```

Seguidamente, debemos configurar Nginx como un proxy inverso, que se encargará de transformar el tráfico HTTPS en WebSockets y viceversa. En prime lugar, se debe crear el fichero `"/etc/nginx/conf.d/dvls.conf"`, con el siguiente contenido:

```
server {  
    listen 443 ssl;  
    server_name dvls.dis.ulpgc.es;  
    ssl_certificate /etc/nginx/ssl/nginx.crt;  
    ssl_certificate_key /etc/nginx/ssl/nginx.key;  
  
    location / {  
        include proxy_params;  
        proxy_pass http://127.0.0.1:5000;  
    }  
  
    location /static {  
        alias /usr/lib/dvls/static;  
        expires 30d;  
    }  
  
    location /socket.io {  
        include proxy_params;  
        proxy_http_version 1.1;  
        proxy_buffering off;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection "Upgrade";  
        proxy_pass http://127.0.0.1:5000/socket.io;  
    }  
}
```

En ciertas instalaciones, es posible que el sistema no cuente con el fichero `"/etc/nginx/proxy_params"`. En esos casos se debe crear y añadirle lo siguiente:

```
proxy_set_header Host $http_host;  
proxy_set_header X-Real-IP $remote_addr;  
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
proxy_set_header X-Forwarded-Proto $scheme;
```

Con estas configuraciones, ya se debería poder acceder a la interfaz web. En caso de que no sea posible, consultar la siguiente sección.

SSH

Para el despliegue de las máquinas, es necesario generar una clave RSA sin contraseña, que se usará para la conexión mediante SSH con los clientes.

En primer lugar, es necesario asegurarse de que existe el directorio `"/home/dvls/.ssh"`, y se debe cambiar los permisos de acceso a este:

```
# chmod 0700 /home/dvls/.ssh
```

A continuación, se debe generar una pareja de claves público/privadas:

```
# ssh-keygen -t rsa
```

Durante la ejecución del comando anterior, es importante dejar las opciones por defecto.

Problemas

Es posible que, acabada la instalación, un usuario se encuentre con algunos problemas.

En primer lugar, es posible que al intentar acceder a la interfaz web, se obtenga un mensaje de error `"502"`. Esto se debe a las políticas de SELinux. Para solucionarlo es necesario instalar el paquete `"policycoreutils-python"`, y ejecutar ciertos comandos que permitan un funcionamiento correcto.

```
# yum install policycoreutils-python  
# grep nginx /var/log/audit/audit.log | audit2allow -M nginx  
# semodule -i nginx.pp
```

Además, si el certificado ha sido autofirmado, el navegador no lo reconocerá como de una autoridad de confianza, y el cliente VNC no funcionará. Para solucionarlo, tras intentar acceder al cliente, se debe visitar el sitio `"https://localhost:6080"`, y añadir una excepción de seguridad al navegador. Por razones de funcionamiento, este arreglo solo funciona cuando se accede desde el propio equipo, por lo que si no se cuenta con certificados de una autoridad de confianza no se podrá usar el cliente VNC desde fuera.

Anexo II: Configuración de los clientes

Los clientes a los que se desplegarán las máquinas virtuales necesitan cierta configuración.

Además de tener una instalación de CentOS 7 mínima, se les deben instalar los siguientes paquetes:

- “qemu-virt”
- “libvirt”
- “virt-install”
- “ntfs-utils”

Una vez se han instalado los paquetes, conviene reiniciar el equipo.

Se debe crear el directorio “/var/lib/dvls”, donde se deberá montar el directorio NFS compartido por el servidor.

Por último, desde el servidor se debe compartir la clave ssh pública a cada uno de los clientes. Así, desde el servidor, se ejecutará el siguiente comando:

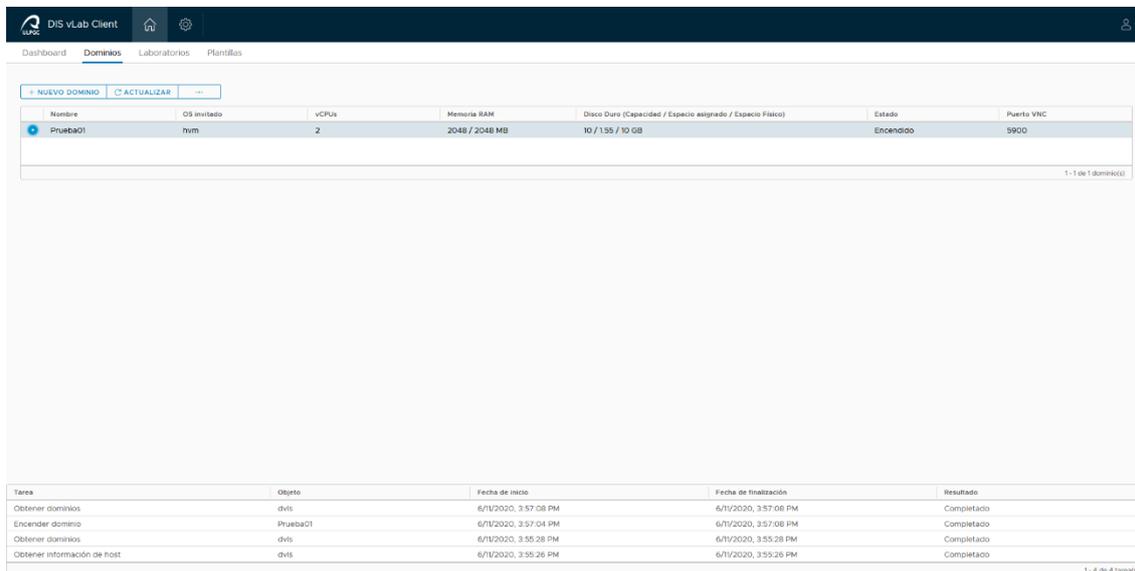
```
# ssh-copy-id -i /home/dvls/.ssh/id_rsa.pub root@<ip_cliente>
```

Anexo III: Manual de usuario

A continuación, se pasará a dar un breve repaso en el uso de las funcionalidades añadidas.

Conexión VNC

Para llevar a cabo una conexión de escritorio remoto con un dominio, en primer lugar, se debe seleccionar uno que esté encendido, como se muestra en la Ilustración 17.

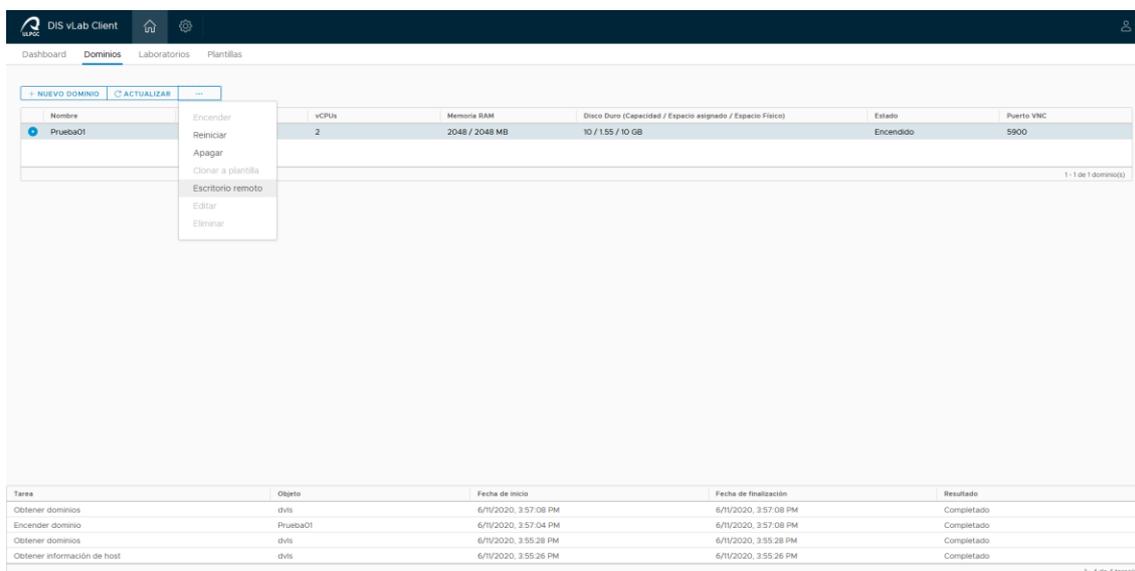


Nombre	OS invitado	vCPUs	Memoria RAM	Disco Duro (Capacidad / Espacio asignado / Espacio Físico)	Estado	Puerto VNC
Prueba01	hvm	2	2048 / 2048 MB	10 / 155 / 10 GB	Encendido	5900

Tarea	Objeto	Fecha de inicio	Fecha de finalización	Resultado
Obtener dominios	dvfs	6/1/2020, 3:57:08 PM	6/1/2020, 3:57:08 PM	Completado
Encender dominio	Prueba01	6/1/2020, 3:57:04 PM	6/1/2020, 3:57:08 PM	Completado
Obtener dominios	dvfs	6/1/2020, 3:55:28 PM	6/1/2020, 3:55:28 PM	Completado
Obtener información de host	dvfs	6/1/2020, 3:55:26 PM	6/1/2020, 3:55:26 PM	Completado

Ilustración 17: Selección de un dominio en ejecución

A continuación, en el submenú desplegable a la derecha de “Actualizar”, seleccionamos “Escritorio remoto”, como en la Ilustración 18.



Nombre	OS invitado	vCPUs	Memoria RAM	Disco Duro (Capacidad / Espacio asignado / Espacio Físico)	Estado	Puerto VNC
Prueba01	hvm	2	2048 / 2048 MB	10 / 155 / 10 GB	Encendido	5900

Tarea	Objeto	Fecha de inicio	Fecha de finalización	Resultado
Obtener dominios	dvfs	6/1/2020, 3:57:08 PM	6/1/2020, 3:57:08 PM	Completado
Encender dominio	Prueba01	6/1/2020, 3:57:04 PM	6/1/2020, 3:57:08 PM	Completado
Obtener dominios	dvfs	6/1/2020, 3:55:28 PM	6/1/2020, 3:55:28 PM	Completado
Obtener información de host	dvfs	6/1/2020, 3:55:26 PM	6/1/2020, 3:55:26 PM	Completado

Ilustración 18: Opción de “Escritorio Remoto”

Esto nos abrirá un modal como el de la Ilustración 19, que solicita la contraseña configurada durante la creación del dominio.

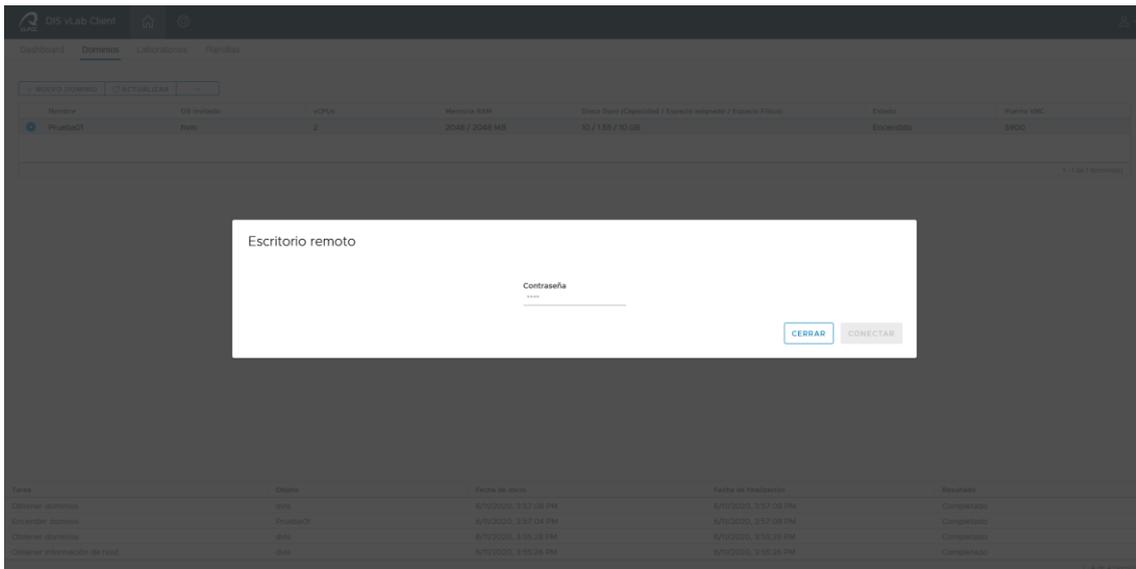


Ilustración 19: Modal del escritorio remoto solicitando la contraseña

Una vez introducida, se debe pulsar conectar, y se cargará el escritorio remoto. Se puede encontrar un ejemplo de funcionamiento en la Ilustración 20



Ilustración 20: Escritorio remoto en funcionamiento

Editar información de un dominio

Para poder editar la información de un dominio, este debe estar apagado. Una vez se haya seleccionado, se debe pulsar en “Editar”, como en la Ilustración 21.

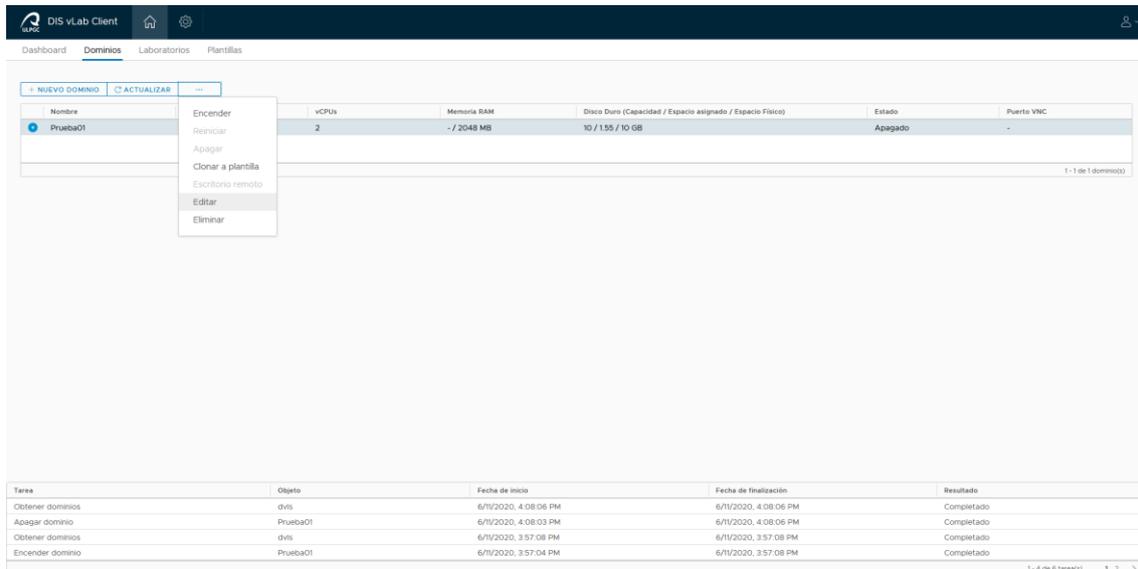


Ilustración 21: Selección de la opción "Editar" de un dominio

Una vez pulsado, se abrirá un modal que permitirá insertar los nuevos valores para las vCPUs y la memoria RAM, como el de la Ilustración 22.

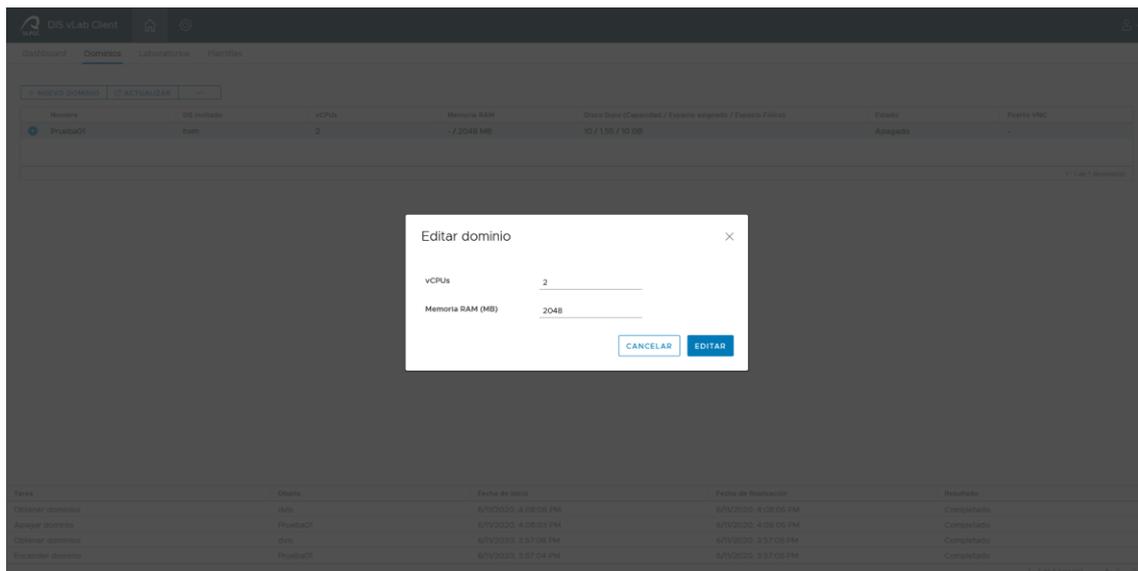


Ilustración 22: Modal de edición de RAM y vCPUs

Una vez añadidos los valores nuevos, se debe pulsar en “Editar” para confirmar los cambios.

Ver *hosts* de un laboratorio

Ver los *hosts* de un laboratorio se consigue seleccionándolo y, pulsando en la opción “Ver *hosts*” del submenú, como en la Ilustración 23

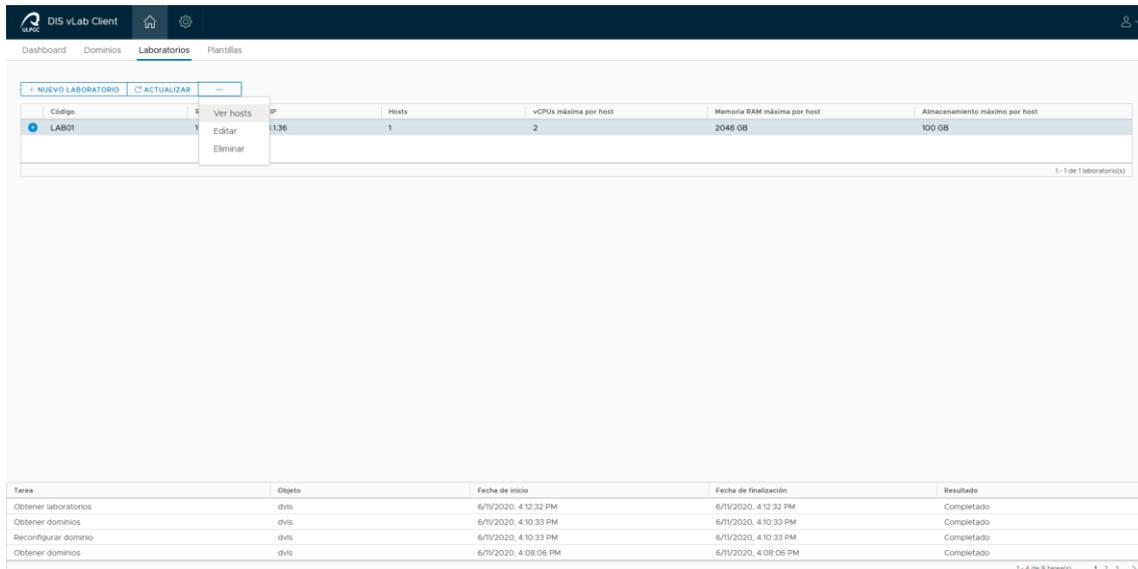


Ilustración 23: Selección de la opción “Ver *hosts*” de un laboratorio

Una vez pulsado, se abrirá el modal que mostrará la información de estos *hosts*. Como es de la Ilustración 24.

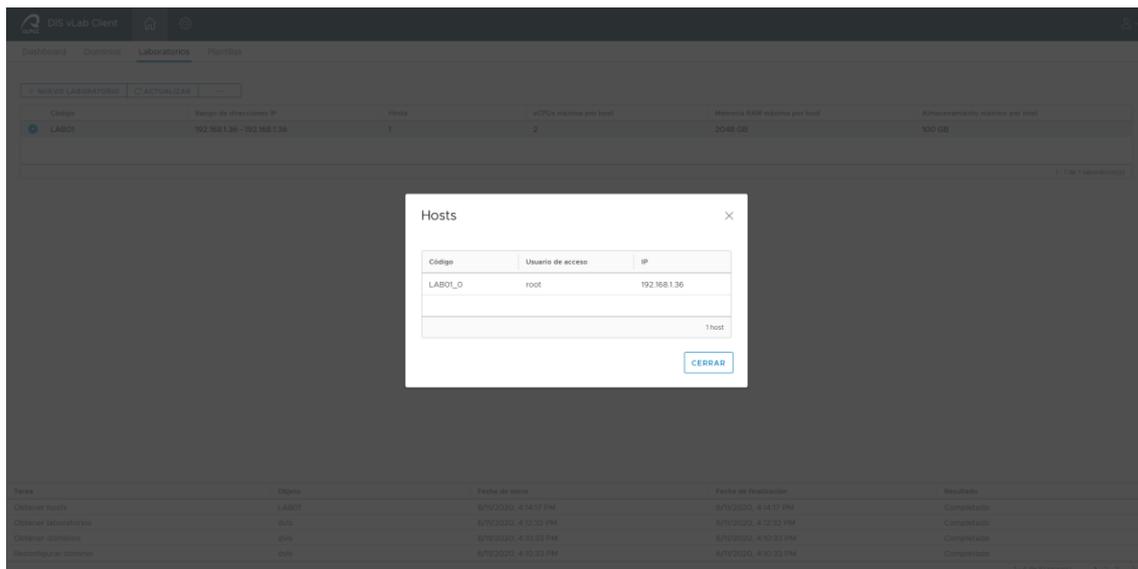
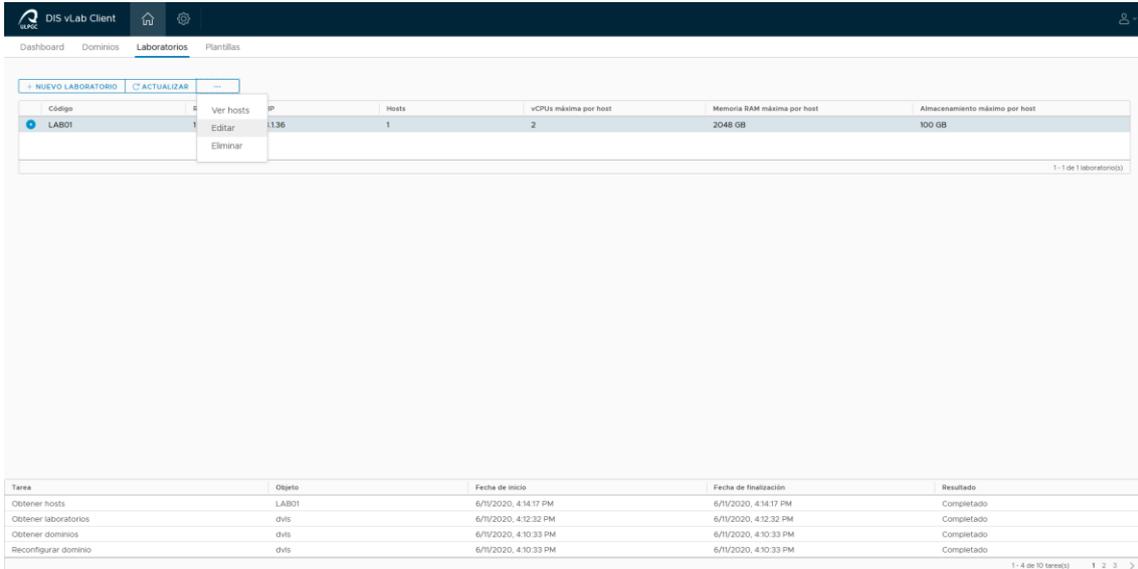


Ilustración 24: Modal de información de *hosts* de un laboratorio

Edición de un laboratorio

Para editar un laboratorio, después de seleccionarlo, se debe pulsar en la opción “Editar”, del submenú, como en la Ilustración 25.

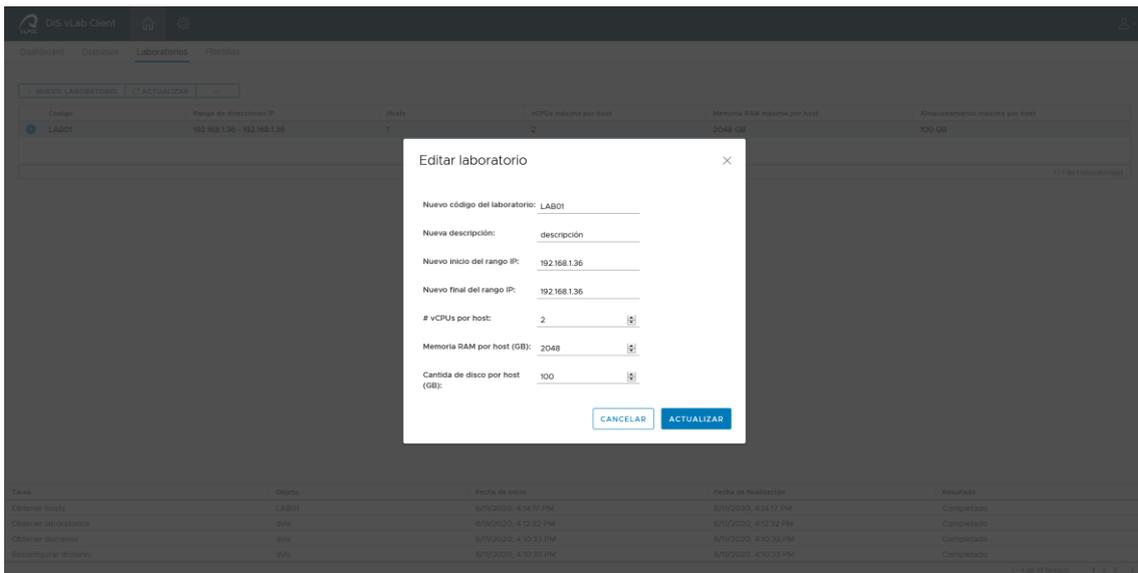


Código	Ver hosts	IP	Hosts	vCPUs máxima por host	Memoria RAM máxima por host	Almacenamiento máximo por host
LAB01	1	136	1	2	2048 GB	100 GB

Tarea	Objeto	Fecha de inicio	Fecha de finalización	Resultado
Obtener hosts	LAB01	6/11/2020, 4:14:17 PM	6/11/2020, 4:14:17 PM	Completado
Obtener laboratorios	dvfs	6/11/2020, 4:12:32 PM	6/11/2020, 4:12:32 PM	Completado
Obtener dominios	dvfs	6/11/2020, 4:10:33 PM	6/11/2020, 4:10:33 PM	Completado
Reconfigurar dominio	dvfs	6/11/2020, 4:10:33 PM	6/11/2020, 4:10:33 PM	Completado

Ilustración 25: Selección de la opción "Editar" de un laboratorio

Una vez seleccionado, se abrirá el modal de edición de los laboratorios. Se deben introducir los datos que se deseen, y pulsar en “Actualizar”, como se muestra en la Ilustración 26.



Editar laboratorio

Nuevo código del laboratorio: LAB01

Nueva descripción: descripción

Nuevo inicio del rango IP: 192.168.1.36

Nuevo final del rango IP: 192.168.1.36

vCPUs por host: 2

Memoria RAM por host (GB): 2048

Cantidad de disco por host (GB): 100

Ilustración 26: Modal de edición de laboratorios

Bibliografía

- [1] A. S. García, «Automatización del despliegue de máquinas virtuales para los laboratorios de la Escuela de Ingeniería Informática | accedaCRIS,» [En línea]. Available: <http://hdl.handle.net/10553/55557>. [Último acceso: 21 Octubre 2020].
- [2] «Virtualización - Wikipedia, la enciclopedia libre,» 18 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/Virtualizaci%C3%B3n>. [Último acceso: 22 Octubre 2020].
- [3] Neosoft, «¿Qué es una aplicación Web? – Blog Neosoft Sistemas,» 2 Enero 2018. [En línea]. Available: <https://www.neosoft.es/blog/que-es-una-aplicacion-web/>. [Último acceso: 22 Octubre 2020].
- [4] P. Stefaniak, «¿Qué es Backend y Frontend? - Descubre Comunicación,» 26 Julio 2019. [En línea]. Available: <https://descubrecomunicacion.com/que-es-backend-y-frontend/>. [Último acceso: 22 Octubre 2020].
- [5] «Representational state transfer - Wikipedia,» 17 Octubre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Último acceso: 22 Octubre 2020].
- [6] «Identificador de recursos uniforme - Wikipedia, la enciclopedia libre,» 6 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme. [Último acceso: 22 Octubre 2020].
- [7] «Protocolo de transferencia de hipertexto - Wikipedia, la enciclopedia libre,» 2 Septiembre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto. [Último acceso: 22 Octubre 2020].
- [8] «Objetivos y Competencias del GII | Web de la Escuela de Ingeniería Informática de la ULPGC,» 26 Julio 2013. [En línea]. Available: https://www.eii.ulpgc.es/tb_university_ex/?q=objtivos-y-competencias-del-gii. [Último acceso: 04 Noviembre 2020].
- [9] «Licencia permisiva - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Licencia_permisiva. [Último acceso: 06 Noviembre 2020].
- [10] «Copyleft - Wikipedia, la enciclopedia libre,» 31 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/Copyleft>. [Último acceso: 06 Noviembre 2020].

- [11] «Licencia MIT - Wikipedia, la enciclopedia libre,» 30 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Licencia_MIT. [Último acceso: 06 Noviembre 2020].
- [12] «Socket.IO - Wikipedia,» 06 Noviembre 2020. [En línea]. Available: <https://en.wikipedia.org/wiki/Socket.IO>. [Último acceso: 06 Noviembre 2020].
- [13] «SQLAlchemy - Wikipedia,» 06 Noviembre 2020. [En línea]. Available: <https://en.wikipedia.org/wiki/SQLAlchemy>. [Último acceso: 06 Noviembre 2020].
- [14] «PyJWT · PyPI,» [En línea]. Available: <https://pypi.org/project/PyJWT/>. [Último acceso: 06 Noviembre 2020].
- [15] «python-pam · PyPI,» [En línea]. Available: <https://pypi.org/project/python-pam/>. [Último acceso: 06 Noviembre 2020].
- [16] «Eventlet Networking Library,» 20 Octubre 2020. [En línea]. Available: <https://eventlet.net/>. [Último acceso: 06 Noviembre 2020].
- [17] «Angular,» 4 Noviembre 2020. [En línea]. Available: <https://angular.io/license>. [Último acceso: 06 Noviembre 2020].
- [18] «clarity/LICENSE at next · vmware/clarity,» 04 Enero 2018. [En línea]. Available: <https://github.com/vmware/clarity/blob/next/LICENSE>. [Último acceso: 06 Noviembre 2020].
- [19] «GNU General Public License - Wikipedia, la enciclopedia libre,» 27 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/GNU_General_Public_License. [Último acceso: 06 Noviembre 2020].
- [20] «CentOS - Wikipedia, la enciclopedia libre,» 25 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/CentOS>. [Último acceso: 26 Octubre 2020].
- [21] «uWSGI - Wikipedia,» 06 Noviembre 2020. [En línea]. Available: <https://en.wikipedia.org/wiki/UWSGI>. [Último acceso: 06 Noviembre 2020].
- [22] «GNU Lesser General Public License - Wikipedia, la enciclopedia libre,» 25 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/GNU_Lesser_General_Public_License. [Último acceso: 06 Noviembre 2020].
- [23] «libvirt - Wikipedia,» 13 Octubre 2020. [En línea]. Available: <https://en.wikipedia.org/wiki/Libvirt>. [Último acceso: 25 Octubre 2020].
- [24] «paramiko · PyPI,» [En línea]. Available: <https://pypi.org/project/paramiko/>. [Último acceso: 06 Noviembre 2020].

- [25] «PolicyKit - Wikipedia, la enciclopedia libre,» 25 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/PolicyKit>. [Último acceso: 06 Noviembre 2020].
- [26] «Flask (web framework) - Wikipedia,» 06 Noviembre 2020. [En línea]. Available: [https://en.wikipedia.org/wiki/Flask_\(web_framework\)](https://en.wikipedia.org/wiki/Flask_(web_framework)). [Último acceso: 06 Noviembre 2020].
- [27] «Celery (software) - Wikipedia,» 06 Noviembre 2020. [En línea]. Available: [https://en.wikipedia.org/wiki/Celery_\(software\)](https://en.wikipedia.org/wiki/Celery_(software)). [Último acceso: 06 Noviembre 2020].
- [28] «Werkzeug · PyPI,» [En línea]. Available: <https://pypi.org/project/Werkzeug/>. [Último acceso: 06 Noviembre 2020].
- [29] «Redis - Wikipedia,» 06 Noviembre 2020. [En línea]. Available: <https://en.wikipedia.org/wiki/Redis>. [Último acceso: 06 Noviembre 2020].
- [30] «Nginx - Wikipedia, la enciclopedia libre,» 31 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/Nginx>. [Último acceso: 06 Noviembre 2020].
- [31] «Python Software Foundation License - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Python_Software_Foundation_License. [Último acceso: 06 Noviembre 2020].
- [32] «History and License — Python 3.6.12 documentation,» 1 Octubre 2020. [En línea]. Available: <https://docs.python.org/3.6/license.html>. [Último acceso: 06 Noviembre 2020].
- [33] «Mozilla Public License - Wikipedia, la enciclopedia libre,» 27 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Mozilla_Public_License. [Último acceso: 06 Noviembre 2020].
- [34] «noVNC,» 14 Julio 2020. [En línea]. Available: <https://novnc.com/info.html>. [Último acceso: 25 Octubre 2020].
- [35] «Desarrollo iterativo y creciente - Wikipedia, la enciclopedia libre,» 21 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente. [Último acceso: 22 Octubre 2020].
- [36] «Angular - Introduction to the Angular Docs,» 21 Mayo 2020. [En línea]. Available: <https://angular.io/docs>. [Último acceso: 23 Octubre 2020].
- [37] «Software framework - Wikipedia,» 24 Octubre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Software_framework. [Último acceso: 23 Octubre 2020].

- [38] «Single-page application - Wikipedia,» 13 Octubre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Single-page_application. [Último acceso: 25 Octubre 2020].
- [39] «Clarity Design System - Documentation - Get Started,» [En línea]. Available: <https://clarity.design/documentation/get-started>. [Último acceso: 25 Octubre 2020].
- [40] «VNC - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/VNC>. [Último acceso: 25 Octubre 2020].
- [41] «General Python FAQ — Python 3.6.12 documentation,» 7 Octubre 2020. [En línea]. Available: <https://docs.python.org/3.6/faq/general.html>. [Último acceso: 25 Octubre 2020].
- [42] «Python - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/Python>. [Último acceso: 25 Octubre 2020].
- [43] J. Pastor, «Python es el BASIC de nuestra era: es el lenguaje de programación de moda según IEEE Spectrum y todos quieren conocerlo,» 27 Julio 2020. [En línea]. Available: <https://www.xataka.com/aplicaciones/python-basic-nuestra-era-lenguaje-programacion-moda-ieee-spectrum-todos-quieren-conocerlo>. [Último acceso: 25 Octubre 2020].
- [44] «Flask | The Pallets Projects,» 17 Diciembre 2019. [En línea]. Available: <https://palletsprojects.com/p/flask/>. [Último acceso: 25 Octubre 2020].
- [45] «Web Server Gateway Interface - Wikipedia,» 13 Octubre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface. [Último acceso: 25 Octubre 2020].
- [46] «SQLite Home Page,» 20 Octubre 2020. [En línea]. Available: <https://www.sqlite.org/index.html>. [Último acceso: 25 Octubre 2020].
- [47] «Sistema de gestión de bases de datos - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos. [Último acceso: 25 Octubre 2020].
- [48] «SQL - Wikipedia, la enciclopedia libre,» 25 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/SQL>. [Último acceso: 25 Octubre 2020].
- [49] «SQLite - Wikipedia, la enciclopedia libre,» 25 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/SQLite>. [Último acceso: 25 Octubre 2020].

- [50] «The uWSGI project — uWSGI 2.0 documentation,» 17 Junio 2020. [En línea]. Available: <https://uwsgi-docs.readthedocs.io/en/latest/>. [Último acceso: 25 Octubre 2020].
- [51] Y. Fernández, «API: qué es y para qué sirve,» 23 Agosto 2019. [En línea]. Available: <https://www.xataka.com/basics/api-que-sirve>. [Último acceso: 25 Octubre 2020].
- [52] «¿Qué Es NGINX Y Cómo Funciona?,» 12 Junio 2020. [En línea]. Available: <https://www.hostinger.es/tutoriales/que-es-nginx/>. [Último acceso: 25 Octubre 2020].
- [53] «Proxy inverso - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Proxy_inverso. [Último acceso: 25 Octubre 2020].
- [54] «Introduction to Celery — Celery 4.4.7 documentation,» 31 Julio 2020. [En línea]. Available: <https://docs.celeryproject.org/en/v4.4.7/getting-started/introduction.html>. [Último acceso: 25 Octubre 2020].
- [55] «WebSocket - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/WebSocket>. [Último acceso: 25 Octubre 2020].
- [56] «Introduction | Socket.IO,» [En línea]. Available: <https://socket.io/docs/>. [Último acceso: 25 Octubre 2020].
- [57] «Hypervisor - Wikipedia,» 16 Octubre 2020. [En línea]. Available: <https://en.wikipedia.org/wiki/Hypervisor>. [Último acceso: 25 Octubre 2020].
- [58] «Hardware virtualization - Wikipedia,» 16 Octubre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Hardware_virtualization. [Último acceso: 25 Octubre 2020].
- [59] «Virtualización de escritorio - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Virtualizaci%C3%B3n_de_escritorio. [Último acceso: 25 Octubre 2020].
- [60] «OS-level virtualization - Wikipedia,» 24 Octubre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/OS-level_virtualization. [Último acceso: 25 Octubre 2020].
- [61] «FAQ - Libvirt Wiki,» 25 Marzo 2020. [En línea]. Available: <https://wiki.libvirt.org/page/FAQ>. [Último acceso: 25 Octubre 2020].
- [62] «libvirt: libvirtd,» 23 Octubre 2020. [En línea]. Available: <https://libvirt.org/manpages/libvirtd.html>. [Último acceso: 25 Octubre 2020].

- [63] «libvirt: virsh,» 23 Octubre 2020. [En línea]. Available: <https://libvirt.org/manpages/virsh.html>. [Último acceso: 25 Octubre 2020].
- [64] «Git,» [En línea]. Available: <https://git-scm.com/>. [Último acceso: 25 Octubre 2020].
- [65] «Control de versiones - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Control_de_versiones. [Último acceso: 25 Octubre 2020].
- [66] «About CentOS,» [En línea]. Available: <https://www.centos.org/about/>. [Último acceso: 25 Octubre 2020].
- [67] «JetBrains - Wikipedia, la enciclopedia libre,» 23 Octubre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/JetBrains>. [Último acceso: 25 Octubre 2020].
- [68] «API Platform: API Tools & Solutions | Postman,» 23 Octubre 2020. [En línea]. Available: https://www.postman.com/api-platform/?utm_source=www&utm_medium=home_hero&utm_campaign=button. [Último acceso: 25 Octubre 2020].
- [69] «Mozilla Firefox - Wikipedia, la enciclopedia libre,» 25 Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Mozilla_Firefox. [Último acceso: 26 Octubre 2020].
- [70] «Angular - Workspace and project file structure,» 23 Octubre 2020. [En línea]. Available: <https://angular.io/guide/file-structure>. [Último acceso: 28 Octubre 2020].
- [71] «Angular - Introduction to modules,» 23 Octubre 2020. [En línea]. Available: <https://angular.io/guide/architecture-modules>. [Último acceso: 28 Octubre 2020].
- [72] «Angular - Introduction to components and templates,» 23 Octubre 2020. [En línea]. Available: <https://angular.io/guide/architecture-components>. [Último acceso: 23 Octubre 2020].
- [73] «¿Cuál es la diferencia entre un modal, un pop up, un pop over y un light box? | IDA Chile,» 03 Mayo 2017. [En línea]. Available: <https://blog.ida.cl/disenio/ldiferencia-modal-pop-up-pop-over-light-box/>. [Último acceso: 29 Octubre 2020].
- [74] «novnc/websockify: Websockify is a WebSocket to TCP proxy/bridge. This allows a browser to connect to any application/server/service. Implementations in Python, C, Node.js and Ruby.,» [En línea]. Available: <https://github.com/novnc/websockify>. [Último acceso: 29 Octubre 2020].
- [75] «Process identifier - Wikipedia,» 15 Octubre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Process_identifier. [Último acceso: 29 Octubre 2020].

- [76] «libvirt: Module libvirt-domain from libvirt,» 26 Octubre 2020. [En línea]. Available: <https://libvirt.org/html/libvirt-libvirt-domain.html>. [Último acceso: 29 Octubre 2020].
- [77] «How to Install RabbitMQ Server on CentOS 7,» HowtoForge, [En línea]. Available: <https://www.howtoforge.com/tutorial/how-to-install-rabbitmq-server-on-centos-7/>. [Último acceso: 18 Agosto 2020].
- [78] «Celery - Distributed Task Queue — Celery 4.4.7 documentation,» [En línea]. Available: <https://docs.celeryproject.org/en/v4.4.7/>. [Último acceso: 18 Agosto 2020].
- [79] «libvirt: The virtualization API,» [En línea]. Available: <https://libvirt.org/>. [Último acceso: 18 Agosto 2020].
- [80] M. Grinberg, «Using Celery With Flask - miguelgrinberg.com,» 15 Enero 2015. [En línea]. Available: <https://blog.miguelgrinberg.com/post/using-celery-with-flask>. [Último acceso: 31 Agosto 2020].
- [81] N. Boukehil, «How to add novnc to an angular project,» 7 Marzo 2020. [En línea]. Available: <https://www.nasserboukehil.com/angular-vnc-client/>. [Último acceso: 12 Octubre 2020].
- [82] N. Alonso, «¿Qué es un WSGI?. La palabra WSGI es una de esas palabras... | by Nacho Alonso | Medium,» 10 Noviembre 2019. [En línea]. Available: <https://medium.com/@nachoalod/que-es-wsgi-be7359c6e001>. [Último acceso: 25 Octubre 2020].
- [83] «Welcome to Flask-SocketIO's documentation! — Flask-SocketIO documentation,» 28 Mayo 2020. [En línea]. Available: <https://flask-socketio.readthedocs.io/en/latest/>. [Último acceso: 2 Noviembre 2020].
- [84] «CentOS - Wikipedia, la enciclopedia libre,» 04 Noviembre 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/CentOS>. [Último acceso: 06 Noviembre 2020].